A Study of Gradient-Based Algorithms

Rasmus Hallén

02 - 03 - 2017

Abstract

Gradient-based algorithms are popular when solving unconstrained optimization problems. By exploiting knowledge of the gradient of the objective function to optimize, each iteration of a gradient-based algorithm aims at approaching the minimizer of said function.

In the age of web-scale prediction problems, many venerable algorithms may encounter difficulties. However, as the data sets grow larger, so does our capability to interpret them. Thanks to developments in the computer-science subfield called Machine Learning, it is possible to produce self-adapting optimizers able to deal with different challenging scenarios, without requiring the user to rewrite algorithms specialized for the experiment at hand.

This thesis shall compare the performance of two different gradient-based algorithms; Gradient Descent (GD) and Stochastic Gradient Descent (SGD). Firstly, a synthetic linear regression model is considered. Performance of the algorithms are evaluated by comparison with the maximum likelihood estimator (MLE). Thereafter, a convergence study is presented. Lastly, SGD is evaluated over increasingly large, random subsets of the data. Such a subset is known as a mini-batch.

When transitioning to multinomial logistic regression setting, a model to recognize handwritten digits is constructed. The data is provided by Mixed National Institute of Standards and Technology (MNIST) database. Each digit is represented by a vector with pixel weights as entires. The algorithms are assessed by how well they estimate a model to predict which vector belongs to which number.

In the case of linear regression, it is found that SGD outperform GD as the data sets grow larger. Furthermore, the precision of SGD does not necessarily improve when increasing the mini-batch size. For multinomial logistic regression, GD produces the most accurate model with a 73.39% success rate. However, the training of the model is time consuming and by partitioning the data into mini-batches, SGD achieves 70.49% success rate in a more resonable timeframe.

Contents

1 Introduction

1.1 Background

In 1959 Arthur Samuel defined Machine learning as a subfield of computer science that "gives computers the ability to learn without being explicitly programmed" [1]. It refers to the automated identification of patterns in data. As such it has been a fertile ground for new statistical and algorithmic developments. The interweaving of statistics and machine learning have gradually become more apparent as computing power has increased. In 1959, a large dataset might have consisted of only a few bytes. However, computers today handle significantly larger quantities of information.

Despite the convergence of the two fields, machine learning and statistics remain separate. In this thesis, the terminological differences are highlighted in the framework section and will be used interchangeably.
Whether working with statistical models or learning machines new algorithms, the problem of optimization is of importance. That is, given a task, how to choose a model that completes said task in the most optimal way. Of course, a great range of problems fit this description. This thesis focus on solving an unconstrained minimization problem, which is to find parameters that minimize some function [2].

Naturally, this is not an unfamiliar problem for any statistician or computer scientist. The solution methods vary with the complexity of the problem, but one of the most popular approach is to utilize the gradient of a function. Such methods are artlessly named gradient methods.

Two popular algorithms for gradient approximation of a function are

1. Gradient Descent [3].
2. Stochastic Gradient Descent [4].

As with many algorithms, these contain inputs that are selected by the user. Such input parameters are referred to as hyper parameters. How to select the hyper parameters is not trivial and poor choices may result in the divergence of the algorithms. This thesis will shed light on the interaction of the hyper parameters for some common optimization problems.

1.2 Motivation and research question

With databases of petabyte size the need for accurate algorithms is considerable. If the hyper parameters are not well chosen, the resulting model may not have any predictive capabilities. As such, it is necessary to provide information on how context and data impact these choices.

The extent of modern datasets requires study of the different methods used for organizing them. To randomly select a subset of the data, and instead iterating over this set is known as mini batching. The size of the mini batch need to be manually tuned and faulty sizes may cause misleading results. Therefore, knowledge on how these sizes affect performance is important.

Both Gradient Descent and Stochastic Gradient Descent require tweaking of hyper parameters to run smoothly. This thesis aim to answer:

- *What are appropriate values of these hyper parameters given different datasets?*
- *Is it possible to minimize the amount of time the algorithms spend looking for solutions by varying the mini-batch size?*
- *Are the results consistent when moving from an OLS setting to multinomial logistic regression?*

1.3 Goals

Both statistical estimation and machines learning considers the problem of minimizing a function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} C_i(\theta) \tag{1}$$

Using different gradient methods, the goal is to compute the solution to this problem. Furthermore, to analyze how varying the hyper parameters affect the performance of the algorithms used to find this solution. Specifically, how choosing different subsets of the data affect the precision of the solution. Additionally, the aim is to provide insight whether the results are consistent when transitioning from a linear regression to a classification setting.
Lastly, the analysis shall be conducted in a statistically robust way and applicable on large scale datasets.

2 Theory

2.1 Framework

Machine learning and statistics share important features, but use a somewhat different terminology. In statistics, linear regression is the task of modeling a relationship between a dependent variable Y and a dataset X. In a machine learning context, this is called a predictive or supervised learning problem. The set $\mathbf{\Omega} = \{(x_i, y_i)\}_{i=1}^m$ is called the training set, a single vector value pair x_i, y_i is called a training example and m is the number of training examples [5].
Each $x_i \in \mathbf{\Omega}$ is a row vector of dimension p, often referred to as the predictor in statistical literature. Interchangeable terms are attributes, features or covariates. The outputs $y_i \in \mathbf{\Omega}$ are called the responses, or more classically the dependent variables.
The purpose of a regression model is to construct prediction rules given input data. Let $\mathbf{X} = (X_1, X_2, ..., X_p)$ where $x_{k,i} \in \mathbf{X}_k$ and $i = 1, ..., m$. Given the input matrix $\mathbf{X}$, since the problem is linear, the output $\mathbf{Y}$ is predicted via the model

$$\hat{\mathbf{Y}} = \mathbf{X}\hat{\boldsymbol{\theta}} \tag{2}$$

A linear model is given by picking the coefficients θ that minimize the residual sum of squares [6]. Which is equivalent to solving the unconstrained optimization problem with respect to the vector $\boldsymbol{\theta}$

$$\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = (\mathbf{Y} - \mathbf{X}\boldsymbol{\theta})^{\mathbf{T}}(\mathbf{Y} - \mathbf{X}\boldsymbol{\theta}) \tag{3}$$

Denote the solution of (3) by $\hat{\boldsymbol{\theta}}$.
For the simulation studies, a synthetic model is constructed for the data generation.

$$\mathbf{Y} = \mathbf{X}^{\mathbf{T}}\boldsymbol{\theta} + \boldsymbol{\epsilon} \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2\mathbf{I}_m) \tag{4}$$

If J denotes the number of simulations, then

$$\mathbf{\Omega_j} = \{\mathbf{X_j}, \mathbf{Y_j}\} \text{ where } j = 1, ..., J. \tag{5}$$

The goal of minimizing (1) translates to estimating the vector $\hat{\boldsymbol{\theta}}$ as close as possible to the known parameter values $\boldsymbol{\theta}$ of the model in (3).
Two measurements that compare a computed estimate $\hat{\boldsymbol{\theta}}$ with the true vector $\boldsymbol{\theta}$ are the root mean square error (RMSE) and the mean absolute bias (MAE). These measurements will be used over a number of simulations J to evaluate performance. Note that $bias$ and $RMSE$ are calculated for a specific element in the vector $\boldsymbol{\theta}$. The notation $RMSE_d$, respectively $bias_d$ will emphasize which parameter is referred. For parameter $d = 1, .., p$, consider

$$RMSE_d = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(\hat{\theta}_j^{(d)} - \theta^{*(d)})^2} \tag{6}$$

And

$$bias_d = \frac{1}{J}\sum_{j=1}^{J} |\hat{\theta}_j^{(d)} - \theta^{(d)}| \tag{7}$$

Consequently, for each simulation, p values of $RMSE$ and $bias$ are presented. By the introduction of a hypothesis h to be determined given some input data $\mathbf{X}$, the linear regression problem in (3) may be rewritten as

$$\min_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(\mathbf{X}) - \mathbf{Y}_i)^2 \tag{8}$$

One way of solving this problem is to utilize gradient methods. These are algorithms that incorporate the gradient of differentiable functions and like any algorithms their efficiency can be monitored. Primarily, the efficiency of an algorithm depends on how accurate of an estimate it produces. Secondly, its ability to produce such an estimate within a feasible timeframe.

Gradient methods produces sequences of numbers $\{\theta_k\}$. To be efficient these sequences should converge to the optimal value. In this thesis, for some small δ, an algorithms efficiency is evaluated by measuring the duration to produce an estimate $\hat{\boldsymbol{\theta}}$ within δ precision

$$|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}| < \delta \tag{9}$$

Instead of a continuous Y, the response could be binary, i.e. $Y \in \{0, 1\}$. In statistics, this is a logistic regression (logit) model. Given certain predictors, it is indeed a supervised learning problem as well. The premises for solving a logit problem are similar to that of a continuous one. Consider the function

$$J(\boldsymbol{\theta}) = -\frac{1}{m}[\sum_{i}^{m} \mathbf{Y}_i \log h_{\boldsymbol{\theta}}(\mathbf{X}) + (1 - \mathbf{Y}_i)\log(1 - h_{\boldsymbol{\theta}}(\mathbf{X}))] \tag{10}$$

Where

$$h_{\boldsymbol{\theta}}(\mathbf{X}) = \frac{1}{1 + e^{-\theta^T \mathbf{X}}} \tag{11}$$

This is a convex and smooth function. Therefore, the solution process is similar to that of (8). But here, the dependent variable Y is categorical. If Y have more than two possible outcomes, the model is called multinomial logistic regression.

2.2 Gradient descent (GD)

Computing the solution of the optimization objective is achieved by minimizing the cost function (8). Its gradient, denoted by ∇C, reveals in which direction the function decreases the fastest. Utilizing ∇C it is possible to repeatedly step towards a local minimum.
The learning rate α is a hyper parameter and is manually tuned. There are methods that approximates a learning rate [7] speeding up convergence, but here Gradient descent is kept in a pure form. When α is chosen far from its optimal value α^* the algorithm will diverge if $\alpha > \alpha^*$and converge very slowly if $\alpha < \alpha^*$ [8].

Recall the optimization objective $C(\boldsymbol{\theta})$ in (8). Let

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} \tag{12}$$

If the number of training examples is m, the number of parameters $j = 1, ..., p$ and $\mathbf{e}_1, ..., \mathbf{e}_p$ is the standard basis, then

$$\nabla C(\boldsymbol{\theta}) = \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_1}\mathbf{e}_1 + ... + \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_p}\mathbf{e}_p \tag{13}$$

Where

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})\mathbf{x}_j^{(i)} \tag{14}$$

The pseudocode for gradient descent is presented in algorithm 1. The initial $\boldsymbol{\theta}_0$ is chosen arbitrarily.

Algorithm 1 Gradient descent

1: Initialize $\boldsymbol{\theta}_0$
2: Set iterations L
3: Set learning rate α
4: **for** k = 0 to L **do**
5: Evaluate $g_k = \nabla C(\boldsymbol{\theta}_k)$
6: $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha g_k$
7: **end for**

Convergence of gradient descent is dependent on choosing an accurate learning rate, as well as enough iterations. But given a convex optimization problem as in (8) and appropriate hyper parameters, i.e. $0 < \alpha < 1$, the algorithm will converge [9].

In (10), which is the optimization objective for logistic regression, another cost function is defined. Given the structure of gradient descent, it is apparent why J has been defined in such a way.

Consider

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^{(i)}(h_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \tag{15}$$

The gradient of this function can be computed as in (13).

Furthermore, since J is a convex, gradient based algorithms can be applied to minimize (10). Formally, this is a logistic regression problem and the model in section 4 will be weighted using (15).

2.3 Stochastic gradient descent (SGD)

Another algorithm for solving the optimization problem is the stochastic version of gradient descent. Instead of having to evaluate the sum of all training examples when computing the gradient, only one training example is used. To clarify,

$$\frac{\partial C_i(\boldsymbol{\theta})}{\partial \theta_j} = (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})\mathbf{x}_j^{(i)} \tag{16}$$

Where the i:th training example is randomly chosen in the dataset. In this thesis, one epoch of stochastic gradient descent is one run through the entire dataset. Therefore, the loop will initialize at 1 and finish when there are no more training examples to compute.

Algorithm 2 Stochastic gradient descent

```
Initialize θ_0
Set number of epochs, L
Set learning rate α
for k = 1 to L do
    θ_0 ← θ_m
    for i = 0 to m do
        Evaluate g_k = ∇C_i(θ_k)
        θ_{k+1} ← θ_k − α g_k
    end for
end for
```

Note that in algorithm 2, m denotes the number of training examples. Consequently, one epoch will run through all these training examples.
Given the same premises as gradient descent, SGD may also be applied to solve a logistic regression problem.

2.4 Mini-batch stochastic gradient descent

A method that sometimes achieves better performance than both SGD and GD is to run the algorithm over portions of the data. Such a portion is referred to as

a mini batch, whereas the whole training set is called batch. These mini-batches are generated by randomly selecting row vectors from the training set $\mathbf{X}$.

Algorithm 3 Mini-batch stochastic gradient descent

Initialize $\boldsymbol{\theta}_0$
Set number of epochs, L
Set learning rate α
for k = 1 to L **do**
    Randomly pick a mini batch of size n.
    $\boldsymbol{\theta_0} \leftarrow \boldsymbol{\theta}_n$
    for i = 0 to n **do**
        Evaluate $g_k = \nabla C_i(\boldsymbol{\theta}_k)$
        $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \alpha g_k$
    end for
end for

In algorithm 3, the mini-batches are of dimension (nxp). Since $n < m$, one epoch will complete faster that the algorithms batch counterparts.
Intuitively, the algorithm converges if SGD converges, but there exist more rigorous study of the convergence rate of mini-batch SGD [10].

2.5 Outline

To evaluate how the algorithms preform in different contexts a simulation study in a linear regression setting is conducted. By varying the complexity of the models, the algorithms may be evaluated by analyzing the parameter estimates individually by (7) and (8), but also the algorithms total cost (3).
The average of bias and RMSE over all simulations are presented in tables. The cost of the final estimate of each simulation is plotted. Also the evaluation time of the algorithms is presented in the results section.

Stochastic gradient descent is random by design. As such, to understand how its estimates are affected by this, an investigation of the convergence is conducted. By choosing a small δ in (9), the randomness of the algorithm is evaluated.

As gradient descent loops through the entire dataset every iteration, it does not scale very well with an increase in data. Given a larger data set, larger mini batches should be used. Therefore, when comparing the mini batches, only stochastic gradient descent is considered.

MNIST (Mixed National Institute of Standards and Technology) is a large database of handwritten digits [11]. This is a common dataset used for comparing and learning machine learning techniques. When transitioning to a multinomial logistic regression setting, this dataset will be used to evaluate the algorithms. Instead of comparing individual thetas (the model contains 741 of

them), the studies are conducted with respect to a test set provided by MNIST. The entire dataset consists of 60 000 images (training set), and the test set contains 10 000. First the algorithms will weigh (estimate) the parameters by looping through the training set. Then, the model is put to the test by classifying the remaining 10 000 images.

First, the results of gradient descent is presented. The resulting estimators are equal over all simulations if the hyper parameters are kept constant. Plots of two different step sizes are presented.
When investigating stochastic gradient descent, an analysis of the step size and how it affects performance is followed by a cost analysis when varying the mini batch sizes.

In the case of multinomial logistic regression, the success rate of GD and and mini-batch SGD are presented in tables. Also an analysis of how different step sizes may cause difficulties is given.

3 Simulation studies for linear models

3.1 Model description

The study covers the performance of different gradient algorithms over varying data sizes. Simulations are conducted by randomly generating data from a known correlation with added noise. Three models are used throughout the simulations follow the linear relation specified in (4). These synthetic models are

(i) $\mathbf{X}$ is a $200x4$ matrix with entries uniformly drawn from $(0, 1)$. The noise follows standard normal distribution.

(ii) $\mathbf{X}$ is a $20000x7$ matrix with entries uniformly drawn from $(0, 1)$. The noise again follows the standard normal distribution.

(iii) $\mathbf{X}$ is a $2000000x14$ matrix with entries uniformly drawn from $(0, 1)$. The noise is the same as above.

The starting value for each element of the vector parameter θ has been independently drawn from a uniform distribution on (0,10). Therefore, the first iterations tend to heavily improve the estimation $\hat{\boldsymbol{\theta}}$.
Throughout the batch simulations, Stochastic Gradient Descent will run only one epoch, while Gradient Descent will run 100 epochs. If 100 epochs is not computationally feasible, the batch size of GD will be chosen to match the speed of SGD.

When working with model (iii), the amount of iterations have been reduced so that the evaluationtime is roughly the same for the two algorithms. The computer used for the simulations is a macbook air with a 1.8 gHz Intel Core i5 processor and 4 GB of ram.

The algorithms are evaluated over 30 simulations by comparing the root mean squared error (7), the mean absolute error (8) and then an analysis of convergence is presented. The evaluation time of the algorithms are also presented. The comparisons are conducted with a gold estimate, θ^* obtained by Matlabs backslash operator. Which in the case of linear regression is the same as the maximum likelihood estimator (MLE).

Following the simulations is an investigation of convergence. Here, the stochasticity of SGD is demonstrated.
In the experiments, average time to convergence is measured, given some threshhold δ.

Lastly, the datasets are distributed into random mini-batches of size L. Using the average of multiple runs of stochastic gradient descent its performance is evaluated over these mini-batches. Because GD scales poorly with larger

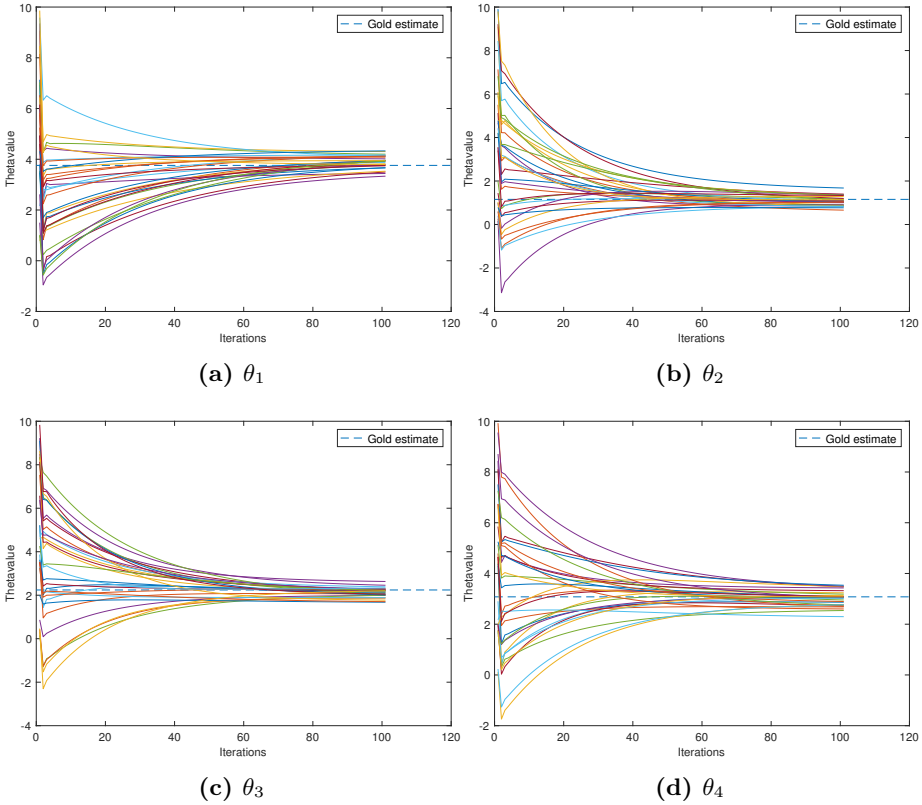
datasets [12], and as the data grows, so does the mini batches necessary to produce a fair estimate. Hence, the data cap is higher for SGD and more interesting to investigate.

3.2 Results

The main result of the simulation studies is that the performance of stochastic gradient descent scales well with an increase in data. Furthermore, the precision does not markedly improve when increasing iterations, which is a reason for partitioning the data.

3.2.1 Comparison of RMSE, bias and cost.

For model (i) in table 1, one run of stochastic gradient descent does not produce a valuable estimate. However, gradient descent does indeed preform well. Of course, the RMSE and bias could be reduced further by increasing the amount of iterations. Table 2 displays the results of the simulations.

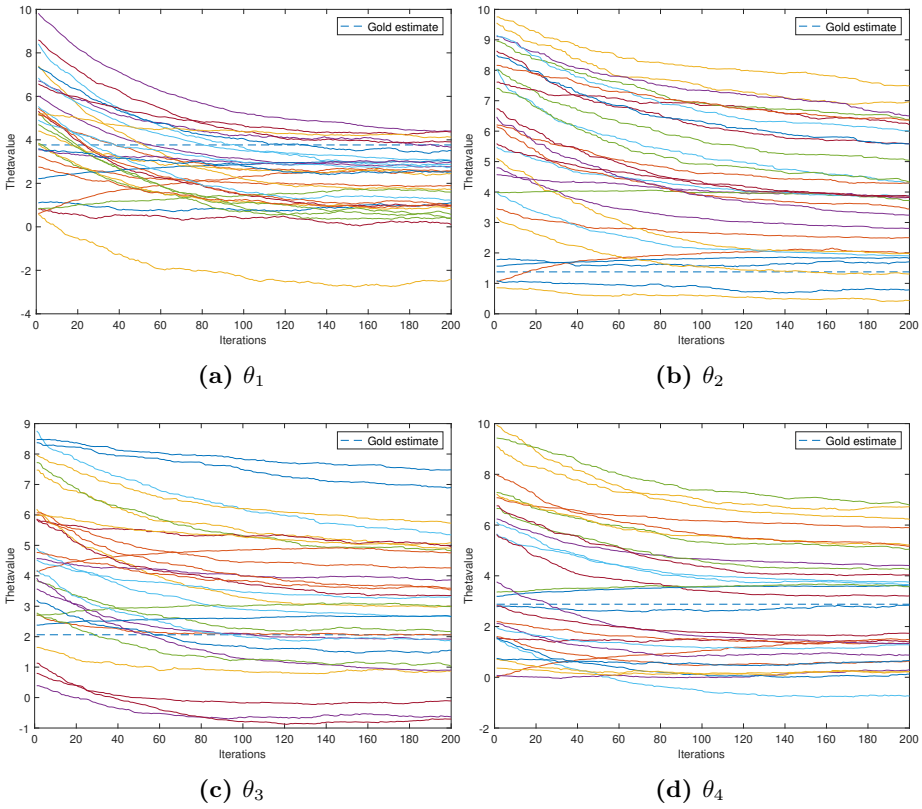


(a) θ_1 **(b)** θ_2

(c) θ_3 **(d)** θ_4

Figure 1: Convergence plot for each theta in the first model (gradient descent).

Figure 1 displays how the thetas are approaching the gold estimate per iteration. Note that first few iterations change the precision of the approximations quite a lot. This is because the initial guess is very far from the true values.



(a) θ_1 **(b)** θ_2

(c) θ_3 **(d)** θ_4

Figure 2: Convergence plot for each theta in model (i), Stochastic gradient descent. It is clear that this approximation is not as well behaved as the thetas in figure 1.

Comparing the convergence of the thetas in model (i) it is clear that Gradient descent works better.

	RMSE	bias
θ_1	0.2680	0.2243
θ_2	0.2998	0.2530
θ_3	0.3530	0.3020
θ_4	0.2394	0.2004
Average	0.2901	0.2449

(a) Gradient descent

	RMSE	bias
θ_1	2.6799	2.2765
θ_2	3.1866	2.5035
θ_3	2.7812	2.4122
θ_4	2.0638	1.6841
Average	2.6779	2.2191

(b) Stochastic gradient descent

Table 1: Comparison for model (i).

In table 1, we can clearly see how GD outperform the stochastic version in terms of precision. The thetas are not near the gold estimate. Hence, the amount of data points provided by model (i) is not enough for SGD to provide a valid estimate. But as the data increase so does the reliability of SGD.

	RMSE	bias
θ_1	0.8986	0.7351
θ_2	0.3404	0.2822
θ_3	0.3098	0.2568
θ_4	0.2811	0.2328
θ_5	0.3084	0.2570
θ_6	0.2856	0.2260
θ_7	0.2934	0.2291
Average	0.3882	0.3170

(a) Gradient descent

	RMSE	bias
θ_1	0.0779	0.0662
θ_2	0.0659	0.0534
θ_3	0.0943	0.0806
θ_4	0.0499	0.0385
θ_5	0.0676	0.0534
θ_6	0.0796	0.0658
θ_7	0.0639	0.0569
Average	0.0713	0.0592

(b) Stochastic gradient descent

Table 2: Comparison for model (ii).

As the data size increases, the precision of SGD is heightened. For model (ii), SGD outperforms GD as shown in table 2.

	RMSE	bias
θ_1	3.4799	2.9844
θ_2	2.1227	1.8957
θ_3	1.5178	1.3441
θ_4	1.2883	1.1263
θ_5	1.7536	1.3983
θ_6	1.5374	1.3414
θ_7	1.4010	1.2269
θ_8	2.0179	1.7741
θ_9	1.5522	1.3789
θ_{10}	1.2556	0.9804
θ_{11}	1.7202	1.4324
θ_{12}	1.5264	1.2604
θ_{13}	1.3458	1.2138
Average	1.7322	1.4890

(a) Gradient descent

	RMSE	bias
θ_1	0.0222	0.0174
θ_2	0.0228	0.0189
θ_3	0.0260	0.0218
θ_4	0.0225	0.0179
θ_5	0.0235	0.0185
θ_6	0.0205	0.0160
θ_7	0.0244	0.0180
θ_8	0.0279	0.0235
θ_9	0.0308	0.0236
θ_{10}	0.0256	0.0209
θ_{11}	0.0234	0.0197
θ_{12}	0.0289	0.0213
θ_{13}	0.0216	0.0176
Average	0.0246	0.0196

(b) Stochastic gradient descent

Table 3: Comparison for model (iii).

For model (iii), gradient descent performs even worse. In table 3, GD runs for 40 iterations and is not able to produce an accurate estimate in a viable timeframe.

	RMSE	bias	α
Model (i)	0.2901	0.2449	0.6
Model (ii)	0.3882	0.3170	0.3
Model (iii)	1.7322	1.4890	0.1

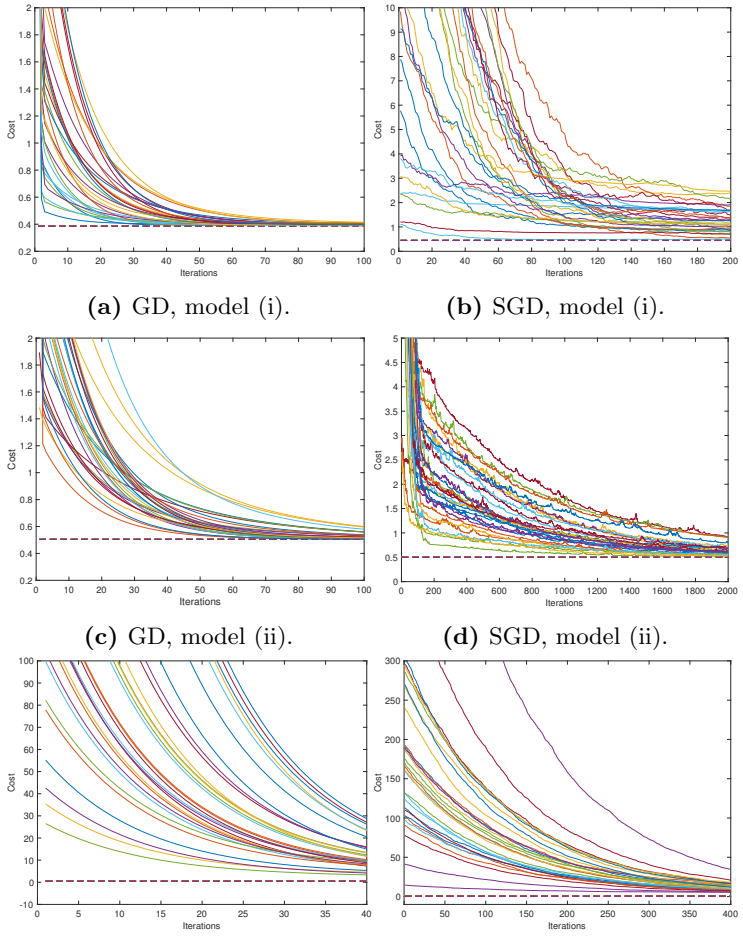
(a) Gradient descent

	RMSE	bias	α
Model (i)	2.6779	2.2191	0.1
Model (ii)	0.0713	0.0592	0.05
Model (iii)	0.0246	0.0196	0.01

(b) Stochastic gradient descent

Table 4: Comparison of average result after 30 simulations.

It is clear that gradient descent is outmatched by its stochastic counterpart when the data sets are sufficiently large. Looking at not only individual thetas, but at total cost confirms those results.



(a) GD, model (i).

(b) SGD, model (i).

(c) GD, model (ii).

(d) SGD, model (ii).

(e) GD, model (iii).

(f) SGD, model (iii).

Figure 3: Total cost for the estimates produced every iteration for model (i) - (iii).

	Time (s)
Model (i)	0.0016
Model (ii)	0.0492
Model (iii)	5.4512

(a) Gradient descent

	Times (s)
Model (i)	0.0007
Model (ii)	0.0625
Model (iii)	6.5890

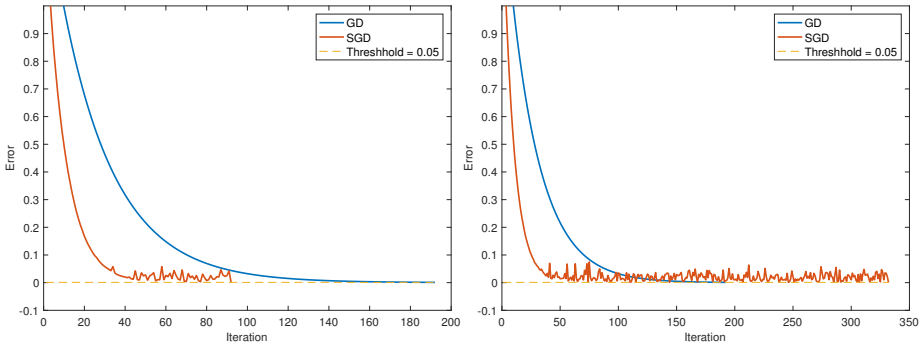
(b) Stochastic gradient descent

Table 5: Comparison of running time.

Figure 3 shows how the cost (3) of the algorithms is increasingly more difficult to minimize as the data set grows. Note the scale of the axes for the latter models.
As stated, the evaluation time of the simulations is roughly the same (as seen in table 5). But for computational purposes, only the first 400 iterations are shown for SGD in figure 3 (f). Hence, the cost is considerably lower than displayed as indicated by the study of individual parameters (table 3).
In figure 3, the α used to produce the plots are displayed in table 4.

3.2.2 Rate of convergence

The results in section 3.1 covers the precision of the algorithms, but that is not the whole story. It is possible to reshuffle the data and run stochastic gradient descent another epoch. With an appropriate learning rate, it may theoretically reach machine precision in less iterations than its non-stochastic counterpart, even with smaller datasets.
In figure 4, a 3 parameter model with 2000 data points is used to evaluate convergence, stopping value is set to $\delta = 0.001$. It is a display of the stochasticity of the algorithm.



(a) SGD 90 iterations

(b) SGD 332 iterations

Figure 4: Two different runs on the same dataset, GD converges in 192 iterations.

As mentioned in section 3.1, SGD scales better than GD when the datasets grow larger. When $n = 2000$, SGD have to be restarted after every epoch (with the

new parameter values) to find the minimum. If the new estimate keeps missing the thetas that minimize the cost, the algorithm runs for a longer time.

figure	GD	SGD
(a)	0.0082	1.0253
(b)	0.0137	4.0625

Table 6: Execution time (s) of the algorithms in figure 4.

In the following figure, the x-axis is the number of iterations to convergence, where each circle is a stopping point for stochastic gradient descent. The batch counterpart is the blue line, steadily approaching convergence at 192 iterations.

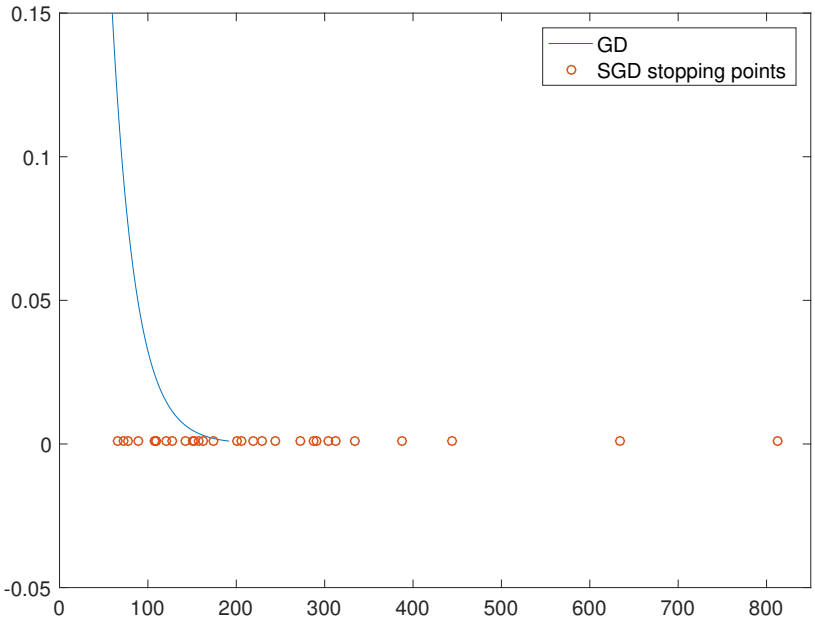


Figure 5: Convergence behavior of SGD.

It is difficult to foresee the convergence behavior of SGD as indicated by the spread of the stopping points in figure 5. Here, SGD runs 30 times on the same dataset, its mean value for number of iterations is 233.4 until convergence. Execution times of the algorithms vary per iteration, their means are displayed in table 7.

	GD	SGD
Time	0.0069	2.3156

Table 7: Execution time (s) over 30 simulations.

3.2.3 Mini-batch SGD

It is time to investigate what happens with SGD when the sizes of the mini-batches are varied.

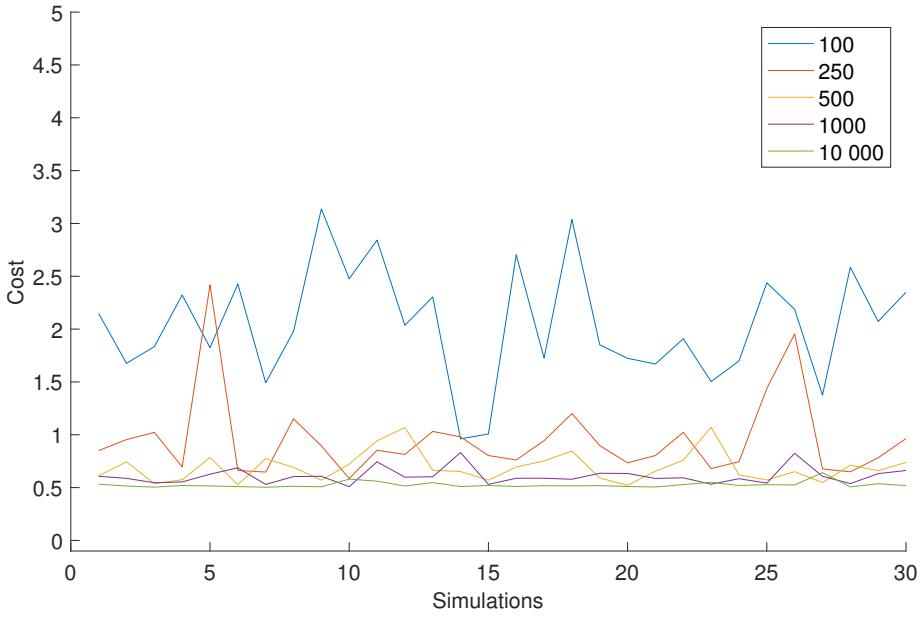


Figure 6: One epoch of SGD over varying mini-batch sizes. Model (iii) is used for the simulations.

The cost is steadily high with a mini-batch of 100 (except simulation number 14 and 15). However, when increasing the mini-batch size, figure 6 shows that the variance is reduced.
A mini batch size of 250 still yields some awkward simulations, but at 500, the cost is very close to that of 1000 and 10 000.

4 Logistic regression on the MNIST dataset

4.1 Introduction

MNIST is a dataset of handwritten digits, comprising 60 000 training examples and a test set of 10 000 training examples. Each vector in the training set is a flatned 28x28 matrix, where each matrix entry is a pixel intensity.

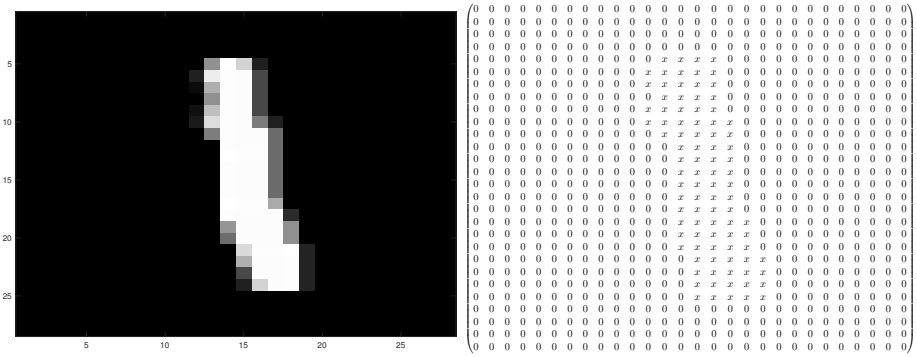


Figure 7: The digit 1 is converted to a matrix with pixel densities x as the nonzero entries.

Given the vectors $\mathbf{x}^{(i)} \in \mathbf{X}$, where $\mathbf{X} \in \mathbf{\Omega}$, each $\mathbf{x}^{(i)}$ is simply the concatenated rows of the matrix in figure 10. Let $\mathbf{y} = \{0, ..., 9\}$, $y^{(i)} \in \mathbf{y}$ and $\boldsymbol{\theta}$ a matrix of dimension $784x10$ representing how the pixels are weighted in the model. In eq. (10), the cost function for a binary problem is presented, however, transitioning to a multivariate setting requires some modifications.
The sought probability is $\Pr(y^{(i)} = k|x^{(i)}; \boldsymbol{\theta})$ for $k = 0, ..., 9$. Consequently, the hypothesis is the vector containing all such probabilities

$$h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \begin{pmatrix} \Pr(y^{(i)} = 0|\mathbf{x}^{(i)}; \boldsymbol{\theta}) \\ \cdots \\ \Pr(y^{(i)} = 9|\mathbf{x}^{(i)}; \boldsymbol{\theta}) \end{pmatrix} \tag{17}$$

Please note that these probabilities sum up to one. Furthermore, the probabilities may be computed using the softmax function

$$\Pr(y^{(i)} = k|\mathbf{x}^{(i)}; \boldsymbol{\theta}_k) = \frac{\exp\left(\boldsymbol{\theta}_k^T \mathbf{x}^{(i)}\right)}{\sum_{j=1}^{K} \exp(\boldsymbol{\theta}_j^T \mathbf{x}^{(i)})} \tag{18}$$

As a decision rule, the category $y^{(k)}$ corresponding to the highest probability in $h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ is chosen.
Given $y^{(i)} = j \in \{0, ..., 9\}$, the objective is to find the $\hat{\boldsymbol{\theta}}$ that maximize $\Pr(y^{(i)} = j)$. That is equivalent to solving

$$\min_{\hat{\boldsymbol{\theta}}} J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} \mathbf{I}(y^{(i)} = j) \log(\Pr(y^{(i)} = k|\mathbf{x}^{(i)}; \boldsymbol{\theta}) \tag{19}$$

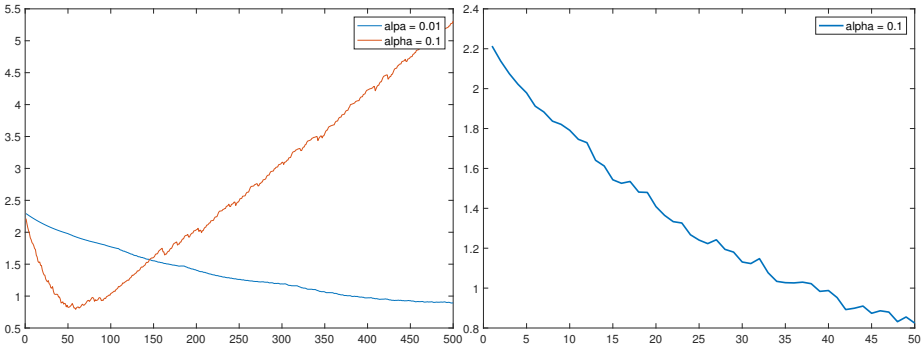
Equation 16 is convex, why gradient methods can be used to solve the problem. The gradient with respect to parameter j is denoted

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \mathbf{x}^{(i)} (\mathbf{I}(y^{(i)} = j) - \Pr(y^{(i)} = j | \mathbf{x}^{(i)}; \theta)) \tag{20}$$

As the parameters are so many, it is unfeasible to compare them individually as in section 3. Therefore, the cost function will be studied, as well as the predictive accuracy on the test set provided.

4.2 Results

In the following sections the results will be presented. The MNIST data provides a test set as well as a training set. Therefore the algorithms performances will be evaluated by, not only their ability to minimize the cost, but how well they classify the digits in the test set.
The parameters are always initialized at zero and the α are chosen to appropriate values.



(a) Changes in cost over 500 iterations **(b)** Hitting minima after 50 iterations.

Figure 8: Gradient descent on the MNIST dataset.

Figure 8 shows how Gradient Descent overshoots the solution if the learning rate is too high. In table 11, the results are sumarized.

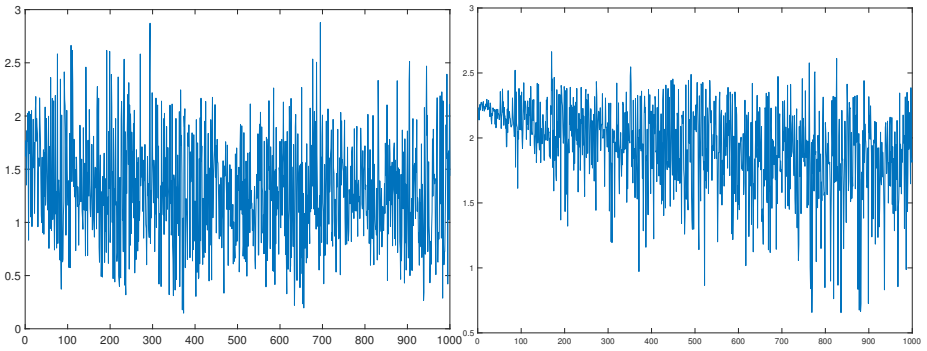
	Cost	Time
50 iterations ($\alpha = 0.1$)	0.8246	172s
500 iterations ($\alpha = 0.01$)	0.8925	1620s

Table 8: Time and cost for GD.

The learning rate has significant impact on the convergence rate of Gradient descent and the best result is with a learning rate of 0.1 running only 50 itera-

tions. Using this model, it correctly predicts 7339 of the 10 000 images in the test set. Corresponding to a 73.39% accuracy.

Stochastic gradient descent may also overshoot the solution if the α is too large. In figure 9, SGD runs one epoch ($m = 60000$), but it is computationally heavy to plot every iteration, therefore only the first 1000 are displayed. The different α correspond to a higher volatility in cost.



(a) $\alpha = 0.01$. **(b)** $\alpha = 0.001$.

Figure 9: Gradient descent on the MNIST dataset using SGD.

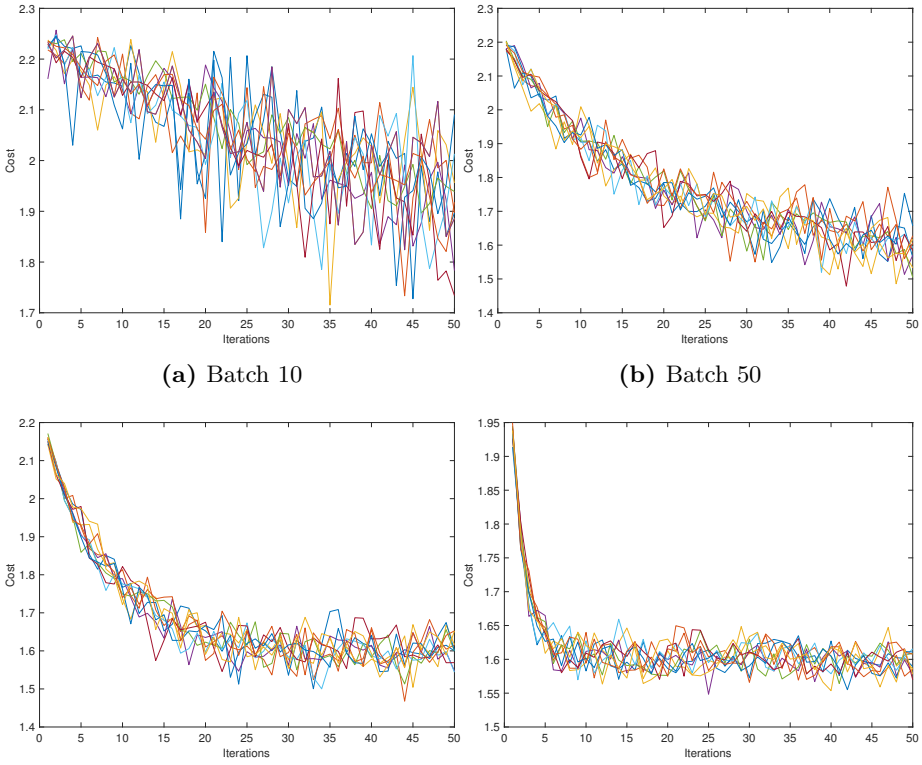
The cost is very random for stochastic gradient descent when only considering a single training example, which is to be expected. Performing the updates like so still yields results above random guessing. Different values of the step size correspond to different volatility in cost, as shown in figure 9. This is consistent with the results of linear regression.

Note that when arbitrarily picking the thetas (as the initial guess), one can expect a success rate of about 10% (or 9.8% in the case of the initial parameters). In table 11, the test score over 10 simulations are presented. The average score of $\alpha = 0.001$ is 6979 and for $\alpha = 0.01$ that number is 4284.

α	1	2	3	4	5	6	7	8	9	10
0.01	4812	3193	4545	2452	3666	5590	5145	5673	4179	3587
0.001	7357	6137	6899	7371	7466	6201	6821	7385	7513	6647

Table 9: Test score over 10 simulations.

When instead iterating numerous times over mini batches, as opposed to running through the entire set. The results do differ.



(a) Batch 10 **(b)** Batch 50

(c) Batch 100 **(d)** Batch 500

Figure 10: Cost function over varying sizes of mini batches.

After some iterations, the change in cost is unaffected of mini-batch size. This is consistent with the results from the simulations studies of linear regression. However, with a larger mini-batch size, less iterations are required.

	Average	Highest	Lowest	Avg. time per iteration
10	4981	6712	3122	0.3317s
50	6800	7410	5660	0.3111s
100	7049	7849	6534	0.3270s
500	6790	7242	6312	0.3493s

Table 10: Results of stochastic gradient over 10 simulations.

Table 12 summarize the results of the stochastic section. The best prediction was achieved with a batch size of 100, with its highest simulation hitting 78,5% accuracy on the test set.
It is interesting that increasing the batch size does not significantly impact the time per iteration. However, it must be noted that the larger the mini batch, the longer it takes to reorder the data.

5 Conclusion

5.1 Summary

This thesis shows how varying hyper parameters affect the performance of gradient based algorithms. In the first section, a comparison of gradient descent and stochastic gradient descent is presented. Here, the algorithms performance are evaluated by comparing the convergence of individual parameters, as well as the cost function (3).
Table 4 gives a comprehensive summary of the main results of section 3.2.1. It describes how stochastic gradient descent gradually outperform its non stochastic relative. If the data set is small, then there are not enough steps for SGD to find its way to the minimum. However, the opposite is true for larger data sets. In the simulations, model (ii) was large enough to produce valid parameter estimates.

The next section is dedicated to a comparison of convergence. Here, the stochasticity of SGD is demonstrated by simulating data from a three parameter model with 2000 data points. Gradient descent has no problem finding a solid estimate within a feasible timeframe given such a data set.

Stochastic gradient descent may be reinitialized after every epoch (or sometimes just converge very quickly), on average, gradient descent needs 0.4664s to produce estimates within epsilon precision. Whereas stochastic gradient descent takes on average 1.6516s to produce the same result.
As seen in table 5 the time per iteration for gradient descent takes longer when the data set grows. This is where SGD really shines. Since only one data example is necessary for a parameter update, the speed of algorithm is not as affected by an increase in data. Keep in mind, that in model (iii), gradient descent runs 40 iterations, whereas SGD runs 2 000 000 in roughly the same amount of time.

A comparison of mini batches concludes section 3. Figure 6 shows how varying sizes affect the cost in equation (3). It is interesting that after a certain point, it seems that increasing the mini batch size does not markedly affect the cost. At size 100, the estimates are not very good, which can be expected. However, they do significantly improve at size 250, but fluctuations do occur. When the mini batch size is 500, there are only smaller fluctuations in cost.
To summarize, when precision is at key, a larger mini batch is suggested. Although, at size 1000, the increase in precision may not compensate for the computational disadvantage of choosing a batch size of 10 000.

Lastly, the consistency of the results are studied by transitioning to a logistic regression setting. A model to recognize handwritten digits provided by MNIST is set up. The optimization target is equation (19) and the study is conducted as follows. First, a solid estimate is produced by gradient descent. Here, the crucial hyper parameter is the learning rate α which can make or break

the model. In figure 8, two different α are shown to have significant impact on the cost (19). With a well chosen α the model can find a good fit after about 50 iterations. A safer convergence may be achieved by reducing the size of α at the cost of time which scales linearly with number of iterations.

	Success rate	Time (s)
Gradient Descent	73.39%	172.3
Stochastic Gradient Descent	69.79%	183
Stochastic Gradient Descent (L = 10)	49.81%	3.317
Stochastic Gradient Descent (L = 50)	68.00%	15.555
Stochastic Gradient Descent (L = 100)	70.49%	32.70
Stochastic Gradient Descent (L = 500)	67.90%	174

Table 11: Success rate of gradient based algorithms on the MNIST data set. L denotes mini batch size.

In table 11, the success rate for standard GD is highest, but using SGD with a mini batch size of 50 or 100 drastically reduce the computation time. The precision of the estimators still are close to those of GD. Increasing the mini batch size from 50 has the contradictory effect of decreasing precision (in the case of L = 500).

Figure 10 displays the cost function (19) over 10 simulations with different mini batch sizes. The larger the batch size, fewer iterations are required for convergence.

5.2 Discussion

There are multiple ways to improve the efficiency of gradient based algorithms. By regularizing one can prevent the common problem of overfitting, improving the error rate for handwritten digit recognition [13]. By also introducing a convolutional neural network, the error have been reduced to 0.48% [14].
The purpose of this thesis is primarily to study how the success of the algorithms is affected by varying the hyper parameters. As well as to present a comparison of GD and SGD, which is why the algorithms have been kept in a pure form.

In a linear regression setting, stochastic gradient descent outperform gradient descent when the data set is larger than that of model (ii). If a smaller model is considered (e.g. model (i)), then gradient descent is a sound choice as the stochasticity of SGD may cause problems. Of course, it is possible to tweak the learning rate to reach valid estimations, but with a smaller data set, the simplicity of gradient descent is preferred.

When working with stochastic gradient descent and the MNIST data set a mini

batch size of 50-100 is proposed. A larger size may improve consistency, but as seen table 11, it actually increased the error rate. Of course, this result may stem from the fact that only 10 simulations are considered. Looking at the plots in figure 10, similar cost is expected over 50 iterations for batches 50-500.
An alternative to reducing mini batch size would be to reduce the amount of iterations. Figure 10 (d) seems to converge solely after 5-10 runs.

The subject of gradient based algorithms is very well studied. But it is interesting to see how the convergence and cost of the algorithms vary with different hyper parameters and model premises. There are capitalizable similarities when working in different settings, but as a whole, the choices are very data dependent. There is no universal answer that is applicable on all data. However, with an understanding of the underlying workings of the algorithms, it is possible to speed up the process of picking the hyper parameters.

5.3 Acknowledgements

I would like to thanks my supervisor Umberto Picchini for his patience and good advice. He readily answered any questions that arose during this process.

Also thanks to anyone who has contributed in the proof reading process.

References

[1] Arthur J. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210.

[2] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*, volume 7. Cambridge University Press, 2009.

[3] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[4] Leon Bottou. Large-scale machine learning with stochastic gradient descent. `http://leon.bottou.org/papers/bottou-2010`, August 2012.

[5] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2010.

[6] Robert Hastie, Trevor Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[7] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[8] Andrew Ng. Gradient descent in practice ii - learning rate. `https://www.coursera.org/learn/machine-learning/lecture/3iawu/gradient-descent-in-practice-ii-learning-rate`.

[9] Geoff Gordon and Robert Tibshirani. Gradient descent revisited. `https://www.cs.cmu.edu/~ggordon/10725-F12/slides/05-gd-revisited.pdf`.

[10] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, (13):165–202, 2012.

[11] Yann LeCun, Corinna Cortes, and Christopher Burges J.C. The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[12] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.

[13] LISA lab. Deep learning 0.1. `http://deeplearning.net/tutorial/gettingstarted.html`.

[14] Dan C. Ciresan, Ueli Meier, Jonathan Masci, M. Gambardella Luca, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. *Joint Conference on Artificial Intellegence*, 2011.