Master's thesis
Master's Programme in Data Science

Hyperparameters and neural architectures in differentially private deep learning

Marlon Tobaben

June 13, 2022

Supervisor(s):	Associate Professor Antti Honkela
Examiner(s):	Associate Professor Antti Honkela
	Doctor Razane Tajeddine

University of Helsinki
Faculty of Science

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Degree programme Master's Programme in Data Science

Tekijä — Författare — Author
Marlon Tobaben

Työn nimi — Arbetets titel — Title
Hyperparameters and neural architectures in differentially private deep learning

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year June 13, 2022	Sivumäärä — Sidantal — Number of pages 65

Tiivistelmä — Referat — Abstract

Using machine learning to improve health care has gained popularity. However, most research in machine learning for health has ignored privacy attacks against the models. Differential privacy (DP) is the state-of-the-art concept for protecting individuals' data from privacy attacks. Using optimization algorithms such as the DP stochastic gradient descent (DP-SGD), one can train deep learning models under DP guarantees. This thesis analyzes the impact of changes to the hyperparameters and the neural architecture on the utility/privacy tradeoff, the main tradeoff in DP, for models trained on the MIMIC-III dataset. The analyzed hyperparameters are the noise multiplier, clipping bound, and batch size. The experiments examine neural architecture changes regarding the depth and width of the model, activation functions, and group normalization. The thesis reports the impact of the individual changes independently of other factors using Bayesian optimization and thus overcomes the limitations of earlier work. For the analyzed models, the utility is more sensitive to changes to the clipping bound than to the other two hyperparameters. Furthermore, the privacy/utility tradeoff does not improve when allowing for more training runtime. The changes to the width and depth of the model have a higher impact than other modifications of the neural architecture. Finally, the thesis discusses the impact of the findings and limitations of the experiment design and recommends directions for future work.

ACM Computing Classification System (CCS):
Applied computing → Life and medical sciences → Health informatics
Computing methodologies → Machine learning
Security and privacy → Human and societal aspects of security and privacy → Privacy protections

Avainsanat — Nyckelord — Keywords
differential privacy, deep learning, hyperparameter optimization, Bayesian optimization

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Acknowledgements

I am grateful to my supervisor, Antti Honkela, and my other colleagues in the research program *Privacy-preserving and secure AI* by the Finnish Center for Artificial Intelligence (FCAI) for all the feedback and advice during this project. The DataLit project has generously supported this work by providing my salary. DataLit is funded by the Strategic Research Council at the Academy of Finland, decision number 336032. I wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

This thesis is the final project of my Master of Science in Data Science. I want to thank both the student and teaching community for supporting my studies during these exceptional times. Special thanks to the Data Science study program staff, especially Professor Jussi Kangasharju and Doctor Pirjo Moen. They create an environment that makes it easy to study. I would also like to thank Assistant Professor Luigi Acerbi, with whom I worked during 2021 as a research assistant, for the opportunity to work in research, the fruitful debates, and the gained new skills. TKO-äly ry and Hyde ry hosted most of the events I attended during my studies. Thanks for the sauna evenings, sitsit, and hikes. Finally, thanks to the friends that made my studies especially enjoyable. The following is a list of some people: Anna, Aino, Bernardo, Dmitrijs, Elio, Mikko, Niclas, Niko, Roope, Sandra and Sini.

Contents

1. Introduction

Machine learning (ML) complements humans in most domains, with some applications aiming to enhance human health. The last breakthrough in that direction is the second version of *AlphaFold* [JEP$^+$21] that solved large parts of the protein prediction challenge, which is essential for enhancing medicine. The trend of ML use expands gradually into large-scale healthcare, where more and more hospitals adopt electronic health record (EHR) systems, leading to sufficient data for machine learning. The scientific community has already published databases originating from EHR systems that document complete stays of patients in the hospital [JPS$^+$16, JBP$^+$21]. ML models can assist the medical personnel by predicting different kinds of tasks such as the length of a stay in the hospital, the probability of mortality, and in diagnosing diseases [PMCL18, HKK$^+$19, WMC$^+$20]. More sophisticated deep learning (DL) models originating from natural language processing (NLP) can be applied to EHR data because both EHR and NLP data are sequential [KIS$^+$21].

While the use of ML in healthcare is promising, the current work on ML in healthcare often ignores one aspect, privacy, that will be the core of this master thesis. Due to ethical reasons and laws like the General Data Protection Regulation (GDPR) in the European Union, it is essential to protect the privacy of the patients when using their data. In Finland, the use of healthcare data for secondary use cases has a long history, and a dedicated data permit authority for the social and health care sector called Findata has been founded [Fin21]. Such dedicated authority points out that while using data for healthcare is considered an advantage for the population, it is crucial to address concerns and risks.

Outside of healthcare, at least two cases in which releasing a dataset in good faith resulted in privacy violations are commonly known. AOL released a subset of its search logs to facilitate research in 2006. Even though AOL obscured the users' identities, journalists of the New York Times [BZ06] were able to identify and interview one of the users within a few days after the release of the dataset. The journalists were able to identify the user based on the search queries that included family names and location data. Because some of the queries contained more sensitive information, such as illnesses, one can easily understand the danger of releasing datasets without precautions. Nevertheless, even if the queries had not contained such sensitive information, the users' privacy would have been violated.

The other example originates from the Netflix Prize. The Netflix Prize was a competition by Netflix that encouraged researchers to propose recommender engines based on a dataset containing time-stamped movie ratings of anonymized users. Narayanan and Shmatikov [NS08] leveraged auxiliary information from the Internet Movie Database (IMDb), including ratings for movies and the users' names. Using this auxiliary information, they identified users from the Netflix prize dataset using correlations between the ratings from the two sources. As Narayanan and Shmatikov [NS08] point out, this potentially harmless information about movie ratings can leak sensitive information such as political views. Furthermore, this example shows that powerful attackers can easily overcome trivial privacy protection mechanisms.

In order to protect datasets and machine learning models from privacy attacks, the state-of-the-art approach is using differential privacy (DP) [DMNS06], which provides a provable guarantee against privacy attacks. The US census is using DP to release its dataset [Abo18]. Tech giants like Google [Gue21] and Microsoft [Kah20] are also conducting extensive research both on methods and application areas of DP. It is preferred over previous approaches like k-anonymity [SS98], which has multiple disadvantages in comparison to DP, such as its unsuitability for data with a high number of dimensions [Agg05].

The central tradeoff in DP is between the provided privacy guarantee and the results' utility [AGD$^+$20]. In order to achieve the best utility with ML one has to determine the best configuration. Determining this configuration is already challenging for the non-DP case. It becomes more challenging when training under DP, because of the additional privacy constraint and the observation that training a model under DP with an identical configuration as in the non-DP case does not have to yield the best possible utility. Several articles suggest that modifications to the ML process are necessary to obtain the best tradeoff between privacy and utility [TB20, PTS$^+$21]. Investigating those modifications will be a significant part of this thesis. Further details of DP and possible modifications to the ML process will be introduced in Chapters 2 and 3.

This Chapter 1 will introduce the research objectives of this thesis, and Chapter 7 will assess the completion of those research objectives. The thesis aims to fulfill the following six research objectives:

1. Review the necessary background on differential privacy, adaptation of deep learning, and related works.

2. Analyse the impact of the hyperparameters introduced when training under differential privacy on the privacy/utility tradeoff.

3. Examine the efficacy of neural architecture changes proposed by previous work.

4. Propose a sensible experimental setup to measure objectives 2 and 3.
5. Compare the non-private baseline models to the private differential models and assess the differentially private models' feasibility.
6. Identify limitations of the thesis and directions for possible further work.

The remainder of this master's thesis is structured as follows: The first two chapters introduce the necessary background. Chapter 2 introduces Differential Privacy, especially its adaptation to ML, a differentially private version of stochastic gradient descent, privacy attacks, and theoretical privacy boundaries in practice. Chapter 3 focuses on Deep Learning under DP with a focus on activation functions, batch and group normalization, and neural network architecture search. Afterward, the used dataset, the benchmark study, the hyperparameter search strategy, and related experiments are reviewed in Chapter 4. Subsequently, the conducted hyperparameter and neural architecture experiments are described in Chapter 5. Finally, the thesis concludes with a discussion of the main findings and limitations of the thesis in Chapter 6 and a conclusion in Chapter 7.

2. Differential Privacy

This chapter will first introduce the concept of DP in general and afterward discuss methods for applying DP to ML, especially the differential private stochastic gradient descent (DP-SGD). Furthermore, the chapter will motivate the need for DP by giving an insight into selected privacy attacks. The chapter will conclude with the relation between theoretical and empirical privacy bounds obtained by attacking models.

After decade-long but sparse research on the topic of privacy in databases, Dwork et al. [DMNS06] formalized privacy guarantees using the concept of differential privacy (DP)[1] in 2006. This contribution allowed for subsequent research, e.g., improvements in the privacy guarantees and applications to machine learning. Because of the extraordinary impact of the concept, the authors received the Gödel prize in 2017[2].

A good starting point for understanding DP is the survey method randomized response [War65, GASH69] originating from social sciences. Before answering an uncomfortable question, the survey participants figure out their answer in secret. They flip a coin and only answer truthfully when the coin shows tails. When the coin shows heads, the participants respond randomly by flipping another coin. The participants can always claim their answer was random, but the researchers can still estimate the actual distribution among all participants [DR$^+$14, p.15-16]

Nowadays, the (ϵ, δ)-DP [DKM$^+$06] is being used most of the time. Definition 2.1 (taken in this form from Dwork and Roth [DR$^+$14]) states this version of DP. For understanding the definition, two concepts have to be defined: First, a randomized mechanism is an algorithm that includes randomness similar to the randomized response described above [DR$^+$14]. Second, adjacent inputs d, d' depend on the application of DP. In machine learning, these can be, e.g., two training datasets that differ in one training example [ACG$^+$16].

The core idea behind DP looks at the ratio of the probabilities for obtaining any output of a randomized mechanism for any pairs of adjacent inputs d, d'. The probability

1. This thesis will mainly use the acronym DP for differential privacy. The wording *under DP* means that a mechanism is executed under differential privacy guarantees.
2. The 2017 Gödel Prize citation from the 2017 Gödel Prize committee is available online at `https://sigact.org/prizes/gödel/citation2017.pdf` (accessed on 13.04.2022)

is taken over the randomized mechanism. This ratio is bounded by e^ϵ where $\epsilon \in \mathbb{R}_{\geq 0}$. The number $\delta \in \mathbb{R}_{\geq 0}$ introduces a small chance that the bound of e^ϵ is violated. The value of δ should be $\delta < \frac{1}{d}$ where d is the size of the dataset [DR+14, ACG+16]. When $\delta = 0$, the definition is the same as the ϵ-DP proposed in the original DP article [DMNS06].

Definition 2.1 ((ϵ, δ)-differential privacy)

A randomized algorithm $\mathcal{M}$ with domain $\mathbb{N}^{|\mathcal{X}|}$ is (ϵ, δ)-diffentially private if for all $\mathcal{S} \subseteq \text{Range}(\mathcal{M})$ and for all $x, y \in \mathbb{N}^{|\mathcal{X}|}$ such that $||x - y||_1 \leq 1$:

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

where the probability space is over the coin flips of the mechanism $\mathcal{M}$.

Desfontaines [Des19] provides an intuitive example for interpreting the bounding by ϵ in his blog on DP. Assume a strong attacker wants to attack a randomized mechanism $\mathcal{M} : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathcal{R}$ and knows the complete database but is unsure whether an entry is part of the database or not. In other words, the attacker does not know if the database is d or d' but might have a prior probability of 0.5 that the database is d'. After observing the output of $\mathcal{M}$ that satisfies $\epsilon = 1.1$, the probability of the attacker can be at minimum ≈ 0.25 and ≈ 0.75 at maximum. With an ϵ that is twice the size ($\epsilon = 2.2$), the probability will be already at minimum ≈ 0.1 and ≈ 0.9 at maximum. This means that the higher the ϵ is, the more the attacker can learn from the randomized mechanism $\mathcal{M}$. For (ϵ, δ)-DP [DKM+06] there is a probability of δ that the privacy bound does not hold and the attacker can learn more, which is, as discussed previously, usually chosen to be tiny.

There are two properties of DP that build the foundation of the machine learning algorithms that will be described in subsequent sections of the thesis. The first property is that the composition of multiple differentially private mechanisms is also differentially private. Theorem 2.1 states this fact and results from the analysis by Dwork et al. [DKM+06] and is taken in this form from the comprehensive DP book *The Algorithmic Foundations of Differential Privacy* by Dwork and Roth [DR+14]. The privacy loss of the combination is at maximum, the sum of the individual (ϵ, δ) values. There are more advanced composition theorems that result in a lower privacy loss, and they can be found in Section 3.5 of the book from Dwork and Roth [DR+14].

Theorem 2.1 (Composition of (ϵ, δ)-mechanisms)

Let $\mathcal{M}_i : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathcal{R}_i$ be an (ϵ_i, δ_i)-differential private algorithm for $i \in [k]$. Then if $\mathcal{M}_{[k]} : \mathbb{N}^{|\mathcal{X}|} \rightarrow \prod_{i=i}^{k} \mathcal{R}_i$ is definied to be $\mathcal{M}_{[k]}(x) = (\mathcal{M}_1(x)), ..., \mathcal{M}_k(x))$, then $\mathcal{M}_{[k]}$ is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$-differential private.

The second property clarifies that a (ϵ, δ)-mechanism cannot be less private when applying postprocessing. Theorem 2.2 formalizes this and can be found in the book from Dwork and Roth [DR+14, p.19].

Theorem 2.2 (Postprocessing of a (ϵ, δ)-mechanism)

Let $\mathcal{M} : \mathbb{N}^{|\mathcal{X}|} \rightarrow R$ be a randomized algorithm that is (ϵ, δ)-differentially private. Let $f : R \rightarrow R'$ be an arbitrary randomized mapping. Then $f \circ \mathcal{M} : \mathbb{N}^{|\mathcal{X}|} \rightarrow R'$ is (ϵ, δ)-differentially private

One might ask how to achieve (ϵ, δ)-DP. A simple method to achieve DP of a function uses the Laplace mechanism, proposed by Dwork et al. [DMNS06]. Before applying the Laplace mechanism, the ℓ_1 sensitivity of the underlying function has to be determined, which can be found in Definition 2.2 (taken from Dwork and Roth [DR+14]). The ℓ_1 sensitivity is the maximum amount the result can change when one entry of the input dataset is present or absent. The ℓ_1 sensitivity for the sum over $\{0, 1\}$ is 1 [DMNS06].

Definition 2.2 (ℓ_1 sensitivity)

The ℓ_1 sensitivity of a function $\mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$ is:

$$\Delta_f = \max_{\substack{x,y \in \mathbb{N}^{|\mathcal{X}|} \\ ||x-y||_1 = 1}} ||f(x) - f(y)||_1$$

After determining the ℓ_1 sensitivity, one must add noise in proportion to it. The noise will be drawn from a Laplace distribution as can be seen in Definition 2.3. The definition of the Laplace mechanism has been taken from the book by Dwork and Roth [DR+14]. The Laplace mechanism can be used to achieve $(\epsilon, \delta = 0)$-DP.

Definition 2.3 (Laplace Mechanism)

Given any function $f : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^k$, the Laplace mechanism is defined as

$$\mathcal{M}_L(x, f(\cdot), \epsilon) = f(x) + (Y_1, ..., Y_d)$$

where Y_i are i.i.d. random variables drawn from Lap(Δ_f/ϵ).

An example is useful for understanding the Laplace mechanism. When imagining a function with an ℓ_1 sensitivity of 2, one can apply Definition 2.3. For achieving $(\epsilon = 0.5, \delta = 0)$ noise from the Laplace distribution scaled to $b = \frac{2}{0.5} = 4$ has to be added. The noise required for reaching $(\epsilon = 1, \delta = 0)$ has a scale of $b = \frac{2}{1} = 2$.

There are other mechanisms besides the Laplace mechanism. The Gaussian mechanism, for example, adds noise sampled from a Gaussian distribution, where the noise for that is scaled by a so-called ℓ_2 sensitivity. Section 3.5.3 of the book from Dwork and Roth elaborates on the differences between Laplace and Gaussian distribution [DR+14].

2.1 Differentially Private Machine Learning

This subsection will discuss methods to perform machine learning under DP, which is complex as various modifications can transform non-DP learning into DP learning. The changes can happen during preprocessing, training, or afterwards [PMSW18]. This section will discuss the methods of input perturbation, objective perturbation, output perturbation, noisy gradient-based optimization, and private aggregation.

Input perturbation

In input perturbation, each user keeps their data locally and adds noise before transmitting it to a central party that conducts the learning. Thus, the complete training runs on perturbed inputs. While there have been previous ideas leading in a similar direction, adding noise locally has been formulated in a DP-way called local DP in [KLN$^+$11]. The advantages of input perturbation are that the central party is not required to be trustworthy. Additionally, every party can decide on the amount of locally added noise and consequently on the privacy guarantee [ACPP18]. The disadvantage is that to ensure privacy, each party has to add a lot of noise, resulting in a high loss of utility. Thus, local privacy is infeasible in many applications. It is, however, viable in applications where a lot of data is available, e.g., in use-cases of Google and Apple [ACPP18].

Objective perturbation

Objective perturbation accomplishes DP by adding noise to the objective function instead. The goal of the training is to minimize the noisy objective function, and then the model parameters that minimize the noisy objective function will be released [ZZX$^+$12]. Objective perturbation is limited to specific models: linear and logistic regression models can be optimized under DP using objective perturbation, but, e.g., cases where the loss function is non-convex are unsuitable for objective pertubation [ZZX$^+$12, CMS11, JE19]. As will be discussed in Chapter 3, non-convex loss functions are standard in deep learning models. There are techniques to replace a non-convex loss function with a convex loss function, but this requires further analysis, more changes to the model, and might limit the performance of the model [JE19].

Output pertubation

Adding noise to the output of a function before releasing the output is known as output perturbation [DMNS06]. The output perturbation idea can be used in machine learning as well. Complicated analysis is necessary to figure out the necessary amount of noise to add to the outputs [CMS11]. Consequently, it will not be further considered in this thesis.

Noisy gradient-based optimisation

Another direction of research is to add noise to the gradients during gradient-based optimization as done for example in DP-SGD [RA12, SCS13, ACG+16]. Modifying a non-DP approach that optimizes a model with a gradient-based algorithm such as stochastic gradient descent towards a model that uses DP-SGD is trivial. Only the optimizer has to be changed, and no model-specific analysis is required. In practice this can be done out of the box with privacy libraries like *Opacus* [YSS+21] for *Pytorch* [PGM+19] or *TensorFlow Privacy* for *TensorFlow* [ABC+16].

However, to achieve optimal utility with DP-SGD, further analysis is required, and also the model may have to be modified. Section 2.2 will elaborate on DP-SGD in more detail as it is the differential private machine learning method that this thesis will use.

Private aggregation

The private aggregation of teacher ensembles (PATE) [PAE+17, PSM+18] is another approach that yields high utility with the constraint that it requires to have additional public data.

The sensitive data is split into n non-overlapping datasets, and in total n so-called teacher models are trained on them (one teacher model per dataset). There are no restrictions on the algorithm used to train the teachers as long as it is suitable for the underlying dataset. The teachers form an ensemble that votes together on the label of a data point, and then the noise is added to this aggregated prediction. The teacher ensemble returns the label with the highest (noisy) count. The teachers are being utilized to perform semi-supervised learning with unlabelled publicly available data and train a student model. The number of predictions from the teacher is limited as otherwise, the privacy loss would further increase. Only the student model is being released and could be attacked [PSM+18].

Further enhancements and subsequent work are out of scope for this thesis because PATE is unsuitable for the thesis even though the idea is promising. PATE depends on the availability of public data and which is not available in the setup of this master thesis (see Section 4.1 for details on the used dataset).

2.2 Differentially Private SGD

A line of work [RA12, SCS13, ACG+16] introduced differentially private stochastic gradient descent (DP-SGD), which is a modification of the widely used stochastic gradient descent (SGD) to allow for optimization of models under DP. DP-SGD consists of two sub-concepts that will be introduced in this section. The first sub-concept includes the changes to the SGD and the newly introduced hyperparameters. The second sub-concept deals with accounting for privacy and is the so-called privacy accountant.

Changes from SGD to DP-SGD

To understand the differences between the default SGD and DP-SGD, the most straightforward way is to go through the DP-SGD algorithm from Abadi et al. [ACG+16] as displayed in Algorithm 1 and highlight the aspects that differ between SGD and DP-SGD. Several hyperparameters are only part of DP-SGD. These are the subsampling ratio, the clipping bound, and the noise multiplier. The additional hyperparameters will also be addressed when discussing the differences between the algorithms.

Algorithm 1 Differentially private stochastic gradient descent (DP-SGD)
(The algorithm is almost identical to the one in [ACG+16].)

```
Input: data points $\{x_1, ..., x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\Sigma_i \mathcal{L}(\theta, x_i))$
Hyperparameters: learning rate $\eta_t$, noise multiplier $\sigma^2$, group size $L$, clipping bound $C$
1: Initialize $\theta_0$ randomly.
2: for $t \in [T]$ do
3:     Take a random sample $L_t$ with sampling probability $L/N$
4:     Compute gradient
5:     For each $i \in L_t$ compute $g_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$
6:     Clip gradient
7:     $\tilde{g}_t(x_i) \leftarrow g_t(x_i)/\max(1, \frac{||g_t(x_i)||_2}{C})$
8:     Add noise
9:     $\tilde{g}_t \leftarrow \frac{1}{|L_t|}(\Sigma_i \tilde{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 I))$
10:    Descent
11:    $\theta_{t+1} = \theta_t - \eta_t \tilde{g}_t$
12: end for
Output: $\theta_T$ and overall privacy cost $(\epsilon, \delta)$ computed using a privacy accountant.
```

The initialization of DP-SGD does not differ from SGD, and also, the loop over multiple steps does not differ. The first difference is in line 3 of Algorithm 1: In SGD, the batches are selected with a fixed size and are non-overlapping within one epoch, but in

DP-SGD, batches are selected with a probability which is also called subsampling ratio q. The random selection implies that each sample has a probability of $q = L/N$ to be chosen for the batch in every optimization step [ACG+16]. While being unlikely, one sample could be part of every optimization step. This subsampling has an effect on privacy accounting which will be discussed later.

After computing the so-called per-example gradients, the gradients will be clipped to have a maximum L2-norm of the clipping bound C (see line 7 of Algorithm 1). The gradient clipping is necessary to bound the sensitivity of the individual samples.

The clipping bound C also influences the amount of noise added to the gradients in a later step. Choosing the clipping bound is not a trivial task as a too low C will reduce the usefulness of the gradient, whereas a too high clipping bound C results in a lot of noise being added [ACG+16]. As discussed by Chen et al. [CWH20], the effects of the clipping bound depend on the distribution of the gradients, and thus general claims are difficult. In practice, one should try out different clipping bounds C or use more sophisticated techniques such as the adaptive clipping proposed by Andrew et al. [ATMR21]. The adaptive clipping bound is based on setting the clipping norm to a DP estimated quantile of the gradient's L2-norm distribution in every optimizer step.

One can find the last difference between DP-SGD and SGD in line 9 of the Algorithm 1 where noise is added to the clipped gradients. The noise is drawn from a Gaussian distribution with a variance of $\sigma^2 C^2$. The variance is a product of the clipping bound C and the so-called noise multiplier σ^2. The noise multiplier is inversely related to the privacy loss ϵ, meaning increasing the noise multiplier decreases the privacy loss per update [ACG+16].

Privacy Accountant

A privacy accountant computes the privacy budget spent during the optimization of the algorithm. One can set the noise multiplier so that each step of the optimizer would spend exactly (ϵ, δ)-privacy. But when taking into account that the sample for each step is taken randomly with a subsampling ratio $q = \frac{L}{N}$, according to the so-called privacy amplification theorem [KLN+11, BBKN14], each step is $(O(q\epsilon), q\delta)$-private.

The moments accountant from Abadi et al. [ACG+16] reaches a tighter bound on the privacy loss than previous works. For Algorithm 1 it results in a $(O(q\epsilon\sqrt{T}), \delta)$-privacy loss. The Rényi DP (RDP) [Mir17] formulizes the ideas of the moments accountant. The accountant of the privacy library *opacus* uses this RDP-accountant[1], and thus the RDP-accountant will also be used in this thesis. Another technique uses Fast Fourier transforms

1. *Privacy Engine* in the documentation of *Opacus* [YSS+21] available at https://opacus.ai/api/privacy_engine.html (accessed on 06.04.2022)

(FFT) to compute tighter bounds than RDP [KJH20].

2.3 Privacy Attacks against Machine Learning

This thesis discusses DP which is essential to protect against privacy attacks. Thus, to understand the importance of the contributions of this thesis, knowledge of privacy attacks is vital. This section will give a brief overview of the fundamentals of privacy attacks.

In practice, privacy attacks depend on the application area, the machine learning method, and access to the model. The latter can be distinguished between either a black-box or a white-box scenario. In a black-box scenario, the attacker[1] can use the trained model to make predictions, e.g., using an API. The inference is also possible in the white-box scenario; however there the attacker has full access to the model. Both access types to models are commercially offered on marketplaces as some users are providing access to their trained model through an API, and others are even enabling downloading the trained model [FJR15].

The following paragraphs will outline two privacy attacks: membership inference and attribute inference. Those attacks are closely connected to DP [JE19]. Also, a brief overview of other attacks will be given.

Membership inference

The goal of a membership inference attack is to infer whether an individual's data has been used during the training of the machine learning model or not. The idea behind the attack is that machine learning models react differently when inferring a data point based on the fact whether it has been part of the training set or not [SSSS17].

One might question why membership inference is harmful. One realistic scenario where membership inference is dangerous is when the attacked model concerns drug response in cancer research. The knowledge that individuals have been part of a dataset that deals with drug response in cancer research can expose the fact that the individual had/has cancer [SSSS17].

The initial membership inference attack uses so-called shadow models, which are models that are architecture-wise similar to the attacked model. The models are trained on comparable but available data. These shadow models are used for training an attack model in a supervised manner. The attack model can then distinguish whether or not a data point has been a part of the training data or not. The number of shadow models increases the efficiency of the attack [SSSS17].

1. This thesis uses the words attacker and adversary synonymously.

While the shadow model attack is efficient according to the evaluations of Shokri et al. [SSSS17], it requires a high amount of computation. Subsequent work advanced the initial attack: For example, the attack of Yeom et al. [YGFJ18] requires significantly less computational power but the additional information about the loss function and achieves slightly worse results. Membership inference is used in privacy research because it resembles precisely the attack that DP tries to protect against [JE19].

Attribute inference

The attribute inference attack aims at extracting unknown attribute values using known attribute values of the same data subject. Fredrikson et al. [FLJ$^+$14] call the attack model inversion attack and formalize the general idea, which previous authors already observed. For instance, Korolova [Kor10] attacked the Facebook system for microtargeted advertisements and Calandrino et al. [CKN$^+$11] recommender systems.

It is also possible to attack more complicated systems. For example, Fredrikson et al. [FJR15] attacked a face recognition system. One can find an example reconstruction from this attack in the figure from Fredrikson et al. [FJR15, p.1323]. It is surprising how accurate the reconstruction is given that the adversary only has access to the facial recognition model, the returned class confidence scores, and the individual's label.

Formally, the attack works as follows: A black-box attack for a model $\mathcal{M} : X \to Y$, that maps inputs X to outputs Y, requires that the adversary possesses some information about the targeted individual α and wants to extract unknown information about α that is part of X. We will call this unknown information x_u^α. The information available to the adversary includes the output of the model y^α and either other data features of $x^\alpha \setminus \{x_u^\alpha\}$ or information about the joint-distribution of the training set of $\mathcal{M}$ [FLJ$^+$14, p. 4].

The idea behind the attack is more straightforward when looking at an example (highly related to the attack on warfarin dosing described by Fredrikson et al. [FLJ$^+$14]). The model $\mathcal{M}$ predicts the dosage of a drug that an individual α takes based on height, age, weight, and genetic information. An attacker would have to know the dosage of the drug that α takes and other information x_u^α such as weight, age, and height. The attacker could then try to infer which genetic information α has.

It is debatable if model inversion is always the essential part of the privacy violation or if knowing the dosage of a drug and utilizing publically available statistical information would lead to the same results [McS16].

Practically the attack can use a maximum a posteriori (MAP) estimator. It is then possible to find x_u^α that is most likely to have been observed given the known values. While one can apply this approach to various models, it requires trying out all possible values of x_u^α in a brute-force manner and is thus not feasible in all contexts. For black-box attacks concerning image recognition, both the number of features and the fact that the value ranges are interval-based make the MAP algorithm impractical. Thus, they require more advanced algorithms [FJR15].

Other attacks

Some other methods for attacking ML are model stealing, hyperparameter stealing, and property inference attacks, which are all not directly related to DP [JE19]. DP can prevent memorization attacks, as studied by Carlini et al. [CLE+19]. Furthermore, poisoning and adversarial training attacks that interfere with the training process threaten the integrity of the model [JE19].

2.4 Theoretical Privacy Boundaries in Practice

The theoretical privacy losses obtained by DP analysis suppose that a powerful adversary will attack the model. For example, in DP-SGD, the theoretical analysis is computed as if the adversary not only can access the final model but would also be able to leverage the intermediate computations of the optimization. Even though this is unrealistic, there is no improved theoretical analysis for the case that the intermediate calculations are unknown to the adversary [NST+21].

One way to obtain more realistic privacy loss estimates is to empirically measure the privacy loss by attacking the model with privacy attacks. Two recent contributions from Malek et al. [MEMP+21], and Nasr et al. [NST+21] show in more detail how privacy loss can be empirically measured. Especially noteworthy is the contribution of Nasr et al. [NST+21] that show how the practical epsilon compares in different scenarios, as can be seen in Figure 2.1.

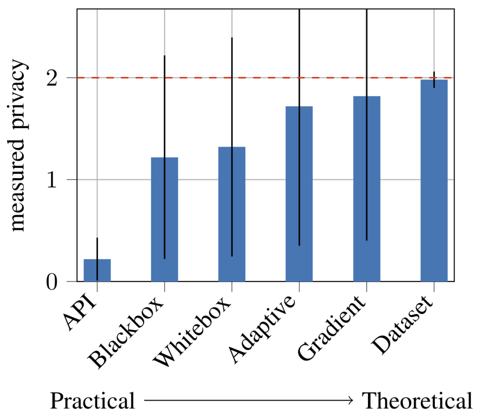


Figure 2.1: Empirically measured privacy loss by adversaries with different powers. The figure is unmodified from the arXiv:2101.04535 version of Nasr et al. [NST+21, p.867] (CC BY 4.0 licence).

The weakest adversary has only access to the model via an API, and thus there, the empirical privacy loss is the lowest. Stronger adversaries have, e.g., access to the model gradients and can modify them. The red line shows the theoretically estimated privacy bound, which is also measured when the most powerful attacker is attacking.

Empirically measuring the privacy led to the discovery that stochastic gradient descent using gradient clipping without adding noise to the clipped gradient results in a weakened performance of privacy attacks [JUO20]. Researchers have also discovered mistakes and bugs in privacy implementations by attacking presumably highly private models [TTS+22].

3. Adaptation of Deep Learning for Differential Privacy

Related ideas to Deep Learning (DL) in the field of artificial intelligence (AI) have been a more or less active research field for roughly half a century. Their popularity varied, and there have been decades where most researchers focused on other areas. The current popularity of DL in research and industry applications started around a decade ago due to more available training data, sophisticated software, and powerful hardware. Researchers discovered only then that methods developed in the 1980s work because it was not feasible to use them before. Nowadays, DL algorithms are the backbone of state-of-the-art machine learning solutions for computer vision, natural language processing, and other domains [GBC16, pp. 11–28]. The DL models used in this thesis and in many applications follow the concept of a multilayer perceptron (MLP), a kind of neural network (NN).

The remainder of this chapter follows two general ideas. On the one hand, it will discuss the necessary fundamentals of DL and, on the other hand, introduce proposed adaptations of DL to allow for a maximized utility under DP guarantees because research suggests altering the model architecture is necessary [TB20, PTS$^+$21]. After an introduction to basic ideas of DL, the sections will cover activation functions, group and batch normalization, and other NN components. The chapter will conclude with an overview of neural network architecture search.

An MLP approximates a function $f^*(x)$ from inputs x to outputs y using the appropriate parameters θ for a function $f(x;\theta) = y$. The function $f(x;\theta) = y$ consists of a chain of multiple intermediate functions, e.g. two functions $f^{(2)}(f^{(1)}(x))$. The intermediate functions are called layers which means that $f^{(1)}$ is the first layer. Because the training dataset does not contain the anticipated values for the layers apart from the output layer, the other layers are called hidden layers.

In contrast to other kinds of neural networks like recurrent neural networks (RNNs), the information in MLPs flows in one direction from inputs to outputs [GBC16, pp. 168-171]. In the thesis, the number of hidden layers plus the output layer is the *depth* of the model; the dimensionality of the layers is called *width*. Figure 3.1 shows an MLP with a depth of three and two hidden layers with a width of five and four.

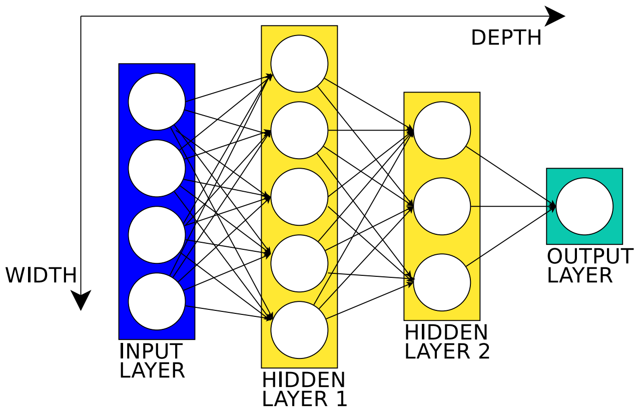


Figure 3.1: A MLP with a depth of three and two hidden layers that have a width of five and four (own illustration by the author).

MLPs overcome traditional linear models by applying non-linear transformations [GBC16, pp. 169–170]. According to the so-called *universal approximation theorem* [HSW89], an MLP with at least one hidden layer and a so-called squashing function (sigmoid and tanh, but not ReLu)[1] can represent any function. Achieving an adequate representation of the function and the generalization of the MLP depends on the size of the hidden layer and the learning algorithm [GBC16, pp. 198–199].

The optimization of MLPs (and NNs in general) is more challenging than the optimization of linear models because the most popular cost functions of NNs are non-convex. The non-convexity results in the fact that the training of NNs most of the time aims for a local minimum of the cost function and thus a good enough minimum of the cost function and not the best possible optimum. The algorithms used to optimize NNs are regularly based on gradient descent [GBC16, pp. 177–178]. The experiments of this thesis use an adaptive stochastic gradient descent variant called Adam introduced by Kingma and Ba [KB14].

It is important to mention that larger DL models behave differently under DP than in the non-DP case: The dimensionality d of both non-DP algorithms as well as DP algorithms influences the error rate of the respective algorithms [SS21, BGM21]. Singhal and Steinke [SS21] introduce a toy example of mean estimation for a distribution supported on $[0, 1]^d$. For this toy example, the relationship between d and the error rate, when n is the number of samples is $\Theta(\frac{\sqrt{d}}{n})$ for the DP case and for the non-private case $\Theta(\sqrt{\frac{log(d)}{n}})$. It depends on the loss function how bad the relationship in practice is, e.g., Bassily et al. [BGM21] distinguish their analysis between smooth and non-smooth, convex and non-convex. One can improve the dependence on d by making assumptions, e.g., that some d do not matter, and introducing a $k << d$. This assumption is applicable in DL because the gradients often tend to follow k and not d [SS21]. These considerations are

1. Refer to the paper of Hornik et al. [HSW89] for more details.

outside of the scope of this master thesis. The conclusion that is important for the rest of this thesis is that in comparison to smaller models, larger models are harder to train under DP than they would be in a non-DP setting.

3.1 Activation Functions

Activation functions are used in neural networks to apply non-linear transformations [GBC16, pp. 169–170]. There are dozens of possible activation functions but recently the rectified linear unit (ReLU) became the state-of-the-art in DL [RZL17]. The equation of the ReLU is simple; as can be seen in Equation (3.1), it returns the value if it is positive and otherwise returns a 0.

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{3.1}$$

Before the widespread use of ReLUs, the sigmoid function (Equation (3.2)) has been widely used. In many cases, using a sigmoid activation functions is challenging because they are only sensitive to values close to 0. They are still used in some architectures, e.g., certain kinds of autoencoders, because in those cases, ReLUs cannot be used [GBC16, p.195].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Another alternative is using the tanh activation function (Equation (3.3)), which can be rewritten using the sigmoid activation function. It regularly performs better than sigmoids [GBC16, p.195].

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{3.3}$$

Many alternatives to the above-listed activation functions have been proposed, but none of them seems to be generally better. In the majority of cases, the above-listed three activation functions are used [RZL17, GBC16]. A list of some other relatively widespread activation functions can be found in Section 6.3.3 of the book *Deep Learning* from Goodfellow et al. [GBC16].

When proposing a new activation function, the main question is how to prove that it is generally better than others. Ramachandran et al. [RZL17] conducted extensive automatic search for new activation functions and proposed an alternative to ReLUs. The article got declined from the ICLR conference because it is hard to prove that the proposed alternative is better[1]. Still, the article has been cited more than 1500 times

1. ICLR 2018 reviews of the Searching for Activation Functions article [RZL17]. Online available at `https://openreview.net/forum?id=SkBYYyZRZ` (accessed on 08.03.2022)

until today, which indicates some relevance of the proposed ideas. One of the problems is understanding what is generally good for an activation function.

While ReLUs are the state-of-the-art for non-DP models, it does not necessarily follow for DP models. Papernot et al. [PTS+21] propose using tempered sigmoids (Equation (3.4)) which are a generalized version of sigmoids that have additional three parameters that can be optimized. These are the inverse temperature T, scale s and offset o. The sigmoid function can be written as a tempered sigmoid with the parameters $s = 1$, $T = 1$ and $o = 0$ and the tanh with $s = 2$, $T = 2$ and $o = 1$. In Figure 3.2 different tempered sigmoids and the ReLU are visulized.

$$\text{tempered sigmoid}(x) = \frac{s}{1 + e^{-Tx}} - o \tag{3.4}$$

Equation (3.4): The parameters are s=scale, T= inverse temperature and o=offset.

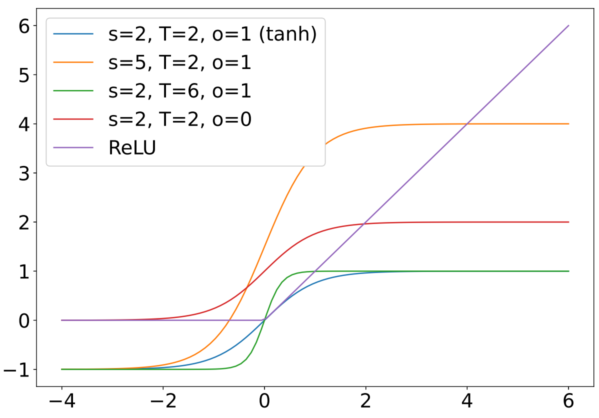


Figure 3.2: Example values for different tempered sigmoids (specified by the tuple [scale=s, inverse temperature=T, offset=o)] and the ReLU activation function in the range of [-4, 6]. The blue line is the tanh activation function. The figure has been modified from Papernot et al. [PTS+21, p.9314] to include the ReLU activation function.

Papernot et al. [PTS+21] claim that tempered sigmoids are better when training models under DP. Their contribution aims at solving the issue of choosing the clipping parameter C in DP-SGD. When the parameter C is too low, it results in a bias that changes the objective, and when it is too high, the noise added also results in inferior training performance. They argue that the inverse temperature T resolves this issue and observe that using tempered sigmoids prevents exploding gradients. Their results showed significant performance improvements on MNIST, Fashion-MNIST, and CIFAR10. One downside is that there are three more hyperparameters per tempered sigmoid unit. The authors found that tanhs are the best performing tempered sigmoids in their example and thus claim just replacing ReLUs with tanhs is sufficient [PTS+21].

It is important to tune the other hyperparameters of the model in order to correctly measure the performance difference between ReLUs and tempered sigmoids [PTS+21]. In Section 5.6 a comparison of different activation functions with extensive hyperparameter tuning will be conducted.

3.2 Batch and Group Normalization

Batch normalization is a reparametrization trick to improve the optimization of (deep) neural networks. It has been proposed by Ioffe and Szegedy [IS15] who report improved performance on ImageNet classification using batch normalization. Equation (3.5) displays batch normalization as one equation in comparison to the original paper, which uses two equations. For every dimension k of the mini-batch, the input $x^{(k)}$ is first normalized to zero mean and unit variance. In order to preserve the ability of the layer to represent functions, two additional trainable parameters $\gamma^{(k)}$ and $\beta^{(k)}$ are added. This means that the unnormalized version of $x^{(k)}$ can be represented if $\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$ and $\beta^{(k)} = \mathrm{E}[x^{(k)}]$, but also other choices are trainable. Both the original paper [IS15] as well as Section 8.7.1 of the book from Goodfellow et al. [GBC16] provide further insights into the method and why it improves learning significantly in some cases.

$$y^{(k)} = \gamma^{(k)} \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} + \beta^{(k)} \tag{3.5}$$

Even though batch normalization is a promising method in many models, it can not be used in this thesis because it is not compatible with DP. In batch normalization, a sample depends on the other samples in the batch and will be normalized to a different value if the content of the batch is different[1]. The maintainers of *opacus* suggest using the normalization method group normalization [WH18] instead, which will be introduced in the remainder of this section.

In contrast to batch normalization, group normalization normalizes the inputs by looking at each sample independently from the others and not looking at a batch. The vector with all features will have zero mean and unit variance for every sample.

The term group comes from specifying which features should be normalized together; when the number of groups is one, all features are normalized together. This case of the group norm is called layer Norm. The term for when the number of features equals the number of groups is instance norm. Figure 3.3 shows the difference between batch, layer, instance, and group norm.

1. validators/batch_norm.py in *Opacus* [YSS+21] GitHub (accessed on 02.03.2022)

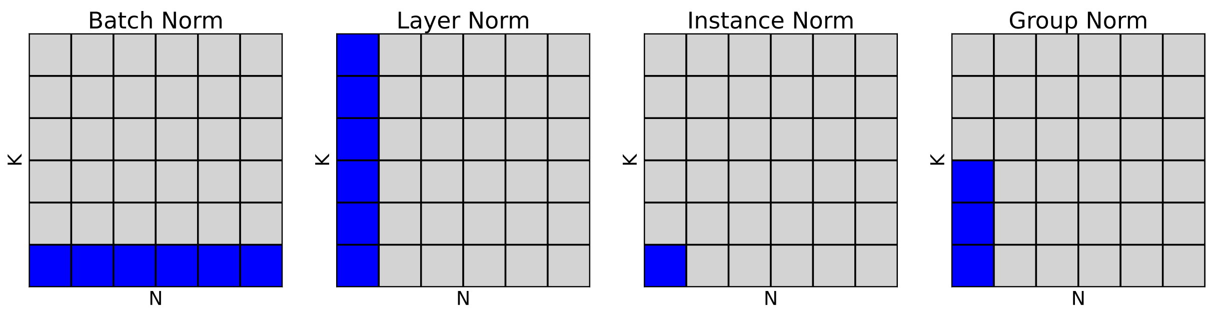


Figure 3.3: Different normalization methods. Each plot displays an input to the normalization method with a batch size of N and a dimensionality of K. The elements highlighted in blue are normalized by the same mean and variance. For the group norm, the group size is 2. The plot is simplified from Wu and He [WH18, p. 4] because the original plot is intended for computer vision applications.

3.3 Other Components

Apart from the previously discussed neural components, the neural network architecture can contain more possible components. When comparing the list of the available components in the *Pytorch* [PGM$^+$19] documentation[1] to the proposed changes, it becomes obvious that this chapter introduced sensible components for training the dataset of the thesis under DP.

Convolutional neural networks, commonly used in computer vision applications, utilize pooling, padding, and convolutional components. These and the other vision components are unsuitable for the dataset used in this thesis (see Section 4.1). Even though the thesis could use recurrent and transformer layers, it is up to future work to use them as they significantly increase the model's complexity. Recall that more complex models are more challenging to train under DP than without DP, and thus it remains to be seen if those components beneficial. Finally, the thesis does not consider dropout layers due to constraints on the time available for performing experiments.

3.4 Neural Network Architecture Search

The term network architecture denotes the overall structure of a neural network, which has to be optimized similarly to the hyperparameters of the learning process [GBC16, p. 197-198]. The automatic search of architectures for neural networks is called neural network search (NAS) [HSG89, PDDC09]. NAS is becoming more used in practice because it outperforms hand-designed architecture. Elsken et al. [EMH19] have published a comprehensive survey on NAS; this section will introduce some of its concepts. The

1. TORCH.NN in the *Pytorch* [PGM$^+$19] documentation. Online available at https://pytorch.org/docs/stable/nn.html (accessed on 10.05.2022)

topic is highly relevant to the thesis as one of the goals is to find the best model under DP. A NAS algorithm consists of three components: the search space, search strategy, and performance estimation strategy [EMH19]. The following paragraphs will describe the three components in more detail. They are also visualized in Figure 3.4.

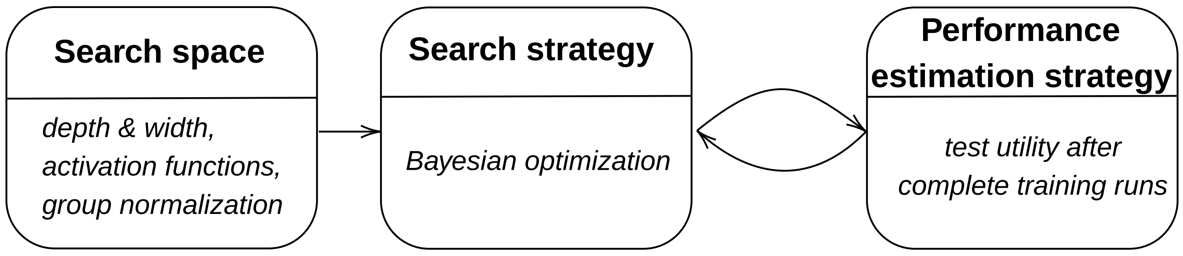

Search space
depth & width, activation functions, group normalization
Search strategy
Bayesian optimization
Performance estimation strategy
test utility after complete training runs

Figure 3.4: Abstract illustration of the neural network search conducted in this thesis. The figure is heavily inspired by the illustration from Elsken et al. [EMH19, p.2].

First, the **search space** describes the set of architectures that the NAS algorithm can select. The search space has a high dimensionality as it includes the depth of the architecture, components, and hyperparameters. All dimensions of the search space are dependent on each other. For example, including batch normalization in one of the layers requires the specification of the number of groups. There is a tradeoff regarding the size of the search space: One can limit the size of the search space by incorporating knowledge regarding possibly reasonable architectures. Reducing the size of the search space can result in inferior performance because it might exclude suitable architectures. There are techniques to reduce the complexity of the search space, e.g., by optimizing sub-architectures. This thesis will focus on a small search space that only changes the depth and width of the model besides introducing different activation functions and group normalization.

Second, the **search strategy** is the method that picks the models from the search space. An ideal search strategy finds an optimal architecture fast, but as this is unrealistic, the strategy balances finding good architectures with the total run time of the NAS. Different search strategies are possible: evolutionary algorithms, reinforcement learning, gradient-based, and Bayesian optimization. Bayesian optimization with recent techniques is promising and is also the backbone of the experiments conducted in this thesis. Details on the technique are listed in Section 4.3. One can find a comprehensive overview of the advantages and disadvantages of different search techniques in Section 3 of the survey by Elsken et al. [EMH19].

Third, a naive **performance estimation strategy** estimates the performance of a model architecture by training the model entirely and reporting the test set performance. This naive approach is computationally expensive and led to early articles on NAS consuming decades of GPU computation. Thus improvements are necessary. One can estimate the performance earlier, e.g., by training shorter or on less data or by extrapolating the learning curve. Other approaches use pre-trained weights or a large model consisting of smaller models that share their weights. All these approaches can result in worse architectures than the naive way, but the tradeoff in computation might be worth it [EMH19]. This thesis only uses the naive way and reports the test utility after complete training runs.

Some libraries perform NAS for Machine Learning, such as *Auto-Pytorch*[1] [ZLH21] for *Pytorch* or *Auto-Sklearn*[2] [FKE$^+$15] for *scikit-learn* that claim to be usable out-of-the-box. The NAS libraries use similar APIs as the underlying basic libraries on top. Still, *Auto-Pytorch*, which would be the applicable library for the experiments in this thesis, seems to be not mature enough. When writing the thesis (March 2022), the version of *Auto-Pytorch* is 0.1.1, and it would require substantial effort to make it work with the underlying libraries and models. It seems like a promising direction for future work as NAS can widen the search space while reducing the computational complexity.

1. Auto-Pytorch GitHub https://github.com/automl/Auto-PyTorch (accessed on 10.05.2022)
2. Auto-sklearn GitHub https://github.com/automl/auto-sklearn (accessed on 10.05.2022)

4. Experimental Setup

This chapter describes the required knowledge for the experiments. It starts by introducing the dataset and the non-DP baseline benchmark library. Finally, the last two sections discuss the hyperparameter optimization method and related experiments.

4.1 Dataset

All experiments in this thesis use the MIMIC-III dataset [JPS$^+$16, JPM16] in version 1.4 retrieved from PhysioNet [GAG$^+$00]. It contains patient data from Beth Israel Deaconess Medical Center in Boston, Massachusetts, the USA, of more than 40,000 admissions to critical care units. Access to the MIMIC-III dataset is credentialed and confidential but can be obtained by completing a credential process and agreeing to the terms. The open-access policy makes it easier to reproduce scientific contributions by other researchers than when using other datasets [JPS$^+$16].

According to the Health Insurance Portability and Accountability Act (HIPAA), the publishers de-identified the dataset. The dataset contains information from various sources such as demographic information, information from the bedside monitoring and laboratory results. Furthermore, it contains information that is of different structures: One table contains free form notes from nurses and doctors, which are not structured, and another one procedures that have been recorded using standardized codes. More details on the dataset and the preprocessing can be found in the article from Johnson et al. [JPS$^+$16]. In general, understanding the dataset is not trivial and thus this thesis relies on already existing benchmark libraries that extract features from the MIMIC-III databases. More details can be found in the next section.

In 2021, MIMIC-IV [JBP$^+$21] was released. It is a successor to MIMIC-III and contains more data and other improvements. While some non-DP models using MIMIC-IV like the one from Kumar et al. [KIS$^+$21] have been proposed, the availability of source code and articles is still too limited in comparison to MIMIC-III.

4.2 Benchmark Study and Models

Multiple libraries provide code for open-source benchmarks for ML models on MIMIC-III, such as the ones from Purushotham et al. [PMCL18][1], Harutyunyan et al. [HKK+19][2] and Wang et al. [WMC+20][3]. All these benchmarks do not take DP into account. This thesis uses the preprocessing, and the model comparison from Harutyunyan et al. [HKK+19]. The main reasons for choosing that benchmark suite are that the influence, measured in citations and GitHub stars, is higher for that benchmark suite than the others. Furthermore, the code base is better maintained, and other researchers of the PROBIC group work with it.

The benchmark study from Harutyunyan et al. [HKK+19] contains multiple different tasks with both preprocessing scripts and benchmark models provided. These tasks are in-hospital mortality prediction, decompensation prediction, length-of-stay prediction, and phenotype classification. The thesis will focus on the task of the phenotype classification, a multi-label classification with 25 sub-tasks that are different diseases like pneumonia or acute myocardial infarction (heart attack). One patient can have multiple diagnoses at the same time [HKK+19]. The phenotyping task is also part of other research projects in the PROBIC group.

Harutyunyan et al. [HKK+19] suggest using the receiver operating characteristics (ROC) metric for the phenotyping task. The ROC space is a two-dimensional space in which classifiers are placed according to their false positive rate (FPR) and true positive rate (TPR). In Equation (4.1) one can find the estimation of the FPR and in Equation (4.2) the estimation of the TPR [Faw06].

$$\text{FPR} \approx \frac{\text{negatives incorrectly classified}}{\text{total negatives}} \tag{4.1}$$

$$\text{TPR} \approx \frac{\text{positives correctly classified}}{\text{total positives}} \tag{4.2}$$

For classifiers that output a probability, one can vary the threshold at which a probability is resulting in the first or second binary class and produce a curve with that [Faw06]. Figure 4.1 shows different AUC curves (red, green and orange) in the ROC space. The dotted blue line resembles the curve for a random classifier.

As one can easily see from Figure 4.1, it is challenging to compare different classifiers using the curves. While it is trivial to see that the orange curve is worse than the green and red curve, it is hard to distinguish between red and green. Thus, the area

1. Purushotham et al. code can be found at https://github.com/USC-Melady/Benchmarking_DL_MIMICIII
2. Harutyunyan et al. code can be found at https://github.com/YerevaNN/mimic3-benchmarks
3. Wang et al. code can be found at https://github.com/MLforHealth/MIMIC_Extract

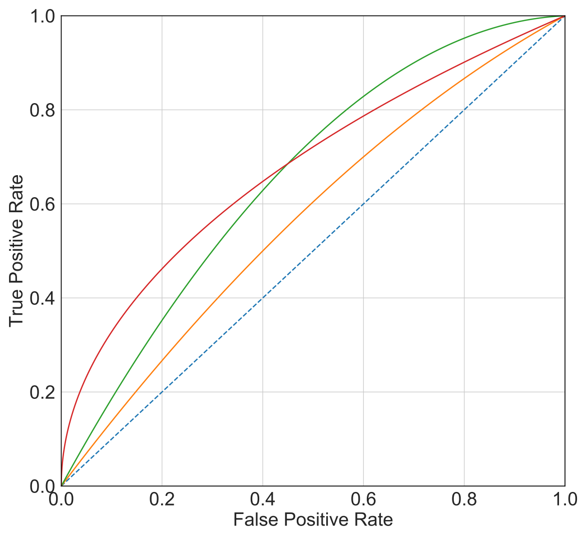


Figure 4.1: Illustration of different ROC curves. This figure has been created by the author.

under the curve (AUC) for the ROC is often determined. The best possible classifier has an AUC-ROC of 1, whereas a random classifier (blue dotted line) has an AUC-ROC of 0.5 [Faw06]. The comprehensive article *An introduction to ROC analysis* by Fawcett [Faw06] contains more technical details on AUC-ROC. The benchmarking library from Harutyunyan et al. [HKK$^+$19] outputs an AUC-ROC for each of the 25 phenotyping tasks. The primary metric of the complete task is a macro averaged AUC-ROC [HKK$^+$19]. The macro averaged AUC-ROC is an unweighted average of the AUC-ROC scores for the 25 tasks and thus gives every task the same importance. Other methods such as the micro averaged AUC-ROC do not give all tasks the same importance[1].

The thesis uses the preprocessing from Harutyunyan et al. [HKK$^+$19] and does not modify it. The preprocessing outputs 714 features based on the studies by Pollack et al. [PPR96] and Lipton et al. [LKEW17]. Pollack et al. [PPR96] propose 17 hand-engineered features, which include e.g. coma, blood pressure and abnormal pupillary reflexes. For each of them, six statistics such as minimum, maximum and mean are computed. This is done for seven intervals of each time series, resulting in the $17 * 6 * 7 = 714$ features [HKK$^+$19].

While some research suggests that machine learning under DP requires better features [TB20], changing the preprocessing is not in the scope of this thesis. Feature engineering demands a deep understanding of the dataset and is likely to have similar complexity as this thesis.

The used preprocessing contains a hard-coded test/train split, and it is not immediately obvious what the criterium for the split is. To ensure that the split is not favorable

1. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html (accessed on 09.06.2022)

for specific hyperparameter combinations, one experiment aims at determining the standard derivation between the macro averaged AUC-ROC scores for different splits. For the best non-DP MLP hyperparameter combination, the standard deviation is 0.0010. The low standard deviation suggests that the utility is not prone to different test/train splits, and thus the experiments will use the default split.

Figure 4.2 shows the results for the phenotyping for multiple different models from Harutyunyan et al. [HKK$^+$19] which display a similar diagram in their paper. This figure has been recreated to integrate a non-DP version of the MLP used in this thesis's experiments. This MLP itself is not part of the benchmark suite but has been created from scratch and optimized with *Optuna* [ASY$^+$19]. The confidence intervals are computed by bootstrapping the test set 10000 times.

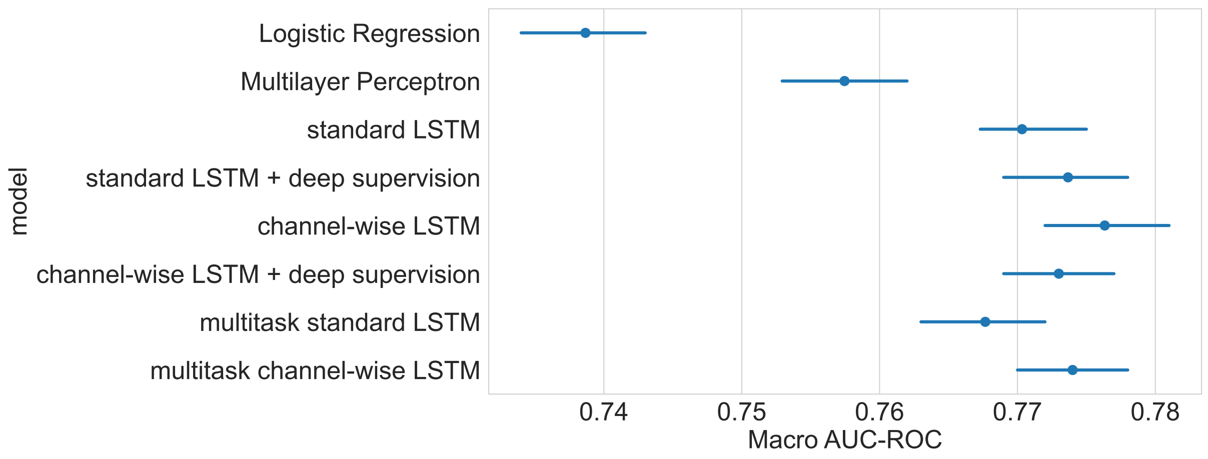


Figure 4.2: Comparison of the performance of the different models of the benchmark study. The figure is an own figure based on the Figure 2 of Harutyunyan et al. [HKK$^+$19, p.6]. Please note that the MLP has been added by the author and is not part of the publication of Harutyunyan et al. [HKK$^+$19].

The figure shows that in the non-DP setting, more complex models like LSTMs perform better than simpler models such as logistic regression or the newly proposed MLP. Recall that optimizing large models under DP is challenging due to the curse of dimensionality (see Section 3). Furthermore, the training time of LSTMs is significantly larger than for much smaller MLPs. Thus this thesis focuses on MLPs. Possible following work could extend the work towards more complex models.

4.3 Bayesian Optimization

Finding the (nearly) optimal hyperparameters is essential for comparing different methods and determining the influence of hyperparameters. As previously noted in Section 3.1 when discussing different activation functions, it is essential to tune the other factors to allow for a fair comparison. There are several methods for optimizing hyperparameters, and this section will give an overview of them.

The common methods listed by Bergstra and Bengio [BB12] are manual search, grid search, random search, and Bayesian optimization. The latter three methods are visualized in Figure 4.3. Manual search relies on the research to optimize the hyperparameters by making informed choices and is thus laborious and hard to reproduce, but requires not much overhead and allows the researcher to gain insight into the effect of different hyperparameters [BB12]. For selecting the hyperparameters in this thesis, a manual search is infeasible.

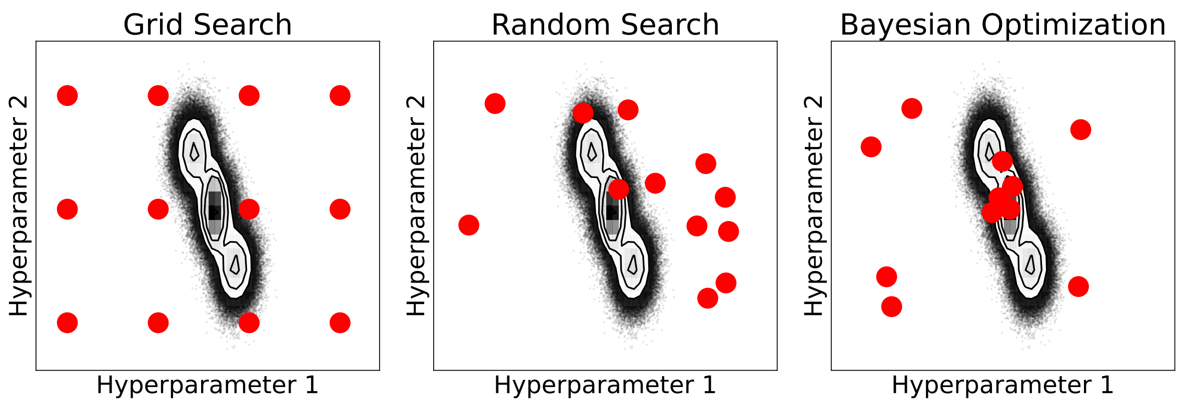


Figure 4.3: Illustration of the differences between Grid Search, Random Search and Bayesian Optimization using a toy example. The figure has been created by the author using the 2D-histograms provided by the Python visualization library *corner.py* [FM16].

Grid search works by spanning a grid over the hyperparameter space and evaluating every combination of the hyperparameters. The method is easy to set up but has multiple drawbacks: The number of trails grows exponentially, e.g., when assuming that there are five hyperparameters with ten possible values each, the number of trails would be already 10^5. A significant amount of those trails do not provide any more insights as they explore uninteresting areas of the hyperparameter space [BB12]. The inefficiency can be, e.g., seen when looking at the left column of points evaluated in Figure 4.3. The value of *hyperparameter 1* is suboptimal and will result in a bad result no matter the value of *hyperparameter 2*. Still, three trails with different values of *hyperparameter 2* will be executed. Both grid and manual search are widely used in research, but as there are more efficient methods, this thesis will not use them [BB12].

Random search works as the name suggests. Random points in the hyperparameter space are evaluated, and a good combination is discovered by chance. Despite its simplicity, random search outperforms grid search and manual search while additionally allowing for easy parallelization and fault tolerance since the trials are i.i.d. [BB12].

Bayesian optimization is a promising approach that uses Bayesian surrogate models to make informed choices about the regions to explore. It naturally trades off between exploring areas of the space that seem most promising and areas that have not been explored yet. Bayesian optimization has shown better performance in some experiments [SLA12]. The overhead of the Bayesian optimization is larger than in the previous methods.

In this thesis, Bayesian optimization with the Python library *Optuna* [ASY+19] is used. *Optuna* uses the tree-structured Parzen Estimator (TPE) [BBBK11] by default which outperforms other techniques [ASY+19]. While there are more complex algorithms for neural architecture search (NAS) like the library *Auto-Pytorch* [ZLH21], *Optuna* is used as it can be easily adapted to perform NAS shown in the documentation[1], this reduces the complexity of the implementation.

The Bayesian optimization library *Optuna* not only provides methods to optimize hyperparameters but additionally contains tools to visualize the optimization runs. Figure 4.3 shows a slice plot obtained during an optimization run prior to the experiments displayed in Section 5. The x-axis shows the hyperparameter values, and the y-axis shows the achieved utility. The points are the results of different trials. Their brightness visualizes the ordering of the trials. The darker the point is, the later *Optuna* executed the trial.

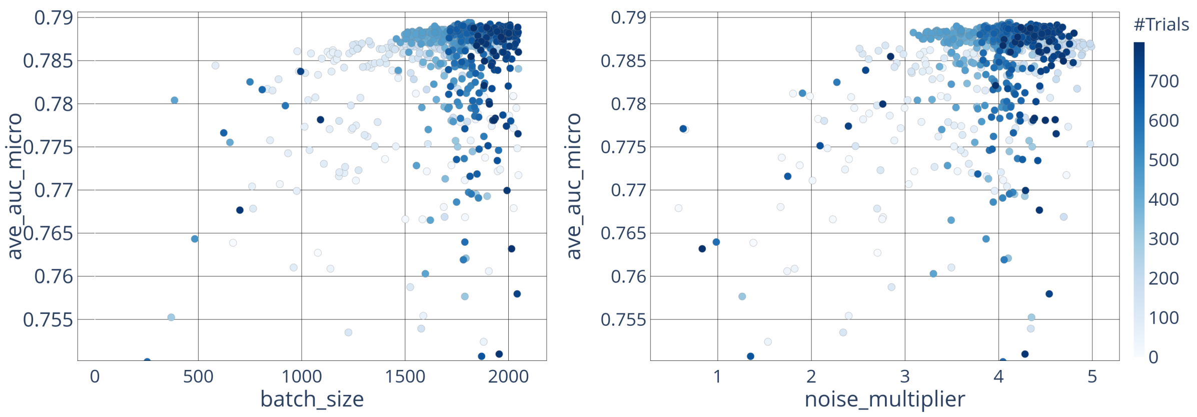


Figure 4.4: *Slice plot* of the hyperparameters batch size and noise multiplier obtained during a Bayesian optimization run with the library *Optuna* [ASY+19]. The basic visualization was created using *Optuna* [ASY+19] but enhanced by the author.

Figure 4.4 serves two purposes: First, it shows that Bayesian Optimization indeed efficiently focuses on the optimal regions of the hyperparameter space. For both hyperparameters, the Bayesian optimization explores all regions of the hyperparameter space. Still, most of the evaluations are in the region with a batch size of around 2000 and a noise multiplier of about 4.2. Second, the user can assess if the search space of the hyperparameters is sufficient. For example, when *Optuna* primarily evaluates values close

1. optuna-examples/pytorch/pytorch_simple.py in *Optuna* [ASY+19] GitHub (accessed on 02.03.2022)

to the bounds of the hyperparameter space, the user might want to extend the bounds to obtain more optimal results.

4.4 Related Experiments

This section will provide an overview of related hyperparameter optimization experiments in other works. Almost every paper in DP research contains experiments, which require choosing hyperparameters. This section gives valuable insight into the techniques used in research to optimize hyperparameters. Future work should aim at providing a survey extending the work here.

The selection of hyperparameters leaks privacy, and this additional cost is ignored in some experiments [TB20, PTS+21] while approaches exist to account for this privacy leakage [LT19, MSH+21, PS21]. This work also ignores the privacy leakage occurring through the selection of hyperparameters. Ignoring the leakage makes sense because this work's experiments aim to provide insights into the effect of hyperparameter choices.

Out of the considered experiments, most experiments perform grid search with different grid densities and sizes. Among those experiments are the experiments from Tramer and Boneh [TB20], Mohapatra et al. [MSH+21] and Jagielski et al. [JUO20]. Tramer and Boneh [TB20] report that some of the hyperparameters show ideal performance at the boundaries of the grid. Despite that extending the grid is easily possible in a grid search, the authors argue that extending the grid is not necessary because the choice of the hyperparameter does not matter much. One distinctive approach by Avent et al. [AGD+20] uses a Bayesian optimization technique to efficiently find the best tradeoffs between utility and privacy. The authors show, similar to the claims made in Section 4.3 that their Bayesian optimization approach outperforms both random and grid search.

The related works all use widely popular ML datasets. The MNIST dataset is used by many experiments [AGD+20, TB20, JUO20, PTS+21, MSH+21]. Three experiments [TB20, JUO20, PTS+21] also report results on the alternative Fashion-MNIST, two [TB20, PTS+21] look at CIFAR-10 and the same number of experiments [AGD+20, MSH+21] work with the Adult dataset.

Out of the hyperparameters of DP-SGD, the clipping norm is often not optimized, or the articles provide no insights on the performance for different clipping bounds. Jayaraman and Evans [JE19] and Tramer and Boneh [TB20] fix the clipping bound to 1/0.1 and claim that the choice does not matter much. Jagielski et al. [JUO20] try out clipping bounds of $0.5, 1$ and 2. Only Avent et al. [AGD+20] tune the clipping bound extensively with values in the interval $[0.1, 4]$ and for a few experiments $[0.1, 12]$.

Mohapatra et al. [MSH+21] show a relationship between the optimal learning rate and the clipping bound when training with a non-adaptive optimizer. This relationship is not there when training with an adaptive optimizer such as a DP version of Adam [KB14]. For hyperparameter tuning Mohapatra et al. [MSH$^+$21] recommend using an adaptive optimizer. This work will use Adam [KB14].

The first set of experiments, with whom Papernot et al. [PTS$^+$21] report the increased performance of their newly proposed activation function, use the same set of hyperparameters for both activation functions. They argue that using the same set means that the comparison is fair and introduce a second set with optimized hyperparameters (that are not the same) to support the claim. It is up to the reader to follow the reasoning and form their own opinion. This work will optimize all hyperparameters to ensure that the reported performance gain is independent of a seemingly arbitrary hyperparameter choice.

Tramer and Boneh [TB20] provide insight into the robustness of their models towards different value choices of hyperparameters. Setting batch size, learning rate, and the number of epochs to a fixed value have a low impact as long as the other hyperparameters are adapted appropriately.

5. Experiments

This chapter presents the experiments of the Master's thesis. The first half of the experiments focus on hyperparameter optimization and gives insight into three of the main hyperparameters of DP-Adam, which is used in replacement to DP-SGD as discussed in Section 4.4. These are the noise multiplier (Section 5.1), the clipping bound (Section 5.2), and the batch size (Section 5.3). Section 5.4 will assess the tradeoff between utility and training runtime. The last three experiments evaluate changes in the architecture to obtain better utility. Section 5.5 elaborates on changing the width and depth of the MLP, the Section 5.6 on different activation functions, and finally, Section 5.7 on group normalization.

The experiments are run on cluster nodes equipped with an NVIDIA V100 GPU with 32 GB VRAM, two CPU cores, and 32 GB RAM. While the runtime depends on the hyperparameters and the model architecture, the hardware is sufficient to perform hyperparameter optimization. There is a tradeoff between obtaining the best configuration and the total runtime of the experiments. The source code for running the experiments with the MIMIC-III dataset is available on GitHub[1].

5.1 Exp. 1: Noise Multiplier

The first experiment looks at the DP-Adam hyperparameter that controls the amount of noise added to the gradients. The goal is to look at the optimal noise multiplier for different levels of privacy guarantee ϵ. Due to the experimental setup, explained in the following paragraph, it is possible to look at the noise multiplier independent of other hyperparameters.

Preliminary experiments suggest that running experiments for the ϵ values $[1, 2, 4, 9]$ is sufficient. Thus, the experiment consists of independent optimization studies with 400 optimization trails with three seeds for each combination of the noise multiplier and the four different ϵ values. The Bayesian optimization library *Optuna* [ASY+19] is optimizing all other hyperparameters, and the architecture of the MLP does not differ between the

1. https://github.com/PROBIC/hyperparameters-neural-architectures-in-dp-dl

different trials of the experiment. See Algorithm 2 for a more details on the optimization run. The algorithm has been executed for each combination of the four ϵ values and noise multiplier values.

Algorithm 2 Optimization run for fixed hyperparameter and target ϵ combination

```
Input: possible values of hyperparameters $\mathcal{H}$, evaluation function $\mathcal{E}$, seeds $\mathcal{S}$,
       number of trials $\mathcal{T}$, target epsilon $\epsilon$
 1: Initialize best hyperparameter configuration $\mathcal{C} \leftarrow \emptyset$
 2: Initialize best macro averaged AUC-ROC $\mathcal{A} \leftarrow 0.5$
 3: for $t \in [T]$ do
 4:     Get hyperparameter configuration candidate $O$ from optuna
 5:     $\mathcal{O} \leftarrow \text{optuna}(H)$
 6:     Evaluate the configuration for each seed and return average score
 7:     if $\mathcal{E}(\mathcal{O}, \mathcal{S}, \epsilon) > \mathcal{A}$ then
 8:         Save score and configuration if better than previous ones
 9:         $\mathcal{C} \leftarrow O$
10:         $\mathcal{A} \leftarrow \mathcal{E}(\mathcal{O}, \mathcal{S}, \epsilon)$
11:     end if
12: end for
Output: best score $\mathcal{A}$, best hyperparameter configuration $C$
```

Figure 5.1 displays the result of this experiment. On the x-axis, the various noise multiplier values are displayed, and on the y-axis, the macro averaged AUC-ROC (utility) of the best result. The different lines display the results for different levels of ϵ.

One can make two observations when looking at Figure 5.1: First, a low amount of noise (noise multiplier = 1) results in worse utility than a medium or high amount of noise (noise multiplier > 1.5). Second, the utility does not improve significantly after a certain threshold. It is worth looking at the values of the other hyperparameters for the optimal trails displayed in the above figure to understand why this might be the case.

Figure 5.2 displays the number of epochs and the batch size for the trails with the best utility for the respective noise multiplier values. Roughly the following correlation holds for both batch size and iterations: the higher the noise multiplier, the larger the number of epochs and batch size.

When recalling the theory from Section 2.2, it is clear why the batch size and iterations can be larger with higher noise given a fixed privacy budget. A higher noise multiplier decreases the privacy spent per step of the optimizer. Thus, training with the same privacy budget can accommodate a higher batch size (subsampling ratio) and more epochs. In terms of the utility/privacy tradeoff, a moderate noise multiplier gives better results than a small noise multiplier. Training for more epochs or a larger batch size

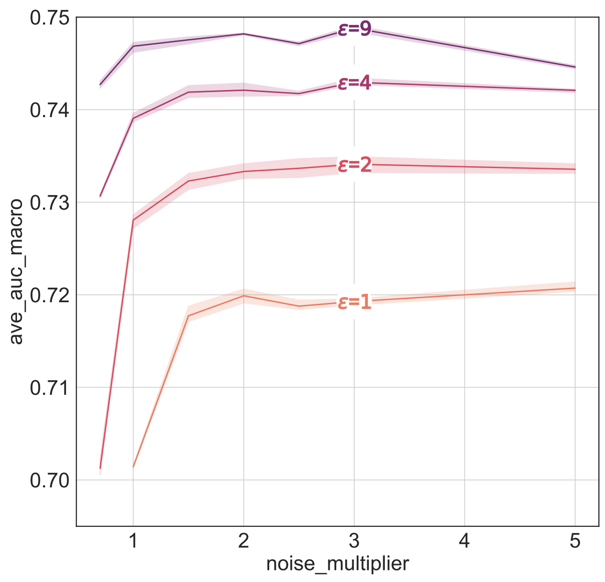


Figure 5.1: Comparison of the best utility for different noise multiplier values and the privacy budgets.

compensates for the added noise.

The most interesting tradeoff in DP is not between the different hyperparameters and the utility but between the utility and the runtime of the training. Figure 5.3 shows this tradeoff. The flaw of Figure 5.3 is that the results are from the noise multiplier experiment, which has not been designed to assess this tradeoff. The Bayesian optimizer does not freely modify the noise multiplier, but the number of possible values of the noise multiplier is finite. In Section 5.4 an experiment will be conducted without this restriction.

After a certain threshold, additional runtime does not result in improved utility. The utility plummets after a certain runtime for the privacy levels of $\epsilon = 1$ and 2. While it is hard to analyze the reasons due to the high dimensional parameter space, possible causes include the fact that a high runtime is hard to achieve with these low epsilon values. It is possible that the experiment is not sufficient for looking at this tradeoff. More details will be discussed in Section 5.4 when a dedicated experiment for looking at this aspect is conducted.

Figure 5.4 displays the optimal clipping bound for the optimal trails obtained in the noise multiplier experiment. Interestingly, the optimal clipping bound and the epsilon value correlate. Even though the experiment has not been designed to look at this relation, one can still observe that the optimal clipping bound for $\epsilon = 1$ lies around 0.7. In contrast, the optimal clipping bound for $\epsilon = 9$ is between 0.9 and 1.0. The following Section 5.2 will look at the clipping bound in more detail.

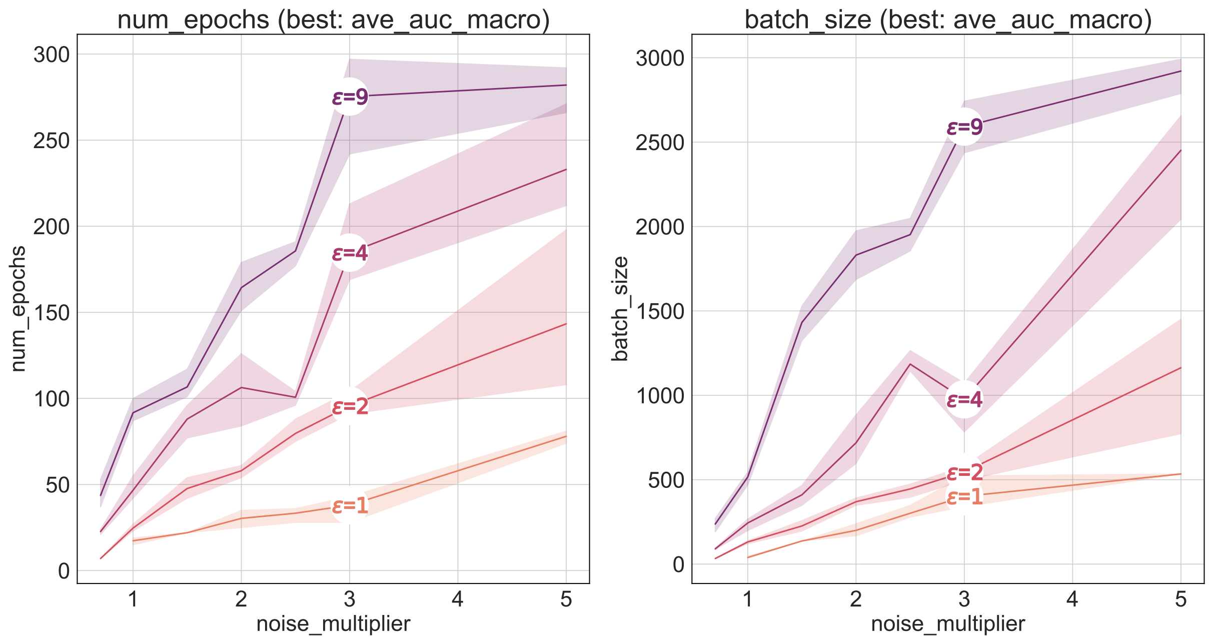


Figure 5.2: Comparison of the number of iterations and batch size of the hyperparameter combination with the best value for different noise multiplier values for the privacy budgets $\epsilon = [1, 2, 4, 9]$.

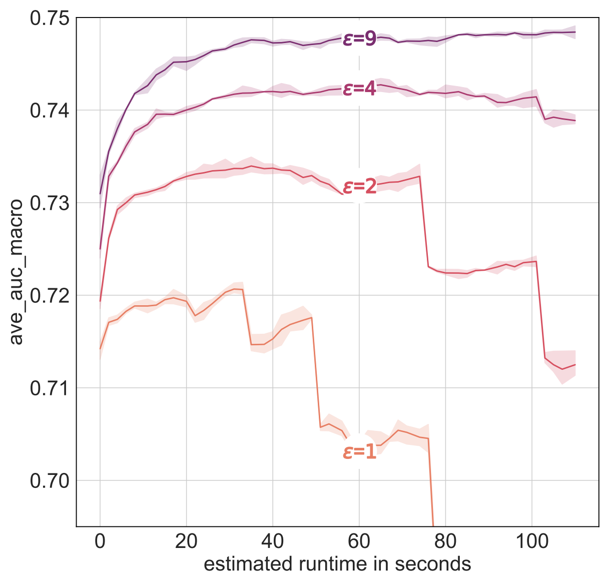


Figure 5.3: Runtime comparison for different privacy budgets based on the noise multiplier experiment.

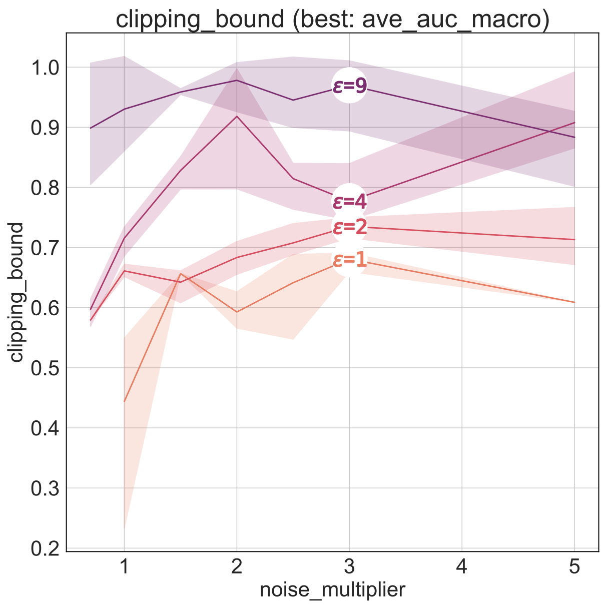


Figure 5.4: Clipping bound of the hyperparameter combination with the best utility for different noise multiplier values and the privacy budgets.

5.2 Exp. 2: Clipping Bound

The second experiment evaluates the effect of the clipping bound on the utility of the models. Recall from Section 2.2 that the clipping bound should be optimized similar to the other hyperparameters of DP-Adam.

Before looking at the optimal clipping bounds, it makes sense to look at the cumulative distribution function of the pre-clipped L2-norms of the gradients. This will put the later obtained optimal clipping bounds into perspective. Figure 5.5 shows the cumulative distribution function of the pre-clipped L2-norms of the gradients for the clipping bound values 0.001, 1 and 10. As one can see, the vast majority of the L2-norms is smaller than two, and for the clipping bound of 0.001, the majority is even smaller than one.

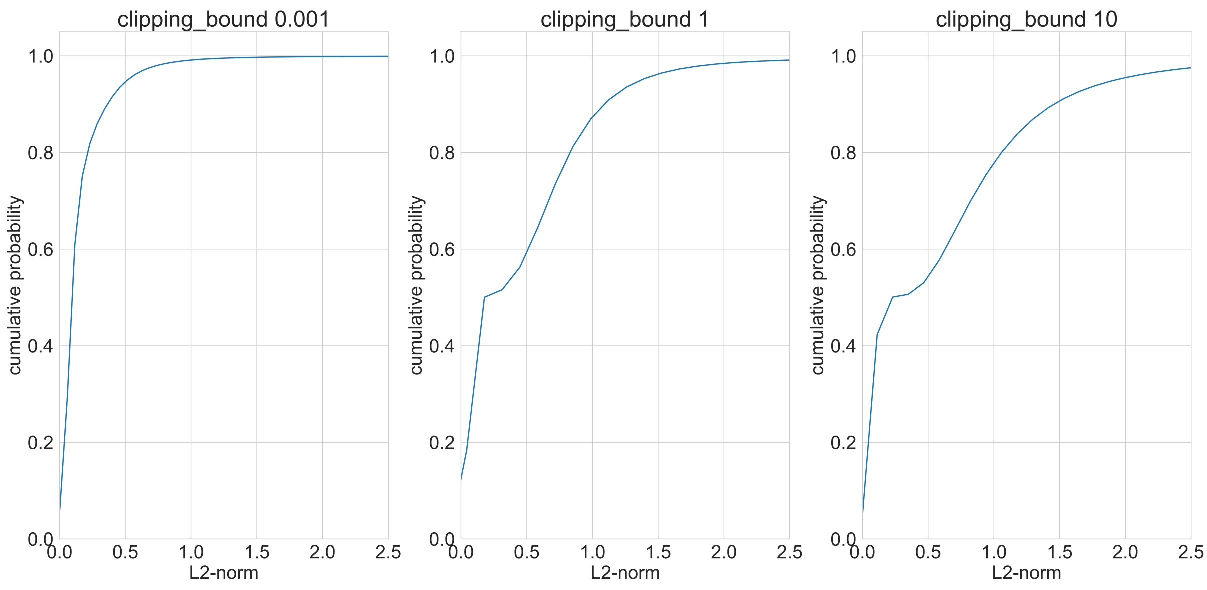


Figure 5.5: Cumulative distribution function of pre-clipped L2-norms of gradients.

The remainder of this section will describe the clipping bound experiment. The setup of the second experiment is identical to the one of the first experiment: The Bayesian optimizer tunes all hyperparameters but the clipping bound and runs the same number of trails (400) with three seeds each per an independent study that represents one combination of ϵ and clipping bound value. Figure 5.6 shows the result of this experiment. Note that the right subplot displays a detail from the left with different scales.

The left subplot of Figure 5.6 visualizes that neither a tiny nor an enormous clipping bound are optimal. As discussed in Section 2.2, a tiny clipping bound will result in gradients with nearly no useful information, and an enormous one results in a high amount of noise. The optimal clipping bound (marked by a dot) for all ϵ is around one, but to distinguish the optimal clipping bound for the different ϵ, looking at the right subplot is necessary.

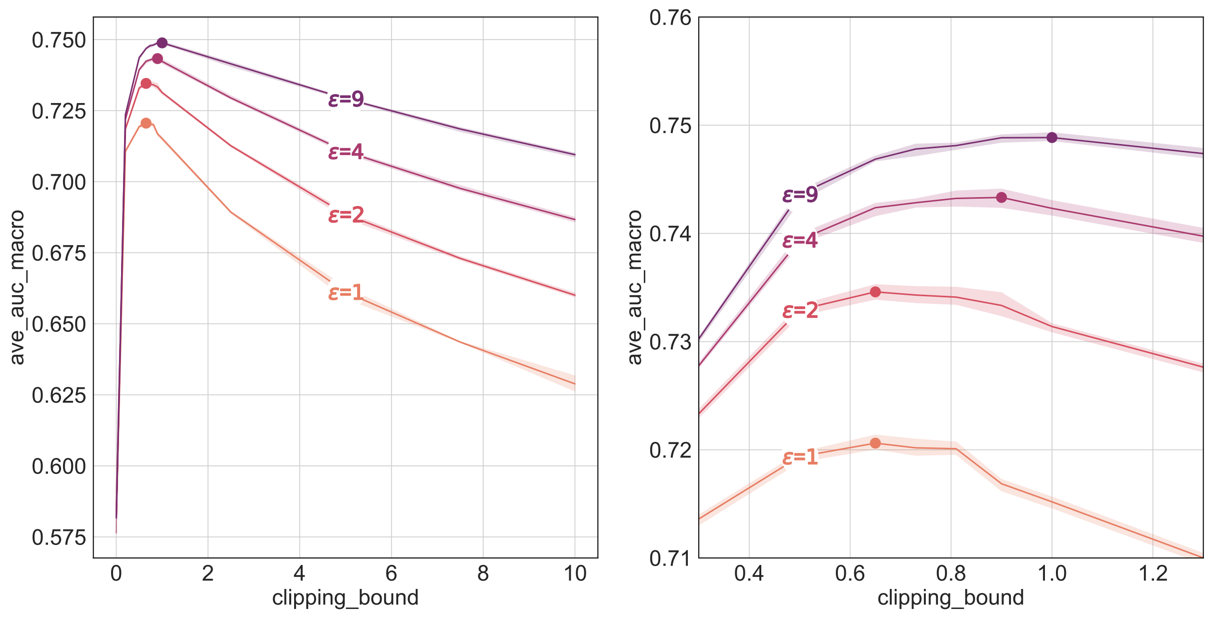


Figure 5.6: Comparison of the best utility for different clipping bounds and privacy budgets. The right subplot shows a detail from the left subplot.

As can be seen from the right subplot of Figure 5.6 the optimal clipping bound is correlated with ϵ, which could already be observed in Figure 5.4 of the previous section. The higher the ϵ is, the higher the optimal clipping bound. As described in Section 2.2, the noise drawn from the Gaussian distribution depends on both the noise multiplier and the clipping bound. It seems that with higher ϵ more noise originating from the clipping bound can be tolerated.

Finally, one can now combine the information from Figures 5.5 and 5.6. The optimal clipping bounds are equivalent to clipping at the L2-norm quantiles of roughly 0.6 for $\epsilon = 1$ and 2 and 0.9 for $\epsilon = 9$. Likely, adaptive clipping, as described in Section 2.2, provides additional interesting insights, but unfortunately, the implementation is only available in *Tensorflow Privacy* and suffers from runtime problems. Fixing those runtime problems, e.g., by implementing adaptive clipping in *opacus* or streamlining it lies outside the scope of this thesis. Future work can extend the work of the thesis here.

5.3 Exp. 3: Batch Size

The third and final experiment regarding the hyperparameters of DP-Adam looks at the effect of the batch size. In the library *opacus* the batch size is the average batch size and thus influences the probability of choosing a sample within one step of DP-Adam. Please see Section 2.2 for more details on the batch size.

This experiment optimizes the other hyperparameters in the same way as the other experiments. Please note that due to the higher memory consumption compared to SGD resulting from keeping track of single sample gradients, the maximum batch size is 3000. The university has NVIDIA A100 GPUs available that provide up to 80 GB of VRAM and would allow for larger batch sizes, but this requires adapting the packages to the respective drivers. Future work could extend this work here. Figure 5.7 shows the results of the experiment.

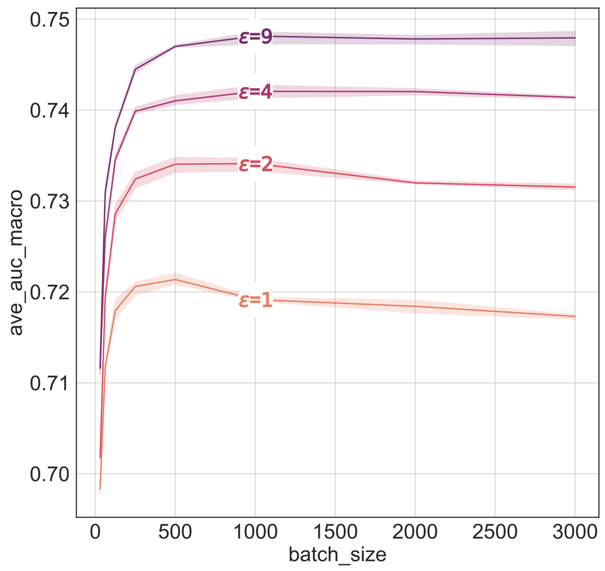


Figure 5.7: Comparison of the best utility for different batch sizes and privacy budgets.

Similar to the findings in Section 5.1 that looked at the noise multiplier, the batch size does not have a significant effect on the utility of the models. After surpassing the batch size of 128, the utility does not differ much. One can observe a slightly improved utility in the area of a batch size of 500 for $\epsilon = 1$ and at a bit larger batch size for the larger epsilons. Especially when looking at the scale of these differences, this area of higher utility does not appear to be as significant. It would be interesting to see how the utility behaves when increasing the batch size to the point where the optimizer uses the whole training dataset in every step. However, as discussed previously, this requires using GPUs with higher memory capacity.

The more thorough analysis of the batch size experiment did not result in insights that provide value to the reader. However, one outcome of this experiment is that it is worth optimizing the batch size as it makes a difference, even though the difference did not seem to be as significant.

5.4 Exp. 4: Runtime/Utility Tradeoff

The main observation regarding the tradeoff between utility and runtime under DP from the noise multiplier experiment in Section 5.1 and Figure 5.3 was that after a certain threshold, more runtime does not yield more utility. Recall that the main limitation of that observation was that the experiment had not been designed to look at this tradeoff. It was a by-product of the noise multiplier experiment where the Bayesian optimization algorithm does not freely modify the noise multiplier. The experiment in this section overcomes this limitation by running independent Bayesian optimization studies for each combination of $\epsilon = [1, 2, 4, 9]$ and the runtime of the training $[10, 20, 30, ..., 100]$. In Figure 5.8 one can find the results of this experiment.

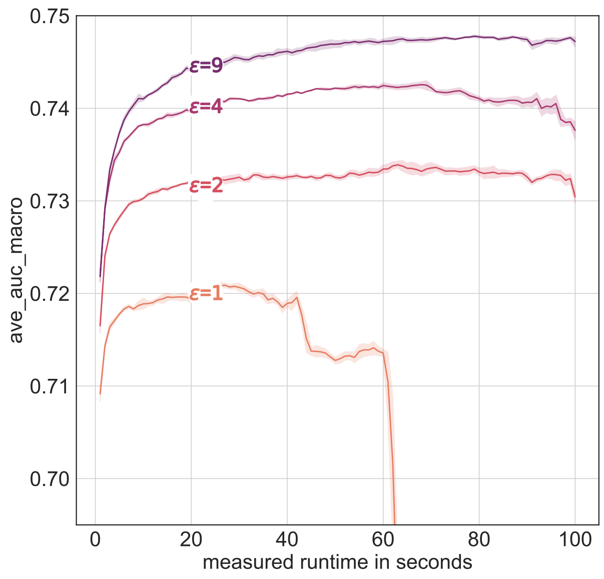


Figure 5.8: Comparison of the best utility for different training run times and privacy budgets.

Figures 5.3 (based on noise multiplier experiment) and 5.8 (dedicated runtime experiment) do not differ much. The general observation that more runtime does not improve the utility holds for both figures. The only difference between the two figures is that in comparison to the noise multiplier experiment, the utility for $\epsilon = 2$ does not plummet with more than 70 seconds runtime. Additionally, some results differ slightly, e.g., the utility of $\epsilon = 1$ decreases earlier, but this can also be attributed to a varying time measurement method and using a slightly different configuration for running the experiment[1].

1. The technical specifications of the cluster nodes are the same, but the machines had to be taken out of a different pool of cluster nodes due to scheduling issues on the previously used pool.

5.5 Exp. 5: Depth and Width

The previous experiments provided insights into the hyperparameters of DP-Adam and the tradeoff between the utility and the model's runtime. As discussed in Section 3, it is not sufficient to tune the hyperparameters, but changes to the architecture can also provide improvements in the utility. The experiments of this section focus on the most general changes to the architecture. The first experiment will provide insight into the effect of the depth of the model (the number of layers), and the second experiment changes the model's width with the optimal depth obtained in experiment one.

The Bayesian optimizer is optimizing the hyperparameters in these experiments. Keeping the hyperparameters unfixed results in insights about the architecture unbiased by any hyperparameter choices. In the first experiment, independent optimization studies are run for each combination of the number of layers $[1, 2, 3, 4, 6, 8, 10]$ and the $\epsilon = [1, 2, 4, 9]$. Per optimization run, 500 trails are run, and the width of the respective layers is also being optimized. In Figure 5.9 one can see the results of this experiment.

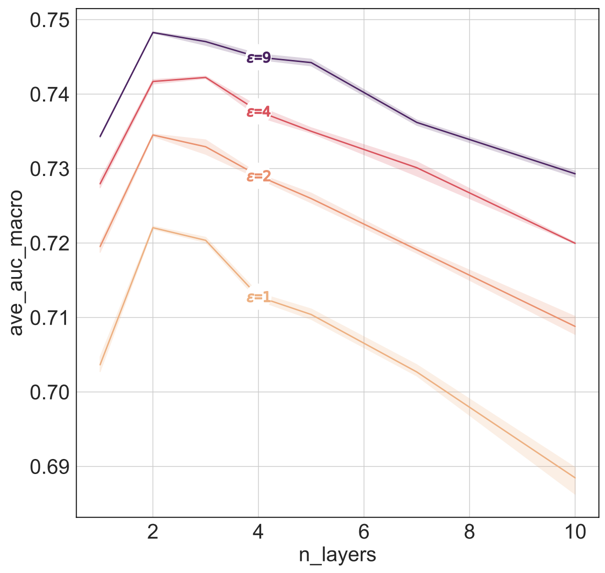


Figure 5.9: The best utility for different number of layers, which are counted as the number of hidden layers plus the output layer, and privacy budgets..

As one can see from Figure 5.9, two layers seem to be optimal, but models with three layers yield almost the same utility. In fact, for $\epsilon = 4$, the utility is slightly better with three layers than with two layers. A model with one layer, which means that it has no hidden layer, performs worse. Increasing the number of layers above three decreases the utility. The result of this experiment is that the number of layers matters and that the optimal number of layers is two. Notably, the utility does not seem to be as sensitive to the number of layers as to the hyperparameters.

The previous experiment showed that using two layers is optimal. It is interesting to analyze what width the model should have. The following experiment is almost equivalent to the setup of the previous experiment but fixes the model's depth to two layers. The independent runs then result from the combination of $\epsilon = [1, 2, 4, 9]$ and different widths of the hidden layer. The Figure 5.10 shows the result of this experiment.

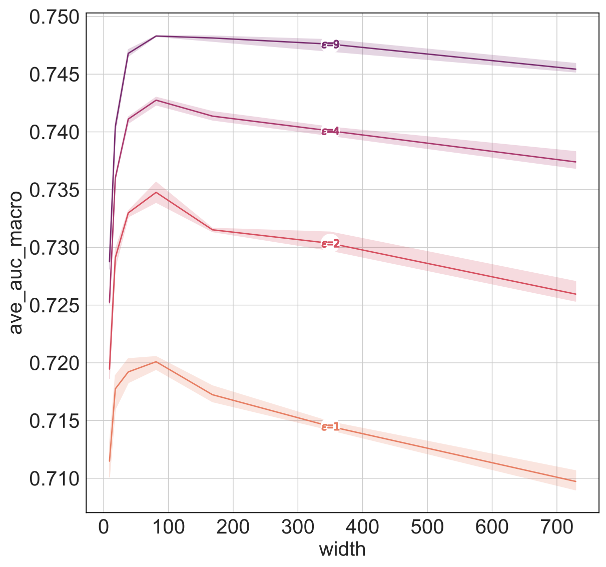


Figure 5.10: Comparison of the best utility for different network widths and privacy budgets.

A model with a hidden layer with a small width performs worse than a model with a hidden layer of 81 units. Increasing the width above this threshold decreases the utility again. A model with a hidden layer of approximately 700 units has a similar utility as a model with a small width.

Interestingly, a high width has a lower negative effect on the utility when the ϵ is higher. This can be seen in Figure 5.10 where the lines for $\epsilon = 1$ and 2 are plummeting for higher width, whereas the lines for $\epsilon = 4$ and 9 decrease more slowly. One could suspect that the higher ϵ allows for more beneficial hyperparameters, which could positively affect the utility. Still, it is hard to analyze this in more detail. Because the optimal width is the same across all ϵ, analyzing this effect in more detail can be left for future work.

Even though the reader might assume otherwise due to the ordering of the experiments, the hyperparameter experiments in the Sections 5.1 to 5.4 include the findings that have been made in this experiment because they have been rerun with the optimal width (81) and depth (two).

5.6 Exp. 6: Activation Functions

As described in Section 3.1 so-called tempered sigmoids have been proposed as an easy way to improve the utility of a model when optimized under DP. Recall that the authors of the tempered sigmoid article [PTS+21] claimed that replacing ReLU activation functions with tempered sigmoids significantly improves the utility. Furthermore, the authors observed that the tanh activation function, which is one specific tempered sigmoid, also yields good performance. As the activation function used in this thesis is a ReLU, it is straightforward to try to reproduce the results. As there is no tempered sigmoid implementation available in *Pytorch*, the author had to implement and unit test a tempered sigmoid himself.

The following experiment optimized the model's hyperparameters using Bayesian optimization for the different $\epsilon = [1, 2, 4, 9]$. Independent optimization has been performed for each of the activation functions. The tempered sigmoid activation function has three additional hyperparameters: the offset, inverse temperature, and scale. One can find the results of this experiment in Figure 5.11.

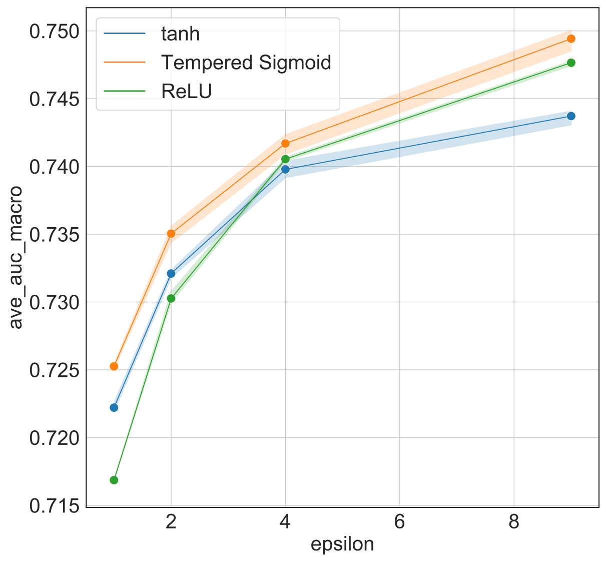
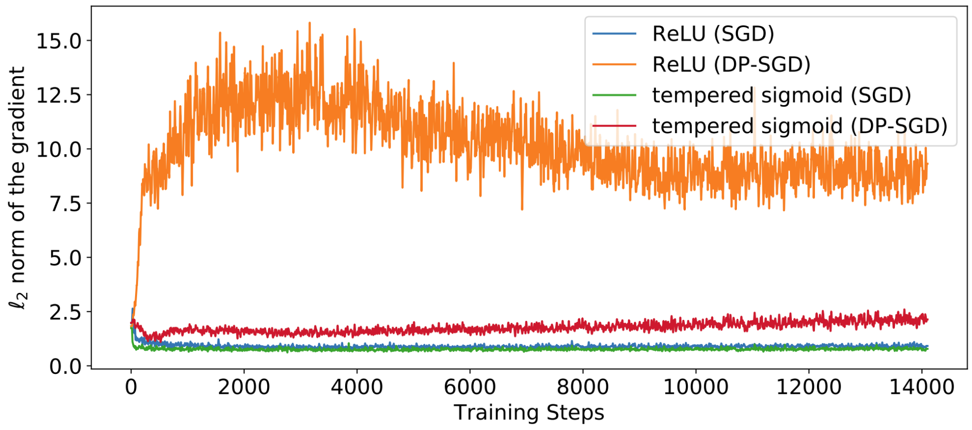


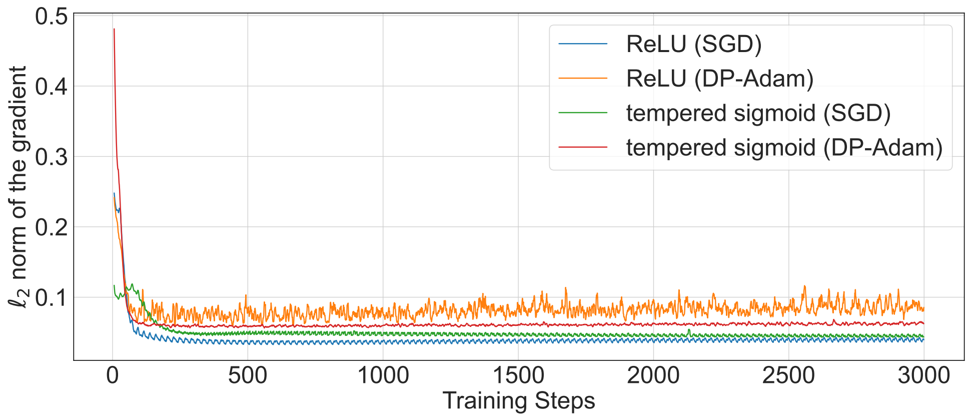
Figure 5.11: Comparison of best utility when using the activation functions tempered sigmoids, ReLU and tanh for different privacy budgets.

In contrast to the findings reported by Papernot et al. [PTS+21] in the tempered sigmoid article, using the tempered sigmoids does not result in significant performance improvements in comparison to the ReLU. While this might be due to the already good performance of the DP model, one has to note that three more hyperparameters have to be tuned to obtain a slightly better utility. It is not sufficient to use the tanh activation function as it has only a slightly better performance when looking at $\epsilon = 1, 2$ but at the same time a worse performance for $\epsilon = 9$. Thus, contrary to the findings reported in the tempered sigmoid article [PTS+21], the best hyperparameter combination of the tempered sigmoid is not a tanh.

Figure 5.12a extracted from the article by Papernot et al. [PTS+21] suggests that using tempered sigmoids prevents exploding gradients that otherwise occur when training with DP-SGD.



(a) Comparison when using SGD/DP-SGD (Figure from Papernot et al. [PTS+21])



(b) Comparison when using SGD/DP-Adam with the MIMIC-III MLPs.

Figure 5.12: Comparison of the L2-norm of the gradients recorded at every training step for models with ReLU and tempered sigmoids. Moving averages have been applied.

As the experiment in this thesis suggests that tempered sigmoids are not beneficial for the MLP used, it makes sense to reproduce the figure of Papernot et al. with the MLP. Figure 5.12b shows the L2-norms of the gradients for each combination of SGD and DP-Adam with tempered sigmoids and with ReLUs. Similar to the Figure 5.12a of Papernot et al. a moving average smoothes the graph.

The L2-norms of the model that uses DP-Adam and ReLUs are larger than the model's norms that use tempered sigmoids. They are both bigger than the models that use SGD. Still, in comparison to the Figure 5.12a by Papernot et al. [PTS+21] which shows a magnitude of a difference between the L2-norms of DP-Adam with ReLUs and tempered sigmoids, the difference is negligible. The general scale of the gradient norms is different, which suggests that no exploding gradients can be observed in the MLP model of this thesis.

5.7 Exp. 7: Group Normalization

Using group normalization instead of the popular batch normalization that cannot be used to optimize models under DP has been proposed in Section 3.2. Recall that group normalization normalizes the inputs within groups of equal size. Thus, the following experiment uses a width of 120 for the hidden layer. This allows for running experiments with group sizes of $[1, 2, 4, 8, 12, 20, 30, 60, 120]$. Due to the findings in Section 5.5, the depth of the MLP is two. As in the previous experiments, independent studies are executed for each combination of group size and $\epsilon = [1, 2, 4, 9]$. Additionally, a benchmark experiment that uses all other assumptions but does not use group normalization at all is being executed. Each study consists of 600 trials that evaluate three seeds each. Figure 5.13 reports the results of the experiment. Note that the right subplot shows a detail of the left subplot and has a different scale. The horizontal lines that are dashed and labeled with *NG* show the baseline that does not use group normalization.

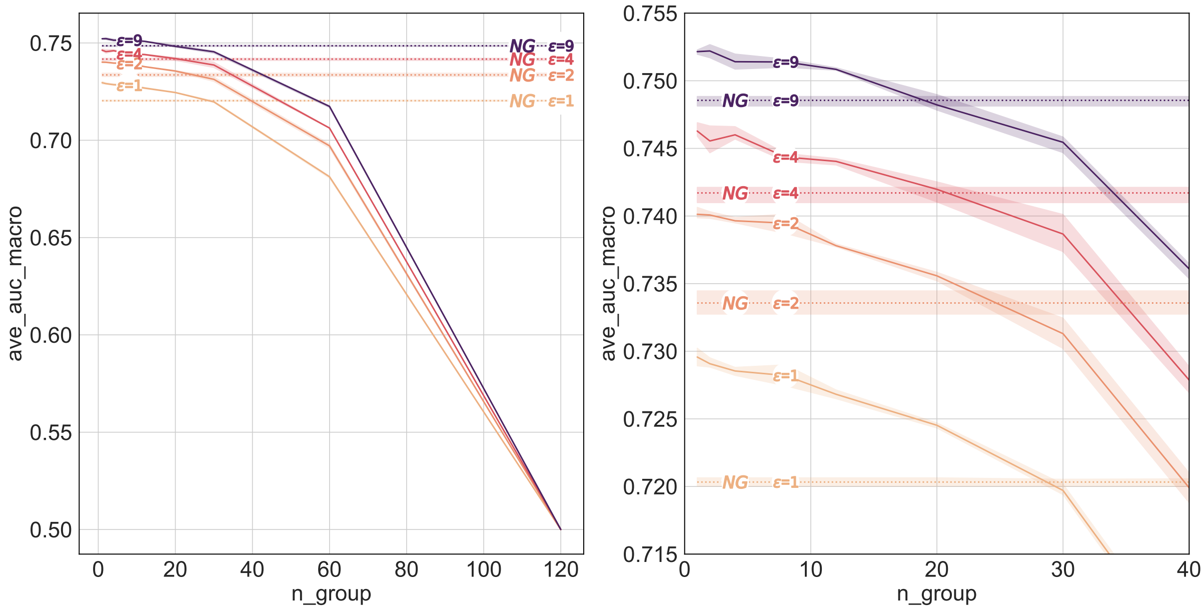


Figure 5.13: Comparison of the best utility for a different number of groups and $\epsilon = [1, 2, 4, 9]$. The solid lines are the results for different numbers of groups, whereas the dotted lines that are also marked with *NG* are the results without group normalization. The right subplot shows a detail of the left subplot.

There are two main insights that this experiment provides. First, the lower groups are being used across all privacy budgets, the better the utility is. Second, with the same number of groups as the number of inputs (120), the macro averaged AUC-ROC drops to

0.5, the same as random. The lousy performance makes sense as a number of groups of 120 means that every input of every sample is normalized to zero mean and unit variance, which does not make much sense.

The difference between using group normalization with a small number of groups and not using group normalization is more apparent in the right subplot than in the left subplot. One can see that using group normalization with a few groups is better than not using group normalization. The number of groups for which the utility with group norm is worse than without group norm differs. However, group normalization with up to 12 groups is better than not using group normalization for all privacy budgets. Using only one group called layer norm shows the best performance across all privacy budgets. The utility gain differs between 0.01 for the macro averaged AUC-ROC for $\epsilon = 1$ and 0.03 for $\epsilon = 9$. The utility gain might seem to be slight, but the utility with one group (layer norm) and $\epsilon = 1$ is significantly closer to the $\epsilon = 2$ utility without group norm than the $\epsilon = 1$ without group norm.

In General, one can conclude from this experiment that across all ϵ, group normalization with one group (layer norm) is superior to not using any group normalization. This section will introduce two subsequent experiments: First, it makes sense to validate if the observation that layer norm is beneficial holds for different widths. The experiment will compare the findings from Section 5.5 that looked at the utility across different widths of the MLP with the same setup but additionally using layer norm.

The setup for the layer norm experiment is equivalent to the experiment in Section 5.5. Please find the details there. The only difference between the two experiments is that additional studies that use group normalization complement the results of the experiment in Section 5.5. One can find the results of the experiment in the left plot of Figure 5.14.

As one can see from the left plot of Figure 5.14 using layer normalization is beneficial for all evaluated widths. Future work could validate this observation over multiple datasets and models.

The second experiment compares the effect of layer normalization in the DP case with the effects when training without DP. DP research must always validate positive effects to the setup without DP because it might be that the layer norm is also beneficial for the non-DP model. Then the gained utility resulting from using the layer norm is not connected to the effects of applying DP-Adam but rather something that has to do with the model itself.

The experiment setup does not differ from the previous experiments, apart from the noticeable changes due to using SGD instead of DP-Adam. Not using DP-SGD reduces the number of hyperparameters, which might affect the performance of the optimization. Besides evaluating layer normalization, the experiment will look at batch normalization, which is not possible to use with *Opacus* as discussed in Section 3.2. The right plot of

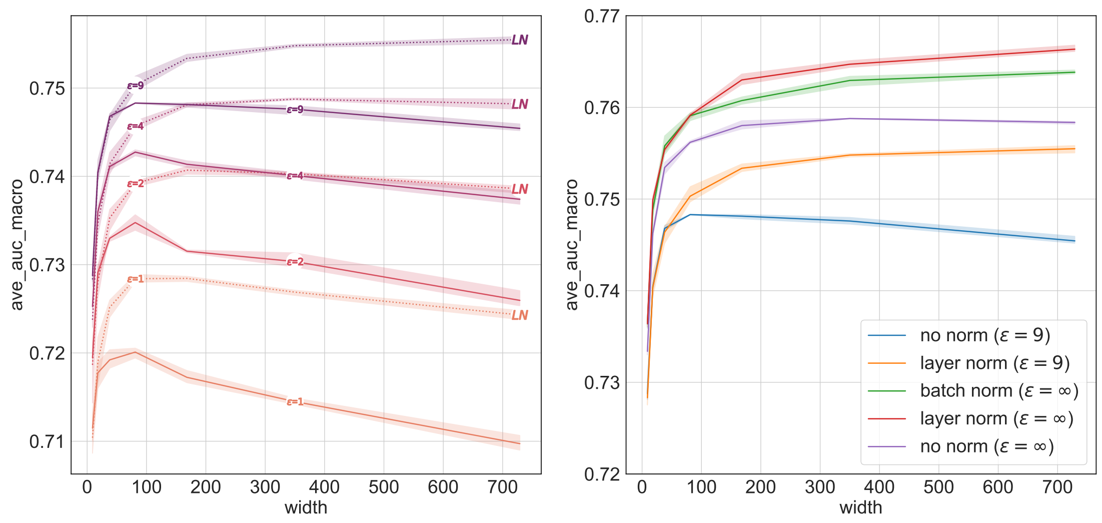


Figure 5.14: *Left plot:* Comparison of the utility with layer normalization (dotted lines, denoted by LN) and without layer normalization for different widths of the network. *Right plot:* The figure compares the utility with and without DP for different widths of the models. Some models contain batch or layer normalization.

Figure 5.14 displays the results from this experiment.

The non-DP model without normalization (purple line) achieves better utility across all model widths than the DP models (orange and blue lines). Especially for a width of 700, the difference is slight, which would suggest that the DP model with layer normalization is almost as good as the non-DP model. The non-DP with batch normalization (green line) and with layer norm (red line) have a better utility than the non-DP model without normalization (purple line). Thus, the boost in utility is not limited to the training under DP but rather beneficial for both training under DP and without DP. In contrast to the claims regarding the tempered sigmoids by Papernot et al. [PTS+21] one cannot conclude that this architecture change might make up for drawbacks from DP-SGD.

6. Discussion

This chapter will discuss the experiments of this thesis on a higher level than in the previous Chapter 5. First, Section 6.1 elaborates on the effect of hyperparameters and the architecture on the utility of the model. Second, Section 6.2 will compare the utility of the DP and non-DP models. Third, Section 6.3 discusses the most obvious limitations that future work should aim to overcome.

6.1 Analysis of Hyperparameters and Architecture

Sections 5.1 to 5.3 assessed the effect of different choices for the hyperparameters noise multiplier, clipping bound, and batch size. The observations are independent of other hyperparameter choices due to the experiment design. This section will start by discussing the sensitivity of the utility to choices of the hyperparameters, which is shown in Figure 6.1. The new figure is necessary to compare the sensitivity toward the different hyperparameters.

As one can observe in Figure 6.1, the utility is less sensitive to choices of the hyperparameters noise multiplier and batch size than to choices of the clipping bound. The high sensitivity to choices of the clipping bound is especially noteworthy as some previous work mainly focuses on optimizing the other hyperparameters and not the clipping bound (see the overview regarding related experiments in Section 4.4). Furthermore, the optimal clipping bound and ϵ are correlated, as observed in Figure 5.6. Given that state-of-the-art articles already include sophisticated hyperparameter optimization, it is not difficult to add the clipping bound to the optimized hyperparameters to obtain ideal privacy/utility tradeoffs.

Throughout the hyperparameter experiments, a tradeoff between the different hyperparameters was apparent. For example, a larger noise multiplier allowed for training more epochs and with bigger batch size. The utility for most of these configurations was nearly the same. However, the exact hyperparameter values get less attention when applying DP to industry applications than the tradeoffs between the privacy guarantee, the utility, and other characteristics. These other characteristics include the training time of the model and the model's size. Section 5.4 briefly examined the tradeoff between

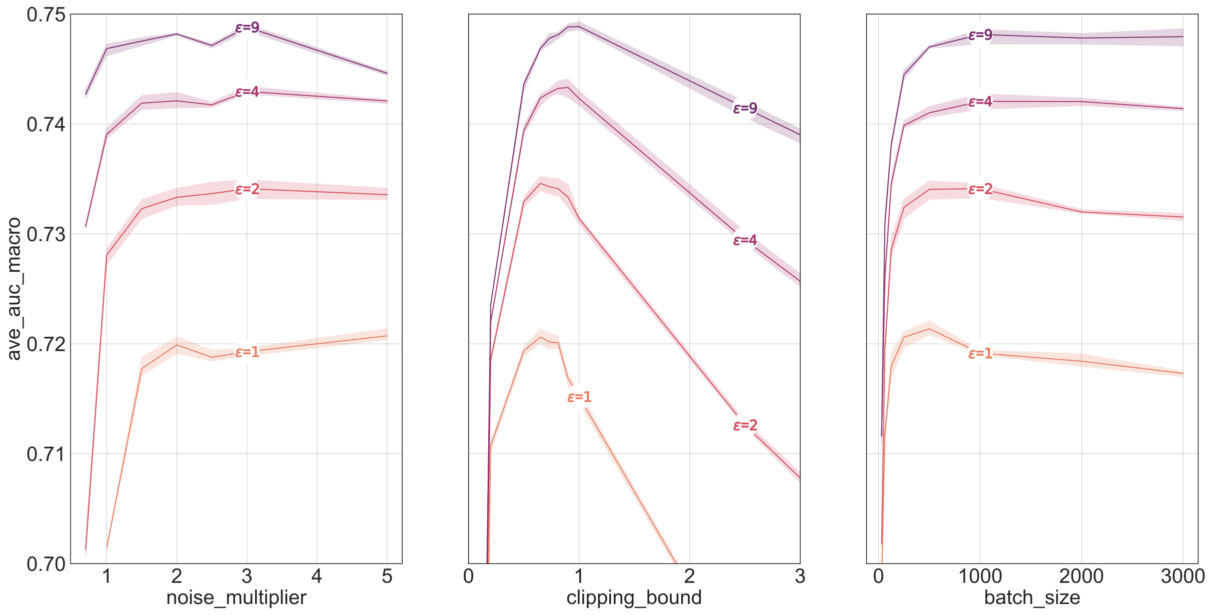


Figure 6.1: The effect of noise multiplier, clipping bound, and batch size on the utility. Please note that this figure is a composition of the Figures 5.1, 5.6 and 5.7, but the plots have been slightly adapted to better show to sensitivity of the utility towards different hyperparameter values.

privacy, utility, and runtime and observed that after a certain threshold, added runtime does not increase the utility much more. Subsequent work could further extend this work by assessing the tradeoff for different models and datasets.

After comparing the effect of the different hyperparameters, the rest of this section will focus on the examined architecture changes. Figure 6.2 contrasts the impact on the utility of changing the depth of the network (left), the width and introducing layer normalization (middle), and different activation functions (right).

As one can see from Figure 6.2, it is first and foremost essential to optimize the depth and width of the network. In comparison to introducing different activation functions or layer normalization, the utility gain of optimizing these two is more significant.

In contrast to the claims by Papernot et al. [PTS+21], discussed in Section 3.1, introducing tempered sigmoids only yields a minor utility improvement for some ϵ and no improvement for others. It is questionable if this improvement is worth it since three more hyperparameters have to be optimized. The tanh performs mostly worse than the ReLU. This debate shows similarities to the dilemma of introducing new activation functions as they have to outperform traditional ones to be accepted by the community.

For the models in this thesis, one can observe a far more significant boost in utility by introducing group normalization. Group normalization with one group, called layer normalization, outperforms models without it across all ϵ and widths. In contrast to the introduction of tempered sigmoids, the improvement is so significant that, e.g., the performance of a network with layer normalization and $\epsilon = 4$ exceeds the performance

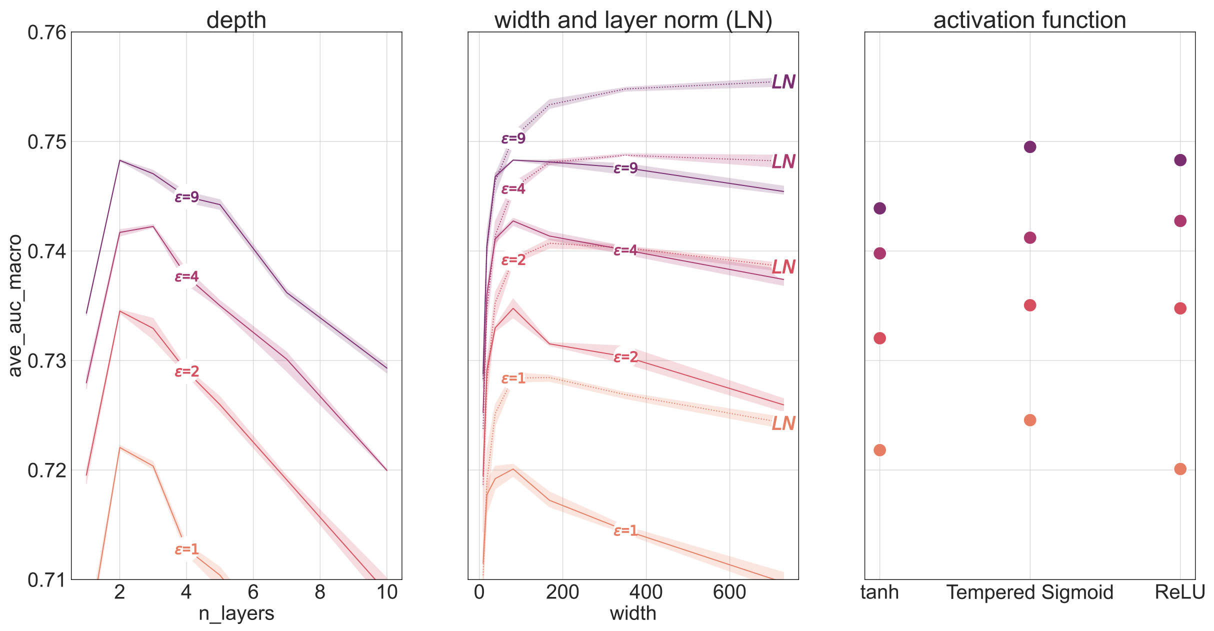


Figure 6.2: The effect of changes to the depth and width of the network compared to introducing a layer norm normalization and changing the activation function. Please note that this figure is a composition of the Figures 5.9, 5.11 and 5.14, but the plots have been slightly adapted to allow for a better comparison.

of one without it but with a higher privacy loss ($\epsilon = 9$). As discussed in the final part of Section 5.7, the boost in utility is not limited to the DP model but instead applies to non-DP models as well. Thus, the effect does not seem to help overcome the drawbacks of DP-SGD.

6.2 Performance of the DP models

As discussed in the previous chapters, the principal aspect of research concerning DP is the tradeoff between utility and privacy. This section will look at two details of this tradeoff. First, the section will compare the models of this thesis with the best utility/privacy tradeoff and the non-DP models reported by Harutyunyan et al. [HKK$^+$19]. Second, it will show using one example how the per-task performance compares. The per-task performance considers the AUC-ROC for each of the 25 diseases of the phenotyping task and does not look at the macro averaged AUC-ROC as the other sections did.

Figure 6.3 compares the utility of different models considered in the thesis. The blue dots represent models trained without DP, and the orange dots show the utility of models trained with DP-SGD. The MLPs trained with a high epsilon ($\epsilon = 9$) show a better utility than the logistic regression from Harutyunyan et al. [HKK$^+$19]. This observation shows that, at least for this task, training with DP does not have to be worse than training without DP. The logistic regression cannot capture the complexity of the task and is thus

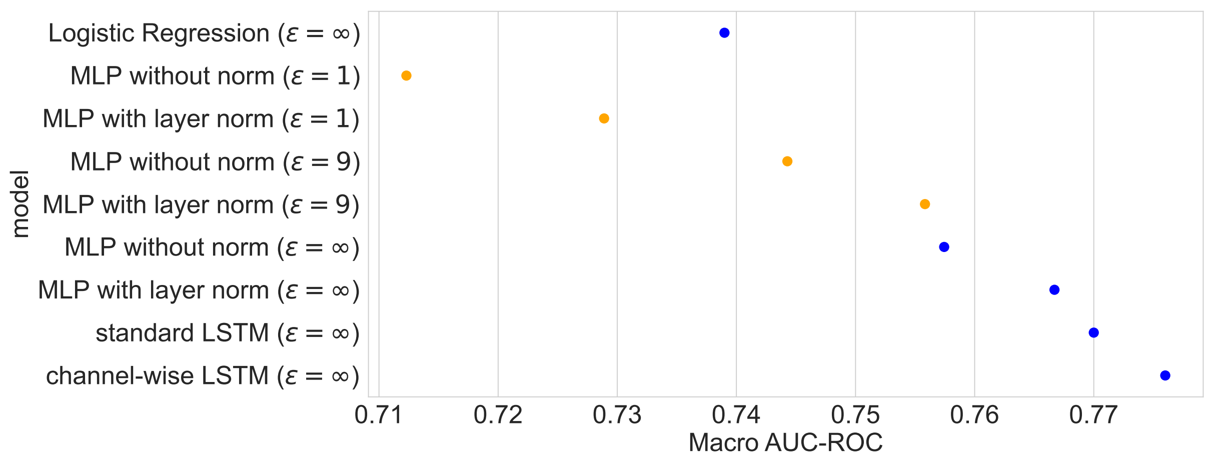


Figure 6.3: Comparision of the AUC-ROC for different models considered in this thesis. Note that the results for the Logistic Regression and LSTMs are from Harutyunyan et al. [HKK+19].

unsuitable. While it makes sense from a complexity argument to prefer simple models over more complex models, the training time of the MLP is shorter than one minute. It would be interesting to attack both models with privacy attacks and see how the empirically measured epsilon (see Section 2.4) compares between the two models.

Another observation is that adding layer normalization improves the performance across all models. Interestingly, such a minor improvement can boost the utility significantly. In the non-DP regime, the MLP with layer normalization yields almost the same utility as the LSTM. At the same time, the LSTM is more complex than the two-layer MLP. Future work could build on these findings, add more DP models, and compare them to existing ones. It would be exciting to compare privacy, runtime, and utility, e.g., by introducing three-dimensional visualizations.

The Figures 6.4 and 6.5 compare the per-task performance of models trained without DP and with DP ($\epsilon = 1/\epsilon = 9$). The models are identical MLPs without any architecture changes. Only the hyperparameters are optimal and thus might vary between the models. Similar to Figure 4.2 of this thesis and Figure 2 of Harutyunyan et al. [HKK+19, p.6] the test set has been 10000 times bootstrapped to obtain the 95% confidence interval.

One can see in Figure 6.4 that while the macro averaged AUC-ROC differs significantly between the model trained under DP and the non-DP model, the 95% confidence interval of some tasks like tasks one, two, and 15 are partially overlapping. The means are wider apart for some tasks than for others. One potential reason for this is that given a relatively strict ϵ of one, some tasks are easier to solve for a DP model than others. The varying difficulty of the tasks is also evident in the fact that the AUC-ROC differs significantly between the different tasks. While the DP model yields a near-random utility (≈ 0.57) for task 19, the AUC-ROC for tasks 2 and 23 is more optimal (≈ 0.84). When training without DP, the same effect is apparent. Note that contrary to the observations

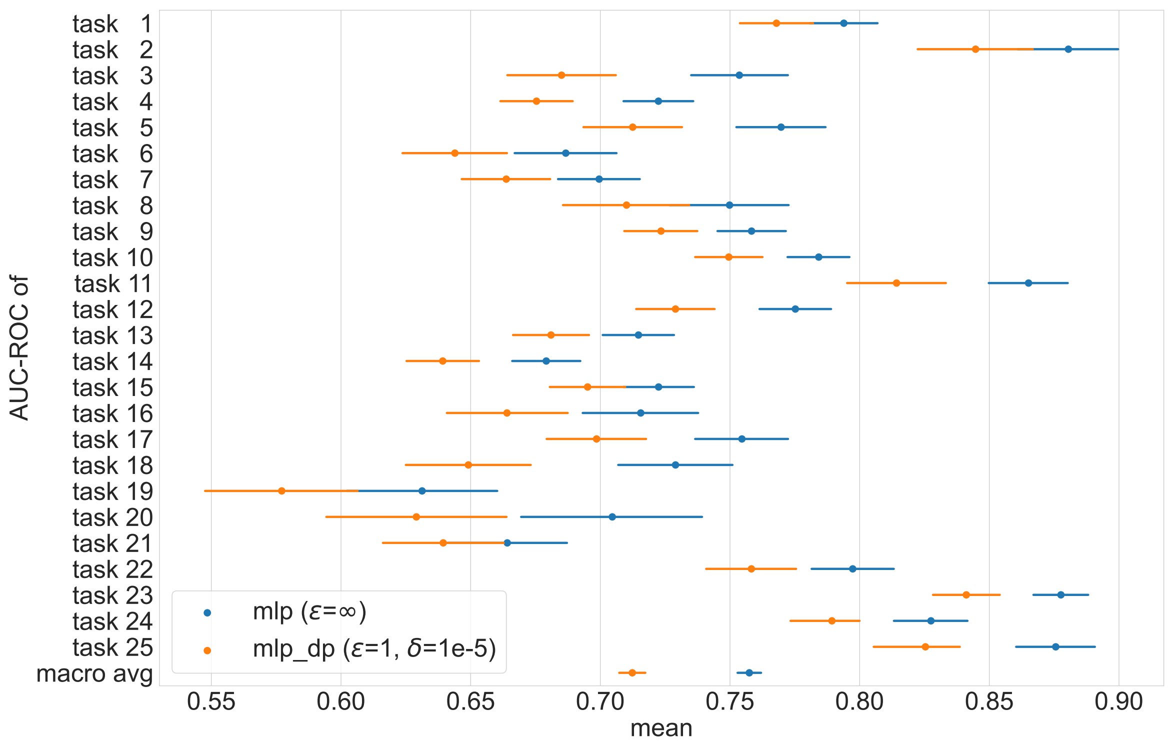


Figure 6.4: Comparison of the per-task performance for identical MLPs trained with/without DP ($\epsilon = \infty/\epsilon = 1$).

regarding the best LSTM reported by Harutyunyan et al. [HKK+19], the worst-performing two tasks are also the ones with the lowest number of occurrences in the dataset. However, task 14, which is among the worst-performing tasks, occurs the most times in the dataset.

Figure 6.5 makes the same comparison as Figure 6.4 but for $\epsilon = 9$. Compared to the previous findings, the per-task performance for $\epsilon = 9$ shows almost indistinguishable utility for some tasks. The 95%-confidence intervals of all tasks are overlapping. For most tasks, the means of the AUC-ROC lie in the confidence interval of each other. For task 21, the means of the AUC-ROC for training without DP and with DP are almost identical. Future work could further analyze the tradeoffs between the different tasks when optimizing for a macro averaged AUC-ROC.

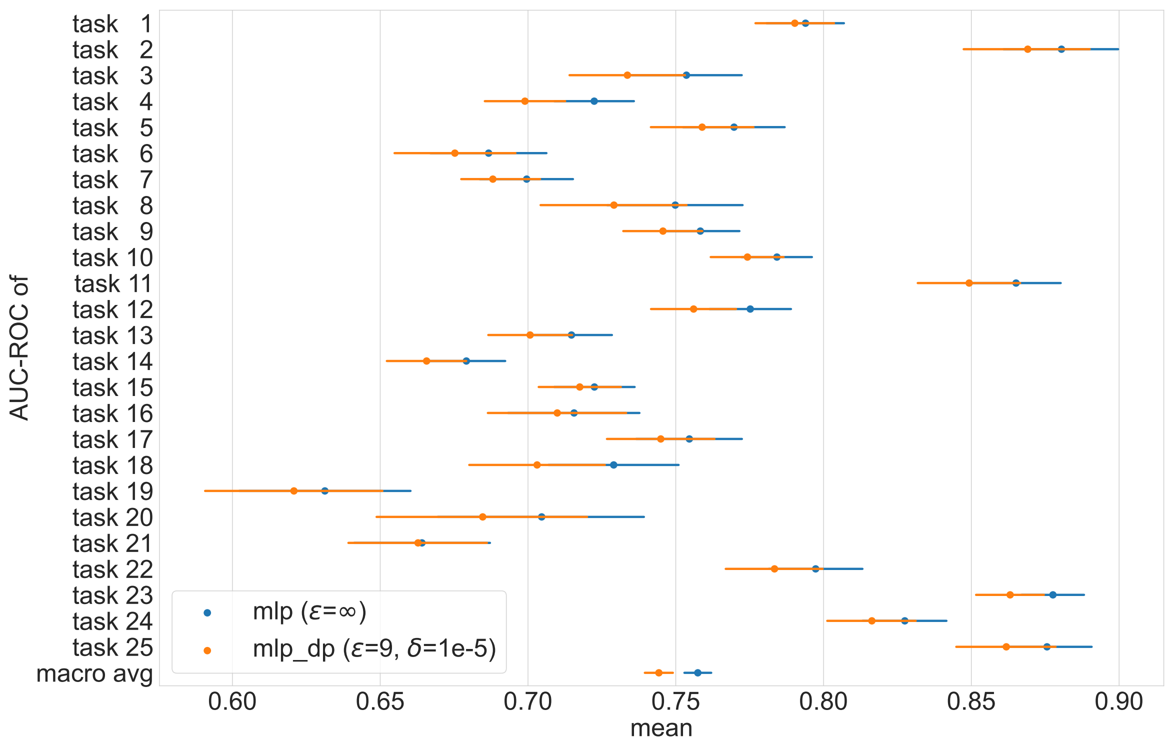


Figure 6.5: Comparison of the per-task performance for identical MLPs trained with/without DP ($\epsilon = \infty / \epsilon = 9$).

6.3 Limitations of the Experiments

The experiments have several limitations that subsequent work could overcome. This section will describe and discuss some of the significant limitations.

First, the experiments are limited to only **one dataset** as they only focus on the phenotyping MIMIC-III task. Thus, the experiments do not provide insights into the behavior of DL models for other MIMIC-III tasks or on other datasets such as MNIST, Fashion-MNIST, or Adult, which are often used in other works (see Section 4.4). Conducting and analyzing the experiments for this scenario provides sufficient material for one Master's thesis. However, the main contributions of this work are the methods, codebase, and experiments necessary to conduct further experiments. In comparison to setting up the experiments for the first time, it is substantially simpler to conduct a selection of the experiments of this thesis for more datasets and tasks. Future work could aim at repeating some experiments for more datasets and models. Especially the obtained marginals of the DP-SGD hyperparameters and the tradeoff between runtime and utility under DP could provide exciting avenues for future research.

Second, the experiments look at **simple models** and do not aim at training more complex models such as the LSTMs provided by the benchmark study of Harutyunyan et al. [HKK+19] under DP. Although training more complex models under DP could yield better utility than the simple models considered in this thesis, there are several reasons why the experiments focussed on simpler models. Besides the longer training time of complex models, more complex models are harder to train under DP (refer to Section 3 for details). As discussed in Section 6.2, the simple models trained under DP already yield nearly the same utility as the non-DP models. It did not seem necessary to extend the work of the thesis in this direction. However, when looking at more complex datasets or tasks in subsequent work, introducing more complex models might be required.

Third, the **neural network architecture search** is limited in his thesis. Similar to Section 3.4, the following paragraphs will discuss the limitations regarding the search space, search strategy, and performance estimation strategy. In this thesis, Bayesian optimization is the search strategy used, and there are no apparent limitations to it.

The experiments' search space contained various architecture modifications, such as the width, depth, different activation functions, and group normalization. However, most architecture experiments aim to measure the effect of changing one aspect of the architecture at a time which might not result in the best overall architecture. For example, the group normalization proposed in Section 5.7 yielded good results across a wide range of widths. Future work could assess how deeper networks benefit from group normalization or if introducing group normalization with a new activation function yields better results than only changing one of the two. Furthermore, as mentioned in Section 3.4, the search space can always be more extensive and could contain more modifications, although the most promising ones have been used as discussed in Section 3.3.

Arguably the most significant limitation of the NAS in this thesis is the performance estimation strategy because the NAS does not use sophisticated methods to assess the performance of a model. Instead, performance evaluation results from fully training the model and reporting the test error over multiple seeds. Neither early stopping nor shared weights are part of the performance estimation strategy. Not using any sophisticated method does not appear to be harmful in the thesis due to the models' small size and training time. However, future work should evaluate more sophisticated strategies before using more complex models.

7. Conclusions

The thesis started by introducing the necessary theoretical background. Chapter 2 introduced DP [DMNS06], which is a state-of-the-art concept to prevent privacy attacks and has preferable properties in comparison to previous concepts. DP bounds the amount of information an attacker can retrieve about single entries of a database while at the same time allowing for learning about the complete database. The most common version of DP is (ϵ, δ)-DP [DKM+06]. The thesis introduced several methods for performing ML under DP that inject noise at different parts of the learning process.

DP-SGD [RA12, SCS13, ACG+16], which injects noise into the gradients of the loss function, is preferable for the experiments in this thesis. One has to optimize the hyperparameters of DP-SGD and the model's architecture as heuristics, and non-DP model architectures do not always yield the ideal utility. The tightness of the privacy guarantees returned by DP methods depends on the accounting method. The privacy guarantees are pessimistic compared to empirically measuring privacy loss obtained by launching more realistic attacks that do not have access to the complete training process.

The models used in this thesis are deep learning models, and Chapter 3 concentrated both on the theory of deep learning and adaptations to the model architecture that can yield better utility. Noteworthy is that larger DL models are more challenging to optimize under DP than without DP. The discussed alternations reported by previous works concern the width and depth of the model, the activation function, and introducing group normalization, which is related to batch normalization. As discussed in Section 3.4, optimizing the architecture is a complex task that requires the use of sophisticated techniques.

As clarified in Chapter 4, the thesis uses the phenotyping task by Harutyunyan et al. [HKK+19] based on the MIMIC-III ICU patient dataset. Bayesian optimization using the library *Optuna* is preferable over a manual, grid, or random search to obtain ideal hyperparameters under limited use of computation power. Previous related experiments only perform grid search, show the utility/privacy tradeoff without specifying it regarding selected hyperparameters or even fix some important hyperparameters. These are all drawbacks this thesis overcame. The previously summarized Chapters 2 to 4 fulfilled the first research objective, which has been to *review the necessary background on differential* *privacy, adaptations of deep learning, and related works.*

The first half of the experiments (Sections 5.1 to 5.3) analyzed the effect of the hyperparameters noise multiplier, clipping bound, and batch size. As discussed in Section 6.1, the utility is particularly sensitive to the clipping bound, whereas the experiments for noise multiplier and clipping bound reported wide intervals that result in similar utility. As a result of these observations, Section 5.4 described another experiment that investigates the tradeoff between training runtime and utility. The utility is not sensitive to the training runtime for the thesis' models, but subsequent work is necessary to confirm this observation for more datasets and models.

The second half of the experiments (Sections 5.5 to 5.7) performed a neural architecture search and examined changes to the depth and width of the network as well as introduced different activation functions and group normalization. Recall from the discussion in Section 6.1 that changing the width and depth of the network has a higher impact on the utility than the other architecture changes. Contrary to existing work by Papernot et al. [PTS+21] neither tempered sigmoids nor tanhs result in significant utility improvements. However, layer normalization improves the utility for the DP models, but this improvement does not seem to be limited to training under DP, as subsequent experiments revealed.

Together with the discussions in Section 6.1, Chapter 5 reached the research objectives two, three and four. The second and third research objective aimed at *analyzing the impact of the hyperparameters introduced by DP-SGD on the privacy/utility tradeoff* and *examining the efficacy of neural architecture changes proposed by previous work.* In order to fulfill the previous two research objectives, the thesis had to *propose a sensible experimental setup to measure them*, which Chapters 4 and 5 described.

Chapter 6 discussed that the models trained under DP yield relatively high utility in comparison to the non-DP models reported by Harutyunyan et al. [HKK+19]. The performance of some tasks is almost indistinguishable when considering a high epsilon ($\epsilon = 9$). Furthermore, limitations of the experiments that could be the subject of future work are the topic of Section 6.3. The experiments' main limitations are the focus on only one dataset and simple models. Future work could extend neural network architecture search. Because Section 6.3 discussed the limitations in-depth, this conclusion does not discuss them further but refers to that section. This chapter reached the fifth and sixth research objectives. Recall that the objectives formulated the need to *compare the non-private baseline models to the private differential models and assess the differentially private models' feasibility* and *identify limitations of the thesis and directions for possible further work.*

This thesis overcame the limitations of previous work by conducting the experiments in a way that allows for accessing the impact of changes of hyperparameter and neural architecture independent from other factors. Thus, this thesis is only a starting point for future work in the area. Extending the work to a broader set of models and datasets could provide additional information for both future research as well as industry applications.

Bibliography

[ABC$^+$16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[Abo18] John M. Abowd. The U.S. census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2867, New York, NY, USA, 2018. Association for Computing Machinery.

[ACG$^+$16] 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*, CCS '16, pages 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[ACPP18] Mário Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Anna Pazii. Local differential privacy on metric spaces: optimizing the trade-off with utility. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 262–267. IEEE, 2018.

[AGD$^+$20] Brendan Avent, Javier González, Tom Diethe, Andrei Paleyes, and Borja Balle. Automatic discovery of privacy–utility Pareto fronts. *Proceedings on Privacy Enhancing Technologies*, 4:5–23, 2020.

[Agg05] Charu C Aggarwal. On k-anonymity and the curse of dimensionality. In *VLDB*, volume 5, pages 901–909, 2005.

[ASY$^+$19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[ATMR21] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *Advances in Neural Information Processing Systems*, 34, 2021.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[BBBK11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[BBKN14] Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. Bounds on the sample complexity for private learning and private data release. *Machine learning*, 94(3):401–437, 2014.

[BGM21] Raef Bassily, Cristóbal Guzmán, and Michael Menart. Differentially private stochastic optimization: New results in convex and non-convex settings. *Advances in Neural Information Processing Systems*, 34, 2021.

[BZ06] Michael Barbaro and Tom Zeller. A face is exposed for AOL searcher no. 4417749. *The New York Times*, 09.08.2006. https://www.nytimes.com/2006/08/09/technology/09aol.html.

[CKN$^+$11] Joseph A. Calandrino, Ann Kilzer, Arvind Narayanan, Edward W. Felten, and Vitaly Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *Proceedings - 2011 IEEE Symposium on Security and Privacy, SP 2011*, Proceedings - IEEE Symposium on Security and Privacy, pages 231–246, 2011.

[CLE$^+$19] 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)*, pages 267–284, 2019.

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.

[CWH20] Xiangyi Chen, Steven Z Wu, and Mingyi Hong. Understanding gradient clipping in private SGD: A geometric perspective. *Advances in Neural Information Processing Systems*, 33:13773–13782, 2020.

[Des19] Damien Desfontaines. Differential privacy in (a bit) more detail. In *Ted is writing things. On privacy, research, and privacy research.*, 20-02-2019. https://desfontain.es/privacy/differential-privacy-in-more-detail.html.

[DKM$^+$06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 486–503. Springer, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[DR$^+$14] 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.

[EMH19] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[Faw06] Tom Fawcett. An introduction to ROC analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[Fin21] Findata - Finnish Social and Health Data Permit Authority. How to ensure efficient and secure use of health data beyond borders? France and Finland collaborating to find answers. In *findata.fi/news*, 23.11.2021. https://findata.fi/en/news/how-to-ensure-efficient-and-secure-use-of-health-data-beyond-borders-france-and-finland-collaborating-to-find-answers/.

[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*, CCS '15, pages 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery.

[FKE$^+$15] Matthias Feurer, Aaron Klein, Jost Eggensperger, Katharina Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970, 2015.

[FLJ+14] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 17–32, USA, 2014. USENIX Association.

[FM16] Daniel Foreman-Mackey. corner.py: Scatterplot matrices in Python. *The Journal of Open Source Software*, 1(2):24, jun 2016.

[GAG+00] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 13.06.2000. Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[GASH69] Bernard G. Greenberg, Abdel-Latif A. Abul-Ela, Walt R. Simmons, and Daniel G. Horvitz. The unrelated question randomized response model: Theoretical framework. *Journal of the American Statistical Association*, 64(326):520–539, 1969.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Gue21] Miguel Guevara. How we are helping developers with differential privacy. In *Google Developers Blog*. Google LLC, 28.01.2021. https://developers.googleblog.com/2021/01/how-were-helping-developers-with-differential-privacy.html.

[HKK+19] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific data*, 6(1):1–18, 2019.

[HSG89] Steven Harp, Tariq Samad, and Aloke Guha. Designing application-specific neural networks using the genetic algorithm. *Advances in neural information processing systems*, 2, 1989.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[JBP+21] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark. MIMIC-IV (version 1.0). *PhysioNet*, 2021.

[JE19] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, pages 1895–1912, USA, 2019. USENIX Association.

[JEP+21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

[JPM16] A. Johnson, T. Pollard, and R. Mark. MIMIC-III clinical database (version 1.4). *PhysioNet*, 2016.

[JPS+16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[JUO20] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private SGD? *Advances in Neural Information Processing Systems*, 33:22205–22216, 2020.

[Kah20] John Kahan. How differential privacy enhances Microsoft's privacy and security tools: SmartNoise early adopter acceleration program launched. In *blogs.microsoft.com*. Microsoft Corporation, 10.12.2020. https://blogs.microsoft.com/on-the-issues/2020/12/10/differential-privacy-smartnoise-early-adopter-acceleration-program/.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KIS$^+$21] Yogesh Kumar, Alexander Ilin, Henri Salo, Sangita Kulathinal, Maarit K Leinonen, and Pekka Marttinen. Medical SANSformers: Training self-supervised transformers without attention for electronic medical records. *arXiv preprint arXiv:2108.13672*, 2021.

[KJH20] Antti Koskela, Joonas Jälkö, and Antti Honkela. Computing tight differential privacy guarantees using FFT. In *International Conference on Artificial Intelligence and Statistics*, pages 2560–2569. PMLR, 2020.

[KLN$^+$11] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[Kor10] A. Korolova. Privacy violations using microtargeted ads: A case study. In *2010 IEEE International Conference on Data Mining Workshops*, volume 1, pages 474–482, 2010.

[LKEW17] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzel. Learning to diagnose with LSTM recurrent neural networks. *International Conference on Learning Representations*, 2017.

[LT19] Jingcheng Liu and Kunal Talwar. Private selection from private candidates. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 298–309, 2019.

[McS16] F. McSherry. Statistical inference considered harmful. https://github.com/frankmcsherry/blog/blob/master/posts/2016-06-14.md, 2016. Accessed: 2022-04-13.

[MEMP$^+$21] Mani Malek Esmaeili, Ilya Mironov, Karthik Prasad, Igor Shilov, and Florian Tramer. Antipodes of label differential privacy: PATE and ALIBI. *Advances in Neural Information Processing Systems*, 34, 2021.

[Mir17] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017.

[MSH$^+$21] Shubhankar Mohapatra, Sajin Sasy, Xi He, Gautam Kamath, and Om Thakkar. The role of adaptive optimizers for honest private hyperparameter selection. *arXiv preprint arXiv:2111.04906*, 2021.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[NST+21] Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlin. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 866–882, 2021.

[PAE+17] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *International Conference on Learning Representations (ICLR)*, 2017.

[PDDC09] Nicolas Pinto, David Doukhan, James J DiCarlo, and David D Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS computational biology*, 5(11):e1000579, 2009.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PMCL18] Sanjay Purushotham, Chuizheng Meng, Zhengping Che, and Yan Liu. Benchmarking deep learning models on large healthcare datasets. *Journal of Biomedical Informatics*, 83:112–134, 2018.

[PMSW18] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[PPR96] Murray M Pollack, Kantilal M Patel, and Urs E Ruttimann. PRISM III: an updated pediatric risk of mortality score. *Critical care medicine*, 24(5):743–752, 1996.

[PS21] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with Renyi differential privacy. *arXiv preprint arXiv:2110.03620*, 2021.

[PSM+18] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfarlfar Erlingsson. Scalable private learning with PATE. In *International Conference on Learning Representations (ICLR)*, 2018.

[PTS+21] Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9312–9321, May 2021.

[RA12] Arun Rajkumar and Shivani Agarwal. A differentially private stochastic gradient descent algorithm for multiparty classification. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 933–941, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[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*, pages 245–248, 2013.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[SS98] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International, 1998.

[SS21] Vikrant Singhal and Thomas Steinke. Privately learning subspaces. *Advances in Neural Information Processing Systems*, 34, 2021.

[SSSS17] 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)*, pages 3–18, 2017.

[TB20] Florian Tramer and Dan Boneh. Differentially private learning needs better features (or much more data). In *International Conference on Learning Representations*, 2020.

[TTS$^+$22] Florian Tramer, Andreas Terzis, Thomas Steinke, Shuang Song, Matthew Jagielski, and Nicholas Carlini. Debugging differential privacy: A case study for privacy auditing. *arXiv preprint arXiv:2202.12219*, 2022.

[War65] Stanley L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965. PMID: 12261830.

[WH18] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[WMC$^+$20] Shirly Wang, Matthew B. A. McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C. Hughes, and Tristan Naumann. MIMIC-Extract: A data extraction, preprocessing, and representation pipeline for MIMIC-III. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, CHIL '20, pages 222–235, New York, NY, USA, 2020. Association for Computing Machinery.

[YGFJ18] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pages 268–282. IEEE, 2018.

[YSS$^+$21] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

[ZLH21] Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-PyTorch tabular: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 3079 – 3090, 2021. also available under https://arxiv.org/abs/2006.13799.

[ZZX$^+$12] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. Functional mechanism: Regression analysis under differential privacy. *Proc. VLDB Endow.*, 5(11):1364–1375, jul 2012.