École Polytechnique Fédérale de Lausanne

Empirical Study of Gradient Descent Algorithms in High-Dimensional Regime

by Daniil Dmitriev

Master Thesis

Approved by the Examining Committee:

Dr. Federica Gerace
Thesis Advisor

Prof. Dr. Sebastian Goldt
External Expert

Prof. Dr. Lenka Zdeborová
Thesis Supervisor

EPFL SB IPHYS SPOC
BSP 722 (Cubotron UNIL)
Rte de la Sorge
CH-1015 Lausanne

June 25, 2021

In the memory of my brother, Nikita. Твоё доброе сердце, улыбка и любовь к науке всегда будут частью меня.

Acknowledgments

I would like to thank Lenka and Federica, for guiding my first steps in research and always answering my questions. I really liked the environment and people in the SPOC lab, the balance of research- and non-research-related discussions we had was just perfect. In my first two years at EPFL, Martin and the whole lab of MLO (and TML) were incredibly helpful and became my friends, with whom we did a lot of things outside of work. In general, I am thankful for the opportunities that EPFL is creating, both scientific and social. From the very first day, the beauty of the campus and the number of interactions and events happening there astonished me. I would also like to thank the team of EPFL-UNIL chaplaincy, the range of activities and the friendliness of students whom I met there were something very important for me during these years.

From a more personal side, I would like to thank my whole family, that is always supporting me. My mother is my role model for how much love and care she gives me and for her interests, spanning from Oriental studies to astronomy. My grandmother Lilya was on my side no matter what I did, and I hope she would have been proud of me now. With my fiancée Dasha we were sharing our love and laughs every day, even across thousands of kilometers, and she was always giving me a lot of motivation and strength. My friend Alisa helped me during one of the most difficult times, and I am very grateful to have her in my life. I met a lot of great people here in Lausanne, and many random encounters have grown into friendships. And even at the beginning of the pandemic, we shared plenty of fun moments with our quarantine crew.

Lastly, I would like to thank a couple of things that I believe enormously contributed to my well-being in the past three years. Surely, it is Lausanne itself, which is an amazing city with a beautiful lake, cathedral, parks and people. I enjoyed walking along its streets and creating memories with different places over time. And my bike, which was one of the first things I purchased in Switzerland, was my loyal companion during everyday commutes and long trips in the surrounding areas. Now I am sure that there is no better way to start a day than to have a refreshing downhill bike ride and no better way to finish a day after studies than to bike up the hilly streets of Lausanne.

This work was done using `https://github.com/hexhive/thesis_template/`

Lausanne, June 25, 2021 Daniil Dmitriev

Abstract

Gradient descent (GD) algorithms perform extremely well on real-life high-dimensional and non-convex problems. To understand the properties of GD, one can use some prototypical high-dimensional problem, e.g. phase retrieval, which can recreate some features of real-life data and also can be analyzed theoretically. In this work, we perform experiments in this setting, using teacher-student framework, both when student shares the same architecture as teacher and when student is using an overparametrized model. In particular, we look at recently proposed Persistent Stochastic Gradient Descent (P-SGD) and at common Stochastic Gradient Descent (SGD) and GD. We find that momentum significantly improves only the performance of P-SGD in the non-overparametrized setting, and improves performances of all algorithms in the overparametrized setting.

Contents

Chapter 1

Introduction

Empirical success of deep learning brings many challenging and intriguing problems for theoreticians to study. There are various approaches for looking into the nature of neural networks, e.g., from the perspective of numerical analysis [22], or learning theory [8], or information theory [1]. In this work, we will use the statistical physics [24] approach. The field of statistical physics has been very actively developing for more than fifty years and achieved a great success in the study of spin glass models [20], graph coloring [25], matrix factorization [5], generalized linear models [3], etc. One of the reasons of this progress is the fact that tools that are developed from one of the problems can be adapted to understand another problem. For our task, the main point of reference is the generalized linear models (GLM).

Since one of the difficulties of analyzing deep neural networks is their high-dimensionality and non-convexity, we will use a model of phase retrieval [3] problem to simulate these properties. Phase retrieval is a practical problem that arises in different areas of physics [2, 13, 21], where the task is to recover a (possibly complex) signal from its projections on multiple vectors. In practice, these vectors may have a certain structure, but we will assume that they come from gaussian distribution, as will be specified further. This allows to study theoretical properties of the problem.

Approximate Message Passing (AMP) [10] algorithms are the state of the art models for almost all of the theoretical problems mentioned above. They benefit from the total knowledge of the task (Bayes-optimality and Nishimori conditions [14]) and are believed to achieve the best possible solution in the class of polynomial algorithms. However, since in real problems, the context of the setting is almost always not given, one has to look for another approaches, e.g. gradient descent algorithms [16], which

iteratively compute the gradient of a certain function, called loss function. They became by far the most common choice in practice, since they are quite easy to implement, allow for many additional heuristics, have a very intuitive explanation, and, most importantly, achieve extremely good results. In contrast to AMP, which can be analyzed very well theoretically with the use of State Evolution [24] formulas, there are many white spots [26] around the behavior of the family of gradient descent algorithms. Our goal in this work is to look at the variants of gradient descent, at the effect of the architecture of the neural network (single layer perceptron model and the committe machine [18]) and at the role of loss function.

Gradient descent algorithm is very computationally expensive to run on large datasets, since for each iteration it requires to compute the gradient based on all samples. Therefore, its stochastic variant (SGD) is being used, where a random subset is selected for each step. Somewhat suprisingly, not only this helps to run models on more data, but also succeeds in finding solutions with very small generalization error. Understanding this phenomena is a very active topic [7, 26]. It is believed that the noise, which is introduced by selecting only the subset, plays a crucial role in escaping the local minimas. Furthermore, such heuristics as momentum and adaptive step size show a significant improvement in the performance on real datasets and speed of convergence. In this work we analyze the effect of momentum in the task of phase retrieval.

One of the ways to study the dynamics of gradient descent is by looking at the gradient flow, which is equivalent to take a infinitesimall learning rate and write the PDEs describing the trajectory. This is called dynamical mean field theory (DMFT) [4]. Directly applying this approach for SGD it not possible, since because of random subsampling of data for each step it does not have a continuous limit when learning rate is close to zero. Therefore, in [11], the persistent-SGD (P-SGD) was introduced, that samples data from Poisson distribution and enjoys the theoretical analysis with DMFT. Interestingly, it seems that there are some significant differences between the behaviors of SGD and PSGD as noted in [12]. We try to further investigate these differences.

A common setting for theoretical deep learning is called teacher-student framework, where teacher and student are two neural networks, possibly same. The labels are generated by the teacher and passed to student, which needs to recover the weights of teacher. The problem of phase retrieval naturally falls into this scenario. The easiest case to analyze is when teacher and student share the same architecture (Bayes-Optimal case), where aforementioned AMP achieves very good results [3] (for phase retrieval, AMP converges to perfect solution at sample complexities around 1.3). Obviously, in practice, we are almost never aware of the exact procedure used to generate

the data. The progress of the last years is due to the fact that very large networks tend to show extremely good result in recovering this hidden dependency between the labels and the data. Therefore, it is important to study this overparametrized setting in our theoretical models. In this work, we look at the simple case of overparametrization, called committee machine, when the last layer is fixed and the student only learns the weights of the first layer.

The scope of this project was very wide in order to potentially find interesting properties for further theoretical analysis. Therefore, we ran many experiments, although each of them might greatly benefit from a closer and more detailed view. Our main contributions in this work are:

1. We look at the effect of momentum for GD, SGD and P-SGD in the Bayes-Optimal setting and in overparametrized setting (committee machine),
2. We show the dependency of the performance based on such hyperparameters as batch size and learning rate,
3. We compare overparametrized setting and single layer setting by looking at the dynamics of convergence and spectrum of the hessian of the loss function,
4. We compare two different losses that can be used for phase retrieval: quadratic and absolute loss,
5. We investigate the role of noise in the SGD and P-SGD with the help of Langevin Dynamics,
6. Lastly, we show part of the same experiments for another problem, symmetric door function.

This manuscript is organized the following way: in Chapter 2, we cover the definitions and assumptions that are used throughout the text. In Chapter 3, we make some practical comments about our experimental setup. In Chapter 4, we describe in detail our experiments and the results. In Chapter 5, we outline some take-aways and future directions.

Chapter 2

Background

2.1 Problem setting

Let n be the number of samples and p the number of dimensions. The data is $X_{i\mu} \sim \mathcal{N}(0, 1/p)$ for $i \in [p], \mu \in [n]$. In theory, we assume that $p \to \infty$ and $n = \mathcal{O}(p)$ with sample complexity $\alpha := n/p$; we recreate this scenario in practice by choosing large values of p.

We are using a teacher-student framework. Let $w_i^* \sim \mathcal{N}(0, 1)$, for $i \in [p]$, be the ground truth *teacher* weights. The labels y_μ are generated as a function $f_0(\mathbf{w}^{*T}\mathbf{X}_\mu)$. The architecture of the *student* network is represented by function $f(\mathbf{w}, \mathbf{X}_\mu)$. Depending on the setting, we can have $f = f_0$ or $f \neq f_0$ (e.g. overparametrized case). Given $(\mathbf{X}, \mathbf{y})$, we find the weights $\mathbf{w}$ by iteratively optimizing some loss function

$$\mathcal{L}(\mathbf{w}) \coloneqq \sum_{\mu=1}^{n} \ell(y_\mu, f(\mathbf{w}, \mathbf{X}_\mu)) \tag{2.1}$$

In this project we will analyze different optimization algorithms in different settings.

2.2 Network architecture

The simplest setting of our framework is when the student is using the same architecture as the teacher, i.e., $\mathbf{w} \in \mathbb{R}^p$ and

$$f(\mathbf{w}, \mathbf{X}_\mu) \coloneqq f_0(\mathbf{w}^T \mathbf{X}_\mu) \tag{2.2}$$

We can also try a setting, where $\mathbf{w} \in \mathbb{R}^{d \times p}$, i.e., we have d hidden units, and the function f can be written as

$$f(\mathbf{w}, \mathbf{X}_\mu) \coloneqq \phi\left(\frac{1}{d}\sum_{i=1}^{d} g(\mathbf{w}_i^T \mathbf{X}_\mu)\right), \tag{2.3}$$

where $g(x)$ is some non-linearity and $\phi(x) = x$ for regression tasks (e.g. phase retrieval) and can be a non-linearity for classification tasks (e.g. symmetric door function). Note that the case without hidden layers is identical to setting $d = 1$.

Preactivations For classification tasks, it is useful to define preactivations, as follows:

$$\begin{aligned} &a_\mu^i \coloneqq g(\mathbf{w}_i^T \mathbf{X}_\mu), \quad a_\mu \coloneqq \frac{1}{d}\sum_{i=1}^{d} a_\mu^i, \\ &\text{so that} \quad f(\mathbf{w}, \mathbf{X}_\mu) = \phi(a_\mu). \end{aligned} \tag{2.4}$$

2.3 Tasks

2.3.1 Phase Retrieval

For the phase retrieval, we have $y_\mu = |\sum_{i=1}^{p} w_i^* X_{i\mu}|$. We minimize one of the following loss functions (which respectively correspont to having quadratic or absolute value activations):

$$\ell(y_\mu, \hat{y}) \coloneqq (y_\mu^2 - \hat{y}^2)^2 \tag{2.5}$$

$$\ell(y_\mu, \hat{y}) \coloneqq (y_\mu - |\hat{y}|)^2 \tag{2.6}$$

In our experiments we mostly focus on the quadratic loss (2.5), but also show some comparison with (2.6).

2.3.2 Symmetric Door Function

This is a standard classification task, where $y_\mu = \text{sign}(|\sum_{i=1}^{p} w_i^* X_{i\mu}| - K)$. Throughout the experiments, we fix $K = 0.67449$ (as in [3]). Also, we use here weights w_i^* from Rademacher distribution: $w_i^* = \pm 1$ with probabilities 1/2. This task is commonly known to be difficult for gradient based optimization algorithms, whereas the GAMP algorithm can reach perfect performance already for $\alpha_{\text{AMP}} = 1.566$. We have two choices of loss functions, both computed on preactivations $a_\mu^i = |\mathbf{w}_i^T \mathbf{X}_\mu|$ (note that for training we smooth the modulus function and use $a_\mu^i = \sqrt{\left(\mathbf{w}_i^T \mathbf{X}_\mu\right)^2 + \varepsilon}$:

$$\ell_{\text{square}}(y_\mu, a_\mu) \coloneqq (y_\mu - a_\mu + \hat{K})^2, \tag{2.7}$$

$$\ell_{\text{logloss}}(y_\mu, a_\mu) \coloneqq \log\left(1 + \exp\left\{-y_\mu(a_\mu - \hat{K})\right\}\right) \tag{2.8}$$

Note that $\hat{K}$ can be either learned or fixed. For the current experiments, we assume that the student is aware of the true value, i.e., $\hat{K} = K$.

2.4 Optimization algorithms

We find the best parameters $\mathbf{w}$ by creating a sequence of weights $\mathbf{w}^t$, with $w_i^0 \sim \mathcal{N}(0,1)$, for $i \in [p]$, and, for $t \in [T] \cup \{0\}$, using the gradient of (2.1),

$$\begin{aligned} \mathbf{g}^t &\coloneqq \sum_{\mu \in \mathcal{M}_t} \nabla \ell(y_\mu, f(\mathbf{w}^t, \mathbf{X}_\mu)) \\ \mathbf{w}^{t+1} &\coloneqq \mathbf{w}^t - \eta_t \mathbf{g}^t \end{aligned} \tag{2.9}$$

We introduced here a learning rate η_t and a set of indices $\mathcal{M}_t$ (both may depend on the step t), which will be chosen depending of the specific algorithm and setting. Let

$$b := \frac{\mathbb{E}|\mathcal{M}_t|}{n} \tag{2.10}$$

be the *expected* batch size.

2.4.1 Full Batch Gradient Descent

In this standard approach, we simply use $\mathcal{M}_t = [n]$, i.e., each time we iterate over the whole set of training samples. Here, $b = 1$. We also use a constant learning rate $\eta_t = \eta$.

2.4.2 Stochastic Gradient Descent

This is another common approach, where we use small batches of samples. In theory, we add each sample in the batch independently with probability b:

$$\forall \mu \in [n], \mu \in \mathcal{M}_t \text{ w.p. } b, \tag{2.11}$$

so that in the end, the expected size of $\mathcal{M}_t$ is indeed bn.

In practice, for each epoch, we randomly divide the training dataset into $1/b$ chunks of size nb each. In our experiments, we **only** use large values of b, between 0.1 and 0.5. As before, learning rate is constant, $\eta_t = \eta$.

Note that SGD the expected value of the gradient based on small batch is the true gradient, but the absence of some elements introduces a noise with a certain variance (the smaller the batch, the larger the noise). This noise implicitly depends on the exact samples selected in the batch.

2.4.3 Persistent Stochastic Gradient Descent

Here, following [12], we use a modified version of SGD, where samples that are selected in the batch may be kept there for several iterations. The formal rule is the following:

for $t = 0$, we use the rule (2.11), and for $t \in [T]$, we use the following:

$$\begin{aligned} P\left(\mu \in \mathcal{M}_t \wedge \mu \notin \mathcal{M}_{t-1}\right) &= \frac{\eta}{\tau} \\ P\left(\mu \notin \mathcal{M}_t \wedge \mu \in \mathcal{M}_{t-1}\right) &= \frac{(1-b)\eta}{b\tau} \end{aligned} \tag{2.12}$$

Here, we introduced a hyperparameter τ – persistence time. We do not thoroughly study the role of persistence time, which was done in [12], and in most of our experiments we use $\tau = 2.0$. We also keep constant learning rate $\eta_t = \eta$ here.

A very interesting property of P-SGD, as we will see further, is that the noise of the gradients is still very large even with large batch sizes. This is because when persistence time τ is large (as in our case), algorithm tend to be biased towards the samples that stay longer in the batch, and when new elements are introduces, this causes large oscillations. As we understand, this leads to the better performance of P-SGD, and study this effect is one of the goals of this work.

2.4.4 Langevin Dynamics

Langevin Dynamics (LD) is closely related to GD, except that at each step together with the gradient a random noise is added to the parameters. This is used to simulate Boltzmann measure [23] and is practical for understanding the effect of noise in SGD and P-SGD. It is believed that the noise is accountable for the success of SGD (see, e.g., [12]), and this noise depends on the particular samples that are selected in the batch. For the LD, on the opposite, the noise does not depend on the samples. By comparing performances of LD with SGD and P-SGD we can see whether this dependency is important.

In [12], P-SGD and SGD are also compared against LD, but they use a more common choice of noise: it is controlled by the temperature parameter T, in the beginning T is fixed and large value (i.e. large noise), and in the end of the training T is set to zero (i.e. no noise, pure GD). This allows to first explore different regions of the loss landscape and in the end optimize in one region.

On the other hand, our goal was to simulate the behaviour of P-SGD, but with random noise. To do this, we first tried to estimate noise as the difference between the batch gradient and the true gradient (scaled). However, this difference appeared to be very large, and the dynamics always diverged. Then, we recorded the difference between the current batch gradient and the previous one. Interestingly, this difference

is much larger for SGD than for P-SGD (for small alpha, e.g., 2.5). Then we use the norm of this difference as the norm of gaussian noise that we add at each step to the gradient in LD (each step the amplitude of noise is different).

In the end, we also perform some training without any noise (as in [12]) in order to fine-tune in the certain region of the loss landscape. Note that this approach cannot be used in practice, since we require the full dynamics of another algorithm on the same data. Also, the noise of P-SGD or SGD can self-adjust and become very close to zero near the optimal point, while in our method we need to set this manually. Therefore, we use it purely as a potential method of analysing P-SGD and SGD.

2.4.5 Momentum

We can modify the algorithm in (2.9) and introduce *momentum*, which accelerates and stabilizes the convergence of the gradient descent algorithms. There exists two formulations – **standard** momentum:

$$\begin{aligned} \mathbf{g}^t &\coloneqq \sum_{\mu \in \mathcal{M}_t} \nabla \ell(y_\mu, f(\mathbf{w}^t, \mathbf{X}_\mu)) \\ \mathbf{m}^{t+1} &\coloneqq \gamma \mathbf{m}^t + \mathbf{g}^t \\ \mathbf{w}^{t+1} &\coloneqq \mathbf{w}^t - \eta_t \mathbf{m}^{t+1} \end{aligned} \tag{2.13}$$

and **Nesterov Accelerated** momentum:

$$\begin{aligned} \mathbf{g}^t &\coloneqq \sum_{\mu \in \mathcal{M}_t} \nabla \ell(y_\mu, f(\mathbf{w}^t, \mathbf{X}_\mu)) \\ \mathbf{m}^{t+1} &\coloneqq \gamma \mathbf{m}^t + \mathbf{g}^t \\ \mathbf{w}^{t+1} &\coloneqq \mathbf{w}^t - \eta_t (\mathbf{g}^t + \gamma \mathbf{m}^{t+1}) \end{aligned} \tag{2.14}$$

In our experiments, momentum coefficient is always equal to 0.9 (if not zero).

2.4.6 Adam

There exist a huge number of GD variants which use adaptive learning rates based on the local gradient properties, Adam [6] is one of them. We will use it in our experiments, as another common accelerated optimization method.

Chapter 3

Implementation Details

Initialization In the work of [12], the setting is a little different from ours: both teacher weights and data are initialized as $\mathcal{N}(0,1)$ and labels are defined as $f_0\left(\frac{1}{\sqrt{p}}\mathbf{w}^{*T}\mathbf{X}_\mu\right)$. Sometimes teacher weights are initialized from $\mathcal{N}(0,1/p)$ and data from $\mathcal{N}(0,1)$. Although all these settings seem to be identical (the distribution of the labels y_μ is the same), the dynamics of these models differ and one need to be very careful with the choice of learning rate when trying to reproduce results from different setting.

Number of features In order to get rid of small size effects and in the meantime be able to compute the results in a reasonable time, we use 1024 features throughout most of the experiments, if not written otherwise.

Projection Note that in the literature it is common to apply projection on the p-dimensional sphere with radius $\sqrt{p}$. In our results, we **always use** projection, unless specified otherwise. Note that projection should not be applied when weights are initialized from $\mathcal{N}(0,1/p)$.

Stopping criteria Due to computational limits we need to specify the conditions when the training will be stopped. One of the main difficulties here is that when the train performance reached a plateau it is very hard to say if it is possible to escape the plateau in the future. Therefore, we used the following criteria:

1. Train error is below 1×10^{-8},

2. Test error is below 1×10^{-3},
3. $\left\|\mathbf{g}^t\right\|_\infty$ is below 5×10^{-10}.

Number of epochs In most of our experiments, for the case of single layer model, we use $20\,000$ epochs. For sample complexities close to the convergence threshold, the actual time needed for algorithms to converge may exceed this time limit. For P-SGD, sometimes we need to go to even $200\,000$ epochs, which improves results. In the case of multiple layer models, we use large number of epochs, up to $100\,000$. All of this shows that it is tricky to directly compare the performance of the algorithm regardless of their speed of convergence.

Averaging over seeds For the plots, we run several experiments with the same parameters and plotted the average. We fix the teacher vector across seeds and for each seed generate a different data (loss landscape) and student vector (initialization). For most of the plots we show the success probabilities. A trial is considered as a success is the final test error is below 5×10^{-3}. In some experiments, we use the fixed loss landscape and only change the initialization. We explicitly mention if this is the case.

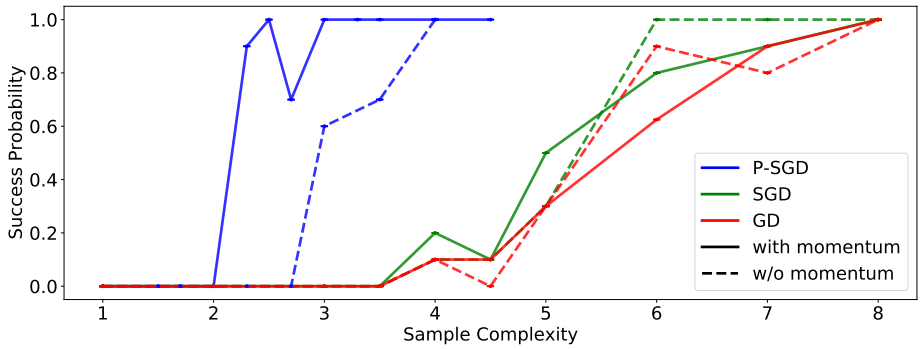
Code We used PyTorch [15] for the implementation. The codebase can be found at `https://github.com/daniildmitriev/spoc_master_project`. For the P-SGD, we implemented a PoisSampler class that selects batches based on the Poisson Distribution. For the Langevin Dynamics, the code for SGD was adapted to include an arbitrary noise at each step. The experiments were run on the FIDIS cluster of EPFL.

Chapter 4

Experiments

4.1 Phase retrieval, no hidden units

Figure 4.1: Phase retrieval, no hidden units, quadratic loss. Learning rate: 0.005, batch size (for P-SGD and SGD): 0.5, momentum coefficient: 0.9 (as for all runs with momentum for all experiments). Runs with momentum use **Nesterov** momentum. Here, there is a large difference between P-SGD and other algorithms; moreover, momentum appears to further help P-SGD, by making it converge at sample complexities around 2.0. Each data point is an average over 10 runs.



We start, following [12] and [19], by looking in the setting without hidden units and with quadratic loss, with small learning rate 0.005 and large batch size 0.5, at how do P-SGD, SGD and GD compare to each other, and what is the effect of momentum. On the Figure 4.1 we see several effects:

1. Adding momentum to GD or SGD does not have a strong effect (a similar result, but in another setting of matrix-tensor model was obtained in [9]),

2. The advantage of P-SGD over GD and SGD is significant, as also found in [12],

3. Adding momentum to P-SGD even further improves the performance.

To test for the finite size effects, we conducted experiments with other system sizes (from 512, up to 4096), see results on Figure 4.2. From the plot we can say that the same behavior is present both on small and large sizes, thus the finite size effects are small. We also observe a similar shift in the success probabilities when increasing system size for GD, as in [19]. But for P-SGD without and with momentum, the differences between curves are smaller, e.g. for P-SGD with momentum all curves have transition around $\alpha = 2.0$. For the experiment in Figure 4.1 we used Nesterov momentum, but we did not find any difference between it and standard momentum, therefore, in Figure 4.2 and for further experiments, we just use standard momentum.

Now, let us look at the role of batch size and learning rate for P-SGD (Figure 4.3). Overall, we see that decreasing batch size and increasing learning rate has positive effect. However, for smaller batch sizes or larger learning rates, the dynamics becomes unstable and many experiments diverge. Therefore, we decided to use $b = 0.5$ and $\eta = 0.005$ in most of the experiments.

Figure 4.2: Comparison for different system sizes. Same setting, as in Figure 4.1, except runs with momentum use **standard** momentum (as for all further experiments). It is clear that the effect of momentum for P-SGD and advantage of P-SGD over GD remains even for larger system sizes. Each data point is an average over 20 runs for 512, 1024, and 2048 features and over 10 runs for 4096 features.

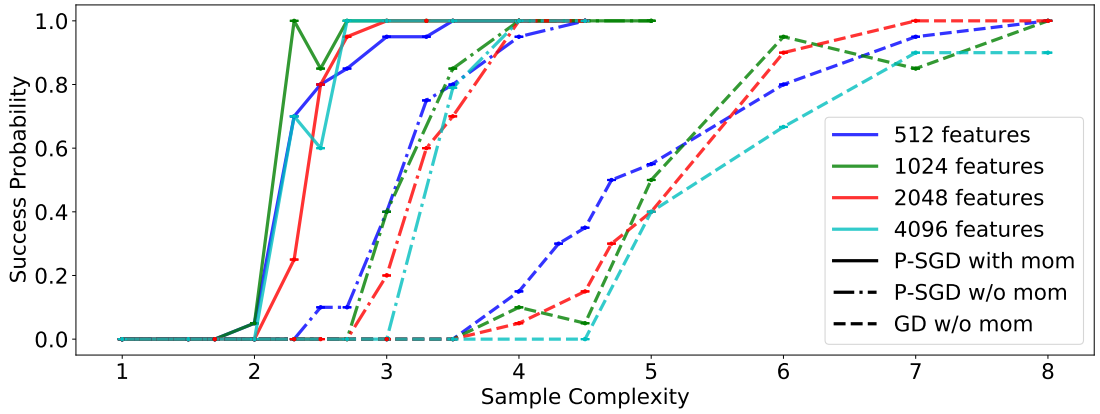
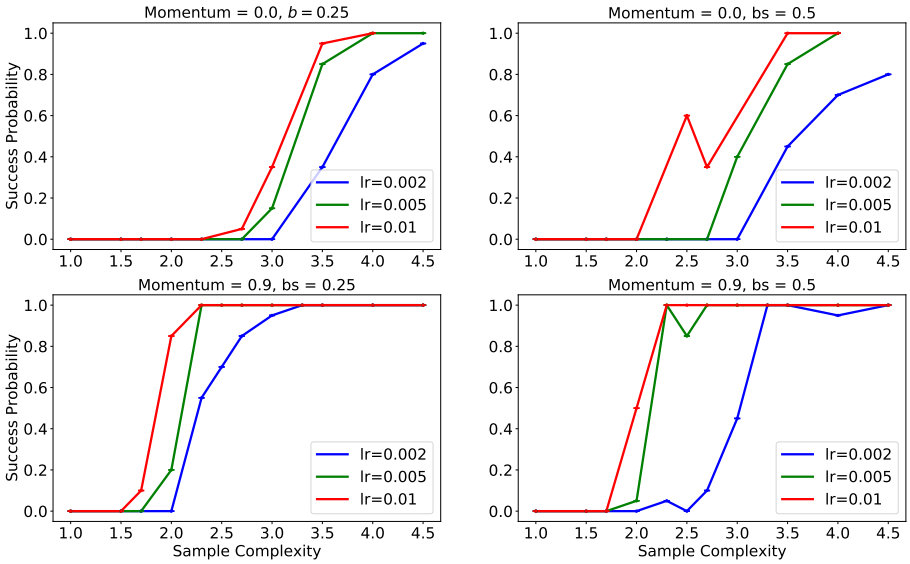


Figure 4.3: We look at P-SGD without momentum (**top**) and P-SGD with momentum (**bottom**), for batch size = 0.25 (**left**) and batch size = 0.5 (**right**), for 3 different learning rates. Overall, the result is that larger learning rates perform better, and reducing the batch size may help a bit. Although, it may happen that smaller learning rates perform exactly the same, but only require more time to converge. Further increase of learning rate and/or decrease of batch size may lead to unstable dynamics. Each data point is an average over 20 runs.

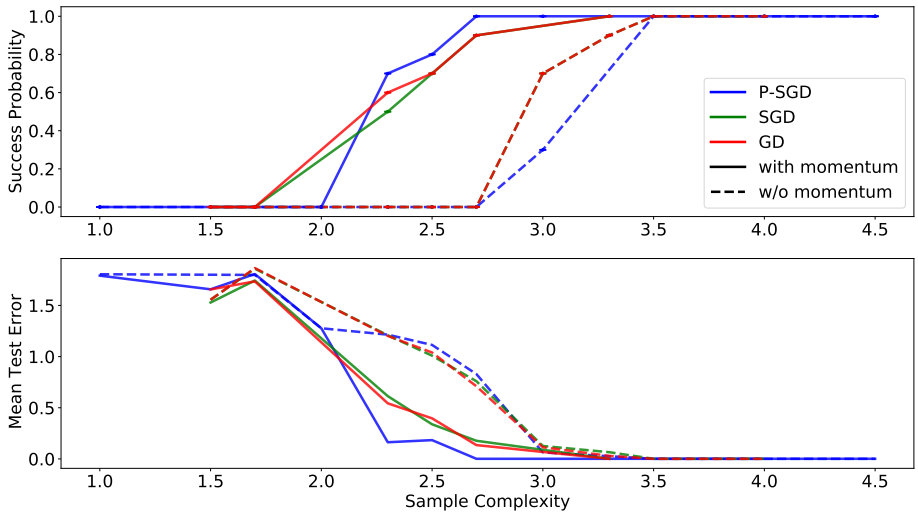


4.2 Adding hidden units

Next, we look into overparametrized regime. This is a very interesting setting, since i) in practice, we are never aware of the exact procedure used to generate data and using very large neural networks is a very common approach and ii) in the paper [17], for the overparametrized GD and phase retrieval, the critical threshold $\alpha_c = 2.0$ for the convergence was theoretically proven. Here we look closer at the dynamics of the algorithms. To do this, we first reproduce the Figure 4.1 by adding to the model a hidden layer with 2 neurons. This falls intro the setting of [17]. On Figure 4.4 the plots of success probabilities and mean test error are shown.

Interestingly, now there is almost no difference between algorithms, but the effect of momentum is clearly visible in all of them. The latter can be explained by the fact that we restrict the number of epochs (to 20 000), and the convergence becomes

Figure 4.4: Models with 2 hidden units. We show the success probability (**top**) and mean test error (**bottom**). Now, we see that there is almost no difference between algorithms (in fact, GD and SGD behave almost exactly the same), and the performance for GD and SGD is improved. The effect of momentum is also clear now for all models. Each data point is an average over 10 runs.

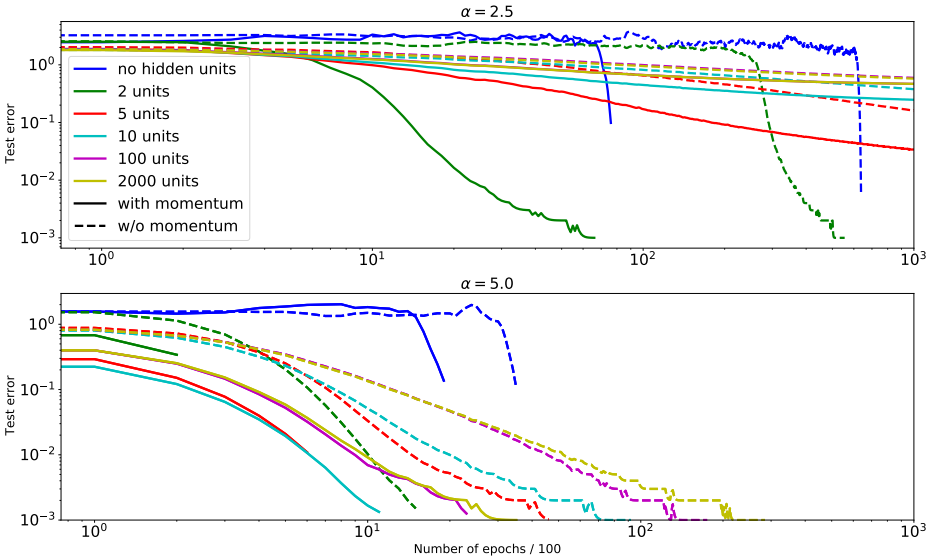


slower, when the architectures of student and teacher are different (see [17]). Therefore, momentum helps to speed up the convergence, and reach smaller error in given time.

In practice, however, architectures are highly overparametrized, therefore, we performed experiments with many more hidden units (up to 2000). Interestingly, this only slows down the convergence. We looked closer at this phenomena by plotting the dynamics of the runs for some specific seed. On Figure 4.5, we see that for models with 2 hidden units converge faster or similarly to the no hidden units models, but if we further increase number of units, the performance slows down significantly. Momentum helps for all the architectures. We tried to increase the learning rate, e.g., for the model with 5 hidden units, but this did not give any result — the model either diverges, when the learning rate is too high, or is still stuck at similar errors. These result are in line with the results obtained in [17] (see, e.g., Figure 2 in [17]).

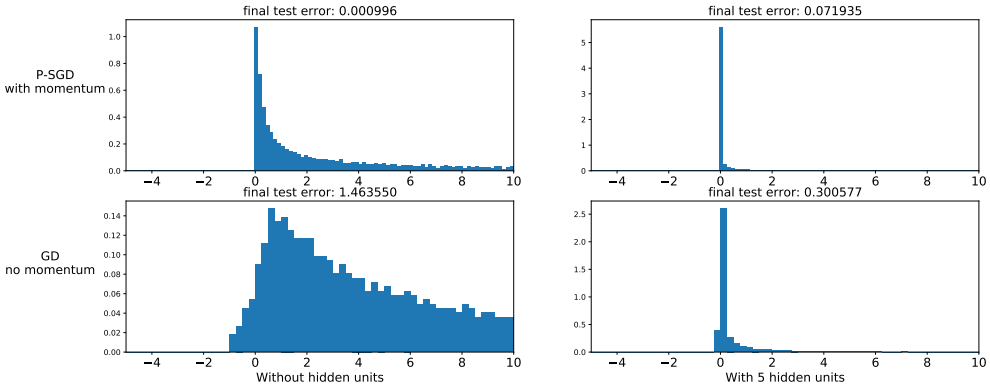
We can look into whether the algorithms are stuck in the local minimas by plotting the spectrum of the eigenvalues of the Hessian, see Figure 4.6. For P-SGD with momentum and no hidden units, all the eigenvalues are positive, and many have a large absolute value. When we add hidden units, the eigenvalues start to be much more

Figure 4.5: Dynamics of one seed, for P-SGD, for $\alpha = 2.5$ (**top**) and $\alpha = 5.0$ (**bottom**). Here, we see in more detail the optimisation process. When there are no hidden units, the model spends a lot of time in the very high test error regime, until it overcomes some threshold and starts to quickly converge to perfect recovery. **Note:** when dynamics stops before the end (e.g. for runs without hidden units / with 2 hidden units), it means that the model reached the test error 1×10^{-3}, which was on of the stopping criteria, but this point is not shown on the plot. In case with 2 hidden units, somewhat similar picture, with smoother convergence. For 5, 10, 100 and 2000 hidden units, the convergence is significantly slowed down, and even after 100k epochs, the models cannot reach reasonable performance. Momentum appears to help across the algorithms.



concentrated around zero, and in fact, some of them are negative. This supports the idea that the optimization landscape is difficult and the model requires much more time to escape saddle points. For the GD without hidden units, many eigenvalues are negative, which means that the model is very far from the local minimum. And adding hidden units has similar effect: more eigenvalues become positive, but most of them are very close to zero.

Figure 4.6: Here we show the distribution of the eigenvalues at the **end** of training for $\alpha = 2.5$: P-SGD with momentum (**top**), GD without momentum (**bottom**), architectures without hidden units (**left**), architectures with 5 hidden units (**right**). For each plot, the final test error is shown on top. Among these models, the top left (P-SGD with momentum, without hidden units) can reach perfect performance.



4.3 Additional experiments

In this section, we show some directions, that were less thoroughly investigated, but that still are of interest.

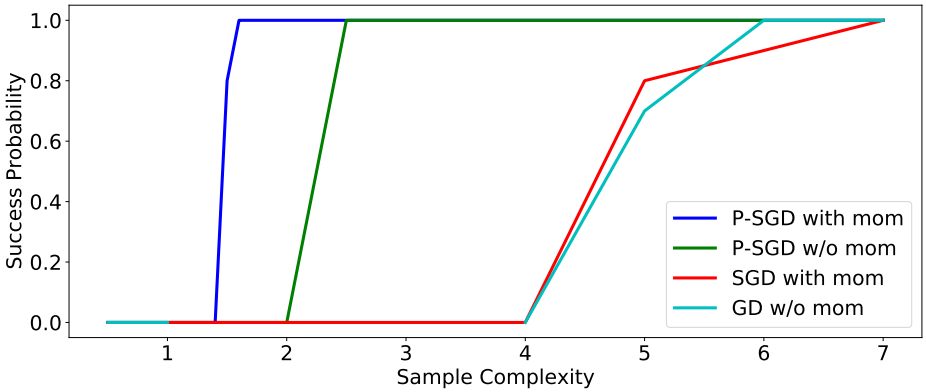
4.3.1 Threshold for P-SGD with momentum

We run a set of experiments to look if the threshold for P-SGD with momentum can be further improved. For this, we increased number of epochs and the learning rate, see Figure 4.7 and Table 4.1. Also, in this experiments, the data was kept the same throughout all 10 runs. Interestingly, P-SGD with momentum still works for sample complexities around 1.5. For more reliable results, experiments for different seeds and for different system sizes need to be performed.

4.3.2 Langevin dynamics

To see what are the important differences between P-SGD and GD that result in the much better performance, we extracted the noise as the difference between the consequent gradients from P-SGD and recorded its absolute value. Then, we ran Langevin

Figure 4.7: Success probabilities for phase retrieval, 1024 features, no hidden units. For this plot, we tune the learning rate and the total training time for the P-SGD with momentum to find a smallest sample complexity, when it still can converge. See Table 4.1 for the specific hyperparameters. Each data point is an average over 10 runs. Data distribution is kept the same for all runs, only student initialization is different.

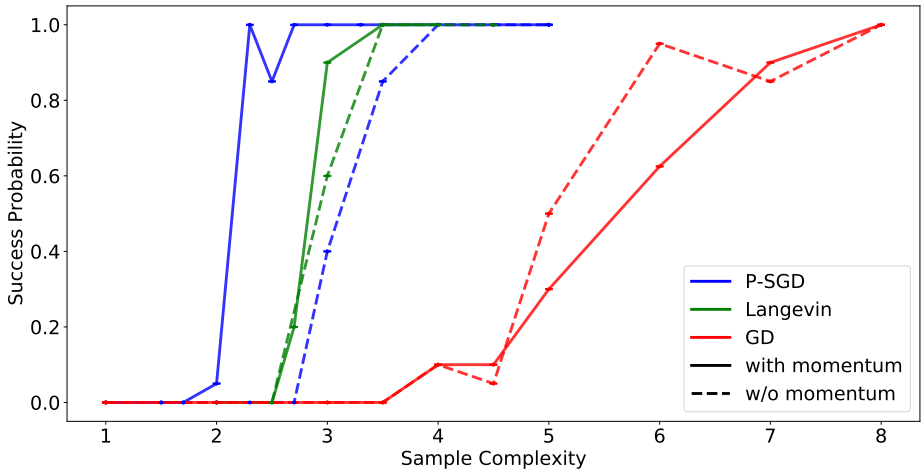


Dynamics, where at each step we additionally add gaussian noise with the same magnitude as the recorded gradient difference from P-SGD. See the more detailed description at Section 2.4.4. On Figure 4.8, we can see that LD without momentum even outperformes P-SGD without momentum, and is much better that GD. Momentum does not help a lot, and P-SGD with momentum is still the best model.

Sample complexity	Learning rate	Epochs to converge
2.5	0.005	6k-10k
2.0	0.005	25k-60k
2.0	0.01	15k-25k
1.8	0.01	15k-50k
1.7	0.01	35k-70k
1.6	0.01	45k-60k
1.5	0.012	85k-200k+
1.4	does not converge	

Table 4.1: Required number of epochs for P-SGD with momentum to converge. As we approach the threshold sample complexity, below which the algorithm does not converge, this number increases a lot.

Figure 4.8: Comparison of the Langevin Dynamics to P-SGD and GD. The temperature for LD is chosen at each step specifically to recreate the difference between consequent gradients of P-SGD (as described in Section 2.4.4). LD models are also additionally trained without any noise in the end. We see that LD without momentum can even outperform P-SGD without momentum. This requires running more detailed experiments.



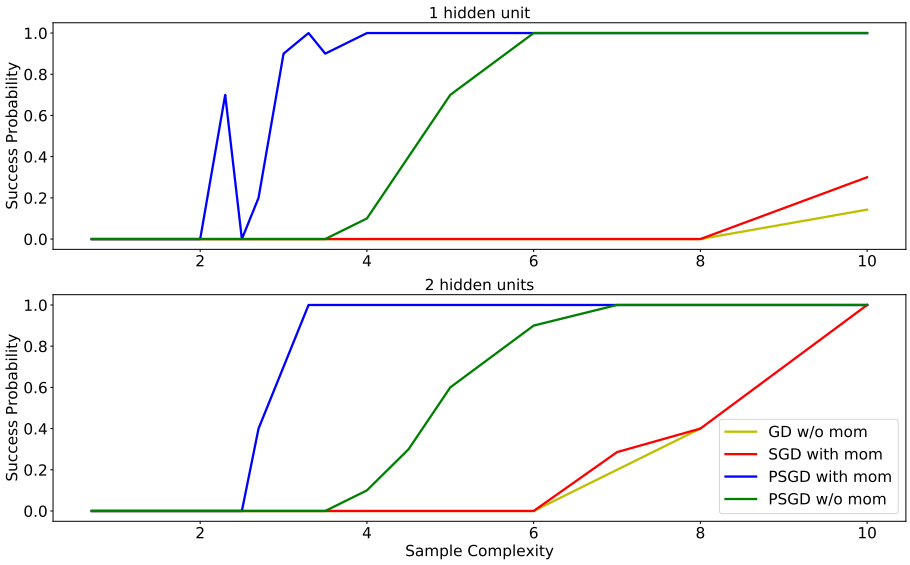
4.3.3 Phase retrieval with absolute loss

So far we only worked with quadratic loss. In Section 2.3.1, we also introduce the absolute loss, and in Figure 4.9 we look at its performance. From the data, it seems that absolute loss is inferior to quadratic loss. However, this needs to be checked, e.g., for SGD with small batch size (remember, we always use batch size $= 0.5$ in our experiments).

4.3.4 Symmetric door function

This is another prototypical problem (see Section 2.3.2 for definition), where gradient descent algorithms perform much worse than AMP (see [3]). Therefore it would be interesting to see if some heuristics or different architectures can help to improve the performance. On Figures 4.10 and 4.11 we look at this problem. From the results, one can say that P-SGD is indeed improving the performance a little, but otherwise, there is no clear effect neither of momentum, nor of adding hidden units.

Figure 4.9: Absolute loss function for phase retrieval. We compare models without hidden units (**top**) and with two hidden units (**bottom**). We see that the absolute loss hinders GD and SGD, but P-SGD performs roughly the same. Adding hidden units helps GD and SGD, and does not have a significant effect for P-SGD. Quadratic loss performs better across all the optimizers. Each data point is an average over 10 runs. Data distribution is kept the same for all runs, only student initialization is different.



4.3.5 Adam dynamics

In order to tackle the problem of overparametrized models getting stuck and not reaching good performance in required time, we used a very common acceleration method, Adam. We repeat the experiments from Figure 4.5 for the Adam algorithm and show the dynamics of two seeds for the same sample complexity 2.5 on Figure 4.12.

We can see that not only Adam cannot improve models with large number of hidden units, but also models without hidden units or with two hidden units perform much worse. However, proper hyperparameter tuning was not performed for Adam and it may benefit from more careful study. But it is possible to say that it is not straightforward to significantly accelerate highly overparametrized models.

Figure 4.10: Symmetric door function, 1024 features, comparison of square (**left**) and logistic (**right**) loss functions. Overall, logistic loss performs better, especially for GD and SGD. As in phase retrieval, we also see the advantage of P-SGD here, but now adding momentum does not have a significant effect. Each data point is an average over 10 runs. Data distribution is kept the same for all runs, only student initialization is different.

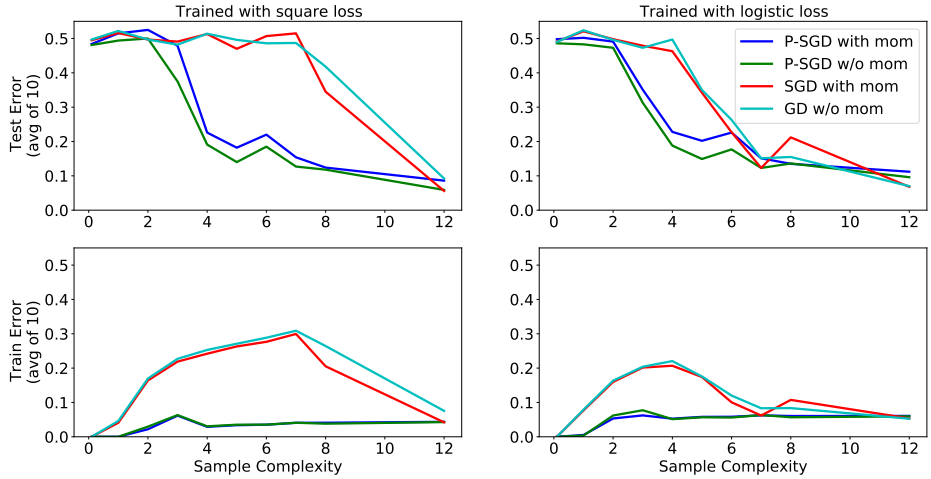


Figure 4.11: Symmetric door function, comparison of models without hidden units (**left**) and with two hidden units (**right**). It seems that, in contrast to phase retrieval, adding hidden units does not help GD or SGD and even hinders P-SGD. Each data point is an average over 10 runs. Data distribution is kept the same for all runs, only student initialization is different.

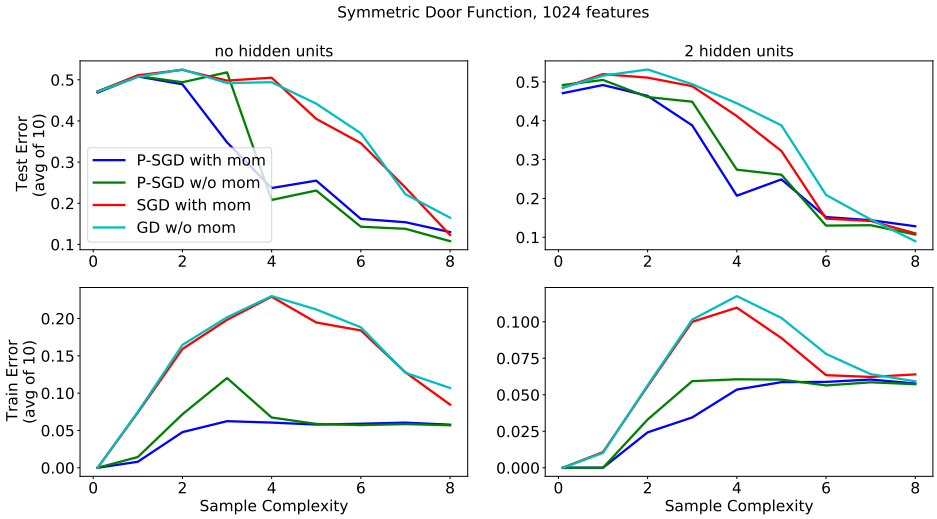
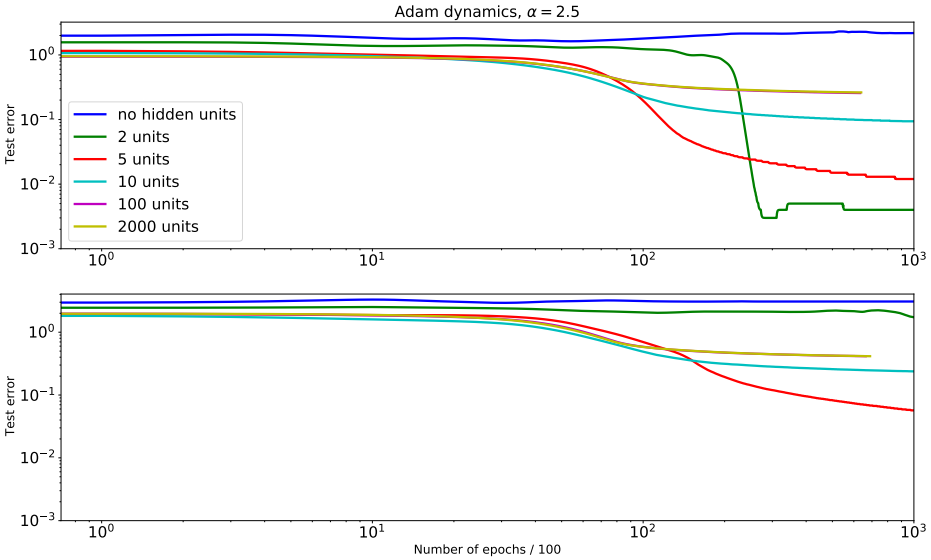


Figure 4.12: Dynamics for two different seeds, for Adam, for $\alpha = 2.5$. Here, we see in more detail the optimisation process. Unlike P-SGD, Adam cannot reach test error 1×10^{-3} even for models without hidden units or with two hidden units. For the dynamics on the top, we see that model with two hidden units reached a small test error at some point, but is still stuck at relatively large values.



Chapter 5

Conclusion

As the title of the project says, we performed many empirical experiments that uncover some interesting phenomena — the advantage of P-SGD with momentum in the setting without hidden units, the effect of overparametrization and slowing down of extreme overparametrization, and some insights on the role of noise in P-SGD (with the help of LD). Now, the reasonable next step would be to answer the question *why* is this happening. To be precise, we think that the following directions are promising:

1. Obtain, e.g., with DMFT [9, 12], a theoretical validation of the fact that momentum helps P-SGD, and does not help GD for the phase retrieval problem,
2. Look into the overparametrized case and explain i) why we observe the slowing down when adding more hidden units (since this is not what happens in practice) and ii) why and to what extent does momentum help,
3. Understand what is the role of noise in P-SGD, perhaps with the help of LD. Also see if SGD with small batch size (what people usually use in practice) behaves similarly to P-SGD.

Further, similar questions can be answered for the symmetric door function. We believe that all these questions contain many more mysteries within them.

Bibliography

[1] Emmanuel Abbe and Colin Sandon. “Poly-time universality and limitations of deep learning”. In: *arXiv preprint arXiv:2001.02992* (2020).

[2] Radu Balan, Pete Casazza, and Dan Edidin. “On signal reconstruction without phase”. In: *Applied and Computational Harmonic Analysis* 20.3 (2006), pp. 345–356.

[3] Jean Barbier, Florent Krzakala, Nicolas Macris, Léo Miolane, and Lenka Zdeborová. “Optimal errors and phase transitions in high-dimensional generalized linear models”. In: *Proceedings of the National Academy of Sciences* 116.12 (2019), pp. 5451–5460.

[4] Antoine Georges, Gabriel Kotliar, Werner Krauth, and Marcelo J Rozenberg. “Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions”. In: *Reviews of Modern Physics* 68.1 (1996), p. 13.

[5] Yoshiyuki Kabashima, Florent Krzakala, Marc Mézard, Ayaka Sakata, and Lenka Zdeborová. “Phase transitions and sample complexity in bayes-optimal matrix factorization”. In: *IEEE Transactions on information theory* 62.7 (2016), pp. 4228–4265.

[6] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[7] Shengchao Liu, Dimitris Papailiopoulos, and Dimitris Achlioptas. “Bad global minima exist and sgd can reach them”. In: *arXiv preprint arXiv:1906.02613* (2019).

[8] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. “On the computational efficiency of training neural networks”. In: *arXiv preprint arXiv:1410.1141* (2014).

[9] Stefano Sarao Mannelli and Pierfrancesco Urbani. “Just a Momentum: Analytical Study of Momentum-Based Acceleration Methods in Paradigmatic High-Dimensional Non-Convex Problems”. In: *arXiv preprint arXiv:2102.11755* (2021).

[10] Marc Mézard, Giorgio Parisi, and Miguel Angel Virasoro. *Spin glass theory and beyond: An Introduction to the Replica Method and Its Applications*. Vol. 9. World Scientific Publishing Company, 1987.

[11] Francesca Mignacco, Florent Krzakala, Pierfrancesco Urbani, and Lenka Zdeborová. "Dynamical mean-field theory for stochastic gradient descent in Gaussian mixture classification". In: *arXiv preprint arXiv:2006.06098* (2020).

[12] Francesca Mignacco, Pierfrancesco Urbani, and Lenka Zdeborová. "Stochasticity helps to navigate rough landscapes: comparing gradient-descent-based algorithms in the phase retrieval problem". In: *Machine Learning: Science and Technology* (2021).

[13] Rick P Millane. "Phase retrieval in crystallography and optics". In: *JOSA A* 7.3 (1990), pp. 394–411.

[14] Hidetoshi Nishimori. "Exact results and critical properties of the Ising model with competing interactions". In: *Journal of Physics C: Solid State Physics* 13.21 (1980), p. 4071.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *arXiv preprint arXiv:1912.01703* (2019).

[16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[17] Stefano Sarao Manelli, Eric Vanden-Eijnden, and Lenka Zdeborová. "Optimization and Generalization of Shallow Neural Networks with Quadratic Activation Functions". In: *Proceeding of the 2020 Advances in Neural Information Processing Systems* 33.CONF (2020), pp. 13445–13455.

[18] Henry Schwarze and John Hertz. "Generalization in a large committee machine". In: *EPL (Europhysics Letters)* 20.4 (1992), p. 375.

[19] Sarao Mannelli Stefano, Giulio Biroli, Chiara Cammarota, Florent Krzakala, Pierfrancesco Urbani, and Lenka Zdeborová. "Complex Dynamics in Simple Neural Networks: Understanding Gradient Flow in Phase Retrieval". In: *2020 Conference on Neural Information Processing Systems-NeurIPS 2020*. 2020.

[20] David J Thouless, Philip W Anderson, and Robert G Palmer. "Solution of'solvable model of a spin glass'". In: *Philosophical Magazine* 35.3 (1977), pp. 593–601.

[21] Adriaan Walther. "The question of phase retrieval in optics". In: *Optica Acta: International Journal of Optics* 10.1 (1963), pp. 41–49.

[22] E Weinan, Chao Ma, Stephan Wojtowytsch, and Lei Wu. “Towards a mathematical understanding of neural network-based machine learning: What we know and what we don’t”. In: *arXiv preprint arXiv:2009.10713* (2020).

[23] Max Welling and Yee W Teh. “Bayesian learning via stochastic gradient Langevin dynamics”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. Citeseer. 2011, pp. 681–688.

[24] Lenka Zdeborová and Florent Krzakala. “Statistical physics of inference: Thresholds and algorithms”. In: *Advances in Physics* 65.5 (2016), pp. 453–552.

[25] Lenka Zdeborová and Florent Krząkała. “Phase transitions in the coloring of random graphs”. In: *Physical Review E* 76.3 (2007), p. 031131.

[26] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. “Understanding deep learning requires rethinking generalization”. In: *arXiv preprint arXiv:1611.03530* (2016).