1. Introduction

The topic of *machine learning* encompasses a broad class of algorithms that use core ideas of Artificial Intelligence to mimic human decisions [9]. *Deep learning* is a subset of machine learning where models are built based on a technique that comes naturally to humans: learning by example [14]. While most machine learning techniques require direction on how to proceed with accurate prediction through the use of additional data, deep learning models are able to learn through its own method of computing [5]. There are many applications to social and biological sciences, where it has proven to be useful in image recognition, speech recognition, predicting mutations in non-coding DNA, and natural language understanding [8]. However, these high-dimensional problems often require optimization to be performed, which can be challenging using second-order methods. Gradient Descent has proven to be a powerful technique for optimization within a machine-learning setting.

Gradient Descent (GD) is an efficient method to optimize an objective function if this objective function is differentiable with respect to its parameters. It does not require second-order derivatives, so it can be less computationally complex as other optimization techniques, such as Newton-Raphson. Stochastic Gradient Descent (SGD) is often used as an efficient optimization method for stochastic objective functions, or when our objective function can be broken down into a sum of its components. This implies that there are less computational costs in SGD than if we were to use GD. In our case, we will use the fact that the log-likelihood for a dataset is simply a sum of the log-likelihood evaluated at each of its components. SGD has proven to be effective in many machine learning applications. Specifically, it has been recently used for speech research, acoustic modeling, and image recognition [1, 6, 4]. In Algorithm 1 below, we see the procedure for SGD. This is the same as GD, except in the GD the first step would be the full gradient, instead of perhaps only a certain subset of the gradient sub-functions. This is determined by the batch size, which determines how many sub-functions are included in our calculation of the gradient. In Algorithm 1, η refers to the learning rate.

Algorithm 1 Stochastic Gradient Descent

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$ (gradient wrt stochastic objective)
2: $\theta_t \leftarrow \theta_{t-1} - \eta g_t$

There are several computational challenges when using SGD. First, the step size can be difficult to tune, depending on our data and our objective function. If the step sizes are too small, the algorithm may take a long time to converge, and if it is too large, then our algorithm will not be stable as we can overshoot the optimum. Also, the main disadvantage of SGD, which we will address in this project, is that SGD is not always effective for higher-dimensional parameter spaces.

In this project, we will explore many improvements on SGD as stated above, all of which are supposed to address both the difficult choice of tuning parameter and the inefficiency of SGD in high-dimensional parameter spaces. First, we will introduce the idea of momentum, also referred to as classical momentum, which was developed many decades ago to speed up convergence of GD and SGD algorithms. Then, we will introduce the idea of Nesterov's Accelerated Gradient (NAG), which is a version of momentum but it is shown to improve the speed of convergence. Next, we will briefly introduce the idea of L_2 norm methods for gradient descent, and will explain the idea of Adam (adaptive moment estimation), which is a combination of L_2 norm methods and classical momentum. Next, we will introduce the idea of Nadam (Adam with NAG momentum instead of classical momentum). Lastly, we will investigate if the bias correction that takes place in Adam and Nadam, which will be discussed in detail later, is necessary to improve convergence. We will apply our analysis to two datasets in the logistic regression framework.

2. Methods

In this section, we will give a description of the progression in the development of momentum-based optimization algorithms. We will concentrate on the idea of momentum, and will briefly mention L_2 norm methods. In total, we will present our seven algorithms that will be used in our analysis stage. To be clear, all of the ideas here could be applied to either Gradient Descent or Stochastic Gradient Descent. We choose to investigate SGD rather than GD because of the advantage of less computational cost, which is advantageous for many machine learning applications. Details of the values chosen for the tuning parameters, $\epsilon, \mu,$ and ν, as well as the learning rate η, are given in Section Appendix a).

a) Momentum

Momentum is an idea that was developed early in optimization literature to improve the convergence of GD and SGD algorithms [11]. As seen in comparing Algorithm 1 with Algorithm 2, classical momentum incorporates a decaying sum of the previous gradients. Then, the momentum vector is said to be **m**, which is used instead of the gradient. The decay rate is determined by the constant μ. In other words, momentum gives SGD a short-term memory. The learning rate in all of the following algorithms will be η.

Algorithm 1 SGD

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $\theta_t \leftarrow \theta_{t-1} - \eta g_t$

Algorithm 2 Classical Momentum

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

2: $m_t \leftarrow \mu m_{t-1} + g_t$

3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

There are many advantages to using momentum in our algorithms. First, momentum is proven to speed the rate of convergence when compared to normal SGD [12]. Second, it prevents the algorithm from getting stuck in certain points such as local minima and it accelerates convergence in the case of a ravine. An example of the first case can be seen in Figure 1. With GD, in Figure 1b, the algorithm is not able to locate the global minimum, but with momentum, the algorithm is able to move from point D to E, and then converges to the global minimum [13]. If there is a local minimum, SGD may continue to oscillate in the local minimum where momentum will allow the algorithm to move past this local minimum because of the slope of the gradient on the other side of the local minimum is much steeper. Another example of the benefit of using classical momentum comes in the example of a ravine. If the surface of the objective function is a ravine, normal SGD will oscillate many times between the walls of the ravines, where momentum will allow the algorithm to move quickly to the minimum.

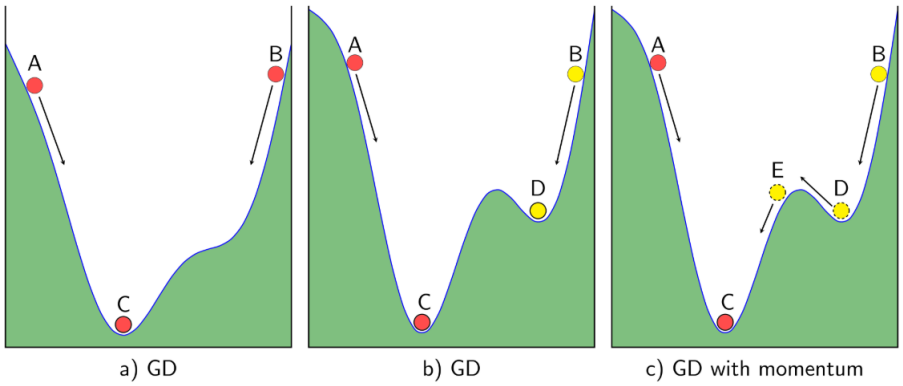

Figure 1: GD with momentum, [13]

As mentioned above, classical momentum has been shown to speed up convergence compared to normal SGD. However, recent advancements have led to the development of another version of momentum which further speeds up convergence in certain situations. Specifically, Nesterov's Accelerated Descent (NAG) has a better convergence guaranteed compared to classical momentum [12, 10]. We also note that NAG can also be written

in the same form as classical momentum. Steps 1 and 2 are the same as in classical momentum, but we add the step $\bar{m}_t \leftarrow g_t + \mu m_t$ after step 2, and the update is $\theta_t \leftarrow \theta_{t-1} - \eta \bar{m}_t$. This will be useful later when we add NAG into the adaptive moment estimation algorithm (Nadam), and we will refer to this notation as the "$\bar{m}$" notation.

Algorithm 2 Classical Momentum (CM)

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + g_t$
3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

Algorithm 3 Nesterov's Accelerated Gradient (NAG)

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta \mu m_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + g_t$
3: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$

In the paper "On the importance of initialization and momentum in deep learning", the authors give an example of an advantage of using NAG, as opposed to classical momentum. In Figure 2, we see a figure similar to the one presented in the paper that was adapted for our purposes. This figure shows that if our decaying sum of momentum μm_{t-1} is quite large, then if we were to use NAG, instead of classical momentum, our update would be more conservative, and is closer to our last estimate of θ, θ_{t-1}. This creates a more stable algorithm, and is therefore able to speed up the algorithm that does not overshoot the minimum. This is much like the choice in learning rate, where we do not want it to be so large that it goes beyond our optimal point.

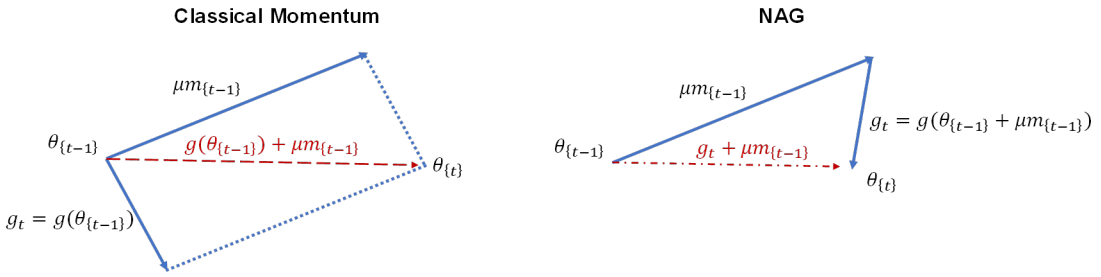

Figure 2: Advantage of NAG

b) Adam and Nadam

Next, we will discuss the Adam algorithm, which originates from the phrase **ada**ptive **m**oment estimation. We will also discuss Adam with NAG (Nadam), which is an extension of the Adam algorithm meant to improve the incorporation of momentum. Adam is an efficient method for optimization for stochastic objectives with high-dimensional parameter spaces, which is the main computational challenge we are addressing in SGD. It is used in many deep learning applications such as deep adversarial networks, image generation, and image-to-image translation. Lastly, it is very popular in classification problems.

First, we begin with a brief description of L_2 norm methods, to motivate the use of the Adam algorithm. However, the focus of this paper is momentum, not L_2 norm methods. There are many algorithms that have been developed recently to advance L_2 norm methods, but Adam is considered to be one of the most advanced methods. In general, L_2 norm methods allow the algorithm to slow down learning along dimensions that have already changed significantly and speeds up along dimensions that have only changed slightly. Other recent algorithms developed before Adam include AdaGrad and RMSProp. These two algorithms are included in the appendix, in Section Appendix b), as motivation to why Adam was developed.

In Algorithm 4, we see the procedure for the Adam algorithm. We will walk through each step of the algorithm. In Step 1, we see the evaluation of the gradient at the previous estimate of our parameters. In Step 2, there is a weighted average of the sum of the previous weighted averaged gradients, which is incorporating classical momentum. Step 3 is one of two bias correction steps, which will be discussed in detail later. In Step 4, the squared gradient is the part that is developed through the series of advances in L_2 norm methods. Step 5 is the second bias correction step and step 6 is where we evaluate our next estimate of our parameters. In Adam, the parameters μ and ν control the exponential decay rates. In the appendix, in Section Appendix c), we specify the Adam algorithm in detail, including all initial conditions.

Algorithm 4 Adam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
3: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
4: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
5: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
6: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon}$

Algorithm 5 Adam with NAG (Nadam)

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $\hat{g}_t \leftarrow \frac{g_t}{1-\mu^t}$
3: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
4: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
5: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
6: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
7: $\bar{m}_t \leftarrow (1-\mu)\hat{g}_t + \mu \hat{m}_t$
8: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{m}_t}{\sqrt{\hat{n}_t} + \epsilon}$

In Algorithm 5, we see Adam with NAG, or Nadam. Adam was developed with classical momentum but it has been shown that NAG is proven to converge faster than classical momentum [3]. This motivates the addition of NAG to Adam, instead of classical momentum. The key differences are highlighted in Steps 2 and 7, where we use the $\bar{m}$ notation discussed earlier in section 2a to add NAG to the Adam framework. This was first proposed in a recent paper called "Incorporating Nesterov Momentum into Adam" in 2016. We use their method of incorporating NAG, instead of classical momentum, into Adam. In our analysis, we compare Nadam and Adam to see if NAG gives us faster convergence compared to classical momentum when combined with L_2 norm methods in this framework.

c) Bias Correction

Lastly, we will analyze the two bias correction steps in Adam and Nadam, highlighted in red below. These two steps were proposed by the authors of the original Adam paper, "Adam: A method for stochastic optimization", as a means to correct for the initialization bias. For both Adam and Nadam, the vectors of m_t and n_t are initialized to be 0. The authors claim that this initialization can cause instability, and they have a proof that illustrates the statistical bias that happens when we have this initialization. Specifically, in their paper, they prove that $E(n_t) = E(g_t^2) \cdot (1-\nu^t) + \zeta$ where ζ is 0 if the true second moment is stationary [7]. In our analysis we study whether omitting the bias correction creates any instability, as is suggested [3]. We remove steps 3 and 5 from Adam as well as steps 4 and 6 from Nadam, all of which are highlighted in red below, in order to form the 6th and 7th algorithm for analysis. We note that step 2 in Nadam also appears to have a similar form as the other bias-correcting stages, but g_t is not initialized to 0, so this step performs a different function other than bias correction and is not mentioned to be one of the bias correction steps in the formulation of the algorithm [3]. We proceed by removing steps 4 and 6 from the algorithm, but in future work we would also like to consider the removal of step 2, although initial analysis shows that the results are similar.

Algorithm 4 Adam

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
3: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
4: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
5: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
6: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t} + \epsilon}$

Algorithm 5 Adam with NAG (Nadam)

1: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
2: $\hat{g}_t \leftarrow \frac{g_t}{1-\mu^t}$
3: $m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$
4: $\hat{m}_t \leftarrow \frac{m_t}{1-\mu^t}$
5: $n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
6: $\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
7: $\bar{m}_t \leftarrow (1-\mu)\hat{g}_t + \mu \hat{m}_t$
8: $\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{m}_t}{\sqrt{\hat{n}_t} + \epsilon}$

Therefore, in summary, we proceed with the comparison of seven algorithms in our analysis. Specifically, we compare the following algorithms:

1. SGD
2. SGD with Classical Momentum
3. Nesterov's Accelerated Gradient (NAG)
4. Adam
5. Nadam (Adam with NAG)
6. Adam without bias correction
7. Nadam without bias correction

3. Objective Function and Data

In the literature on optimization, and also specifically on Adam and Nadam, logistic regression and classification problems are very common. Most of the papers on this topic start off with a comparison of the algorithms in how they perform under logistic regression. Therefore, for our study, we will use two datasets and analyze how our algorithms perform in computing the MLE under logistic regression. In logistic regression, for $i = 1, ..., n$

$$Y_i \sim Ber(p_i(\theta)); \text{ where } \theta = (\beta_1, \beta_2)$$

$$p_i(\theta) = exp(\beta_1 X_{1i} + \beta_2 X_{2i})/(1 + exp(\beta_1 X_{1i} + \beta_2 X_{2i}))$$

where each Y_i is a binary response corresponding to predictors X_{1i}, X_{2i}. All of our algorithms will be used in order to find the Maximum Likelihood Estimates, or the MLEs, $(\hat{\beta}_1, \hat{\beta}_2)$.

a) Data

For our study of these optimization algorithms in the logistic regression setting, we have two datasets. The first dataset is used to see how the algorithms perform in a low-dimensional parameter space, where there are two parameters. This dataset is simply the same dataset we used in Homework 3, where there are 200 observations, 2 covariates, and a binary response.

For our second study, we use study logistic regression in a higher-dimensional parameter space. We use a a dataset from the UCI Machine Learning Repository: the Spambase Dataset [2]. This is a very common Kaggle and Machine Learning dataset, as it is high-dimensional classification problem. The task at hand is to classify whether emails are spam or not. The original dataset has 57 attributes, of which we selected six, for computational limits. If we include all 57 attributes, the algorithm takes much longer to converge, so for our case, we chose to compare in a higher dimensional space where there is an increase in the difficulty of convergence but not so much as to limit our capability to analyze the data. These six covariates are continuous, and represent a word frequency as a percentage of the total number of words. These covariates are words commonly associated with spam emails. There are 4,601 observations in our dataset.

4. Results

Next, we present our results for both the 2-dimensional and 6-dimensional Logistic Regression analysis. Details of our convergence criterion for our algorithm are presented in Section Appendix 2e). In the plots of the estimates, we will present the average estimate over many runs of the algorithms. This is in order to show the variation between the iterations of the algorithms, and to be able to see how the algorithm behaves in general, as there can be some variation between runs due to the randomness involved in SGD when choosing the batch. Details of how many times the algorithms were run are found in each case below. We also present the convergence of the objective functions in Section Appendix 2f).

a) Homework data: 2-dimensional case

For the homework dataset, we compare our seven algorithms by looking at the time and the number of iterations until convergence. We will do so by running our algorithm 100 times, and averaging over the 100 runs to find the average time and the average number of iterations until the algorithm converged. For our algorithms, we use a batch size of one so there is randomness involved when choosing the sub-function at each iteration.

In Table 1, we see that all of the algorithms are quick to find the optimal point when we have 200 observations and two parameters to estimate. All of the algorithms take, on average, less than one second to converge. Also,

all of the algorithms converge in relatively few iterations. However, it is notable that SGD takes quite a bit longer than any of the other algorithms, both in terms of time and in terms of number of iterations. We also notice that Nadam takes longer than Adam, both with and without the bias correction. Lastly, we see that when we do not include the bias correction, the algorithms converge faster than when we do include the bias correction. We also notice that SGD with momentum and NAG are both converging quickly, although Adam is the fastest. The relative speed of these two algorithms may be because this problem is not high dimensional enough to realize the advantages of Adam and Nadam. We also see that the times are quite similar, for example between Nadam and Nadam without the bias correction, which shows the importance of considering a higher dimensional case.

	SGD	SGD w/ Mom	NAG	Adam	Nadam	Adam w/o BC	Nadam w/o BC
time (sec)	0.31	0.04	0.04	0.02	0.10	0.03	0.11
iter	267.81	35.21	40.55	49.43	114.04	30.87	104.96

Table 1: Comparison of Time and Iterations, 2-dimensional case

In Figure 3, we are able to once again verify some of the findings we saw above. We plot the convergence of the parameters, rather than the objective function, because we are interested in the impact of removing the bias correction on the estimates of the parameters. However, the convergence of the objective function is available in the Appendix Section f). In Figure 3, we see once again that SGD takes the longest to converge. We also notice that Adam and Nadam may be more stable, especially in case of the second parameter, than Adam and Nadam without the bias correction. However, even without the bias correction, the convergence is relatively stable, and converges quite quickly. It is also interesting to note that the Adam and Nadam algorithms without the bias correction are more stable than SGD with momentum and NAG in the case of the first parameter.

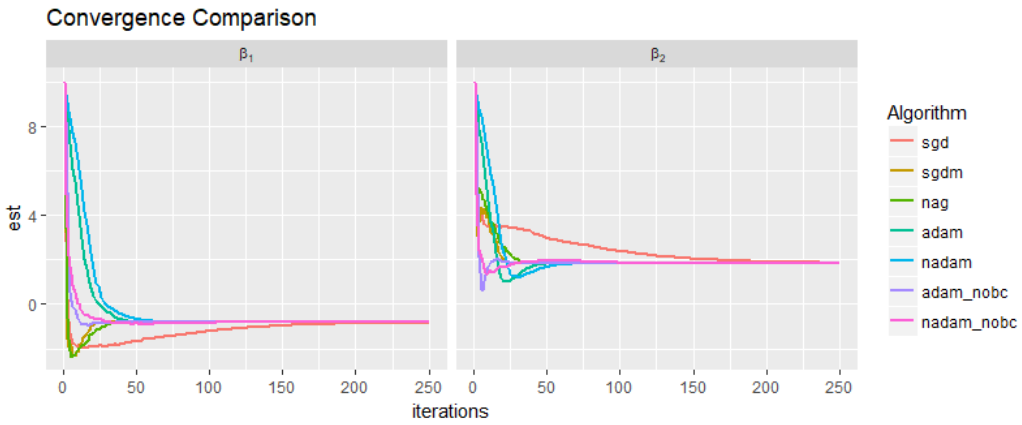

Figure 3: Comparison of 2-dimensional homework dataset

b) Spam dataset: 6-dimensional case

We continue our analysis by applying our optimization methods to a larger dataset. This dataset is larger both in terms of number of observations and number of parameters. In this case, as mentioned earlier, we will use 4,601 observations to estimate six parameters for the spam classification dataset, once again in a logistic regression framework. We have a higher batch size in this case. In the previous case, we had a batch size of one. However, in our case if the batch size is small, our algorithm is much less stable. Therefore, we use a high batch size of 3,000 in our case, which makes this much more similar to Gradient Descent, but still is a smaller dimension. Also, we average the time and the number of iterations over only 10 runs, instead of 100 as in the two dimensional case.

In Table 2, we see that once again SGD takes the longest both in terms of time and the number of iterations. We also see that SGD with momentum and NAG are quick to converge compared to our other algorithms. Nadam

takes consistently longer than Adam to converge, both with and without the bias correction. This time, it takes quite a bit longer than Nadam to converge; where Adam still converges in less than a second, Nadam now takes several seconds to converge. The bias correction had mixed results in this case. For Adam, the bias correction continues to slow down convergence compared to when we remove the bias correction. However, for Nadam removing the bias correction slows convergence of the algorithm.

	SGD	SGD w/ Mom	NAG	Adam	Nadam	Adam w/o BC	Nadam w/o BC
time (sec)	6.89	0.23	0.29	0.55	4.49	0.41	5.63
iter	749.00	60.67	79.11	213.89	624.67	94.56	704.33

Table 2: Comparison of Time and Iterations, 6-dimensional case

In Figure 4 we can verify some of the findings stated above. We have plotted just the first two parameters out of six to see the effect of the bias correction on two example parameters. Once again, we see that SGD is the slowest to converge, especially in the case of the second parameter. Adam and Nadam without the bias correction both begin to show some of the instability that was mentioned in the paper when we compare to Adam and Nadam that include the bias correction.

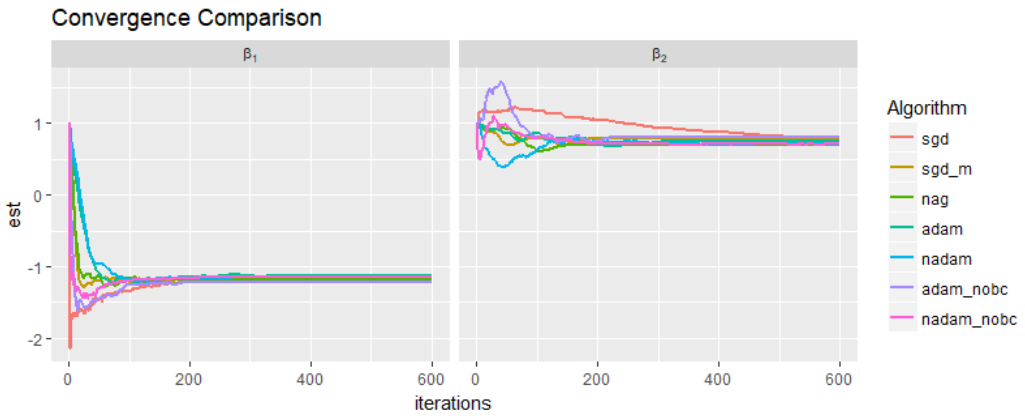

Figure 4: Comparison of 6-dimensional spam dataset

5. Conclusion and Future work

In summary, there are some common findings across both of our studies in the 2-dimensional and 6-dimensional case. First, we see that all versions of momentum and L_2 norm algorithms that we tested are an improvement on SGD. Next, we see that Nadam (Adam with NAG) takes longer than Adam to converge both in terms of time and the number of iterations. Also, the bias correction does not seem to have a large impact on convergence, and can actually slow down convergence. Therefore, the advantage of the bias correction in terms of decreasing instability is not obvious from our analysis. This bias correction has not been studied in any other readily available papers.

One of the main things learned through this project is how cumbersome it can be to tune these optimization algorithms. Although it may not be apparent in the results, we would like to also note that the Nadam algorithm is substantially easier to tune than the Adam algorithm.

In the future, there are many additional circumstances we would like to test to gain a better understanding of these algorithms. First, we would like to consider noisier objective functions, in addition to logistic regression. This may show advantages of Adam and Nadam compared to NAG and SGD with momentum that were not clear in our examples. Second, we would like to consider higher-dimensional datasets, both in terms of number of parameters and in terms of number of observations. Lastly, we would like to consider other applications such as machine learning and neural networks, as this method is used frequently in these settings.

6. Supplementary material

References

[1] Li Deng et al. “Recent advances in deep learning for speech research at Microsoft”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8604–8608.

[2] Dua Dheeru and Efi Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[3] Timothy Dozat. “Incorporating Nesterov Momentum into Adam”. In: (2016).

[4] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE. 2013, pp. 6645–6649.

[5] Brett Grossfeld. *A Simple Way to Understand Machine Learning vs Deep Learning*. July 2017. URL: https://www.zendesk.com/blog/machine-learning-and-deep-learning/.

[6] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

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

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[9] Bernard Marr. *What Is The Difference Between Deep Learning, Machine Learning and AI?* Dec. 2016. URL: https://www.forbes.com/sites/bernardmarr/2016/12/08/what-is-the-difference-between-deep-learning-machine-learning-and-ai/#41ae410f26cf.

[10] Yurii Nesterov. “A method of solving a convex programming problem with convergence rate O (1/k2)”. In: *Soviet Mathematics Doklady*. Vol. 27. 2. 1983, pp. 372–376.

[11] Boris T Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[12] Ilya Sutskever et al. “On the importance of initialization and momentum in deep learning”. In: *International conference on machine learning*. 2013, pp. 1139–1147.

[13] Tiep Vu. *Bai 8: Gradient Descent (phan 2/2)*. Jan. 2017. URL: https://machinelearningcoban.com/2017/01/16/gradientdescent2/.

[14] *What Is Deep Learning? How It Works, Techniques and Applications*. URL: https://www.mathworks.com/discovery/deep-learning.html.

a) Details on Notation for algorithms

We will explain some of the notation in our algorithms, that were omitted above, after receiving some questions during the presentation. First, all of the terms in our algorithms represent either vectors or constants. We will go through an example for the Adam algorithm, which is seen in detail later in the Section Appendix c). The following are simply a vector of length θ (either two or six dimensional in our case): $g_t, m_t, v_t, \theta_t, \theta_{t-1}$. The following are scalars/constants that are multiplied, divided, or added to a vector, element-wise: μ, ν, η, ϵ. These scalars are all specified as tuning parameters and most of them are used as specified in the paper, with $\mu = 0.9$, $\nu = 0.999$, and $\epsilon = 1e - 7$. The learning rate, η, requires some tuning as we vary the algorithms. Generally, it is higher in SGD and lower once we incorporate momentum.

b) Other L_2 Norm Methods

There are several other L_2 norm methods in the literature. These include, among others, RMS Prop, AdaDelta, and AdaGrad. RMSProp is an improvement on AdaGrad, as shown below in Algorithms 6 and 7. In AdaGrad, the norm vector n_t can become so large that the model no longer reaches any local minimum. Therefore, in RMSProp, they replace the sum in n_t with a weighted average, which allows the model to learn indefinitely [3]. We also see that this weighted average, instead of a sum, is also incorporated into the momentum vector in Adam.

Algorithm 6 AdaGrad

$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

$n_t \leftarrow n_{t-1} + g_t^2$

$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{n_t + \epsilon}}$

Algorithm 7 RMSProp

$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$

$n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$

$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{n_t + \epsilon}}$

c) Details of Adam

In this section, we present the Adam algorithm in detail, including all necessary initializations and requirements for the algorithms. In all other discussion of algorithmic details, including the SGD and NAG algorithms, we only include steps that are inside the while loop.

Algorithm 8 Detailed Adam Algorithm Description

Require: η Stepsize

Require: $\mu, \nu \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0: Initial parameter vector

Require: ϵ: a very small number to prevent the denominator from being equal to 0

Require: Initializations as follows:

- $m_0 \leftarrow 0$ (1st moment vector)
- $v_0 \leftarrow 0$ (2nd moment vector)
- $t \leftarrow 0$ (timestep)

while not converged **do**

  $t \leftarrow t + 1$

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Gradients wrt stochastic objective)

  $m_t \leftarrow \mu \cdot m_{t-1} + (1-\mu) \cdot g_t$ (Update biases first moment estimate)

  $v_t \leftarrow \nu \cdot v_{t-1} + (1-\nu) \cdot g_t^2$ (Update biases second raw moment estimate)

  $\hat{m}_t \leftarrow \frac{m_t}{(1-\mu^t)}$ (Compute bias-corrected first moment estimate)

  $\hat{v}_t \leftarrow \frac{v_t}{(1-\nu^t)}$ (Compute bias-corrected second raw moment estimate)

  $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

  Check convergence criterion

end while

return θ_t

This is where:

- g_t^2 is elementwise square $g_t \odot g_t$
- μ^t, ν^t are just μ and ν to the power t
- $f(\theta)$ stochastic objective function
- The last three steps can also be written as: $\eta_t = \eta \cdot \sqrt{1-\nu^t}/(1-\mu^t)$ and $\theta_t \leftarrow \theta_{t-1} - \eta_t \cdot m_t / (\sqrt{v_t} + \hat{\epsilon})$ (improves efficiency)

d) Backtracking

For all of the algorithms presented in this paper, we include backtracking in the implementation. The backtracking part of the algorithm is as follows:

Algorithm 9 Backtracking Method

if obj_f^t − obj_f^{t-1} ≥ 0 **then**
  backtrack = T
  s = η
else
  backtrack = F
end if
while backtrack = T **do**
  $s = s/2$
  Proceed through normal update procedure with s instead of η, creating a new estimate of obj_f^t
  backtrack = (obj_f^t − obj_f^{t-1} ≥ 0)
end while

e) Convergence Criterion

We have several convergence criteria for our algorithms. First, we would like to see if the two successive estimates of the parameters are reasonably close to each other. This is determined by the absolute value of the difference between the two estimates, scaled by the previous estimate. Due to the fact that we have several parameters to estimate in some cases, we take the maximum scaled difference and ensure it is below a certain tolerance level. In our case, it is 1e-10. We also look at the successive values of the log-likelihood, or our objective function. We also scale this by the previous evaluation of the objective function, and we ensure it is below a certain tolerance level before stopping our algorithm. Again, for our algorithm this is a tolerance level of 1e-10. We also have a maximum number of iterations where if the algorithm goes too far above this maximum number of iterations, we will stop and reassess our starting criteria. We typically use a very high number that is never reached in our examples, such as 1,000.

So, the main two convergence criteria are as follows:

$$\frac{|\beta_i^t - \beta_i^{t-1}|}{|\beta_i^{t-1}|} < \text{tol and } \frac{|\text{obj\_f}^t - \text{obj\_f}^{t-1}|}{|\text{obj\_f}^{t-1}|} < \text{tol}$$

f) Convergence of Objective Function

In addition the the convergence of the coefficients, we are also interested in the convergence of the objective function. We see in Figure 5 similar results to what we found in Table 1. Once again, we see that Stochastic Gradient Descent takes the longest to converge. However, we can also clearly here that Adam and Nadam also take a bit longer to converge, and are more gradual in their path to the optimal point. We also notice that Adam without the bias correction converges quite quickly.

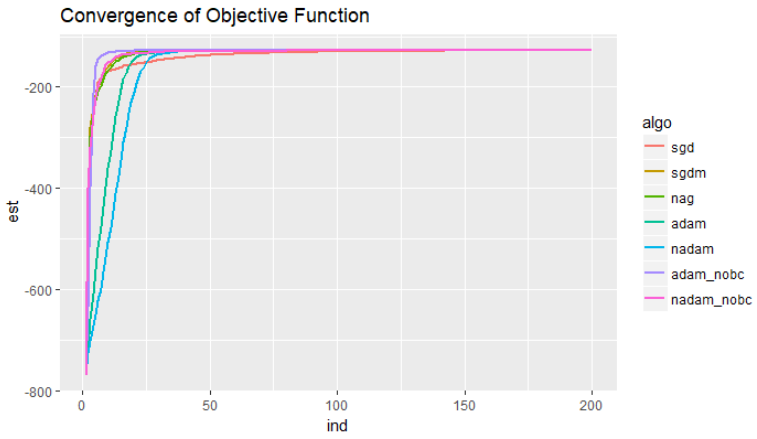

Figure 5: Comparison of Objective Function Convergence, 2-dim case

In the case of the 6-dimensional problem, we see that it is harder to find the same pattern as in the time and iterations until convergence. It appears that Adam is the slowest to converge from Figure 6. However, this may be due to it satisfying the convergence criteria faster than SGD, which may still vary even close to the optimal. In fact, we can see there is similar behavior to the convergence in the 2-dimensional case, where SGD seems to converge relatively quickly, but then takes quite a while to find the optimal point, as is the case with the 6-dimensional case. In Figure 6, we have just included five of the algorithms, those related to Adam and Nadam as well as SGD, so we can see the convergence clearly.

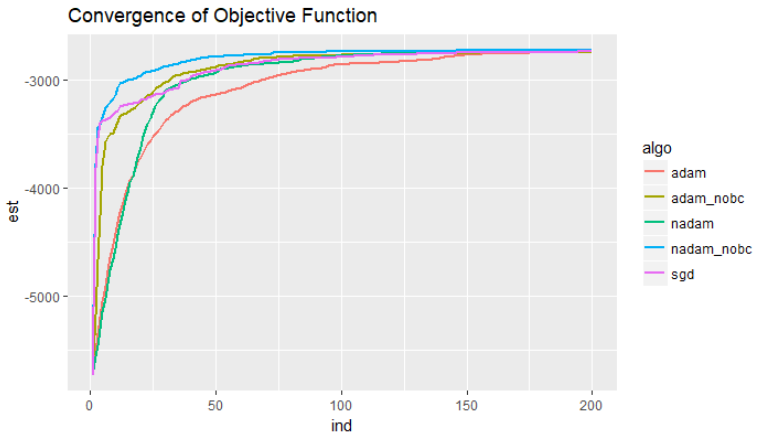

Figure 6: Comparison of Objective Function Convergence, 6-dim case