Title
Unlearning and Privacy in Deep Neural Networks

Author
Golatkar, Aditya

Publication Date
2023

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Unlearning and Privacy in Deep Neural Networks

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Aditya Sharad Golatkar

2023

ABSTRACT OF THE DISSERTATION

Unlearning and Privacy in Deep Neural Networks

by

Aditya Sharad Golatkar
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2023
Professor Stefano Soatto, Chair

We explore the problem of selectively forgetting (or *unlearning*) a particular subset of the data used for training a deep neural network. While the effects of the data to be forgotten can be hidden from the output of the network, insights may still be gleaned by probing deep into its weights. We propose methods for "scrubbing" the weights clean of information about a particular set of training data without requiring retraining from scratch. In the "white-box" setting, the weights are modified so that any probing function of the weights is indistinguishable from the same function applied to the weights of a network trained without the data to be forgotten. This condition is a generalized and weaker form of Differential Privacy. Then we improve upon the white-box forgetting method by generalizing it across different readout functions, and show that it can be extended to ensure forgetting in the final activations of the network in a "black-box" setting. We introduce a new bound on how much information can be extracted per query about the forgotten cohort from a black-box network for which only the input-output behavior is observed. The proposed forgetting procedure has a deterministic part derived from the differential equations of a linearized version of the model, and a stochastic part that ensures information destruction by adding noise tailored to the geometry of the loss landscape. We exploit the connections between the final activations and

weight dynamics of a DNN inspired by Neural Tangent Kernels to compute the information in the final activations. To improve the deterministic part of the forgetting procedure, we present the first method for linearizing a pre-trained model (Linear Quadratic Fine-tuning) that achieves comparable performance to non-linear fine-tuning on most of real-world image classification tasks tested, thus enjoying the interpretability of linear models without incurring punishing losses in performance. LQF consists of simple modifications to the architecture, loss function and optimization typically used for classification which is sufficient to approach the performance of non-linear fine-tuning. We use this to introduce a novel notion of forgetting in mixed-privacy setting, where we know that a "core" subset of the training samples does not need to be forgotten. While this variation of the problem is conceptually simple, we show that working in this setting significantly improves the accuracy and guarantees of forgetting methods applied to vision classification tasks. Moreover, our method allows efficient removal of all information contained in non-core data by simply setting to zero a subset of the weights with minimal loss in performance and can achieve close to the state-of-the-art accuracy on large scale vision tasks. To cover the other end of the privacy spectrum, we introduce *AdaMix*, an adaptive differentially private algorithm for training deep neural network classifiers using both private and public image data. AdaMix incorporates few-shot training, or cross-modal zero-shot learning, on public data prior to private fine-tuning, to improve the trade-off. AdaMix reduces the error increase from the non-private upper bound from the 167-311% of the baseline, on average across 6 datasets, to 68-92% depending on the desired privacy level selected by the user. AdaMix tackles the trade-off arising in visual classification, whereby the most privacy sensitive data, corresponding to isolated points in representation space, are also critical for high classification accuracy. In addition, AdaMix comes with strong theoretical privacy guarantees and convergence analysis.

The dissertation of Aditya Sharad Golatkar is approved.

Guido Montufar Cuartas

Cho-Jui Hseih

Quanquan Gu

Lieven Vandenberghe

Stefano Soatto, Committee Chair

University of California, Los Angeles

2023

To my parents, especially my mother!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I started my journey at UCLA with an admission to the masters program in the electrical engineering department. Although I was curious to pursue a Ph.D. I neither had any clarity nor any plan. Thanks to Pratik Chaudhari, I was able to meet Stefano Soatto at the start of 2019, which marked the birth of a beautiful chapter in my life. Stefano has been someone who has influenced my life in each and every aspect, be it simply being my advisor, to teaching me about the most important thing in life, how to be a good human being. I owe the last few years of my life to Stefano, for his constant support, providing me with research directions, helping me with my internships, to supporting me during the most difficult personal phase of my life. He helped me grow at a professional and a personal level throughout this journey of life. I dont have enough words to thank Stefano, but would like to say the following; the living god next to my parents for me is Stefano.

The second important person I met during this time is a former labmate of mine, Alessandro Achille, who has been extremely critical for my Ph.D. Our research journey started with trying to simulate dreams for deep neural networks, to helping me complete my dream of pursuing a Ph.D. He is someone who inspires me academically every single day through his mathematically philosophical thinking, bias for action and ability to deliver during conferences. Working with Stefano and Alessandro has been an absolute dream, and I am hoping to be fortunate enough to work alongside them in the near future.

I will forever be grateful towards my committee, Quanquan Gu, Cho Jui Hsieh, Guido Montufar, Leiven Vandenberghe for being so helpful during the entire program while being kind and compassionate during the defense which enabled my mother to watch my defense. Those days have forever been etched in my heart and I am obliged beyond words towards my committee for this!

Friends play a major role in every aspect of life, and I have been blessed to have made some amazing friends during this journey, which includes friends I made in the Vision Lab at UCLA, Alex Wong, Tian Yu Liu, Stephanie Tsuei, Albert Zhao, Alfred Wu, Safa Cicek, Alex Tiard, Tong He, Xiaohan Fe, Xinzhu Bei, Dong Lao, Yanchao Yang, Pratik Chaudhari, Parth Agarwal, Simone

Bombari, Rida Rammal, my roommates and friends from undergrad, Rudrajit Das, Debarnab Mitra, Ravi Teja, Pradyumna Chari, Dipti Sahu.

I was able to enhance my practical knowledge through multiple internships where I was fortunate to work with some really brilliant people like Ashwin Swaminathan, Marzia Polito, Luis Goncalves, Avinash Ravichandran, Vijay Mahadevan, Benjamin Bowman, Yonathan Dukler, Luca Zancato, Matthew Trager, Pramudhita Perera.

My mothers oncologists Dr Sudeep Gupta, Dr Hollis Dsouza and Dr Rajendra Badwe have played a phenomenal role in taking care of my mother.

Finally, my family has been instrumental in supporting me throughout this entire journey. The list is long but everyone deserves a praise for their efforts towards maintaining the health of my mother in the last two years. I would like to thank, *Thakurs:* Jayant, Yatin, Rakesh, Vishal, Kishore, Rajkumar, Prakash, Ajay, Santosh, Praveen, Varun, Pratik, Akash, Rugved, Deep, Shakuntala, Nutan, Vidya, Madhuri, Vrushali, Sarita, Neha, Niharika, Sonam, Neel, Taneesh, *Redkars:* Yogita, Advait, *Deshpandes:* Vandana, Ajinkya, Ulhas, *Wares:* Vaishali, Sunil, Nilima Chaudhari, Sandhya Raut, Swapnil Raut, Praful Patil, Pradeep Kambli, Hema Patil, Surekha Nagvekar, *Deshpandes:* Muktai, Abhay, Anjana, Smruti Deshpande and many more!

Whatever I am today is because of my parents, so my last thanks goes to them for making me who I am today. Without them I am incomplete and nothing!

I will be eternally grateful to UCLA, along with all the research grants (ARO, ONR, MURI) which helped fund my PhD.

VITA

2014-2018	B.Tech (Electrical Engineering) and Minor (Computer Science), Indian Institute of Technology Bombay
2019	Applied Scientist Intern, Amazon Go
2020	Applied Scientist Intern, AWS AI Labs
2021	Applied Scientist Intern, AWS AI Labs
2022	Applied Scientist Intern, AWS AI Labs
2018-Current	Ph.D student, Computer Science, University of California, Los Angeles

PUBLICATIONS

Golatkar, A., Achille, A. and Soatto, S., 2020. *Eternal sunshine of the spotless net: Selective forgetting in deep networks.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 9304-9312).

Golatkar, A., Achille, A. and Soatto, S., 2020. *Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations.* In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX 16 (pp. 383-398). Springer International Publishing.

Golatkar, A.S., Achille, A. and Soatto, S., 2019. *Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence.* Advances in Neural Information Processing Systems, 32.

Achille, A., **Golatkar, A.**, Ravichandran, A., Polito, M. and Soatto, S., 2021. *Lqf: Linear quadratic fine-tuning.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 15729-15739).

Golatkar, A., Achille, A., Ravichandran, A., Polito, M. and Soatto, S., 2021. *Mixed-privacy forgetting in deep networks.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 792-801).

Golatkar, A., Achille, A., Wang, Y.X., Roth, A., Kearns, M. and Soatto, S., 2022. *Mixed differential privacy in computer vision.* In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8376-8386).

Rammal, M.R., Achille, A., **Golatkar, A.**, Diggavi, S. and Soatto, S., 2022. *On Leave-One-Out Conditional Mutual Information For Generalization.* Advances in Neural Information Processing Systems

Dukler, Y., Bowman, B., Achille, A., **Golatkar, A.**, Swaminathan, A. and Soatto, S., 2023. *Safe: Machine unlearning with shard graphs.* arXiv preprint arXiv:2304.13169, Submitted

Golatkar, A., Achille, A., Swaminathan, A. and Soatto, S., 2023. *Training Data Protection with Compositional Diffusion Models.*, Submitted

CHAPTER 1

Introduction

1.1 Motivation

Say you are the number ‘6’ in the MNIST handwritten digit database. You are proud of having nurtured the development of convolutional neural networks and their many beneficial uses. But you are beginning to feel uncomfortable with the attention surrounding the new “AI Revolution,” and long to not be recognized everywhere you appear. You wish a service existed, like that offered by the firm Lacuna INC in the screenplay *The Eternal Sunshine of the Spotless Mind*, whereby you could submit your images to have your identity scrubbed clean from handwritten digit recognition systems. Before you, the number ‘9’ already demanded that digit recognition systems returned, instead of a ten-dimensional “pre-softmax” vector (meant to approximate the log-likelihood of an image containing a number from 0 to 9) a *nine*-dimensional vector that excluded the number ‘9’. So now, every image showing ‘9’ yields an outcome at random between 0 and 8. Is this enough? It could be that the system still contains *information* about the number ’9,’ and just suppresses it in the output. How do you know that the system has truly forgotten about you, even *inside the black box*? Is it possible to scrub the system so clean that it behaves as if it had never seen an image of you? Is it possible to do so without sabotaging information about other digits, who wish to continue enjoying their celebrity status?

We study the problem of removing information pertaining to a given set of data points from the weights of a trained network. After removal, a potential attacker should not be able to recover information about the forgotten cohort, where an attacker is any agent intent to extract information

about the data used for training, from the knowledge of the weights or outputs at inference time. We consider both the cases in which the attacker has full access to the weights of the trained model, and the less-studied case where the attacker can only query the model by observing some input data and the corresponding output, for instance through a web API.

Machine unlearning or selective forgetting is a nascent field of increasing relevance with the advent of large scale generative models, be it language, images, videos or a combination of different modalities. Such large-scale models are also accompanied by stringent laws which enables user to have their data forgotten from the weights of trained networks, some of these include the General Data Protection Regulation in the European Union [Man13], California Consumer Protection Act [Tor18], PIPEDA privacy legislation in Canada [Cof20]. Thus its important for researchers to study the problem of unlearning, along with its connections and differences from Differential Privacy [Dwo08].

1.2 A Proposed solution and Thesis outline

We provide the first result [GAS19a] to perform selective forgetting for trained deep networks. In chapter 2 [GAS19a], we define the problem of selective forgetting information theoretically by bounding the amount of remaining information after data removal, and attempt to provide a solution by injecting noise into the weights of a network proportional to the Fisher Information Matrix. More importantly, we show what does not constitute unlearning. Then we improve this solution by providing a single shot forgetting update using inspiration from Neural Tangent Kernel [JGH18] along with the Fisher Forgetting noise perturbation. This solution attempts to approximate the non-linear deep networks with a linear approximation in infinite width. Using this we provide information theoretic bounds for information leakage from the activations after data removal, and provide unlearning update in the function space instead of weights space. We build on the previous observations to propose linearized networks (chapter 5), which perform comparable to non-linear fine-tuning by optimizing a quadratic loss function (Linear Quadratic Fine-tuning -LQF), which

enables theory to meet practice for deep networks. Using LQF we propose a novel forgetting procedure which exploits publically available core data which is not required to be forgotten in a mixed-private setting which improves upon all the previously proposed methods, along with rigorous theoretical guarantees and quantifiable information theoretic bounds. Finally, we extend the mixed-private setting to Differential privacy [Dwo08] which aims to protect the privacy of every training sample by limiting its influence in the training optimization. We show that our proposed algorithm AdaMix, is able to outperform all the previous state of the art methods for differentially private image classification tasks and also provide new benchmarks for the same. Overall, we analyse the unlearning spectrum, (i) by starting from proposing forgetting procedures for deep networks by simple perturbing their weights with gaussian noise, (ii) adding a single shot perturbation to the weights of the non-linear network in addition to the scrubbing gaussian noise, (iii) linearizing deep networks (LQF) in such a way that they perform comparable to non-linear fine-tuning, (iv) propose a mixed-forgetting framework in a mixed-linear setting using the linearized deep networks with access to publically available data, (v) explore mixed-private setting for differentially private learning which aims to protect the privacy of every individual sample in the training data.

1.3 Summary of Contributions

In chapter 2 propose a definition of selective forgetting for trained neural network models. It is not as simple as obfuscating the activations, and not as restrictive as Differential Privacy. Second, we propose a scrubbing procedure that removes information from the trained weights, without the need to access the original training data, nor to re-train the entire network. We compare the scrubbed network to the gold-standard model(s) trained from scratch without any knowledge of the data to be forgotten. We also prove the optimality of this procedure in the quadratic case. The approach is applicable to both the case where an entire class needs to be forgotten (e.g. the number '6') or multiple classes (e.g., all odd numbers), or a particular subset of samples within a class, while still maintaining output knowledge of that class. Our approach is applicable to networks pre-trained

using standard loss functions, such as cross-entropy, unlike Differential Privacy methods that require the training to be conducted in a special manner. Third, we introduce a computable upper bound to the amount of the retained information, which can be efficiently computed even for DNNs. We further characterize the optimal tradeoff with preserving complementary information. We illustrate the criteria using the MNIST and CIFAR-10 datasets, in addition to a new dataset called "Lacuna."

In chapter 3 we address the "black-box" setting, we introduce methods to scrub information from, and analyze the content of deep networks from their final activations (black-box attacks). We introduce a "one-shot" forgetting algorithms that work better than the previous method [GAS19a] for both white-box and black-box attacks for DNNs. This is possible thanks to an elegant connection between activations and weights dynamics inspired by the neural tangent kernel (NTK), which allows us to better deal with the null-space of the network weights. Unlike the NTK formalism, we do not need infinite-width DNNs. We show that better bounds can be obtained against black-box attacks than white-box, which gives a better forgetting vs error trade-off curve. More precisely, we show that we can quantify the maximum amount of information that an attacker can extract from observing inputs and outputs (black-box attack), as well as from direct knowledge of the weights (white-box).

To understand the role of the pre-trained backbone in unlearning in chapter 4 we seek to identify the "critical period" during training after which the model behaves more convexly, as a results linear approximations to a neural network can used a proxy to construct an unlearning procedure. We find that regularization via L_2 penalty or data augmentation has the same effect on generalization when applied *only* during the initial epochs of training. Conversely, if regularization is applied only in the latter phase of convergence, it has little effect on the final solution, whose generalization is as bad as if regularization never happened. This suggests that, contrary to classical models, the mechanism by which regularization affects generalization in deep networks is not by changing the landscape of critical points at convergence, but by influencing the early transient of learning. Applying L_2 regularization during the initial period of the training enables the network to pass through narrow bottlenecks during training and enter regions with flat or more convex loss landscapes. Thus

linearizing networks after they cross the initial critical periods provides linear models with similar generalization capabilities as that of non-linear fine-tuning. We hypothesis that pre-training is one such way of crossing these narrow bottlenecks, to provide convex models during fine-tuning.

To enjoy the perks of linear optimization for unlearning we linearize deep networks to obtain large linear models for downstream tasks, in our case image classification. chapter 5 introduces *Linear Quadratic Fine-Tuning* (LQF), a method to linearize and train deep neural networks that achieves comparable performance to non-linear fine-tuning (NLFT) on real-world image classification, while enjoying all the benefits of linear-quadratic optimization. LQF performs fine-tuning without optimizing hyper-parameters such as learning rate or batch size, enables predicting the effect of even individual training samples on the trained classifier, and easily allows incorporating linear constraints during training. LQF achieves performance comparable to NLFT, and better in the low-data regime which is the most relevant to many real applications. The key enablers of LQF are simple and known in the literature, although not frequently used: (i) We replace the cross-entropy loss with the mean-squared error loss, making the optimization problem quadratic [GDN13, HB21, BD20], (ii) we replace ReLU with Leaky-ReLU [MHN13], and (iii) we perform pre-conditioning using Kronecker factorization (K-FAC) [MG15]. Individually, these changes bring limited improvements to standard training of non-linear models. However, we show that their combined use has a much larger impact on the performance of linearized models.

When building a classification system, one rarely has all the data to be used for training available at the outset. More often, one starts by pre-training a model with some "core" dataset (e.g. ImageNet, or datasets close to the target task) and then incorporates various cohorts of task-specific data as they become available from diverse sources. In chapter 6 we introduce the problem of forgetting (unlearning or data deletion or scrubbing) in a mixed-privacy setting which, compared to previous formalizations, is better taylored to standard practice, and allows for better privacy guarantees. In this setting we propose ML-Forgetting. ML-Forgetting trains a set of non-linear *core* weights and a set of linear *user* weights, which allow it to achieve both good accuracy, thanks to the flexibility of the non-linear weights, and strong privacy guarantees thanks to the linear weights. As a side

effect, all the user data may be forgotten completely with the lowest possible drop in performance by simply erasing the user weights. We show that ML-Forgetting can be applied to large-scale vision datasets, and enjoys both strong forgetting guarantees and test time accuracy comparable to standard training of a Deep Neural Network (DNN). To the best of our knowledge, this is the first forgetting algorithm to do so. Furthermore, we show that ML-Forgetting can handle multiple sequential forgetting requests without degrading its performance, which is important for real world applications.

Unlearning removes data from the weights of training neural models, however, if the weights are obtained after training privately such that the weights dont contain information about any specific sample, the need for forgetting vanishes. Differential privacy is able to mathematically tackle this problem and provides algorithms and rigorous guarantees. As a result we seek to provide an improved algorithm which exploits public data in a mixed setting for private learning. To do so, we change the setting from most work on DP to include *labeled* public data sourced with the *same* labels as the target task. We call this setting *mixed differential privacy*, or MixDP. To address MixDP, we propose to use the public data not just for pre-training the backbone, but for few-shot or zero-shot learning of a classifier on the target tasks, prior to private fine-tuning. We show that indeed in the MixDP setting it is possible to achieve significant gains compared to training with only the private or only the public data, even using a small amount of the latter. To do so, we have to modify existing DP training algorithms to the mixed setting, which leads us to the development of AdaMix, a method for MixDP that uses public data to tune and adapt all major steps of private training, in particular model initialization, clipping of the gradients, and projection onto a lower-dimensional sub space. Some of these ideas to use auxiliary public data were applied in different contexts in isolation, but have suboptimal privacy/performance trade-offs for visual tasks. We show significant improvement w.r.t. baselines on a novel benchmark for MixDP vision tasks. We show that the zero-shot text information can improve the performance of private models and analyze the goodness of our method both theoretically (new theorems) and empirically (pDP analysis).

1.4 Related Works

1.4.1 Unlearning or selective forgetting

The term "machine unlearning" was introduced by [CY15], who shows an efficient forgetting algorithm in the restricted setting of statistical query learning, where the learning algorithm cannot access individual samples. [MKK17] provided the first framework for instantaneous data summarization with data deletion using robust streaming submodular optimization. [GGV19] formalizes the problem of efficient data elimination, and provides engineering principles for designing forgetting algorithms. However, they only provide a data deletion algorithms for k-means clustering. These unlearning procedures were proposed for classical pre-deep learning algorithms, which does not involve removing information from trained deep neural networks. The problem of unlearning is more applicable to trained deep networks compared to classical machine learning models as trained deep networks contain millions (or billions with current large language models) of parameters, as a results unlearning is essential and re-training models will cost exorbitant amount of monetary resources and compute.

Unlearning methods can be broadly categorized in two directions, (i) approximate unlearning methods, and (ii) sharding based forgetting methods. Approximate unlearning procedures are inspired from differential privacy and also involve information theoretic bounds. In chapter 2 we propose the first unlearning method which involves removing information from trained deep networks using Fisher forgetting. We talk about the weights of a network as containing "information," even though we have *one* set of weights whereas information is commonly defined only for random variables. While this has caused some confusion in the literature, [AS19] proposes a viable formalization of the notion, which is compatible with our framework. Thus, we will use the term "information" liberally even when talking about a particular dataset and set of weights. In chapter 3 we extend the framework proposed in chapter 2 to activations. They also show that an approximation of the training process based on a first-order Taylor expansion of the network (NTK theory) can be used to the estimate the weights after forgetting. This approximation works well on small scale

vision datasets. However, the approximation accuracy and the computational cost degrade for larger datasets (in particular the cost is quadratic in the number of samples). We also use linearizarion but, crucially, instead of linearly approximating the training dynamics of a non-linear network, we show that we can directly train a linearized network for forgetting. This ensures that our forgetting procedure is correct, and it allows us to scale easily to standard real-world vision datasets. [BSZ20] proposed a forgetting method for logit-based classification models by applying linear transformation to the output logits, but do not remove information from the weights. [GGH19] formulates data removal mechanisms using differential privacy, and provides an algorithm for convex problems based on a second order Newton update. They suggest applying this method on top of the features learned by a differentially private DNN, but do not provide a method to remove information from a DNN itself. [ISC20] provides a projective residual update based method using synthetic data points to delete data points for linear regression based models. [KL17, GSL19] provide a newton based method for computing the influence of a training point on the model predictions in the context of model interpretation and cross-validation, however, such an approach can also be used for data removal. [WDD20] proposes to forgetting training samples by exploiting the iterative training algorithm. They cache the intermediate statistics of training and when asked to forget a sample they compute the total gradient accumulated by that sample during the training and subtract it from the final weights. [NRS20] proposed a gradient descent based method for data deletion in convex settings, with theoretical guarantees for multiple forgetting requests. They also introduce the notion of statistical indistinguishably of the entire state or just the outputs similar to the information theoretic framework of [GAS20].

The second direction in unlearning is using sharded models: [BCC19] propose a forgetting procedure based on sharding the dataset and training multiple models. Aside from the storage cost, they need to retrain subset of the models. Such methods enable complete information removal from the models, as the weights affected by the samples to forget are discarded, however, if information about the training samples may still exist in the choice of splitting.

Readout functions: The success of the forgetting procedure, can be measured by looking at

whether a discriminator function $R(w)$ that can guess – at better than chance probability – whether a set of weights w was trained with or without $\mathcal{D}_f$ or whether it was trained with $\mathcal{D}_f$ and then scrubbed. We call such functions *readout functions*. A popular example of readout function is the confidence of the network (that is, the entropy of the output softmax vector) on the samples in $\mathcal{D}_f$: Since networks tend to be overconfident on their training data [GPS17, KHH20], a higher than expected confidence may indicate that the network was indeed trained on $\mathcal{D}_f$. Alternatively, we can measure the success of the forgetting procedure by measuring the amount of remaining mutual information $\mathcal{I}(S(w); \mathcal{D}_f)$. [1] between the scrubbed weights $S(w)$ and the data $\mathcal{D}_f$ to be forgotten. While this is more difficult to estimate, it can be shown that $\mathcal{I}(S(w); \mathcal{D}_f)$ upper-bounds the amount of information that any read-out function can extract. Said otherwise, it is an upper-bound on the amount of information that an attacker can extract about $\mathcal{D}_f$ using the scrubbed weights $S(w)$.

Stability of SGD: In [HRS15], a bound is derived on the divergence of training path of models trained with the same random seed (*i.e.*, same initialization and sampling order) on datasets that differ by one sample (the "stability" of the training path). This can be considered as a measure of memorization of a sample and, thus, used to bound the generalization error. While these bounds are often loose, we introduce a novel bound on the residual information about a set of samples to be forgotten, which exploits ideas from both the stability bounds and the PAC-Bayes bounds [McA13], which have been successful even for DNNs [DR17].

Catastrophic forgetting This occurs when a network trained on a task rapidly loses accuracy on that task when fine-tuned for another. But while the network can forget a *task,* the information on the *data* it used may still be accessible from the weights. Hence, even catastrophic forgetting does not satisfy our stronger definition. Interestingly, however, our proposed solution for forgetting relates to techniques used to *avoid* forgetting: [KPR17a] suggests adding an L_2 regularizer using the Fisher Information Matrix (FIM) of the task. We use the FIM, restricted to the samples we wish to retain, to compute the optimal noise to destroy information, so that a cohort can be forgotten

[1] $\mathcal{I}(x, y) = \mathbb{E}_{P_{x,y}}[\log(P_{x,y}/P_x P_y)]$ is the mutual information between x and y, where $P_{x,y}$ is the joint distribution and P_x, P_y are the marginal distributions.

while maintaining good accuracy for the remaining samples. Part of our forgetting procedure in Chapter 2 can be interpreted as performing "optimal brain damage" [LDS90] in order to remove information from the weights if it is useful only or mainly to the class to be forgotten.

1.4.2 Membership Inference Attacks

[TLG19, HAP17, HMD19, PTD17, SS15, SRS17, SDO19] try to guess if a particular sample was used for training a model. Since a model has forgotten only if an attacker cannot guess at better than chance level, these attacks serve as a good metric for measuring the quality of forgetting. In our experiments we construct a black-box membership inference attack similar to the shadow model training approach in [SS15]. Such methods relate to model inversion methods [FJR15] which aim to gain information about the training data from the model output.

1.4.3 Neural Tangent Kernel

[JGH18, LXS19] show that the training dynamics of a linearized version of a Deep Network — which are described by the so called NTK matrix — approximate increasingly better the actual training dynamics as the network width goes to infinity. [ADH19, LWY19] extend the framework to convolutional networks. [SA19] compute information-theoretic quantities using the closed form expressions for various quantities that can be derived in this settings. While we do use the infinite width assumption, in Chapter 3 we show that the same linearization framework and solutions are a good approximation of the network dynamics during fine-tuning, and use them to compute an optimal scrubbing procedure.

1.4.4 Differential privacy

[DR14] focuses on guaranteeing that the parameters of a trained model do not leak information about *any* particular individual. While this may be relevant in some applications, the condition is often too difficult to enforce in deep learning (although see [ACG16]), and not always necessary. It

requires the possible distribution of weights, given the dataset, $P(w|\mathcal{D})$ to remain almost unchanged after replacing a sample. Our definition of selective forgetting in chapter 2 can be seen as a generalization of differential privacy. In particular, we do not require that information about *any* sample in the dataset is minimized, but rather about a particular subset $\mathcal{D}_f$ selected by the user. Moreover, we can apply a "scrubbing" function $S(w)$ that can perturb the weights in order to remove information, so that $P(S(w)|\mathcal{D})$, rather than $P(w|\mathcal{D})$, needs to remain unchanged. This less restrictive setting allows us to train standard deep neural networks using stochastic gradient descent (SGD), while still being able to ensure forgetting.

Most work on Differential Privacy [CM08, SCS13, WG19, WG19, YNB21, BST14, SS15, INS19, JW18, LK18] uses public data either for: generic pre-training unrelated to the task [TB21], or to tune parameters [KRR20, ZWB21, YZC21], or as additional *unlabeled* data [PAE17, PSM18]. Instead, we use a *small amount of labeled public data related to the task* to improve the accuracy of a private model under a given privacy parameter ϵ.

Unlabeled public data. PATE [PAE17, PSM18, UNK21] trains teacher models on different partitions of the data to predict labels, and privately vote on how to label public (unlabelled) data. Then, a student model is trained in semi-supervised fashion on public data, now partially labelled using the private data. This strategy does not directly allow to train a model on a mix of labeled public and private data. [ZYC20] noted that the number of partitions required for meaningful privacy guarantees may be prohibitive for vision datasets. As an alternative, they suggest labeling with a differentially private KNN. In MixDP, where we assume that the public data is already labeled, these algorithms would perform at best similarly to our *Only-Public* baseline. Instead, what we are interested in effectively using labeled private data to improve performance relative to already labeled public data.

Differential privacy for Language models. [LTL21, YNB21] show that a large language model pre-trained on generic public data can be fine-tuned on task-specific private data with only modest loss in accuracy. In contrast, our focus is on using small amounts of public data for fine-tuning. We

note that in language models strong transfer of information from the pre-training is facilitated by the fact that the output space (a sequence of tokens) is the same for all tasks. Conversely, vision models need to relearn different output spaces based on the task (i.e., the last layer is always trained from scratch) reducing the transfer of information. We show that we can partly recover this advantage by either using an initialization based on public data *of the same task*, or a *multi-modal* vision-language model. We use CLIP [RKH21] for zero-shot learning.

Adaptive clipping. Similarly to us, [WZ20] studies DP training with access to a small amount of public data, and use adaptive gradient clipping [ZCH21, ATM21, BPS19, VSB18] based on the public gradients. Contrary to us, they first train a private model separately and then a mixed public-private model using the private one as a regularizer with compelling results on tabular data. We found this approach ineffective for computer vision, where usually models are pre-trained on public data. In Sec. 7.2 we show that AdaMix works better on vision tasks, and is further improved with multi-modal data.

Adaptive projection. [KRR20, ZWB21, YZC21] suggest improving DP-SGD by projecting the gradients into a lower-dimensional space estimated from public data. However, these methods do not leverage the performance improvements derived from training on public data, which are needed to make vision models viable. Using public data both to train and to estimate the subspace introduces more challenges, since the gradients of the public data and private data will then behave differently. Moreover, while they perform an SVD on the individual public sample gradients, we do the SVD of the total gradient matrix. Unlike the SVD of the matrix of individual sample gradients, this is cheaper to compute at each step.

CHAPTER 2

Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks

In this chapter, we propose the definition of *selective forgetting or unlearning* for trained neural network models. It is not as simple as obfuscating the activations, and not as restrictive as differential privacy. Second, we propose a scrubbing procedure that removes information from the trained weights, without the need to access the original training data, nor to re-train the entire network. We compare the scrubbed network to the gold-standard model(s) trained from scratch without any knowledge of the data to be forgotten. We also prove the optimality of this procedure in the quadratic case. The approach is applicable to both the case where an entire class needs to be forgotten (e.g. the number '6') or multiple classes (e.g., all odd numbers), or a particular subset of samples within a class, while still maintaining output knowledge of that class. Our approach is applicable to networks pre-trained using standard loss functions, such as cross-entropy, unlike differential privacy methods that require the training to be conducted in a special manner. Third, we introduce a computable upper bound to the amount of the retained information, which can be efficiently computed even for DNNs. We further characterize the optimal tradeoff with preserving complementary information. We illustrate the criteria using the MNIST and CIFAR-10 datasets, in addition to a new dataset called "Lacuna."

This chapter serves as an introductory chapter which provides the basic foundations used in the subsequent chapters. It aims to provide the basic idea of unlearning or selective forgetting for deep neural networks. We also discuss what does it mean for a network to forget a training sample, and more importantly what does not constitute forgetting i.e. what are some of the incorrect unlearning

procedures which may seem to be correct.

2.0.1 Preliminaries and Notation

Let $\mathcal{D} = \{x_i, y_i\}_{i=1}^N$ be a dataset of images x_i, each with an associated label $y_i \in \{1, \dots, K\}$ representing a class (or label, or identity). We assume that $(x_i, y_i) \sim P(x, y)$ are drawn from an unknown distribution P.

Let $\mathcal{D}_f \subset \mathcal{D}$ be a subset of the data (cohort), whose information we want to remove (scrub) from a trained model, and let its complement $\mathcal{D}_r := \mathcal{D}_f^\complement$ be the data that we want to retain. The data to forget $\mathcal{D}_f$ can be any subset of $\mathcal{D}$, but we are especially interested in the case where $\mathcal{D}_f$ consists of all the data with a given label k (that is, we want to completely forget about a class), or a subset of a class.

Let $\phi_w(\cdot) \in \mathbb{R}^K$ be a parametric function (model), for instance a DNN, with parameters w (weights) trained using $\mathcal{D}$ so that the k-th component of the vector ϕ_w in response to an image x approximates the optimal discriminant (log-posterior), $\phi_w(x)_k \simeq \log P(y = k|x)$, up to a normalizing constant.

2.0.2 Training algorithm and distribution of weights

Given a dataset $\mathcal{D}$, we can train a model — or equivalently a set of weights — w using some training algorithm A, that is $w = A(\mathcal{D})$, where $A(\mathcal{D})$ can be a stochastic function corresponding to a stochastic algorithm, for example stochastic gradient descent (SGD). Let $P(w|\mathcal{D})$ denote the distribution of possible outputs of algorithm A, where $P(w|\mathcal{D})$ will be a degenerate Dirac delta if A is deterministic. The scrubbing function $S(w)$ — introduced in the next section — is also a stochastic function applied to the weights of a trained network. We denote by $P(S(w)|\mathcal{D})$ the distribution of possible weights obtained after training on the dataset $\mathcal{D}$ using algorithm A and then applying the scrubbing function $S(w)$. Given two distributions $p(x)$ and $q(x)$, their Kullback-Leibler (KL) divergence is defined by $\mathrm{KL}\left(p(x) \,\|\, q(x)\right) := \mathbb{E}_{x\sim p(x)}\left[\log\left(p(x)/q(x)\right)\right]$. The KL-divergence

is always positive and can be thought of as a measure of similarity between distributions. In particular it is zero if and only if $p(x) = q(x)$. Given two random variables x and y, the amount of Shannon Mutual Information that x has about y is defined as $I(x;y) := \mathbb{E}_x\big[\,\mathrm{KL}\left(p(y|x) \,\|\, p(y)\right)\big]$.

2.1 Definition and Testing of Forgetting

Let ϕ_w be a model trained on a dataset $\mathcal{D} = \mathcal{D}_f \sqcup \mathcal{D}_r$ Then, a forgetting (or "scrubbing") procedure consists in applying a function $\mathrm{S}(w;\mathcal{D}_f)$ to the weights, with the goal of forgetting, that is to ensure that an "attacker" (algorithm) in possession of the model ϕ_w cannot compute some "readout function" $f(w)$, to reconstruct information about $\mathcal{D}_f$.

It should be noted that one can always infer *some* properties of $\mathcal{D}_f$, even without having ever seen it. For example, if $\mathcal{D}$ consists of images of faces, we can infer that images in $\mathcal{D}_f$ are likely to display two eyes, even without looking at the model w. What matters for forgetting is the amount of *additional* information $f(w)$ can extract from a cohort $\mathcal{D}_f$ by exploiting the weights w, that could not have been inferred simply by its complement $\mathcal{D}_r$. This can be formalized as follows:

Definition 1. *Given a readout function f, an optimal scrubbing function for f is a function $\mathrm{S}(w;\mathcal{D}_f)$ — or $\mathrm{S}(w)$, omitting the argument $\mathcal{D}_f$ — such that there is another function $\mathrm{S}_0(w)$ that does not depend on $\mathcal{D}_f$[1] for which:*

$$\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \,\|\, P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right) = 0. \tag{2.1}$$

The function $S_0(w)$ in the definition can be thought as a *certificate* of forgetting, which shows that $S(w)$ is indistinguishable from a model that has never seen $\mathcal{D}_f$. Satisfying the condition above is trivial by itself, *e.g.*, by choosing $S(w) = S_0(w) = c$ to be constant. The point is to do so while retaining as much information as possible about $\mathcal{D}_r$, as we will see later when we introduce the

[1] *If S_0 could depend on $\mathcal{D}_f$, we could take $S(w) = w$ to be the identity, and let $S_0(w)$ ignore w and obtain new weights by training from scratch on $\mathcal{D}$ — that is $S_0(w) = w'$ with $w' \sim p(w|\mathcal{D})$. This brings the KL to zero, but does not scrub any information, since $S(w)$ is the identity.*

Forgetting Lagrangian in eq. (2.4). The formal connection between this definition and the amount of Shannon Information about $\mathcal{D}_f$ that a readout function can extract is given by the following:

Proposition 1. *Let the forgetting set $\mathcal{D}_f$ be a random variable, for instance, a random sampling of the data to forget. Let Y be an attribute of interest that depends on $\mathcal{D}_f$. Then,*

$$I(Y; f(S(w))) \leq \mathbb{E}_{D_f}[\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \parallel P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right)]. \tag{2.2}$$

Yet another interpretation of eq. (2.1) arises from noticing that, if that quantity is zero then, given the output of the readout function $f(w)$, we cannot predict with better-than-chance accuracy whether the model $w' = \mathrm{S}(w)$ was trained with or without the data. In other words, after forgetting, membership attacks will fail.

In general, we may not know what readout function a potential attacker will use, and hence we want to be robust to every $f(w)$. The following lemma is useful to this effect:

Lemma 1. *For any function $f(w)$ we have:*

$$\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \parallel P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right) \qquad \leq \qquad \mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \parallel P(\mathrm{S}_0(w)|\mathcal{D}_r)\right)$$

Therefore, we can focus on minimizing the quantity

$$\mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \parallel P(\mathrm{S}_0(w)|\mathcal{D}_r)\right), \tag{2.3}$$

which guarantees robustness to any readout function. For the sake of concreteness, we now give a first simple example of a possible scrubbing procedure.

Example 1 (Forgetting by adding noise)**.** *Assume the weights w of the model are bounded. Let $S(w) = S_0(w) = w + \sigma n$, where $n \sim \mathcal{N}(0, I)$, be the scrubbing procedure that adds noise sampled from a Gaussian distribution. Then, as the variance σ increases, we achieve total forgetting:*

$$\mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \parallel P(\mathrm{S}_0(w)|\mathcal{D}_r)\right) \xrightarrow{\sigma \to \infty} 0.$$

While adding noise with a large variance does indeed help forgetting, it throws away the baby along with the bath water, rendering the model useless. Instead, we want to forget as much as possible about a cohort while retaining the accuracy of the model. This can be formalized by minimizing the **Forgetting Lagrangian**:

$$\mathcal{L} = \mathbb{E}_{S(w)}\big[L_{\mathcal{D}_r}(w)\big] + \lambda \,\mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \,\|\, P(\mathrm{S}_0(w)|\mathcal{D}_r)\right), \tag{2.4}$$

where $L_{\mathcal{D}_r}(w)$ denotes the loss of the model w on the retained data $\mathcal{D}_r$. Optimizing this first term is relatively easy. The problem is doing so while also minimizing the second (forgetting) term: For a DNN, the distribution $P(w|\mathcal{D})$ of possible outcomes of the training process is complex making difficult the estimation of the KL divergence above, a problem we address in the next section. Nonetheless, the Forgetting Lagrangian, if optimized, captures the notion of **selective forgetting** at the core of this work.

2.1.1 Stability and Local Forgetting Bound

Given a stochastic training algorithm $A(\mathcal{D})$, we can make the dependency on the random seed ϵ explicit by writing $A(\mathcal{D}, \epsilon)$, where we assume that $A(\mathcal{D}, \epsilon)$ is now a deterministic function of the data and the random seed. We now make the following assumptions: (1) the cohort to be forgotten, $\mathcal{D}_f$, is a small portion of the overall dataset $\mathcal{D}$, lest one is better-off re-training than forgetting, and (2) the training process $A(\mathcal{D}, \epsilon)$ is stable, *i.e.*, if $\mathcal{D}$ and $\mathcal{D}'$ differ by a few samples, then the outcome of training $A(\mathcal{D}, \epsilon)$ is close to $A(\mathcal{D}', \epsilon)$. Under stable training, we expect the two distributions $P(\mathrm{S}(w)|\mathcal{D})$ and $P(\mathrm{S}_0(w)|\mathcal{D}_r)$ in eq. (2.3) to be close, making forgetting easier. Indeed, we now show how we can exploit the stability of the learning algorithm to bound the Forgetting Lagrangian.

Proposition 2 (Local Forgetting Bound)**.** *Let $A(\mathcal{D}, \epsilon)$ be a training algorithm with random seed $\epsilon \sim P(\epsilon)$. Notice that in this case $P(S(w)|\mathcal{D}) = \mathbb{E}_\epsilon[P(S(w)|\mathcal{D}, \epsilon)]$. We then have the bound:*

$$\mathrm{KL}\left(P(S(w)|\mathcal{D}) \,\|\, P(S_0(w)|\mathcal{D}_r)\right) \leq \mathbb{E}_\epsilon\Big[\,\mathrm{KL}\left(P(S(w)|\mathcal{D}, \epsilon) \,\|\, P(S_0(w)|\mathcal{D}_r, \epsilon)\right)\Big] \tag{2.5}$$

In the local forgetting bound we do not look at the global distribution of possible outcomes as

the random seed varies, but only at the average of forgetting using a particular random seed. To see the value of this bound, consider the following example.

Corollary 1 (Gaussian forgetting)**.** *Consider the case where $S(w) = h(w) + n$ and $S_0(w) = w + n'$, where $n, n' \sim N(0, \Sigma)$ is Gaussian noise and $h(w)$ is a deterministic function. Since for a fixed random seed ϵ the weights $w = A(\mathcal{D}, \epsilon)$ are a deterministic function of the data, we have $P(S(w)|\mathcal{D}, \epsilon) = \mathcal{N}(h(A(\mathcal{D}, \epsilon)), \Sigma)$ and similarly $P(S_0(w)|\mathcal{D}_r, \epsilon) = \mathcal{N}(A(\mathcal{D}_r, \epsilon), \Sigma)$. Then, using the previous bound, we have:*

$$\mathrm{KL}\left(P(S(w)|\mathcal{D}) \,\|\, P(S_0(w)|\mathcal{D}_r)\right) \leq \frac{1}{2}\mathbb{E}_\epsilon\Big[(h(w) - w')^T\Sigma^{-1}(h(w) - w')\Big] \tag{2.6}$$

where $w = A(\mathcal{D}, \epsilon)$ and $w' = A(\mathcal{D}_r, \epsilon)$.

That is, we can upper-bound the complex term $\mathrm{KL}\left(P(S(w)|\mathcal{D}) \,\|\, P(S_0(w)|\mathcal{D}_r)\right)$ with a much simpler one obtained by averaging the results of training and scrubbing with different random seeds.

Moreover, this suggests three simple but general procedures to forget. Under the stability assumption, we can either (i) apply a function $h(w)$ that bring w and w' closer together (*i.e.*, minimize $h(w) - w'$ in eq. (2.6)), or (ii) add noise whose covariance Σ is high in the direction $h(w) - w'$, or (iii) both. Indeed, this will be the basis of our forgetting algorithm, which we describe next.

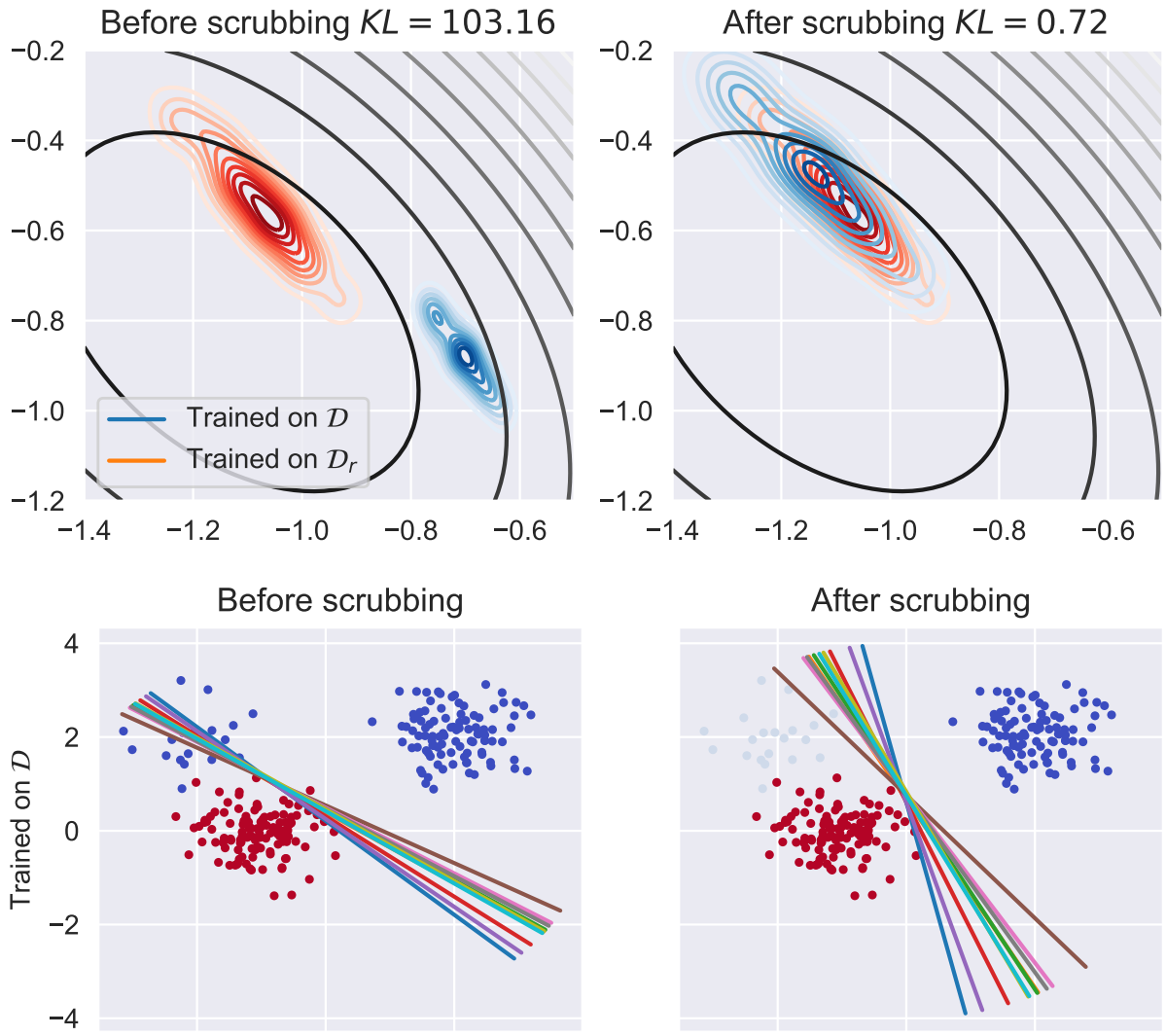

Figure 2.1: **(Top)** Distributions of weights $P(w|\mathcal{D})$ and $P(w|\mathcal{D}_r)$ before and after the scrubbing procedure is applied to forget the samples $\mathcal{D}_f$. The scrubbing procedure makes the two distributions indistinguishable, thus preventing an attacker from extracting any information about $\mathcal{D}_f$. The KL divergence measures the maximum amount of information that an attacker can extract. After forgetting, less than 1 NAT of information about the cohort $\mathcal{D}_f$ is accessible. **(Bottom)** The effect of the scrubbing procedure on the distribution of possible classification boundaries obtained after training. After forgetting the subject on the top left blue cluster, the classification boundaries adjust as if she never existed, and the distribution mimics the one that would have been obtained by training from scratch without that data.

2.2 Optimal Quadratic Scrubbing Algorithm

In this section, we derive an optimal scrubbing algorithm under a local quadratic approximation. We then validate the method empirically in complex real world problems where the assumptions are violated. We start with strong assumptions, namely that the loss is quadratic and optimized in the limit of small learning rate, giving the continuous gradient descent optimization

$$A_t(\mathcal{D}, \epsilon) = w_0 - (I - e^{-\eta A t})A^{-1}\nabla_w L_{\mathcal{D}}(w)|_{w=w_0},$$

where $A = \nabla^2 L_{\mathcal{D}}(w)$ is the Hessian of the loss. We will relax these assumptions later.

Proposition 3 (Optimal quadratic scrubbing algorithm)**.** *Let the loss be $L_{\mathcal{D}}(w) = L_{\mathcal{D}_f}(w) + L_{\mathcal{D}_r}(w)$, and assume both $L_{\mathcal{D}}(w)$ and $L_{\mathcal{D}_r}(w)$ are quadratic. Assume that the optimization algorithm $A_t(\mathcal{D}, \epsilon)$ at time t is given by the gradient flow on the loss with random initialization. Consider the scrubbing function*

$$h(w) = w + e^{-Bt}e^{At}d + e^{-Bt}(d - d_r) - d_r,$$

where $A = \nabla^2 L_{\mathcal{D}}(w)$, $B = \nabla^2 L_{\mathcal{D}_r}(w)$, $d = A^{-1}\nabla_w L_{\mathcal{D}}$ and $d_r = B^{-1}\nabla_w L_{\mathcal{D}_r}$. Then, $h(w)$ is such that $h(A_t(\mathcal{D}, \epsilon)) = A_t(\mathcal{D}_r, \epsilon)$ for all random initializations ϵ and all times t. In particular, $S(w) = h(w)$ scrubs the model clean of all information in $\mathcal{D}_f$:

$$\mathrm{KL}\left(P(S(w)|\mathcal{D}, \epsilon) \,\|\, P(w|\mathcal{D}_r, \epsilon)\right) = 0.$$

Note that when $t \to \infty$, that is, after the optimization algorithm has converged, this reduces to the simple Newton update*:*

$$S_\infty(w) = w - B^{-1}\nabla L_{\mathcal{D}_r}(w).$$

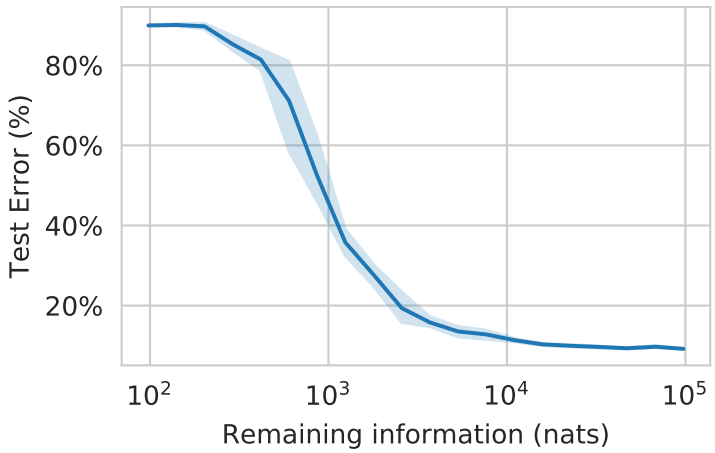

Figure 2.2: Trade-off between information remaining about the class to forget and test error, mediated by the parameter λ in the Lagrangian: We can always forget more, but this comes at the cost of decreased accuracy.

2.2.1 Robust Scrubbing

Proposition 3 requires the loss to be quadratic, which is typically not the case. Even if it was, practical optimization proceeds in discrete steps, not as a gradient flow. To relax these assumptions, we exploit the remaining degree of freedom in the general scrubbing procedure introduced in Corollary 1, which is the noise.

Proposition 4 (Robust scrubbing procedure)**.** *Assume that $h(w)$ is close to w' up to some normally distributed error $h(w) - w' \sim N(0, \Sigma_h)$, and assume that $L_{\mathcal{D}_r}(w)$ is (locally) quadratic around $h(w)$. Then the optimal scrubbing procedure in the form $S(w) = h(w) + n$, with $n \sim N(0, \Sigma)$, that minimizes the Forgetting Lagrangian eq.* (2.4) *is obtained when $\Sigma B \Sigma = \lambda \Sigma_h$, where $B = \nabla^2 L_{\mathcal{D}_r}(w)$. In particular, if the error is isotropic, that is $\Sigma_h = \sigma_h^2 I$ is a multiple of the identity, we have $\Sigma = \sqrt{\lambda \sigma_h^2} B^{-1/2}$.*

Putting this together with the result in Proposition 3 gives us the following robust scrubbing procedure:

$$S_t(w) = w + e^{-Bt} e^{At} d + e^{-Bt}(d - d_r) - d_r + (\lambda \sigma_h^2)^{\frac{1}{4}} B^{-1/4} n, \tag{2.7}$$

where $n \sim N(0, I)$ and B, d and d_r are as in Proposition 3. In Figure 2.1 we show the effect of the scrubbing procedure on a simple logistic regression problem (which is not quadratic) trained with SGD (which does not satisfy the gradient flow assumption). Nonetheless, the scrubbing procedure manages to bring the value of the KL divergence close to zero. Finally, when $t \to \infty$ (*i.e.*, the optimization is near convergence), this simplifies to the noisy Newton update which can be more readily applied:

$$\boxed{S_t(w) = w - B^{-1}\nabla L_{\mathcal{D}_r}(w) + (\lambda\sigma_h^2)^{\frac{1}{4}} B^{-1/4}\epsilon.} \tag{2.8}$$

Here λ is a hyperparameter that trades off residual information about the data to be forgotten, and accuracy on the data to be retained. The hyperparameter σ_h reflect the error in approximating the SGD behavior with a continuous gradient flow.

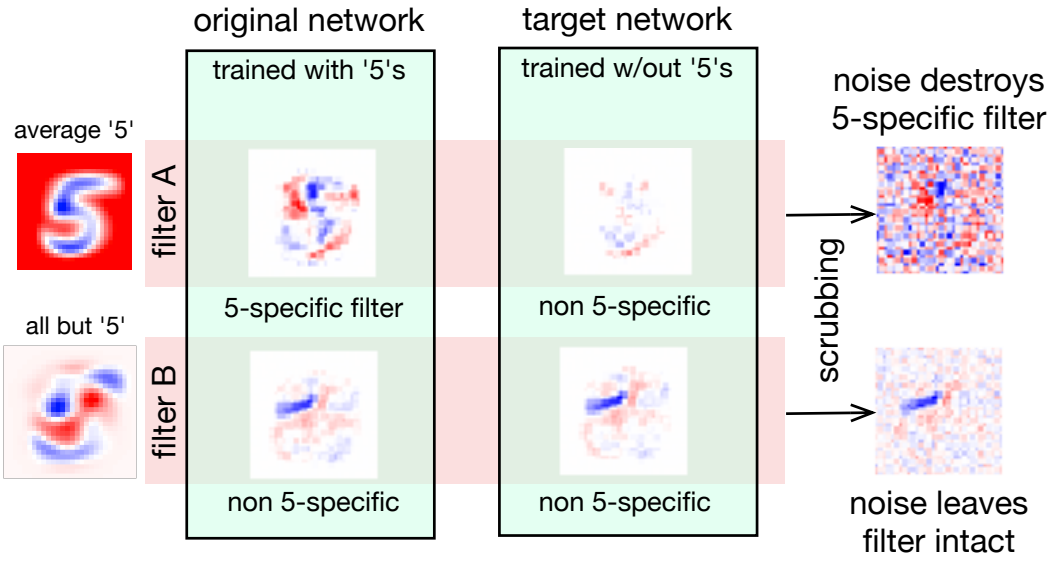

Figure 2.3: Filters of a network trained with the same random seed, with and without 5's. Some filters specialize to be 5-specific (filter A), and differ between the two networks, while others are not 5-specific (filter B), and remain identical. The scrubbing procedure brings original and target network closer by destroying 5-specific filters, effectively removing information about 5's.

2.2.2 Forgetting using a subset of the data

Once a model is trained, a request to forget $\mathcal{D}_f$ may be initiated by providing that cohort, as in the fictional service of Lacuna INC, but in general one may no longer have available the remainder of the dataset used for training, $\mathcal{D}_r$. However, assuming we are at a minimum of $L_{\mathcal{D}}(w)$, we have $\nabla L_{\mathcal{D}}(w) = 0$. Hence, we can rewrite $\nabla L_{\mathcal{D}_r}(w) = -\nabla L_{\mathcal{D}_f}(w)$ and $\nabla^2 L_{\mathcal{D}_f}(w) = \nabla^2 L_{\mathcal{D}}(w) - \nabla^2 L_{\mathcal{D}_r}(w)$. Using these identities, instead of recomputing the gradients and Hessian on the whole dataset, we can simply use those computed on the cohort to be forgotten, provided we cached the Hessian $\nabla^2 L_{\mathcal{D}}(w)$ we obtained at the end of the training on the original dataset $\mathcal{D}$. Note that this is not a requirement, although recommended in case the data to be remembered is no longer available.

2.2.3 Hessian approximation and Fisher Information

In practice, the Hessian is too expensive to compute for a DNN. In general, we cannot even ensure it is positive definite. To address both issues, we use the Levenberg-Marquardt semi-positive-definite approximation:

$$\nabla^2 L_{\mathcal{D}}(w) \simeq \mathbb{E}_{x\sim\mathcal{D}, y\sim p(y|x)}[\nabla_w \log p_w(y|x) \nabla_w \log p_w(y|x)^T]. \tag{2.9}$$

This approximation of the Hessian coincides with the Fisher Information Matrix (FIM) [Mar14], which opens the door to information-theoretic interpretations of the scrubbing procedure. Moreover, this approximation is exact for some problems, such as linear (logistic) regression.

2.3 Deep Network Scrubbing

Finally, we now discuss how to robustly apply the forgetting procedure eq. (2.8) to deep networks. We present two variants. The first uses the FIM of the network. However, since this depends on the network gradients, it may not be robust when the loss landscape is highly irregular. To solve this, we present a more robust method that attempts to minimize directly the Forgetting Lagrangian

eq. (2.4) through a variational optimization procedure.

Fisher forgetting: As mentioned in eq. (2.9), we approximate the Hessian with the Fisher Information Matrix. Since the FIM is too large to store in memory, we can compute its diagonal, or a better Kronecker-factorized approximation [MG15]. In our experiments, we find that the diagonal is not a good enough approximation of B for a full Newton step $h(w) = w - B^{-1}\nabla L_{\mathcal{D}_r}(w)$ in eq. (2.8). However, the diagonal is still a good approximation for the purpose of adding noise. Therefore, we simplify the procedure and take $h(w) = w$, while we still use the approximation of the FIM as the covariance of the noise. This results in the simplified scrubbing procedure:

$$\mathrm{S}(w) = w + (\lambda\sigma_h^2)^{\frac{1}{4}} F^{-1/4},$$

where F is the FIM (eq. 2.9) computed at the point w for the dataset $\mathcal{D}_r$. Here λ is a hyperparameter that trades off forgetting with the increase in error, as shown in Figure 3.3. Notice that, since $h(w) = w$, instead of a Newton step, this procedure relies on w and w' already being close, which hinges on the stability of SGD. This procedure may be interpreted as adding noise to destroy the weights that may have been informative about $\mathcal{D}_f$ but not $\mathcal{D}_r$ (Figure 2.3).

Variational forgetting: Rather than using the FIM, we may optimize for the noise in the Forgetting Lagrangian in eq. (2.4): Not knowing the optimal direction $w - w'$ along which to add noise (see Corollary 1), we may add the maximum amount of noise in all directions, while keeping the increase in the loss to a minimum. Formally, we minimize the proxy Lagrangian:

$$\mathcal{L}(\Sigma) = \mathbb{E}_{n\sim N(0,\Sigma)}\big[L_{\mathcal{D}_r}(w+n)\big] - \lambda \log|\Sigma|.$$

The optimal Σ may be seen as the FIM computed over a smoothed landscape. Since the noise is Gaussian, $\mathcal{L}(\Sigma)$ can be optimized using the local reparametrization trick [KSW15].

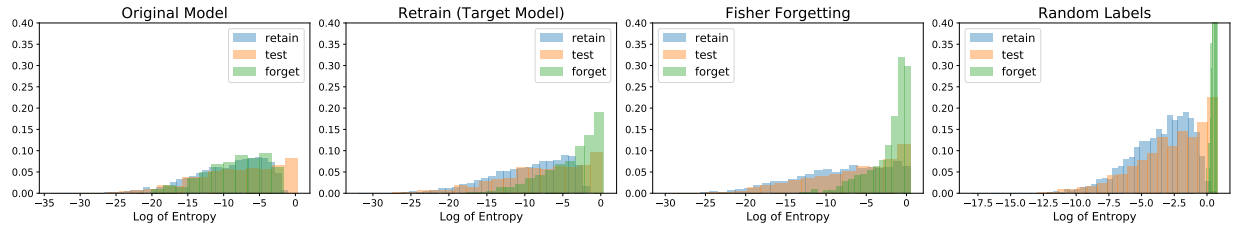

Figure 2.4: **Streisand Effect:** Distribution of the entropy of model output (confidence) on: the retain set $\mathcal{D}_r$, the forget set $\mathcal{D}_f$, and the test set. The **original model** has seen $\mathcal{D}_f$, and its prediction on it are very confident (matching the confidence on the train data). On the other hand, a model **re-trained** without seeing $\mathcal{D}_f$ has a lower confidence $\mathcal{D}_f$. After applying our scrubbing procedures (**Fisher** and **Variational**) to the original model, the confidence matches more closely the one we would have expected for a model that has never seen the data (column 3 is more similar to 2 than 1). For an incorrect method of forgetting, like training with random labels, we observe that the entropy of the forgotten samples is very degenerate and different from what we would have expected if the model had actually never seen those samples (it is concentrated only around chance level prediction entropy). That is, attempting to remove information about a particular cohort using this method, may actually end up providing more information about the cohort than the original model.

2.4 Experiments

We report experiments on MNIST, CIFAR10 [Kri09], Lacuna-10 and Lacuna-100, which we introduce and consist respectively of faces of 10 and 100 different celebrities from VGGFaces2 [CSX18] (see Appendix for details). On both CIFAR-10 and Lacuna-10 we choose to forget either an entire class, or a hundred images of the class.

For images (Lacuna-10 and CIFAR10), we use a small All-CNN (reducing the number of layers) [SDB14], to which we add batch normalization before each non-linearity. We pre-train on Lacuna-100/CIFAR-100 for 15 epochs using SGD with fixed learning rate of 0.1, momentum 0.9 and weight decay 0.0005. We fine-tune on Lacuna-10/CIFAR-10 with learning rate 0.01. To simplify the analysis, during fine-tuning we do not update the running mean and variance of batch normalization, and rather reuse the pre-trained ones.

2.4.1 Linear logistic regression

First, to validate the theory, we test the scrubbing procedure in eq. (2.7) on logistic regression, where the task is to forget data points belonging to one of two clusters comprising the class (see Figure 2.1). We train using a uniform random initialization for the weights and SGD with batch size 10, with early stopping after 10 epochs. Since the problem is low-dimensional, we easily approximate the distribution $p(w|\mathcal{D})$ and $p(w|\mathcal{D}_r)$ by training 100 times with different random seeds. As can be seen in Figure 2.1, the scrubbing procedure is able to align the two distributions with near perfect overlap, therefore preventing an attacker form extracting any information about the forgotten cluster. Notice also that, since we use early stopping, the algorithm had not yet converged, and exploiting the time dependency in eq. (2.7) rather than using the simpler eq. (2.8) is critical.

2.4.2 Baseline forgetting methods

Together with our proposed methods, we experiment with four other baselines which may intuitively provide some degree of forgetting. (i) **Fine-tune**: we fine-tune the model on the remaining data $\mathcal{D}_r$ using a slightly large learning rate. This is akin to catastrophic forgetting, where fine-tuning without $\mathcal{D}_f$ may make the model forget the original solution to $\mathcal{D}_f$ (more so because of the larger learning rate). (ii) **Negative Gradient**: we fine-tune on $\mathcal{D}$ by moving in the direction of increasing loss for samples in $\mathcal{D}_f$, which is equivalent to using a negative gradient for the samples to forget. This aims to damage features predicting $\mathcal{D}_f$ correctly. (iii) **Random Labels**: fine-tune the model on $\mathcal{D}$ by randomly resampling labels corresponding to images belonging to $\mathcal{D}_f$, so that those samples will get a random gradient. (iv) **Hiding**: we simply remove the row corresponding to the class to forget from the final classification layer of the DNN.

2.4.3 Readout functions used

Unlike our methods, these baselines do not come with an upper bound on the quantity of remaining information. It is therefore unclear how much information is removed. For this reason, we introduce

the following *read-out functions*, which may be used to gauge how much information they were able destroy: (i) **Error on the test set** $\mathcal{D}_{\text{test}}$ (ideally small), (ii) **Error on the cohort to be forgotten** $\mathcal{D}_f$ (ideally the same as a model trained without seeing $\mathcal{D}_f$), (iii) **Error on the residual** $\mathcal{D}_r$ (ideally small), (iv) **Re-learn time** (in epochs) time to retrain the scrubbed model on the forgotten data (measured by the time for the loss to reach a fixed threshold, ideally slow). If a scrubbed model can quickly recover a good accuracy, information about that cohort is likely still present in the weights. (v) **Model confidence:** We plot the distribution of model confidence (entropy of the output prediction) on the retain set $\mathcal{D}_r$, forget set $\mathcal{D}_f$ and the test set (should look similar to the confidence of a model that has never seen the data). (vi) **Information bound**: For our methods, we compute the information upper-bound about the cohort to be forgotten in NATs using Proposition 2.

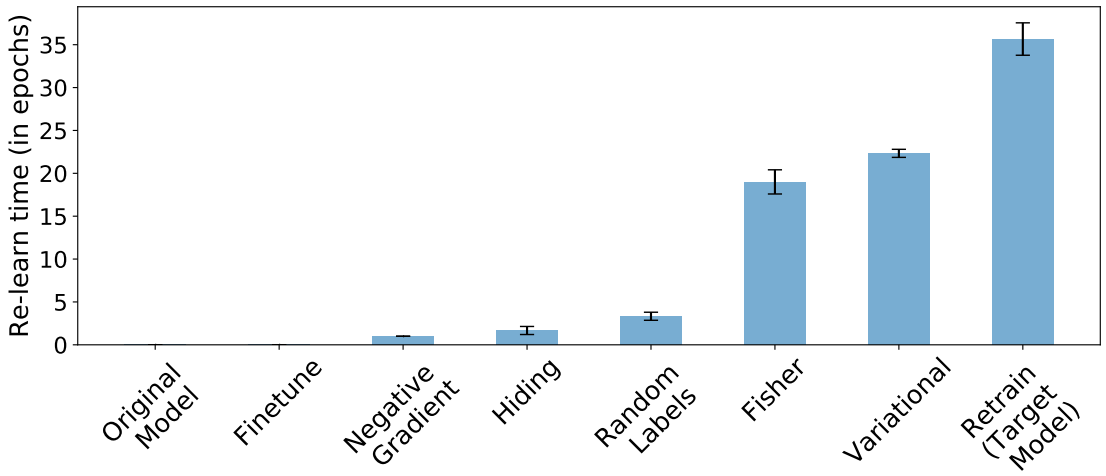

Figure 2.5: **Re-learn time** (in epochs) for various forgetting methods. All the baselines method can quickly recover perfect performance on $\mathcal{D}_f$, suggesting that they do not actually scrub information from the weights. On the other hand, the relearn time for our methods is higher, and closer to the one of a model that has never seen the data, suggesting that they remove more information.

2.4.4 Results

First, in Table 2.1 we show the results of scrubbing $\mathcal{D}_f$ from model trained on all the data. We test both the case we want to forget only a subset of 100-images from the class, and when we

want to forget a whole identity. We test on CIFAR-10 and Lacuna-10 with a network pretrained on CIFAR-100 and Lacuna-100 respectively.

Retrain denotes the gold standard which every scrubbing procedure should attempt to match for the error readout function. From the case where we want to scrub a subset of a class (first and third row of Retrain) it is clear that scrubbing does not mean merely achieving 100% error on $\mathcal{D}_f$. In fact, the reference Retrain has 15.3% and 19.3% error respectively on $\mathcal{D}_f$ and not 100%. Rather it means removing the information from the weights so that it behaves identically to a re-trained model. Forgetting by fine-tuning on $\mathcal{D}_r$, performs poorly on the error readout function (error on $\mathcal{D}_f$ and $\mathcal{D}_r$), suggesting that using catastrophic forgetting is the not the correct solution to selective forgetting.

The Negative Gradient and Random Labels methods perform well on the error readout function, however, when we use the re-learn time as a readout function (Figure 2.5) it becomes clear that very little information is actually removed, as the model relearn $\mathcal{D}_f$ very quickly. This suggests that merely scrubbing the activations by hiding or changing some output is not sufficient for selective forgetting; rather, information needs to be removed from the weights as anticipated. Moreover, applying an incorrect scrubbing procedure may make the images to forget *more* noticeable to an attacker (Streisand effect), as we can see by from the confidence values in Figure 2.4. The ease of forgetting a learnt cohort also depends on its size. In particular, in Figure 2.7 we observe that, for a fixed value of λ in eq. (2.1), the upper-bound on the information retained by the model after scrubbing increases with the size of the cohort to forget.

2.5 Discussion

Our approach is rooted in the connection between Differential Privacy (which our framework generalizes) and the stability of SGD. Forgetting is also intrinsically connected with information: Forgetting may also be seen as minimizing an upper-bound on the amount of information that the weights contain about $\mathcal{D}_f$ [AS19] and that an attacker may extract about that particular cohort

$\mathcal{D}_f$ using some readout function f. We have studied this problem from the point of view of Shannon Information, which allows for an easy formalization. However, it also has the drawback of considering the worst case of an attacker that has full knowledge of the training procedure and can use arbitrarily complex readout functions which may, for example, simulate all possible trainings of the network to extract the result. Characterizing forgetting with respect to a viable subset of realistic readout functions $f(w)$ is a promising area of research. We also exploit stability of the training algorithm after pretraining. Forgetting without the pretraining assumption is an interesting challenge, as it has been observed that slight perturbation of the initial critical learning period can lead to large difference in the final solution [ARS19, GAS19b].

2.6 Appendix

In Figure 3.6 we plot the loss function along a line joining the original model trained on all data and the ground-truth optimal model that was trained from the beginning without seeing the data to forget. The plots confirm that the two models are close to each other, as expected from the stability of SGD, and that the loss function in a neighborhood of the two points is convex (at least on the line joining the two). This justifies the hypotheses that we made to apply our forgetting method (derived in the case of a convex loss) to the more challenging case of deep networks.

To further confirm that our method can be applied to different architectures, in addition to the experiments on All-CNN that we show in the main paper we run additional experiments on a ResNet-18 architecture. In particular, Table 2.2 show the errors obtained by various forgetting techniques and Figure 2.10 shows the relearn time for a class after forgetting. In Figure 2.9, we show that ResNets too suffer from a Streisand effect if an improper forgetting procedure is applied.

2.6.1 Implementation details

The term $e^{-Bt}e^{At}$ in eq. (2.7) may diverge for $t \to \infty$ if the actual dynamics of the optimization algorithm do not exactly satisfy the hypotheses. In practice, we found it useful to replace it with

$e^{\text{clamp}(A-B,1/t)t}$, where $\text{clamp}(A, 1/t)$ clamps the eigenvalues of A so that they are smaller or equal to $1/t$, so that the expression does not diverge.[2] Notice also that in general $e^{-Bt}e^{At} \neq e^{(A-B)t}$, unless A and B commute, but we found this to be a good approximation in practice.

For DNNs, we perform experiments on Lacuna-10 and CIFAR-10 using All-CNN and ResNet. For All-CNN, we use the model proposed in [SDB14] and for ResNet we use the ResNet-18 model; however, we reduce the number of filters by half in each layer and remove the final residual block. For all the experiments we first pre-train the network on Lacuna-100 (CIFAR-100), and then fine-tune it on Lacuna-10 (CIFAR-10). We do not use data augmentation in any of the experiments. While pre-training the network we use a constant learning rate of 0.1 and train for 30 epochs and while finetuning we use a constant learning rate of 0.01 and train for 50 epochs. In all the experiments we use weight decay regularization with value 0.0005. We use PyTorch to perform all the experiments.

In Table 2.1, 'Original' model (OM) denotes the case when we train the model on the entire dataset, $\mathcal{D}$. 'Retrain' denotes the case when we train on $\mathcal{D}_r$ (we can do this by simply replacing the corresponding samples from the data loader). 'Finetune' (FT) is a possible scrubbing procedure when we use the Original model and fine-tune it on $\mathcal{D}_r$ (we can simply use the data loader from the Retrain case for fine-tuning). We run fine-tuning for 10 epochs with a learning rate 0.01 and weight decay 0.0005. 'Negative Gradient' denotes the case when we fine-tune on the complete data; however, for the samples belonging to $\mathcal{D}_f$ we simply multiply the gradient by -1 (i.e. maximizing the cross-entropy loss for the samples to forget). To prevent the loss from diverging, we clamp the maximum loss to chance level. We train for 10 epochs with learning rate 0.01 and weight decay 0.0005. In 'Random Labels,' (RL) we randomly shuffle the labels for the samples we want to forget. We use the same training hyper-parameters as the previous methods. For 'Hiding,' (HD) we replace the row corresponding to the class of the samples to be forgotten in the final classifier layer of the DNN with random initialization. Fisher denotes our method where we estimate the Fisher noise to

[2] If $A = SDS^T$ is an eigenvalue decomposition of the symmetric matrix A, we define $\text{clamp}(A, m) = S \min(D, m) S^T$.

add to the model. The Fisher noise is computed using a positive semi-definite approximation to the actual Fisher Information Matrix. We compute the trace of the Fisher Information which can be obtained by computing the expected outer product of the gradients of the DNN. In the experiments we observe that $F^{-\frac{1}{2}}$ approximates the Fisher noise better compared to $F^{-\frac{1}{4}}$. In Variational, we compute the optimal noise to be added for scrubbing a cohort by solving a variational problem. For Fisher and Variational forgetting we choose $\lambda = 5 \cdot 10^{-7}$.

In order to compute the Information bound (Fisher and Variational forgetting) we use the Local Forgetting Bound i.e. we apply the scrubbing procedure to both the original and the retrain (target) model and then compute the KL divergence of the two distributions. We use the same random seed while training both the models and use the same scrubbing procedures for both, that is, $S = S_0$. Fisher and Variational forgetting method essentially consists of adding noise to the model weights, i.e., $S(w) = w + n$ where $n \sim N(0, \Sigma)$. Let w_o, Σ_o and w_r, Σ_r denote the weights and noise covariance for the original and the target retrained model respectively, then the Information bound is given by the following relation:

$$\mathrm{KL}\left(N(w_o, \Sigma_o) \,\|\, N(w_r, \Sigma_r)\right) = \frac{1}{2}\Big(\mathrm{tr}(\Sigma_r^{-1}\Sigma_o) + (w_r - w_o)^T \Sigma_r^{-1}(w_r - w_o) - k + \log \frac{|\Sigma_r|}{|\Sigma_o|}\Big),$$

where k is the dimension of w. This bound should be computed for multiple values of the initial random seed and then averaged to obtain the Local Forgetting Bound in Proposition 2. In our experiments we compute the bound using a single random seed.

To compute the Relearn-time we train the scrubbed model on the dataset $\mathcal{D}$ for 50 epochs using a constant learning rate 0.01. We report the first epoch when the loss value falls below a certain threshold as the relearn-time.

2.6.1.1 Pre-training improves the forgetting bound

Our local forgetting bound assumes stability of the algorithm, which may not be guaranteed when training a large deep network over many epochs. This can be obviated by pre-training the network,

so all paths will start from a common configuration of weights and training time is decreased, and with it the opportunity for paths to diverge. As we show next, the resulting bound is greatly improved. The drawback is that the bound cannot guarantee forgetting of any information contained in the pre-training dataset (that is, $\mathcal{D}_f$ needs to be disjoint from $\mathcal{D}_{\text{pretrain}}$).

2.6.1.2 Datasets

We report experiments on MNIST, CIFAR10 [Kri09], Lacuna-10 and Lacuna-100 which we introduce. **Lacuna-10** consists of face images of 10 celebrities from VGGFaces2 [CSX18], randomly sampled with at least 500 images each. We split the data into a test set of 100 samples for each class, while the remaining form the training set. Similarly, **Lacuna-100** randomly samples 100 celebrities with at least 500 images each. We resize the images to 32x32. There is no overlap between the two Lacuna datasets. We use Lacuna-100 to pre-train (and hence assume that we do not have to forget it), and fine-tune on Lacuna-10. The scrubbing procedure is required to forget some or all images for one identity in Lacuna-10. On both CIFAR-10 and Lacuna-10 we choose to forget either the entire class '5,' which is chosen at random, or a hundred images of the class.

2.6.2 Proofs

Proposition 1 Let the forgetting set $\mathcal{D}_f$ be a random variable, for instance, a random sampling of the data to forget. Let Y be an attribute of interest that depends on $\mathcal{D}_f$. Then,

$$I(Y; f(S(w))) \leq$$
$$\mathbb{E}_{D_f}[\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \parallel P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right)].$$

Proof. We have the following Markov Chain:

$$Y \longleftarrow \mathcal{D}_f \longrightarrow w \longrightarrow S(w) \longrightarrow f(S(w))$$

We assume that the retain set $\mathcal{D}_r$ is fixed and thus deterministic. Also, $\mathcal{D}_f \rightarrow w$ implies that we first sample $\mathcal{D}_f$ and then use $\mathcal{D} = \mathcal{D}_r$(fixed and known) $\cup\, \mathcal{D}_f$ for training w. From Data Processing

Inequality, we know that: $I(Y; f(S(w))) \leq I(\mathcal{D}_f; f(S(w)))$. In order to prove the proposition we bound the RHS term (i.e $I(\mathcal{D}_f; f(S(w)))$).

$$
\begin{aligned}
&= I(\mathcal{D}_f; f(S(w))) \\
&= \mathbb{E}_{\mathcal{D}_f}\big[\,\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S(w)))\right)\big] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f(S(w))|\mathcal{D}_f\cup\mathcal{D}_r)}\Big[\log \frac{p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f(S(w)))}\Big] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f(S(w))|\mathcal{D}_f\cup\mathcal{D}_r)}\Big[\log \frac{p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f(S_0(w))|\mathcal{D}_r)} \\
&\qquad\qquad + \log \frac{p(f(S_0(w))|\mathcal{D}_r)}{p(f(S(w)))}\Big] \\
&= \mathbb{E}_{\mathcal{D}_f}\big[\,\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S_0(w))|\mathcal{D}_r)\right)\big] \\
&\qquad\qquad - \mathrm{KL}\left(p(f(S_0(w))) \,\|\, p(f(S(w))|\mathcal{D}_r)\right) \\
&\leq \mathbb{E}_{\mathcal{D}_f}\big[\,\mathrm{KL}\left(p(f(S(w))|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f(S_0(w))|\mathcal{D}_r)\right)\big]
\end{aligned}
$$

where the last inequality follows from the fact that KL-divergence is always non-negative.

□

Lemma 1 For any function $f(w)$ have

$$\mathrm{KL}\left(P(f(\mathrm{S}(w))|\mathcal{D}) \,\|\, P(f(\mathrm{S}_0(w))|\mathcal{D}_r)\right) \qquad \leq \qquad \mathrm{KL}\left(P(\mathrm{S}(w)|\mathcal{D}) \,\|\, P(\mathrm{S}_0(w)|\mathcal{D}_r)\right),$$

Proof. For simplicity sake, we will consider the random variables to be discrete. To keep the notation uncluttered, we consider the expression

$$\mathrm{KL}\left(Q(f(x)) \,\|\, R(f(x))\right) \leq \mathrm{KL}\left(Q(x) \,\|\, R(x)\right)$$

where $Q(w) = P(\mathrm{S}(w)|D)$ and $R(w) = P(\mathrm{S}_0(w)|D_{-k})$. Let $W_c = \{w|f(w) = c\}$, so that we have $Q(f(w) = c) = \sum_{w\in W_c} Q(w)$ and similarly $R(f(w) = c) = \sum_{w\in W_c} R(w)$. Rewriting the

LHS with this notation, we get:

$$\begin{aligned}
&\mathrm{KL}\left(Q(f(w)) \,\|\, R(f(w))\right) \\
&\quad = \sum_c Q(f(w)=c) \log \frac{Q(f(w)=c)}{R(f(w)=c)} \\
&\quad = \sum_c Q(f(w)=c) \log \frac{\sum_{w \in W_c} Q(w)}{\sum_{w \in W_c} R(w)}
\end{aligned} \tag{2.10}$$

We can similarly rewrite the RHS:

$$\mathrm{KL}\left(Q(w) \,\|\, R(w)\right) = \sum_c \sum_{w \in W_c} \log \frac{Q(w)}{R(w)} \tag{2.11}$$

From the log-sum inequality, we know that for each c in eq. (2.10) and eq. (2.11):

$$\Big(\sum_{w \in W_c} Q(w)\Big) \log \frac{\sum_{w \in W_c} Q(w)}{\sum_{w \in W_c} R(w)} \leq \sum_{w \in W_c} Q(w) \log \frac{Q(w)}{R(w)} \tag{2.12}$$

Summation over all c on both sides of the inequality concludes the proof. □

Proposition 2 Let $A(\mathcal{D})$ be a (possibly stochastic) training algorithm, the outcome of which we indicate as $w = A(\mathcal{D}, \epsilon)$ for some deterministic function A and $\epsilon \sim N(0, I)$, for instance a random seed. Then, we have $P(S(w)|\mathcal{D}) = \mathbb{E}_\epsilon[P(S(w)|\mathcal{D}, \epsilon)]$. We have the bound:

$$\mathrm{KL}\left(P(S(w)|\mathcal{D}) \,\|\, P(S_0(w)|\mathcal{D}_r)\right) \qquad \leq \qquad \mathbb{E}_\epsilon\Big[\, \mathrm{KL}\left(P(S(w)|\mathcal{D}, \epsilon) \,\|\, P(S_0(w)|\mathcal{D}_r, \epsilon)\right)\Big]$$

Proof. To keep the notation uncluttered,we rewrite the inequality as:

$$\mathrm{KL}\left(Q(w) \,\|\, R(w)\right) \leq \mathbb{E}_\epsilon\Big[\, \mathrm{KL}\left(Q(w|\epsilon) \,\|\, R(w|\epsilon)\right)\Big]$$

where $Q(w) = P(\mathrm{S}(w)|\mathcal{D})$ and $R(w) = P(\mathrm{S}(w)|\mathcal{D}_r)$ The LHS can be equivalently written as:

$$\begin{aligned}
&\mathrm{KL}\left(Q(w) \,\|\, R(w)\right) = \\
&\quad = \int Q(w) \log \frac{Q(w)}{R(w)}) dw \\
&\quad = \int \mathbb{E}_\epsilon[Q(w|\epsilon)] \log \frac{\mathbb{E}_\epsilon[Q(w|\epsilon)]}{\mathbb{E}_\epsilon[P(w|\epsilon)]} dw
\end{aligned}$$

$$
\begin{aligned}
&\overset{(*)}{\leq} \int \mathbb{E}_\epsilon\Big[Q(w|\epsilon)\log\frac{Q(w|\epsilon)}{P(w|\epsilon)}\Big]dw \\
&= \mathbb{E}_\epsilon\Big[\int Q(w|\epsilon)\log\frac{Q(w|\epsilon)}{P(w|\epsilon)}dw\Big] \\
&= \mathbb{E}_\epsilon\big[\,\mathrm{KL}\left(Q(w|\epsilon)\,\|\,R(w|\epsilon)\right)\big],
\end{aligned}
$$

where in (*) we used the log-sum inequality. □

Proposition 3 Let the loss be $L_{\mathcal{D}}(w) = L_{\mathcal{D}_f}(w) + L_{\mathcal{D}_r}(w)$, and assume both $L_{\mathcal{D}}(w)$ and $L_{\mathcal{D}_r}(w)$ are quadratic. Assume that the optimization algorithm $A_t(\mathcal{D}, \epsilon)$ at time t is given by the gradient flow of the loss with a random initialization, and let $h(w)$ be the function:

$$h(w) = e^{-Bt}e^{At}d + e^{-Bt}(d - d_r) - d_r,$$

where $A = \nabla^2 L_{\mathcal{D}}(w)$, $B = \nabla^2 L_{\mathcal{D}_r}(w)$, $d = A^{-1}\nabla_w L_{\mathcal{D}}$ and $d_r = B^{-1}\nabla_w L_{\mathcal{D}_r}$, and e^{At} denotes the matrix exponential. Then $h(A_t(\mathcal{D}, \epsilon)) = A_t(\mathcal{D}_r, \epsilon)$ for all random initializations ϵ and all times t.

Proof. Since $L_{\mathcal{D}}$ and $L_{\mathcal{D}_r}$ are quadratic, assume without loss of generality that:

$$
\begin{aligned}
L_{\mathcal{D}}(w) &= \frac{1}{2}(w - w_A^*)^T A(w - w_A^*) \\
L_{\mathcal{D}_f}(w) &= \frac{1}{2}(w - w_B^*)^T B(w - w_B^*)
\end{aligned}
$$

Since the training dynamic is given by a gradient flow, the training path is the solution to the differential equation:

$$
\begin{aligned}
\dot{w_A}(t) &= A(w(t) - w_A^*) \\
\dot{w_B}(t) &= B(w(t) - w_B^*)
\end{aligned}
$$

which is given respectively by:

$$
\begin{aligned}
w_A(t) &= w_A^* + e^{-At}(w_0 - w_A^*) \\
w_B(t) &= w_B^* + e^{-Bt}(w_0 - w_B^*)
\end{aligned}
$$

We can compute w_0 from the first expression:

$$w_0 = e^{At}(w_A(t) - w_A^*) + w_A^* = e^{At}d_A + w_A^*,$$

where we defined $d_A = w_A(t) - w_A^* = A^{-1}\nabla_w L_{\mathcal{D}}(w_A(t))$. We now replace this expression of w_0 in the second expression to obtain:

$$\begin{aligned} w_B(t) &= e^{-Bt}e^{At}d_A + e^{-Bt}(w_A^* - w_B^*) + w_B^* \\ &= w_A(t) + e^{-Bt}e^{At}d_A + e^{-Bt}(d_B - d_A) - d_B \end{aligned}$$

where $d_B := w_A(t) - w_B^* = B^{-1}\nabla_w L_{\mathcal{D}_r}(w_A(t))$. □

Proposition 4

Assume that $h(w)$ is close to w' up to some normally distributed error $h(w) - w' \sim N(0, \Sigma_h)$, and assume that $L_{\mathcal{D}_r}(w)$ is (locally) quadratic around $h(w)$. Then the optimal scrubbing procedure in the form $S(w) = h(w) + n, \quad n \sim N(0, \Sigma)$,that minimizes the Forgetting Lagrangian

$$\mathcal{L} = \mathbb{E}_{\tilde{w}\sim S(w)}[L_{\mathcal{D}_r}(\tilde{w})] + \lambda\, \mathbb{E}_\epsilon\Big[\,\mathrm{KL}\left(P(S(w)|\mathcal{D},\epsilon)\,\|\,P(S_0(w)|\mathcal{D}_r,\epsilon)\right)\Big]$$

is obtained when $\Sigma B \Sigma = \lambda \Sigma_h$, where $B = \nabla^2 L_{\mathcal{D}_r}(k)$. In particular, if the error is isotropic, that is $\Sigma_h = \sigma_h^2 I$ is a multiple of the identity, we have $\Sigma = \sqrt{\lambda \sigma_h^2} B^{-1/2}$.

Proof. We consider the following second order approximation to the loss function in the neighbourhood of the parameters at convergence:

$$\begin{aligned} &\mathbb{E}_{\tilde{w}\sim S(w)}[L_{\mathcal{D}_r}(\tilde{w})] = \mathbb{E}_{n\sim N(0,\Sigma)}[L_{\mathcal{D}_r}(h(w)+n)] \\ &= \mathbb{E}_{n\sim N(0,\Sigma)}\Big[L_{D_r}(h(w)) + \\ &\qquad \nabla L_{D_r}(h(w))^T \Sigma^{\frac{1}{2}} n + \frac{1}{2}(\Sigma^{\frac{1}{2}}n)^T B(\Sigma^{\frac{1}{2}}n) + o(n^2)\Big] \\ &\simeq L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\mathbb{E}_{n\sim N(0,\Sigma)}\Big[(\Sigma^{\frac{1}{2}}n)^T B(\Sigma^{\frac{1}{2}}n)\Big] \end{aligned}$$

$$= L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\mathbb{E}_{n\sim N(0,\Sigma)}\Big[\operatorname{tr}(B(\Sigma^{\frac{1}{2}}nn^T\Sigma^{\frac{1}{2}}))\Big]$$
$$= L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\operatorname{tr}(B\Sigma)$$

Recall that we take S_0 to be $S_0(w) = w + n'$ with $n' \sim N(0, \Sigma)$, so that $P(S_0(w)|\mathcal{D}_r, \epsilon) \sim N(w, \Sigma)$. Thus, we get the following expression for the KL divergence term:

$$\mathbb{E}_\epsilon\Big[\operatorname{KL}\left(P(S(w)|\mathcal{D}, \epsilon) \,\|\, P(S_0(w)|\mathcal{D}_r, \epsilon)\right)\Big]$$
$$= \frac{1}{2}\mathbb{E}_\epsilon\Big[(h(w) - w')^T\Sigma^{-1}(h(w) - h(w'))\Big]$$
$$= \frac{1}{2}\operatorname{tr}(\Sigma^{-1}\Sigma_h),$$

where in the last equality we have used that by hypothesis $h(w) - w' \sim N(0, \Sigma_h)$. Combining the two terms we get:

$$\mathcal{L} = L_{\mathcal{D}_r}(h(w)) + \frac{1}{2}\operatorname{tr}(B\Sigma) + \frac{1}{2}\operatorname{tr}(\Sigma^{-1}\Sigma_h)$$

We now want to find the optimal covariance Σ of the noise to add in order to forget. Setting $\nabla_\Sigma\mathcal{L} = 0$, we obtain the following optimality condition $\Sigma B\Sigma = \lambda\Sigma_h$. If we further assume the error Σ_h to be isotropic, that is, $\Sigma_h = \sigma_h^2 I$, then this condition simplifies to $\Sigma = \sqrt{\lambda\sigma_h^2}B^{-1/2}$. $\square$

	Metrics	OM	Retrain (target)	FT	NG	RL	HD	Fisher (ours)	Variational (ours)
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	10.2 ±0.5	10.3 ±0.4	10.2 ±0.6	10.0 ±0.4	12.0 ±0.2	18.2 ±0.4	14.5 ±1.6	13.7 ±1.0
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	15.3 ±0.6	0.0 ±0.0	0.0 ±0.0	6.0 ±3.6	100 ±0.0	8.0 ±2.7	8.0 ±3.6
All-CNN	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.1 ±0.1	6.5 ±0.0	4.8 ±2.8	4.8 ±2.4
	Info-bound (kNATs)							3.3 ±1.1	3.0 ±0.5
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	10.2 ±0.5	18.4 ±0.6	10.0 ±0.6	18.4 ±0.6	18.8 ±0.6	18.2 ±0.4	21.0 ±1.3	20.9 ±0.4
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	0.0 ±0.0	100 ±0.2	90.2 ±1.5	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
All-CNN	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	3.3 ±2.3	2.8 ±1.4
	Info-bound (kNATs)							13.2 ±2.8	12.0 ±2.9
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	14.4 ±0.6	14.6 ±0.7	13.5 ±0.1	13.4 ±0.1	13.8 ±0.1	21.0 ±0.5	19.8 ±2.8	20.9 ±4.8
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	19.3 ±4.5	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	100.0 ±0.0	23.3 ±2.1	6.3 ±2.5
All-CNN	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	9.9 ±0.1	8.0 ±4.3	8.8 ±5.2
	Info-bound (kNATs)							33.4 ±16.7	21.6 ±5.2
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	14.4 ±0.7	21.1 ±0.6	14.3 ±0.1	20.2 ±0.1	20.7 ±0.4	21.0 ±0.5	23.7 ±0.9	22.8 ±0.3
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	10.0 ±0.4	100 ±0.2	88.1 ±4.6	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
All-CNN	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	2.6 ±1.8	2.3 ±0.7
	Info-bound (kNATs)							458.1 ±172.2	371.5 ±51.3

Table 2.1: **Original model (OM)** is the model trained on all data $\mathcal{D} = \mathcal{D}_f \sqcup \mathcal{D}_r$. The forgetting algorithm should scrub information from its weights. **Retrain** denotes the model obtained by retraining from scratch on $\mathcal{D}_r$, without knowledge of $\mathcal{D}_f$. The metric values in the Retrain column is the optimal value which every other scrubbing procedure should attempt to match. We consider the following forgetting procedures: **Fine-tune (FT)** denotes fine-tuning the model on $\mathcal{D}_r$. **Negative Gradient (NG)** denotes fine-tuning on $\mathcal{D}_f$ by moving in the direction of increasing loss. **Random Label (RL)** denotes replacing the labels of the class with random labels and then fine-tuning on all $\mathcal{D}$. **Hiding (HD)** denotes simply removing the class from the final classification layer. **Fisher** and **Variational** are our proposed methods, which add noise to the weights to destroy information about $\mathcal{D}_f$ following the Forgetting Lagrangian. We benchmark these methods using several readout functions: errors on $\mathcal{D}_f$ and $\mathcal{D}_r$ after scrubbing, time to retrain on the forgotten samples after scrubbing, distribution of the model entropy. In all cases, the read-out of the scrubbed model should be closer to the target retrained model than to the original. Note that our methods also provide an upper-bound to the amount of information remaining. We report mean/std over 3 random seeds.

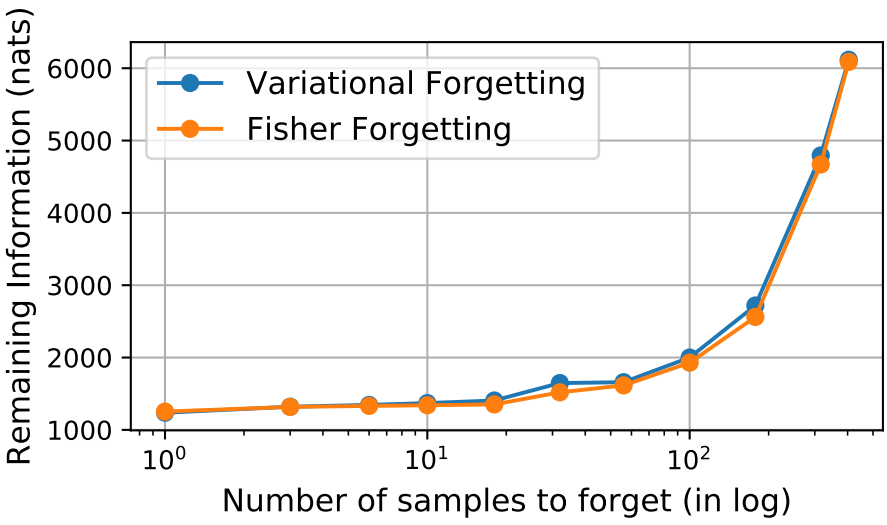

Figure 2.6: **Difficulty of forgetting increases with cohort size.** For a fixed λ (forgetting parameter), we plot the amount of information remaining after scrubbing as a function of the cohort size ($|\mathcal{D}_f|$).

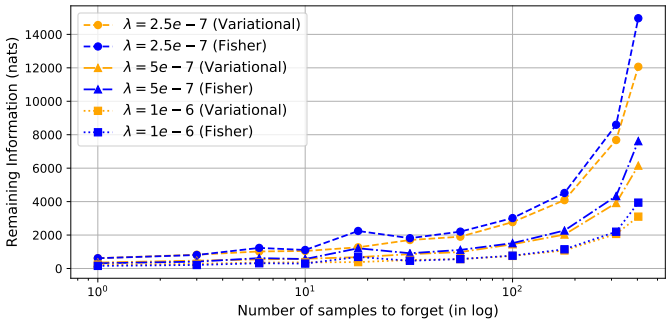

Figure 2.7: **Difficulty of forgetting increases with cohort size (for ResNet)** We plot the upper-bound on the remaining information (i.e. the information the model contains about the cohort to forget after scrubbing) as a function of the number of samples to forget for class '5' for different values of λ (Forgetting Lagrangian parameter). Increasing the value of λ decreases the remaining information, but increases the error on the remaining samples. The number of samples to forget in the plot varies between one sample and the whole class (404 samples).

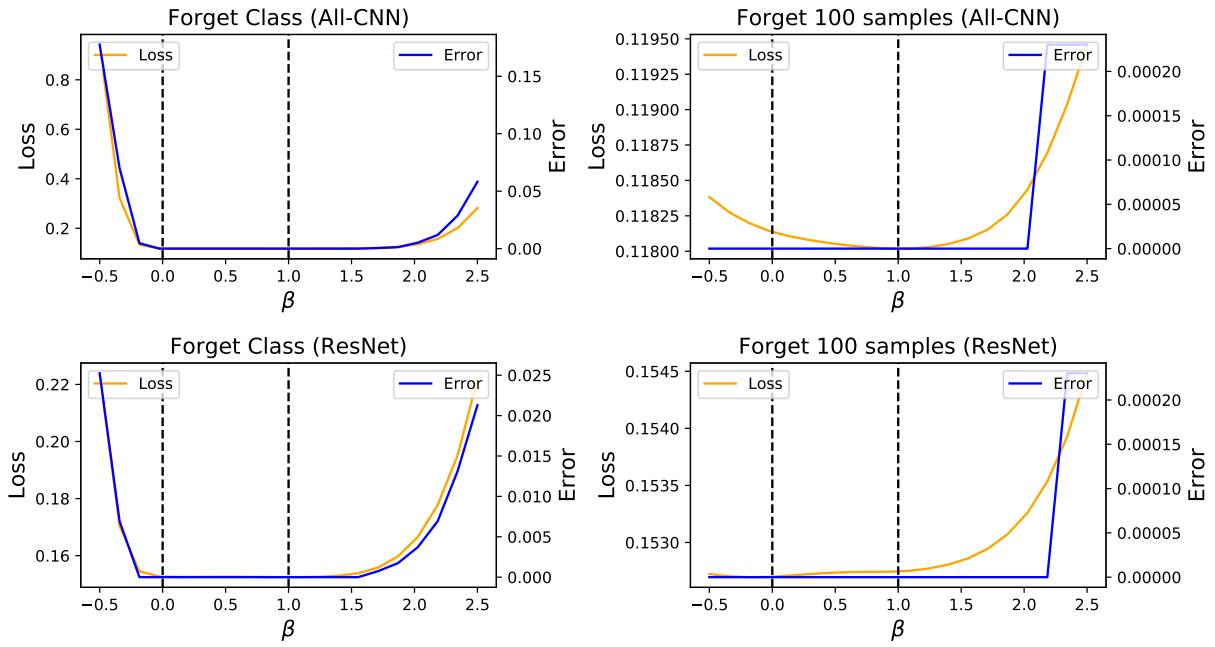

Figure 2.8: **Loss landscape of $L_{\mathcal{D}_r}$.** Plot of loss and error on the dataset $\mathcal{D}_r$ of the remaining samples, interpolating along the line joining the *original model* (at $t = 0$) and the (target) *retrained model* at $t = 1$. Precisely, let w_o be the original model and let w_r be the retrained model, we plot $L(w(t))$, where $w(t) = (1 - t) \cdot w_o + t \cdot w_r$ by varying $t \in [-0.5, 2.5]$. We observe that the loss along the line joining the original and the target model is convex, and almost flat throughout. In particular, there exists at least an optimal direction (the one joining the two models) along which we can add noise without increasing the loss on the remaining data, and which would allow to forget the extra information. This inspires and justifies our forgetting procedure.

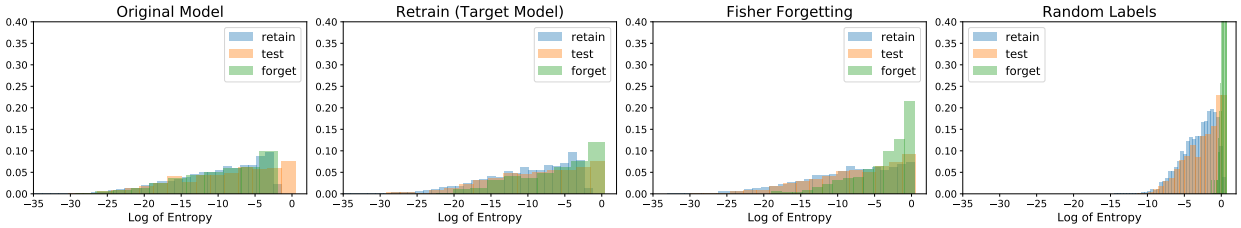

Figure 2.9: **Same plot as in Figure 2.4 but with a different architecture: Streisand Effect for ResNet model**

Table 2.2: **Same experiment as Table 2.1 but with a different architecture: Error readout functions for ResNet trained on Lacuna-10 and CIFAR-10.**

	Metrics	**OM**	**Retrain** (target)	**FT**	**NG**	**RL**	**HD**	**Fisher** (ours)	**Variational** (ours)
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	18.0 ±0.5	18.2 ±0.3	18.1 ±0.2	18.0 ±0.5	19.3 ±0.7	25.1 ±0.1	24.5 ±0.7	28.2 ±3.2
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	29.0 ±0.1	0.0 ±0.0	0.0 ±0.0	14.3 ±6.7	100 ±0.0	17.7 ±6.7	3.0 ±1.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.2 ±0.0	6.5 ±0.0	10.4 ±0.8	11.7 ±5.8
	Info-bound (kNATs)							2.6 ±0.1	2.6 ±0.3
Lacuna-10	Error on $\mathcal{D}_{\text{test}}$ (%)	18.0 ±0.5	24.5 ±0.4	18.1 ±0.4	25.0 ±0.4	24.6 ±0.8	22.0 ±5.2	26.6 ±1.0	26.8 ±0.6
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	0.0 ±0.0	99.7 ±0.3	90.3 ±1.5	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.2 ±0.4	1.0 ±0.9
	Info-bound (kNATs)							33.2 ±2.2	25.7 ±7.8
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	17.3 ±0.2	17.5 ±1.0	17.3 ±0.5	17.2 ±0.3	17.9 ±1.2	23.6 ±1.3	21.9 ±2.2	23.1 ±2.3
Scrub 100	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	25.2 ±5.2	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	100.0 ±0.0	16.0 ±7.2	13.3 ±7.4
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	8.7 ±1.9	7.6 ±4.0	9.8 ±3.9
	Info-bound (kNATs)							46.0 ±19.3	29.8 ±8.0
CIFAR-10	Error on $\mathcal{D}_{\text{test}}$ (%)	17.1 ±0.3	22.8 ±0.0	17.2 ±0.1	22.8 ±0.1	23.1 ±0.2	22.8 ±0.1	25.8 ±0.1	25.1 ±0.3
Forget class	Error on $\mathcal{D}_f$ (%)	0.0 ±0.0	100 ±0.0	0.7 ±0.5	100 ±0.1	94.2 ±5.7	100.0 ±0.0	100.0 ±0.0	100.0 ±0.0
ResNet	Error on $\mathcal{D}_r$ (%)	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	0.0 ±0.0	3.8 ±0.4	2.4 ±1.0
	Info-bound (kNATs)							235.4 ±4.9	234.8 ±6.1

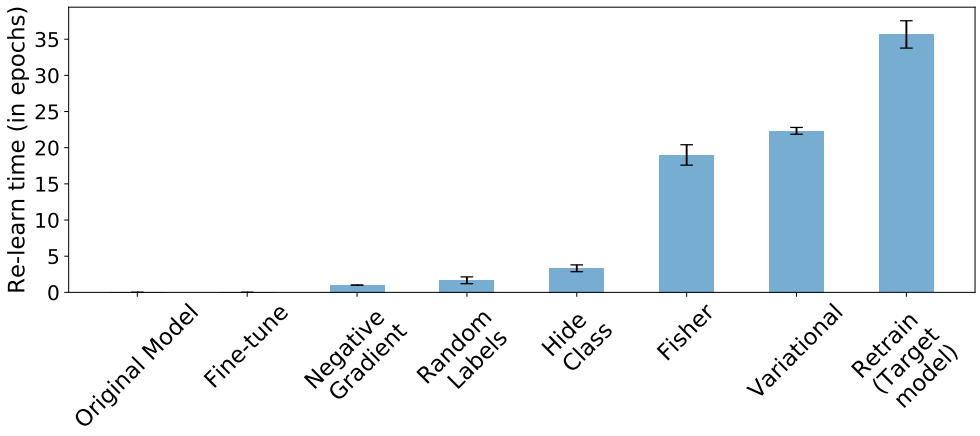

Figure 2.10: **Same plot as in Figure 2.5 but with a different architecture: Re-learn time (in epochs) for various forgetting methods using ResNet model.**

CHAPTER 3

Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations

In this chapter, we propose a forgetting procedure which compute a single shot update to remove information about a cohort of data from a fine-tuned neural network using the idea of linearization of deep networks from NTK. Even though the models trained are highly non-linear the unlearning perturbation obtained from the linearizing the output of the network still provides sufficiently accurate unlearning performance against different readout functions.

More formally, let a dataset $\mathcal{D}$ be partitioned into a subset $\mathcal{D}_f$ to be forgotten, and its complement $\mathcal{D}_r$ to be retained, $\mathcal{D} = \mathcal{D}_f \sqcup \mathcal{D}_r$. A (possibly stochastic) training algorithm A takes $\mathcal{D}_f$ and $\mathcal{D}_r$ and outputs a weight vector $w \in \mathbb{R}^p$:

$$A(\mathcal{D}_f, \mathcal{D}_r) \to w.$$

Assuming an attacker knows the training algorithm A (*e.g.*, stochastic gradient descent, or SGD), the weights w, and the retainable data $\mathcal{D}_r$, she can exploit their relationship to recover information about $\mathcal{D}_f$, at least for state-of-the-art deep neural networks (DNNs) using a "readout" function [GAS19a], that is, a function $R(w)$ that an attacker can apply on the weights of a DNN to extract information about the dataset.

For example, an attacker may measure the confidence of the classifier on an input, or measure the time that it takes to fine-tune the network on a given subset of samples to decide whether that input or set was used to train the classifier (membership attack). We discuss additional examples in Section 3.3.4. Ideally, the forgetting procedure should be robust to different choices of

readout functions. Recent work [GAS19a, GGH19], introduces a "scrubbing procedure" (forgetting procedure or data deletion/removal) $S_{\mathcal{D}_f}(w) : \mathbb{R}^p \to \mathbb{R}^p$, that attempts to remove information about $\mathcal{D}_f$ from the weights, *i.e.*,

$$A(\mathcal{D}_f, \mathcal{D}_r) \to w \to S_{\mathcal{D}_f}(w)$$

with an upper-bound on the amount of information about $\mathcal{D}_f$ that can be extracted after the forgetting procedure, provided the attack has access to the scrubbed weights $S_{\mathcal{D}_f}(w)$, a process called "white-box attack."

Bounding the information that can be extracted from a white-box attack is often complex and may be overly restrictive: Deep networks have large sets of equivalent solutions – a "null space" – that would give the same activations on all test samples. Changes in $\mathcal{D}_f$ may change the position of the weights in the null space. Hence, the position of the weight in the null-space, even if irrelevant for the input-output behavior, may be exploited to recover information about $\mathcal{D}_f$.

This suggests that the study of forgetting should be approached from the perspective of the *final activations*, rather than the weights, since there could be infinitely many different models that produce the same input-output behavior, and we are interested in preventing attacks that affect the behavior of the network, rather than the specific solution to which the training process converged. More precisely, denote by $f_w(x)$ the final activations of a network on a sample x (for example the softmax or pre-softmax vector). We assume that an attacker makes queries on n images $\mathbf{x} = (x_1, \ldots, x_n)$, and obtains the activations $f_w(\mathbf{x})$. The pipeline then is described by the Markov Chain

$$A(\mathcal{D}_f, \mathcal{D}_r) \to w \to S_{\mathcal{D}_f}(w) \to f_{S_{\mathcal{D}_f}(w)}(\mathbf{x}).$$

The key question now is to determine how much information can an attacker recover about $\mathcal{D}_f$, starting from the final activations $f_{S_{\mathcal{D}_f}(w)}(\mathbf{x})$? We provide a new set of bounds that quantifies the average information per query an attacker can extract from the model.

The forgetting procedure we propose is obtained using the Neural Tangent Kernel (NTK). We show that this forgetting procedure is able to handle the null space of the weights better than previous

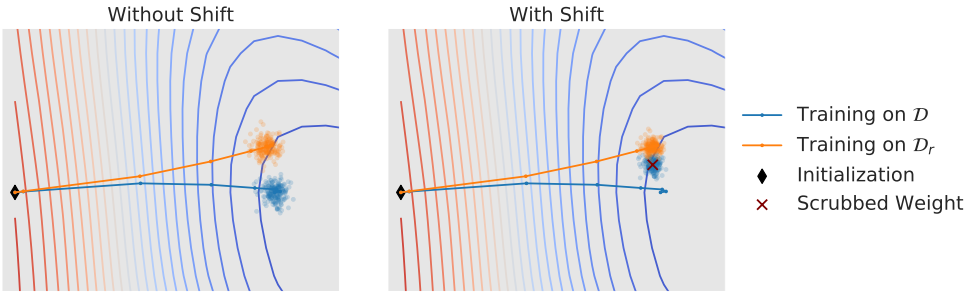

Figure 3.1: **Scrubbing procedure:** PCA-projection of training paths on $\mathcal{D}$ (blue), $\mathcal{D}_r$ (orange) and the weights after scrubbing, using **(Left)** The Fisher method of [GAS19a], and **(Right)** the proposed scrubbing method. Our proposed scrubbing procedure (red cross) moves the model towards $w(\mathcal{D}_r)$, which reduces the amount of noise (point cloud) that needs to be added to achieve forgetting.

approaches when using over-parametrized models such as DNNs. In experiments, we confirm that it works uniformly better than previous proposals on all forgetting metrics introduced [GAS19a], *both in the white-box and black-box case* (Figure 3.1).

Note that one may think of forgetting in a black-box setting as just changing the activations (*e.g.*, adding noise or hiding one class output) so that less information can be extracted. This, however, is not proper forgetting as the model still contains information, it is just not visible outside. We refer to forgetting as removing information from the weights, but we provide bounds for how much information can be extracted after scrubbing in the black-box case, and show that they are order of magnitudes smaller than the corresponding bounds for white boxes for the same target accuracy.

3.1 Out of the box forgetting

In this section, we derive an upper-bound for how much information can be extracted by an attacker that has black-box access to the model, that is, they can query the model with an image, and obtain the corresponding output. We will then use this to design a forgetting procedure (Section 6.4).

While the problem itself may seem trivial — can the relation $A(\mathcal{D}_f, \mathcal{D}_r) = w$ be inverted to extract $\mathcal{D}_f$? — it is made more complex by the fact that the algorithm is stochastic, and that

the map may not be invertible, but still partially invertible, that is, only a subset of information about $\mathcal{D}_f$ can be recovered. Hence, we employ a more formal information-theoretic framework, inspired by [GAS19a] and that in turns generalizes Differential Privacy [DR14]. We provide a-posteriori bounds which provide tighter answers, and allow one to design and benchmark scrubbing procedures even for very complex models such as deep networks, for which a-priori bounds would be impossible or vacuous. In this work, we focus on a-posteriori bounds for Deep Networks and use them to design a scrubbing procedure.

3.1.1 Information Theoretic formalism

We start by modifying the framework of [GAS19a], developed for the weights of a network, to the final activations. We expect an adversary to use a readout function applied to the final activations. Given a set of images $\mathbf{x} = (x_0, \ldots, x_n)$, we denote by $f_w(\mathbf{x}) = (f(x_0), \ldots, f(x_m))$ the concatenation of their respective final activations. Let $\mathcal{D}_f$ be the set of training data to forget, and let y be some function of $\mathcal{D}_f$ that an attacker wants to reconstruct (*i.e.*, y is some piece of information regarding the samples). To keep the notation uncluttered, we write $S_{\mathcal{D}_f}(w) = S(w)$ for the scrubbing procedure to forget $\mathcal{D}_f$. We then have the following Markov chain

$$y \longleftarrow \mathcal{D}_f \longrightarrow w \longrightarrow S(w) \longrightarrow f_{S(w)}(\mathbf{x})$$

connecting all quantities. Using the Data Processing Inequality [CT12] we have the following inequalities:

$$\underbrace{I(y; f_{S(w)}(\mathbf{x}))}_{\text{Recovered information}} \leq \underbrace{I(\mathcal{D}_f; f_{S(w)}(\mathbf{x}))}_{\text{Black-box upper bound}} \leq \underbrace{I(\mathcal{D}_f; S(w))}_{\text{White box upper-bound}} \tag{3.1}$$

where $I(x; y)$ denotes the Shannon Mutual Information between random variables x and y. Bounding the last term — which is a general bound on how much information an attacker with full access to the weights could extract — is the focus of [GAS19a]. In this work, we also consider the case where the attacker can only access the final activations, and hence focus on the central term. As we will show, if the number of queries is bounded then the central term provides a sharper bound compared to the black-box case.

3.1.2 Bound for activations

The mutual information in the central term is difficult to compute,[1] but in our case has a simple upper-bound:

Lemma 2 (Computable bound on mutual information)**.** *We have the following upper bound:*

$$I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) \leq \mathbb{E}_{\mathcal{D}_f}\left[\mathrm{KL}\left(p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)\right)\right], \tag{3.2}$$

where $p(f_{S(w)}(\mathbf{x})|\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_r)$ is the distribution of activations after training on the complete dataset $\mathcal{D}_f \sqcup \mathcal{D}_r$ and scrubbing. Similarly, $p(f_{S_0(w)}(\mathbf{x})|\mathcal{D} = \mathcal{D}_r)$ is the distribution of possible activations after training only on the data to retain $\mathcal{D}_r$ and applying a function S_0 (that does not depend on $\mathcal{D}_f$) to the weights.

The lemma introduces the important notion that we can estimate how much information we erased by comparing the activations of our model with the activations of a reference model that was trained in the same setting, but without $\mathcal{D}_f$. Clearly, if the activations after scrubbing are identical to the activations of a model that has never seen $\mathcal{D}_f$, they cannot contain information about $\mathcal{D}_f$.

We now want to convert this bound in a more practical expected information gain per query. This is not yet trivial due to them stochastic dependency of w on $\mathcal{D}$: based on the random seed ϵ used to train, we may obtain very different weights for the same dataset. Reasoning in a way similar to that used to obtain the "local forgetting bound" of [GAS19a], we have:

Lemma 3. *Write a stochastic training algorithm A as $A(\mathcal{D}, \epsilon)$, where ϵ is the random seed and $A(\mathcal{D}, \epsilon)$ is a deterministic function. Then,*

$$I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) \leq \mathbb{E}_{\mathcal{D}_f,\epsilon}\Big[\mathrm{KL}\left(p(f_{S(w_{\mathcal{D}})}(\mathbf{x})) \,\|\, p(f_{S_0(w_{\mathcal{D}_r})}(\mathbf{x}))\right)\Big] \tag{3.3}$$

where we call $w_{\mathcal{D}} = A(\mathcal{D}, \epsilon)$ the deterministic result of training on the dataset $\mathcal{D}$ using random seed ϵ. The probability distribution inside the KL accounts only for the stochasticity of the scrubbing map $S(w_{\mathcal{D}})$ and the baseline $S_0(w_{\mathcal{D}_r})$.

[1] Indeed, it is even difficult to define, as the quantities at play, $\mathcal{D}_f$ and $\mathbf{x}$, are fixed values, but that problem has been addressed by [AS19, AS18] and we do not consider it here.

The expression above is general. To gain some insight, it is useful to write it for a special case where scrubbing is performed by adding Gaussian noise.

3.1.3 Close form bound for Gaussian scrubbing

We start by considering a particular class of scrubbing functions $S(w)$ in the form

$$S(w) = h(w) + n, \quad n \sim N(0, \Sigma(w, \mathcal{D})) \tag{3.4}$$

where $h(w)$ is a deterministic shift (that depends on $\mathcal{D}$ and w) and n is Gaussian noise with a given covariance (which may also depend on w and $\mathcal{D}$). We consider a baseline $S_0(w) = w + n'$ in a similar form, where $n' \sim N(0, \Sigma_0(w, \mathcal{D}_r))$.

Assuming that the covariance of the noise is relatively small, so that $f_{h(w)}(\mathbf{x})$ is approximately linear in w in a neighborhood of $h(w)$ (we drop $\mathcal{D}$ without loss of generality), we can easily derive the following approximation for the distribution of the final activations after scrubbing for a given random seed ϵ:

$$f_{S(w_\mathcal{D})}(\mathbf{x}) \sim N(f_{h(w_\mathcal{D})}(\mathbf{x}), \nabla_w f_{h(w_\mathcal{D})}(\mathbf{x}) \Sigma \nabla_w f_{h(w_\mathcal{D})}(\mathbf{x})^T) \tag{3.5}$$

where $\nabla_w f_{h(w_\mathcal{D})}(\mathbf{x})$ is the matrix whose row is the gradient of the activations with respect to the weights, for each sample in $\mathbf{x}$. Having an explicit (Gaussian) distribution for the activations, we can plug it in Lemma 3 and obtain:

Proposition 5. *For a Gaussian scrubbing procedure, we have the bounds:*

$$I(\mathcal{D}_f; S(w)) \leq \mathbb{E}_{\mathcal{D}_f, \epsilon}\big[\Delta w^T {\Sigma_0}^{-1} \Delta w + d(\Sigma, \Sigma_0)\big] \qquad \textit{(white-box)} \tag{3.6}$$

$$I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) \leq \mathbb{E}_{\mathcal{D}_f, \epsilon}\big[\Delta f^T {\Sigma'_\mathbf{x}}^{-1} \Delta f + d(\Sigma_\mathbf{x}, \Sigma'_\mathbf{x})\big] \qquad \textit{(black-box)} \tag{3.7}$$

where for a $k \times k$ matrix Σ we set $d(\Sigma, \Sigma_0) := \operatorname{tr}(\Sigma\Sigma_0^{-1}) + \log|\Sigma\Sigma_0^{-1}| - k$, and

$$\Delta w := h(w_\mathcal{D}) - w_{\mathcal{D}_r}, \quad \Delta f := f_{h(w_\mathcal{D})}(\mathbf{x}) - f_{w_{\mathcal{D}_r}}(\mathbf{x})$$
$$\Sigma_\mathbf{x} := \nabla_w f_{h(w_\mathcal{D})}(\mathbf{x})\, \Sigma\, \nabla_w f_{h(w_\mathcal{D})}(\mathbf{x})^T, \quad \Sigma'_\mathbf{x} := \nabla_w f_{w_{\mathcal{D}_r}}(\mathbf{x})\, \Sigma_0\, \nabla_w f_{w_{\mathcal{D}_r}}(\mathbf{x})^T.$$

Avoiding curse of dimensionality: Proposition 5. Comparing eq. (3.6) and eq. (3.7), we see that the bound in eq. (3.6) involves variables of the same dimension as the number of weights, while eq. (3.7) scales with the number of query points. Hence, for highly overparametrized models such as DNNs, we expect that the black-box bound in eq. (3.7) will be much smaller if the number of queries is bounded, which indeed is what we observe in the experiments (Figure 3.3).

Blessing of the null-space: The white-box bound depends on the difference Δw in weight space between the scrubbed model and the reference model $w_{\mathcal{D}_r}$, while the black-box bound depends on the distance Δf in the activations. As we mentioned in **??**, over-parametrized models such as deep networks have a large null-space of weights with similar final activations. It may hence happen that even if Δw is large, Δf may still be small (and hence the bound in eq. (3.7) tighter) as long as Δw lives in the null-space. Indeed, we often observe this to be the case in our experiments.

Adversarial queries: Finally, this should not lead us to think that whenever the activations are similar, little information can be extracted. Notice that the relevant quantity for the black box bound is $\Delta f(J_{\mathbf{x}}\Sigma_0 J_{\mathbf{x}}^T)^{-1}\Delta f^T$, which involves also the gradient $J_{\mathbf{x}} = \nabla_w f_{w_{\mathcal{D}_r}}(\mathbf{x})$. Hence, if an attacker crafts an adversarial query $\mathbf{x}$ such that its gradient $J_{\mathbf{x}}$ is small, they may be able to extract a large amount of information even if the activations are close to each other. In particular, this happens if the gradient of the samples lives in the null-space of the reference model, but not in that of the scrubbed model. In Figure 3.3 (right), we show that indeed different images can extract different amount of information.

3.2 An NTK-inspired forgetting procedure

We now introduce a new scrubbing procedure, which aims to minimize both the white-box and black-box bounds of Proposition 5. It relates to the one introduced in [GAS19a, GGH19], but it enjoys better numerical properties and can be computed without approximations (Section 3.2.1). In Section 7.2 we show that it gives better results under all commonly used metrics.

The main intuition we exploit is that most networks commonly used are fine-tuned from pre-trained networks (*e.g.*, on ImageNet), and that the weights do not move much during fine-tuning on $\mathcal{D} = \mathcal{D}_r \cup \mathcal{D}_f$ will remain close to the pre-trained values. In this regime, the network activations may be approximated as a linear function of the weights. This is inspired by a growing literature on the the so called Neural Tangent Kernel, which posits that large networks during training evolve in the same way as their linear approximation [LXS19]. Using the linearized model we can derive an analytical expression for the optimal forgetting function, which we validate empirically. However, we observe this to be misaligned with weights actually learned by SGD, and introduce a very simple "isoceles trapezium" trick to realign the solutions (see Supplementary Material).

Using the same notation as [LXS19], we linearize the final activations around the pre-trained weights θ_0 as:

$$f_t^{\text{lin}}(x) \equiv f_0(x) + \nabla_\theta f_0(x)|_{\theta=\theta_0} w_t$$

where $w_t = \theta_t - \theta_0$ and gives the following expected training dynamics for respectively the weights and the final activations:

$$\dot{w}_t = -\eta \nabla_\theta f_0(\mathcal{D})^T \nabla_{f_t^{\text{lin}}(\mathcal{D})} \mathcal{L} \tag{3.8}$$

$$\dot{f}_t^{\text{lin}}(x) = -\eta \Theta_0(x, \mathcal{D}) \nabla_{f_t^{\text{lin}}(\mathcal{D})} \mathcal{L} \tag{3.9}$$

The matrix $\Theta_0 = \nabla_\theta f(\mathcal{D}) \nabla_\theta f(\mathcal{D})^T$ of size $c|\mathcal{D}| \times c|\mathcal{D}|$, where c the number of classes, is called the Neural Tangent Kernel (NTK) matrix [LXS19, JGH18]. Using this dynamics, we can approximate in closed form the final training point when training with $\mathcal{D}$ and $\mathcal{D}_r$, and compute the optimal "one-shot forgetting" vector to jump from the weights $w_{\mathcal{D}}$ that have been obtained by training on $\mathcal{D}$ to the weights $w_{\mathcal{D}_r}$ that would have been obtained training on $\mathcal{D}_r$ alone:

Proposition 6. *Assuming an L_2 regression loss,*[2] *the optimal scrubbing procedure under the NTK approximation is given by*

$$\boxed{h_{NTK}(w) = w + P \nabla f_0(\mathcal{D}_f)^T M V} \tag{3.10}$$

[2]*This assumption is to keep the expression simple, in the Supplementary Material we show the corresponding expression for a softmax classification loss.*

where $\nabla f_0(\mathcal{D}_f)^T$ *is the matrix whose columns are the gradients of the sample to forget, computed at* θ_0 *and* $w = A(\mathcal{D}_r \cup \mathcal{D}_f)$. $P = I - \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \nabla f_0(\mathcal{D}_r)$ *is a projection matrix, that projects the gradients of the samples to forget* $\nabla f_0(\mathcal{D}_f)$ *onto the orthogonal space to the space spanned by the gradients of all samples to retain. The terms* $M = \left[\Theta_{ff} - \Theta_{rf}^T \Theta_{rr}^{-1} \Theta_{rf}\right]^{-1}$ *and* $V = [(Y_f - f_0(\mathcal{D}_f)) + \Theta_{rf}^T \Theta_{rr}^{-1}(Y_r - f_0(\mathcal{D}_r))]$ *re-weight each direction before summing them together.*

Given this result, our proposed scrubbing procedure is:

$$\boxed{S_{\text{NTK}}(w) = h_{\text{NTK}}(w) + n} \tag{3.11}$$

where $h_{\text{NTK}}(w)$ is as in eq. (3.10), and we use noise $n \sim N(0, \lambda F^{-1})$ where $F = F(w)$ is the Fisher Information Matrix computed at $h_{\text{NTK}}(w)$ using $\mathcal{D}_r$. The noise model is as in [GAS19a], and is designed to increase robustness to mistakes due to the linear approximation.

3.2.1 Relation between NTK and Fisher forgetting

In [GAS19a] and [GGH19], a different forgetting approach is suggested based on either the Hessian or the Fisher Matrix at the final point: assuming that the solutions $w(\mathcal{D}_r)$ and $w(\mathcal{D})$ of training with and without the data to forget are close and that they are both minima of their respective loss, one may compute the shift to to jump from one minimum to the other of a slightly perturbed loss landscape. The resulting "scrubbing shift" $w \mapsto h(w)$ relates to the newton update:

$$h(w) = w - H(\mathcal{D}_r)^{-1} \nabla_w L_{\mathcal{D}_r}(w). \tag{3.12}$$

In the case of an L_2 loss, and using the NTK model, the Hessian is given by $H(w) = \nabla f_0(\mathcal{D}_r)^T \nabla f_0(\mathcal{D}_r)$ which in this case also coincides with the Fisher Matrix [Mar14]. To see how this relates to the NTK matrix, consider deterimining the convergence point of the linearized NTK model, that for an L_2 regression is given by $w^* = w_0 + \nabla f_0(\mathcal{D}_r)^+ \mathcal{Y}$, where $\nabla f_0(\mathcal{D}_r)^+$ denotes the matrix pseudo-inverse, and $\mathcal{Y}$ denotes the regression targets. If $\nabla f_0(\mathcal{D}_r)$ is a tall matrix (more samples in the dataset than parameters in the network), then the pseudo-inverse is $\nabla f_0(\mathcal{D}_r)^+ = H^{-1} \nabla f_0(\mathcal{D}_r)^T$, recovering

the scrubbing procedure considered by [GAS19a, GGH19]. However, if the matrix is wide (more parameters than samples in the network, as is often the case in Deep Learning), the Hessian is not invertible, and the pseudo-inverse is instead given by $\nabla f_0(\mathcal{D}_r)^+ = \nabla f_0(\mathcal{D}_r)^T \Theta^{-1}$, leading to our proposed procedure. In general, when the model is over-parametrized there is a large null-space of weights that do not change the final activations or the loss. The degenerate Hessian is not informative of where the network will converge in this null-space, while the NTK matrix gives the exact point.

3.3 Experiments

3.3.1 Datasets

We report experiments on smaller versions of CIFAR-10 [Kri09] and Lacuna-10 [GAS19a], a dataset derived from the VGG-Faces [CSX18] dataset. We obtain the small datasets using the following procedure: we randomly sample 500 images (100 images from each of the first 5 classes) from the training/test set of CIFAR-10 and Lacuna-10 to obtain the small-training/test respectively. We also sample 125 images from the training set (5 classes $\times$ 25 images) to get the validation set. So, in short, we have 500 (5×100) examples for training and testing repectively, and 125 (5×25) examples for validation. On both the datasets we choose to forget 25 random samples (5% of the dataset). Without loss of generality we choose to forget samples from class 0.

3.3.2 Models and Training

We use All-CNN [SDB14] (to which we add batch-normalization before non-linearity) and ResNet-18 [HZR16] as the deep neural networks for our experiments. We pre-train the models on CIFAR-100/Lacuna-100 and then fine-tune them (all the weights) on CIFAR-10/Lacuna-10. We pre-train using SGD for 30 epochs with a learning rate of 0.1, momentum 0.9 and weight decay 0.0005. Pre-training helps in improving the stability of SGD while fine-tuning. For fine-tuning we use a

learning rate of 0.01 and weight decay 0.1. While applying weight decay we bias the weights with respect to the initialization. During training we always use a batch-size of 128 and fine-tune the models till zero training error. Also, during fine-tuning we do not update the running mean and variance of the batch-normalization parameters to simplify the training dynamics. We perform each experiment 3 times and report the mean and standard deviation.

3.3.3 Baselines

We consider three baselines for comparison: (i) **Fine-tune**, we fine-tune $w(\mathcal{D})$ on $\mathcal{D}_r$ (similar to catastrophic forgetting) and (ii) **Fisher forgetting** [GAS19a], we scrubs the weights by adding Gaussian noise using the inverse of the Fisher Information Matrix as covariance matrix, (iii) **Original** corresponds to the original model trained on the complete dataset ($w(\mathcal{D})$) without any forgetting. We compare those, and our proposal, with optimal reference the model $w(\mathcal{D}_r)$ trained from scratch on the retain set, that is, without using $\mathcal{D}_f$ in the first place. Values read from this reference model corresponds to the green region in Figure 3.2 and represent the gold standard for forgetting: In those plots, an optimal algorithm should lie inside the green area.

3.3.4 Readout Functions

We use multiple readout functions similar to [GAS19a]: (i) **Error on residual** (should be small), (ii) **Error on cohort to forget** (should be similar to the model re-trained from scratch on $\mathcal{D}_r$), (iii) **Error on test set** (should be small), (iv) **Re-learn time**, measures how quickly a scrubbed model learns the cohort to forget, when fine-tuned on the complete data. Re-learn time (measured in epochs) is the first epoch when the loss during fine-tuning (the scrubbed model) falls below a certain threshold (loss of the original model (model trained on $\mathcal{D}$) on $\mathcal{D}_f$). (v) **Blackbox membership inference attack**: We construct a simple yet effective blackbox membership inference attack using the entropy of the output probabilities of the scrubbed model. Similar to the method in [SS15], we formulate the attack as a binary classification problem (class 1 - belongs to training set and class

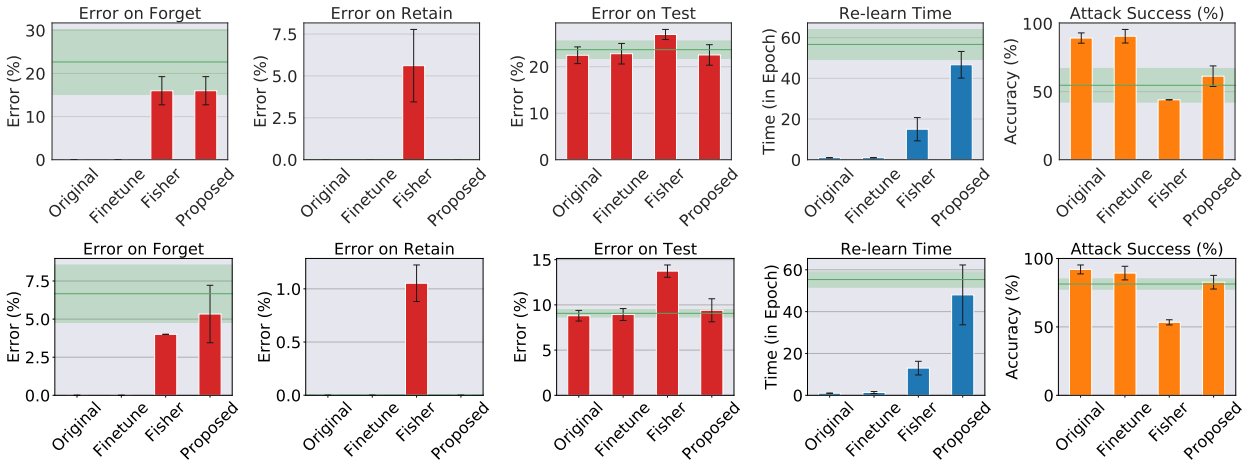

Figure 3.2: Comparison of different models baselines (original, finetune) and forgetting methods (Fisher [GAS19a] and our NTK proposed method), using several readout functions ((**Top**) CIFAR and (**Bottom**) Lacuna). We benchmark them against a model that has never seen the data (the gold reference for forgetting): values (mean and standard deviation) measured from this models corresponds to the green region. Optimal scrubbing procedure should lie in the green region, or they will leak information about $\mathcal{D}_f$. We compute three read-out functions: **(a)** Error on forget set $\mathcal{D}_f$, **(b)** Error on retain set $\mathcal{D}_r$, **(c)** Error on test set $\mathcal{D}_{\text{test}}$. **(d)** Black-box membership inference attack: We construct a simple yet effective membership attack using the entropy of the output probabilities. We measures how often the attack model (using the activations of the scrubbed network) classify a sample belonging $\mathcal{D}_f$ as a training sample rather than being fooled by the scrubbing. **(e)** Re-learn time for different scrubbing methods: How fast a scrubbed model learns the forgotten cohort when fine-tuned on the complete dataset. We measure the re-learn time as the first epoch when the loss on $\mathcal{D}_f$ goes below a certain threshold.

0 - belongs to test set). For training the attack model (Support Vector Classifier with Radial Basis Function Kernel) we use the retain set ($\mathcal{D}_r$) as class 1 and the test set as class 0. We test the success of the attack on the cohort to forget ($\mathcal{D}_f$). Ideally, the attack accuracy for an optimally scrubbed model should be the same as a model re-trained from scratch on $\mathcal{D}_f$, having a higher value implies incorrect (or no) scrubbing, while a lower value may result in Streisand Effect, (vi) **Remaining information in the weights** [GAS19a] and (vii) **Remaining information in the activations**: We compute an upper bound on the information the activations contain about the cohort to forget ($\mathcal{D}_f$) (after scrubbing) when queried with images from different subsets of the data ($\mathcal{D}_r$, $\mathcal{D}_r$).

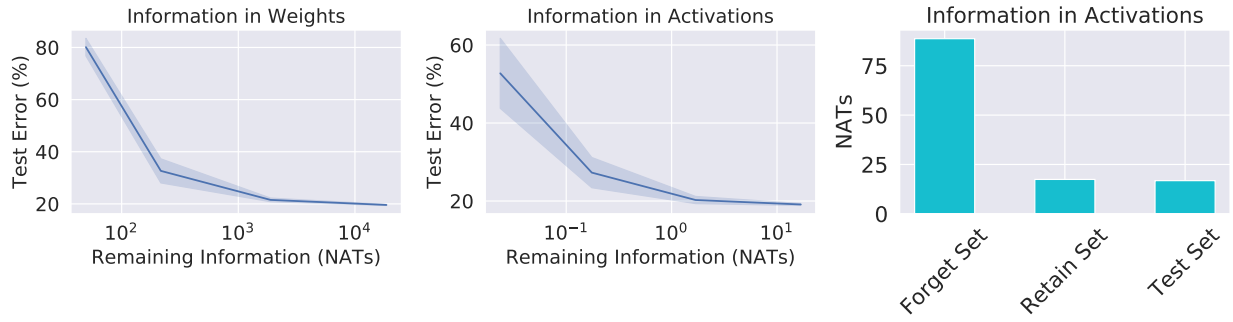

Figure 3.3: **Error-forgetting trade-off** Using the proposed scrubbing procedure, by changing the variance of the noise, we can reduce the remaining information in the weights **(white-box bound, left)** and activations **(black-box bound, center)**. However, it comes at the cost of increasing the test error. Notice that the bound on activation is much sharper than the bound on error at the same accuracy. **(Right) Different samples leak different information.** An attacker querying samples from $\mathcal{D}_f$ can gain much more information than querying unrelated images. This suggest that adversarial samples may be created to leak even more information.

3.3.5 Results

Error readouts: In Figure 3.2 (a-c), we compare error based readout functions for different forgetting methods. Our proposed method outperforms Fisher forgetting which incurs high error on the retain ($\mathcal{D}_r$) and test ($\mathcal{D}_{\text{Test}}$) set to attain the same level of forgetting. This is due the large distance between $w(\mathcal{D})$ and $w(\mathcal{D}_r)$ in weight space, which forces it to add too much noise to erase information about $\mathcal{D}_f$, and ends up also erasing information about the retain set $\mathcal{D}_r$ (high error on $\mathcal{D}_r$ in Figure 3.2). Instead, our proposed method first moves $w(\mathcal{D})$ in the direction of $w(\mathcal{D}_r)$, thus minimizing the amount of noise to be added (Figure 3.1). Fine-tuning the model on $\mathcal{D}_r$ (catastrophic forgetting) does not actually remove information from the weights and performs poorly on all the readout functions.

Relearn time: In Figure 3.2 (d), we compare the re-learn time for different methods. Re-learn time can be considered as a proxy for the information remaining in the weights about the cohort to forget ($\mathcal{D}_f$) after scrubbing. We observe that the proposed method outperforms all the baselines which is in accordance with the previous observations (in Figure 3.2(a-c)).

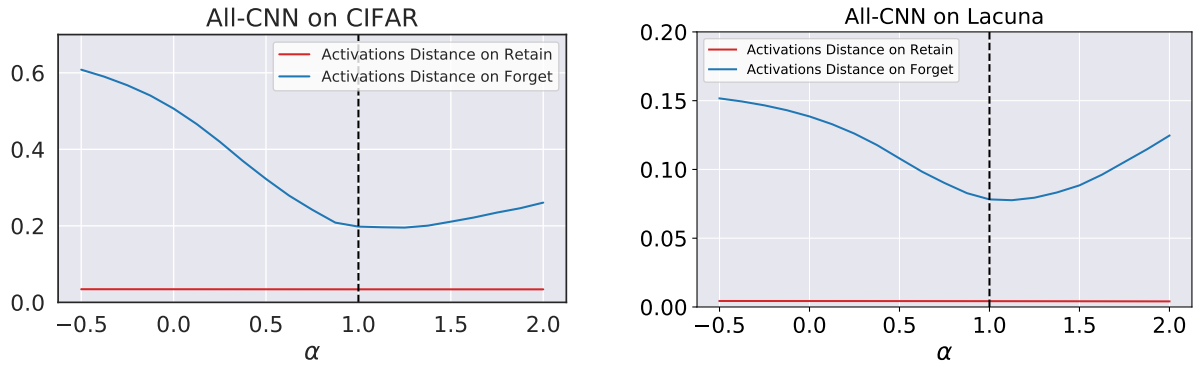

Figure 3.4: **Scrubbing brings activations closer to the target.** We plot the L_1 norm of the difference between the final activations (post-softmax) of the target model trained only on $\mathcal{D}_r$, and models sampled along the line joining the original model $w(D)$ ($\alpha = 0$) and the proposed scrubbed model ($\alpha = 1$). The distance between the activations decreases as we move along the scrubbing direction. The L_1 distance is already low on the retain set ($\mathcal{D}_r$) (red) as it corresponds to the data common to $w(\mathcal{D})$ and $w(\mathcal{D}_r)$. However, the two models differ on the forget set ($\mathcal{D}_f$) (blue) and we observe that the L_1 distance decreases as move along the proposed scrubbing direction.

Membership attacks: In Figure 3.2 (e), we compare the robustness of different scrubbed models against blackbox membership inference attacks (attack aims to identify if the scrubbed model was ever trained on $\mathcal{D}_f$). This can be considered as a proxy for the remaining information (about $\mathcal{D}_f$) in the activations. We observe that attack accuracy for the proposed method lies in the optimal region (green), while Fine-tune does not forget ($\mathcal{D}_f$), Fisher forgetting may result in Streisand effect which is undesireable.

Closeness of activations: In Figure 3.4, we show that the proposed scrubbing method brings the activations of the scrubbed model closer to retrain model (model retrained from scratch on $\mathcal{D}_r$). We measure the closeness by computing: $\mathbb{E}_{x\sim\mathcal{D}_f/\mathcal{D}_r}\left[\|f_{\text{scrubbed}}(x) - f_{\text{retrain}}(x)\|_1\right]$ along the scrubbing direction, where $f_{\text{scrubbed}}(x)$, $f_{\text{retrain}}(x)$ are the activations (post soft-max) of the proposed scrubbed and retrain model respectively. The distance between the activations on the cohort to forget ($\mathcal{D}_f$) decreases as we move along the scrubbing direction and achieves a minimum value at the scrubbed model, while it almost remains constant on the retain set. Thus, the activations of the scrubbed model shows desirable behaviour on both $\mathcal{D}_r$ and $\mathcal{D}_f$.

Error-forgetting trade-off In Figure 3.3, we plot the trade-off between the test error and the remaining information in the weights and activations respectively by changing the scale of the variance of Fisher noise. We can reduce the remaining information but this comes at the cost of increasing the test error. We observe that the black-box bound on the information accessible with one query is much tighter than the white box bound at the same accuracy (compare left and center plot x-axes). Finally, in (right), we show that query samples belonging to the cohort to be forgotten ($\mathcal{D}_f$) leaks more information about the $\mathcal{D}_f$ rather than the retain/test set, proving that indeed carefully selected samples are more informative to an attacker than random samples.

3.4 Discussion

Recent work [AS19, GAS19a, GGH19] has started providing insights on both the amount of information that can be extracted from the weights about a particular cohort of the data used for training, as well as give constructive algorithms to "selectively forget." Note that forgetting alone could be trivially obtained by replacing the model with a random vector generator, obviously to the detriment of performance, or by retraining the model from scratch, to the detriment of (training time) complexity. In some cases, the data to be retained may no longer be available, so the latter may not even be an option.

We introduce a scrubbing procedure based on the NTK linearization which is designed to minimize both a white-box bound (which assumes the attacker has the weights), and a newly introduced black-box bound. The latter is a bound on the information that can be obtained about a cohort using only the observed input-output behavior of the network. This is relevant when the attacker performs a bounded number of queries. If the attacker is allowed infinitely many observations, the matter of whether the black-box and white-box attack are equivalent remains open: Can an attacker always craft sufficiently exciting inputs so that the exact values of all the weights can be inferred? An answer would be akin to a generalized "Kalman Decomposition" for deep networks. This alone is an interesting open problem, as it has been pointed out recently that good

"clone models" can be created by copying the response of a black-box model on a relatively small set of exciting inputs, at least in the restricted cases where every model is fine-tuned from a common pre-trained models [KTP19].

While our bounds are tighter that others proposed in the literature, our model has limitations. Chiefly, computational complexity. The main source of computational complexity is computing and storing the matrix P in eq. (3.10), which naively would require $O(c^2|\mathcal{D}_r|^2)$ memory and $O(|w| \cdot c^2|\mathcal{D}_r|^2)$ time. For this reason, we conduct experiments on a relatively small scale, sufficient to validate the theoretical results, but our method is not yet scalable to production level. However, we notice that P is the projection matrix on the orthogonal of the subspace spanned by the training samples, and there is a long history [BNS78] of numerical methods to incrementally compute such operator incrementally and without storing it fully in memory. We leave these promising option to scale the method as subject of future investigation.

[**?**] has recently shown that linearized models with gradients as features perform comparable to deep networks. This means that we can train a linear model for tasks like image classification, without significantly compromising the accuracy compared to a DNN. This is an interesting avenue for future work, as using a linearized DNN will mitigate the problems arising due to the poor stability of the non-convex loss landscape of DNNs, thus allowing us to enjoy the convex nature of the training optimization enabling better forgetting.

Federated learning is a modern machine learning technique which allows different clients to train their models locally, and then communicate their weights through a central server. Lets say that one of the client decides to withdraw itself from the federated learning setup, and also wishes that the effect of its data be removed from all the other weights (both clients and central server). This motivates another future direction for designing forgetting/machine unlearning procedures which will scrub all the remaining weights clean, every time a client withdraws itself. Of course, differential privacy can be enforced when the clients communicate with the central server, however, this will come at the cost of the model accuracy, while a forgetting procedure will ensure high accuracy without compromising accuracy.

[TSC18] performed an empirical study of example forgetting during training (catastrophic forgetting) and observed that certain examples can be forgotten easily, while the unforgettable examples generalize across DNN architectures. A similar study for selective forgetting is also an interesting direction for future work.

Even though we propose a method for scrubbing deep networks using linearization from NTK, removing information directly from the weights of highly non-linear deep networks without any approximation still remains an unsolved problem at large.

3.5 Appendix

We present the following:

- Section 3.5.1: Provide implementation details for our scrubbing procedure;
- Section 3.5.2: Show further experiments on more datasets (CIFAR-10, Lacuna, TinyImagenet) and models (AllCNN, ResNet).
- Section 3.5.3: Provide proofs for all propositions and equations in the paper;

3.5.1 Experimental Details

We train our models with SGD (learning rate $\eta = 0.01$ and momentum $m = 0.9$) using weight decay ($\lambda = 0.1$). All models are trained to convergence (we stop the training 5 epochs after the model achieves zero training error).

Pre-training and weight-decay. In all cases, we use pre-trained models (we pre-train the models on CIFAR-100/Lacuna-100/TinyImageNet (first 150 classes) respectively), and denote by w_0 the pre-trained weight configuration. One important point is that we change the weight decay regularization term from the standard $\|w\|_2^2$ (which pulls the weights toward zero) to $\|w - w_0\|_2^2$ (which pulls the weights toward the initialization). This serves two purposes, (i) It ensures that the weights remain

close to the initialization (in our case, a pre-trained network). This further helps the weight during training to remain in the neighborhood of the initialization where the the linear approximation (NTK) is good; (ii) With this change, the training dynamics of the weights/activations of a linearized network only depend on the relative change in the weights from its initial value (see [LXS19, Section 2.2] for more details).

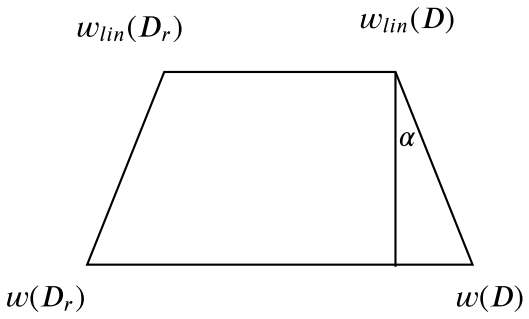

Figure 3.5: Isosceles Trapezium Trick: $\|w(\mathcal{D}_r) - w(\mathcal{D})\| = \|w_{\text{lin}}(\mathcal{D}_r) - w_{\text{lin}}(\mathcal{D})\| + 2\sin\alpha\|w_{lin}(\mathcal{D}) - w(\mathcal{D})\|$. This allows us to match outputs of the linear dynamic model with the real output, without having to match the effective learning rate of the two, and while being more robust to wrong estimation of the curvature by the linearized model.

NTK matrix, weight decay and cross-entropy. For clarity, in Section 4 and Proposition 2 we only considered a unregularized MSE regression problem. To apply the theory to the more practical case of a classification cross-entropy loss with weight-decay regularization, we need the following changes.

First, using weight decay the NTK matrix becomes $\Theta = \nabla f_0(\mathcal{D})\nabla f_0(\mathcal{D})^T + \lambda I$, where λ is the weight decay coefficient. Second, when using the cross-entropy loss, the gradients $\nabla_{f_t^{\text{lin}}(\mathcal{D})} L$ of the loss function can be approximated as $\nabla_{f_w(x)} L \approx \nabla_{f_{w_0(x)}} L + H_{f_{w_0}(x)}(f_w(x) - f_{w_0}(x))$, where $H_{f_{w_0}(x)}$ is the Hessian of the loss with respect to the output activations. With these two together, we obtain $\Theta = H_{f_{w_0}(x)}\nabla f_0(\mathcal{D})\nabla f_0(\mathcal{D})^T + \lambda I$ as the NTK matrix to use in our setting.

We did however notice that replacing $H_{f_{w_0}(x)}$ with the identity matrix I works better in practice. This may be due to $H_{f_{w_0}(x)}$ estimating the wrong curvature when the softmax saturates. Also

we found that — while in principle identical as long as the network remains in the linear regime — linearizing around $w(\mathcal{D})$ provides a better estimate of the scrubbing direction compared to linearizing around w_0.

Trapezium trick. We observe that the linearized dynamics of in eq. (8) and eq. (9) correctly approximate the training direction but they usually undershoot and give a smaller norm solution than SGD. This may be due to difficulty in matching the learning rate of continuous gradient descent and discrete SGD. To overstep these issues in a robust way, we use the following simple "trapezium trick" (Figure 3.5) to renormalize the scrubbing vector obtained with the linear dynamics: Instead of trying to predicting the unknown $w(\mathcal{D}_r)$ directly using the scrubbing vector suggested by eq. (3.10), we compute the two final points of the linearized dynamics $w_{\text{lin}}(\mathcal{D})$ and $w_{\text{lin}}(\mathcal{D}_r)$, and approximate $w(\mathcal{D}_r)$ by constructing the isosceles trapezium in Figure 3.5. Effectively, this rescales the ideal linearized forgetting direction $w_{\text{lin}}(\mathcal{D}_r) - w_{\text{lin}}(\mathcal{D})$ to correct for the undershooting.

3.5.2 Additional Experiments

In Figure 3.6 we use a PCA projection to show the geometry of the loss landscape after convergence and the training paths obtained by training the weights on the full dataset $\mathcal{D}$ or only on $\mathcal{D}_r$. We observe that the loss landscape around the pretraining point is smooth and almost convex. Moreover, the two training paths remain close to each other. This supports our choice of using a simple linearized approximation to compute the shift that jumps from one path to the other.

In Figure 3.7 and Figure 3.8 we show additional experiments on several architectures (ResNet-18, All-CNN) and datasets (Lacuna, CIFAR-10, TinyImageNet). In all cases we observe a similar qualitative behavior to the one discussed in the paper.

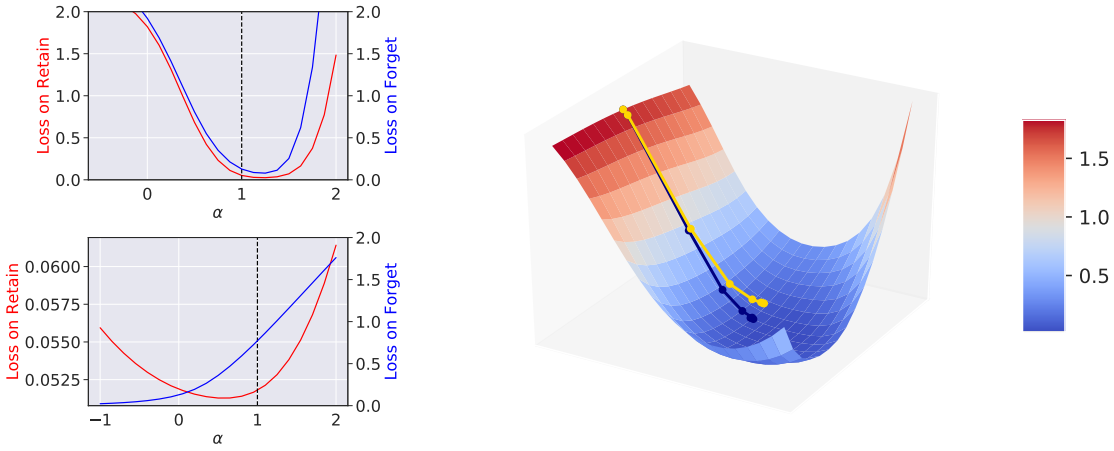

Figure 3.6: **(Right) The loss landscape and training dynamics after pretraining are smooth and regular.** This justifies our linearization approach to study the dynamics. The black and yellow lines are the training paths on $\mathcal{D}$ and $\mathcal{D}_r$ respectively. Notice that they remain close. **(Upper left) Loss along the line** joining the model at initialization ($\alpha = 0$) and the model after training on $\mathcal{D}$ ($\alpha = 1$) (the black path). **(Lower left)** Loss along the line joining the end point of the two paths ($\alpha = 0$ and 1 respectively), which is the ideal scrubbing direction.

3.5.3 Proofs

Markov chain in Section 3.1.1. We consider the retain set $\mathcal{D}_r$ (not shown) as an observed random variable, while the cohort to forget $\mathcal{D}_f$ is an hidden variable sampled randomly from the data distribution. The directed edge $\mathcal{D}_f \to w$ in the Markov chain derives from the fact that we first sample $\mathcal{D}_f$, to obtain the full complete training set $\mathcal{D} = \mathcal{D}_r \sqcup \mathcal{D}_f$, and then train the network on $\mathcal{D}$ to obtain the weights w.

Proof of Lemma 1. We have the following upper-bound for $I(\mathcal{D}_f; f_{S(w)}(\mathbf{x}))$:

$$
\begin{aligned}
I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) &= I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) \\
&= \mathbb{E}_{\mathcal{D}_f}\left[\,\mathrm{KL}\left(p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f_{S(w)}(\mathbf{x})|\mathcal{D}_r)\right)\right] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f\cup\mathcal{D}_r)}\left[\log \frac{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_r)}\right] \\
&= \mathbb{E}_{\mathcal{D}_f}\mathbb{E}_{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f\cup\mathcal{D}_r)}\left[\log \frac{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r)}{p(f_{S(w)}(\mathbf{x})|\mathcal{D}_r)}\right.
\end{aligned}
$$

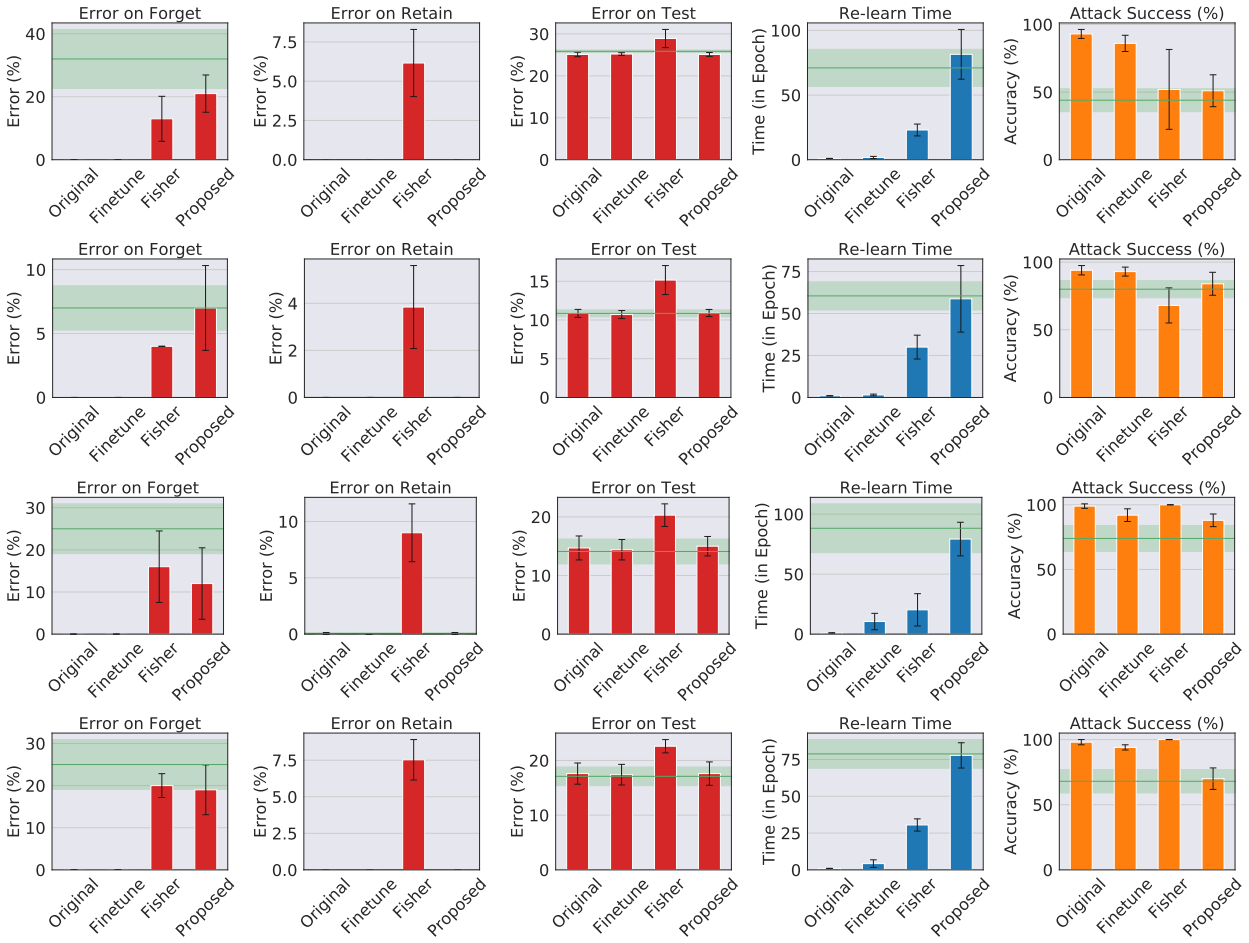

Figure 3.7: Same experiment as Figure 3.2 for different architectures and datasets. **(Row 1)**: ResNet-18 on CIFAR, **(Row 2)**: ResNet-18 on Lacuna, **(Row 3)**: All-CNN on TinyImageNet and **(Row 4)**: ResNet-18 on TinyImageNet. In all the experiments we observe that for different readout functions the proposed method lies in the green (target) region.

$$
\begin{aligned}
&\qquad\qquad + \log \frac{p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)}{p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)}\Big] \\
&= \mathbb{E}_{\mathcal{D}_f}\Big[\mathrm{KL}\left(p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)\right)\Big] \\
&\qquad\qquad - \mathrm{KL}\left(p(f_{S(w)}(\mathbf{x})|\mathcal{D}_r) \,\|\, p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)\right) \\
&\leq \mathbb{E}_{\mathcal{D}_f}\Big[\mathrm{KL}\left(p(f_{S(w)}(\mathbf{x})|\mathcal{D}_f \cup \mathcal{D}_r) \,\|\, p(f_{S_0(w)}(\mathbf{x})|\mathcal{D}_r)\right)\Big]
\end{aligned}
$$

where the last inequality follows from the fact that KL-divergence is always non-negative.

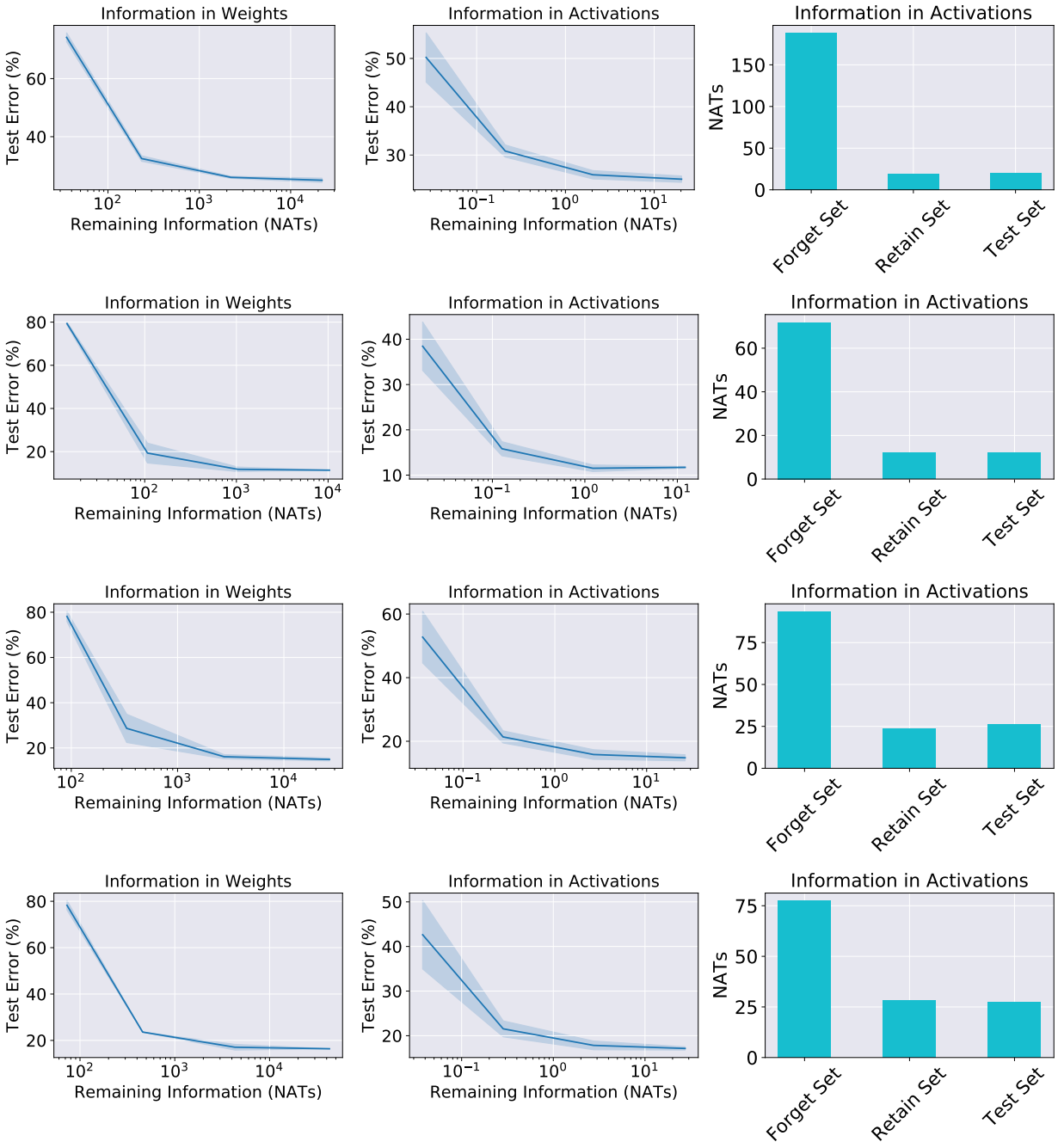

Figure 3.8: Same experiment as Figure 3.3 for different architectures and datasets. **(Row-1)**: ResNet-18 on CIFAR, **(Row-2)**: ResNet-18 on Lacuna, **(Row-3)**: All-CNN on TinyImageNet, **(Row-4)**: ResNet-18 on TinyImageNet. We observe consistent behaviour across different architectures and datasets.

Proof of Lemma 2. We have the inequalities:

$$\begin{aligned} I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) &\leq \mathbb{E}_{\mathcal{D}_f}\left[\,\mathrm{KL}\left(p(f_{S(w_{\mathcal{D}})}(\mathbf{x}))\,\|\,p(f_{S_0(w_{\mathcal{D}_r})}(\mathbf{x}))\right)\right] \\ &\leq \mathbb{E}_{\mathcal{D}_f,\epsilon}\Big[\,\mathrm{KL}\left(p(f_{S(w_{\mathcal{D}})}(\mathbf{x}))\,\|\,p(f_{S_0(w_{\mathcal{D}_r})}(\mathbf{x}))\right)\Big] \end{aligned}$$

where the first inequality comes from Lemma 1, and the second inequality is from [GAS19a, Proposition 2].

Proof of eq. (5). The activations of a scrubbed network (using the Gaussian scrubbing procedure in eq. (3.4)) for a given sample $\mathbf{x}$ are given by $f_{h(w)+n}(\mathbf{x})$, where $n \sim N(0, \Sigma)$. By linearizing the activations (using NTK formalism) around $h(w)$, we obtain the distribution of the scrubbed activations:

$$f_{h(w)+n}(\mathbf{x}) \sim N(f_{h(w)}(\mathbf{x}), \nabla_w f_{h(w)}(\mathbf{x})\Sigma\nabla_w f_{h(w)}(\mathbf{x})^T)$$

For the original model we compute this at $w = w_{\mathcal{D}}$ and take h to be the the NTK scrubbing shift in eq. (3.10). The baseline model does not use any shift, so $h(w) = w$, and is computed at $w = w_{\mathcal{D}_r}$.

Proof of Proposition 1, eq. (3.6).

As in [GAS19a, Example 2].

Proof of Proposition 1, eq. (3.7).

Using Equation (3.5) we write the distribution of the activations of a scrubbed network:

$$f_{S(w_{\mathcal{D}})}(\mathbf{x}) \sim N(f_{h(w_{\mathcal{D}})}(\mathbf{x}), J\Sigma J^T)$$

where $J = \nabla_w f_{h(w_{\mathcal{D}})}(\mathbf{x})$. We can similarly write the activations for baseline as:

$$f_{S_0(w_{\mathcal{D}_r})}(\mathbf{x}) \sim N(f_{w_{\mathcal{D}_r}}(\mathbf{x}), J'\Sigma_0 J'^T)$$

where $J' = \nabla_w f_{w_{\mathcal{D}_r}}(\mathbf{x})$ Using the two distributions, we rewrite the bound in Lemma 2 as:

$$\begin{aligned} I(\mathcal{D}_f; f_{S(w)}(\mathbf{x})) &\leq \mathbb{E}_{\mathcal{D}_f,\epsilon}\big[\,\mathrm{KL}\left(N(f_{h(w_{\mathcal{D}})}(\mathbf{x}), J\Sigma J^T) \,\|\, N(f_{w_{\mathcal{D}_r}}(\mathbf{x}), J'\Sigma_0 J'^T\right)\big] \\ &= \mathbb{E}_{\mathcal{D}_f,\epsilon}\big[\Delta f^T \Sigma_{\mathbf{x}}'^{-1} \Delta f + \mathrm{tr}(\Sigma_{\mathbf{x}}\Sigma_{\mathbf{x}}'^{-1}) - \log|\Sigma_{\mathbf{x}}\Sigma_{\mathbf{x}}'^{-1}| - n\big] \end{aligned}$$

where $\Delta f = f_{h(w_{\mathcal{D}})}(\mathbf{x}) - f_{w_{\mathcal{D}_r}}(\mathbf{x})$, $\Sigma_{\mathbf{x}} = J\Sigma J^T$, $\Sigma'_{\mathbf{x}} = J'\Sigma_0 J'^T$, and we used the closed form expression for the KL divergence of two normal distributions.

Proof of Proposition 2.

Let $\mathcal{D} = \mathcal{D}_r \cup \mathcal{D}_f$ be the complete training set. To keep the notation simpler, we assume that the loss is an mean-square-error regression loss (we discuss classification using cross-entropy in Appendix B).

Let w_0 be the weights obtained after pre-training on $\mathcal{D}_{\text{pre-train}}$. Taking inspiration from the NTK analysis [JGH18, LXS19] we approximate the activations $f_w(x)$ for a test datum x after fine-tuning on $\mathcal{D}$ using the liner approximation $f_w^{\text{lin}}(x) = f_{w_0}(x) + \nabla_w f_{w_0}(x)(w - w_0)$. To keep the notation uncluttered, we write $f_0(x)$ instead of $f_{w_0}(x)$.

The training dynamics under the linear approximation (assuming continuous gradient descent) are then given by eq. (8) and (9), and will converge at the final solution (see [LXS19] for more details):

$$w_{\text{lin}}(\mathcal{D}) = \nabla f_0(\mathcal{D})^T \Theta^{-1}(f_0(\mathcal{D}) - Y) + w_0.$$

Here $\nabla f_0(\mathcal{D}) \in \mathbb{R}^{Nc \times p}$ is the gradient of the output with respect to the parameters (at initialization) for all the samples in $\mathcal{D}$ stacked along the rows to form a matrix (p is the number of parameters in the model, $N = |\mathcal{D}|$ and c is the number of classes), $\Theta = \nabla f_0(\mathcal{D}) \nabla f_0(\mathcal{D})^T$ ($\in \mathbb{R}^{Nc \times Nc}$) is the NTK matrix. Similarly, Y is the matrix formed by stacking all ground-truth labels one below the other and $f_0(\mathcal{D}) \in \mathbb{R}^{(Nc \times 1)}$ are the stacked outputs of the DNN at initialization on $\mathcal{D}$.

Similarly the baseline solution (training only on the data to retain) is:

$$w_{\text{lin}}(\mathcal{D}_r) = \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1}(f_0(\mathcal{D}_r) - Y_r) + w_0,$$

where $\Theta_{rr} = \nabla f_0(\mathcal{D}_r)^T \nabla f_0(\mathcal{D}_r)$. The optimal scrubbing vector (at least for the linearized model) would then be $\delta w := w_{\text{lin}}(\mathcal{D}_r) - w_{\text{lin}}(\mathcal{D})$: adding δw to the weights obtained by training on $\mathcal{D}$ makes us forget the extra examples ($\mathcal{D}_f$), so that we obtain weights equivalent to training on $\mathcal{D}_r$ alone. We now derive the simplified expression in eq. (3.10) for the optimal scrubbing vector δw. We start by rewriting $w_{\text{lin}}(\mathcal{D})$ using block matrixes:

$$\begin{aligned} w_{\text{lin}}(\mathcal{D}) &= \nabla f_0(\mathcal{D})^T \Theta^{-1}(f_0(\mathcal{D}) - Y) + w_0 \\ &= \begin{bmatrix} \nabla f_0(\mathcal{D}_r)^T \nabla f_0(\mathcal{D}_f)^T \end{bmatrix} \begin{bmatrix} \Theta_{rr} & \Theta_{rf} \\ \Theta_{rf}^T & \Theta_{ff} \end{bmatrix}^{-1} \begin{bmatrix} f_0(\mathcal{D}_r) - Y_r \\ f_0(\mathcal{D}_f) - Y_f \end{bmatrix} + w_0 \end{aligned}$$

We can expand the inverse of the NTK matrix using the following equations:

$$\begin{bmatrix} \Theta_{rr} & \Theta_{rf} \\ \Theta_{rf}^T & \Theta_{ff} \end{bmatrix}^{-1} = \begin{bmatrix} \begin{bmatrix} \Theta_{rr} - \Theta_{rf}\Theta_{ff}^{-1}\Theta_{rf}^T \end{bmatrix}^{-1} & -\Theta_{rr}^{-1}\Theta_{rf}M \\ -M\Theta_{rf}^T\Theta_{rr}^{-1} & M \end{bmatrix}$$

Where $M = \begin{bmatrix} \Theta_{ff} - \Theta_{rf}^T\Theta_{rr}^{-1}\Theta_{rf} \end{bmatrix}^{-1}$. Using Woodbury Matrix Identity:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

We obtain:

$$\begin{bmatrix} \Theta_{rr} - \Theta_{rf}\Theta_{ff}^{-1}\Theta_{rf}^T \end{bmatrix}^{-1} = \Theta_{rr}^{-1} + \Theta_{rr}^{-1}\Theta_{rf}M\Theta_{rf}^T\Theta_{rr}^{-1}$$

Thus,

$$\begin{bmatrix} \Theta_{rr} & \Theta_{rf} \\ \Theta_{rf}^T & \Theta_{ff} \end{bmatrix}^{-1} = \begin{bmatrix} \Theta_{rr}^{-1} + \Theta_{rr}^{-1}\Theta_{rf}M\Theta_{rf}^T\Theta_{rr}^{-1} & -\Theta_{rr}^{-1}\Theta_{rf}M \\ -M\Theta_{rf}^T\Theta_{rr}^{-1} & M \end{bmatrix}$$

Using the above relation we get

$$\begin{aligned} w_{\text{lin}}(\mathcal{D}) = \nabla f_0(\mathcal{D}_r)^T &\Big(\Theta_{rr}^{-1} + \Theta_{rr}^{-1}\Theta_{rf}M\Theta_{rf}^T\Theta_{rr}^{-1}\Big)(f_0(\mathcal{D}_r) - Y_r) \\ &- \nabla f_0(\mathcal{D}_f)^T M\Theta_{rf}^T\Theta_{rr}^{-1}(f_0(\mathcal{D}_r) - Y_r) \\ &- \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1}\Theta_{rf}M(f_0(\mathcal{D}_f) - Y_f) \\ &+ \nabla f_0(\mathcal{D}_f)^T M(f_0(\mathcal{D}_f) - Y_f) + w_0. \end{aligned}$$

Finally, using this the optimal shift δw for scrubbing the weights is

$$\begin{aligned}
\Delta w &= w_{\text{lin}}(\mathcal{D}_r) - w_{\text{lin}}(D) \\
&= -\nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \Theta_{rf} M \Theta_{rf}^T \Theta_{rr}^{-1} (f_0(\mathcal{D}_r) - Y_r) \\
&+ \nabla f_0(\mathcal{D}_f)^T M \Theta_{rf}^T \Theta_{rr}^{-1} (f_0(\mathcal{D}_r) - Y_r) \\
&+ \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \Theta_{rf} M (f_0(\mathcal{D}_f) - Y_f) \\
&- \nabla f_0(\mathcal{D}_f)^T M (f_0(\mathcal{D}_f) - Y_f) \\
&= \Big[I - \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \nabla f_0(\mathcal{D}_r) \Big] \nabla f_0(\mathcal{D}_f)^T M \Big[\Theta_{rf}^T \Theta_{rr}^{-1} (f_0(\mathcal{D}_r) - Y_r) \Big] \\
&+ \Big[I - \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \nabla f_0(\mathcal{D}_r) \Big] \nabla f_0(\mathcal{D}_f)^T M \Big[(Y_f - f_0(\mathcal{D}_f)) \Big] \\
&= P \nabla f_0(\mathcal{D}_f)^T M V,
\end{aligned}$$

where $P = I - \nabla f_0(\mathcal{D}_r)^T \Theta_{rr}^{-1} \nabla f_0(\mathcal{D}_r)$, $M = \left[\Theta_{ff} - \Theta_{rf}^T \Theta_{rr}^{-1} \Theta_{rf} \right]^{-1}$ and $V = [(Y_f - f_0(\mathcal{D}_f)) + \Theta_{rf}^T \Theta_{rr}^{-1} (Y_r - f_0(\mathcal{D}_r))]$.

CHAPTER 4

Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence

In chapter 3 we provide a procedure for unlearning by computing a closed form update which can be subtracted from the weights of a trained deep neural network. This update enables us to remove information from the weights or the activations of the network in a single shot, while also providing tight information theoretic bounds. chapter 3 is inspired from the idea of neural tangent kernels, which show that under appropriate initialization, and neural network width, deep networks behave as linear models around the initialization random gaussian initialization. This theory has inspired an entire field which aims to study the deep learning phenomenon through the eyes of linear models. However, in our experiments in chapter 3 we observed that models linearized around initialization random initialization have poor stability and as a results have worse forgetting bounds. Instead we showed that using pre-trained networks as initialization NTK linearization provides better empirical results.

In this chapter we seek to study which part of training dynamics of deep networks is critical for the final performance, and thus around which point should the deep network be linearized to obtain the forgetting perturbation from chapter 3. In this chapter we show that the initial part of training is extremely critical for the final performance, as the network crosses different bottlenecks during the initial transients to reach convex basins in the loss landscape around which the model can be linearized. Such periods of training are ”critical periods”. Once the model crosses the critical

period, it can be linearized and then the dynamics of a highly non-convex model can be mimicked by a linear model.

Towards this end we try to understand the role of initial training dynamics through the lens of regularization in deep networks. We try two types of regularization, (i) weight decay (or L_2 regularization) and (ii) data augmentation. Of the two L_2 regularization is more interesting to us as it aims to convexity the training loss function. Through our experiments we observe that L_2 regularization is only required until the network completes the initial critical period during training. Once the initial critical period is over, removal of L_2 regularization does not change the model performance as the model already reaches a convex basin, thus the smooth effect of L_2 regularization if no longer required for better generalization. This empirical justifies our choice of computing the NTK linearization in chapter 3 around some pre-trained weights which have crossed the critical periods for the fine-tuning task compared to the initial point. In the further chapters we show that instead of computing the optimal unlearning step for a linearized network and then applying it to a non-linear model, we can directly train linearized models which perform comparable (or even outperform) their non-linear counterparts. In the rest of the chapter we provide some rigorous empirical evidence to study critical periods for regularization in deep networks.

4.1 Preliminaries and notation

Given an observed input x (*e.g.*, an image) and a random variable y we are trying to infer (*e.g.*, a discrete label), we denote with $p_w(y|x)$ the output distribution of a deep network parameterized by weights w. For discrete y, we usually have $p_w(y|x) = \mathrm{softmax}(f_w(x))$ for some parametric function $f_w(x)$. Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, the cross-entropy loss of the network $p_w(y|x)$ on the dataset $\mathcal{D}$ is defined as $L_{\mathcal{D}}(w) := \frac{1}{N}\sum_{i=1}^N \ell(y, f_w(x_i)) = \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}}[-\log p_w(y_i|x_i)]$.

When minimizing $L_{\mathcal{D}}(w)$ with stochastic gradient descent (SGD), we update the weights w with an estimate of the gradient computed from a small number of samples (mini-batch). That is, $w_{t+1} \leftarrow w_t - \eta\mathbb{E}_{i\in\xi_t}[\nabla\ell(y_i, f_w(x_i))]$ where $\xi_t \subseteq \{1, \ldots, N\}$ is a random subset of indices

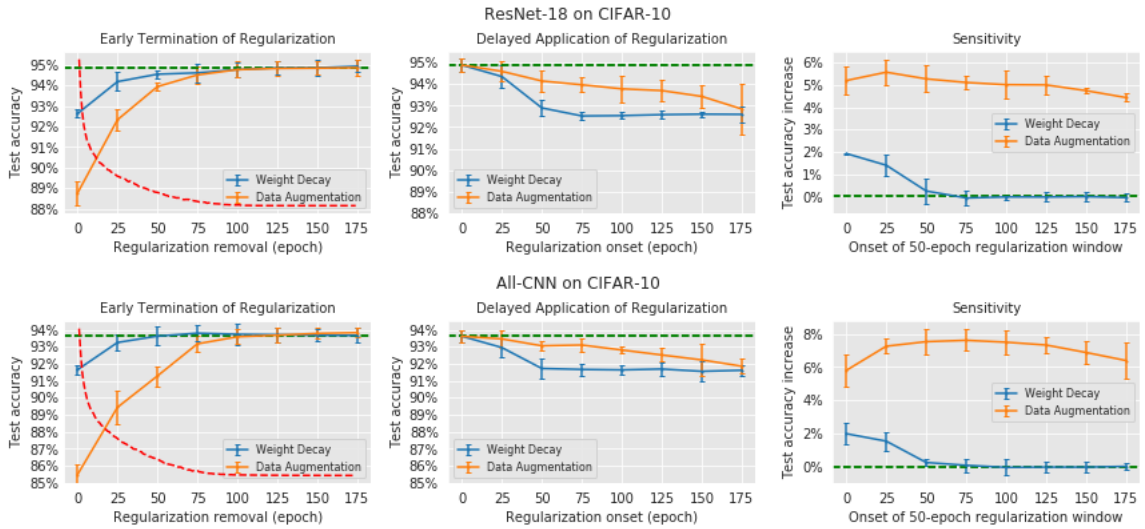

Figure 4.1: **Critical periods for regularization in DNNs : (Left)** Final test accuracy as a function of the epoch in which the regularizer is removed during training. Applying regularization beyond the initial transient of training (around 100 epochs) produces no appreciable increase in the test accuracy. In some cases, early removal of regularization *e.g.*, at epoch 75 for All-CNN, actually improves generalization. Despite the loss landscape at convergence being un-regularized, the network achieves accuracy comparable to a regularized one. **(Center)** Final test accuracy as a function of the onset of regularization. Applying regularization after the initial transient changes the convergence point (Fig. 4.2, B), but does not improve regularization. Thus, regularization does not influence generalization by re-shaping the loss landscape near the eventual solution. Instead, regularization biases the solution towards regions with good generalization properties during the initial transient. Weight decay (blue) shows a more marked time dependency than data augmentation (orange). The dashed line (green) in (Left) and (Center) corresponds to the final accuracy when we regularize throughout the training. **(Right)** Sensitivity (change in the final accuracy relative to un-regularized training) as a function of the onset of a 50-epoch regularization window. Initial learning epochs are more sensitive to weight decay compared to the intermediate training epochs for data augmentation. The shape of the sensitivity curve depends on the regularization scheme as well as the network architecture. For experiments with weight decay (or data augmentation), we apply data augmentation (or weight decay) throughout the training. Critical period for regularization occurs during the initial rapid decreasing phase of the training loss (red dotted line), which in this case is from epoch 0 to 75. The error bars indicate thrice the standard deviation across 5 independent trials.

of size $|\xi_t| = B$ (mini-batch size). In our implementation, weight decay (WD) is equivalent to imposing a penalty to the L_2 norm of the weights, so that we minimize the regularized loss $\mathcal{L} = L_{\mathcal{D}}(w) + \frac{\lambda}{2}\|w\|^2$.

Data augmentation (DA) expands the training set by choosing a set of random transformations of the data, $x' = g(x)$ (e.g, random translations, rotations, reflections of the domain and affine transformations of the range of the images), sampled from a known distribution P_g, to yield $\mathcal{D}'(g) = \{(g_j(x_i), y_i)\}_{g_j \sim P_g}$.

In our experiments, we choose g to be random cropping and horizontal flipping (reflections) of the images; $\mathcal{D}$ are the CIFAR-10 and CIFAR-100 datasets [Kri09], and the class of functions f_w are ResNet-18 [HZR16] and All-CNN [SDB14]. For all experiments, unless otherwise noted, we train with SGD with momentum 0.9 and exponentially decaying learning rate with factor $\gamma = 0.97$ per epoch, starting from learning rate $\eta = 0.1$ (see also Section 4.4.1).

4.2 Experiments

To test the hypothesis that regularization can have different effects when applied at different epochs of training, we perform three kinds of experiments. In the first, we apply regularization up to a certain point, and then switch off the regularizer. In the second, we initially forgo regularization, and switch it on only after a certain number of epochs. In the third, we apply regularization for a short window during the training process. We describe these three experiments in order, before discussing the effect of batch normalization, and analyzing changes in the loss landscape during training using local curvature (Fisher Information).

Regularization interrupted. We train standard DNN architectures (ResNet-18/All-CNN on CIFAR-10) using weight decay (WD) during the first t_0 epochs, then continue without WD. Similarly, we augment the dataset (DA) up to t_0 epochs, past which we revert to the original training set. We train both the architectures for 200 epochs. In all cases, the training loss converges to essentially zero for all values of t_0. We then examine the final test accuracy as a function of t_0 (Figure 4.1, Left).

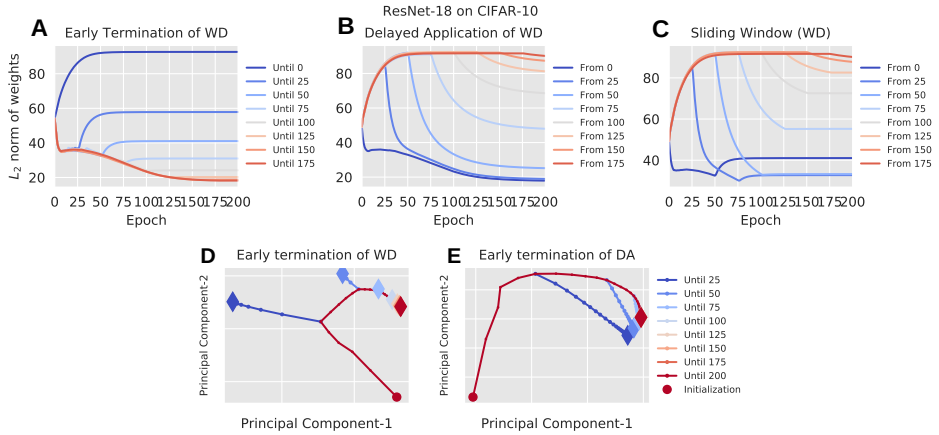

Figure 4.2: **Intermediate application or removal of regularization affects the final solution:** **(A-C)** L_2 norm of the weights as a function of the training epoch (corresponding to Figure 4.1 (Top)). The weights of the network move after application or removal of regularization, which can be seen by the change in their norm. Correlation between the norm of the weights and generalization properties is not as straightforward as lower norm implying better generalization. For instance, **(C)** applying weight decay only at the beginning (curve 0) reduces the norm only during the critical period, and yields higher norm asymptotically than, for example, curve 25. Yet it has better generalization. This suggests that the having a lower norm mostly help only during the critical period. We plot the norm of the weights for 200 training epochs to confirm that the weights stabilize and would not improve further with additional training. **(D)** PCA-projection of the training paths obtained removing weight decay at different times (see Section 4.4.1.1). Removing WD *before* the end of the critical period (curves 25, 50) makes the network converge to different regions of the parameter space. Removing WD *after* the critical period (curves 75 to 200) still sensibly changes the final point (in particular, critical periods are not due the optimization being stuck in a local minimum), but all points lie in a similar area, supporting the Critical Period interpretation of [ARS19]. **(E)** Same plots, but for DA, which unlike WD does not have a sharp critical period: all training paths converge to a similar area.

We observe that applying regularization beyond the initial transient (around 100 epochs) produces no measurable improvement in generalization (test accuracy). In Figure 4.3 (Left), we observe similar results for a different data distribution (CIFAR-100). Surprisingly, limiting regularization to the initial learning epochs yields final test accuracy that is as good as that achieved by regularizing to the end, even if the final loss landscapes, and hence the minima encountered at convergence, are different.

It is tempting to ascribe the imperviousness to regularization in the latter epochs of training (Figure 4.1, Left) to the optimization being stuck in a local minimum. After all, the decreased learning rate, or the shape of the loss around the minimum, could prevent the solution from moving. However, Figure 4.2 (A, curves 75/100) shows that the norm of the weights changes significantly after switching off the regularizer: the optimization is not stuck. The point of convergence *does* change, just not in a way that improves test accuracy.

The fact that applying regularization only at the very beginning yields comparable results, suggests that regularization matters *not* because it alters the shape of the loss function at convergence, reducing convergence to spurious minimizers, but rather because it "directs" the initial phase of training towards regions with multiple extrema with similar generalization properties. Once the network enters such a region, removing regularization causes the solution to move to different extrema, with no appreciable change in test accuracy.

Regularization delayed. In this experiment, we switch on regularization starting at some epoch t_0, and continue training to convergence. We train the DNNs for 200 epochs, except when regularization is applied late (from epoch 150/175), where we allow the training to continue for an additional 50 epochs to ensure the network's convergence. Figure 4.1 (Center) displays the final accuracy as a function of the onset t_0, which shows that there is a "critical period" to perform regularization (around epoch 50), beyond which adding a regularizer yields no benefit.

Absence of regularization can be thought of as a form of learning *deficit*. The permanent effect of temporary deficits during the early phases of learning has been documented across different tasks

Figure 4.3: **(Top) Critical periods for regularization are independent of the data distribution:** We repeat the same experiment as in Figure 4.1 on CIFAR-100. We observe that the results are consistent with Figure 4.1. The dashed line (green) in (Left) and (Right) denotes the final accuracy when regularization is applied throughout the training. The dashed line on top corresponds to ResNet-18, while the one below it corresponds to All-CNN. **(Center)** In the middle row (Left and Center), we show critical regularization periods for models trained on SVHN [NWC11] and ImageNet [DDS09a]. Critical periods for regularization also exists for regularization methods apart from weight decay and data augmentation, for example, Mixup [ZCD17] (Center Right). In fact, we observe that applying Mixup only during the critical period (first 75-100 epochs) results in better generalization compared to applying it throughout the training. **(Bottom) Critical regularization periods with a piecewise constant learning rate schedule:** We repeat experiment in Figure 4.1, but change the learning rate scheduling. Networks trained with piecewise constant learning rate exhibit behavior that is qualitatively similar to the exponentially decaying learning rate. The same experiment with constant learning rate is inconclusive since the network does not converge (see Appendix, Figure 4.11).

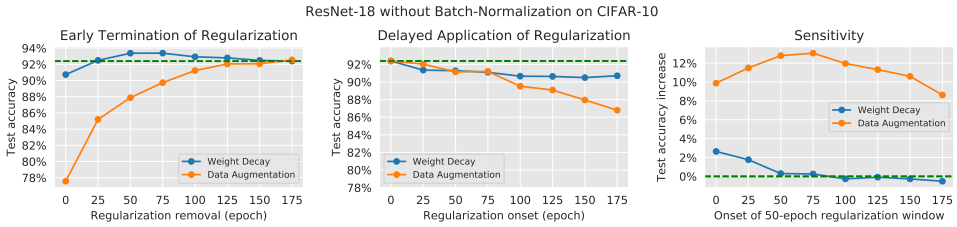

Figure 4.4: **Critical periods for regularization are independent of Batch-Normalization:** We repeat the same experiment as in Figure 4.1, but without Batch-Normalization. The results are largely compatible with previous experiments, suggesting that the effects are not caused by the interaction between batch normalization and regularization. **(Left)** Notice that, surprisingly, removal of weight decay right after the initial critical period actually improves generalization. **(Center)** Data augmentation in this setting shows a more marked dependency on timing. **(Right)** Unlike weight decay which mainly affects initial epochs, data augmentation is critical for the intermediate epochs.

and systems, both biological and artificial [ARS19]. *Critical periods* thus appear to be fundamental phenomena, not just quirks of biology or the choice of the dataset, architecture, learning rate, or other hyperparameters in deep networks.

In Figure 4.1 (Top Center), we see that delaying WD by 50 epochs causes a 40% increase in test error, from 5% regularizing all along, to 7% with onset $t_0 = 50$ epochs. This is despite the two optimization problems sharing the same loss landscape at convergence. This reinforces the intuition that WD does not improve generalization by modifying the loss function, lest Figure 4.1 (Center) would show an increase in test accuracy after the onset of regularization.

Here, too, we see that the optimization is not stuck in a local minimum: Figure 4.2 (B) shows the weights changing even after late onset of regularization. Unlike the previous case, in the absence of regularization, the network enters prematurely into regions with multiple sub-optimal local extrema, seen in the flat part of the curve in Figure 4.1 (Center).

Note that the magnitude of critical period effects depends on the kind of regularization. Figure 4.1 (Center) shows that WD exhibits more significant critical period behavior than DA. At convergence, data augmentation is more effective than weight decay. In Figure 4.3 (Center), we observe critical periods for DNNs trained on CIFAR-100, suggesting that they are independent of the data distribution.

Sliding Window Regularization. In an effort to regress which phase of learning is most impacted by regularization, we compute the maximum sensitivity against a sliding window of 50 epochs during which WD and DA are applied (Figure 4.1 Right). The early epochs are the most sensitive, and regularizing for a short 50 epochs yields generalization that is almost as if we had regularized all along. This captures the *critical period for regularization*. Note that the shape of the sensitivity to critical periods depends on the type of regularization: Data augmentation has essentially the same effect throughout training, whereas weight decay impacts critically only the initial epochs. Similar to the previous experiments, we train the networks for 200 epochs except, when the window onsets late (epoch 125/150/175), where we train for 50 additional epochs after the termination of the regularization window which ensures that the network converges.

Reshaping the loss landscape. L_2 regularization is classically understood as trading classification loss against the norm of the parameters (weights), which is a simple proxy for model complexity. The effects of such a tradeoff on generalization are established in classical models such as linear regression or support-vector machines. However, DNNs need not trade classification accuracy for the L_2 norm of the weights, as evident from the fact that the training error can always reach zero regardless of the amount of L_2 regularization. Current explanations [GBC16] are based on asymptotic convergence properties, that is, on the effect of regularization on the loss landscape and the minima to which the optimization converges. In fact, for learning algorithm that reduces to a convex problem, this is the only possible effect. However, Figure 4.1 shows that for DNNs, the critical role of regularization is to change the dynamics of the initial transient, which biases the model towards regions with good generalization. This can be seen in Figure 4.1 (Left), where despite halting regularization after 100 epochs, thus letting the model converge in the un-regularized loss landscape, the network achieves around 5% test error. Also in Figure 4.1 (Top Center), despite applying regularization after 50 epochs, thus converging in the regularized loss landscape, the DNN generalizes poorly (around 7% error). Thus, while there is reshaping of the loss landscape at convergence, this is not the mechanism by which deep networks achieve generalization. It is commonly believed that a smaller L_2 norm of the weights at convergence implies better generaliza-

tion [WRS17, NBS18]. Our experiments show no such causation: Slight changes of the training algorithm can yield solutions with larger norm that generalize better (Figure 4.2, (C) & Figure 4.1, Top right: onset epoch 0 vs 25/50).

Effect of Batch-Normalization. One would expect L_2 regularization to be ineffective when used in conjunction with Batch-Normalization (BN) [IS15], since BN makes the network's output invariant to changes in the norm of its weights. However, it has been observed that, in practice, WD improves generalization even, or especially, when used with BN. Several authors [ZWX19, HBG18, Laa17] have observed that WD increases the effective learning rate $\eta_{eff,t} = \eta_t / \|w_t\|_2^2$, where η_t is the learning rate at epoch t and $\|w_t\|_2^2$ is the squared-norm of weights at epoch t, by decreasing the weight norm, which increases the effective gradient noise, which promotes generalization [NVL15, JKA17, HHS17]. However, in the sliding window experiment for L_2 regularization, we observe that networks with regularization applied around epoch 50, despite having smaller weight norm (Figure 4.2 (C), compare onset epoch 50 to onset epoch 0) and thus a higher effective learning rate, generalize poorly (Figure 4.1 Top Right: onset epoch 50 has a mean test accuracy increase of 0.24% compared to 1.92% for onset epoch 0). We interpret the latter (onset epoch 0) as having a higher effective learning rate during the critical period, while for the former (onset epoch 50) it was past its critical period. Thus, previous observations in the literature should be considered with more nuance: we contend that an increased effective learning rate induces generalization only insofar as it modifies the dynamics *during the critical period*, reinforcing the importance of studying *when* to regularize, in addition to how. In Figure 4.9 in the Appendix, we show that the initial effective learning rate correlates better with generalization (Pearson coefficient 0.96, p-value ¡ 0.001) than the final effective learning rate (Pearson coefficient 0.85, p-value ¡ 0.001).

We repeat the experiments in Figure 4.1 without Batch-Normalization (Figure 4.4). We observe a similar result, suggesting that the positive effect of weight decay during the transient cannot be due solely to the use of batch normalization and an increased effective learning rate.

Weight decay, Fisher and flatness.

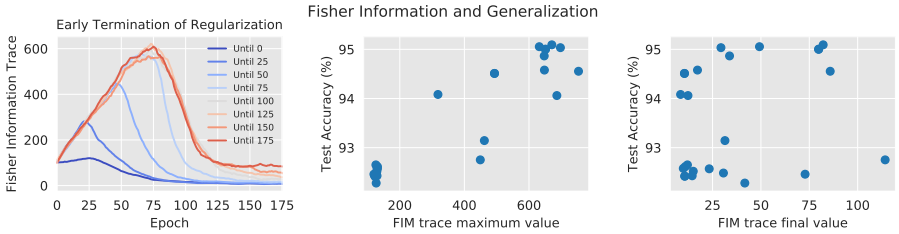

Figure 4.5: **Fisher Information and generalization: (Left)** Trace of the Fisher Information Matrix (FIM) as a function of the training epochs. Weight decay increases the peak of the FIM during the transient, with negligible effect on the final value (see left plot when regularization is terminated beyond 100 epochs). The FIM trace is proportional to the norm of the gradients of the cross-entropy loss. FIM trace plots for delayed application/sliding window can be found in the Appendix (Figure 4.7) **(Center) & (Right): Peak vs. final Fisher Information correlate differently with test accuracy:** Each point in the plot is a ResNet-18 trained on CIFAR-10 achieving 100% training accuracy. Surprisingly, the maximum value of the FIM trace correlates far better with generalization than its final value, which is instead related to the local curvature of the loss landscape ("flat minima"). The Pearson correlation coefficient for the peak FIM trace is 0.92 (p-value ¡ 0.001) compared to 0.29 (p-value ¿ 0.05) for the final FIM trace.

Generalization for DNNs is often correlated with the flatness of the minima to which the network converges during training [HS97, LXT18, KMN16, CCS16], where solutions corresponding to flatter minima seem to generalize better. In order to understand if the effect of regularization is to increase the flatness at convergence, we use the Fisher Information Matrix (FIM), which is a semi-definite approximation of the Hessian of the loss function [Mar14] and thus a measure of the curvature of the loss landscape. We recall that the Fisher Information Matrix is defined as:

$$F := \mathbb{E}_{x\sim\mathcal{D}'(x)}\mathbb{E}_{y\sim p_w(y|x)}[\nabla_w \log p_w(y|x)\nabla_w \log p_w(y|x)^T].$$

In Figure 4.5 (Left) we plot the trace of FIM against the final accuracy. Notice that, contrary to our expectations, weight decay increases the FIM norm, and hence curvature of the convergence point, but this still leads to better generalization. Moreover, the effect of weight decay on the curvature is more marked *during the transient* (Figure 4.5). This suggests that the peak curvature reached during the transient, rather than its final value, may correlate with the effectiveness of regularization. To test this hypothesis, we consider the DNNs trained in Figure 4.1 (Top) and plot the relationship between peak/final FIM value and test accuracy in Figure 4.5 (Center, Right):

Indeed, while the peak value of the FIM strongly correlates with the final test performance (Pearson coefficient 0.92, p-value ¡ 0.001), the final value of the FIM norm does not (Pearson 0.29, p-value ¿ 0.05). We report plots of the Fisher Norm for delayed/sliding window application of WD in the Appendix (Figure 4.7).

The FIM was also used to study critical period for changes in the data distribution in [ARS19], which however in their setting observe an anti-correlation between Fisher and generalization. Indeed, the relationship between the flatness of the convergence point and generalization established in the literature emerges as rather complex, and we may hypothesize a more complex bias-variance trade-off like a connection between the two, where either too low or too high curvature can be detrimental.

Jacobian norm. [ZWX19] relates the effect of regularization to the norm of the Gauss-Newton matrix, $G = \mathbb{E}[J_w^T J_w]$, where J_w is the Jacobian of $f_w(x)$ w.r.t w, which in turn relates to norm of the networks input-output Jacobian. The Fisher Information Matrix is indeed related to the GN matrix (more precisely, it coincides with the generalized Gauss-Newton matrix, $G = \mathbb{E}[J_w^T H J_w]$, where H is the Hessian of $\ell(y, f_w(x))$ w.r.t. $f_w(x)$). However, while the GN norm remains approximately constant during training, we found the changes of the Fisher-Norm during training (and in particular its peak) to be informative of the critical period for regularization, allowing for a more detailed analysis.

4.3 Discussion and Conclusions

We have tested the hypothesis that there exists a *"critical period"* for regularization in training deep neural networks. Unlike classical machine learning, where regularization trades off the training error in the loss being minimized, DNNs are not subject to this trade-off: One can train a model with sufficient capacity to zero training error regardless of the norm constraint imposed on the weights. Yet, weight decay works, even in the case where it seems it should not, for instance when the network is invariant to the scale of the weights, *e.g.*, in the presence of batch normalization. We

believe the reason is that regularization affects the early epochs of training by biasing the solution towards regions that have good generalization properties. Once there, there are many local extrema to which the optimization can converge. Which to is unimportant: Turning the regularizer on or off changes the loss function, and the optimizer moves accordingly, but test error is unaffected, at least for the variety of architectures, training sets, and learning rates we tested.

We believe that there are universal phenomena at play, and what we observe is not the byproduct of accidental choices of training set, architecture, and hyperparameters: One can see the *absence* of regularization as a learning deficit, and it has been known for decades that deficits that interfere with the early phases of learning, or *critical periods*, have irreversible effects, from humans to songbirds and, as recently shown by [ARS19], deep neural networks. Critical periods depend on the type of deficits, the task, the species or architecture. We have shown results for two datasets, two architectures, two learning rate schedules.

While our exploration is by no means exhaustive, it supports the point that considerably more effort should be devoted to the analysis of the transient dynamics of Deep Learning. To this date, most of the theoretical work in Deep Learning focuses on the asymptotics and the properties of the minimum at convergence.

Our hypothesis also stands when considering the interaction with other forms of generalized regularization, such as batch normalization, and explains why weight decay still works, even though batch normalization makes the activations invariant to the norm of the weights, which challenges previous explanation of the mechanisms of action of weight decay.

We note that there is no trade-off between regularization and loss in DNNs, and the effects of regularization cannot (solely) be to change the shape of the loss landscape (WD), or to change the variety of gradient noise (DA) preventing the network from converging to some local minimizers, as without regularization in the end, everything works. The main effect of regularization ought to be on the transient dynamics before convergence.

At present, there is no viable theory on transient regularization. The empirical results we present

should be a call to arms for theoreticians interested in understanding Deep Learning. A possible interpretation advanced by [ARS19] is to interpret critical periods as the (irreversible) crossing of narrow bottlenecks in the loss landscape. Increasing the noise – either by increasing the effective learning rate (WD) or by adding variety to the samples (DA) – may help the network cross the right bottlenecks while avoiding those leading to irreversibly sub-optimal solutions. If this is the case, can better regularizers be designed for this task?

4.4 Appendix

4.4.1 Details of the Experiments

We use the standard ResNet-18 architecture [HZR16] in all the experiments, stated unless otherwise. We train all the networks using SGD (momentum = 0.9) with a batch-size of 128 for 200 epochs, except when regularization is applied late during the training, where we train for an extra 50 epochs, to ensure the convergence of the networks. For experiments with ResNet-18, we use an initial learning rate of 0.1, with learning rate decay factor of 0.97 per epoch and a weight decay coefficient of 0.0005. In the piece-wise constant learning rate experiment, we use an initial learning rate of 0.1 and decay it by a factor of 0.1 every 60 epochs. While in the constant learning experiment we fix it to 0.001. For the All-CNN [SDB14] experiments, we use an initial learning rate of 0.05 with a weight decay coefficient of 0.001 and a learning rate decay of 0.97 per epoch. For All-CNN, we do not use dropout and instead we add Batch-Normalization to all layers.

4.4.1.1 Path Plotting

We follow the method proposed by [LXT18] to plot the training trajectories of the DNNs for varying duration of regularization (Figure 4.2 A). More precisely, we combine the weights of the network (stored at regular intervals during the training) for different duration of regularization into a single matrix M and then project them on the first two principal components of M.

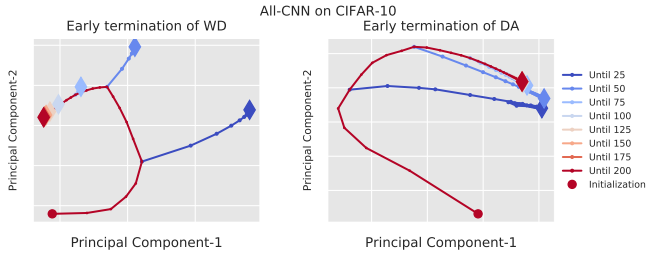

Figure 4.6: PCA-projection of the training paths for All-CNN on CIFAR-10.

4.4.2 Additional Experiments

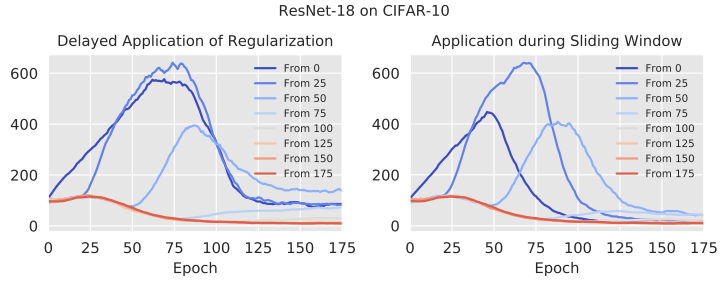

Figure 4.7: Trace of FIM as a function of training epochs for delayed and sliding window application of weight decay for ResNet-18 on CIFAR-10.

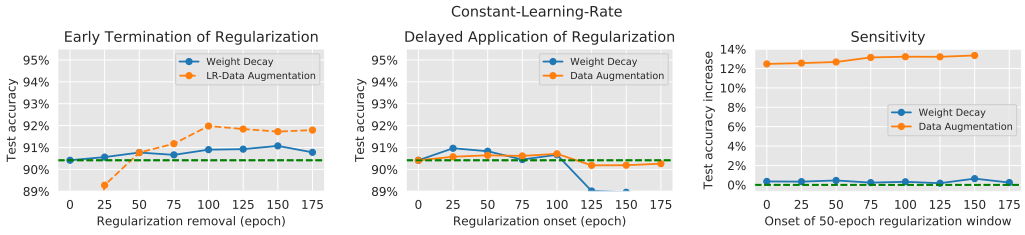

Figure 4.8: We repeat the experiments from Figure 4.1 and replace the exponential learning rate with a constant learning rate (lr = 0.001). However, those are inconclusive since the error achievable with a constant rate does not saturate performance on the datasets we tested them, with the architectures we used. Test error is almost twice than what was achieved with an exponential or piecewise constant learning rate. It is possible that careful tuning of the constant could achieve baseline performance and also display critical period phenomena.

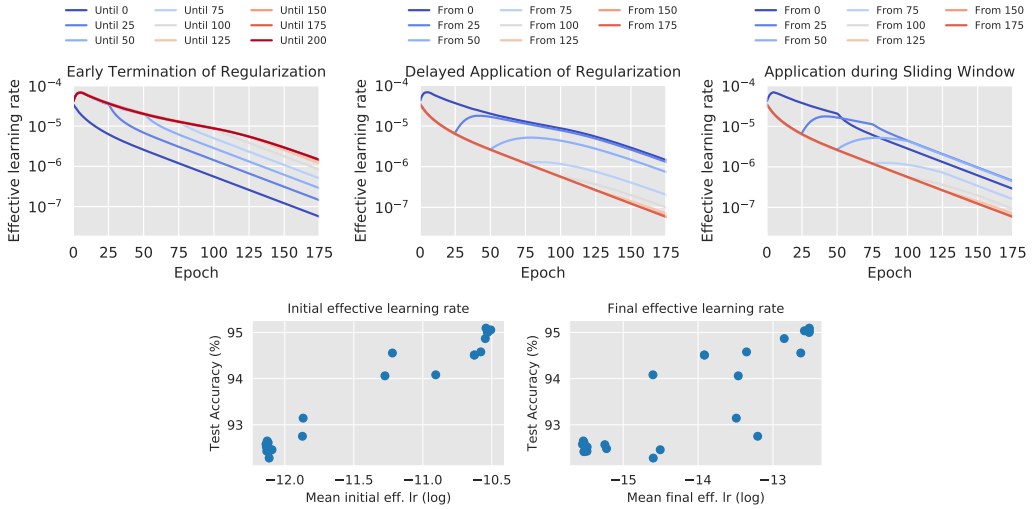

Figure 4.9: **Effect of learning rate on generalization: (Top)** The effective learning rate $\eta_{eff,t} = \eta_t / \|w_t\|_2^2$, where η_t is the learning rate at epoch t and $\|w_t\|_2^2$ is the norm of weights at epoch t. [ZWX19, HBG18, Laa17], shown as a function of the training epoch for the experiments from Figure 4.1 (Top blue) / Figure 4.2. **(Left)** and **(Center)** The effective learning rate is higher throughout the training (which includes the critical periods) for models which generalize better (Figure 4.1 Top blue) **(Right)** When we apply weight decay in a sliding window, the effective learning rate for the dark blue curve (weight decay from 0-50) is higher during the critical period but lower afterwards, compared to light blue curves (weight decay from 25-75, 50-100). However, the dark blue curve yields a higher final accuracy (test accuracy increase of 1.92%) compared to light blue curves (test accuracy increase of 1.39%, 0.24% respectively), despite having a lower effective learning rate for the majority of the training. This suggests that having a higher effective learning rate during the critical periods is more conducive to good generalization when using weight decay. **(Bottom)** We test this further by plotting the relationship between the effective learning rate and final test accuracy. Mean initial effective learning rate is the average effective learning rate over the first 100 epoch, while mean final effective learning rate is the average computed over the final 100 epochs. The initial effective learning rate correlates better with generalization (Pearson correlation coefficient of **0.96** (p-value ¡ 0.001)) compared to the final effective learning rate (Pearson correlation coefficient of **0.85** (p-value ¡ 0.001)).

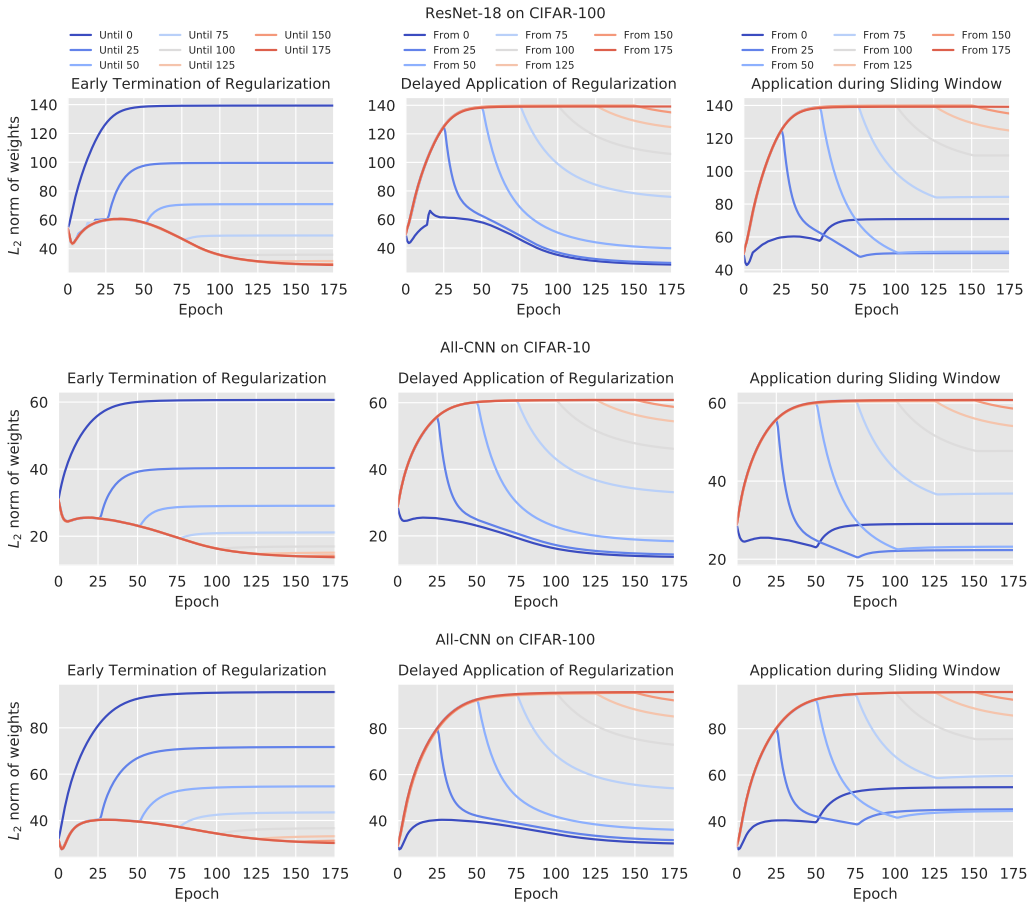

Figure 4.10: L_2 norm of the weights as a function of the training epoch. We observe similar results (compared to Figure 4.2) for different architectures (ResNet-18 and All-CNN) and different datasets (CIFAR-10 and CIFAR-100).

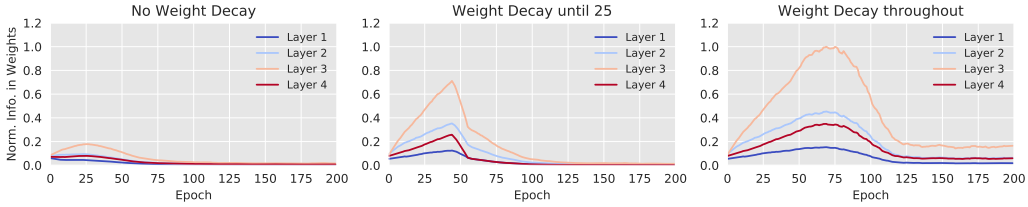

Figure 4.11: Plot of layer-wise Fisher Information for ResNet-18 trained on CIFAR-10. Unlike [ARS19] we do not observe changes in the relative ordering of the21layers, which makes sense since unlike the experiments in [ARS19], the underlying data distribution is not changing

CHAPTER 5

Linear-Quadratic Fine-tuning

In chapter 3 we provided a single shot forgetting procedure for non-convex models using the NTK linearization around some pre-trained network. This procedure enables to perform quick forgetting from the weights or the activations. Even though the procedure performs well empirically, it still involves some approximations, most importantly applying the optimal forgetting update of a linear model to a non-linear model. While in chapter 4 we show that the initial training dynamics are critical for the final performance. Thus fine-tuning pre-trained models for practical downstream tasks provides better empirical approximation for forgetting. To overcome all these limitations in this chapter we provide a training procedure for fine-tuning large-scale linearized networks. These networks are first-order Taylor series approximations of the activations with respect of the weights around some pre-training point (for instance ImageNet pre-trained networks). The Taylor series perturbation is what we learn while fine-tuning these models for different downstream tasks. Unlike chapter 3 where the network fine-tuned is non-linear but the optimal forgetting step is for a linear model, here we inherently train a linear model, and in chapter 6 provide a perfect forgetting procedure for this linearized network. Since these models are linear in their parameters, and trained by optimizing a convex loss function, have predictable behavior with respect to changes in the training data, initial conditions, and optimization.

We show that by linearizing the networks around a pre-trained point, along with some modifications of the training loss landscape and the optimization algorithm, we can fine-tune large linearized models which perform comparable to their non-linear counterparts. This enables us to exploit the strength of non-linear models in obtaining good generalization and the theoretical properties of

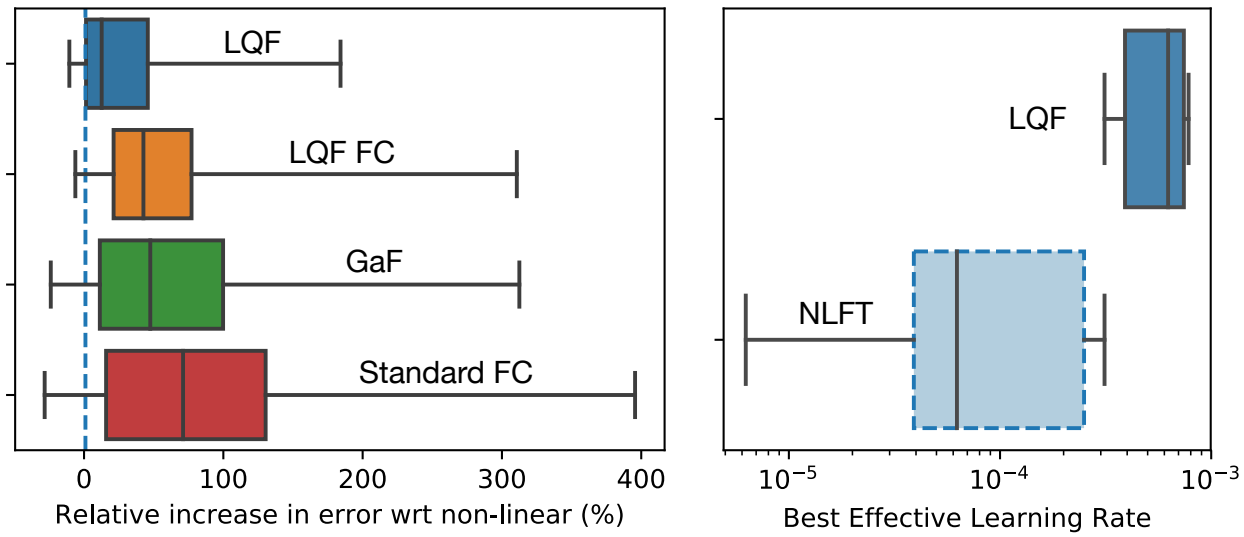

Figure 5.1: **Linear vs. nonlinear fine-tuning (NLFT).** (**Left**) Box-plot of the distribution of test errors achieved by different linearization methods on the datasets in Table 5.1, relative to the error achieved by NLFT (dashed line at origin). Whiskers show best/worst results, boxes extend from lower to upper quartiles (the results on half of the datasets are concentrated in the box), the central line represents the median increase in error. GaF [MLL20] (green, 47% median error increase) is better than training a linear classifier on a fixed embedding (red, 71% increase), but slightly worse than LQF applied just to that linear classifier (LQF FC, orange, 42% increase). LQF is the closest linear method to the non-linear paragon (blue, 12% increase). (**Right**) We show the distribution of best effective learning rates as the task varies. While for NLFT we need to search in a wide range to find the optimal training parameters for a task (wide dashed box), for LQF the same learning rate works almost equally well for all tasks (narrow solid box).

linear models to perform efficient and optimal forgetting. Since we linearize around a pre-trained network, the fine-tuning model is already past its critical period as a result training a linearized model performs comparably to a non-linear model which partly justifies the success of this method.

In the subsequent sections we describe our method, provide a detailed describe of its subcomponents, along with its applications for different downstream tasks which is not restricted to forgetting, for instance, few-shot learning, robustness and interpretibility.

5.1 Method

Linearized model. LQF addresses the lack of robustness and interpretability of non-linear deep models by replacing the network itself with a first-order linear approximation. Let $f_w(x)$ denote the output of a deep network with weighs w on an input image x, let w_0 denotes an initial set of weights, for example obtained after pre-training on ImageNet. We consider the *linearization* $f_w^{\text{lin}}(x)$ of the network $f_w(x)$ given by the first-order Taylor expansion of $f_w(x)$ around w_0:

$$f_w^{\text{lin}}(x) = f_{w_0}(x) + \nabla_w f_{w_0}(x) \cdot (w - w_0). \tag{5.1}$$

Note that $f_w^{\text{lin}}(x)$ is linear with respect to the weights w; however, due to the non-linear activation functions, it is still a highly non-linear function of the input x. If the approximation is accurate, ideally we want the linearized f_w^{lin} and the original f_w to reach similar accuracy when trained:

$$\text{accuracy}(f_{\hat{w}}^{\text{lin}}, \mathcal{D}_{\text{test}}) \approx \text{accuracy}(f_w, \mathcal{D}_{\text{test}}) \tag{5.2}$$

where $\hat{w}$ and w are obtained by minimizing, respectively, the loss of the linear and non-linear model on a training set $\mathcal{D}$. The obvious advantage of the linear model is that, for strongly convex loss functions, the global optimum $\bar{w}$ is unique and, for a quadratic loss functions, can even be written in closed form. So, if eq. (5.2) was satisfied, we would stand to gain in terms of speed of fine-tuning, performance (convergence is guaranteed to the global optimum), and interpretability (the effect of a training sample on the test prediction can be computed exactly).

Training a linearized model requires computing the product of the Jacobian $\nabla_w f_{w_0}(x)$ with the weights w, which would require a separate backward pass for each sample. However, [MLL20] shows that using Gradients as Features (GaF) the Jacobian-Vector product of eq. (5.1) can be computed with a modified forward pass, at a cost comparable to that of the running the original network.

Accuracy of linearized models. In Figure 7.1 and Table 5.1 we show that the accuracy of linearized models suffers compared to the paragon of NLFT. In most datasets, even the simple linear baseline

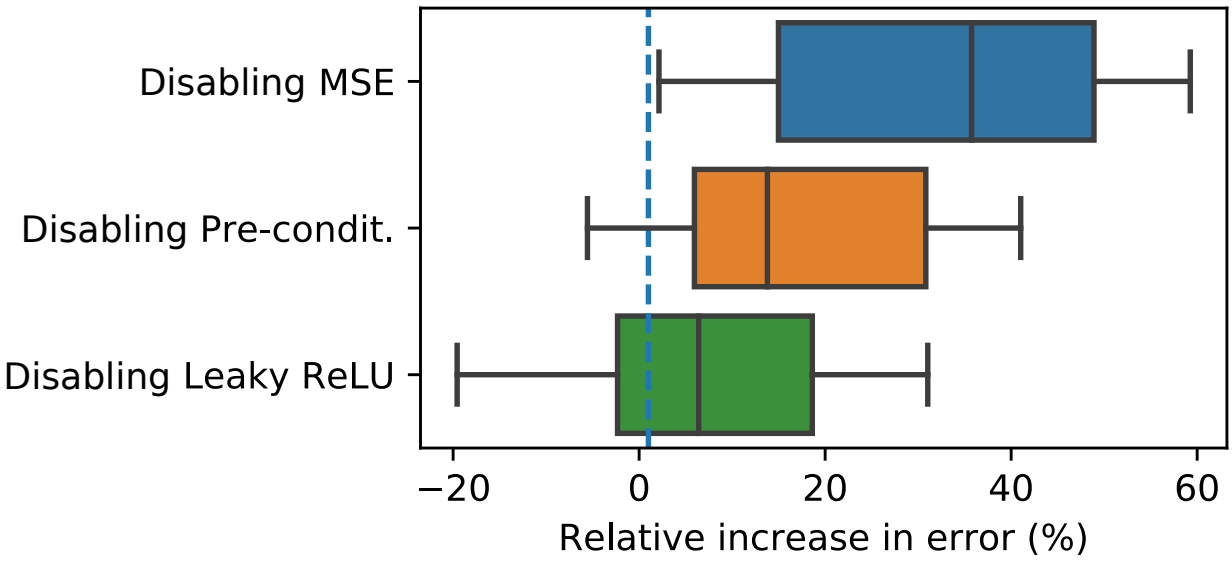

Figure 5.2: **Ablation study.** Box plot showing relative error increase on the datasets in Table 5.1 when removing consititutive components of LQF. Removing MSE in favor of the standard CE loss yields the largest increase in error (blue). Removing K-FAC pre-conditioning, leaving standard SGD (orange), has also a sizeable effect. Finally, on most datasets, Leaky-ReLU performs better than standard ReLU (green).

obtained by fine-tuning just the linear classifier (fully-connected, or FC, layer) is only slightly worse than GaF, violating (5.2). We now analyze the causes of these differences between the dynamics of fine-tuning a non-linear network and its linearized version, and propose a set of changes that lead to LQF.

5.1.1 Loss function

We use a regularized mean-squared error (MSE) loss, even if the original model is pre-trained by minimizing the regularized cross-entropy loss:

$$L_{\text{MSE}}(f;\mathcal{D}) = \sum_{(x_i,y_i)\in\mathcal{D}} \|\alpha y_i - f(x_i)\|^2 + \frac{\lambda}{2}\|w - w_0\|^2, \tag{5.3}$$

where y_i is the one-hot vector encoding the class label and α is a scaling factor for the targets (we set $\alpha = 15$ as in [HB21]).[1] There are several reasons to make this change. First, while standard DNNs have several normalization layers (such as batch normalization) that keep the output bounded, the linear model $f_w^{\text{lin}}(x)$ in eq. (5.1) directly outputs the dot product $\nabla_w f_{w_0}(x) \cdot w$, which involves tens of millions of weights. This product can easily grow large and saturate the softmax of the cross-entropy loss, making the training process unstable. When that happens, we observe that only the last layer features are effectively trained. On the other hand, the MSE loss does not saturate and forces the output to remain close to the target one-hot vectors, and hence bounded. We also note that using MSE gives better results *even* for standard non-linear training (see Appendix), especially when training on small datasets.

A secondary effect of using the MSE loss in conjunction with a linear model is that we can write the unique global optimum in closed form:

$$w^* = (J^T J + \lambda I)^{-1} J(Y - f_0(X)) \tag{5.4}$$

where $J \equiv \nabla_w f_{w_0}(X)^T$ is the matrix of the Jacobians of all the samples in the training set and Y is the matrix of all target vectors. While w^* cannot be computed directly using this formula due to large size of the matrices, in Section 5.1.4 we show that eq. (5.4) allows us to easily compute the influence of a given training sample on the test predictions, enhancing interpretability of the training process.

5.1.2 Pre-conditioning with K-FAC

The MSE loss in eq. (5.3) is strictly convex, so SGD is guaranteed to converge to the unique global minimum, provided an appropriate learning rate decaying schedule is used [Nes13]. However, different directions in the loss landscape may have different curvature, slowing convergence: the same learning rate may be too fast high-curvature directions and too small for flat ones. Convergence

[1] Note that we penalize the distance of the weights w from the pretraining w_0. This is more natural than the weight decay penalty $\frac{\lambda}{2}\|w\|^2$ in our setting, since we are optimizing for the perturbation $w - w_0$. It may also lead to better transfer results on some tasks [LGD18].

speed is governed by the condition number of the Hessian matrix (ratio between the largest and smallest eigenvalues, i.e., maximum and minimum curvature), which can be modified by multiplying the update with a *pre-conditioning* matrix A_t

$$w_{t+1} = w_t - \eta A_t g_t, \tag{5.5}$$

where g_t is the batch gradient computed at step t and η is the learning rate. The matrix A_t allows training at different speed in each direction. In a quadratic problem with constant preconditioning $A_t = A$, the expected distance from the optimum w^* at time t using adaptive SGD is given by:

$$\mathbb{E}[w_t - w^*] = (I - \eta AH)^t (w_0 - w^*), \tag{5.6}$$

where H is the (constant) Hessian of the loss function. In particular, the fastest convergence is obtained when $A = H^{-1}$ is the inverse of the hessian. For this reason, most adaptive methods try to approximate H^{-1}.

Adaptive methods are commonplace in convex optimization and some [KB15] are widely used in deep learning. However, for large-scale visual classification, standard SGD with momentum performs comparably or better than adaptive methods. We hypothesize that this may be due to the changing curvature during training, which causes the condition number to decrease, making the problem well-conditioned even without adaptation (see Figure 5.3). However, this cannot happen when training a linearized network, since the curvature of eq. (5.3) is constant and fully determined in pre-training. For this reason, we add a pre-conditioner for eq. (5.3) in LQF.

In our case, since the problem is quadratic, the Hessian coincides with the Fisher Information Matrix (FIM). We thus use the Kronecker-Factorized (K-FAC) approximation of the FIM as pre-conditioner [MG15]. Unlike diagonal approximations, K-FAC allows us to estimate off-diagonal terms while remaining tractable.

K-FAC is efficient for LQF. On standard non-linear networks, K-FAC may make the weights converge too quickly to suboptimal sharp minima that do not generalize well and, since the curvature changes over time, the approximation of the Hessian needs to be updated. However, in LQF there is

a unique minimizer and the curvature is constant. This makes the use of K-FAC especially efficient in our setting. Compared to other adaptive methods like Adam [KB15], in our setting K-FAC provides an accurate estimation of the Hessian inverse which can be used to interpret the effect of a training sample (Section 5.1.4) and to easy select an appropriate learning rate.

Learning rate selection. From eq. (5.6) it follows that, to guarantee convergence, the norm of the eigenvalues of $I - \eta AH$ should be smaller than one. In particular, if A is a good approximation of H, the eigenvalues of AH are centered around 1 and $\eta = 1$ is optimal. In our case A will not be a perfect approximation of H^{-1}, however since we expect $\lambda_{\text{max}}(A^{-1}H) \approx 1$ and we need $\eta < 2/\lambda_{\text{max}}(A^{-1}H)$ to guarantee convergence, we choose a conservative $\eta = 0.1$, which we observe gives consistent performance on all datasets without further tuning (Figure 7.1, right).

5.1.3 Leaky-ReLU activations

Rectified linear units (ReLUs) [NH10, SWT15] leave the positive component of the input unchanged and replace the negative component with zero, which blocks information from certain weights. This is not harmful during normal training, since batch normalization helps avoiding dead filters. However, in linearized models only the gradient of the filter at initialization matters in computing the Jacobian-Vector Product. If this is zero, it will stunt training for that filter. To prevent this, we simply replace ReLU with Leaky-ReLu [MHN13], which does not annihilate the negative component but simply replaces it with a value proportional to the input. Another advantage of Leaky-ReLU is that its first order approximation near zero is more accurate than that of ReLU, making the first-order approximation in eq. (5.1) more accurate.

Performance of LQF. After accounting for these changes, we repeat the previous experiments on different datasets and report the overall results in Figure 7.1 and detailed in Table 5.1. LQF outperforms other linearization methods on all the datasets, and achieves performance comparable to non-linear fine-tuning on most. In the low data regime, the strong inductive bias of the linear model acts as a regularizer and the linearized model outperforms the non-linear fine-tuned one

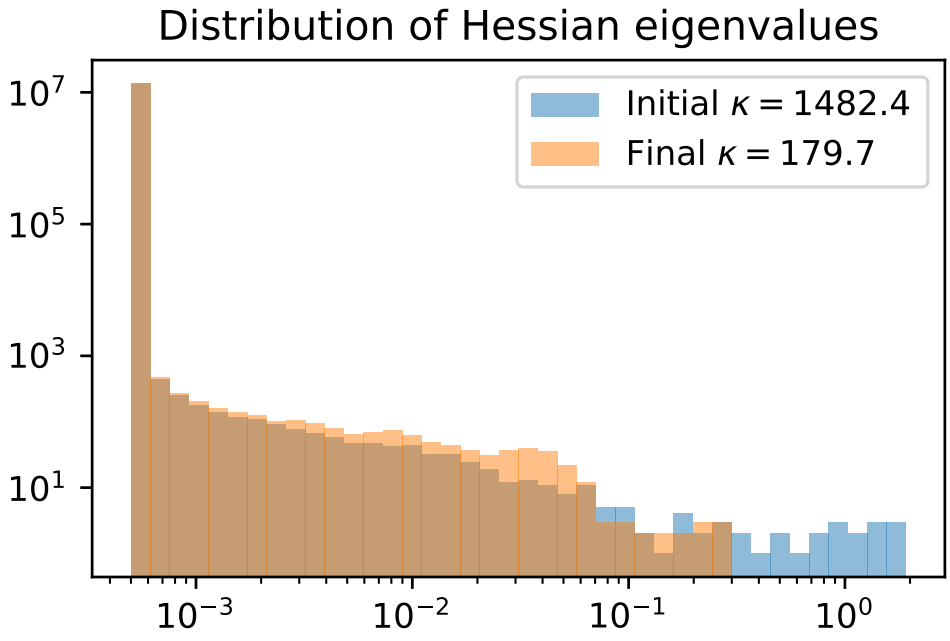

Figure 5.3: **Unlike SGD, LQF requires pre-conditioning.** Histogram of the eigenvalues of the Hessian loss function at initialization (blue) and at the end of optimization of a non-linear network (orange). Note that at initialization a few directions have very large eigenvalues while many directions have much smaller eigenvalues, resulting in a high condition number $\kappa = 1482.4$. However, SGD naturally moves to areas of the loss landscape that are better conditioned, with $\kappa = 179.7$ at the end of training (note that the largest eigenvalues decrease), thus making convergence easier. This "automatic conditioning" makes pre-conditioning un-necessary in non-linear networks. On the other hand, since in LQF the Hessian is fixed throughout the training, we need to use K-FAC pre-conditioning to facilitate convergence.

(Figure 5.4). We discuss the results in detail in Section 7.2.

5.1.4 Interpretability

LQF gives us an efficient way of estimating the influence, or "effect" of a given training sample on the final weights and on the test predictions of the DNN, as well as measuring the informativeness to the training process. This can be used for dataset summarization (Sect. 5.2.6) and to interpret model predictions: Had a particular sample (cause) not been used for training, how would the final model (effect) differ? Using eq. (5.4) we have the following expression for the weights w_{-i} we would have obtained if training without the sample x_i:

$$w^*_{-i} = w^* + (F_{-i} + \lambda I)^{-1} g_i. \tag{5.7}$$

where $g_i \equiv \nabla_{w^*} f_w(x)$ is the Jacobian of the i-th sample, $e_i \equiv \frac{1}{N}(y - w^* g_i)$ is the weighted prediction error on the sample x_i, N is the number of training samples, and F_{-i} is the hessian (or, equivalently, the FIM) computed without x_i.

Using eq. (5.7) we can estimate how much the prediction $f_w(x_{\text{test}})$ on a test sample x_{test} would change had we not seen the training sample x_i. That is, we can measure which training sample has contributed more to a correct or incorrect decision of the network. Using eq. (5.7), in the Appendix we show that

$$\begin{aligned} f_{w^*}(x_{\text{test}}) - f_{w^*_{-i}}(x_{\text{test}}) \approx \\ \Big(1 - \frac{\alpha_i}{N - 1 + \alpha_i}\Big) e_i g^T_{\text{test}} (F + \lambda I)^{-1} g_i, \end{aligned} \tag{5.8}$$

where $\alpha_i \simeq g_i (F + \lambda I)^{-1} g_i^T$. Since K-FAC already provides an approximation of $(F + \lambda I)^{-1}$, we can estimate the expression efficiently.

More generally, one can define the unique functional information [HAP21] of a training sample x_i as the expected influence that the sample has on the network output on a validation set:

$$\text{F-SI}((x_i, y_i)) = \mathbb{E}_{x \sim \mathcal{D}_{\text{val}}} \big[\| f_w(x) - f_{w_{-i}}(x) \|^2 \big] \tag{5.9}$$

This measure – which can be used for dataset summarization and to interpret what the network learn from a training samples (Section 5.2.6) – is also easily estimated using eq. (5.8).

5.2 Results

We test LQF on several standard vision classification tasks used to benchmark fine-tuning learning [LGD18, LCY20] and additional tasks to test cross-domain transfer (see Appendix for details on the datasets). We find that LQF outperforms other linearization methods on almost all tasks. LQF also outperforms standard non-linear training (NLFT) in tasks such as few-shot learning and on-line learning, where scarce data causes training instabilities. We also illustrate the benefits of interpretability.

Architecture and training. As the base model for linearization we use a ResNet-50 with Leaky-ReLU pre-trained on ImageNet. We implement the forward pass of eq. (5.1) using [MLL20], and minimize the MSE loss, with K-FAC pre-conditioning (using the implementation of [GLB18]). For NLFT, we also use a pre-trained ResNet-50 with ReLU activations, and minimize the cross-entropy loss. All experiments are conducted with SGD with momentum 0.9, learning rates $\eta \in \{0.01, 0.001\}$, weight decay $\in \{10^{-4}, 10^{-5}\}$ and report the best results. We discuss further training details in the Appendix.

5.2.1 Comparison

Ideally, LQF should achieve close performance to NLFT on all tasks. We also compare with two other linearization methods: Standard FC, which uses the pre-trained network as a feature and only retrains the last FC layer, and Gradients as Features (GaF) [MLL20] which linearizes the network without any of the changes described in Section 5.1. We also introduce a further baseline, LQF FC, which uses the changes in Section 5.1, but only to train the final FC layer.

We train each model on several "coarse-grained" datasets, covering different tasks and domains

	NLFT	LQF*	LQF FC*	GaF	FC
Caltech-256 [GHP]	14.1	**14.5**	17.4	15.4	15.7
Chest X-Ray [KGC18]	4.6	**7.1**	8.3	9.5	11.2
Malaria Cells [RAP18]	3.2	**4.1**	5.5	4.8	6.4
MIT-67 [QT09]	20.5	**20.7**	24.7	23.1	24.3
Oxford Pets [PVZ12]	6.7	**6.9**	7.7	7.4	7.7
Fine-grained datasets (sorted by ImageNet distance)					
Stanf. Dogs [KJY11]	13.8	12.4	12.9	10.5	**9.9**
Ox. Flowers [NZ06]	7.1	**7.1**	9.7	13.3	13.6
CUB-200 [WBM10]	19.6	**24.0**	29.1	28.4	29.2
Aircrafts [MKR13]	14.5	**34.5**	43.8	48.4	54.2
Stanf. Cars [KSD13]	9.6	**27.1**	39.2	39.4	47.3

Table 5.1: **Test error of linear and non-linear fine-tuning on coarse-and fine-grained classification**. On almost all datasets LQF outperforms all other linear methods. Moreover, on most coarse datasets LQF is within 0.5% absolute error from standard non-linear fine-tuning. Linear methods performs worse than NLFT on fine-grained machine classification tasks that have a large domain gap with respect to ImageNet pretraining – e.g., *Stanford Cars* and *FGVC Aircrafts*. However, even in this case LQF reduces the error by up to 20% with respect to competing methods.

and report the results in Table 5.1. In Figure 7.1 we show the distribution of error of each methods relative to the NLFT paragon. We see that GaF improves only marginally over the FC baseline, and is usually far from the non-linear performance. On the other hand, LQF performs comparably with NLFT on all the coarse-grained datasets. Surprisingly, training only the final classifier using LQF (LQF FC) improves over Standard FC, suggesting that the changes discussed in Section 5.1 are helpful in broader scenarios.

5.2.2 Fine-grained classification and Bilinear Pooling

Since LQF is based on a first-order Taylor expansion, it has an implicit bias toward remaining close to the pre-training (ImageNet in our case). [LCY20] notes that such a bias may improves accuracy on tasks that are easier or nearby (e.g., Stanford Dogs, Caltech-256), but it is detrimental on fine-grained tasks with a large domain gap from ImageNet (e.g., CUB-200, Aircrafts, Cars). We expect the same to hold LQF. Indeed, in Table 5.1 we show that while performance is close or better on similar tasks, it degrades for farther tasks (sorted as in [LCY20]). However, even in this case, we observe that LQF significantly outperforms other linearization techniques.

B-LQF. These results suggest that finding a pre-training closer to the target task may help improve the linearizaion performance [ALT19, NHA20, CSS18], which is indeed an intereting direction of research. On the other hand, [LRM15] suggests that even for generic ImageNet pretraining, the covariance of the last-layer features, rather than their mean, is more suited as a feature for fine-grained visual classification. Following this intuition, we replace the last layer global spatial mean pooling with square root bilinear pooling: if $z_{i,j}$ are the features of the last-layer feature map, instead of feeding their mean $\mu = \mathbb{E}_{i,j}[z_{i,j}]$ to the fully connected layer, we feed the square root $\sqrt{\Sigma}$ of their covariance matrix Σ [LRM15, LM17, GLB18]. In Table 5.2 we see that – while bilinear pooling does not improve the performace of NLFT too much – combining bilinear-pooling with LQF (B-LQF) significantly boosts its performance on fine-grained tasks.

5.2.3 Ablation study

We measure the effect of each component of LQF on its final accuracy. In Figure 7.5 we show the relative increase in error when we ablate the changes discussed in Section 5.1. Using the MSE loss instead of cross-entropy yields the largest improvement. Using K-FAC also carries significant weight in the final solution. This is interesting since, in principle, SGD without pre-conditioning should recover the same minimum. This reinforces the point that the optimization problem is badly conditioned (Figure 5.3), and proper pre-conditioning is necessary. Using Leaky-ReLU activations

	NLFT	MPN-COV	B-LQF*	LQF*	GaF
CUB-200	19.6	17.0	**17.1**	24.0	29.2
Aircrafts	14.5	15.4	**29.2**	34.5	54.2
Cars	9.6	9.1	**16.3**	27.1	47.3

Table 5.2: **Fine-grained classification and bilinear pooling.** Normalized bilinear pooling (MPN-COV) [LM17,GLB18] does not significantly improve ordinary classification results (NLFT) in our setting. However, LQF with bilinear pooling (B-LQF) significantly improves test accuracy compared to average pooling (LQF).

instead of ReLUs also improve the final results on average, even if by a lesser margin than the other two changes. However, in additional ablation studies where we ablate pairs of components, we observe that using Leaky ReLU makes a larger difference if K-FAC is not present, possibly because Leaky ReLU simplifies the optimization of the linear model and is more useful when K-FAC is disabled.

5.2.4 Low-shot data regime

LQF, like any linearization, cannot be expected to outperform NLFT in general. However, there may be scenarios where it does systematically. In the low-data regime, non-linear networks can easily memorize samples using spurious correlation without having a chance to converge to fit relevant features. On the other hand, LQF should have a strong inductive bias arising from the existence of a unique global minimum. To test this, in Figure 5.4 we plot the test error of LQF relative to the NLFT paragon. In the k-shot scenario (i.e., training using only k samples per class) LQF outperforms NLFT systematically. This suggests that LQF may be a better base model than NLFT in few-shot learning, also because it facilitates the incorporation of any convex regularizer.

5.2.5 On-line learning

In many applications, training data is not all available at the outset. Instead, it trickles in continuously and triggers re-training upon reaching a sufficiently large volume. This protocol engenders tradeoffs between cost and obsolescence. Fine-tuning with relatively small batches is a typical compromise, but can lead to suboptimal solutions when the network memorizes the new samples or catastrophically forgets old ones. LQF is not subject to this tradeoff as it has a unique minimum that is adjusted incrementally as new data is received.

In Figure 5.5, we compare incremental LQF incremental re-training with incremental NLFT training and with the paragon of re-training from scratch every time we obtain more data on the MIT-67 dataset. Let $\mathcal{D}_t$ be the dataset containing all the samples seen up to step t; we train on $\mathcal{D}_t$ starting from the weights w_{t-1} obtained at the previous step and stop training when the train error on $\mathcal{D}_t$ is below 0.5% for 5 consecutive epochs. In Figure 5.5, we split all the samples of MIT-67 in 30 incremental steps (each step containing 178 samples sampled uniformly from all classes) and show the test accuracy of the model at different steps. As expected, incremental NLFT performs substantially worse than the paragon, as it gets stuck in suboptimal local minima. On the other hand, LQF is always close to the paragon even if trained incrementally while being much cheaper to train since we do not need to retrain from scratch.

5.2.6 Informativeness of samples

LQF is interpretable in that it allows measuring the effect of individual data on the trained model, which can be used to summarize a dataset. We use eq. (5.9) to measure how informative a training sample is for the predictions of the final model. In Figure 5.6, we display the most and least informative samples: Qualitatively, we observe that easy samples and near-duplicates are not informative. On the other hand, samples of classes that are easy to confuse are considered very informative by the model, since they affect the decision boundary. To test this, in Figure 5.6 (right) we plot the test error of the model when the most/least informative training samples are

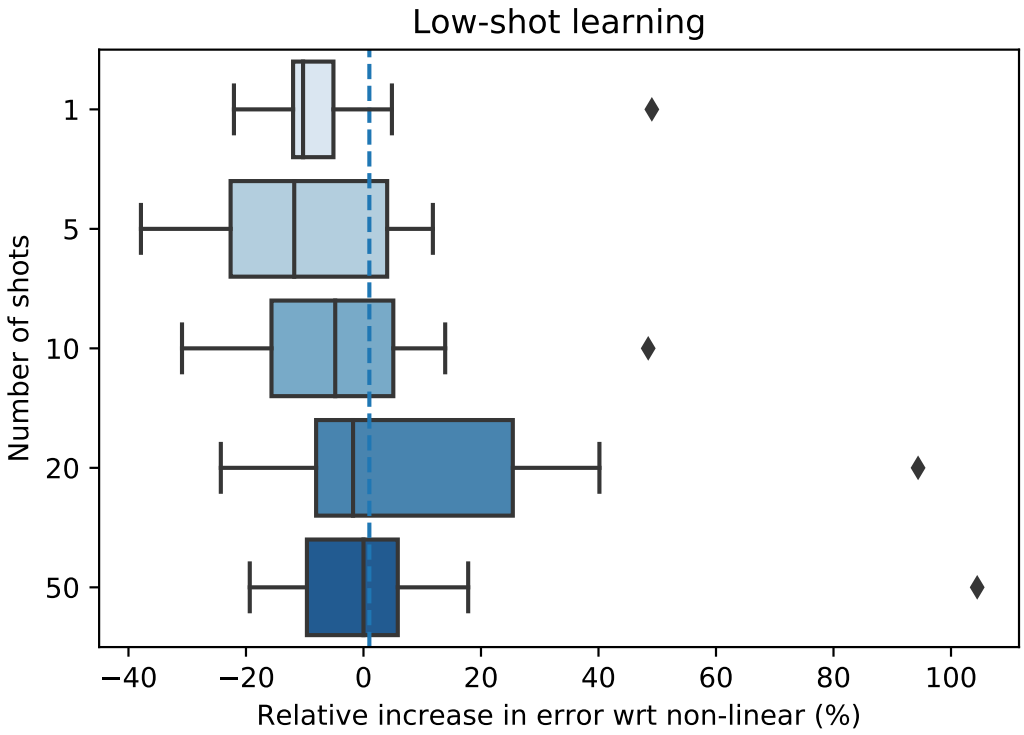

Figure 5.4: **Efficiency of LQF at different data regimes.** Comparison of LQF and standard fine-tuning for different number of training samples per category ("shots") in the same datasets as Table 5.1. On average, LQF performs better than standard fine-tuning when the datasets are relatively small. The single outlier (diamond) is FGVC Aircraft, where NLFT tends to perform better (see Section 5.2.2).

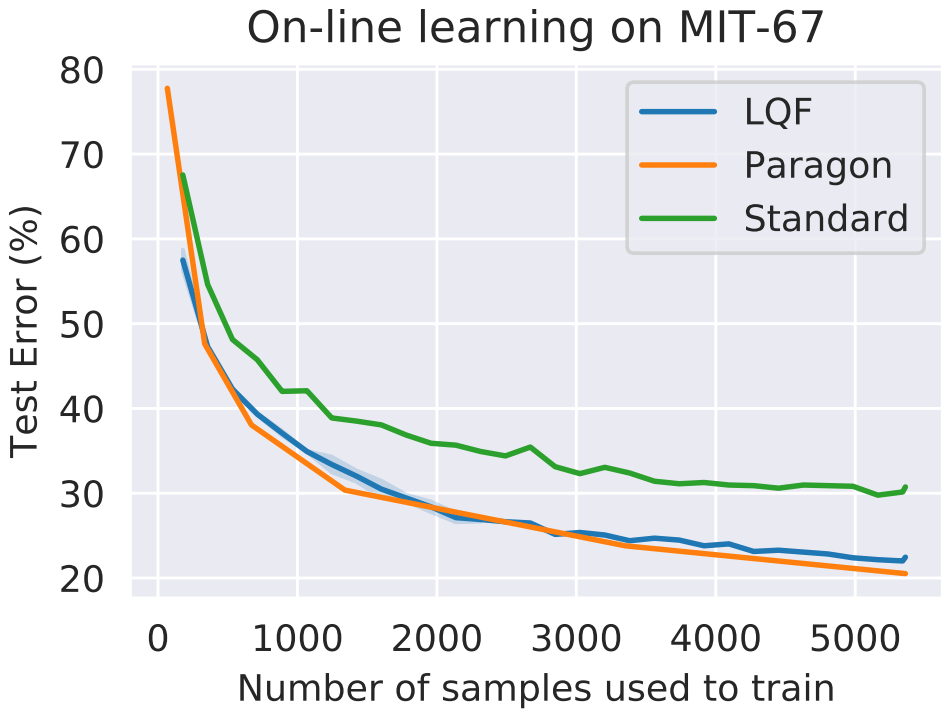

Figure 5.5: **Comparison of on-line learning methods.** Standard non-liner networks may converge to a bad local minimum when trained incrementally, and consequently have a sub-optimal performance compared to a non-linear network re-trained from scratch every time more data is obtained (paragon). On the other hand, LQF has a unique minimum to which it is guaranteed to converge, ensuring that the performance will be close to the paragon at all steps.

removed. As we expect, removing informative samples degrades the performance more than removing uninformative samples.

5.2.7 Robustness to optimization hyper-parameters

Non-linear fine-tuning is highly sensitive to the choice of hyperparameters (Figure 7.1, right). This has a direct impact on cost, as training requires broad hyper-parameter search. It would be desirable if one could fix hyperparameters once for fine-tuning. In Section 5.1.2 we argued that, due to the use of MSE loss and K-FAC pre-conditioning, the optimal learning rate is independent of the task for LQF. When using SGD with momentum, the situation is slightly more complex, as we also need to select batch size b and the momentum m. As already noted by [LCY20], when fine-tuning, different

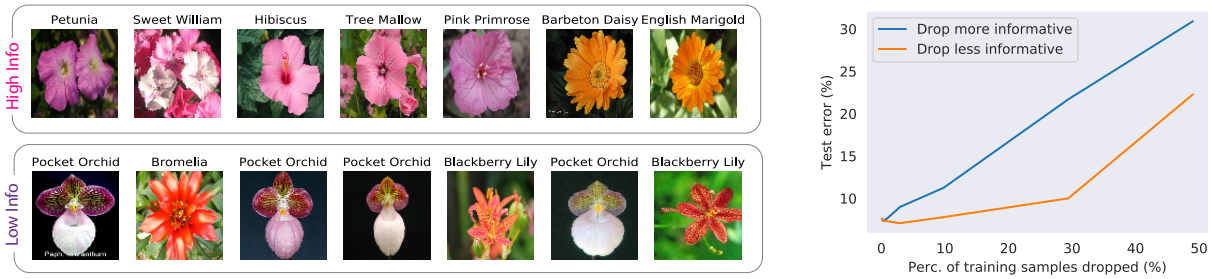

Figure 5.6: **Informativeness of samples. (Top row)** Seven representative examples from the 25 most informative images for a network trained on Oxford Flowers (complete set in appendix). The network considers more informative samples of flowers that are hard to distinguish, such as *English Marigold* and *Barbeton Daisy*, or *Hibiscus*, *Petunia* and *Pink Primrose*. On the other hand, images of flowers that have a distinctive shape (e.g., *Orchid*) or near duplicated images are not informative for the training **(bottom row)**. **(Right) Dataset summarization.** We plot the final test accuracy when training LQF after dropping the N most informative training examples (blue line) and the N less informative (orange line). The test performance decreases faster when dropping informative examples, as we would expect, validating our informativeness measure.

runs typically have the same test accuracy as long as the effective learning rate ELR $= \frac{\eta}{(1-m)b}$ is the same. We claim that, for LQF, there is a single ELR which is optimal for most tasks. To test this, we search the best learning rate $\eta \in \{0.05, 0.01, 0.005, 0.001, 0.0001\}$ and batch size $b \in \{16, 32, 64\}$, and compute the ELR of the best combination for each task. In Figure 7.1 (right) we report the distribution of effective learning rates that yield the best performance. The interval of optimal ELRs for LQF is small, so one choice of ERL is nearly optimal for all tasks. Moreover, we do not observe significant changes in test accuracy for minor variations of ELR. On the other hand, for NLFT the ELR can vary by orders of magnitude depending on the task, and test performance can vary widely within that interval. Thus, fine-grained search of the ELR is necessary for NLFT but not for LQF. This robustness comes at the cost of a slight decrease in performance (Section 5.2.1, left), as expected due to the linearization.

5.3 Discussion

Usage of deep neural networks in practice often involves fine-tuning pre-trained models followed by heavy hyper-parameter optimization (HPO). This method can achieve good performance, but even minor variations of hyper-parameters can spoil performance. The low-data regime is also treacherous, as models can easily memorize the samples provided. It is difficult for non-expert to predict the effect of regularizers and hyper-parameters beyond trying it and seeing. Linear models, on the other hand, are supported by decades of theory that allows to predict the effect of changes in the data, or in the parameters, and bounds on the generalization error. However, their performance has not been comparable to that of black-box DNNs. LQF is a linear model that gets closer, to the point of making linearization viable beyond a theoretical analysis tool, as a practical method. The small drop in performance is offset by improved ease of training, interpretability, and cheaper hyper-parameter search.

LQF is not the solution to all fine-tuning problems. Being a first-order approximation, it can realistically be expected to work well when the task on which we wish to fine-tune is sufficiently close to the task on which the model was pre-trained. In practice, this limitation may be circumvented by a "zoo" of different pretrained models, together with a selection technique to find the best initialization for the task [ALT19]. Better understanding of the distance between tasks may also help characterizing the range in which linearization can be expected to work well.

Linearization can be made more flexible with architecture changes. For example, fine-grained classification tasks are typically far from generic models pre-trained on ImageNet. Indeed we observe that LQF drops in performance in fine-grained tasks, which have been noted to be further from ImageNet [LCY20]. However, we have shown in Section 5.2.2, that for this specific case, simply replacing average pooling with bilinear pooling can stretch the performance of LQF in the fine-grained setting.

There are many possible uses for LQF, like any other linear model. Rather than providing an exhaustive list of all possibilities, we showed some illustrative examples. We leave various

applications of LQF to future work, including its use in (i) uncertainty quantification, (ii) backward compatibility [SXX20], (iii) leave-one-out cross validation without a validation set [GSL19], (iv) active/online learning [SL07], (v) counterfactual analysis in the first order, (vi) continual learning [KPR17b], (vii) selective forgetting or machine unlearning [GAS19a, GAS20, GGH19], (viii) training including privacy or fairness constraints [Dwo08, KR19], .

5.4 Appendix

In this appendix we provide additional empirical results (Section 5.4.1), more details about the training procedure and implementation used (Section 5.4.2), and we derive the expression in eq. (5.8) for the change in activations when a training sample is removed (Section 5.4.3).

5.4.1 Additional results

Preconditioning with Adam vs. K-FAC. In Section 5.2.1 we train LQF and LQF FC using K-FAC for preconditioning, instead of alternatives like Adam. K-FAC provides several advantages: the learning rate choice is easy to interpret in the case of a quadratic problem (Section 5.1.2), it easy to analyze theoretically (e.g., in eq. 5.6), and it provides better convergence guarantees that Adam. Moreover, it provides an approximation of the inverse of the Hessian which we need to compute eq. (5.8). We now test how Adam and K-FAC preconditioning compare to each other in terms of raw test error after hyper-parameter optimization. We train LQF, LQF FC and GaF with Adam on all datasets and report the results in Table 5.3. For each dataset we try different learning rates $\eta \in \{0.001, 0.0004, 0.0001\}$, weight decay $\lambda \in \{10^{-5}, 10^{-6}\}$ and augmentation schemes (central crop, random crop, random resized crop) and report the best result. We observe that after hyper-parameter optimization LQF obtains similar final accuracies when trained with Adam and K-FAC (Table 5.3). Similarly, for GaF we observe that Adam achieves errors comparable with SGD (in this case without K-FAC). However, K-FAC performs better than Adam when training only the last layer (LQF FC). This suggests that the more sophisticated pre-conditioning of K-FAC is more

	LQF	LQF FC	GaF
Caltech-256	**14.2**	17.2	16.0
Chest X-Ray	6.6	9.6	**6.1**
Malaria Cells	**4.2**	6.1	4.8
MIT-67	**20.4**	25.4	23.1
Oxford Pets	**6.7**	7.8	7.2
Fine-grained datasets			
Stanford Dogs	12.4	13.3	**11.7**
Oxford Flowers	**6.5**	10.8	13.8
CUB-200	**23.1**	29.3	28.5
Aircrafts	**33.1**	45.9	45.6
Stanford Cars	**24.0**	39.2	37.0

Table 5.3: **Test errors using Adam.** We report the test errors obtained by training different linear methods with Adam instead of SGD (in the case of LQF, we also train without K-FAC preconditioning). We note that Adam gives comparable results to SGD+K-FAC for LQF and to standalone SGD for GaF (Table 5.1), but is slighly worse than SGD+K-FAC for LQF FC, where we only optimize the last layer. The only exception where Adam improved results for LQF and GaF is *Chest X-Ray*, possibly due to the more exhaustive hyper-parameter search we used for Adam.

important to ensure good convergence when optimizing the lower dimensional, badly conditioned problem of LQF FC.

MSE loss for non-linear fine-tuning. In Figure 5.8 we train a standard non-linear network using cross-entropy loss and MSE loss with different number of training samples. We train with learning rate $\eta \in \{0.1, 0.05, 0.01, 0.001, 0.0001\}$ and weight decay $\lambda \in \{0.0001, 0.00001\}$ and report the best result. We observe that the MSE loss tends to outperform cross-entropy in the low-data regime, suggesting that some of the benefits of the MSE loss also apply to non-linear fine-tuning.

Ablation study for on-line learning. In addition to Figure 5.5, in Figure 5.7 we show the result

of using LQF with CE loss instead of MSE, ReLU instead of Leaky ReLU or not using K-FAC. This shows that all components contribute to better on-line performance, but their contribution is relatively minor with respect to the difference between using LQF (solving a convex problem) or NLFT (non-convex).

Tuning weight-decay. We did not observe major differences in accuracy by tuning the weight decay parameter suggesting that the inductive bias of linearization is enough to grant good generalization.

However, if required, we note that LQF provides a way to tune weight decay efficiently. Since there the LQF loss function is strongly convex, it has a unique global minimum. This implies that, after training with a given value of weight decay, we can increase the value of weight decay and fine-tune the previous solution to converge to the new global minimum. This is in contrast with DNNs, which may remain stuck in a suboptimal local minimum if weight decay is changed after convergence [GAS19b]. In Figure 5.9 we show the test accuracy obtained on Caltech-256 when training from scratch with different values $\lambda \in \{10^{-3}, 5 \cdot 10^{-4}, 10^{-4}, 5 \cdot 10^{-5}, 10^{-5}\}$ of weight decay. We then load the solution obtained with $\lambda_0 = 5 \cdot 10^{-5}$ and fine-tune it with different values λ of weight decay and compare this with the results obtained training from scratch. We observe that the two approaches (training from scratch and fine-tuning) obtain similar errors, suggesting that indeed for LQF we cab use a cheaper hyper-parameter search based on fine-tuning.

We can also go a step further and automatically optimize weight decay using a validation set. Recall from eq. (5.4) that the weights at convergence as a function of the weight decay λ can be written as:

$$w^*(\lambda) = (J^T J + \lambda I)^{-1} J(Y - f_0(X)). \tag{5.10}$$

In particular, $w^*(\lambda)$ is a differentiable function of λ, so we expect to be able to optimize λ using gradient descent. In order to do that, consider the validation loss:

$$L_{\text{val}}^{\text{MSE}}(w^*(\lambda)) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x,y)\in\mathcal{D}_{\text{val}}} \|y - f_0(x) - \nabla_w f_0(x) w^*(\lambda)\|^2. \tag{5.11}$$

We want to find the value of λ that minimizes L_{val}. The gradient with respect to λ is given by

$$\partial_\lambda L_{\text{val}} = -w^* \cdot \underbrace{(F + \lambda I)^{-1} \nabla_w L_{\text{val}}}_{\text{preconditioned gradient of val. set}} \tag{5.12}$$

Note that the second term of $\partial_\lambda L_{\text{val}}$ is simply the pre-conditioned gradient of the validation loss, which can approximated using K-FAC. We leave further exploration of this research direction to future work.

5.4.2 Experimental details

Choice of datasets. We have compared the algorithms on a several datasets (Table 5.1). Most of the datasets are standard in the fine-tuning or fine-grained classification literature [LCY20, LGD18, LRM15]. We have added additional datasets to this list (*Chest X-ray, Malaria Cells*) so that we could compare on a different domain (medical images instead of natural images). Some of the datasets we use do not have a standard train/test split. In those cases, we use the following splits when reporting the results: For Oxford Flowers we do not merge training and validation data, we only use the training images (1020 samples, instead of the 2040 of train+val). For Caltech-256, we randomly sample 60 images per class to train and test on the remaining images (this scheme is sometimes called Caltech-256-60 in the literature). For Malaria Cell, we randomly select 75% of samples for training, and test of the remaining ones.

Pre-training. In all our experiments we use a ResNet-50 backbone pretrained on ImageNet using SGD (we use the reference PyTorch pretraining scheme: 90 epochs with cross-entropy loss, learning rate 0.1 decayed by a factor 10 every 30 epochs, momentum 0.9). We use Leaky ReLU activations for LQF, while we use standard ReLU activations for all other methods. To ensure that the comparison is as fair as possible, we obtain the weights of the Leaky ReLU network by starting from the pretrained ReLU network, changing the activations, and then fine-tune on ImageNet with SGD for another epoch.[2] We also tried training the backbone from scratch on ImageNet using

[2]Interestingly, we observe that even without fine-tuning the weights learned for ReLU activations still achieve a

Leaky ReLU activations, but did not observe any major difference in performance between the two.

Optimization. In all experiments, we train using SGD with momentum 0.9 and batch size 28. We search for the learning rate in $\eta \in \{0.01, 0.001\}$ and weight decay $\lambda \in \{10^{-4}, 10^{-5}\}$. We report the best result. Instead of applying weight decay directly to the gradient update as often done, we add the ℓ_2 weight penalty to the loss function. This is required when using K-FAC to compute the correct pre-conditioned update, but it does not otherwise change the udpate equation when not using K-FAC.

In Section 5.4.1 we present additional results using Adam. In this case we search for the learning rate in $\eta \in \{0.001, 0.0004, 0.0001\}$ and weight decay $\lambda \in \{10^{-5}, 10^{-6}\}$. When training GaF with Adam we use $\beta_1 = 0.5$ as suggested in [MLL20], otherwise we use $\beta_1 = 0.9$.

Data augmentation. Since different tasks may require different types of data augmentation, we train with different augmentation schemes (center crop, random crop, resized random crop) and report the best result. For *center crop*, we resize the image to 256×256 and extract the central crop of size 224×224, while for *random crop* we extract a 224×224 in a random position and also apply a random horizontal flip. In both cases we test with center crops. We also try the *resized random crops* augmentation commonly used for ImageNet pre-training. We found this augmentation to be too strong for some tasks, and it gives significantly better results only in the case of *Chest X-Rays*. For this reason, and to save computation time, we only train random crop with one combination of hyper-parameters (learning rate $\eta = 0.001$ and weight decay $\lambda = 10^{-5}$).

Linearization. We follow the procedure of [MLL20] to linearize the network. The main difference is that we do not fuse the batch norm layer with the previous convolutional layer, but rather we also linearize the batch norm layer. We did not observe major difference in performance by using one or the other solution, but we opted for the latter to keep the structure of the linearized and original network as close as possible when performing the comparison. In order to ensure linearity, for all

---

good test error when used with Leaky ReLU activations. This makes fine-tuning with the new activations particularly easy.

linear models (LQF, LQF FC, GaF, Standard FC) we put batch normalization in eval mode when training (that is, we use the (frozen) running mean and variance to whithen the features rather than using the current mini-batch statistics).

B-LQF. The only change necessary for B-LQF with respect to stnadard LQF is to replace the global average pooling before the classification layer with a linearized version of bilinear pooling with square root normalization [LM17]. Let $z \in \mathbb{R}^{HW \times C}$ be the last convolutional layer features, where H and W are the spatial dimensions and C is the number of channels. Their covariance matrix $\Sigma \in \mathbb{R}^{C \times C}$ can be written as:

$$\Sigma = \frac{1}{N} z^T (I - \frac{1}{N} \mathbb{1}) z = z^T A z \tag{5.13}$$

where I denotes the identity matrix and $\mathbb{1}$ denotes the matrix of all ones and $N = H \cdot W$. Using the same notation as [MLL20], let $h(z(r)) = \sqrt{\Sigma} = \sqrt{z^T(r) A z(r)}$. To compute the linearization forward pass of the layer we need to compute:

$$\partial_r h(z(r)) = \frac{1}{2} \sqrt{\Sigma}^{-1} (\partial_r z^T A z + z^T A \partial_r z). \tag{5.14}$$

Note that we already have $\sqrt{\Sigma}$ from the forward pass of the base model, so computing the forward pass of the linearized model does not add much to the complexity.

Robustness to optimization hyper-parameters. As mentioned in Section 5.2.7, to obtain the plot in Figure 7.1 (right) we train LQF and NLFT with SGD with $\eta \in \{0.05, 0.01, 0.005, 0.001, 0.0001\}$, batch size $b \in \{16, 32, 64\}$ and weight decay $\lambda \in \{10^{-5}, 10^{-4}, 5 \cdot 10^{-4}\}$ and we report the best result. Due to the larger search space, we report the results only for a representative subset of the datasets: *Oxford Flowers, MIT-67,CUB-200, Caltech-256, Stanford Dogs, FGVC Aircrafts*.

5.4.3 Derivation of interpretability (eq. 5.8)

Recall that in the case of LQF the hessian of the MSE loss (eq. (5.3)) is given by $H = F + \lambda I$ where $F = \frac{1}{N} J J^T = \frac{1}{N} \sum_{j=1}^{N} g_j^T g_j$, where $g_j = \nabla_w f_0(x_j)$ are the Jacobian of the i-th sample computed at

the linearization point w_0. To keep the notation simple, assume this a binary classification problem, so that the label $y \in \{0, 1\}$ is a scalar and the Jacobian g_i is a row vector (a similar derivation holds for a multi-class problem).

Let w^* be the optimum of the loss function computed using N training samples. Since in a quadratic problem a Newton-update converges to the optimum in a single step, we can write the new optimal weights obtained after removing the i-th training sample – that is, training with only $N - 1$ samples – as:

$$w^*_{-i} = w^* - H^{-1}_{-i} \nabla_w L_{-i}(w^*) \tag{5.15}$$

where L_{-i} and H_{-i} are respectively the training loss computed without the sample i and H_{-i} is its hessian. Note that we can write $L_{-i}(w)$ as

$$L_{-i}(w) = L(w) - \frac{1}{N} \|y_i - f_0(x_i) - g_i w\|^2.$$

Since w^* is the optimum of the original problem, we have $\nabla L(w^*) = 0$. Using this and the previous equation we get:

$$\nabla_w L_{-i}(w^*) = \nabla_w L(w^*) - g_i^T e_i = -g_i^T e_i$$

where $e_i \equiv \frac{1}{N}(y - g_i w^*)\cdot$ is the (weighted) prediction error on the sample x_i. Plugging this in eq. (5.15) we have

$$w^*_{-i} = w^* + H^{-1}_{-i} g_i^T e_i.$$

We now derive an expression for H^{-1}_{-i} as a function of F. Note that

$$H_{-i} = F_{-i} + \lambda I = \frac{N}{N-1} F + \lambda I - \frac{1}{N-1} g_i^T g_i.$$

Let $A = \frac{N}{N-1} F + \lambda I$. Note that H_{-i} is a rank-1 update of A. Using the Sherman–Morrison formula for the inverse of a rank-1 update (or, more generally, the Woodbury identity in the case of a multi-class problem), we have

$$H^{-1}_{-i} = (A - \frac{1}{N-1} g_i^T g_i)^{-1} = A^{-1} - \frac{A^{-1} g_i^T g_i A^{-1}}{N - 1 + g_i A^{-1} g_i^T}$$

The activation change $\Delta f(x_{\text{test}}) = f_{w^*}(x_{\text{test}}) - f_{w^*_{-1}}(x_{\text{test}})$ after removing a training sample is then given by:

$$\begin{aligned}\Delta f(x_{\text{test}}) &= g_{\text{test}}(w^* - w^*_{-i}) \\ &= g_{\text{test}} H_{-i}^{-1} g_i^T e_i \\ &= e_i g_{\text{test}} A^{-1} g_i - \frac{e_i g_{\text{test}} A^{-1} g_i^T g_i A^{-1} g_i}{N - 1 + g_i A^{-1} g_i^T}\end{aligned}$$

Reorganizing the terms, we obtain the following expression for eq. (5.8) in the main paper:

$$\boxed{\Delta f(x_{\text{test}}) = (1 - \frac{\alpha_i}{N - 1 + \alpha_i}) e_i g_{\text{test}} A^{-1} g_i^T}$$

where we defined $\alpha_i = g_i A^{-1} g_i^T$. In particular, note that for large datasets ($N \gg 1$) the change in activations is simply given by $e_i g_{\text{test}} A^{-1} g_i^T$ which measures the similarity of the Jacobian of the sample x_{test} and the jacobian of x_i under the metric induced by the kernel A^{-1}. Finally, note that A^{-1} is the inverse of the Fisher Information Matrix F plus a multiple of the identity, which we can easily estimate using K-FAC [MG15]. This is particularly convenient since we are already computing the K-FAC approximation to pre-condition the gradients.

In the case of a multiclass problem, the Jacobian g_i is a $C \times D$ matrix, where C is the number of classes and D is the number of parameters. In this case, we get a similar expression:

$$H_{-i}^{-1} = A^{-1} - A^{-1} g_i^T ((N-1) I_c + g_i A^{-1} g_i^T)^{-1} g_i A^{-1}$$

The activation change $\Delta f(x_{\text{test}}) = f_{w^*}(x_{\text{test}}) - f_{w^*_{-1}}(x_{\text{test}})$ after removing a training sample is then given by:

$$\begin{aligned}\Delta f(x_{\text{test}}) =& g_{\text{test}}(w^* - w^*_{-i}) \\ =& g_{\text{test}} H_{-i}^{-1} g_i^T e_i \\ =& g_{\text{test}} A^{-1} g_i^T e_i \\ & - g_{\text{test}} A^{-1} g_i^T ((N-1) I_c + g_i A^{-1} g_i^T)^{-1} g_i A^{-1} g_i^T e_i\end{aligned}$$

Reorganizing the terms, we have:

$$\Delta f(x_{\text{test}}) = g_{\text{test}} A^{-1} g_i^T (I_C - M^{-1}\alpha_i) e_i$$

where $M = (N-1)I_c + \alpha_i$, $\alpha_i = g_i A^{-1} g_i^T$ is now a $C \times C$ matrix and I_C denotes the $C \times C$ identity.

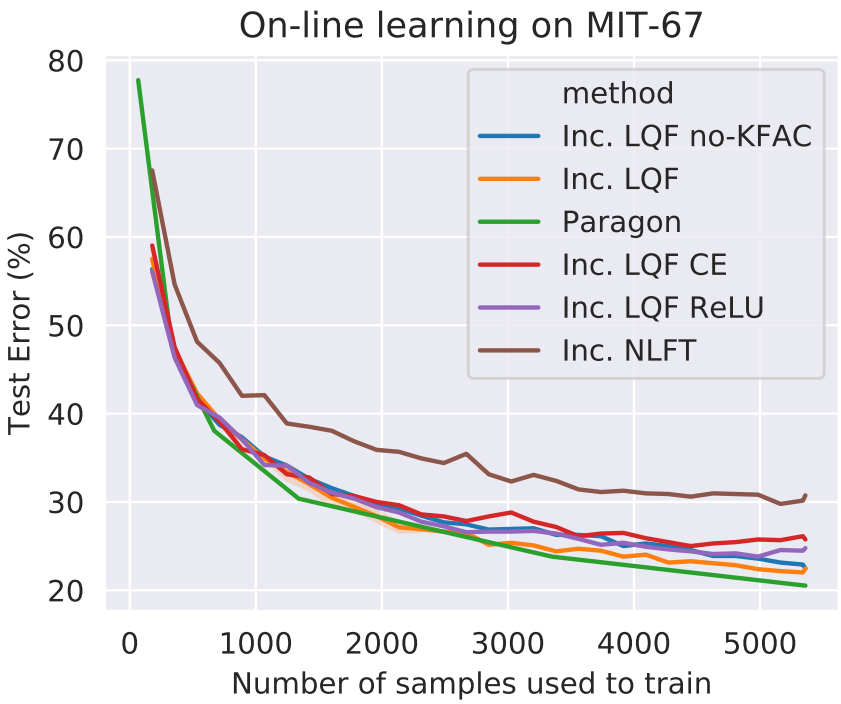

Figure 5.7: **Ablation study on on-line learning.**

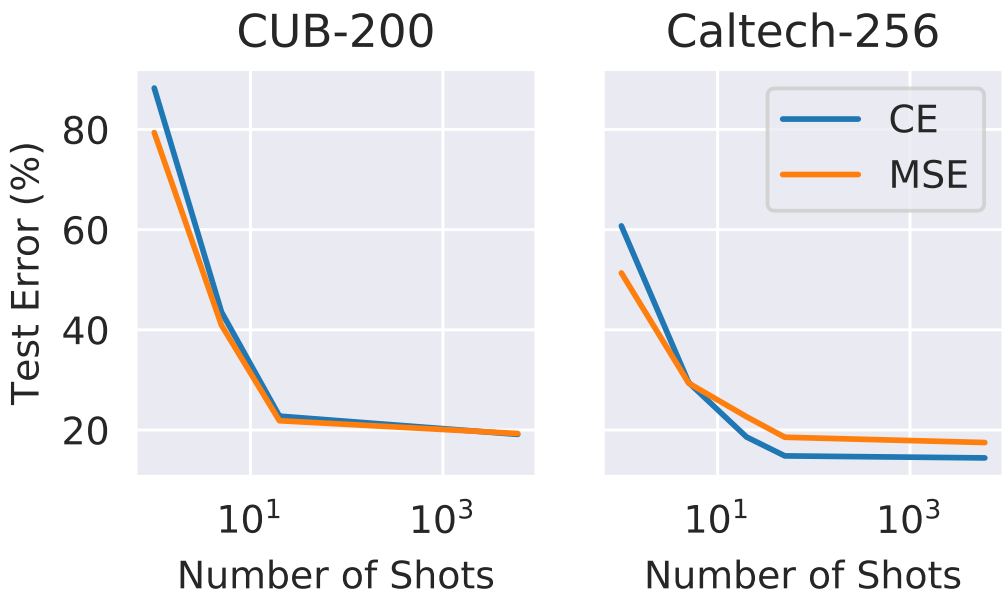

Figure 5.8: **Using MSE loss with NLFT.** Plot of the test error obtained by training with NLFT using either cross-entropy of MSE loss for different number of training samples per class (shots). While cross-entropy loss is comparably or better than MSE loss when training with many samples, we observe that in the low-shot regime MSE tends to always outperform CE. This suggests that using MSE loss is not beneficial only for lienarized models.

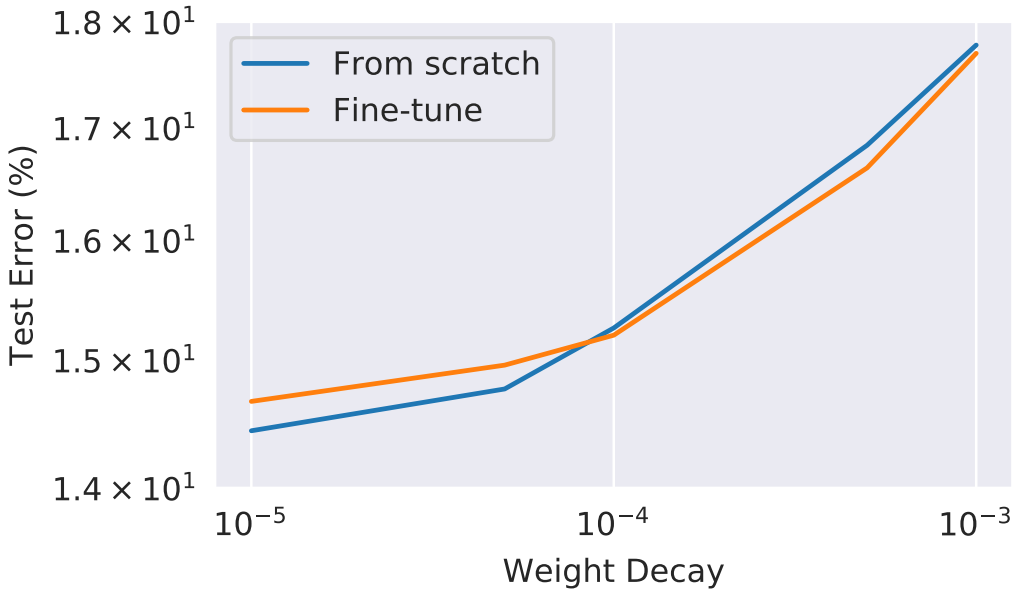

Figure 5.9: **Exploring different weight decay values via fine-tuning.** We compare the test error obtained by training from scratch with a given weight decay value λ, and the test error obtained by loading the solution found with $\lambda_0 = 5 \cdot 10^{-4}$ and fine-tuning with a different value of λ.

CHAPTER 6

Mixed-Privacy Forgetting in Deep Neural Networks

Building on the work from the previous chapters, in this chapter we show that the influence of a subset of the training samples can be removed – or "forgotten" – from the weights of a network trained on large-scale image classification tasks, and we provide strong computable bounds on the amount of remaining information after forgetting. Inspired by real-world applications of forgetting techniques, we introduce a novel notion of forgetting in mixed-privacy setting, where we know that a "core" subset of the training samples does not need to be forgotten. While this variation of the problem is conceptually simple, we show that working in this setting significantly improves the accuracy and guarantees of forgetting methods applied to vision classification tasks. Moreover, our method allows efficient removal of all information contained in non-core data by simply setting to zero a subset of the weights with minimal loss in performance. We achieve these results by replacing a standard deep network with a suitable linear approximation. With opportune changes to the network architecture and training procedure, we show that such linear approximation achieves comparable performance to the original network and that the forgetting problem becomes quadratic and can be solved efficiently even for large models. Unlike previous forgetting methods on deep networks, ours can achieve close to the state-of-the-art accuracy on large scale vision tasks. In particular, we show that our method allows forgetting without having to trade off the model accuracy. In this chapter we provide a method to overcome the limitations of chapter 3, here the model is linear, the optimization problem is quadratic and as a result the unlearning procedure is exact and optimal. Also, since the NTK approximation is valid in setting where the width of the network limits to infinity we were restricted with smaller datasets in chapter 3, while in this chapter since we train a simple linear model (Taylor series approximation) without the NTK approximation, we are

able to depict the efficacy of our method on large-scale practical tasks.

In the subsequent sections, we describe our forgetting procedure using mixed linear models, provide a rigorous theoretical bound for the amount of information the weights contain about the removed training samples, and provide extensive empirical analysis showing the robustness of our procedure to different read out functions.

6.1 Preliminaries and Notations

We use the empirical risk minimization (ERM) framework throughout this paper for training. Let $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$ be a dataset where $\mathbf{x}_i \in \mathcal{X}$ denotes the input datum (for example, images) and $\mathbf{y}_i \in \mathcal{Y}$ the corresponding output (for example, one hot vector in classification). Given an input image $\mathbf{x}$, let $f_{\mathbf{w}}(\mathbf{x}) : \mathcal{X} \times \mathbb{R}^d \to \mathcal{Y}$ (for instance, a DNN) be a function parameterized by $\mathbf{w} \in \mathbb{R}^d$ used to model the relation $\mathcal{X} \to \mathcal{Y}$. Given a input-target pair $(\mathbf{x}, \mathbf{y}) \in (\mathcal{X}, \mathcal{Y})$, we denote empirical risk or the training loss for $(\mathbf{x}, \mathbf{y})$ by $\ell(f_{\mathbf{w}}(\mathbf{x}), \mathbf{y}) : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^d \to \mathbb{R}^+$. We will sometimes abuse notation and use $\ell(\mathbf{w})$ by dropping $\mathbf{x}, \mathbf{y}$. For a training dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n \subset (\mathcal{X}, \mathcal{Y})^n$, we denote the empirical risk/total training loss on $\mathcal{D}$ by $L_{\mathcal{D}}(\mathbf{w}) \triangleq \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \ell(f_{\mathbf{w}}(\mathbf{x}_i), \mathbf{y}_i)$. We will interchangebly use $L_{\mathcal{D}}(f_{\mathbf{w}})$ with $L_{\mathcal{D}}(\mathbf{w})$. Let $\mathcal{A}_\tau(L_{\mathcal{D}}(\mathbf{w}_0)) : (\mathcal{X}, \mathcal{Y})^n \times \mathbb{R}^d \times \mathbb{N} \to \mathbb{R}^d$, denote the weights obtained after τ steps of a training algorithm $\mathcal{A}$ using $\mathbf{w}_0$ as the initialization (for examples, SGD in our case). We denote with $\|\mathbf{w}\|$ the L_2 norm of a vector $\mathbf{w}$ and with $\lambda_{\text{Max}}(Q)$ the largest eigenvalue of a matrix Q. To keep the notation uncluttered, we also use the shorthand $\nabla_{\mathbf{w}} L(\mathbf{w}') \triangleq \nabla_{\mathbf{w}} L(\mathbf{w})|_{\mathbf{w}=\mathbf{w}'}$.

6.2 The Forgetting Problem

The weights $\mathbf{w}$ of a trained deep network $f_{\mathbf{w}}(\mathbf{x})$ are a (possibly stochastic) function of the training data $\mathcal{D}$. As such, they may retain information about the training samples which an attacker[1] can

[1]An attacker is any agent intent to extract information about the data used for training.

extract from knowledge of the weights or outputs at inference time. A forgetting procedure is a function $S(\mathbf{w}, \mathcal{D}, \mathcal{D}_F)$[2] (also called scrubbing function) which, given a set of weights $\mathbf{w}$ trained on $\mathcal{D}$ and a subset $\mathcal{D}_f \subset \mathcal{D}$ of images to forget, outputs a new set of weights $\mathbf{w}'$ which are indistinguishable from weights obtained by training without $\mathcal{D}_f$.

Readout functions. The success of the forgetting procedure, can be measured by looking at whether a discriminator function $R(\mathbf{w})$ that can guess – at better than chance probability – whether a set of weights $\mathbf{w}$ was trained with or without $\mathcal{D}_f$ or whether it was trained with $\mathcal{D}_f$ and then scrubbed. Following [GAS19a, GAS20] we call such functions *readout functions*. A popular example of readout function is the confidence of the network (that is, the entropy of the output softmax vector) on the samples in $\mathcal{D}_f$: Since networks tend to be overconfident on their training data [GPS17, KHH20], a higher than expected confidence may indicate that the network was indeed trained on $\mathcal{D}_f$. We discuss more read-out functions in Section 6.6.1. Alternatively, we can measure the success of the forgetting procedure by measuring the amount of remaining mutual information $\mathcal{I}(S(\mathbf{w}); \mathcal{D}_f)$[3] between the scrubbed weights $S(\mathbf{w})$ and the data $\mathcal{D}_f$ to be forgotten. While this is more difficult to estimate, it can be shown that $\mathcal{I}(S(\mathbf{w}); \mathcal{D}_f)$ upper-bounds the amount of information that any read-out function can extract [GAS19a, GAS20]. That is, it is an upper-bound on the amount of information that an attacker can extract about $\mathcal{D}_f$ using the scrubbed weights $S(\mathbf{w})$.

Quadratic forgetting. An important example is forgetting in a linear regression problem, which has a quadratic loss function $L_{\mathcal{D}}(\mathbf{w}) = \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}} \|\mathbf{y}_i - \mathbf{w}^T \mathbf{x}_i\|^2$. Given the weights $\mathbf{w} = \mathcal{A}_\tau(L_{\mathcal{D}}(\mathbf{w}))$ obtained after training on $L_{\mathcal{D}}(\mathbf{w})$ using algorithm $\mathcal{A}$, the optimal forgetting function is given by:

$$\mathbf{w} \mapsto \mathbf{w} - H_{\mathcal{D}_r}^{-1} \nabla_{\mathbf{w}} L_{\mathcal{D}_r}(\mathbf{w}), \tag{6.1}$$

where $H_{\mathcal{D}_r}^{-1}, \nabla_{\mathbf{w}} L_{\mathcal{D}_f}(\mathbf{w})$ is the hessian and gradient of the loss function computed on the remaining

[2]We will abuse the notation and write $S(\mathbf{w})$ when its arguments are clear from the context.

[3]$\mathcal{I}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{P_{\mathbf{x},\mathbf{y}}}[\log(P_{\mathbf{x},\mathbf{y}} / P_{\mathbf{x}} P_{\mathbf{y}})]$ is the mutual information between $\mathbf{x}$ and $\mathbf{y}$, where $P_{\mathbf{x},\mathbf{y}}$ is the joint distribution and $P_{\mathbf{x}}, P_{\mathbf{y}}$ are the marginal distributions.

data respectively. When $\mathcal{A}_\tau(L_\mathcal{D}(\mathbf{w})) = \mathbf{w}^* \triangleq \text{argmin}_\mathbf{w} L_\mathcal{D}(\mathbf{w})$, we can replace $L_{\mathcal{D}_r}(\mathbf{w}^*)$ with $-L_{\mathcal{D}_f}(\mathbf{w}^*)$ in which case it can be interpreted as a reverse Newton-step that unlearns the data $\mathcal{D}_f$ [GGH19, GAS19a]. Since, as we will see later, the "user weights" of ML-Forgetting minimize a similar quadratic loss function, eq. (6.1) also describes the optimal forgetting procedure for our model. The main challenge for us will be how to accurately compute the forgetting step since the Hessian matrix can't be computed or stored in memory due to the high-number of parameters of a deep network (Section 6.4).

Convex forgetting. Unfortunately, for more general machine learning models we do not have a close form expression for the optimal forgetting step. However, it can be shown [KL17] that eq. (6.1) is always a first-order approximation of the optimal forgetting. [GGH19] shows that for strongly convex Lipschitz loss functions, the discrepancy between eq. (6.1) and the optimal forgetting is bounded. Since this discrepancy – even if bounded – can leak information, a possible solution is to add a small amount of noise after forgetting:

$$\mathbf{w} \mapsto \mathbf{w} + H^{-1}_{\mathcal{D}_r}(\mathbf{w}) \nabla_\mathbf{w} L_{\mathcal{D}_f}(\mathbf{w}) + \sigma^2 \epsilon, \tag{6.2}$$

where $\epsilon \sim N(0, I)$ is a vector of random Gaussian noise, which aims to destroy any information that may leak due to small discrepancies. Increasing the variance σ of the noise destroys more information, thus making forgetting more secure, but also reduces the accuracy of the model since the weights are increasingly random. The curve of possible Pareto-optimal trade-offs between accuracy and forgetting can be formalized with the Forgetting Lagrangian [GAS19a].

Alternatively, to forget data in a strongly convex problem, one can fine-tune the weights on the remaining data using perturbed projected-GD [NRS20]. Since projected-GD converges to the unique minimum of a strongly convex function regardless of the initial condition (contrary to SGD, which may not converge unless proper learning rate scheduling is used), this is guaranteed to remove all influence of the initial data [NRS20]. The downside is that gradient descent (GD) is impractical for large-scale deep learning applications compared to SGD, projection based algorithms are not popular in practice, and the commonly used loss functions are not generally Lipschitz.

Non-convex forgetting. Due to their highly non-convex loss-landscape, small changes of the training data can cause large changes in the final weights of a deep network. This makes application of eq. (6.2) challenging. [GAS19a] shows that pre-training helps increasing the stability of SGD and derives a similar expression to eq. (6.2) for DNNs, and also provides a way to upper-bound the amount of remaining information in a DNN. [GAS19a] builds on recent results in linear approximation of DNNs, and approximate the training path of a DNN with that of its linear approximation. While this improves the forgetting results, the approximation is still not good enough to remove all the information. Moreover, computing the forgetting step scales quadratically with the number of training samples and classes, which restricts the applicability of the algorithm to smaller datasets.

6.3 Mixed-Linear Forgetting

Let $f_{\mathbf{w}}(\mathbf{x})$ be the output of a deep network model with weights $\mathbf{w}$ computed on an input image $\mathbf{x}$. For ease of notation, assume that the core dataset and the user dataset share the same output space (for example, the same set of classes, for a classification problem). After training a set of weights $\mathbf{w}_c$ on a core-dataset $\mathcal{D}_c$ we would like to further perturb those weights to fine-tune the network on user data $\mathcal{D}$. We can think of this as solving the two minimization problems:

$$\mathbf{w}_c^* = \arg\min_{\mathbf{w}_c} L_{\mathcal{D}_c}(f_{\mathbf{w}_c}) \tag{6.3}$$

$$\mathbf{w}_u^* = \arg\min_{\mathbf{w}_u} L_{\mathcal{D}}\big(f_{\mathbf{w}_c^*+\mathbf{w}_u}\big) \tag{6.4}$$

where we can think of the user weights $\mathbf{w}_u$ a perturbation to the core weights that adapts them to the user task. However, since the deep network $f_{\mathbf{w}}$ is not a linear function in of the weights $\mathbf{w}$, the loss function $L_{\mathcal{D}}\big(f_{\mathbf{w}_c^*+\mathbf{w}_u}\big)$ can be highly non-convex. As discussed in the previous section, this makes forgetting difficult. However, if the perturbation $\mathbf{w}_u$ is small, we can hope for a linear approximation of the DNN around $\mathbf{w}_c^*$ to have a similar performance to fine-tuning the whole network [MLL20], while at the same time granting us easiness of forgetting.

Motivated, by this, we introduce the following model, which we call Mixed-Linear Forgetting model (ML-model):

$$f^{\text{ML}}_{\mathbf{w}^*_c,\mathbf{w}_u}(\mathbf{x}) \triangleq \underbrace{f_{\mathbf{w}^*_c}(x)}_{\text{trained on } \mathcal{D}_c} + \overbrace{\nabla_{\mathbf{w}} f_{\mathbf{w}^*_c}(\mathbf{x}) \cdot \mathbf{w}_u}^{\text{train on } \mathcal{D}}. \tag{6.5}$$

The model $f^{\text{ML}}_{\mathbf{w}^*_c,\mathbf{w}_u}(\mathbf{x})$ can be seen as first-order Taylor approximation of the effect of fine-tuning the original deep network $f_{\mathbf{w}^*_c+\mathbf{w}_u}(\mathbf{x})$. It has two sets of weights, a set of non-linear core weights $\mathbf{w}_c$, which enters the model through the non-linear network $f_{\mathbf{w}_c}(\mathbf{x})$, and a set linear user-weights $\mathbf{w}_u$ which enters the model linearly. Even though the model is linear in $\mathbf{w}_u$, it is still a highly non-linear function of $\mathbf{x}$ due to the non-linear activations in $\nabla_{\mathbf{w}} f(\mathbf{w}^*_c)$.

We train the model solving two separate minimization problems:

$$\mathbf{w}^*_c = \arg\min_{\mathbf{w}_c} L^{\text{CE}}_{\mathcal{D}_c}(f_{\mathbf{w}_c}), \tag{6.6}$$

$$\mathbf{w}^*_u = \arg\min_{\mathbf{w}_u} L^{\text{MSE}}_{\mathcal{D}}\big(f^{\text{ML}}_{\mathbf{w}^*_c+\mathbf{w}_u}\big). \tag{6.7}$$

Eq. (6.6) is akin to pretraining the weights $\mathbf{w}_c$ on the core dataset $\mathcal{D}_c$, while eq. (6.7) fine-tunes the linear weights on all the data $\mathcal{D}$. This ensures the weights $\mathbf{w}_c$ will only contain information about the core dataset $\mathcal{D}_c$, while all information about the user data $\mathcal{D}$ is contained in $\mathbf{w}_u$. Also note that we introduce two separate loss functions for the core and user data. To train the user weights we use a mean square error (i.e., L_2) loss [HB21, MNS20, GDN13]:

$$L^{\text{MSE}}_{\mathcal{D}}(\mathbf{w}_u) = \frac{1}{2|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \big\| f_{\mathbf{w}^*_c}(\mathbf{x}) + \nabla_{\mathbf{w}} f_{\mathbf{w}^*_c}(\mathbf{x}) \cdot \mathbf{w}_u - \mathbf{y} \big\|^2 + \frac{\mu}{2}\|\mathbf{w}_u\|^2 \tag{6.8}$$

where $\mathbf{y}$ is a one-hot encoding of the class label. This loss has the advantage that the weights $\mathbf{w}_u$ are the solution to a quadratic problem, in which case the optimal forgetting step can be written in closed form (see eq. 6.1). On the other hand, since we do not need to remove any information from the the weights $\mathbf{w}_c$, we can train them using any loss in eq. (6.3). We pick the standard cross-entropy loss, although this choice is not fundamental for our method.

6.3.1 Optimizing the Mixed-Linear model

Ideally, we want the ML-model to have a similar accuracy on the user data to a standard non-linear network. At the same time, we want the ML-model to perform significantly better than simply training a linear classifier on top of the last layer features of $f_{\mathbf{w}_c^*}$, which is the trivial baseline method to train a linear model for an object classification task. In Figure 6.1 (see Section 7.2 for details) we see that this is indeed the case: while linear, the ML-model is still flexible enough to fit the data with a comparable accuracy to the fully non-linear model (DNN). However, some considerations are in order regarding how to train our ML-model.

Training the core model. Eq. (6.3) reduces to the standard training of a DNN on the dataset $\mathcal{D}_c$ using cross-entropy loss. We train using SGD with annealing learnig rate. In case $\mathcal{D}_c$ is composed of multiple datasets, for example ImageNet and a second dataset closer to the user task, we first pretrain on ImageNet, then fine-tune on the other dataset.

Training the Mixed-Linear model. Training the linear weights of the Mixed-Linear model in eq. (6.4) is slighly more involved, since we need to compute the Jacobian-Vector product (JVP) of $\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x}) \cdot \mathbf{w}_u$. While a naïve implementation would require a separate backward pass for each sample, [MLL20, Pea94] show that the JVP of a batch of samples can be computed easily for deep networks using a slightly modified forward pass. The modified forward pass has only double the computational cost of a standard forward pass, and can be further reduced by linearizing only the final layers of the network. Using the algorithm of [MLL20] to compute the model output, eq. (6.4) reduces to a standard optimization, which we perform again with SGD with annealing learning rate. Note that, since the problem is quadratic, we could use more powerful quasi-Netwon methods to optimize, however we avoid that to keep the analysis simpler, since optimization speed is not the focus of this paper.

Architecture changes. We observe that a straightforward application of [MLL20] to a standard pre-trained ResNet-50 tend to under-perform in our setting (fine-tuning on large scale vision tasks). In particular, it achieves only slightly better performance than training a linear classifier on top of

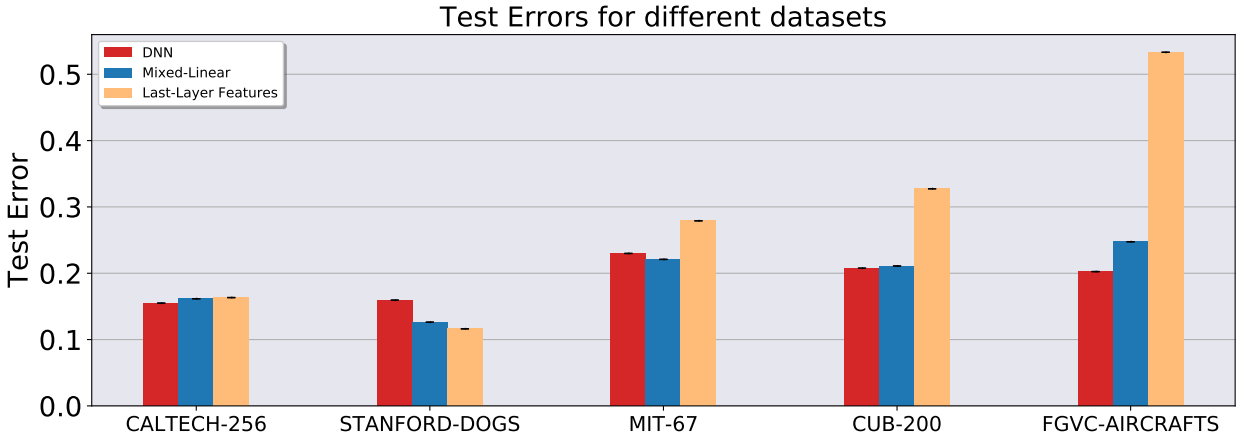

Figure 6.1: **Mixed-Linear model has comparable accuracy to standard DNN.** Plot of the test errors for different datasets using different models. We take a ResNet-50 pretrained on ImageNet and fine-tune it using different procedures, **(DNN)** We fine-tune the whole network on various datasets, **(Mixed-Linear)** We fine-tune the linearized ResNet-50 (eq. (6.5)) in a mixed private setting, **Last-Layer Features**: We simply fine-tune the final fully connected (FC) layer of the ResNet-50. We show that fine-tuning a linearized DNN using the mixed-privacy framework performs comparable to fine-tuning a DNN and outperforms simply fine-tuning the last FC layer.

the last layer features. Following the suggetsion of [AGR21a], we replace the ReLUs with Leaky ReLUs, since it boosts the accuracy of linearized models.

6.4 Forgetting procedure

The user weights $\mathbf{w}_u$ are obtained by minimizing the quadratic loss function $L_{\mathcal{D}}^{\text{MSE}}$ in eq. (6.8) on the user data $\mathcal{D}$. Let $\mathcal{D}_f \subset \mathcal{D}$ denote a subset of samples we want to forget (by hypothesis $\mathcal{D}_c \cap \mathcal{D}_f = \emptyset$, i.e., the core data is not going to change) and let $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$ denote the remaining data. As discussed in Section 6.2, in case of quadratic training loss the optimal forgetting step to delete $\mathcal{D}_f$ is given by:

$$\mathbf{w}_u \mapsto \mathbf{w}_u - H_{\mathcal{D}_r}^{-1}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u), \tag{6.9}$$

where we define $g_{\mathcal{D}_r}(\mathbf{w}_u) \triangleq \nabla_{\mathbf{w}} L_{\mathcal{D}_r}(\mathbf{w}_u)$ and we can explicitly write the Hessian $H_{\mathcal{D}_r}(\mathbf{w}_c)$ of the loss eq. (6.8) as:

$$H_{\mathcal{D}_r}(\mathbf{w}_c) = \sum_{\mathbf{x} \in \mathcal{D}_r} \nabla_{\mathbf{w}} f_{\mathbf{w}_c}(\mathbf{x})^T \nabla_{\mathbf{w}} f_{\mathbf{w}_c}(\mathbf{x}) + \mu I. \tag{6.10}$$

where I is the identity matrix of size d. Thus, forgetting amounts to computing the update step eq. (6.9). Unfortunately, even if we can easily write the hessian $D_{\mathcal{D}_r}$ in closed form, we cannot store it in memory and much less invert it. Instead, we now discuss how to find an approximation of the forgetting step $H^{-1}_{\mathcal{D}_r}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u)$ by solving an optimization problem which does not require constructing or inverting the hessian.

Since $H_{\mathcal{D}_r}(\mathbf{w}_c)$ is positive definite, we can define the auxiliary loss function

$$\hat{L}_{\mathcal{D}_r}(\mathbf{v}) = \frac{1}{2} \mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c) \mathbf{v} - g_{\mathcal{D}_r}(\mathbf{w}_u)^T \mathbf{v} \tag{6.11}$$

By setting the gradient to zero, it is easy to see that the forgetting update $H^{-1}_{\mathcal{D}_r}(\mathbf{w}_c) g_{\mathcal{D}_r}(\mathbf{w}_u)$ is the unique minimizer of $\hat{L}_{\mathcal{D}_r}(\mathbf{v})$, so we can recast computing the forgetting update as simply minimizing the loss $\hat{L}_{\mathcal{D}_r}(\mathbf{v})$ using SGD. In general, the product $\mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c)\mathbf{v}$ of eq. (6.11) can be computed efficiently without constructing the Hessian using the Hessian-Vector product algorithm [KL17]. However, in our case we have a better alternative due to the fact that we use MSE loss and that ML-model is linear in weight-space: Using eq. (6.10), we have that

$$\mathbf{v}^T H_{\mathcal{D}_r}(\mathbf{w}_c)\mathbf{v} = \sum_{\mathbf{x} \in \mathcal{D}_r} \|\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x})\mathbf{v}\|^2 + \mu \|\mathbf{v}\|^2, \tag{6.12}$$

where $\nabla_{\mathbf{w}} f_{\mathbf{w}_c^*}(\mathbf{x})\mathbf{v}$ is a Jacobian-Vector product which can be computed efficiently (see Section 6.3.1). Using this result, we compute the (approximate) minimizer of eq. (6.11) using SGD. When optimizing eq. (6.11), we compute $g_{\mathcal{D}_r}(\mathbf{w}_u)$ exactly on $\mathcal{D}_r$ and approximate eq. (6.10) by Monte-Carlo sampling. In Figure 6.4, we show this method outperforms full stochastic minimization of eq. (6.11).

Mixed-Linear Forgetting. Let $\Delta \mathbf{w}_u \triangleq \mathcal{A}_\tau(\hat{L}_{\mathcal{D}_r})$ be the approximate minimizer of eq. (6.11) obtained by training with SGD for τ iterations. Our Mixed-Linear (ML) forgetting procedure $S(\mathbf{w})$

for the ML-model in eq. (6.5) is:

$$\boxed{\mathbf{w}_u \mapsto \mathbf{w}_u - \Delta\mathbf{w}_u + \sigma^2\epsilon} \tag{6.13}$$

where $\epsilon \sim N(0, I)$ is a random noise vector [GAS19a, GAS20]. As mentioned in Section 6.2, we need to add noise to the weights since $\Delta\mathbf{w}_u$ is only an approximation of the optimal forgetting step, and the small difference may still contain information about the original data. By adding noise, we destroy the remaining information. Larger values of σ ensure better forgetting, but can reduce the performance of the model. In the next sections, we analyze theoretically and practically the role of σ.

Sequential forgetting. In practical applications, we may receive several separate requests to forget the data in a sequential fashion. In such cases, we simply apply the forgetting procedure in eq. (6.13) on the weights obtained at the end of the previous step. A key component is to ensure that the performance of the system does not deteriorate too much after many sequential requests, which we do next.

6.5 Bounds on Remaining Information

We now derive bounds on the amount of information that an attacker can extract from the weights of the model after applying the scrubbing procedure eq. (6.13). This will also guide us in selecting the optimal σ and the number of iterations τ to approximate the forgetting step that are necessary to reach a given privacy level (see fig. 6.2). Let $Y_{\mathcal{D}_F}$ denote some attribute of interest regarding $\mathcal{D}_f$ an attacker might want to access, then from Proposition 1 in [GAS19a] we have:

$$\underbrace{\mathcal{I}(Y_{\mathcal{D}_f}, S(\mathbf{w}))}_{\text{Recovered Information}} \leq \underbrace{\mathcal{I}(\mathcal{D}_f, S(\mathbf{w}))}_{\text{Remaining Information in Weights}}$$

where $S(\mathbf{w}) = S(\mathbf{w}, \mathcal{D}, \mathcal{D}_f)$ is the scrubbing/forgetting method which given weights $\mathbf{w}$ trained on $\mathcal{D}$ removes information about $\mathcal{D}_f$ (which in our case is given by eq. 6.13). Hence, bounding the amount of information about $\mathcal{D}_f$ that remains in the weights $S(\mathbf{w})$ after forgetting uniformly bounds all the information that an attacker can extract.

We now upper-bound the remaining information $\mathcal{I}(\mathcal{D}_f, S(\mathbf{w}))$ after applying the forgetting procedure in eq. (6.13) to our ML-model, over multiple forgetting requests. Let $\cup_{k=1}^{K} {\mathcal{D}_f}^k$ be the total data asked to be forgotten at the end of K forgetting requests and let $\mathbf{w}_u^K$ be the weights obtained using the forgetting procedure in eq. (6.13) sequentially. Then we seek to provide a bound on the mutual information between the two, i.e., $\mathcal{I}(\cup_{k=1}^{K} \mathcal{D}_f^k) \triangleq \mathcal{I}(\cup_{k=1}^{K} \mathcal{D}_f^k, \mathbf{w}_u^K)$. We prove the following theorem.

Theorem 1 (Informal)**.** *Let $\Delta \mathbf{w}_u = \mathcal{A}_\tau(\hat{L}_{\mathcal{D}_r})$ be the approximate update step obtained minimizing $\hat{L}_{\mathcal{D}_r}$ (eq. 6.13) using τ steps of SGD with mini-batch size B. Let $\gamma = 1 - \mu^2/\beta^2$, where β is the smoothness constant of the loss in $eq.$ (6.7). Consider a sequence of K equally sized forgetting requests $\{\mathcal{D}_f^1, \mathcal{D}_f^2, \ldots, \mathcal{D}_f^K\}$ and let $\mathbf{w}_u^K$ be the weights obtained after the K requests using eq.* (6.13)*. Then we have the following bound on the amount of information remaining in the weights $\mathbf{w}_u^K$ about $\cup_{k=1}^{K} \mathcal{D}_f^k$*

$$\mathcal{I}(\cup_{k=1}^{K} \mathcal{D}_f^k) \leq \frac{[L1R]\gamma^{\overset{\text{forgetting}}{\text{steps}}} c_0 \Big([L1R] c_1 \frac{r^{2\ \overset{\text{ratio to}}{\text{forget}}}}{\sigma^2} + [L1R] d^{\overset{\text{num.}}{\text{params}}} \Big) + [L1R] \frac{c_2}{B\sigma^2}^{\overset{\text{batch}}{\text{size}}}}{1 - (1+\alpha)\gamma^\tau}. \tag{6.14}$$

where $c_0, c_1, c_2 > 0$, $r = |\mathcal{D}_f^k|/|\mathcal{D}|$ and $0 < \alpha < 1/\gamma^\tau - 1$, $d = \dim(\mathbf{w})$ and $\gamma < 1$.

In [NRS20] a similar probabilistic bound is given on the distance of the scrubbed weights from the optimal weights for strongly convex Lipschitz loss functions trained using projected GD. We prove our bound for the more general case of a convex loss function with L_2 regularization trained using SGD (instead of GD) and also bound the remaining information in the weights.

Role of σ. We make some observations regarding eq. (6.14). First, increasing the variance σ^2 of the noise added to the weights after the forgetting step further reduces the possible leakage of information from an imperfect approximation. Of course, the downside is that increasing the noise may reduce the performance of the model (see Figure 6.2 (top) for the trade-off between the two).

Forgetting with more iterations. Running the algorithm $\mathcal{A}$ for an increasing number of steps τ improves the accuracy of the forgetting step, and hence reduces the amount of remaining information. We confirm this empirically in Figure 6.2 (bottom). Note however that there is a diminishing return.

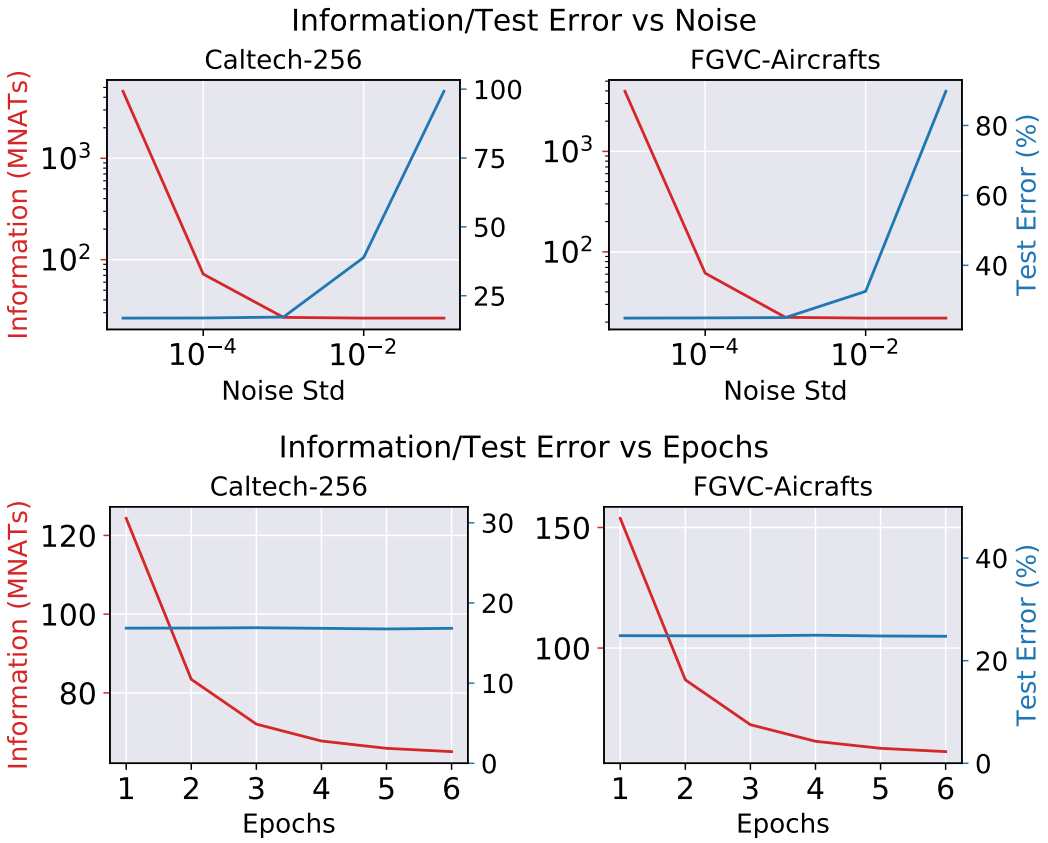

Figure 6.2: **Forgetting-Accuracy Trade-off** Plots of the amount of remaining information in the weights about the data to forget (red, left axis) and test error (blue, right axis) as a function of the (top) scrubbing noise and (bottom) number of optimization iterations used to compute the scrubbed weights in eq. (6.13). We aim to forget 10% of the training data through 10 forgetting requests on the Caltech-256 (left) and Aircrafts datasets (right). Note that the remaining information in the weights decreases with an increase in the forgetting noise or the number of epochs during forgetting as predicted by the bound in Theorem 1. Increasing the forgetting noise increases the test error after forgetting (top). In terms of the computational efficiency/speed, doing 2-3 passes over the data (i.e. 2-3 epochs) is sufficient for forgetting (in terms of the test error and the remaining information) rather than re-training from scratch for 50 epochs (bottom) for each forgetting request $\mathcal{D}_f^k$. Thus providing a 16-25$\times$ speed-up per forgetting request. We fine-tune the ML-Forgetting model for 50 epochs while training the user weights. Values for τ and σ can be chosen using these trade-off curves given a desired privacy level.

Figure 6.3: **Read-out functions for different forgetting methods.** We forget a subset of 10% of the training data through 10 equally-size sequential deletion requests using different forgetting methods, and show the value of the several readout functions for the resulting scrubbed models. Ideally, the value of the readout function should be the same as the value (denoted with the green area) obtained on a model retrained from scratch without those samples. Closer to the green area is better. **(Original)** denotes the trivial baseline where we do apply any forgetting procedure. **(Fisher)** Adds Fisher noise as as described in ([GAS19a]), **(ML-Forgetting)** The model obtained with our method after forgetting. In all cases, we observe that ML-Forgetting obtains a model that is indistinguishable from one trained from scratch without the data, whereas the other methods fail to do so. This is particularly the case for the *Retrain Time* readout functions, which exploits full knowledge of the weights and it is therefore more difficult to defend against.

This is due to the variance of the stochastic optimization overshadowing gains in accuracy from longer optimization (see the additive term depending on the batch size). Increasing the batch-size, B in eq. (6.13) reduces the variance of the estimation and leads to better convergence.

Fraction of data to forget. Finally, forgetting a smaller fraction $r = |\mathcal{D}_f^k|/|\mathcal{D}|$ of the data is easier. On the other hand, increasing the number of parameters d of the model may make the forgetting more difficult.

6.6 Experiments

We use a ResNet-50 [HZR16] as the model $f_{\mathbf{w}}(\mathbf{x})$ in ML-Forgetting. Unless specified otherwise, we forget 10% of randomly chosen training data in all the experiments through 10 sequential forgetting requests each of size 1%. In the appendix, we also provide results for forgetting an entire class and show that our method is invariant to the choice of the subset to be forgotten. More experimental details can be found in the appendix.

Datasets used. We test our method on the following image classification tasks: Caltech-256 [GHP07], MIT-67 [SAS14], Stanford Dogs [KJY], CUB-200 [WBW11], FGVC Aircrafts [MRK13], CIFAR-10 [Kri09]. Readout function and forgetting-accuracy trade-off plots for MIT-67, Stanford-Dogs, CUB-200 and CIFAR-10 can be found in the appendix.

6.6.1 Readout functions

The forgetting procedure should be such that an attacker with access to the scrubbed weights $\mathbf{w}$ should not be able to construct some function $R(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}$, which will leak information about the set to forget $\mathcal{D}_f$. More precisely the scrubbing procedure should be such that for all $R(\mathbf{w})$:

$$\mathrm{KL}\left(\mathbf{P}(\underbrace{R(S(\mathbf{w},\mathcal{D},\mathcal{D}_f))|\mathcal{D}}_{\substack{\text{readout on weights}\\ \text{after forgetting } \mathcal{D}_f}}) \,\|\, \mathbf{P}(\underbrace{R(S_0(\mathbf{w}))|\mathcal{D}_r}_{\substack{\text{readout on weights}\\ \text{after re-training on } \mathcal{D}_r}})\right) = 0 \tag{6.15}$$

where $S_0(\mathbf{w})$ is some baseline function that does not depend on $\mathcal{D}_f$ (it only depends on the subset to retain $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$). Here $\mathbf{P}(\mathbf{w}|\mathcal{D})$, $\mathbf{P}(\mathbf{w}|\mathcal{D}_r)$ corresponds to the distribution of weights (due to the stochastic training algorithm) obtained after minimizing the empirical risk on $\mathcal{D}$, $\mathcal{D}_r$ respectively. $S(\mathbf{w},\mathcal{D},\mathcal{D}_f)$ corresponds to the scrubbing update defined in eq. (6.13). We choose $S_0(\mathbf{w}) = \mathbf{w} + z$, where $z \sim \mathcal{N}(0,\sigma^2 I)$. For an ideal forgetting procedure, the value of the readout functions (or evaluation metrics) should be same for a model obtained after forgetting $\mathcal{D}_f$ and re-trained from scratch without using $\mathcal{D}_f$. Some common choice of readout functions include (see Figure 6.3):

1. **Error on $\mathcal{D}_r$, $\mathcal{D}_f$, $\mathcal{D}_{\text{Test}}$:** The scrubbed and the re-trained model (from scratch on $\mathcal{D}_r$) should

have similar accuracy on all the three subsets of the data

2. **Re-learn Time:** We fine-tune the scrubbed (model after forgetting) and re-trained model for a few iterations on a subset of the training data (which includes $\mathcal{D}_f$) and compute the number of iterations it takes for the models to re-learn $\mathcal{D}_f$. An ideal forgetting procedure should be such that the re-learn time should be comparable to the re-trained model (we plot the relative re-train time in Figure 6.3). Re-learn time serves a proxy for the amount of information remaining in the weights about $\mathcal{D}_f$ (see Figure 6.3).

3. **Activation Distance:** We compute the distance between the final activations of the scrubbed weights and the re-trained model ($\mathbf{w}_{\mathcal{D}_r}$) on different subsets of data. More precisely we compute the following: $\mathbb{E}_{\mathbf{x}\in\mathcal{D}'}[\|\operatorname{softmax}(f_{\mathbf{w}}(\mathbf{x})) - \operatorname{softmax}(f_{\mathbf{w}_{\mathcal{D}_r}}(\mathbf{x}))\|_1]$, where $\mathcal{D}' = \mathcal{D}_r, \mathcal{D}_f, \mathcal{D}_{\text{Test}}$. We compare different $\mathbf{w}$ corresponding to the original weights without any forgetting, weights after adding Fisher noise and ML-forgetting (see Figure 6.3). This serves as a proxy for the amount of information remaining in the activations about $\mathcal{D}_f$.

4. **Membership Attack:** We construct a simple yet effective membership attack similar to [GAS20] using the entropy of the model output. Ideally, a forgetting procedure should have the same attack success as a re-trained model (which is what we observe, see Figure 6.3).

6.6.2 Complete vs Stochastic residual gradient

In eq. (6.13) we compute the residual gradient $g_{\mathcal{D}_r}(\mathbf{w}_u)$ completely once over the remaining data instead of estimating that term stochastically using $\mathcal{A}$. In Figure 6.4, we compare both the methods of computing the residual gradient. We show that in the ideal region of noise (i.e. $\sigma \in [10^{-5}, 10^{-3}]$), both the remaining information and test error after forgetting (10% of the data through 10 requests) is lower when computing the residual gradient completely.

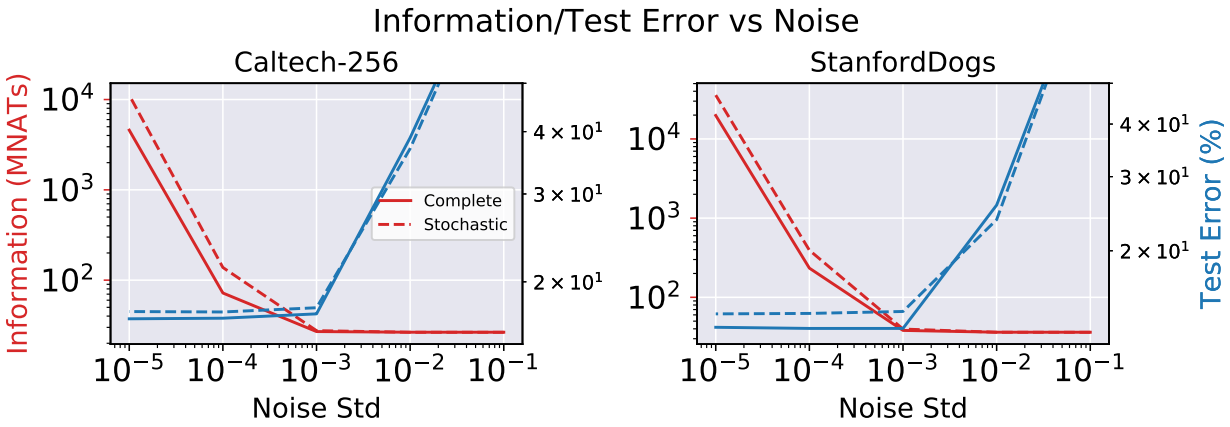

Figure 6.4: **Comparison of complete and stochastic residual gradient estimation for forgetting.** ML-Forgetting uses the complete residual gradient for the forgetting step, exploiting the fact that the loss function is quadratic. However, one can also estimate it stochastically, which is equivalent to fine-tuning on the remaining data to forget. Here we show that indeed both methods work but – when using the same number of steps – complete estimate gives a better solution due to smaller variance and faster convergence (lower test error and information leakage).

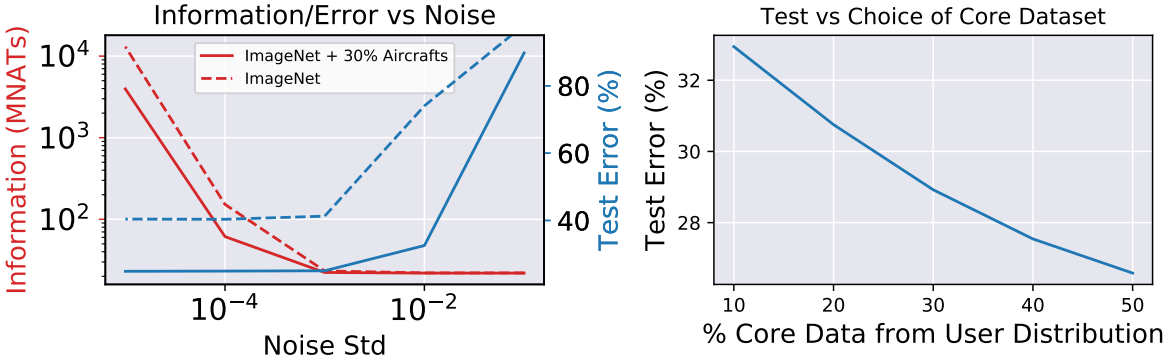

Figure 6.5: **Effect of using a core data close to the task.** Plot of the remaining information and test error on *Aircrafts* using (a) generic ImageNet core data, and (b) ImageNet pre-training + 30% of the *Aircrafts*. When the core data contain information close to the user task that the network does not need to forget, ML-Forgetting can exploit this to create better core-weights and a correspondingly better linearized model. This improves both the accuracy of the model and makes forgetting easier, as seen from the accuracy-forgetting curves in the plot.

6.6.3 Effect of choosing different core datasets

For fine-grained datasets like FGVC-Aircrafts and CUB-200, we show that if the core data has some information about the user task, then it improves forgetting significantly both in terms of the remaining information and the test accuracy. In Figure 6.5, we show that using ImageNet + 30% of the Aircrafts (we assume that we are not asked to forget this 30% of the data) as core data and 100% of the Aircrafts as the user data, performs much better than simply using ImageNet as core. In Figure 6.5 (right), we also show that increasing the percentage of user distribution in the core data improves the test accuracy of the Mixed-Linear model.

6.7 Conclusion

We provide a practical forgetting procedure to remove the influence of a subset of the data from a trained image classification model. We achieve this by linearizing the model using a mixed-privacy setting which enables us to split the weights into a set of core and forgettable user weights. When asked to delete all the user data, we can simply discard the user weights. The quadratic nature of the training loss enables us to efficiently forget a subset of the user data without compromising the accuracy of the model. In terms of the time-complexity, we only need 2-3 passes over the dataset per forgetting query for removing information from the weights rather than the 50 re-training epochs, thus, providing a $16\times$ or more speed-up per request (see Figure 6.2). We test the forgetting procedure against various read-out functions, and show that it performs comparably to a model re-trained from scratch (the ideal paragon). Finally, we also provide theoretical guarantees on the amount of remaining information in the weights and verify the behavior of the information bounds empirically through extensive evaluation in Figure 6.2.

Our forgetting procedure heavily relies on the strongly convex nature of the loss landscape which is induced by L_2 regularization (increasing it improves forgetting but compromises accuracy). The quality of forgetting also relies on the subset of data to be forgotten, however, we will leave this for the future work. Even though we provide a forgetting procedure for deep networks by

linearizing them without compromising their accuracy, directly removing information from highly non-convex deep networks efficiently still remains an unsolved problem at large.

6.8 Appendix

In this appendix we provide additional experiments (Section 6.8.1), experimental details (Section 6.8.3) and theoretical results (Section 6.8.4).

6.8.1 Additional Experiments

6.8.1.1 Forgetting an entire class

In the main paper we considered forgetting a random subset of 10% of the training data. Here we consider instead the problem of completely forgetting all samples of a given class in a single forgetting request. In Figures 6.6 and 6.7, we observe that also in this setting our proposed method outperforms other methods and is robust to different readout functions. Note that for the case of removing an entire class the target forget error (i.e. the error on the class to forget) is 100%.

6.8.1.2 Role of L_2-Regularization

We plot the amount of remaining information and the test error as a function of the L_2 regularization coefficient. Note that instead of incorporating weight decay directly in the optimization step, as it is often done, we explicitly add the L_2 regularization to the loss function. As expected theoretically (Theorem 3), increasing the regularization coefficient makes the training optimization problem more strongly convex, which in turn makes forgetting easy. However, increasing weight decay too much also hurts the accuracy of the model. Hence there is a trade-off between the amount of remaining information and the amount of regularization with respect to the regularization. We plot the trade-off in Figure 6.8.

Figure 6.6: Readout function plot similar to Figure 6.3 for Caltech-256 dataset, where we forget an entire class rather a sequence of randomly sampled data subsets.

Figure 6.7: Readout function plot similar to Figure 6.3 for FGVC-Aicrafts dataset, where we forget an entire class rather a sequence of randomly sampled data subsets.

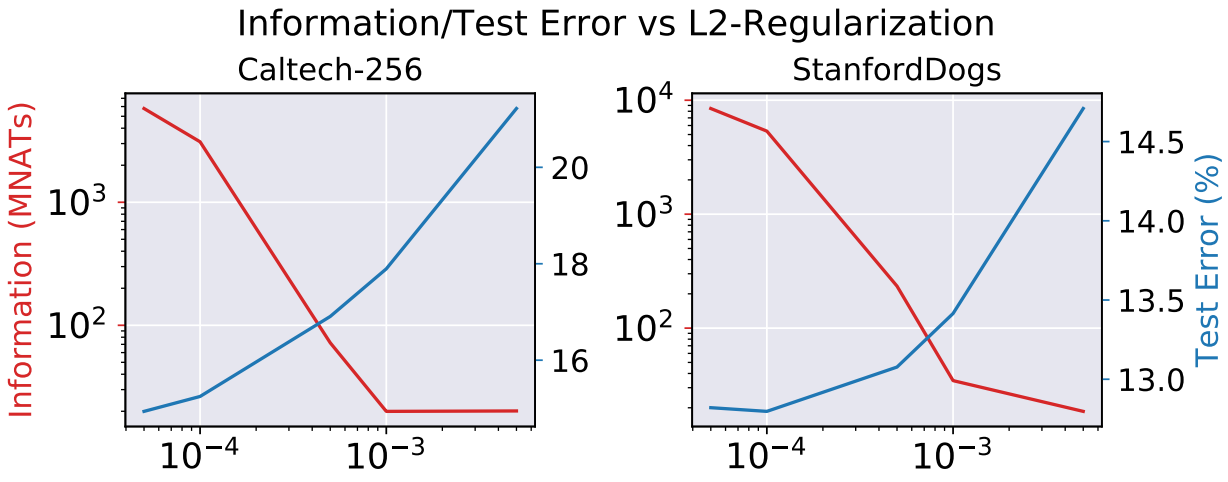

Figure 6.8: Plot of the amount of remaining information and test error vs the L_2 regularization coefficient. We forget 10% of the training data sequentiall through 10 forgetting request.

6.8.1.3 More experiments using SGD for forgetting

We repeat the same experiments as in Figure 6.3 on the following datasets: *Stanford Dogs, MIT-67, CIFAR-10, CUB-200, FGVC Aircrafts*. Overall, we observe consistent results over all datasets.

6.8.2 Information vs Noise/Epochs

6.8.3 Experimental Details

We use a ResNet-50 pre-trained on ImageNet. For the plots in Figure 6.1, we train ML-Forgetting model using SGD for 50 epochs with batch size 64, learning rate lr=0.05, momentum=0.9, weight decay=0.00001 where the learning rate is annealed by 0.1 at 25 and 40 epochs. We explicitly add the L_2 regularization to the loss function instead of incorporating it in the SGD update equation. We only linearize the final layers of ResNet-50 and scale the one-hot vectors by 5 while using the MSE loss. For fine-grained datasets, FGVC-Aircrafts and CUB-200, in addition to the ImageNet pre-training, we also pre-train them using randomly sampled 30% of the training data (which we assume is part of the core set).

Figure 6.9: Same experiments as Figure 6.3 for StanfordDogs.

Figure 6.10: Same experiments as Figure 6.3 for MIT-67.

Figure 6.11: Same experiments as Figure 6.3 for CIFAR-10.

Figure 6.12: Same experiments as Figure 6.3 for CUB-200.

Figure 6.13: Same experiments as Figure 6.3 for FGVC-Aircrafts.

For the training the ML-Forgetting model in the readout functions and information plots using SGD, we use the same experimental setting as above with a increased weight decay=0.0005 for Caltech-256,StanfordDogs and CIFAR-10 and 0.001 for MIT-67,CUB-200 and FGVC-Aircrafts. We use a higher value of weight decay to increase the strong convexity constant of the training loss function, which facilitates forgetting (see Lemma 7).

For forgetting using ML-Forgetting model in the readout function/information plots using SGD (ML-Forgetting to minimize eq. (6.11)), we use momentum=0.999 and decrease the learning rate by 0.5 per epoch. We run SGD for 3 epochs with an initial lr=0.01 for Caltech-256, StanfordDogs and CIFAR-10 and run it for 4 epochs with initial lr=0.025 for MIT-67, CUB-200 and FGVC-Aircrafts.

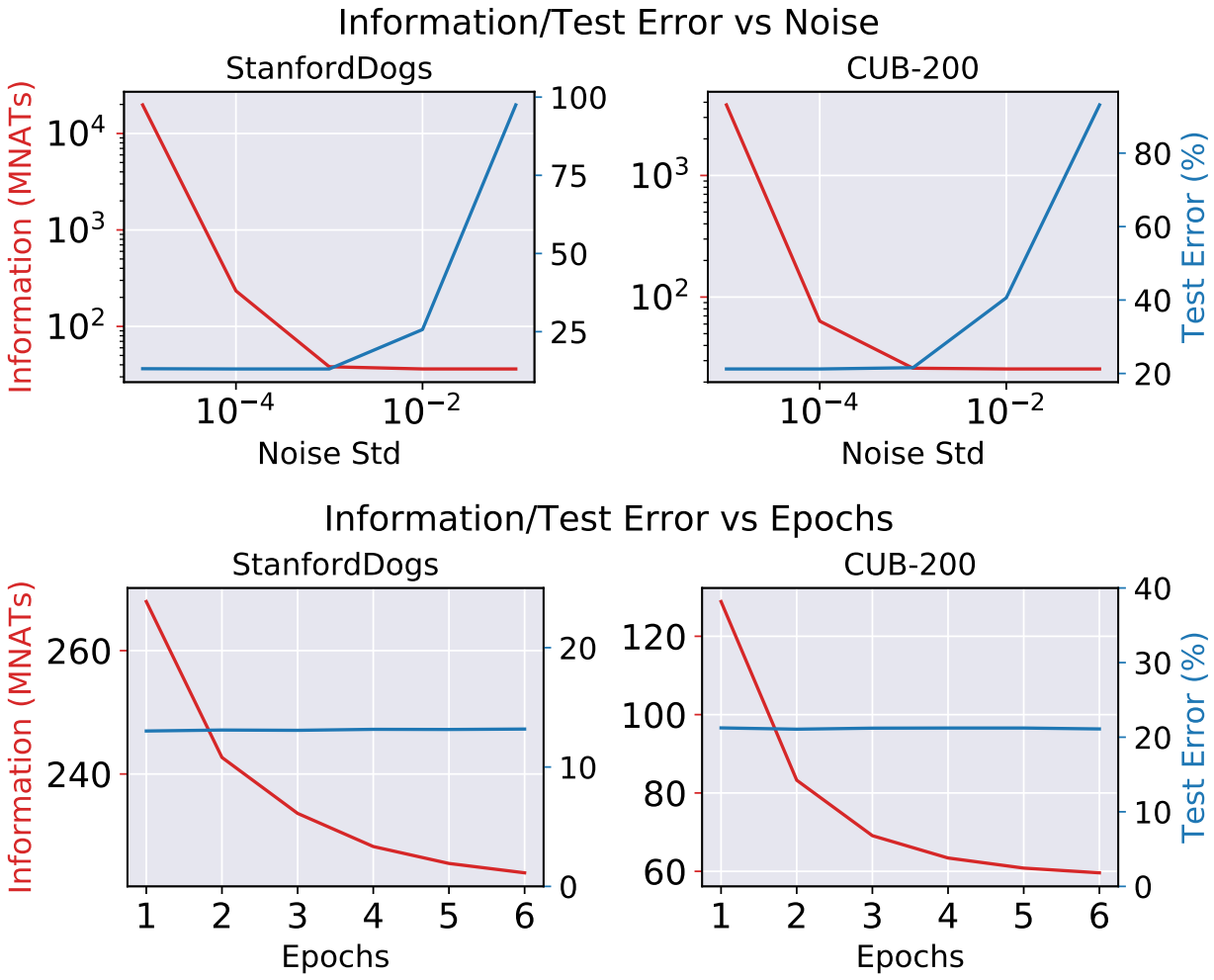

Figure 6.14: Same experiment as Figure 6.2 for Stanforddogs and CUB-200 datasets.

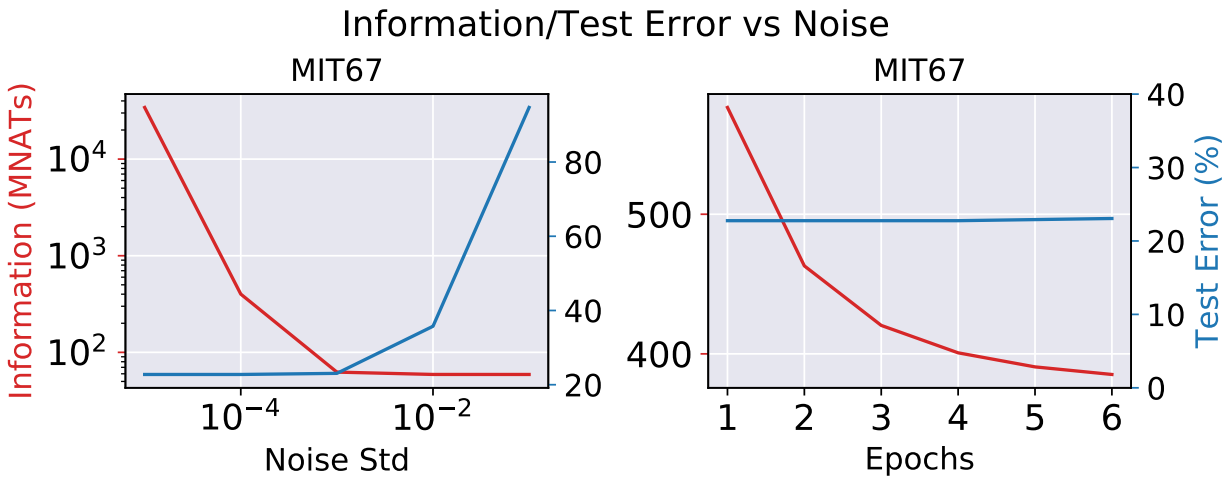

Figure 6.15: Same experiment as Figure 6.2 for MIT67.

6.8.4 Theoretical Results

Lemma 4. *Let* $\mathbf{x}, \mathbf{y}$ *be two random vectors such that* $\mathbb{E}\|\mathbf{x}\|^2, \mathbb{E}\|\mathbf{y}\|^2 > 0$. *Then we have the following, for any* $\alpha > 0$*:*

$$\mathbb{E}\|\mathbf{x}+\mathbf{y}\|^2 \leq (1+\alpha)\mathbb{E}\|\mathbf{x}\|^2 + (1+\frac{1}{\alpha})\mathbb{E}\|\mathbf{y}\|^2$$

Proof.

$$\begin{aligned}
\mathbb{E}\|\mathbf{x}+\mathbf{y}\|^2 &= \mathbb{E}(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\big\langle \mathbf{x}, \mathbf{y}\big\rangle) \\
&\leq \mathbb{E}(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2|\big\langle \mathbf{x}, \mathbf{y}\big\rangle|) \\
&\overset{(a)}{\leq} \mathbb{E}\bigg(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\sqrt{\|\mathbf{x}\|^2}\sqrt{\|\mathbf{y}\|^2}\bigg) \\
&= \mathbb{E}\bigg(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\sqrt{\|\mathbf{x}\|^2\alpha}\sqrt{\frac{\|\mathbf{y}\|^2}{\alpha}}\bigg) \\
&\overset{(b)}{\leq} \mathbb{E}\bigg(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + \|\mathbf{x}\|^2\alpha + \|\mathbf{y}\|^2\frac{1}{\alpha}\bigg) \\
&= (1+\alpha)\mathbb{E}\|\mathbf{x}\|^2 + (1+\frac{1}{\alpha})\mathbb{E}\|\mathbf{y}\|^2
\end{aligned} \tag{6.16}$$

for any $\alpha > 0$, where (a) follows from the Cauchy-Schwarz inequality and (b) follows from the AM-GM inequality. □

Lemma 5. *Let* $C = \{\mathbf{w}|\mathbf{w} \in \mathbb{R}^d$ *and* $\|\mathbf{w}\| \leq R < \infty\}$ *and* $\ell(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ *be a strongly convex function with* $max_{\mathbf{w}\in C}\ell(\mathbf{w}) < \infty$, $G \triangleq max_{\mathbf{w}\in C}\|\nabla\ell(\mathbf{w})\| < \infty$ *and* $\mathbf{w}^* = argmin_{\mathbf{w}\in\mathbb{R}^d}\ell(\mathbf{w})$ *such that* $\|\mathbf{w}^*\| < \infty$. *Then* $\forall \mathbf{w}_1, \mathbf{w}_2 \in C$, *we have that* $|\ell(\mathbf{w}_1) - \ell(\mathbf{w}_2)| \leq G\|\mathbf{w}_1 - \mathbf{w}_2\|$. *When* $\ell(\mathbf{w})$ *is also quadratic with* $\beta = \lambda_{Max}(\nabla^2\ell(\mathbf{w}))$, *the maximum eigen value of the Hessian, we have that* $G = \beta\big(R + \|\mathbf{w}^*\|\big)$.

Proof. Let $g(t) = \ell(\mathbf{w}_1 t + \mathbf{w}_2(1-t))$, where $t \in [0,1]$ then from Mean Value Theorem (MVT) we have that $g(1) - g(0) = g^{'}(c)$ for some c in between 0 and 1. This implies that $g(1) = \ell(\mathbf{w}_1)$, $g(0) = \ell(\mathbf{w}_2)$ and $g^{'}(c) = \big\langle \nabla\ell(\mathbf{w}_1 t + \mathbf{w}_2(1-t)), \mathbf{w}_1 - \mathbf{w}_2\big\rangle$. Thus from MVT we get:

$$
\begin{aligned}
|\ell(\mathbf{w}_1) - \ell(\mathbf{w}_2)| &= |\langle \nabla \ell(\mathbf{w}_1 t + \mathbf{w}_2(1-t)), \mathbf{w}_1 - \mathbf{w}_2 \rangle| \\
&\overset{(a)}{\leq} \|\nabla \ell(\mathbf{w}_1 t + \mathbf{w}_2(1-t))\| \|\mathbf{w}_1 - \mathbf{w}_2\| \\
&\overset{(b)}{\leq} (\max_{\mathbf{w} \in C} \|\nabla \ell(\mathbf{w})\|) \|\mathbf{w}_1 - \mathbf{w}_2\| \qquad (6.17)
\end{aligned}
$$

where (a) follows from the Cauchy-Schwarz inequality and (b) follows from the fact that $\mathbf{w}_1 t + \mathbf{w}_2(1-t) \in C$ and $\forall \mathbf{w} \in C$, $\|\nabla \ell(\mathbf{w})\| \leq \max_{\mathbf{w} \in C} \|\nabla \ell(\mathbf{w})\|$.

When $\ell(\mathbf{w})$ is quadratic, then we can always write $\ell(\mathbf{w}) = \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T Q (\mathbf{w} - \mathbf{w}^*) + c_0$, where $Q = \nabla^2 \ell(\mathbf{w})$ is a constant symmetric matrix and $c_0 = \ell(\mathbf{w}^*)$. From our definition of $\ell(\mathbf{w})$ we can write:

$$
\begin{aligned}
\max_{\mathbf{w} \in C} \|\nabla \ell(\mathbf{w})\| &= \max_{\mathbf{w} \in C} \|Q(\mathbf{w} - \mathbf{w}^*)\| \\
&\overset{(a)}{\leq} \max_{\mathbf{w} \in C} \beta \|\mathbf{w} - \mathbf{w}^*\| \\
&\overset{(b)}{\leq} \beta(\max_{\mathbf{w} \in C} \|\mathbf{w}\| + \|\mathbf{w}^*\|) \\
&\leq \beta(R + \|\mathbf{w}^*\|)
\end{aligned}
$$

where (a) follows from the definition of β and (b) follows from the triangle inequality. Substituting this result in Equation (6.17) we get:

$$
|\ell(\mathbf{w}_1) - \ell(\mathbf{w}_2)| \leq \beta\big(R + \|\mathbf{w}^*\|\big) \|\mathbf{w}_1 - \mathbf{w}_2\|
$$

□

Lemma 6. *Consider a function* $L_{\mathcal{D}}(\mathbf{w}) = \frac{1}{n} \sum_{i \in \mathcal{D}} \ell_i(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w}\|^2$, *where* $\ell_i(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$, $\ell_i(0) \leq M$ *and* $\mathcal{D}$ *is a dataset of size* n. *Let* $\mathbf{w}^*_{\mathcal{D}} = \textit{argmin}_{\mathbf{w} \in \mathbb{R}^d} L_{\mathcal{D}}(\mathbf{w})$, *then* $\|\mathbf{w}^*_{\mathcal{D}}\| \leq \sqrt{\frac{2M}{\mu}}$.

Proof.

$$
\frac{\mu}{2} \|\mathbf{w}^*_{\mathcal{D}}\|^2 \overset{(a)}{\leq} L_{\mathcal{D}}(\mathbf{w}^*_{\mathcal{D}}) \overset{(b)}{\leq} L_{\mathcal{D}}(0) \overset{(c)}{\leq} M
$$

Thus, $\|\mathbf{w}^*_{\mathcal{D}}\| \leq \sqrt{\frac{2M}{\mu}}$, where (a) follows from the assumption that $\ell_i(\mathbf{w})$ is non-negative, (b) follows from the fact that $\mathbf{w}^*_{\mathcal{D}}$ is the minimizer of $L_{\mathcal{D}}(\mathbf{w})$ and (c) follows from the assumption that $\ell_i(0) \leq M$. Note that the result is independent of n, thus, the empirical risk minimizers of the datasets obtained by removing a subset of samples will also lie within a $d-$dimensional sphere of radius $\sqrt{\frac{2M}{\mu}}$.

□

Lemma 7. *Consider a function $L_{\mathcal{D}}(\mathbf{w}) = \frac{1}{n}\sum_{i\in\mathcal{D}}\widehat{\ell}_i(\mathbf{w})$, where $\widehat{\ell}_i(\mathbf{w}) = \ell_i(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}\|^2$, $\ell_i(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ is strongly convex function with $\ell_i(0) \leq M < \infty$. Let $\mathcal{D}_r$ (retain set) be the dataset remaining after removing b samples $\mathcal{D}_f$ (forget set) from $\mathcal{D}$ i.e. $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$. Let $\mathbf{w}^*_{\mathcal{D}} \triangleq$ argmin$_{\mathbf{w}\in\mathbb{R}^d} L_{\mathcal{D}}(\mathbf{w})$ and $\mathbf{w}^*_{\mathcal{D}_r} \triangleq$ argmin$_{\mathbf{w}\in\mathbb{R}^d} L_{\mathcal{D}_r}(\mathbf{w})$. Let $C = \{\mathbf{w}|\mathbf{w} \in \mathbb{R}^d$ and $\|\mathbf{w}\| \leq \sqrt{2M/\mu}\}$ and $G \triangleq$ max$_{\mathbf{w}\in C}\|\nabla\ell(\mathbf{w})\| < \infty$. Then we have that:*

$$\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\| \leq \frac{2bG}{n\mu}$$

When $\ell_i(\mathbf{w})$ is also quadratic with $\beta =$ max$_{i\in\mathcal{D}}\lambda_M(\nabla^2\hat{\ell}_i(\mathbf{w}))$, the smoothness constant of $L_{\mathcal{D}}(\mathbf{w})$, we have that $G = 2\beta\sqrt{2M/\mu}$.

Proof. We use the same technique as proposed in [NRS20].

$$\begin{aligned} L_{\mathcal{D}}(\mathbf{w}^*_{\mathcal{D}_r}) &= \frac{n-b}{n}L_{\mathcal{D}_r}(\mathbf{w}^*_{\mathcal{D}_r}) + \frac{b}{n}L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}_r}) \\ &\overset{(a)}{\leq} \frac{n-b}{n}L_{\mathcal{D}_r}(\mathbf{w}^*_{\mathcal{D}}) + \frac{b}{n}L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}_r}) \\ &= L_{\mathcal{D}}(\mathbf{w}^*_{\mathcal{D}}) + \frac{b}{n}L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}_r}) - \frac{b}{n}L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}}) \end{aligned} \tag{6.18}$$

where, (a) first inequality follows from the fact that $\mathbf{w}^*_{\mathcal{D}_r}$ is the minimizer of $L_{\mathcal{D}_r}(\mathbf{w})$.

From Lemma 6 we know that $\|\mathbf{w}^*_{\mathcal{D}}\|, \|\mathbf{w}^*_{\mathcal{D}_r}\|, \|\mathbf{w}^*_{\mathcal{D}_f}\| \leq \sqrt{\frac{2M}{\mu}}$. Also from the definition of β we have that $\lambda_M(\nabla^2 L_{\mathcal{D}_f}(\mathbf{w})) \leq \beta$. Then applying Lemma 5 with $R = \sqrt{\frac{2M}{\mu}}$ for $L_{\mathcal{D}_f}(\mathbf{w})$ we get

that

$$|L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}}) - L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}_r})| \leq G\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\| \tag{6.19}$$

From the the definition of $L_{\mathcal{D}_f}(\mathbf{w})$ we know that it is a $\mu-$strongly convex function. So we have the following property:

$$L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}_r}) \geq L_{\mathcal{D}_f}(\mathbf{w}^*_{\mathcal{D}}) + \frac{\mu}{2}\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\|^2 \tag{6.20}$$

Substituting Equation (6.19) and Equation (6.20) in Equation (6.18) we get:

$$\frac{\mu}{2}\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\|^2 \leq bG\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\|$$
$$\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\| \leq \frac{2bG}{n\mu}$$

When $\ell_i(\mathbf{w})$ is also quadratic with $\beta = \max_{i\in\mathcal{D}}\lambda_{\text{Max}}(\nabla^2\hat{\ell}_i(\mathbf{w}))$, the smoothness constant, then from Lemma 5 we have $G = \beta\big(R + \|\mathbf{w}^*_{\mathcal{D}}\|\big) \leq \beta(2\sqrt{2M/\mu})$.

$$\|\mathbf{w}^*_{\mathcal{D}} - \mathbf{w}^*_{\mathcal{D}_r}\| \leq \frac{4b\beta\sqrt{2M}}{n\mu^{3/2}}$$

□

Lemma 8. *Let $\ell(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ be a convex and $\beta-$ smooth with minimizer, $\mathbf{w}^* = argmin_{\mathbf{w}\in\mathbb{R}^d}\ell(\mathbf{w})$. Then we have that:*

1. $\ell(\mathbf{w}) - \ell(\mathbf{w}^*) \leq \frac{\beta}{2}\|\mathbf{w} - \mathbf{w}^*\|^2$
2. $\|\nabla\ell(\mathbf{w})\|^2 \leq 2\beta(\ell(\mathbf{w}) - \ell(\mathbf{w}^*))$

Proof. From the definition of $\beta-$smoothness we have that:

$$\ell(\mathbf{w}_1) \leq \ell(\mathbf{w}_2) + \big\langle \nabla \ell(\mathbf{w}_2), \mathbf{w}_1 - \mathbf{w}_2 \big\rangle + \frac{\beta}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \tag{6.21}$$

Setting $\mathbf{w}_1 = \mathbf{w}$, $\mathbf{w}_2 = \mathbf{w}^*$ and using the fact that $\nabla \ell(\mathbf{w}^*) = 0$, we get that:

$$\ell(\mathbf{w}) - \ell(\mathbf{w}^*) \leq \frac{\beta}{2}\|\mathbf{w} - \mathbf{w}^*\|^2$$

For (2) we minimize Equation (6.21) with respect to $\mathbf{w}_1$:

$$\begin{aligned}
\min_{\mathbf{w}_1 \in \mathbb{R}^d} \ell(\mathbf{w}_1) &\leq \min_{\mathbf{w}_1 \in \mathbb{R}^d} \big(\ell(\mathbf{w}_2) + \big\langle \nabla \ell(\mathbf{w}_2), \mathbf{w}_1 - \mathbf{w}_2 \big\rangle + \frac{\beta}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \big) \\
&= \ell(\mathbf{w}_2) + \min_{\mathbf{w}_1 \in \mathbb{R}^d} \big(\big\langle \nabla \ell(\mathbf{w}_2), \mathbf{w}_1 - \mathbf{w}_2 \big\rangle + \frac{\beta}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \big) \\
&\overset{(a)}{=} \ell(\mathbf{w}_2) - \frac{\|\nabla \ell(\mathbf{w}_2)\|^2}{2\beta} \qquad (6.22)
\end{aligned}$$

$$(6.23)$$

where (a) follows from the result that $\mathbf{w}_1 = \mathbf{w}_2 - \dfrac{\nabla \ell(\mathbf{w}_2)}{\beta} = \text{argmin}_{\mathbf{w}_1 \in \mathbb{R}^d} \big(\big\langle \nabla \ell(\mathbf{w}_2), \mathbf{w}_1 - \mathbf{w}_2 \big\rangle + \dfrac{\beta}{2}\|\mathbf{w}_1 - \mathbf{w}_2\|^2 \big)$. Setting $\mathbf{w}_2 = \mathbf{w}$, $\min_{\mathbf{w}_1 \in \mathbb{R}^d} = \ell(\mathbf{w}^*)$ in Equation (6.23) and re-arranging the terms we get (2). □

Theorem 2. *(SGD) Consider $L_{\mathcal{D}}(\mathbf{w}) = \dfrac{1}{n}\sum_{i \in \mathcal{D}} \ell_i(\mathbf{w})$, where $\ell_i(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ is $\beta-$smooth and $\mu-$ strongly convex. Let $\mathbf{w}^* = argmin_{\mathbf{w} \in \mathbb{R}^d} L_{\mathcal{D}}(\mathbf{w})$ and $\mathbf{w}_i^* = argmin_{\mathbf{w} \in \mathbb{R}^d} \ell_i(\mathbf{w})$. Then we have the following result after t steps of SGD with batch-size B and constant learning rate $\eta = \mu/\beta^2$:*

$$\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 \leq \Big(1 - \frac{\mu^2}{\beta^2}\Big)^t \|\mathbf{w}_0 - \mathbf{w}^*\|^2 + \frac{2\sigma_\ell}{B\beta}$$

where $\sigma_\ell = \dfrac{1}{n}\sum_{i=1}^{n} \ell_i(\mathbf{w}^*) - \dfrac{1}{n}\sum_{i=1}^{n} \ell_i(\mathbf{w}_i^*)$

Proof. We do not use the bounded gradient assumption for the convergence of SGD and instead use the smoothness of our loss function [BBM18]. This is because the training loss in our case is a quadratic function whose gradient is linear and not bounded.

Consider a mini-batch SGD update,

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t)$$

where the examples $\{i_{t+1}^{(1)}, i_{t+1}^{(2)}, \cdots i_{t+1}^{(B)}\}$ are sampled uniformly at random with replacement for all the iterations t.

Then by expanding $\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2$,

$$\begin{aligned}
\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 &= \left\|\mathbf{w}_t - \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) - \mathbf{w}^*\right\|^2 \\
&= \|\mathbf{w}_t - \mathbf{w}^*\|^2 - 2\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) \right\rangle + \left\|\eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t)\right\|^2
\end{aligned}$$

Now taking expectation over the randomness of sampling we get that,

$$\begin{aligned}
\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq \mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 - 2 \underbrace{\mathbb{E}\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) \right\rangle}_{(T_1)} \\
+ \underbrace{\mathbb{E}\left\|\eta \cdot \frac{1}{m} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t)\right\|^2}_{(T_2)}
\end{aligned} \tag{6.24}$$

We will lower bound (T_1) and upper bound (T_2). Let $\xi_t = \{i_t^{(1)}, i_t^{(2)}, \cdots i_t^{(B)}\}$. Then for (T_1) we have,

$$\begin{aligned}
\mathbb{E}\left[\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) \right\rangle\right] &= \mathbb{E}_{\xi_1 \cdots \xi_t}\left[\mathbb{E}_{\xi_{t+1}}\left[\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) \right\rangle \middle| \xi_1 \cdots \xi_t\right]\right] \\
&= \mathbb{E}_{\xi_1 \cdots \xi_t}\left[\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \mathbb{E}_{\xi_{t+1}} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t) \right\rangle\right] \\
&= \mathbb{E}\left[\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \cdot \frac{1}{B} \sum_{j=1}^{B} \nabla L_{\mathcal{D}}(\mathbf{w}_t) \right\rangle\right]
\end{aligned}$$

$$= \mathbb{E}\left[\left\langle \mathbf{w}_t - \mathbf{w}^*, \eta \nabla L_{\mathcal{D}}(\mathbf{w}_t)\right\rangle\right]$$
$$\overset{(a)}{\geq} \mu\eta \cdot \mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 \tag{6.25}$$

where (a) follows from the strong convexity of $L_{\mathcal{D}}$. Note that if $\ell_i(\mathbf{w})$ is $\mu-$strongly convex then even $L_{\mathcal{D}}$ is $\mu-$strongly convex.

Using the same conditioning argument as before and the fact that each sample in a batch is sampled i.i.d. for (T_2) we get that:

$$\begin{aligned}
\mathbb{E}\Big\|\eta \cdot \frac{1}{B}\sum_{j=1}^{B} \nabla \ell_{i_{t+1}^{(j)}}(\mathbf{w}_t)\Big\|^2 &= \eta^2\Big(\frac{1}{B}\mathbb{E}\big[\|\nabla \ell_{i_{t+1}^{(1)}}(\mathbf{w}_t)\|^2\big] + \frac{B-1}{B}\mathbb{E}\|\nabla L_{\mathcal{D}}(\mathbf{w}_t)\|^2\Big) \\
&\overset{(a)}{\leq} \eta^2\left(\frac{2\beta}{B}\Big(\mathbb{E}L(\mathbf{w}_t) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}_i^*)\Big) + \frac{2\beta(B-1)}{B}\Big(\mathbb{E}L_{\mathcal{D}}(\mathbf{w}_t) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*)\Big)\right) \\
&= \eta^2\left(\frac{2\beta}{B}\Big(\mathbb{E}L_{\mathcal{D}}(\mathbf{w}_t) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*) + \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}_i^*)\Big)\right. \\
&\left. + \frac{2\beta(B-1)}{B}\Big(\mathbb{E}L_{\mathcal{D}}(\mathbf{w}_t) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*)\Big)\right) \\
&= \eta^2\left(\frac{2\beta}{B}\Big(\frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}_i^*)\Big)\right. \\
&\left. + 2\beta\Big(\mathbb{E}L_{\mathcal{D}}(\mathbf{w}_t) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*)\Big)\right) \\
&\overset{(b)}{\leq} \eta^2\left(\frac{2\beta}{B}\Big(\frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}_i^*)\Big) + \beta^2\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2\right) \\
&\leq \eta^2 \cdot \beta^2 \cdot \mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 + \frac{2\beta\eta^2}{B}\Big(\underbrace{\frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}^*) - \frac{1}{n}\sum_{i=1}^{n}\ell_i(\mathbf{w}_i^*)}_{\sigma_\ell}\Big) \\
&= \eta^2 \cdot \beta^2 \cdot \mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 + \frac{2\beta\eta^2\sigma_\ell}{B}
\end{aligned} \tag{6.26}$$

where (a) follows from applying Lemma 8 (1) to $\ell_{i_{t+1}^{(j)}}(\mathbf{w}_t)$ and $L_{\mathcal{D}}(\mathbf{w}_t)$ and (b) follows from applying Lemma 8 (2).

Now substituting Equation (6.25) and Equation (6.26) in Equation (6.24), we get that,

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq (1 - 2\eta\mu + \eta^2\beta^2)\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 + \frac{2\beta\eta^2\sigma_\ell}{B} \tag{6.27}$$

Minimizing the coefficient with respect to η we get $\eta^* = \frac{\mu}{\beta^2}$, which gives the following update equation:

$$\mathbb{E}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \leq \Big(1 - \frac{\mu^2}{\beta^2}\Big)\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 + \frac{2\mu^2\sigma_\ell}{B\beta^3} \tag{6.28}$$

Now applying this update recursively we get:

$$\mathbb{E}\|\mathbf{w}_t - \mathbf{w}^*\|^2 \leq \Big(1 - \frac{\mu^2}{\beta^2}\Big)^t\mathbb{E}\|\mathbf{w}_0 - \mathbf{w}^*\|^2 + \frac{2\sigma_\ell}{B\beta}$$

□

Definition 2. *(Forgetting: Single Request) Consider an ERM problem with $L_\mathcal{D}(\mathbf{w}) = \frac{1}{n}\sum_{i\in\mathcal{D}}\hat{\ell}_i(\mathbf{w})$, where $\hat{\ell}_i = \ell_i(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}\|^2$, $\ell_i(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ is a quadratic convex function and $\mathcal{D}$ is a dataset of size n. Let $\mathbf{w}_\mathcal{D} = \mathcal{A}_T(L_\mathcal{D}(\mathbf{w}))$ be the weights obtained after ^{T} steps of algorithm $\mathcal{A}$ (SGD in our case) on $L_\mathcal{D}(\mathbf{w})$. Then given a request to forget a set $\mathcal{D}_f \subset \mathcal{D}$ we apply the following scrubbing procedure:*

$$S(\mathbf{w}_\mathcal{D}, \mathcal{D}, \mathcal{D}_f) \triangleq \mathbf{w}_\mathcal{D} - \Delta\mathbf{w}_{\mathcal{D},\mathcal{D}_f} + z \tag{6.29}$$

where $\Delta\mathbf{w}_{\mathcal{D},\mathcal{D}_f} = \mathcal{A}_\tau(\tilde{L}_{\mathcal{D}-\mathcal{D}_f}(\mathbf{w}))$, τ is the number of steps of $\mathcal{A}$ to minimize $\tilde{L}_{\mathcal{D}_r}(\mathbf{w})$ (where $\mathcal{D}_r = \mathcal{D} - \mathcal{D}_f$). Here

$$\tilde{L}_{\mathcal{D}_r}(\mathbf{w}) = 0.5 \cdot \mathbf{w}^T H_{\mathcal{D}_r}\mathbf{w} - \mathbf{w}^T g_{\mathcal{D}_r}(\mathbf{w}_\mathcal{D}) \tag{6.30}$$

$z \sim \mathcal{N}(0, \sigma I)$ and $H_{\mathcal{D}_r}$ is the hessian on the remaining data ($\mathcal{D}_r$). We compute the residual gradient, $g_{\mathcal{D}_r}(\mathbf{w}_\mathcal{D}) = \nabla L_{\mathcal{D}_r}(\mathbf{w}_\mathcal{D})$ once over entire $\mathcal{D}_r$, while $\mathbf{w}^T H_{\mathcal{D}_r}(\mathbf{w}_\mathcal{D})\mathbf{w}$ is compute stochastically in $\mathcal{A}$.

Definition 3. *(Forgetting: Multiple Requests) Consider that we are provided with a sequence of forgetting request $(\mathcal{D}_f^j)$. Let $\mathcal{D}_j \subset \mathcal{D}$ be the dataset remaining, $\mathbf{w}_{\mathcal{D}_j}$ (or simply $\mathbf{w}_j$) be the weights obtained after j forgetting requests. Then given the $j+1^{th}$ request to forget $\mathcal{D}_f^{j+1} \subset \mathcal{D}_j$, from Equation* (6.29) *in Definition 2 we have that:*

$$\mathbf{w}_{j+1} \triangleq S(\mathbf{w}_j, \mathcal{D}_j, \mathcal{D}_f^{j+1}) = \mathbf{w}_j - \Delta \mathbf{w}_{\mathcal{D}_j, \mathcal{D}_f^{j+1}} + z \tag{6.31}$$

where $z \sim \mathcal{N}(0, \sigma I)$ and $\mathbf{w}_0 = \mathbf{w}_{\mathcal{D}}$ are the weights obtained after training on the entire data $\mathcal{D}$.

Theorem 3. *(Formal) Consider an empirical risk, $L_{\mathcal{D}}(\mathbf{w}) = \frac{1}{n}\sum_{i \in \mathcal{D}} \widehat{\ell}_i(\mathbf{w})$, where $\widehat{\ell}_i(\mathbf{w}) = \ell_i(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}\|^2$, $\ell_i(\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^+$ is quadratic convex function with symmetric $\nabla^2 \ell(\mathbf{w})$, $\beta = max_{i \in \mathcal{D}} \lambda_M(\nabla^2 \hat{\ell}_i(\mathbf{w}))$ is the smoothness constant of $L_{\mathcal{D}}(\mathbf{w})$, $\ell_i(0) \leq M < \infty$ and $\mathcal{D}$ is a dataset of size n. Let $\mathcal{A}$ be SGD with mini-batch size B, $\sigma_\ell > 0$ be some constant associated with SGD, $\gamma = 1 - \mu^2/\beta^2$, τ be the number of steps of $\mathcal{A}$ performed while forgetting and $a \in (0, 1/\gamma^\tau - 1)$. From Definition 3, let $\mathbf{w}_{j-1}, \mathcal{D}_{j-1}$ be the scrubbed weights and the dataset remaining after $j-1$ removal requests, then given a forgetting request to remove $\mathcal{D}_f^j$ ($|\mathcal{D}_f^j| = b$) we obtain the following bound on the amount of information remaining in the weights after using the scrubbing procedure in Definition 3:*

$$I(\cup_{k=1}^j \mathcal{D}_f^k, S(\mathbf{w}_{j-1}, \mathcal{D}_{j-1}, \mathcal{D}_f^j)) \leq \frac{2\gamma^\tau(2 + 1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}\sqrt{\sigma}}\right)^2 + d\right] + \frac{8\sigma_\ell}{B\beta\sigma}}{1 - (1+\alpha)\gamma^\tau}$$

Proof. We follow similar proof technique to [NRS20]. Consider a dataset $\mathcal{D}$ of size n and a forgetting sequence $\mathcal{D}_{Forget} = (\mathcal{D}_f^j)_j$ (batches of data of size b that we want to forget). Let $\mathbf{w}_j, \mathbf{w}_j^{'}, \hat{\mathbf{w}}_j, \mathcal{D}_j$ be the scrubbed weights with noise, scrubbed weights without noise, weights obtained by re-training from scratch using SGD and the remaining dataset after j requests of forgetting. Let n_j be the size of $\mathcal{D}_j$ and $\mathbf{w}_j^* = \text{argmin}_{\mathbf{w} \in \mathbb{R}^d} L_{\mathcal{D}_j}(\mathbf{w})$. Then from Definition 3 we have that

$$\mathbf{w}_j = \mathbf{w}_j^{'} + z \tag{6.32}$$

$$\mathbf{w}_j^{'} = \mathbf{w}_{j-1} - \Delta\mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} \tag{6.33}$$

where $z \sim \mathcal{N}(0, \sigma^2 I)$, $\Delta\mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} = \mathcal{A}_\tau(\tilde{L}_{\mathcal{D}_{j-1}-\mathcal{D}_f^j}(\mathbf{w}))$, $\mathcal{A}_\tau$ is τ steps of SGD and $\tilde{L}_{\mathcal{D}_{j-1}-\mathcal{D}_f^j}(\mathbf{w}) = \frac{1}{2|\mathcal{D}_{j-1}-\mathcal{D}_f^j|}\sum_{i\in\mathcal{D}_{j-1}-\mathcal{D}_f^j}\mathbf{w}^T\nabla^2\ell_i(\mathbf{w}_\mathcal{D})\mathbf{w} - \left\langle\nabla L_{\mathcal{D}_{j-1}-\mathcal{D}_f^j}(\mathbf{w}_\mathcal{D}), \mathbf{w}\right\rangle$. Note that $\hat{\mathbf{w}}_j$ are weights obtained at the end of training with SGD while $\mathbf{w}_j^*$ is the true empirical risk minimizer of $L_{\mathcal{D}_j}(\mathbf{w})$.

After re-training from scratch on $\mathcal{D}_j$ for $\frac{T}{j}$ steps using SGD with mini-batch size of B, we have the following relation for $j \geq 0$, using Theorem 2:

$$\mathbb{E}\|\hat{\mathbf{w}}_j - \mathbf{w}_j^*\|^2 \leq \gamma^{\frac{T}{j}}\|\hat{\mathbf{w}}_{\text{init}} - \mathbf{w}_j^*\|^2 + \frac{2\sigma_\ell}{B\beta} \tag{6.34}$$

where $\gamma = 1 - \mu^2/\beta^2$ and $\hat{\mathbf{w}}_{\text{init}} = 0$ is the training initialization, $\hat{\mathbf{w}}_0$ are the weights obtained by training on $\mathcal{D}_0 = \mathcal{D}$, which is the complete dataset before receiving any forgetting request. When training the linearized model the user weights are initialized to 0 since they correspond to the first order perturbation of the non-linear weights.

Let us select ${}^T_j \geq \tau + \frac{2\log(n_j\mu/4b\beta)}{\log 1/\gamma}$, where $n_j \geq \frac{n}{2}$. Here τ is the number of steps of SGD used to remove one batch (b samples) of data during forgetting. Substituting τ_j in Equation (6.34) we get:

$$\mathbb{E}\|\hat{\mathbf{w}}_j - \mathbf{w}_j^*\|^2 \leq \gamma^\tau\left(\frac{4b\beta}{n_j\mu}\right)^2\|\hat{\mathbf{w}}_{\text{init}} - \mathbf{w}_j^*\|^2 + \frac{2\sigma_\ell}{B\beta} \overset{(a)}{\leq} \gamma^\tau\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + \frac{2\sigma_\ell}{B\beta} \tag{6.35}$$

where (a) follows from $\|\hat{\mathbf{w}}_{\text{init}} - \mathbf{w}_j^*\| \leq \sqrt{\frac{2M}{\mu}}$ and $1/n_j \leq 2/n$.

Now we will compute a similar bound for the weights obtained by applying the forgetting procedure. For any $j \geq 1$, using the scrubbing procedure in Definition 3 we have the following relation:

$$\mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^*\|^2 \leq \frac{\gamma^\tau(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^\tau} \tag{6.36}$$

for $0 < \alpha < 1/\gamma^\tau - 1$. Note that $\mathbf{w}_j$ are the weights after applying the newton update but before adding the scrubbing noise.

From the scrubbing procedure described in eq. (6.32) and eq. (6.33) to solve for $\Delta\mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j}$ we minimize $\hat{L}_{\mathcal{D}_{j-1}-\mathcal{D}_f^j}$ using SGD. While the optimal value of $\Delta\mathbf{w}^*_{\mathcal{D}_{j-1},\mathcal{D}_f^j} = \mathbf{w}_{j-1} - \mathbf{w}_j^*$, thus,

$\Delta \mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} - \Delta \mathbf{w}^*_{\mathcal{D}_{j-1},\mathcal{D}_f^j} = \Delta \mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} - \mathbf{w}_{j-1} + \mathbf{w}_j^* = \mathbf{w}_j^* - \mathbf{w}_j^{'}$. We can bound $\mathbb{E}\|\Delta \mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} - \Delta \mathbf{w}^*_{\mathcal{D}_{j-1},\mathcal{D}_f^j}\|^2$ using Theorem 2 and thus also bound $\mathbb{E}\|\mathbf{w}_j^* - \mathbf{w}_j^{'}\|$.

More precisely we have:

$$\begin{aligned}
\mathbb{E}\|\Delta \mathbf{w}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} - \Delta \mathbf{w}^*_{\mathcal{D}_{j-1},\mathcal{D}_f^j}\|^2 &\overset{(a)}{=} \mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^*\|^2 \\
\mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^*\|^2 &\overset{(b)}{\leq} \gamma^\tau \mathbb{E}\|\Delta \mathbf{w}^{(0)}_{\mathcal{D}_{j-1},\mathcal{D}_f^j} - \Delta \mathbf{w}^*_{\mathcal{D}_{j-1},\mathcal{D}_f^j}\|^2 + \frac{2\sigma_\ell}{B\beta} \\
\mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^*\|^2 &\overset{(c)}{=} \gamma^\tau \mathbb{E}\|\mathbf{w}_{j-1} - \mathbf{w}_j^*\| + \frac{2\sigma_\ell}{B\beta} \qquad (6.37)
\end{aligned}$$

where the (a) follows from the definition of the scrubbing update as shown above, (b) follows from Theorem 2 and (c) follows from $\Delta \mathbf{w}^{(0)}_{\mathcal{D}_{j-1},\mathcal{D}_{f_j}} = 0$. Note that both while training ($^T{}_j$ iterations) and forgetting (τ iterations) we use SGD with a constant step-size. We will use eq. (6.37) along with induction to prove eq. (6.39). For $\mathbf{z} \sim \mathcal{N}(0, \sigma^2 I)$, we have:

$$\mathbb{E}\|\mathbf{z}\|^2 = d\sigma^2 \qquad (6.38)$$

To prove eq. (6.39) we use induction. Lets consider the base case $j = 1$.

$$\begin{aligned}
\mathbb{E}\|\mathbf{w}_1^{'} - \mathbf{w}_1^*\|^2 &\overset{(1)}{\leq} \gamma^\tau \mathbb{E}\|\mathbf{w}_0 - \mathbf{w}_1^*\|^2 + \frac{2\sigma_\ell^2}{B\beta} \\
&\overset{(2)}{=} \gamma^\tau \mathbb{E}\|\hat{\mathbf{w}}_0 - \mathbf{w}_0^* + \mathbf{w}_0^* - \mathbf{w}_1^*\|^2 + \frac{2\sigma_\ell}{B\beta} \\
&\overset{(3)}{\leq} \gamma^\tau (1+\alpha)\mathbb{E}\|\hat{\mathbf{w}}_0 - \mathbf{w}_0^*\|^2 + \gamma^\tau(1+1/\alpha)\mathbb{E}\|\mathbf{w}_0^* - \mathbf{w}_1^*\|^2 + \frac{2\sigma_\ell}{B\beta} \\
&\overset{(4)}{\leq} \gamma^\tau (1+\alpha)\mathbb{E}\|\hat{\mathbf{w}}_0 - \mathbf{w}_0^*\|^2 + \underbrace{\gamma^\tau(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}_{(A)} \\
&\overset{(5)}{\leq} \gamma^\tau(1+\alpha)\underbrace{\left[\gamma^\tau\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + \frac{2\sigma_\ell}{B\beta}\right]}_{\leq (A)} + (A) \\
&\leq \gamma^\tau(1+\alpha)(A) + (A) \\
&\leq \frac{\gamma^\tau(1+\alpha)}{1-(1+\alpha)\gamma^\tau}(A) + (A)
\end{aligned}$$

$$= \frac{\gamma^{\tau}(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^{\tau}}$$

where (1) follows from eq. (6.37), (2) follows from the Definition 3, (3) follows from Lemma 4, (4) follows from Lemma 7 and eq. (6.38), (5) follows from eq. (6.35). Note that in the base case $\mathbf{w}_0 = \hat{\mathbf{w}}_0$ which are the weights obtained by training on the complete data. For Lemma 7, $\mathcal{D}_j$ and $\mathcal{D}_{j-1}$ differ in b samples and $n_j \geq n/2$. Also note that the expectation above is with respect to the randomness of SGD.

Now that we have the base case, for any general $j > 1$ we get:

$$\begin{aligned}
\mathbb{E}\|\mathbf{w}'_j - \mathbf{w}^*_j\|^2 &\overset{(1)}{\leq} \gamma^{\tau}\mathbb{E}\|\mathbf{w}_{j-1} - \mathbf{w}^*_j\|^2 + \frac{2\sigma_\ell}{B\beta} \\
&\overset{(2)}{=} \gamma^{\tau}\mathbb{E}\|\mathbf{w}^{'}_{j-1} - \mathbf{w}^*_j + \mathbf{z}\|^2 + \frac{2\sigma_\ell}{B\beta} \\
&= \gamma^{\tau}\mathbb{E}\|\mathbf{w}^{'}_{j-1} - \mathbf{w}^*_{j-1} + \mathbf{w}^*_{j-1} - \mathbf{w}^*_j + \mathbf{z}\| + \frac{2\sigma_\ell}{B\beta} \\
&\overset{(3)}{\leq} \gamma^{\tau}(1+\alpha)\mathbb{E}\|\mathbf{w}^{'}_{j-1} - \mathbf{w}^*_{j-1}\|^2 + \gamma^{\tau}(1+1/\alpha)\mathbb{E}\|\mathbf{w}^*_{j-1} - \mathbf{w}^*_j + \mathbf{z}\|^2 + \frac{2\sigma_\ell}{B\beta} \\
&\overset{(4)}{\leq} \gamma^{\tau}(1+\alpha)\underbrace{\mathbb{E}\|\mathbf{w}^{'}_{j-1} - \mathbf{w}^*_{j-1}\|^2}_{\leq \frac{(A)}{1-\gamma^{\tau}(1+\alpha)}} + \underbrace{\gamma^{\tau}(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}_{(A)} \\
&\leq \frac{\gamma^{\tau}(1+\alpha)}{1-\gamma^{\tau}(1+\alpha)}(A) + (A) \\
&= \frac{\gamma^{\tau}(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^{\tau}}
\end{aligned}$$

where (1) follows from eq. (6.37), (2) follows from the eq. (6.32), (3) follows from Lemma 4, (4) follows from induction update, Lemma 7 and eq. (6.38). For Lemma 7, $\mathcal{D}_i$ and $\mathcal{D}_{i-1}$ differ in b

samples. The expectation above is with respect to the randomness of SGD and the scrubbing noise **z**.

Thus we have:

$$\mathbb{E}\|\mathbf{w}_j' - \mathbf{w}_j^*\|^2 \leq \frac{\gamma^\tau(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{2\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^\tau} \tag{6.39}$$

Combining eq. (6.35) and eq. (6.39) using Lemma 4 we get:

$$\begin{aligned}
\mathbb{E}\|\mathbf{w}_j^{'} - \hat{\mathbf{w}}_j\|^2 &= \mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^* + \mathbf{w}_j^* - \hat{\mathbf{w}}_j\|_2^2 \\
&\overset{(1)}{\leq} 2\mathbb{E}\|\mathbf{w}_j^{'} - \mathbf{w}_j^*\|^2 + 2\mathbb{E}\|\mathbf{w}_j^* - \hat{\mathbf{w}}_j\|^2 \\
&\leq \frac{2\gamma^\tau(1+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{4\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^\tau} + 2\gamma^T\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + \frac{4\sigma_\ell}{m\beta} \\
&\leq \frac{2\gamma^\tau(2+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{8\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^\tau}
\end{aligned}$$

where (1) follows from Lemma 4.

Thus, we get:

$$\mathbb{E}\|\mathbf{w}_j^{'} - \hat{\mathbf{w}}_j\|^2 \leq \frac{2\gamma^\tau(2+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}}\right)^2 + d\sigma^2\right] + \frac{8\sigma_\ell}{B\beta}}{1-(1+\alpha)\gamma^\tau} \tag{6.40}$$

Since the problem is quadratic, the hessian will be same at all points. From Proposition 1 in [GAS20], we have:

$$\begin{aligned}
I(\cup_{k=1}^j \mathcal{D}_f^k, S(\mathbf{w}_{j-1}, \mathcal{D}_{j-1}, \mathcal{D}_f^j)) &\leq \mathbb{E}\big[(\mathbf{w}_j' - \hat{\mathbf{w}}_j)^T(\sigma^2 I)^{-1}(\mathbf{w}_j' - \hat{\mathbf{w}}_j)\big] \\
&\leq \frac{\mathbb{E}\|\mathbf{w}_i' - \hat{\mathbf{w}}_i\|_2^2}{\sigma^2}
\end{aligned}$$

Note that the expectation in the previous expression is with respect to the randomness in the training algorithm and the forgetting algorithm, plus the set $\mathcal{D}_f^k$. Then using eq. (6.40) with the previous equation we obtain that:

$$I(\cup_{k=1}^{j}\mathcal{D}_f^k, S(\mathbf{w}_{j-1}, \mathcal{D}_{j-1}, \mathcal{D}_f^j)) \leq \frac{2\gamma^\tau(2+1/\alpha)\left[\left(\frac{8b\beta\sqrt{2M}}{n\mu^{3/2}\sigma}\right)^2 + d\right] + \frac{8\sigma_\ell}{B\beta\sigma^2}}{1-(1+\alpha)\gamma^\tau} \tag{6.41}$$

□

CHAPTER 7

Mixed Differential Privacy for Vision

Differential privacy enables us to train machine learning models with statistical indistinguiablility guarantees. It involves restricting the information contained in the weights about every individual sample. Differentially private training generates weights such that removal of any training example from the dataset should not change the weights significantly, thus minimizing the impact each sample has during training. It can be considered as an approximately stricter version of unlearning where instead of only removing the influence of subset of training data, we want to remove the influence of all the training samples. After having explored various forgetting procedures in this chapter we seek to improve differentially private training of image classifiers using the mixed-private setting proposed in chapter 6. Using this setting along with a novel training algorithm we show that we can improve on the state of private training algorithms, thus providing another procedure for unlearning in the mixed-private setting.

More precisely, we introduce *AdaMix*, an adaptive differentially private algorithm for training deep neural network classifiers using both private and public image data. While pre-training language models on large public datasets has enabled strong differential privacy (DP) guarantees with minor loss of accuracy, a similar practice yields punishing trade-offs in vision tasks. A few-shot or even zero-shot learning baseline that ignores private data can outperform fine-tuning on a large private dataset. AdaMix incorporates few-shot training, or cross-modal zero-shot learning, on public data prior to private fine-tuning, to improve the trade-off. AdaMix reduces the error increase from the non-private upper bound from the 167-311% of the baseline, on average across 6 datasets, to 68-92% depending on the desired privacy level selected by the user. AdaMix tackles the trade-off

arising in visual classification, whereby the most privacy sensitive data, corresponding to isolated points in representation space, are also critical for high classification accuracy. In addition, AdaMix comes with strong theoretical privacy guarantees and convergence analysis.

In the subsequent sections we describe AdaMix, show that it's differentially private, provide generalization guarantees, along with extensive empirical evaluation for practical vision tasks.

7.1 Method

Notation. Let $z \in \mathcal{Z}$ be a datum, with $z = (x, y)$ being the feature/label pair; $\mathcal{Z}^* = \cup_{n=0}^{\infty} \mathcal{Z}^n$ is the universe of datasets with arbitrary size. For learning, we have access to a private dataset $D_{\text{pri}} := \{z_1, ..., z_{N_{\text{pri}}}\}$ and a public one $D_{\text{pub}} := \{\tilde{z}_1, ..., \tilde{z}_{N_{\text{pub}}}\}$. The loss function $\ell : \mathcal{Z} \times \mathcal{W} \to \mathbb{R}$ takes data $z \in \mathcal{Z}$ and "weights" $w \in \mathcal{W} \subset \mathbb{R}^d$ to yield the total loss on the private and public dataset $\mathcal{L}_{\text{pri}}(w) := \sum_{i=1}^{N_{\text{pri}}} \ell_i(w)$ and $\mathcal{L}_{\text{pub}}(w) := \sum_{j=1}^{N_{\text{pub}}} \tilde{\ell}_j(w)$, with ℓ_i and $\tilde{\ell}_j$ short-hands for $\ell(w, z_i)$ and $\ell(w, \tilde{z}_j)$ respectively. Notice that $\mathcal{L}_{\text{pri}}$ and $\mathcal{L}_{\text{pub}}$ are sums, not averages, of the loss functions. Their gradients are indicated by $g_i^{\text{pri}}(w) = \nabla_w \ell_i(w)$, $g_j^{\text{pub}}(w) = \nabla_w \tilde{\ell}_j(w)$, $G^{\text{pri}}(w) = \nabla_w \mathcal{L}_{\text{pri}}(w)$, $G^{\text{pub}}(w) = \nabla_w \mathcal{L}_{\text{pub}}(w)$. In addition, we say $N = N_{\text{pri}} + N_{\text{pub}}$ and $\mathcal{L}(w) = \mathcal{L}_{\text{pri}}(w) + \mathcal{L}_{\text{pub}}(w)$.

The average *empirical risk* is written as $\hat{\mathcal{R}}(w) := \mathcal{L}(w)/N$. When the data are drawn from some distribution $\mathcal{D}$, the risk, i.e., generalization error, $\mathcal{R}(w) := \mathbb{E}_{z \sim \mathcal{D}}[\ell(z, w)]$. Our goal is to design a differentially private algorithm that returns $\hat{w}$ which minimizes $\mathcal{L}(\hat{w})$. Its performance is measured by the *excess empirical risk*, i.e., $\hat{\mathcal{R}}(\hat{w}) - \min_{w \in \mathcal{W}} \hat{\mathcal{R}}(w)$; and by the *excess risk* $\mathcal{R}(\hat{w}) - \min_{w \in \mathcal{W}} \mathcal{R}(w)$. In the experiments, the latter is measured by the error on a test set.

7.1.1 Differential privacy

Two datasets D, $D' \in \mathcal{Z}^*$ are said to be *neighboring*, written as $D \simeq D'$, if we can obtain D' from D by *adding or removing* a single data point.

Definition 4 (Differential privacy [DMN06, DR14])**.** *For $\epsilon > 0$ and $\delta \geq 0$, a randomized algorithm*

Algorithm 1: (Projected) Noisy Gradient Descent

Data: Constraint set $\mathcal{W}$, initialization w_1, noise level σ, clipping threshold τ, number of iteration T, weight decay parameters λ and w_{ref}, learning rates η_t.

Result: $w_1, ..., w_{T+1}$

for $t = 1, \ldots, T$ **do**

$G_t = \sum_{i=1}^{N_{\text{pri}}} \text{clip}_\tau(g_i(w_t))$;

$n_t \sim N(0, \sigma^2\tau^2 I)$;

$w_{t+1} \leftarrow w_t - \eta_t(G_t + n_t + \lambda(w_t - w_{\text{ref}}))$;

$w_{t+1} \leftarrow \Pi_{\mathcal{W}}(w_{t+1})$;

end

$\mathcal{M}$ is (ϵ, δ)-differentially private if for any neighboring datasets $D \simeq D'$ and any measurable $S \subseteq \text{range}(\mathcal{M})$,

$$\mathsf{Pr}[\mathcal{M}(D) \in S] \leq e^\epsilon \cdot \mathsf{Pr}[\mathcal{M}(D') \in S] + \delta.$$

In our context, $\mathcal{M}$ is the randomized learning algorithm, and $\mathcal{Y}$ is the space of model weights. This condition bounds the maximum amount of information that an attacker, having access to the weights of the model, can extract about one of the training samples. The parameter ϵ is referred to as the *privacy parameter*: lower values of ϵ carry better privacy guarantees, but also make it harder to train an accurate model, as it reduces the amount of information about the training samples that can be represented via the parameters of the model. The goal of a good DP training algorithm A is to train an accurate model while satisfying the (ϵ, δ)-privacy constraint selected by the user where, ideally, $\epsilon \approx 1$ and $\delta = o(1/n)$.

NoisyGD. NoisyGD is defined in Algorithm 1. We now show that NoisyGD is differentially private and that, in the convex case, it converges to the optimal solution.

Proposition 4 (NoisyGD is (ϵ, δ)-DP, informal)**.** *Algorithm 1 with parameter T, σ^2 such that $\rho := \frac{T^2}{2\sigma^2}$ satisfies the $(\epsilon, \delta(\epsilon))$-DP, where $\delta(\epsilon) := \Phi(\frac{\mu}{2} - \frac{\epsilon}{\mu}) - e^\epsilon \Phi(-\frac{\mu}{2} - \frac{\epsilon}{\mu})$, $\mu = \sqrt{2\rho}$ and Φ is the*

CDF of the normal distribution.

For any prescribed privacy parameter (ϵ, δ), it suffices to choose this ρ parameter, thus from here onward we will use ρ as the privacy parameter of interest (note that, asymptotically, $\rho \asymp \min\{\epsilon, \epsilon^2/\log(1/\delta)\}$).

Mixed private learning. We define the problem of differentially private learning when we have access to an additional public dataset *mixed private learning*. We say the algorithm $\mathcal{M}$ (now taking two arguments D_{pri} and D_{pub}) satisfies (ϵ, δ)-DP if $\mathcal{M}(\cdot, D_{\text{pub}})$ satisfies (ϵ, δ)-DP (Def. 4) for every fixed D_{pub}. [ABM19] shows that a small public dataset from the same distribution allows one to privately PAC-learn any classes with bounded VC-dimension, but with an inefficient algorithm. We focus on designing practical DP algorithm that can effectively use both public and private data under the setting of convex empirical risk minimization [CMS11] and convex fine-tuning of deep models [AGR21b].

7.1.2 AdaMix

Model. Following [TB21], we use a linear logistic regression model trained on top of the last layer features of an ImageNet pre-trained network. This reduces the expressivity of the model compared to training a full network. However, the significant reduction in number of parameters makes it a better choice DP algorithms and recent deep networks allow for competitive performance on most tasks even when training only the last layer. We normalize the features before feeding them to the logistic model, which both improves the accuracy and bound the gradients at each step.

Initialization and regularization. The choice of initialization is normally of minor concern for logistic regression: since the problem is convex we will converge to a global optimum regardless of the initialization. L2 regularization (weight decay), when employed, is usually centering at 0 by default. However, centering the regularizer is critical in differential privacy, as we prove in the following result:

Theorem 5 (Convergence of NoisyGD on strongly convex problems)**.** *Assume $\mathcal{W}$ is a convex set satisfying $\sup_{w\in\mathcal{W}} \|w\| \leq B$. Let $w_1, ..., w_{T+1}$ be the parameter vectors returned by running Algorithm 1 on the following regularized loss $J(w) = \mathcal{L}(w) + \frac{\lambda}{2}\|w - w_{ref}\|^2$ with $w_1 = w_{ref} \in \mathcal{W}$, learning rate $\eta_t = \frac{2}{\lambda(t+1)}$ and $\lambda = \sqrt{\frac{\rho}{2\|w^* - w_{ref}\|dL^2}}$. Then, the ensemble solution:*

$$\bar{w} = \frac{2}{T(T+1)} \sum_{t=1}^{T} t\, w_t$$

obeys the bound

$$\mathbb{E}[\hat{\mathcal{R}}(\bar{w})] - \hat{\mathcal{R}}(w^*) \leq \frac{4(N\tau+\lambda B)^2}{\lambda T N} + \frac{\sqrt{d}\tau\|w^* - w_{ref}\|}{\sqrt{2\rho}N}$$

for any $w^* \in \mathcal{W}$, where ρ is the privacy parameter of the algorithm.

Note that as $T \to +\infty$ the first term vanishes and the second term matches the information-theoretic limit for privately solving general convex ERM problems when $w_{\text{ref}} = 0$ [BST14]. However, this can improve if we choose w_{ref} according to the public data such that w_{ref} is closer to w^* than 0. This motivated us to propose choosing w_{ref} by training on the public data. Then, we proceed to fine-tune the model on the concatenation of private and public data, using Noisy-GD while preserving (ϵ, δ)-DP.

Provable benefits of public data access. Next we provide a theoretical guarantee for the above approach under standard statistical learning assumptions.

Theorem 6 (Generalization with access to public data, informal)**.** *Assume that private and public data are drawn i.i.d. from the same distribution $\mathcal{D}$ and that $R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(w,(x,y))]$ is c-strongly convex in $\mathcal{W}$. Let w_{ref} be obtained by training for one epoch on the public dataset and let $\bar{w}$ be as in Thm. 5, then at the limit when T is sufficiently large, the* excess risk *obeys*

$$\begin{aligned}&\mathbb{E}[\mathcal{R}(\bar{w})] - \mathcal{R}(w^*)\\ \leq &\frac{4\sqrt{d}L^2}{c\sqrt{N_{pub}}N\sqrt{2\rho}} + \textit{Gen}(\bar{w}, N) + \textit{Gen}(w^*, N),\end{aligned}$$

where $w^* = \arg\min_{w\in\mathcal{W}} \mathcal{R}(w)$ and $\textit{Gen}(w, N) := \left|\mathbb{E}[\mathcal{R}(w) - \hat{\mathcal{R}}(w)]\right|$ is the expected generalization gap of (a potentially data-dependent) w.

To the best of our knowledge, Theorem 6 is the first demonstrated claim that access to even a very small public dataset can improve private learning in the convex ERM setting. We note that the generalization bound decomposes in a first term, which describes the generalization cost incurred by using DP, and a second part which is common to non-private learners. Interestingly, in the Appendix we show that in MixDP the cost of privacy asymptotically vanishes even if the percentage of public data becomes neglegible (i.e., when $N_{\text{pub}} \to \infty$, $N_{\text{pub}}/N \to 0$).

Besides providing a good initialization, access to a small public dataset also enables ways to further improve Noisy-GD in practice, as we will now see.

Adaptive threshold. Since the norm of the gradients changes significantly during training, a fixed clipping threshold τ is suboptimal throughout training. [WZ20] suggests to tune this threshold adaptively using public data. However, the norm of the average gradient of the public sample, may not yield good clipping thresholds for the *per-sample* gradients. Instead, we find the following adaptive threshold to be more robust:

$$\tau = \text{quantile}_{90}(\{\|g_i^{\text{pub}}(w)\|\}_{i=i}^{N_{\text{pub}}}),$$

that is, we take the 90-th quantile of the norm of the public gradients, where $g_i^{\text{pub}}(w) = \nabla_w \tilde{\ell}_i(w)$ is the gradient of the i-th public sample. Then the clipped gradient we use is

$$\tilde{g}_i^{\text{pri}}(w) = \frac{g_i^{\text{pri}}(w)}{\|g_i^{\text{pri}}(w)\|} \min(\|g_i^{\text{pri}}(w)\|, \tau).$$

This quantile rule is also used by [ATM21], but we differ in the use of public data to determine τ, rather than allocating part of the privacy-budget for releasing the quantile. It was established in [CWH20] that gradient clipping has the effect of Huberization of the loss function in the GLM cases. The convergence to the optimal solution of a modified objective function is guaranteed if we fix the clipping parameter. Our approach on the adaptive clipping has the effect of selecting the hyperparameter adaptively in a homotopy style algorithm, which shows further empirical benefits.

Adaptive subspace projection. From Thm. 6, the utility of a DP algorithm decreases with the number of parameters d of the model. In view of this, we do not train the whole network but rather

use it as a feature extractor and only train a linear model on top. However, for modern vision architectures, even the number of parameters of the final layer is in the order of tens of thousands. Therefore, we use an adaptive mechanism to project private gradients onto a low-dimensional subspace before adding noise. More precisely,

$$G^{\text{pub}}(w) = \nabla_w L_{D_{\text{pub}}}(w) = \sum_{i=1}^{N_{\text{pub}}} \nabla_w \ell(\tilde{z}_i, w)$$

be the total gradient of the public dataset D_{pub}. Since we are using a linear logistic model, $G^{\text{pub}}(w)$ is a $D \times C$ matrix where D is the number of features and C is the number of classes. Consider the SVD decomposition:

$$G^{\text{pub}}(w) = USV.$$

We now project each (clipped) private gradient matrix $\tilde{g}_i^{\text{pri}}$ on the top P components, and obtain a matrix $\hat{g}_i^{\text{pri}} = U^t \tilde{g}_i^{\text{pri}}$ of size $P \times C$. Other methods projecting the gradient using public data [ZWB21, YZC21, KRR20] do not compute the SVD of the total gradient matrix, as noted in Sec. **??**. Rather, use the vectorized *per-sample* gradients of the public examples to compute the principal components to project. Since, in vision, the size of matrix of all per-sample gradients has a much larger size $N \times FC$ (rather than our $F \times C$), several approximations would be required to carry out this strategy.

Final algorithm. The final gradient update step at time t is:

$$w_{t+1} \leftarrow w_t - \eta_t(G^{\text{pub}} + U\hat{G}^{\text{pri}} + \lambda \cdot w_t)$$

where

$$\hat{G}^{\text{pri}} = \textstyle\sum_{i=1}^{N_{\text{pri}}} U^T \tilde{g}_i^{\text{pri}} + n_t.$$

and $n_t \sim \mathcal{N}(0, \sigma^2\tau^2 I_C)$. We then train for T steps. The pseudo-code of algorithm is given in Algorithm 2.

Proposition 7. *Algorithm 2 with parameter T, σ^2 such that $\rho := \frac{T^2}{2\sigma^2}$ satisfies the $(\epsilon, \delta(\epsilon))$-DP of a Gaussian mechanism in Theorem 8 [BW18] with parameter $\mu = \sqrt{2\rho}$.*

Note that the privacy guarantee is the same as long as the ratio $\sigma/\sqrt{T}$ remains constant, which gives a degree of freedom in choosing σ (hence T). Higher values of σ requires more training steps T, but, by Thm. 5, it also ensures better convergence (which we validate this empirically in Sec. 7.2). Hence, the user should generally pick the largest σ allowed by their computational budget.

Algorithm 2: AdaMix training algorithm.

Data: Public dataset D_{pub}, private dataset D_{pri}, privacy parameter (ϵ, δ), noise variance σ, learning rate η, L_2 reg. coefficient λ

Result: $w = A(S)$

$T_{\text{max}} \leftarrow \text{Calibrate}(\epsilon, \delta, \sigma)$;

$w \leftarrow \text{Pretrain}(S_u)$;

for $t = 1, \ldots, T_{max}$ **do**

$\tau_t \leftarrow \text{quantile}_{90}\left(\{\|g_i^{\text{pub}}(w)\|\}_{i=1}^{N_{\text{pub}}}\right)$;

$U, S, V \leftarrow \text{SVD}(G_{\text{public}}(w))$;

$n_t \sim N(0, \sigma^2 \tau_t^2 I)$;

$\tilde{g}_i^{\text{pri}}(w) \leftarrow \frac{g_i^{\text{pri}}(w)}{\|g_i^{\text{pri}}(w)\|} \min(\tau_t, \|g_i^{\text{pri}}(w)\|)$;

$\tilde{G}^{\text{pri}}(w) = \sum_{i=1}^{N_{\text{pri}}} U^T \tilde{g}_i^{\text{pri}}(w) + n_t$;

$w \leftarrow w - \eta(G^{\text{pub}}(w) + U\tilde{G}^{\text{pri}}(w) + \lambda \cdot w)$;

end

7.1.3 Textual side information

CLIP [RKH21] is an embedding trained so that the representation of an image is close to the representation of its textual description. This enables zero-shot learning, i.e., initializing an high-performance image classification model using only the name of the labels or a textual description of the classes, which is generally public. AdaMix can be easily adapted to work in this multi-modal MixDP setting, where the public data comes from a different modality. First, we initialize a linear logistic model using the names of the classes: Let c_i be the text associated to the i-th class, the

	Public Shots	Non-Private (paragon)	Only-Public (baseline)	Fully-Private	PPGD [WZ20]	AdaMix	Fully-Private	PPGD [WZ20]	AdaMix
				$\epsilon = 1$			$\epsilon = 3$		
Ox. Flowers [NZ06]	2	12.34	40.16	95.12	40.99	**39.48**	81.42	39.22	**36.11**
CUB-200 [WBM10]	2	31.97	64.01	96.79	64.20	**63.58**	81.61	61.59	**59.15**
Stanf. Dogs [KJY11]	2	10.08	16.58	33.87	16.42	**15.38**	15.93	15.61	**12.72**
MIT-67 [QT09]	5	25.17	43.58	69.55	42.02	**40.46**	42.99	39.60	**33.03**
Oxford Pets [PVZ12]	5	7.17	12.83	26.02	12.85	**11.54**	12.37	11.15	**9.53**
Caltech-256 [GHP]	5	14.64	24.88	60.72	24.61	**23.74**	27.15	23.71	**20.85**

Table 7.1: **Mixed privacy, full privacy and AdaMix.** We report the result (test errors) of different methods on a diverse set of vision tasks (see text for description). Surprisingly **Only-Public**, which throws away private data and only trains a model on the small amount of public data outperforms **Fully-Private**, which trains on all data as if it was private. **AdaMix** is instead able to effectively use both the private and public data at the same time, and achieves a significant improvement even at relatively small value of ϵ (e.g., 10% improvement on MIT-67 w.r.t. Only-Public). Instead, the closest method to us, **PPGD** does not significantly improve over the Only-Public baseline.

weight matrix is [RKH21]:

$$W = \beta[\text{enc}(c_1), \ldots, \text{enc}(c_C)],$$

that is, each row is the normalized encoding of a label scaled by a parameter β (we use $\beta = 100$). Starting from this public initialization, we train with AdaMix on image data.

7.2 Results

Setting. Unless otherwise specified, we train a linear logistic model on top of the last layer features of a ResNet-50 [HZR16] pre-trained on ImageNet [DDS09b]. For NoisyGD, we initalize the logistic weights to zero, fix $\sigma = 20$ and $\delta = 10^{-5}$. We train with $\lambda = 1e - 2$, weight decay centered at zero, learning rate $\eta \in \{0.0005, 0.001, 0.0025, 0.005\}$, and select the best learning rate for each value of ϵ. For AdaMix, we initialize the weights by training on the public data until convergence. We use the Gaussian mechanism implementation of `autodp`[1] to calibrate the parameters of the training

[1] `https://github.com/yuxiangw/autodp`

algorithm, in particular $T_{\max}$, in order to achieve the desired (ϵ, δ)-DP requirement.

Datasets. We test our algorithm on the following vision classification datasets commonly used in transfer learning (see Tab. 7.1). We split each training dataset into a "public" dataset consisting of N-samples per class (see Tab. 7.1), and consider the remaining training data as private. In all cases, the public data is less than 10% of the private data.

Baselines considered. We compare AdaMix with the following baselines: **(Fully-Private)** Ignore the MixDP setting and simply train on all data as if it was private, starting form a zero initialization and using NoisyGD; **(Only-Public)** Train a standard non-private model using only the few public samples and discarding the private ones; **(PPGD)** A version of PPSGD [WZ20], but trained with NoisyGD rather than DP-SGD for easier comparison. Unlike AdaMix, PPGD has several hyper-parameters: we manually searched for the best combination to ensure the best comparison; **(NGD)** An ablated version of AdaMix which uses fixed hyper-parameters rather than the adaptive components.

7.2.1 Experiments

Mixed Privacy vs Full Privacy. In Table 7.1 we see that, perhaps surprisingly, Only-Public is always significantly better than Fully-Private even if it actually throws away most of the available data. This suggests that, in vision, collecting a small amount of public data may be preferred to collecting a much larger amount of private data (Fig. 7.2). However, we see that AdaMix is able to effectively use both the private and public data at the same time, and achieves a significant improvement even at relatively small value of ϵ (e.g., 10% improvement on MIT-67 w.r.t. Only-Public at $\epsilon = 3$). Conversely, the method closest to us, PPGD, does not significantly improve over the Only-Public baseline even after our changes to adapt it better to vision tasks.

Ablation study and finer comparison. In Fig. 7.1 (left and center) we plot the average accuracy obtained by various methods and ablations of AdaMix for different privacy parameters ϵ. Both

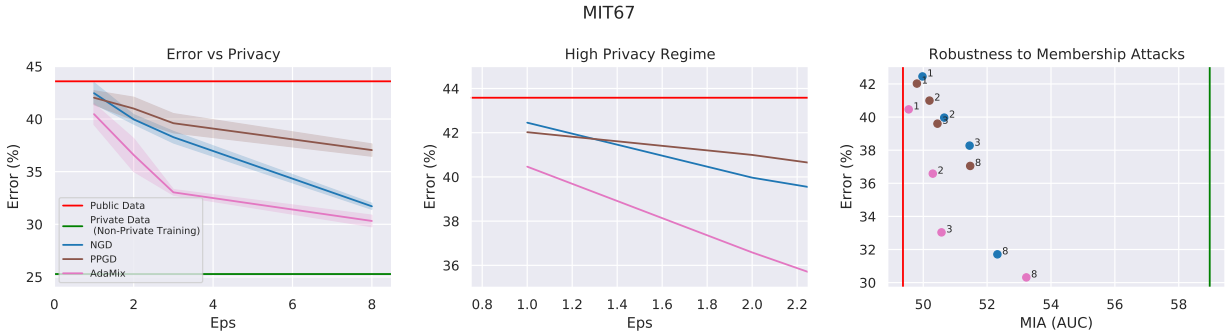

Figure 7.1: **Test error vs privacy for several methods. (Left)** On MIT-67, we show the test error obtained by different methods at different privacy parameters ϵ. The top horizontal red line shows the perfectly private Only-Public baseline (not using the private data at all). The bottom green line shows the non-private paragon (training on all data as if they where public). By changing the privacy parameter ϵ, we can interpolate between the two extrema. We see that AdaMix performs uniformly better than other methods, and can use private and public data together to obtain significant boost at all ϵ. **(Center)** Same as the before, but we zoom in to show the behavior of the different solutions in the low-ϵ regime. **(Right)** For the same algorithms, we plot the robustness to adversarial attacks (measured by AUC) vs test error obtained using different values of ϵ (annotated near each point). AdaMix obtains the best trade-off between accuracy and robustness.

AdaMix and the ablated NGD are able to improve over the Only-Public baseline. Interestingly, this holds true even when considering relatively low privacy parameters $\epsilon \approx 1$ which is usually considered too restrictive to train in a fully private setting (as seen in Tab. 7.1). On the other hand, thanks to the MixDP training we can obtain high-accuracy models even at restrictive privacy settings. We note that AdaMix performs uniformly better than other methods at all privacy regimes. From the ablation of AdaMix, we see that initializing the model with the public data has the largest impact, followed by adaptive clipping and adaptive projection.

Membership attacks. Differential privacy is a very strong requirement, and is it often assumed that using higher value of ϵ may still provide valuable defense to attacks [SSS17, CLE19, CTW21, RRL18, BGR19, ZH20, SDO19, LAG19, MZH19] even if the theoretical guarantees are weak. Conversely, [JUO20] suggests using attacks to give a lower-bound on the privacy parameter of the algorithm. In Fig. 7.1 (right) we plot the Area Under Curve of a membership attack executed against models for different values of ϵ. We use a thresholding based membership attack as proposed in [SDO19]. The loss for samples in the private training set are labelled as 0 and test samples are labelled as 1. Then we simply compute the AUC for this. We find that the sorting of the algorithms does not change: algorithms with better theoretical bounds are also more robust to attacks.

Performance Boost and size of public data. We hypothesize that, when enough public data is present, the contribution of using private data may be less noticeable. To test this, we define the Performance Boost of a Mixed Privacy method as $\text{PB} = (\text{err}_{\text{public}} - \text{err}_{\text{paragon}})/(\text{err}_{\text{DP}} - \text{err}_{\text{paragon}})$, where $\text{err}_{\text{paragon}}$ is the error obtained by training on non-private model on both public and private data, $\text{err}_{\text{public}}$ is the error obtained by training only on the public data, and err_{DP} is the error obtained by using training a mixed privacy model. This measures how well we can learn from the private data in the MixDP setting compared to the non-private upper-bound. In Fig. 7.2 we plot the performance boost for different sizes of the public set. We observe that the performance boost derived of using mixed privacy is very high when the public data is scarce, disappears when the amount of public data is comparable or larger than the private data.

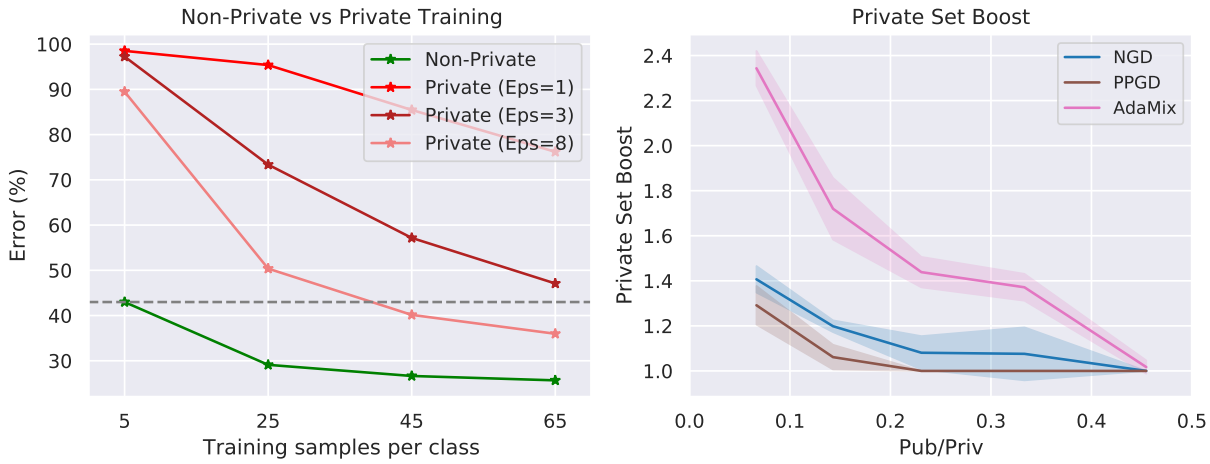

Figure 7.2: **(Left) Trade-off between public and private examples.** We show the accuracy reached on MIT-67 using N samples per class (x-axis) when they are public (green) or private (in red for different values of ϵ). Fully private training requires as much as 10 times more samples. This leaves users with a trade-off between collecting more private data or, if possible, a smaller amount of public data. **(Right) Performance boost using a private set for different methods.** We plot the Performance Boost (PB) for different DP methods as the ratio of public to private samples changes (see text for definition). We see that AdaMix achieves the best performance boost. As expected, when the public set is relatively large, the boost from adding private data decreases.

Multi-modal initialization. In Fig. 7.4 we compare CLIP models initialized with (1) a random initialization, (2) a zero-shot initialization based on the public label names (Sec. 7.1.3), and (3) using few public image samples. While the latter performs the best at a given privacy level ϵ, the zero-shot initialization still improves significantly over the random initialization, thus showing that even just small amount of textual information (which is often available) can be used to significantly boost accuracy. Multi-modal models may also learn a better lower-dimensional representation of the data that is more amenable to DP. In Fig. 7.4 we compare AdaMix using an ImageNet pretrained ResNet-50 and using CLIP. This is not a direct comparison, but we see a significant performance boost by using CLIP at a given privacy level ϵ, suggesting that the move to multi-modal model may be a logical next step for DP.

Effect of differential privacy on individual samples. DP is a worst-case guarantee which may penalize the accuracy of a model on unusual data, or data on the long-tails of the distribution, which

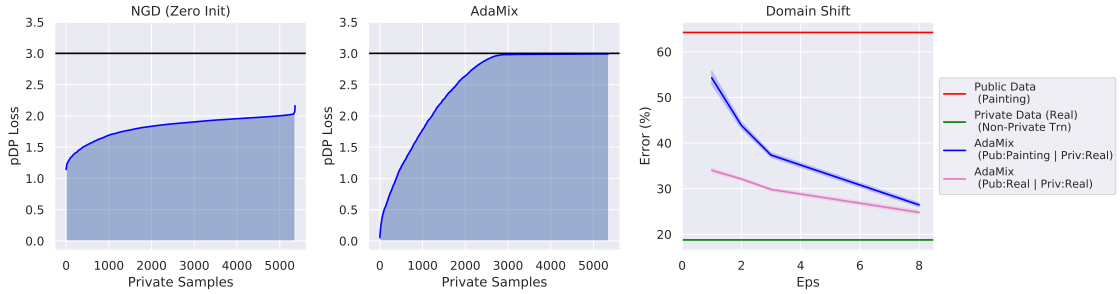

Figure 7.3: **(Left and center) Effect of public data on per-instance DP.** For $\epsilon = 3$, we plot the sorted pDP scores for each sample in the dataset with Fully-Private training **(left)** and with AdaMix **(center)**. AdaMix makes the the target ϵ (horizontal line) closer to pDP value used by most samples (blue curve). **(Right) MixDP and domain-shifts.** We plot the performance of AdaMix when (purple curve) the public data come from the same domain as the test (real images), and (blue curve) from different domains (public images are paintings). We see that AdaMix use the private data from the target domain (real images) to adapt the solution to the target task. This significantly improves over using only the public data from the wrong domain (red line) and for large ϵ approaches the non-private limit (green line).

however may play an important role for generalization in vision [Fel20, FZ20]. MixDP may ease the problem, since it allows to collect public data to ensure that each sub-population is sufficiently covered, lessening the accuracy cost of making that data private. We show this effect using the per-instance DP (pDP) analysis of [Wan19], which measures the privacy loss ϵ' incurred by each individual z during the private training. This allows us to provide a finer analysis of different methods by comparing the histogram of pDP loss ϵ for each individual in the dataset D on top of the worst case DP bounds, which we do in Fig. 7.3 (see Appendix for details and theoretical analysis). We observe that, when using AdaMix, the pDP losses of most samples are the same, and is close to the maximum privacy budget $\epsilon = 3$, indicating that we are using the prescribed privacy budget more effectively and more uniformly across samples, thereby improving utility.

Domain shift. Often, the private and public data may come from slightly different domains. To test the robustness of AdaMix to domain shifts, we use the DomainNet dataset, which aims to classify objects in different domains. In particular, we train a model to classify objects in painting, using however public data coming from real pictures (while the private data is in the painting domain). In Fig. 7.3 (right), we show that, even if the public comes from a different domain, AdaMix is still

able to use the public data to significantly improve performance of private training.

Effect of σ. Outside of the learning rate η, the main other sensitive hyper-parameter of AdaMix is the noise variance σ. From Prop. 5 we expect that the best accuracy will be obtained using a large σ and training for a respectively larger amount of epochs T. In the Appendix, we plot the test error as a function of σ and we see that indeed, a larger σ give better results at the expense of longer training time.

7.3 Discussion

We study Mixed Differential Privacy learning in computer vision, and show that in this setting differential privacy can be used to provide high accuracy models while respecting a given privacy parameter. To do this, we introduce AdaMix, an adaptive differential privacy mechanism that uses the public data to initialize and compute, at each step, an optimal subspace and clipping threshold. We also show that multi-modal vision and text can significantly improve accuracy under a privacy constrain.

Limitations. In our analysis we train a linear model on pre-trained features, rather than fine-tuning the whole network. This is common in DP, since fine-tuning multiple layers may not significantly improve the accuracy while the additional parameters negatively affects the privacy bounds. This limitation is partially offset by new models that are trained to have generic and transferable last-layer features without the need of additional fine-tuning. We discuss several theoretical bounds that guide us in understanding the behavior of the algorithm. However, these bounds refer to a slighlty simplified version of AdaMix (see Appendix for details) and use hyper-parameters that are asymptotically optimal, but may not be practical.

Conclusions. Differentially Private models in computer vision often cannot reach the same level of performance as non-private model for realistic privacy parameters. Using AdaMix in the MixDP learning setting, we have shown that, assuming the presence of a small amount of public data,

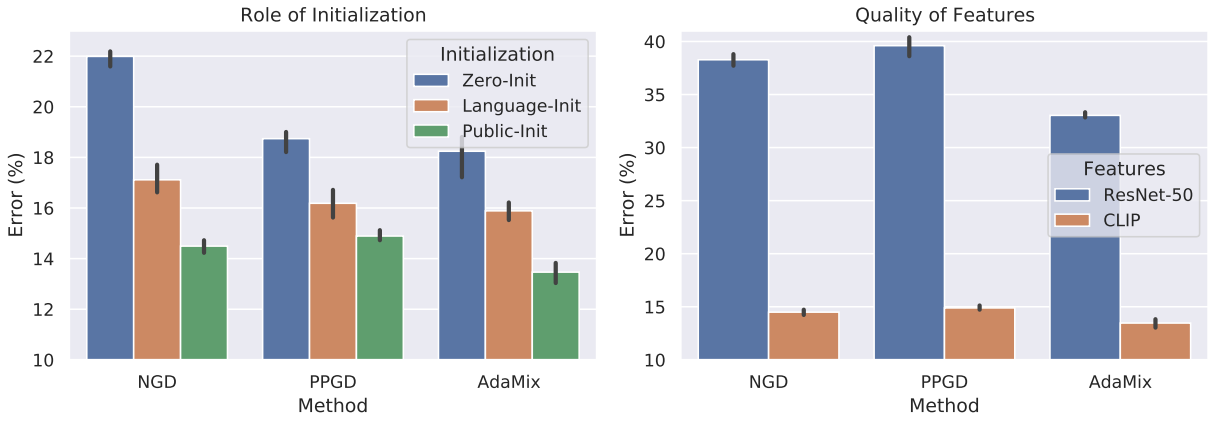

Figure 7.4: **(Left) Multi-modal initialization is better than zero initialization.** We compare the test accuracy on MIT-67 of a CLIP model trained with $\epsilon = 3$ using different initialization strategies. We observe that initializing the logistic weights to zero leads to the worst results under a given privacy parameter. Using the public label names to initialize the weights (language initialization) performs significantly better. However, using the few available public images to initialize the weights still outperforms the other choices. **(Right) Multi-modal models.** We compare the performance of ResNet-50 model, and a CLIP model. We see that CLIP performs significantly better under a given privacy parameter. While the ResNet-50 model and the CLIP model are not directly comparable due to different pre-training, this further reinforces that stronger pre-training can indeed benefit the privacy setting and that the feature space learned by multi-modal models may be better suited for privacy than standard vision networks.

accurate networks can be produced that reach better accuracies than fully private training without compromising the privacy of the data. We hope that further research in this field may help the adoption of private models in more areas of computer vision.

7.4 Appendix

In the supplementary material we perform, (1) Ablation studies analyzing different components of our algorithm (section 7.4.1), (2) tuning the adaptive clipping threshold can further improve AdaMix (7.4.2), (3) show that using a larger σ (training for longer) is better (7.4.3), (4) provide additional experimental results (7.4.4), (5) provide details experimental details (7.4.5, 7.4.6), and (6) proofs for all the theoretical results presented in the paper (section 7.5, 7.6, 7.7).

7.4.1 Ablation Studies

In fig. 7.5, we ablate the two components of our algorithm, namely, Subspace Projection and Adaptive Clipping. We consider the Noisy-GD (NGD) as the baseline and analyze the improvement provided by the two components separately and finally in combination (AdaMix). We observe that adaptive clipping is the key component of the AdaMix algorithm. We show that using the two components together performs the best across multiple datasets. In the next section we show that further tuning the adaptive clipping threshold can improve the performance even more on some datasets. In fig. 7.6, we plot the relative reconstruction error for the private gradients on the public gradient subspace and a random subspace. We observe that the reconstruction error on the public gradient subspace is at least half the random subspace throughout training, which suggests the importance of the public gradients for AdaMix.

7.4.2 Effect of AdaMix clip threshold

We plot the effect of adaptive clipping threshold (percentile) on the test error for AdaMix (we use 90 percentile in all of our experiments) in fig. 7.7. However for CUB-200 and Caltech-256 tuning it to 75 percentile works better indicating that AdaMix has more scope for improvement. Note, however that using 90 percentile for CUB-200 and Caltech-256 still outperforms all the baselines.

7.4.3 Longer training with more noise

Higher values of σ requires more training steps, however, theorem 5 shows that it leads to better convergence, which leads to better generalization. In fig. 7.8 we see that indeed, at the same level of privacy, using a large of σ significantly reduces the test error compared to using a smaller σ.

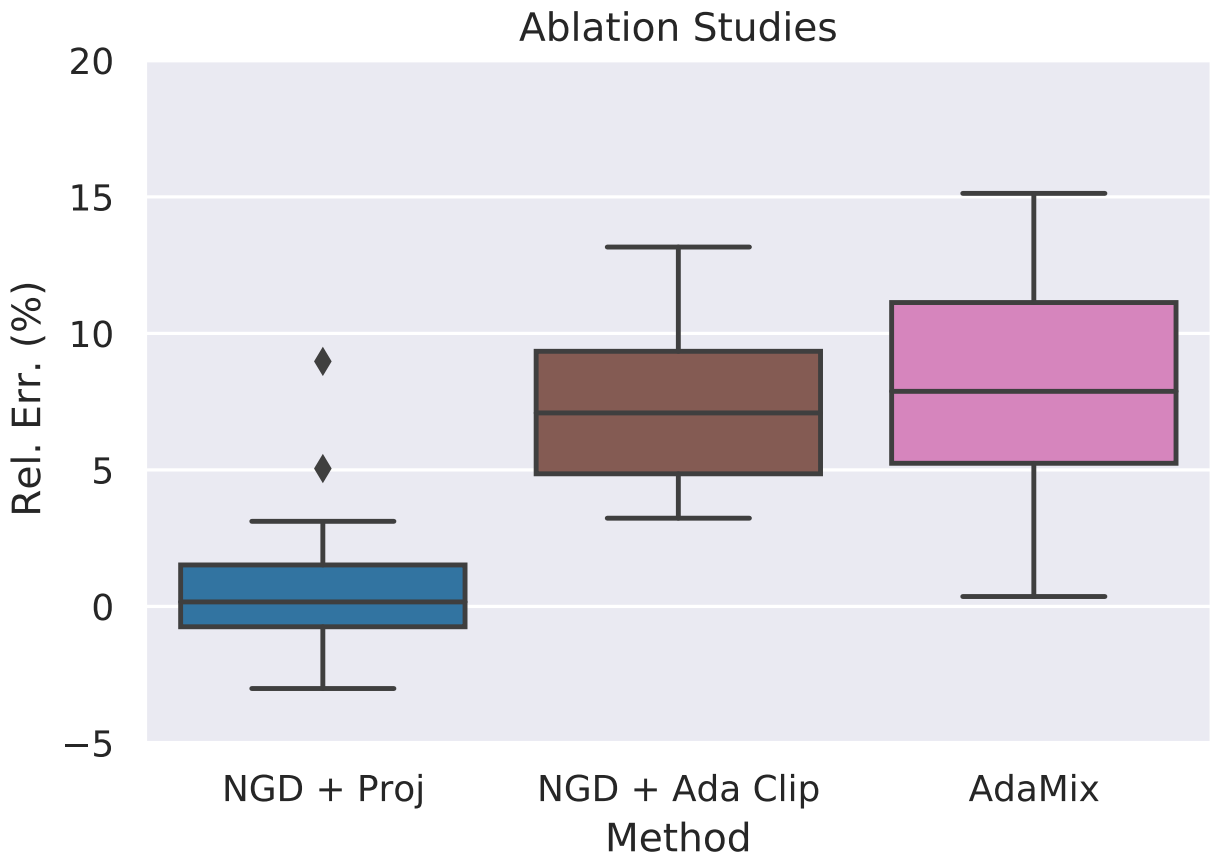

Figure 7.5: **Ablation Studies** Box plot showing the relative decrease in the test error across multiple datasets when we add Subspace Projection, Adaptive Clipping and Subspace Projection + Adaptive Clipping (AdaMix) to NGD. We observe that adaptive clipping provides more improvement compared to subspace projection, and the combination of the two works the best across 6 datasets.

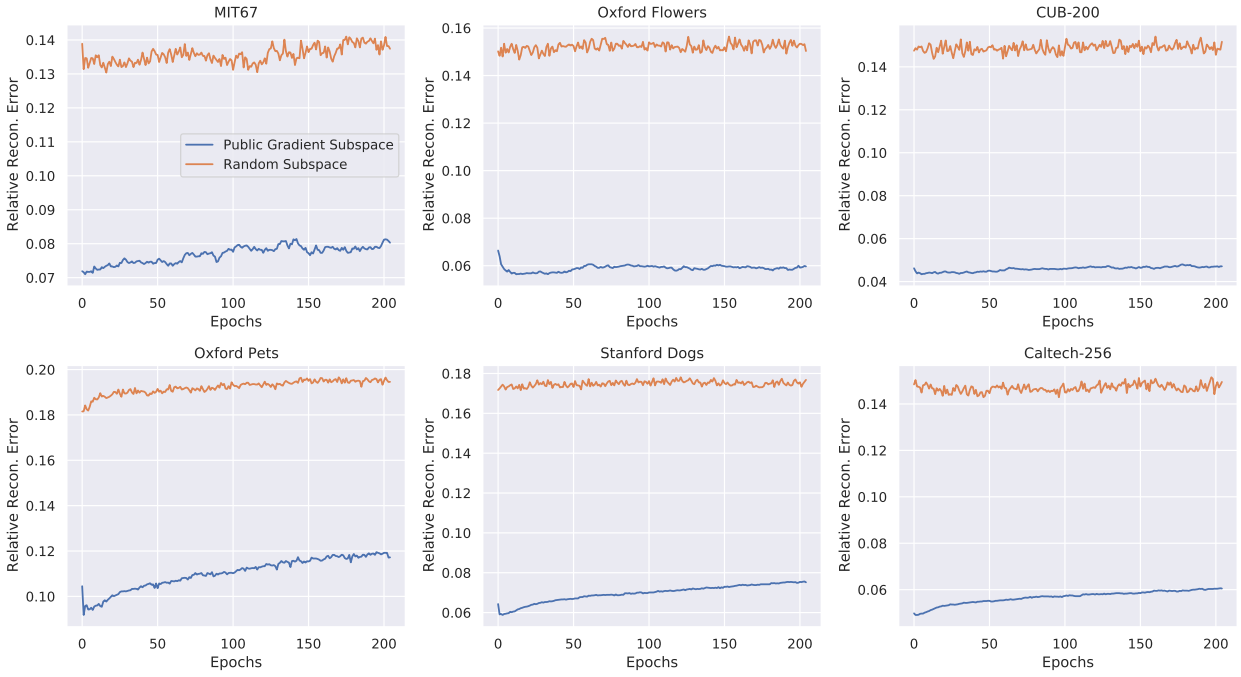

Figure 7.6: **Random Subspace Projection** We plot the relative reconstruction error of the private gradients when projecting on the subspace spanned by the public gradients or on a random subspace during training (using AdaMix and $\epsilon = 3$). The reconstruction error when projecting on the random subspace is generally more than twice the error obtained projecting on the subspace of public gradients, highlighting the importance of using the latter.

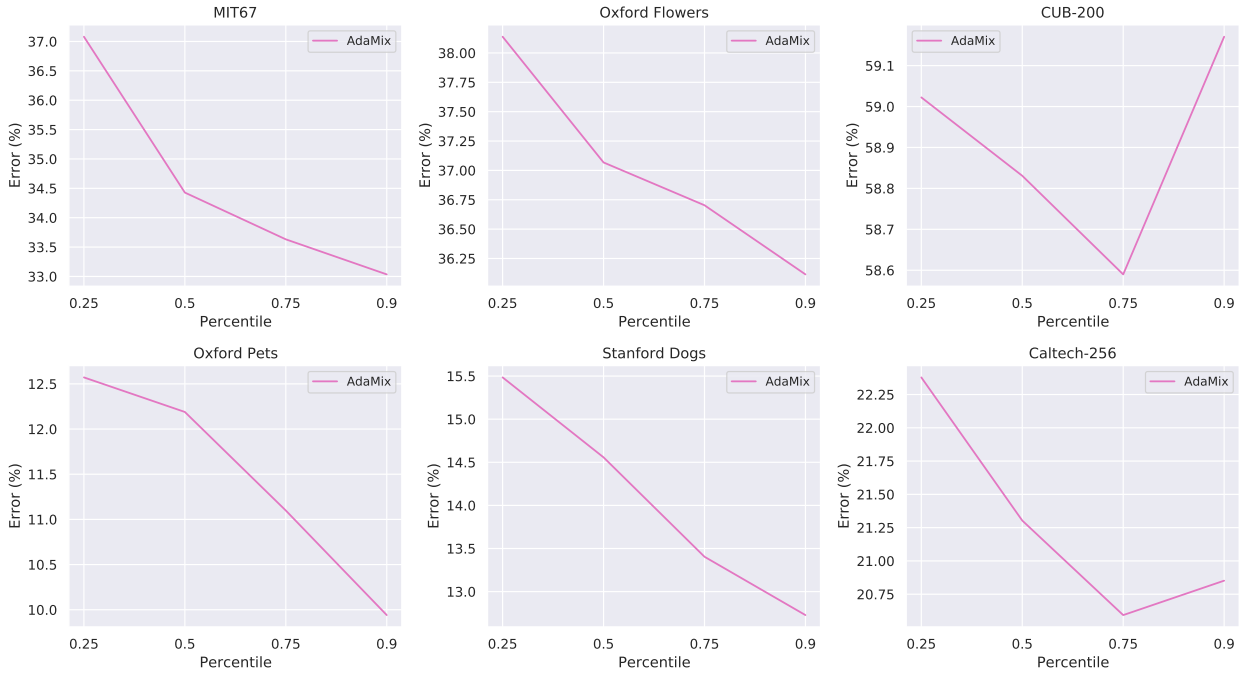

Figure 7.7: **Effect of adaptive clipping threshold** We plot the test accuracy of AdaMix using different values of adaptive clipping threshold percentile.

7.4.4 Additional Experiments

In the main paper we presented detailed experimental results on MIT-67, here we present detailed results on all the remaining datasets.

7.4.4.1 Test Error vs Privacy

We show the test error obtained by different methods for different levels of privacy ϵ and the robustness to membership attack, similar to fig. 7.1 for different datasets in figs. 7.9 to 7.13

7.4.4.2 Per-Instance Privacy

We show the pDP loss for NGD and AdaMix for different datasets in figs. 7.14 to 7.18, similar to fig. 7.3 (we use $\epsilon = 3$).

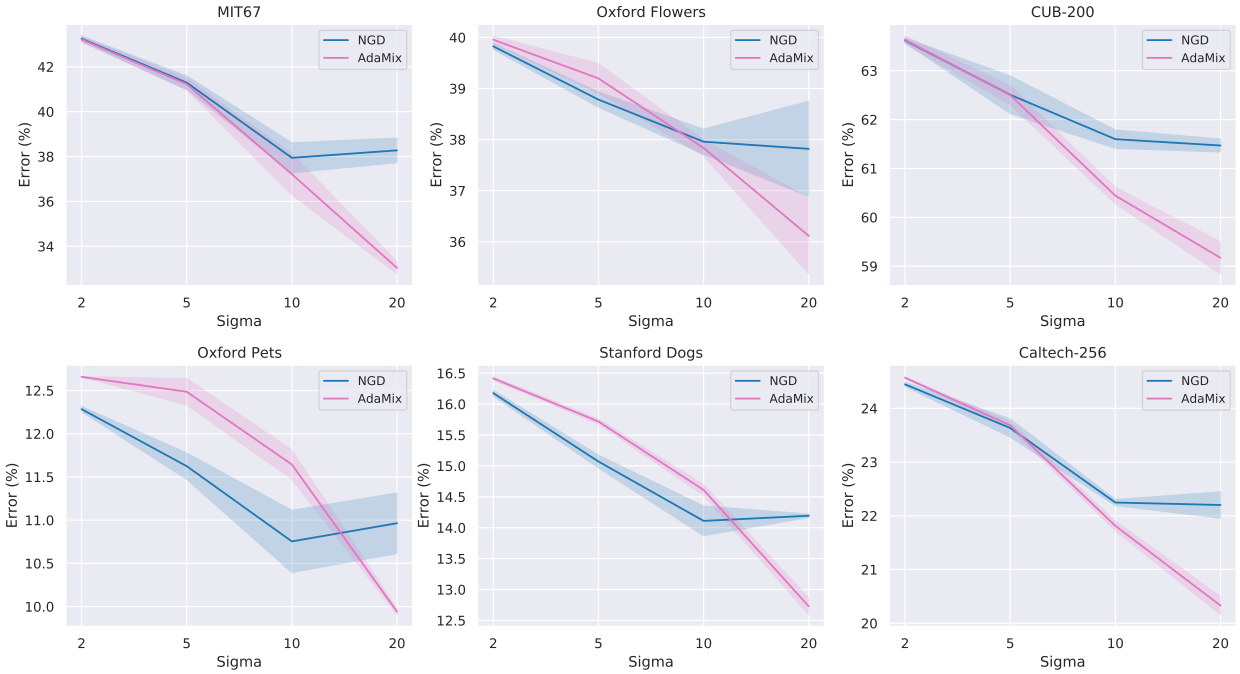

Figure 7.8: **Larger values of σ performs better.** We plot the test accuracy of NGD and AdaMix using different values of σ for $\epsilon = 3$. A larger σ performs better but needs more training steps thus verifying the claim empirically across multiple datasets.

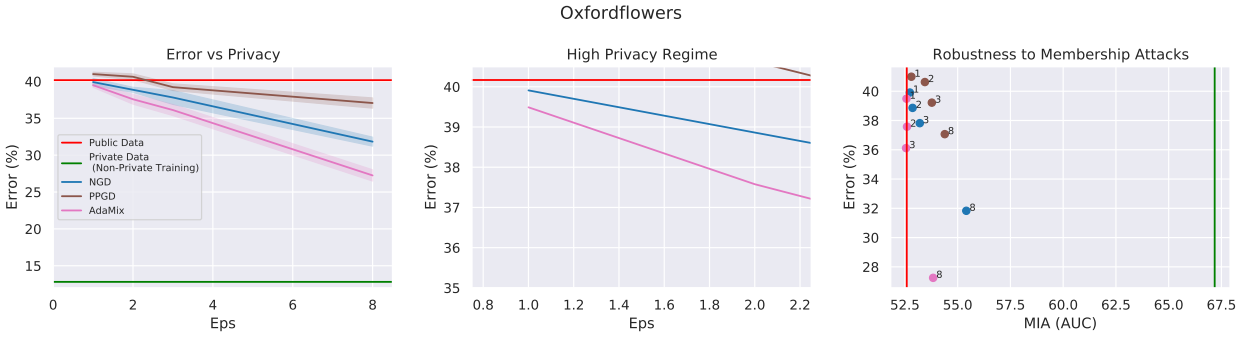

Figure 7.9: Test error vs Privacy and Robustness to Membership Attacks on Oxford-Flowers

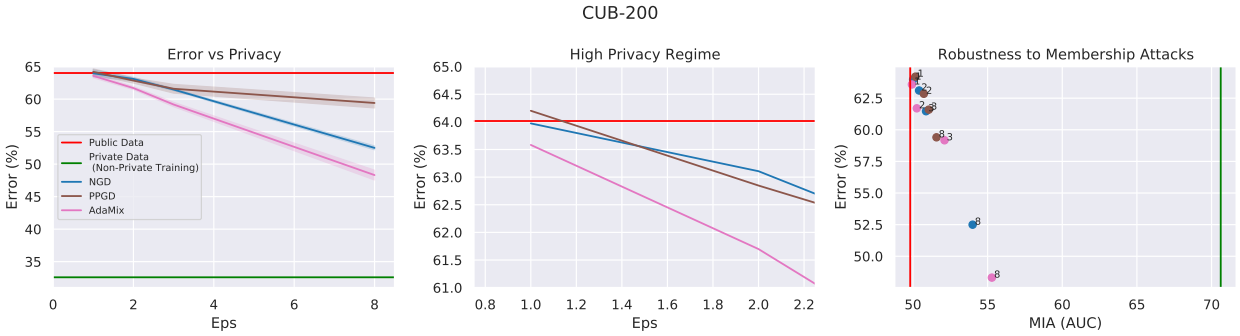

Figure 7.10: Test error vs Privacy and Robustness to Membership Attacks on CUB-200

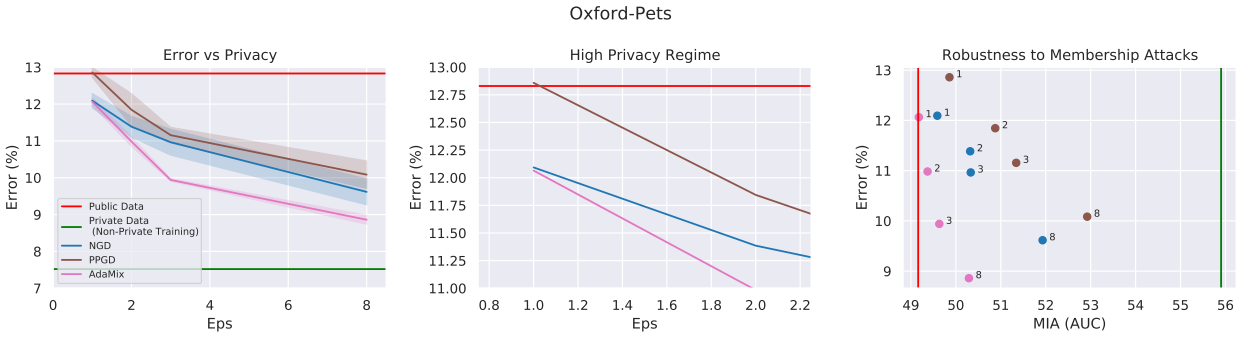

Figure 7.11: Test error vs Privacy and Robustness to Membership Attacks on Oxford-Pets

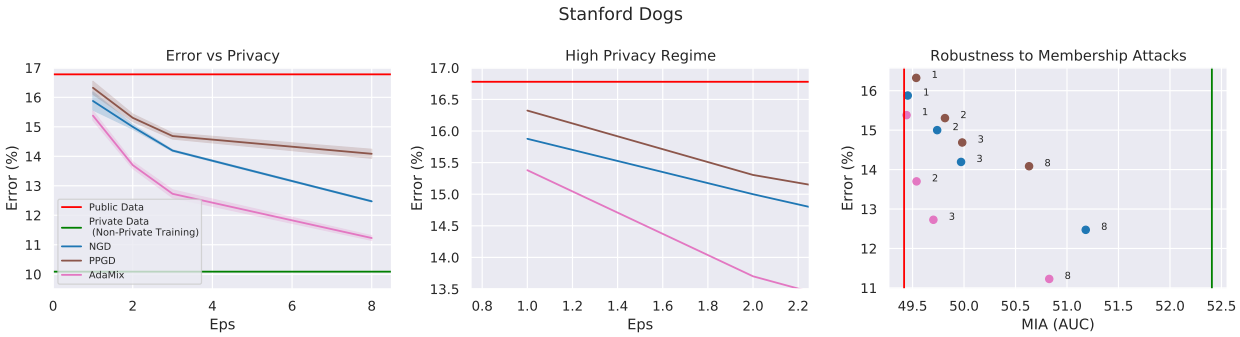

Figure 7.12: Test error vs Privacy and Robustness to Membership Attacks on Stanford Dogs

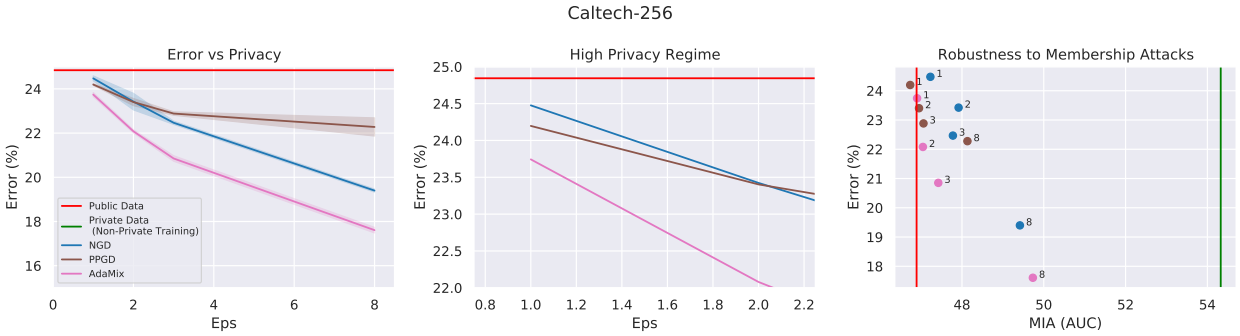

Figure 7.13: Test error vs Privacy and Robustness to Membership Attacks on Caltech-256

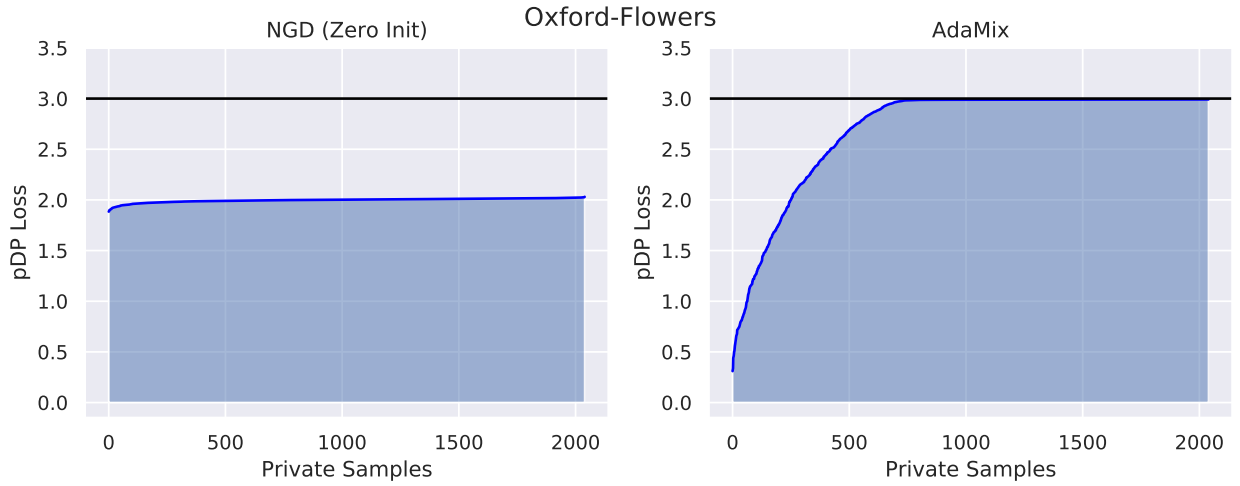

Figure 7.14: Effect of public data on per-instance DP for Oxford Flowers

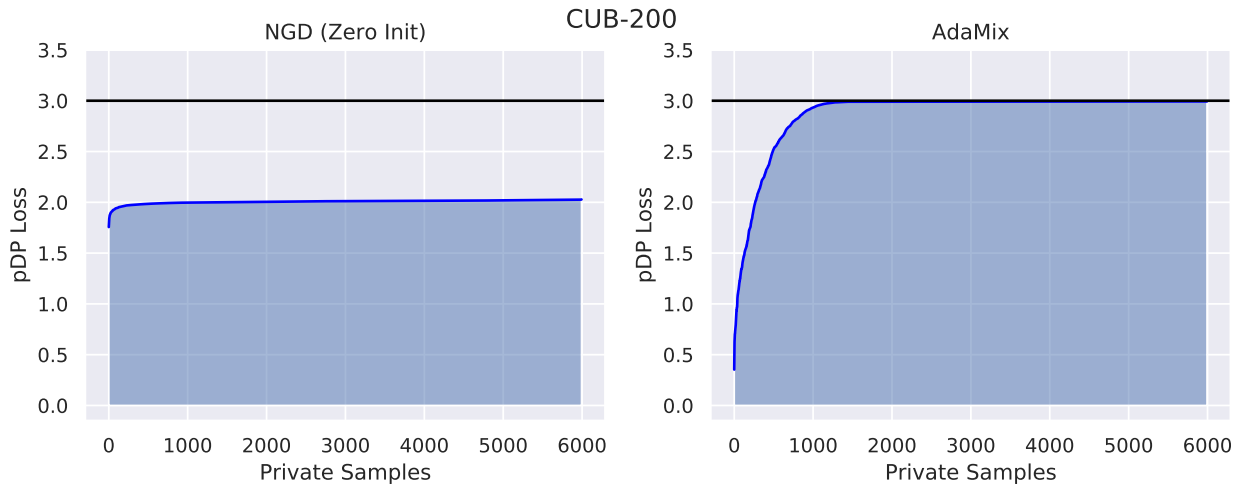

Figure 7.15: Effect of public data on per-instance DP for CUB-200

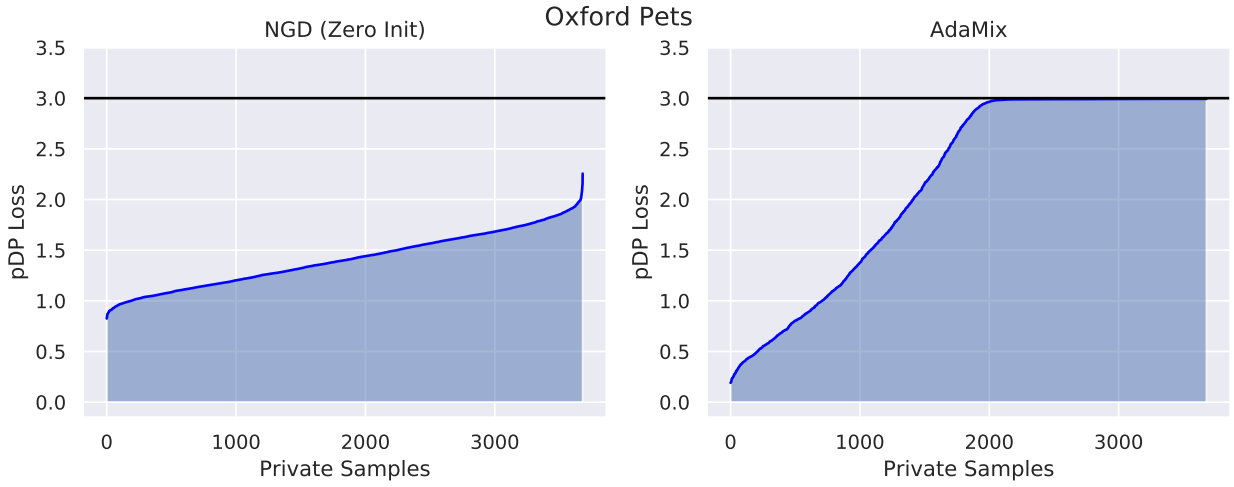

Figure 7.16: Effect of public data on per-instance DP for Oxford Pets

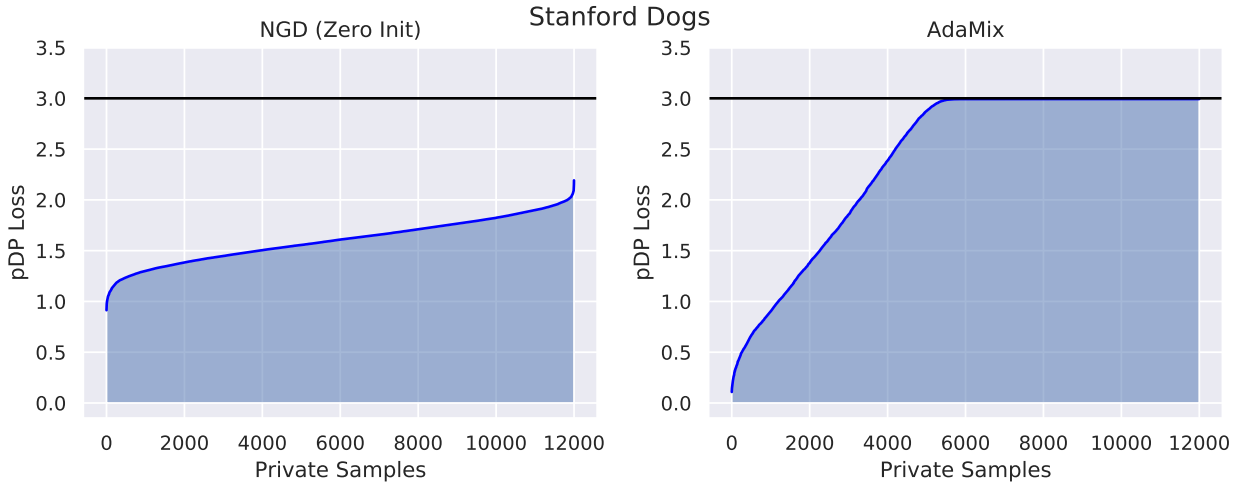

Figure 7.17: Effect of public data on per-instance DP for Stanford-Dogs

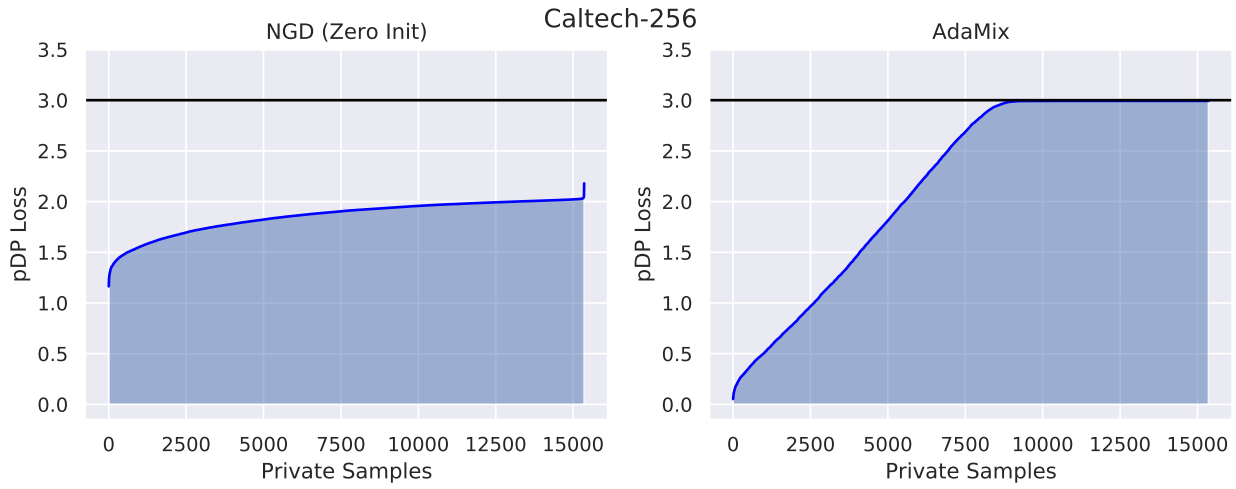

Figure 7.18: Effect of public data on per-instance DP for Caltech-256

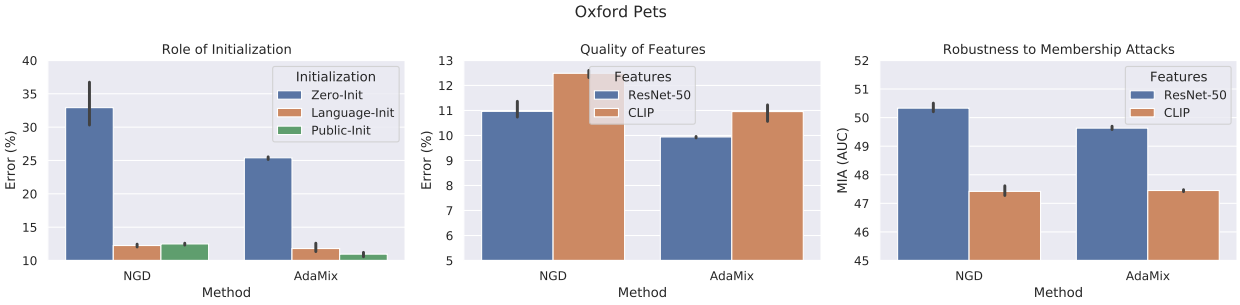

Figure 7.19: Multi-model initialization and models for Oxford Pets

7.4.5 Multi-modal Initialization

We show the effect of using different initializations and CLIP features for different datasets in figs. 7.19 to 7.22, similar to fig. 7.4 (we use $\epsilon = 3$). We show that for Stanford Dogs and Oxford Pets datasets (fig. 7.19 and fig. 7.20) which have images which are very similar to images in ImageNet, ResNet-50 trained on ImageNet performs better than CLIP features. However, the trend for the effect of initialization remains the same across all the datasets.

7.4.6 Experimental Details

We use https://github.com/yuxiangw/autodpAuto-DP library for all of our experiments. For ResNet-50 features we use the `torchvision` version of the ResNet-50 model and for CLIP features we

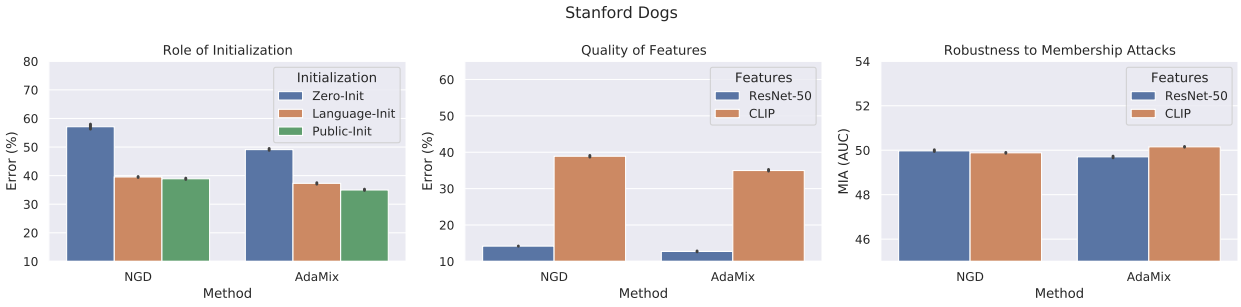

Figure 7.20: Multi-model initialization and models for Stanford Dogs

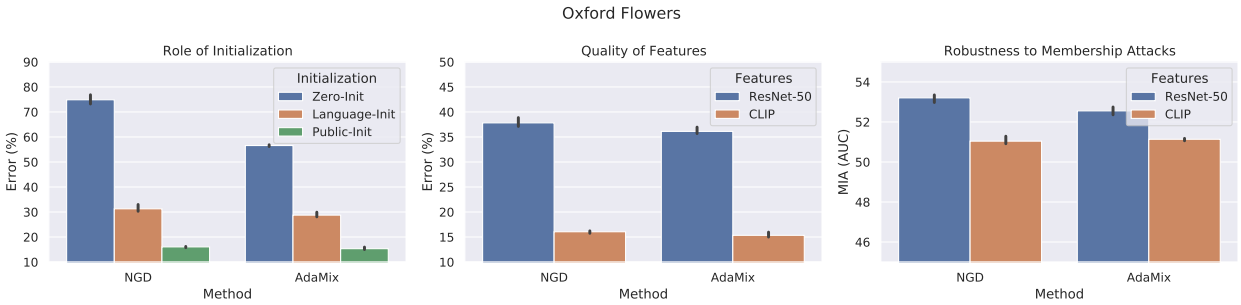

Figure 7.21: Multi-model initialization and models for Oxford Flowers

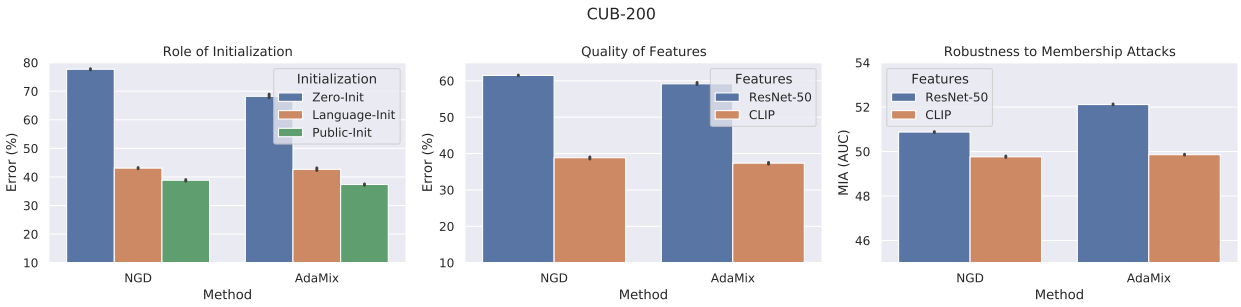

Figure 7.22: Multi-model initialization and models for CUB-200

use the model provided.[2]

To create the public and private datasets, we take 2 samples (for Oxford-Flowers and CUB-200) and 5 samples (for MIT-67, Stanford Dogs, Oxford Pets, Caltech-256) from each class as the public dataset and the remaining samples as the private dataset. In this way, the public set is less than 10% of the private set. We repeat all the experiments with 3 random seeds and report its mean and std.

For all the experiments we try multiple values for the learning rate: $\{1e-3, 2.5e-3, 5e-3\}$ for ResNet-50 experiments and $\{1e-6, 5e-7, 1e-7\}$ for CLIP experiments, and report the best values across 3 random seeds. We use $1e-2$ as the L_2 regularization coefficient across all the experiments. For subspace projection we project the gradients on a 2000-dimensional subspace for experiments with ResNet-50 features and 500-dimensional subspace for experiments with CLIP features. For the clipping threshold percentile we use a constant value of 90 percentile across all the experiments in the paper. In fig. 7.7, we show that further tuning the clipping threshold percentile can improve performance on certain datasets, however, even with a pre-decided value of 90 percentile it still out-performs all the other methods.

We train the logistic model on the public data (few shot data) for 200 epochs (sames as iterations since we use gradient descent) for multiple values of learning rate: $\{1e-1, 5e-2, 1e-2\}$ for ResNet-50 features and $\{1e-4, 5e-5, 1e-5\}$ for CLIP features and choose the best performing model. For the private training, we use $\sigma = 20$ for all the experiments and calculate the number of iterations required for private training using methods provided in the library mentioned above. In the experiments we observe that choosing a higher value of σ (thus training for more iterations) generally results in better performance.

[2] `https://github.com/openai/CLIP`

7.5 Differential privacy basics

In this section, we describes tools we need from differential privacy and use them to prove that Algorithm 2 and Algorithm 3 satisfies a family of differential privacy parameters.

Lemma 9 (Analytical Gaussian mechanism [BW18]). *For a numeric query $f : \mathcal{X}^* \to \mathbb{R}^d$ over a dataset $\mathcal{D}$, the randomized algorithm that outputs $f(\mathcal{D}) + Z$ where $Z \sim \mathcal{N}(0, \sigma^2 I_d)$ satisfies $(\epsilon, \delta(\epsilon))$-DP for all $\epsilon \geq 0$ and $\delta(\epsilon) = \Phi(\frac{\mu}{2} - \frac{\epsilon}{\mu}) - e^{\epsilon}\Phi(-\frac{\mu}{2} - \frac{\epsilon}{\mu})$ with parameter $\mu := \Delta/\sigma$, where $\Delta := \Delta_2^{(f)} = \max_{\mathcal{D}\sim\mathcal{D}'} \|f(\mathcal{D}) - f(\mathcal{D}')\|_2$ is the global L2 sensitivity of f and Φ is the CDF function of $\mathcal{N}(0, 1)$.*

The above lemma tightly characterizes the (ϵ, δ)-DP guarantee of a single invocation of the Gaussian mechanism, and the following lemma shows that we can use the same result for an adaptive composition of a sequence of Gaussian mechanisms.

Definition 5 (Gaussian Differential privacy [DRS21]). *We say a mechanism $\mathcal{M}$ satisfies μ-Gaussian differential privacy (GDP), if it satisfies $(\epsilon, \delta(\epsilon))$-DP for all $\epsilon \geq 0$ and $\delta(\epsilon)$ being that of a single Gaussian mechanism (in Lemma 9) with parameter μ.*

Lemma 10 (Composition of Gaussian mechanisms [DRS21]). *The adaptive composition of a sequence of Gaussian mechanism with noise level $\sigma_1, \sigma_2, \ldots$ and global L2 sensitivity $\Delta_1, \Delta_2, \ldots$ satisfies μ-GDP with parameter $\mu = \left(\sum_i (\Delta_i/\sigma_i)^2\right)^{1/2}$.*

Specifically, the noisy gradient descent (NoisyGD) algorithm (Algorithm 1) we use is a composition of T Gaussian mechanisms for releasing the gradients and its privacy guarantee is equivalent to that of a single Gaussian mechanism.

Proposition 8. *Let $\nabla f(w)$ be a function of the private dataset with global L2 sensitivity smaller than L for any $w \in \mathcal{W}$, then Algorithm 1 with parameter T, σ^2 such that $\rho := \frac{T^2 L^2}{2\sigma^2}$ satisfies $\sqrt{2\rho}$-Gaussian differential privacy.*

Proof. The proof follows from Lemma 10 as Algorithm 1 is an adaptive composition of T Gaussian mechanisms with global sensitivity L. □

7.6 Per-instance differential privacy

In this section, we provide details on the per-instance differential privacy [Wan19] that we used to generate Figure 7.3. To cleanly define per-instance differential privacy, we first define indistinguishability.

Definition 6 ((ϵ, δ)-indistinguishability)**.** *We say two distributions P, Q are (ϵ, δ)-indistinguishable if for any measurable set S*

$$\mathsf{Pr}_P[S] \leq e^\epsilon \cdot \mathsf{Pr}_Q[S] + \delta \text{ and } \mathsf{Pr}_Q[S] \leq e^\epsilon \cdot \mathsf{Pr}_P[S] + \delta.$$

Definition 7 ([Wan19])**.** *We say a randomized algorithm $\mathcal{M}$ is $(\epsilon(\cdot), \delta)$-per-instance differentially private (pDP) for scalar $\delta \geq 0$ and function $\epsilon : \mathcal{Z}^* \times \mathcal{Z} \to \mathbb{R}_+$, such that for any dataset $D \in \mathcal{Z}^*$, individual $z \in \mathcal{Z}$, $\mathcal{M}(D)$ and $\mathcal{M}(D \cup \{z\})$ are $(\epsilon(D, z), \delta)$-indistinguishable.*

pDP loss ϵ is a function of one particular pair of neighboring datasets. It describes the formal privacy loss incurred to the particular individual z that is added (if z is not part of the input dataset) or removed (if z is part of the input dataset). pDP is a strict generalization of DP, as we can recover DP from pDP by maximizing $\epsilon(\cdot)$ over D, z.

Lemma 11. *If $\mathcal{M}$ is $(\epsilon(\cdot), \delta)$-pDP, then $\mathcal{M}$ is also $(\sup_{D \in \mathcal{Z}^*, z \in \mathcal{Z}} \epsilon(D, z), \delta)$-DP.*

We emphasize that the pDP loss $\epsilon(\cdot)$ itself is data-independent, but specific evaluations of the pDP losses (e.g., $\epsilon(D_{-z}, z)$ or $\epsilon(D, z)$) depends on the private dataset D, thus should not be revealed unless additional care is taken to privately release these numbers.

For our purpose, we are interested in the distribution of pDP losses of individuals in the dataset induced by different DP algorithms. This is used to provide a theoretically-sound alternative to the prevalent practices of using specific attacks (such as membership inference attacks) for evaluating the data-dependent privacy losses. Before we state the pDP bounds of our algorithms, we extend the standard $(\epsilon(\cdot), \delta)$-pDP definition to a per-instance version of the Gaussian DP.

Definition 8. *We say a mechanism is $\mu(\cdot)$-per-instance Gaussian Differentially Private (pGDP), if $(D, D \cup z)$ (and $(D \cup z, D)$) are (ϵ, δ)-indistinguishable for all ϵ, δ parameters described by $\mu(D, z)$-GDP.*

This allows us to obtain precise pDP bounds under composition.

Proposition 9 (pDP analysis of AdaMix)**.** *Let $z_1, ..., z_n$ be the data points of the private dataset. Algorithm 3 satisfies $\mu(\cdot)$-pGDP with*

$$\mu(D_{-i}, z_i) = \sqrt{\sum_{t=1}^{T} \frac{\min\{\|\nabla \ell_i(w_t)\|, \tau\}^2}{\sigma^2}}.$$

Similarly, Algorithm 2 satisfies $\mu(\cdot)$-pGDP with

$$\mu(D_{-i}, z_i) = \sqrt{\sum_{t=1}^{T} \frac{\|U^T \tilde{g}_i^{pri}(w_t)\|_2^2}{\tau_t^2 \sigma^2}}.$$

Proof. Both algorithms are the composition of T-Gaussian mechanisms. Thus the results follow by the composition of pGDP. The composition of pGDP is implied by the composition theorem of GDP (Lemma 10) by choosing the space of datasets to be just $\{D, D \cup \{z\}\}$. □

Fixing any δ, we can then compute the corresponding $\epsilon(\cdot)$ for $(\epsilon(\cdot), \delta)$-pDP using the formula of Gaussian mechanism with μ taken to be $\mu(\cdot)$ pointwise.

7.7 Proofs of the technical results

We will be using the following $O(1/t)$ convergence bound due to Lacoste-Julien, Schmidt and Bach [LSB12], which uses a decaying learning rate and a non-uniform averaging scheme.

[3]As we discussed earlier, per-example gradient clipping can be viewed as Huberizing the loss function in GLMs [SST21], all our results apply to the updated loss function. The adaptive clipping approach we took, can be viewed as an heuristic that automatically identifies an appropriate level of Huberization.

Theorem 10 (Convergence of SGD for Strongly Convex Objectives [LSB12]). *Let f be a m-strongly convex and defined on a convex set $\mathcal{W}$. Assume stochastic gradient oracle g_t satisfies that $\mathbb{E}[g_t|w_t] \in \partial f(w_t)$ and $\mathbb{E}[\|g_t\|^2] \leq G^2$ for all $t = 1, ..., T$. Then the (projected) stochastic gradient descent with learning rate $\eta_t = \frac{2}{m(t+1)}$ satisfies*

$$\mathbb{E}[f(\sum_{t=1}^{T} \frac{2t}{T(T+1)} w_t)] - f(w^*) \leq \frac{2G^2}{m(T+1)} \tag{7.1}$$

$$\text{and } \mathbb{E}[\|w_{T+1} - w^*\|^2] \leq \frac{4G^2}{m^2(T+1)}. \tag{7.2}$$

Corollary 2 (NoisyGD for Strongly Convex Objectives). *Let $J(w) = \mathcal{L}(w) + \frac{\lambda}{2}\|w\|^2$ with individual loss functions ℓ being L-Lipschitz on $\mathcal{W}$. Assume $\sup_{w \in \mathcal{W}} \|w\| \leq B$. Let the learning rate be $\eta_t = \frac{2}{\lambda(t+1)}$, then*

$$\mathbb{E}\left[J\left(\frac{2}{T(T+1)} \sum_{t=1}^{T} t w_t\right)\right] - J^* \leq \frac{2(NL + \lambda B)^2}{\lambda T} + \frac{2d\sigma^2}{\lambda T} \tag{7.3}$$

$$= \frac{2(NL + \lambda B)^2}{\lambda T} + \frac{dL^2}{\lambda \rho}, \tag{7.4}$$

where $\rho := \frac{TL^2}{2\sigma^2}$ is the privacy parameter of the algorithm ($\sqrt{2\rho}$-GDP).

Proof. First check that $NL + \lambda B$ upper bounds the Lipschitz constant of J on because the Lipschitz constant of $\frac{\lambda}{2}\|w\|^2$ is smaller than λB due to the bounded domain. Second, check that the noisy gradient oracle satisfies that it is unbiased, and the added noise has a variance of σ^2 per coordinate for all d coordinates. Thus

$$\mathbb{E}[\|g_t\|^2|w_t] = \mathbb{E}[\|\nabla J(w_t)\|^2|w_t] + \mathbb{E}[\|n_t\|^2|w_t] \leq (NL + \lambda B)^2 + d\sigma^2.$$

Thus by taking expectation on both sides we verify that we can take $G^2 = (NL + \lambda B)^2 + d\sigma^2$.

It remains to substitute these quantities and apply the first statement of Theorem 10. □

Corollary 3 (One-Pass SGD on public data). *Assume the public data with N samples are drawn from the same distribution of the private data. Assume that the (population risk)*

$$R(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(\theta^*, (x, y))]$$

is c-strongly convex at θ^* for some constant c. Then the one-pass SGD algorithm below

$$w_{t+1} = w_t - \frac{2}{c(t+1)} \nabla \ell(w_t, (x_t, y_t))$$

for $t = 1, ..., N_{pub}$ obeys that

$$\mathbb{E}[\|w_{N_{pub}+1} - w^*\|^2] \leq \frac{4L^2}{c^2 N_{pub}}$$

where $w^* = \arg\min_{w \in \mathcal{W}} \mathcal{R}(w)$.

Proof. First note that since the data is drawn iid, running one-pass SGD by going through the data points in a random order uses a *fresh sample* to update the parameters. This is is equivalent to optimizing the population risk directly. Check that for any fixed w and all $t = 1, ..., N_{\text{pub}}$ $\mathbb{E}_{(x_t,y_t)\sim\mathcal{D}}[\nabla_w \ell(w, (x_t, y_t))] = \nabla \mathcal{R}(w)$. Moreover, we need this stochastic gradient oracle to satisfy $\mathbb{E}_{(x,y)\sim\mathcal{D}}[\|\nabla \ell(\theta, (x, y))\|^2] \leq G^2$. Notice that by our assumption $\|\nabla \ell(\theta, (x, y))\|^2 \leq L^2$, thus we can take $G = L$. By invoking the second statement of Theorem 10 the result follows. □

With these two corollaries stated, we are now ready to prove Theorem 5 and Theorem 11.

Proof of Theorem 5. The proof relies on Corollary 2, and the following argument. When additional regularization with parameter λ, the utility we consider should still be considered in terms of $\mathcal{L}(\hat{w}) - \mathcal{L}(w^*)$. Let w^* be any comparator satisfying $B > \|w^*\|$

$$\begin{aligned}
&\mathcal{L}(\bar{w}) - \mathcal{L}(w^*) = J(\bar{w}) - J_\lambda(w_\lambda^*) \\
&\quad + J_\lambda(w_\lambda^*) - J(w^*) + \frac{\lambda}{2}\|w^* - w_{\text{ref}}\|^2 - \frac{\lambda}{2}\|\hat{w} - w_{\text{ref}}\|^2 \\
\leq &J(\hat{w}) - J_\lambda(w_\lambda^*) + \frac{\lambda}{2}\|w^* - w_{\text{ref}}\|^2.
\end{aligned}$$

Take expectation on both sides and apply Corollary 2

$$\mathbb{E}[\mathcal{L}(\bar{w})] - \mathcal{L}(w^*) \leq \frac{2(NL + \lambda B)^2}{\lambda T} + \frac{dL^2}{\lambda \rho} + \frac{\lambda}{2}\|w^* - w_{\text{ref}}\|^2.$$

Finally, choosing $\lambda = \sqrt{\frac{\rho}{2\|w^* - w_{\text{ref}}\| dL^2}}$ yields

$$\mathbb{E}[\mathcal{L}(\hat{w})] - \mathcal{L}(w^*) \leq \frac{4(nL + \lambda B)^2}{\lambda T} + \frac{\sqrt{d}L\|w^* - w_{\text{ref}}\|}{\sqrt{2\rho}}$$

as claimed (dividing N on both sides to get $\hat{R}$). □

Theorem 11. *Assume the private data and public data are drawn i.i.d. from the same distribution $\mathcal{D}$ and that $R(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(w,(x,y))]$ is c-strongly convex in $\mathcal{W}$. Let $w_{\text{ref}} = w_{N_{pub}+1}$ — the last iterate of a single pass stochastic gradient descent on $\hat{\mathcal{R}}_{pub}(w)$ (initializing from $w_0 = 0$) that goes over the public dataset exactly once one data-point at a time with learning rate $\eta_t = \frac{2}{c(t+1)}$. Let w_{ref} be passed into Theorem 5's instantiation of NoisyGD, which returns $\bar{w}$ (The pseudo-code of this algorithm is summarized in Algorithm 3 in the appendix), then at the limit when T is sufficiently large, the* excess risk *obeys that*

$$\begin{aligned}&\mathbb{E}[\mathcal{R}(\bar{w})] - \mathcal{R}(w^*)\\ \leq&\frac{4\sqrt{d}L^2}{c\sqrt{N_{pub}}N\sqrt{2\rho}} + \textit{Gen}(\bar{w},N) + \textit{Gen}(w^*,N),\end{aligned}$$

where $w^* = \arg\min_{w\in\mathcal{W}}\mathcal{R}(w)$ and $\textit{Gen}(w,N) := \left|\mathbb{E}[\mathcal{R}(w) - \hat{\mathcal{R}}(w)]\right|$ is the expected generalization gap of (a potentially data-dependent) w.

Proof of Theorem 11. Let $w^* = \arg\min_{w\in\mathcal{W}}\mathcal{R}(w)$. By Corollary 3, we have

$$\mathbb{E}[\|w_{\text{ref}} - w^*\|^2] \leq \frac{4L^2}{c^2 N_{\text{pub}}},$$

which implies (by Jensen's inequality) that $\mathbb{E}[\|w_{\text{ref}} - w^*\|] \leq \sqrt{\frac{4L^2}{c^2 N_{\text{pub}}}}$.

Now by plugging in the w^* in theorem, take expectation over the public dataset, and substitute the above bound, we get

$$\mathbb{E}[\mathbb{E}[\hat{\mathcal{R}}(\bar{w})] - \hat{\mathcal{R}}(w^*)] \leq \frac{4(NL+\lambda B)^2}{\lambda T N} + \frac{2\sqrt{d}L^2}{c\sqrt{N_{\text{pub}}}N\sqrt{2\rho}}.$$

Take T to be sufficiently large so that the second term dominates, we obtain the stated bound.

Finally, to convert the above bound into that of the excess risk:

$$\begin{aligned}&\mathbb{E}[\mathbb{E}[\hat{\mathcal{R}}(\bar{w})] - \hat{\mathcal{R}}(w^*)] - (\mathbb{E}[\mathcal{R}(\bar{w})] - \mathcal{R}(w^*))\\ \leq&|\mathbb{E}[\mathbb{E}[\hat{\mathcal{R}}(\bar{w})] - \mathcal{R}(\bar{w})]| + |\mathbb{E}[\hat{\mathcal{R}}(w^*)] - \mathcal{R}(w^*)|\\ :=&\text{Gen}(\bar{w},N) + \text{Gen}(w^*,N),\end{aligned}$$

which completes the proof. □

We make two additional remarks. First, we do not require the *empirical* objective $\mathcal{L}_{\text{pub}}$ to be strongly convex. In practice, we do not have strong convexity when $N_{\text{pub}} < d$. The assumption of c-strong convexity is on the *population-level* risk function $\mathcal{R}$. Second, our bound decomposes the excess (population) risk of the private learner into a (local) uniform-convergence bound (which is required by a non-private learner too) and an additional cost due to privacy. Note that $\text{Gen}(N)$ is usually $O(1/\sqrt{N})$ but could be $O(1/N)$ when certain data-dependent "fast rate" conditions are met, e.g., realizability, low-noise, or curvature (see, e.g., [KL15]). Our results suggest that the cost of privacy asymptotically vanishes (fix ρ, $N_{\text{pub}} \to \infty$, and $N_{\text{pub}}/N \to 0$) even under these fast rate conditions relative to the non-private rate.

Algorithm 3: (Theoretical version of) AdaMix training algorithm (no clipping, no adaptive projection, slightly different pretraining, theoretically chosen learning rate).

Data: Public dataset D_{pub}, private dataset D_{pri}, privacy parameter (ϵ, δ), noise variance σ, Lipschitz constant L^3, population-level strong convex parameter c, regularization parameter λ and a constraint set $\mathcal{W}$.

Result: $\bar{w}$

Public Pretraining Phase (OnePassSGD on D_{pub}): $w_1 = 0$. **for** $t = 1, \ldots, N_{pub}$ **do**

In a shuffled order $\eta_t = \frac{2}{c(t+1)}$;

$w_{t+1} \leftarrow \Pi_{\mathcal{W}}\big(w_t - \eta_t \nabla \ell(w_t, (\tilde{x}_t, \tilde{y}_t))\big)$;

end

$w_{\text{ref}} \leftarrow w_{N_{\text{pub}}+1}$.

Mix Training Phase (NoisyGD on $D_{\text{pub}} \cup D_{\text{pri}}$):

$T \leftarrow \text{Calibrate}(\epsilon, \delta, \sigma)$ (i.e., $\frac{T\tau^2}{2\sigma^2} =: \rho$) NoisyGD on objective function

$\mathcal{L}(w) + \frac{\lambda}{2}\|w - w_{\text{ref}}\|^2$

$w_1 = w_{\text{ref}}$;

for $t = 1, \ldots, T$ **do**

$\eta_t \leftarrow \frac{2}{\lambda(t+1)}$;

$n_t \sim N(0, \sigma^2 I)$;

$w_{t+1} \leftarrow \Pi_{\mathcal{W}}\big(w_t - \eta_t(\sum_{i=1}^{N_{\text{pri}}} \nabla \ell_i(w_t) + \sum_{j=1}^{N_{\text{pub}}} \nabla \tilde{\ell}_j(w_t) + n_t)\big)$;

end

The following averaging can be implemented incrementally without saving w_t

$\bar{w} \leftarrow \sum_{t=1}^{T} \frac{2t}{T(T+1)} w_t$

CHAPTER 8

Discussion and Future Work

Machine unlearning is a nascent field with widespread applications in data removal, private analysis, fairness, copyright protection etc. Our work [GAS19a] provided the initial result in selectively forgetting samples from the weights of a trained neural network. We provide few methods to remove data which includes perturbing the weights by adding Fisher noise, add a single shot update, linearizing the network in a mixed-private setting, and training models in a differentially private manner with adaptive updates. We show applications of these methods for image classification tasks, however, our method is not restricted to any domain or modality. We also provide computable information theoretic bounds to estimate the amount of residual information after forgetting. Our results attempts to answer some questions but also provides a lot of directions for future works:

Compartmentalized Training: Sharded learning is another approach to perform selective forgetting, where the training data is split into multiple shards and a separate model is trained for each shard. During inference based on the application model predictions are combined appropriately. When asked to remove a subset of the dataset, one can simply discard the subset of models contaminated by those samples, and re-train them from scratch. We explored this direction in a recent work [DBA23], where we seek to combine sharded learning with differentially private algorithms. Compartmentalized training opens a few directions: (i) how many splits should the data be sharded into, (ii) which samples should be present in each, (iii) what properties should samples belonging to the same splits have, (iv) how should the individual splits be trained, (v) depending on the application, whats the ideal mechanism to combine the predictions. For instance in a classification setting, its easy to gauge that choosing the top-k predictions could be the best way

for combining predictions, however there is need for more work to address tasks like generative modelling, sentence completion, image segmentation, object detection and so on.

Approximate Unlearning: Our work majorly provides algorithm for approximately removing the influence of training samples from the weights, where we bound the approximation error information theoretically. However, majority of the unlearning algorithms are focused on removing information using linear approximations or linearized versions of deep networks. Addressing approximate removal for non-linear networks without the use of Newton-style updates is an interesting direction for future work, including the effect of stability of training algorithms which measures the divergence of different solutions.

Linearized Models: Our work in chapter 5 prompted an important observation, that for fine-tuning tasks linearized deep networks perform comparable to their non-linear counterparts. Since majority of the models used in practical tasks are often fine-tuned its important to understand the efficacy of such models in different applications as linear models offer significant theoretical benefits compared to non-linear models. Linearized models should be explored in domains other than image classification, for instance, fine-tuning massive generative models on few-shot data, as in chapter 5 we show that linear models perform significantly better than non-linear in few shot regimes

Continual and Composable learning: Unlearning correlates heavily with continual learning, as continual learning involves adding small subsets of data to trained models while unlearning involves removing data from trained weights. An unlearning method can be reversed while handling the subtle details to obtain a continual learning method. Similarly inspiration from continual learning methods can be used to obtain better forgetting methods as continual learning is a much older field than unlearning. Using linearized models also enables one to learn models in a composable way, as the activations are a linear combination of the weights, as a result combining multiple activations is equivalent to simply combining the weights correspondingly. This is especially useful while employing large scale models as the cost of inference through such models can be significantly high,

which can be bypassed by composing linearized deep networks.

Computable information theoretic bounds: Our bounds also enables us to estimate information theoretic quantities like Conditional Mutual Information for generalization of deep neural networks [RAG22], and are not restricted to privacy applications. In chapter 3 we show how these bounds can be computed efficiently for deep neural networks, and how the activations leak lesser information compared to the weights. This provides an interesting way to compute such quantities however it also necessitates the need for future research to improve the empirical tightness of these bounds.

Copyright Protection: With the increase in deployment of large scale generative models, it is becoming necessary to protect the copyrights of the training data. Especially when the training data belongs to artists, or professionals in art/photography. [VKB23] recently proposed a method for generative modelling with copyright protection. Their method involves training multiple models on sharded data, and using a privacy-preserving combining technique which generates samples which are copyright protected with respect to the training data. Thus training models in a sharded fashion can not only enable unlearning samples efficiently, but also generate data in a copyright protected manner, which prompts the need for additional research at the intersection of sharded learning, copyright protection and differential privacy.

REFERENCES

[ABM19] Noga Alon, Raef Bassily, and Shay Moran. "Limits of private learning with access to public data." In *Advances in Neural Information Processing Systems (NeurIPS'19)*, pp. 10342–10352, 2019.

[ACG16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep learning with differential privacy." In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318. ACM, 2016.

[ADH19] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. "On exact computation with an infinitely wide neural net." In *Advances in Neural Information Processing Systems*, pp. 8139–8148, 2019.

[AGR21a] Alessandro Achille, Aditya Golatkar, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. "LQF: Linear Quadratic Fine-Tuning." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[AGR21b] Alessandro Achille, Aditya Golatkar, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. "Lqf: Linear quadratic fine-tuning." In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15729–15739, 2021.

[ALT19] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless Fowlkes, Stefano Soatto, and Pietro Perona. "Task2Vec: Task Embedding for Meta-Learning." *International Conference on Computer Vision*, 2019.

[ARS19] Alessandro Achille, Matteo Rovere, and Stefano Soatto. "Critical Learning Periods in Deep Neural Networks." In *International Conference of Learning Representations*, 2019.

[AS18] Alessandro Achille and Stefano Soatto. "Emergence of invariance and disentanglement in deep representations." *The Journal of Machine Learning Research*, **19**(1):1947–1980, 2018.

[AS19] Alessandro Achille and Stefano Soatto. "Where is the Information in a Deep Neural Network?" *arXiv e-prints*, p. arXiv:1905.12213, May 2019.

[ATM21] Galen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. "Differentially private learning with adaptive clipping." In *Advances in Neural Information Processing Systems (NeurIPS'21)*, 2021.

[BBM18] Raef Bassily, Mikhail Belkin, and Siyuan Ma. "On exponential convergence of sgd in non-convex over-parametrized learning." *arXiv preprint arXiv:1811.02564*, 2018.

[BCC19] Lucas Bourtoule, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. "Machine Unlearning." *arXiv preprint arXiv:1912.03817*, 2019.

[BD20] Bjorn Barz and Joachim Denzler. "Deep learning on small datasets without pre-training using cosine loss." In *The IEEE Winter Conference on Applications of Computer Vision*, pp. 1371–1380, 2020.

[BGR19] Daniel Bernau, Philip-William Grassal, Jonas Robl, and Florian Kerschbaum. "Assessing differentially private deep learning with membership inference." *arXiv preprint arXiv:1912.11328*, 2019.

[BNS78] James R Bunch, Christopher P Nielsen, and Danny C Sorensen. "Rank-one modification of the symmetric eigenproblem." *Numerische Mathematik*, **31**(1):31–48, 1978.

[BPS19] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. "Differential privacy has disparate impact on model accuracy." *Advances in Neural Information Processing Systems*, **32**:15479–15488, 2019.

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. "Private empirical risk minimization: Efficient algorithms and tight error bounds." In *Annual Symposium on Foundations of Computer Science*, pp. 464–473. IEEE, 2014.

[BSZ20] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. "Machine Unlearning: Linear Filtration for Logit-based Classifiers." *arXiv preprint arXiv:2002.02730*, 2020.

[BW18] Borja Balle and Yu-Xiang Wang. "Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising." In *International Conference on Machine Learning*, pp. 394–403. PMLR, 2018.

[CCS16] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. "Entropy-sgd: Biasing gradient descent into wide valleys." *arXiv preprint arXiv:1611.01838*, 2016.

[CLE19] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. "The secret sharer: Evaluating and testing unintended memorization in neural networks." In *28th USENIX Security Symposium (USENIX Security 19)*, pp. 267–284, 2019.

[CM08] Kamalika Chaudhuri and Claire Monteleoni. "Privacy-preserving logistic regression." *Advances in neural information processing systems*, **21**:289–296, 2008.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. "Differentially private empirical risk minimization." *Journal of Machine Learning Research*, **12**(Mar):1069–1109, 2011.

[Cof20] Ignacio N Cofone. *The right to be forgotten: A Canadian and comparative perspective*. Routledge, 2020.

[CSS18] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. "Large scale fine-grained categorization and domain-specific transfer learning." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4109–4118, 2018.

[CSX18] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. "VGGFace2: A dataset for recognising faces across pose and age." In *International Conference on Automatic Face and Gesture Recognition*, 2018.

[CT12] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[CTW21] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. "Extracting Training Data from Large Language Models." In *USENIX Security Symposium*, 2021.

[CWH20] Xiangyi Chen, Steven Z Wu, and Mingyi Hong. "Understanding gradient clipping in private SGD: a geometric perspective." *Advances in Neural Information Processing Systems*, **33**, 2020.

[CY15] Yinzhi Cao and Junfeng Yang. "Towards making systems forget with machine unlearning." In *2015 IEEE Symposium on Security and Privacy*, pp. 463–480. IEEE, 2015.

[DBA23] Yonatan Dukler, Benjamin Bowman, Alessandro Achille, Aditya Golatkar, Ashwin Swaminathan, and Stefano Soatto. "Safe: Machine unlearning with shard graphs." *arXiv preprint arXiv:2304.13169*, 2023.

[DDS09a] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[DDS09b] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[DMN06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. "Calibrating noise to sensitivity in private data analysis." In *Theory of cryptography conference*, pp. 265–284. Springer, 2006.

[DR14] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy." *Foundations and Trends® in Theoretical Computer Science*, **9**(3–4):211–407, 2014.

[DR17] Gintare Karolina Dziugaite and Daniel M Roy. "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data." *arXiv preprint arXiv:1703.11008*, 2017.

[DRS21] Jinshuo Dong, Aaron Roth, and Weijie Su. "Gaussian Differential Privacy." *Journal of the Royal Statistical Society*, 2021.

[Dwo08] Cynthia Dwork. "Differential privacy: A survey of results." In *International conference on theory and applications of models of computation*, pp. 1–19. Springer, 2008.

[Fel20] Vitaly Feldman. "Does learning require memorization? a short tale about a long tail." In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 954–959, 2020.

[FJR15] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures." In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333. ACM, 2015.

[FZ20] Vitaly Feldman and Chiyuan Zhang. "What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation." In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pp. 2881–2891. Curran Associates, Inc., 2020.

[GAS19a] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. "Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks.", 2019.

[GAS19b] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. "Time Matters in Regularizing Deep Networks: Weight Decay and Data Augmentation Affect Early Learning Dynamics, Matter Little Near Convergence." In *Advances in Neural Information Processing Systems*, pp. 10677–10687, 2019.

[GAS20] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. "Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations." *arXiv preprint arXiv:2003.02960*, 2020.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[GDN13] Pavel Golik, Patrick Doetsch, and Hermann Ney. "Cross-entropy vs. squared error training: a theoretical and experimental comparison." In *Interspeech*, volume 13, pp. 1756–1760, 2013.

[GGH19] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens van der Maaten. "Certified Data Removal from Machine Learning Models." *arXiv preprint arXiv:1911.03030*, 2019.

[GGV19] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. "Making AI forget you: Data deletion in machine learning." In *Advances in Neural Information Processing Systems*, pp. 3513–3526, 2019.

[GHP] G. Griffin, AD. Holub, and Pietro Perona. "The Caltech 256.".

[GHP07] Gregory Griffin, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset." 2007.

[GLB18] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. "Fast approximate natural gradient descent in a kronecker factored eigenbasis." In *Advances in Neural Information Processing Systems*, pp. 9550–9560, 2018.

[GPS17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. "On calibration of modern neural networks." *arXiv preprint arXiv:1706.04599*, 2017.

[GSL19] Ryan Giordano, William Stephenson, Runjing Liu, Michael Jordan, and Tamara Broderick. "A swiss army infinitesimal jackknife." In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1139–1147, 2019.

[HAP17] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. "Deep models under the GAN: information leakage from collaborative deep learning." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 603–618. ACM, 2017.

[HAP21] Hrayr Harutyunyan, Alessandro Achille, Giovanni Paolini, Orchid Majumder, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. "Estimating informativeness of samples with Smooth Unique Information." In *International Conference on Learning Representations*, 2021.

[HB21] Like Hui and Mikhail Belkin. "Evaluation of Neural Architectures Trained with Square Loss vs Cross-Entropy in Classification Tasks." In *International Conference on Learning Representations*, 2021.

[HBG18] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. "Norm matters: efficient and accurate normalization schemes in deep networks." In *Advances in Neural Information Processing Systems*, pp. 2160–2170, 2018.

[HHS17] Elad Hoffer, Itay Hubara, and Daniel Soudry. "Train longer, generalize better: closing the generalization gap in large batch training of neural networks." In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

[HMD19] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. "LOGAN: Membership inference attacks against generative models." *Proceedings on Privacy Enhancing Technologies*, **2019**(1):133–152, 2019.

[HRS15] Moritz Hardt, Benjamin Recht, and Yoram Singer. "Train faster, generalize better: Stability of stochastic gradient descent." *arXiv preprint arXiv:1509.01240*, 2015.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Flat minima." *Neural Computation*, **9**(1):1–42, 1997.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[INS19] Roger Iyengar, Joseph P Near, Dawn Song, Om Thakkar, Abhradeep Thakurta, and Lun Wang. "Towards practical differentially private convex optimization." In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 299–316. IEEE, 2019.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015.

[ISC20] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. "Approximate Data Deletion from Machine Learning Models: Algorithms and Evaluations." *arXiv preprint arXiv:2002.10077*, 2020.

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. "Neural tangent kernel: Convergence and generalization in neural networks." In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

[JKA17] Stanisław Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. "Three factors influencing minima in sgd." *arXiv preprint arXiv:1711.04623*, 2017.

[JUO20] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. "Auditing Differentially Private Machine Learning: How Private is Private SGD?" In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pp. 22205–22216. Curran Associates, Inc., 2020.

[JW18] Bargav Jayaraman and Lingxiao Wang. "Distributed learning without distress: Privacy-preserving empirical risk minimization." *Advances in Neural Information Processing Systems*, 2018.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KGC18] Daniel S Kermany, Michael Goldbaum, Wenjia Cai, Carolina CS Valentim, Huiying Liang, Sally L Baxter, Alex McKeown, Ge Yang, Xiaokang Wu, Fangbing Yan, et al. "Identifying medical diagnoses and treatable diseases by image-based deep learning." *Cell*, **172**(5):1122–1131, 2018.

[KHH20] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. "Being Bayesian, Even Just a Bit, Fixes Overconfidence in ReLU Networks." *arXiv preprint arXiv:2002.10118*, 2020.

[KJY] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. "Novel dataset for fine-grained image categorization: Stanford dogs.".

[KJY11] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. "Novel Dataset for Fine-Grained Image Categorization." In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.

[KL15] Tomer Koren and Kfir Y Levy. "Fast Rates for Exp-concave Empirical Risk Minimization." In *Neural Information Processing Systems*, pp. 1477–1485, 2015.

[KL17] Pang Wei Koh and Percy Liang. "Understanding black-box predictions via influence functions." *arXiv preprint arXiv:1703.04730*, 2017.

[KMN16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima." *arXiv preprint arXiv:1609.04836*, 2016.

[KPR17a] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. "Overcoming catastrophic forgetting in neural networks." *Proceedings of the national academy of sciences*, **114**(13):3521–3526, 2017.

[KPR17b] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. "Overcoming catastrophic forgetting in neural networks." *Proceedings of the National Academy of Sciences*, **114**(13):3521–3526, 2017.

[KR19] Michael Kearns and Aaron Roth. *The ethical algorithm: The science of socially aware algorithm design*. Oxford University Press, 2019.

[Kri09] Alex Krizhevsky et al. "Learning multiple layers of features from tiny images." Technical report, Citeseer, 2009.

[KRR20] Peter Kairouz, Mónica Ribero, Keith Rush, and Abhradeep Thakurta. "Fast dimension independent private adagrad on publicly estimated subspaces." *arXiv preprint arXiv:2008.06570*, 2020.

[KSD13] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. "3D Object Representations for Fine-Grained Categorization." In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[KSW15] Durk P Kingma, Tim Salimans, and Max Welling. "Variational dropout and the local reparameterization trick." In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

[KTP19] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P. Parikh, Nicolas Papernot, and Mohit Iyyer. "Thieves on Sesame Street! Model Extraction of BERT-based APIs.", 2019.

[Laa17] Twan van Laarhoven. "L2 regularization versus batch and weight normalization." *arXiv preprint arXiv:1706.05350*, 2017.

[LAG19] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. "Certified robustness to adversarial examples with differential privacy." In *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 656–672. IEEE, 2019.

[LCY20] Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. "Rethinking the Hyperparameters for Fine-tuning." In *International Conference on Learning Representations*, 2020.

[LDS90] Yann LeCun, John S Denker, and Sara A Solla. "Optimal brain damage." In *Advances in neural information processing systems*, pp. 598–605, 1990.

[LGD18] Xuhong Li, Yves Grandvalet, and Franck Davoine. "Explicit Inductive Bias for Transfer Learning with Convolutional Networks." In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2825–2834, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[LK18] Jaewoo Lee and Daniel Kifer. "Concentrated differentially private gradient descent with adaptive per-iteration privacy budget." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1656–1665, 2018.

[LM17] Tsung-Yu Lin and Subhransu Maji. "Improved bilinear pooling with cnns." *arXiv preprint arXiv:1707.06772*, 2017.

[LRM15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. "Bilinear cnn models for fine-grained visual recognition." In *Proceedings of the IEEE international conference on computer vision*, pp. 1449–1457, 2015.

[LSB12] Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. "A simpler approach to obtaining an O (1/t) convergence rate for the projected stochastic subgradient method." *arXiv preprint arXiv:1212.2002*, 2012.

[LTL21] Xuechen Li, Florian Tramèr, Percy Liang, and Tatsunori Hashimoto. "Large Language Models Can Be Strong Differentially Private Learners." *arXiv preprint arXiv:2110.05679*, 2021.

[LWY19] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. "Enhanced Convolutional Neural Tangent Kernels." *arXiv preprint arXiv:1911.00809*, 2019.

[LXS19] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. "Wide neural networks of any depth evolve as linear models under gradient descent." In *Advances in neural information processing systems*, pp. 8570–8581, 2019.

[LXT18] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. "Visualizing the loss landscape of neural nets." In *Advances in Neural Information Processing Systems*, pp. 6389–6399, 2018.

[Man13] Alessandro Mantelero. "The EU Proposal for a General Data Protection Regulation and the roots of the 'right to be forgotten'." *Computer Law & Security Review*, **29**(3):229–235, 2013.

[Mar14] James Martens. "New insights and perspectives on the natural gradient method." *arXiv preprint arXiv:1412.1193*, 2014.

[McA13] David McAllester. "A PAC-Bayesian tutorial with a d ropout bound." *arXiv preprint arXiv:1307.2118*, 2013.

[MG15] James Martens and Roger Grosse. "Optimizing neural networks with kronecker-factored approximate curvature." In *International conference on machine learning*, pp. 2408–2417, 2015.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models." In *Proceedings of the International Conference on Machine Learning*, volume 30, p. 3, 2013.

[MKK17] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. "Deletion-robust submodular maximization: Data summarization with "the right to be forgotten"." In *International Conference on Machine Learning*, pp. 2449–2458, 2017.

[MKR13] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. "Fine-Grained Visual Classification of Aircraft." Technical report, 2013.

[MLL20] Fangzhou Mu, Yingyu Liang, and Yin Li. "Gradients as features for deep representation learning." *arXiv preprint arXiv:2004.05529*, 2020.

[MNS20] Vidya Muthukumar, Adhyyan Narang, Vignesh Subramanian, Mikhail Belkin, Daniel Hsu, and Anant Sahai. "Classification vs regression in overparameterized regimes: Does the loss function matter?" *arXiv preprint arXiv:2005.08054*, 2020.

[MRK13] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. "Fine-grained visual classification of aircraft." *arXiv preprint arXiv:1306.5151*, 2013.

[MZH19] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. "Data Poisoning against Differentially-Private Learners: Attacks and Defenses." In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4732–4738. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[NBS18] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks." In *International Conference on Learning Representations*, 2018.

[Nes13] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In *Proceedings of the International Conference on Machine Learning*, 2010.

[NHA20] Cuong V Nguyen, Tal Hassner, Cedric Archambeau, and Matthias Seeger. "LEEP: A New Measure to Evaluate Transferability of Learned Representations." *arXiv preprint arXiv:2002.12462*, 2020.

[NRS20] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. "Descent-to-Delete: Gradient-Based Methods for Machine Unlearning.", 2020.

[NVL15] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. "Adding gradient noise improves learning for very deep networks." *arXiv preprint arXiv:1511.06807*, 2015.

[NWC11] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning." 2011.

[NZ06] Maria-Elena Nilsback and Andrew Zisserman. "A Visual Vocabulary for Flower Classification." In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pp. 1447–1454, 2006.

[PAE17] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. "Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data." In *Proceedings of the International Conference on Learning Representations*, 2017.

[Pea94] Barak A Pearlmutter. "Fast exact multiplication by the Hessian." *Neural computation*, **6**(1):147–160, 1994.

[PSM18] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. "Scalable Private Learning with PATE." In *International Conference on Learning Representations*, 2018.

[PTD17] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. "Knock knock, who's there? Membership inference on aggregate location data." *arXiv preprint arXiv:1708.06145*, 2017.

[PVZ12] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. "Cats and Dogs." In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[QT09] Ariadna Quattoni and Antonio Torralba. "Recognizing indoor scenes." In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420. IEEE, 2009.

[RAG22] Mohamad Rida Rammal, Alessandro Achille, Aditya Golatkar, Suhas Diggavi, and Stefano Soatto. "On Leave-One-Out Conditional Mutual Information For Generalization." *arXiv preprint arXiv:2207.00581*, 2022.

[RAP18] Sivaramakrishnan Rajaraman, Sameer K Antani, Mahdieh Poostchi, Kamolrat Silamut, Md A Hossain, Richard J Maude, Stefan Jaeger, and George R Thoma. "Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images." *PeerJ*, **6**:e4568, 2018.

[RKH21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning transferable visual models from natural language supervision." *arXiv preprint arXiv:2103.00020*, 2021.

[RRL18] Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. "Membership Inference Attack against Differentially Private Deep Learning Model." *Trans. Data Priv.*, **11**(1):61–79, 2018.

[SA19] Ravid Schwartz-Ziv and Alexander A Alemi. "Information in Infinite Ensembles of Infinitely-Wide Neural Networks." *arXiv preprint arXiv:1911.09189*, 2019.

[SAS14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

[SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. "Stochastic gradient descent with differentially private updates." In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 245–248. IEEE, 2013.

[SDB14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." *arXiv preprint arXiv:1412.6806*, 2014.

[SDO19] Alexandre Sablayrolles, Matthijs Douze, Yann Ollivier, Cordelia Schmid, and Hervé Jégou. "White-box vs black-box: Bayes optimal strategies for membership inference." *arXiv preprint arXiv:1908.11229*, 2019.

[SL07] Alexander Strehl and Michael Littman. "Online linear regression and its application to model-based reinforcement learning." *Advances in Neural Information Processing Systems*, **20**:1417–1424, 2007.

[SRS17] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. "Machine learning models that remember too much." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 587–601. ACM, 2017.

[SS15] Reza Shokri and Vitaly Shmatikov. "Privacy-preserving deep learning." In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1310–1321. ACM, 2015.

[SSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. "Membership inference attacks against machine learning models." In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18. IEEE, 2017.

[SST21] Shuang Song, Thomas Steinke, Om Thakkar, and Abhradeep Thakurta. "Evading the Curse of Dimensionality in Unconstrained Private GLMs." In *International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 2638–2646. PMLR, 13–15 Apr 2021.

[SWT15] Yi Sun, Xiaogang Wang, and Xiaoou Tang. "Deeply learned face representations are sparse, selective, and robust." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2892–2900, 2015.

[SXX20] Yantao Shen, Yuanjun Xiong, Wei Xia, and Stefano Soatto. "Towards backward-compatible representation learning." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6368–6377, 2020.

[TB21] Florian Tramèr and Dan Boneh. "Differentially private learning needs better features (or much more data)." *ICLR*, 2021.

[TLG19] Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. "Demystifying Membership Inference Attacks in Machine Learning as a Service." *IEEE Transactions on Services Computing*, 2019.

[Tor18] Lydia de la Torre. "A guide to the california consumer privacy act of 2018." *Available at SSRN 3275571*, 2018.

[TSC18] Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. "An empirical study of example forgetting during deep neural network learning." *arXiv preprint arXiv:1812.05159*, 2018.

[UNK21] Archit Uniyal, Rakshit Naidu, Sasikanth Kotti, Sahib Singh, Patrik Joslin Kenfack, Fatemehsadat Mireshghallah, and Andrew Trask. "DP-SGD vs PATE: Which Has Less Disparate Impact on Model Accuracy?" *arXiv preprint arXiv:2106.12576*, 2021.

[VKB23] Nikhil Vyas, Sham Kakade, and Boaz Barak. "Provable copyright protection for generative models." *arXiv preprint arXiv:2302.10870*, 2023.

[VSB18] Koen Lennart van der Veen, Ruben Seggers, Peter Bloem, and Giorgio Patrini. "Three tools for practical differential privacy." *PPML18: Privacy Preserving Machine Learning, NIPS 2018 Workshop*, 2018.

[Wan19] Yu-Xiang Wang. "Per-instance differential privacy." *Journal of Privacy and Confidentiality*, **9**(1), 2019.

[WBM10] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. "Caltech-UCSD Birds 200." Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[WBW11] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. "The caltech-ucsd birds-200-2011 dataset." 2011.

[WDD20] Yinjun Wu, Edgar Dobriban, and Susan B Davidson. "DeltaGrad: Rapid retraining of machine learning models." *arXiv preprint arXiv:2006.14755*, 2020.

[WG19] Lingxiao Wang and Quanquan Gu. "Differentially private iterative gradient hard thresholding for sparse learning." In *28th International Joint Conference on Artificial Intelligence*, 2019.

[WRS17] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. "The marginal value of adaptive gradient methods in machine learning." In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

[WZ20] Jun Wang and Zhi-Hua Zhou. "Differentially private learning with small public data." In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6219–6226, 2020.

[YNB21] Da Yu, Saurabh Naik, Arturs Backurs, Sivakanth Gopi, Huseyin A Inan, Gautam Kamath, Janardhan Kulkarni, Yin Tat Lee, Andre Manoel, Lukas Wutschitz, et al. "Differentially Private Fine-tuning of Language Models." *arXiv preprint arXiv:2110.06500*, 2021.

[YZC21] Da Yu, Huishuai Zhang, Wei Chen, and Tie-Yan Liu. "Do not Let Privacy Overbill Utility: Gradient Embedding Perturbation for Private Learning." In *International Conference on Learning Representations*, 2021.

[ZCD17] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. "mixup: Beyond empirical risk minimization." *arXiv preprint arXiv:1710.09412*, 2017.

[ZCH21] Xinwei Zhang, Xiangyi Chen, Mingyi Hong, Zhiwei Steven Wu, and Jinfeng Yi. "Understanding Clipping for Federated Learning: Convergence and Client-Level Differential Privacy." *arXiv preprint arXiv:2106.13673*, 2021.

[ZH20] Ligeng Zhu and Song Han. "Deep leakage from gradients." In *Federated learning*, pp. 17–31. Springer, 2020.

[ZWB21] Yingxue Zhou, Steven Wu, and Arindam Banerjee. "Bypassing the Ambient Dimension: Private {SGD} with Gradient Subspace Identification." In *International Conference on Learning Representations*, 2021.

[ZWX19] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. "Three Mechanisms of Weight Decay Regularization." In *International Conference on Learning Representations*, 2019.

[ZYC20] Yuqing Zhu, Xiang Yu, Manmohan Chandraker, and Yu-Xiang Wang. "Private-knn: Practical differential privacy for computer vision." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11854–11862, 2020.