Optimization Methods for Deep Learning

Project Report for Convex Optimization

Convex Optimization

Prepared by: Xunzhe Wen
Electrical and Computer Engineering
University of Ottawa, Ottawa, ON, Canada

Winter 2018

List of contents

List of contents

List of contents

I. ABSTRACT

The goal for this project is to understand the optimization algorithms used in deep learning. This is about optimization algorithm, and machine learning.

This project begin by reviewing the references which introduces the optimization algorithm used in machine learning techniques. In this part, I will go through a portion of first order and second order optimization algorithms which are commonly used in machine learning, providing essential mathematical fundamentals, their mechanisms and compare them with respect to the performance they achieved in the researches or industries. Next, a classical algorithm in optimization of machine learning field, Adam method, will be discussed in details for specification, which will also gives us some insights about how to choose the proper optimization algorithm for a specific conditions. Gradient descent algorithms are based reference [1], Momentum methods are based reference [2], and the adaptive adjust methods are based on reference [3]. More references have been also reviewed and they are served as the supportive materials for this report.

The latter part of this project report will focus on the simulations of optimization algorithms. In this part, the classical optimization algorithm are simulated in order to intuitively understand the properties and performance of previously described algorithms. Gradient-based optimization algorithms are simulated and compared under the circumstance of solving monadic non-constraints optimization problem (one-dimensional); the gradient based and Newton method are simulated and compared under a two-dimensional function optimizations. Finally, the optimization algorithms are equipped in a neural network framework for reorganization task (high-dimensional), which involves high dimensional input and parameters.

Key words: Optimization, Machine learning, Algorithms.

Contributions: Conclude most of existing optimization algorithms used in machine learning, and verify the conclusion about their performance in different kinds of tasks. In the end, I personally proposed a learning strategy which combines different algorithms for a learning procedure, in which case we can make better use of the advantages of each combined algorithm.

II. INTRODUCTION

Machine learning has been an active technique used in research and industries, and it based on statistics and computing techniques. The learning algorithm serves a key role so that the designed machine learning models can learn knowledge from the data human provided. A reliable optimization algorithm within the learning procedure can speed up the learning speed and convergence speed.

In the machine learning model, there are some parameters, such as weights and bias which are used to compute the differences between the estimated and desired value in the test set, and the loss function are based on these parameters. The purpose of the optimization algorithm is to minimize or maximize the loss function in the learning model.

The parameters in the model server the significant role in training the model to produce precise results, and that is also the reason why we should use all sorts of optimization strategies and algorithms to update the network parameters in the training model.

During my master studies, I met with many situations which are required the knowledge about optimization in machine learning or deep learning. For many times I just used an algorithm that other scientist mentioned in their papers. Moreover, there is a common sense that the machine learning problems are the optimization problems at the end. In this project, I will systemically go through the optimization algorithms used in machine learning, or deep learning.

The existing and commonly-used optimization algorithms are studied based on the reference [1, 2, 3]. To better understand each algorithm, not only the mathematical fundamentals are demonstrated, but also the advantages and disadvantages of each algorithm are compared and discussed in this report. A specific algorithm, Adam algorithm [3], has been studied in details for simulations. Moreover, a portion of optimization algorithms are simulated to solving optimization problems with one, two or multi-variables problems, including for a deep neural networks for recognition tasks. Those simulation used Python for coding tasks.

III. OPTIMIZATION PROBLEM OVERVIEW

In the optimization problem can be significant to the science and industries, and it can be named as operational research. In the machine learning framework, it needs to construct a proper target function, and the minimum or maximum of this function can be the optimal solution we want to achieve. Therefore, the key is to find a way how to get that optimal values within the target function.

Optimization problem can be categorized based on there are constrains or not, and the constraints are be distinguished from the equality or inequality constraints. Fortunately, the scientists have found the solutions and strategies to solve those optimization problems, and professor Sergey Loyka has also explained the methods in the lecture. Here, I will make the summary to identify those strategies

1. Optimization without constrains:

   The optimization can be derived from the case where the derivative of the target function are equal to zero;

2. Optimization with equality constrains:

   Usually convert the optimization problem with equality constrains into the optimization problem without constraints by using the Lagrange multiplier;

3. Optimization with inequality constraints:

   Using KKT conditions (Karush-Kuhn-Tucker conditions) to convert inequality constrains into non-constraints optimization problems.

Fortunately, the optimization techniques which deals with or without constraints have been developed for a few decades. A large variety of optimization algorithms are readily available and widely used in the research field and industries. Let’s quickly go through the commonly used optimization algorithms and Table 1 shows the optimization algorithms in terms of there are constraints or not.

The optimization algorithms can also be clustered in terms of the derivative order the algorithm requires, as the Table 2 shows.

The first order algorithms used the gradient of each parameter to minimize or maximize the loss function in the learning model. The gradient is the multi-variables expression for derivative of a function, and we use gradient to replace derivatives in order to compute the derivatives of a multi-variables function. In a word, derivative can be computed to analyze single variable function, while gradients are used for multi-variables function. Gradient of a multi-variables function provides a vector, but I will not go into more theoretical details in this report.

The second order optimization algorithm, also called Hessian method to optimize the loss function. It requires large space and expensive computation complexity to get Hessian matrix, so this can be one of the reasons that second order optimizers are not that widely used in machine learning.

Optimization algorithms	
Without Constraints	***With Constraints***
1. Gradient descent 2. Newton method 3. Quasi-newton method 4. Conjugate gradient method ……	1. Monte Carlo method 2. Lagrange multiplier method 3. KKT conditions ……

Table 1. The optimization algorithms classified in terms of constraints.

Optimization algorithms	
First order	***Second order***
1. Gradient descent: Stochastic gradient descent Batch gradient descent 2. Nesterov method 3. Adaptive learning method: RMSProp, Adagrad, AdaDelta Adam ……	1. Newton methods 2. Quasi-newton methods: BFGS 3. Hessian –free method 4. Subsampled Hessian method ……

Table 2. The optimization algorithms classified in terms of derivative orders.

Usually there could be no analytical form of solutions in many machine learning problems, or the analytical solution will result in huge computational complexity, thus the common sense to solve this problem is to using iterative procedures to reach the optimal solution, and this is the strategy for many state-of-the-art optimization algorithms used in machine learning.

Before we go, it is necessary to introduce a framework for better understand of following algorithms. First we define the parameters needs to be optimized: θ; loss function: $J(\theta)$; and the initial learning rate η_0; the learning step in each iteration k are:

i. Compute the current gradient of loss function:

$$g_k = \nabla J(\theta_k) \tag{1}$$

ii. Compute the first and second order momentum according to former gradient:

$$m_k = \phi(g_1, g_2, \ldots, g_k) \tag{2}$$

$$V_k = \psi(g_1, g_2, \ldots, g_k) \tag{3}$$

iii. Compute the current gradient descent:

$$\eta_k = \eta_0 \cdot m_k / \sqrt{V_k} \tag{4}$$

iv. Update the parameter using current gradient descent:

$$\theta_{k+1} = \theta_k - \eta_k \tag{5}$$

By using this framework, for many algorithms, the third and fourth step are almost the same, but the first and second step differs.

IV. OPTIMIZATION ALGORITHMS

In this part, I will choose some first order and second order optimization algorithms which are commonly used in machine learning to analysis based on what I reviewed from the online sources. The analysis will provides essential mathematical fundamentals, mechanisms, and I will also compare them with respect to the performance they achieved in the researches or industries. Not only compare the algorithms mentioned in the selected references, but I also did a few small paper review to discuss them.

1. Batch Gradient Descent

Gradient descent is the most important but fundamental technique to train a system, it makes the model converge by updating the parameters according to the gradient computations. The parameter updating follows the equation [1]:

$$\theta := \theta - \eta \cdot \nabla(\theta)\,|_{J(\theta)} \tag{6}$$

Where the θ is the parameters, η is the learning rate and is fixed in this case, and $\nabla(\theta)|_{J(\theta)}$ is the gradient of the loss function $J(\theta)$. Gradient descent is the most commonly-used optimization algorithm in neural networks, for parameter update in the negative gradient direction so as to minimize the loss function. The convergence speed depends on the condition that the loss function is strongly convex or merely convex [4].

Advantages:

(1) The gradient descent method is simple to implement because the fixed learning rate and no need to decay.

(2) When the objective function is convex, the solution of the gradient descent method is the global solution. But the solution will be stuck in the local minimum if the problem is not convex.

(3) Moreover, the gradient is generated with unbiased estimation, the standard error will be low if used larger samples.

Disadvantages:

(1) When the dataset is large enough, the process can be slow to go over the whole batch.

(2) When gradient descent method approaches the target value, the update step can become smaller, so it slows the convergence progress.

In order to overcome these disadvantages, mini-batch gradient descent and stochastic gradient descent were proposed.

2. Mini-Batch Gradient descent

In this case, we will go through the batch of the whole data, thus the learning process will be executed on every smaller batch of the whole data. We used *b* samples in each update and the corresponding update can be expressed as [1]:

$$\theta := \theta - \eta \cdot \nabla\left(x^{\{i:i+b\}}, y^{\{i:i+b\}}; \theta\right)|_{J(\theta)} \tag{7}$$

Note that the each mini-batch is formed by randomly picking up the data and the gradient is computed over this mini batch. The batch size can be another hyperparameter. In order to achieve a better run time for GPUs, the batch size usually has been chosen as an integer powered by 2.

Advantages:

(1) Obviously, instead of computing the gradient over whole dataset, it will be faster in each smaller batch since the computation time per update is dependent on the size of dataset. It usually used for extremely large datasets.

(2) Randomly chosen samples can reduce the fluctuation of parameter update.

Disadvantages:

(1) It might not converge to the end, because the learning step may go back and forth due to the noise on each iteration, thus, it will wander around the minimum region.

(2) Due to the noise, the learning steps have more oscillations, which need learning rate decay.

3. Stochastic Gradient Descent

Unlike the former gradient based algorithm, the stochastic gradient descent will focus on only one sample to make parameter update, so the updating can be expressed as [1]:

$$\theta := \theta - \eta \cdot \nabla\left(x^{\{i\}}, y^{\{i\}}; \theta\right)|_{J(\theta)} \tag{8}$$

Since stochastic compute the gradient based on a single sample, the training speed can be much faster than the batch gradient descent if the dataset is huge. However, this method comes with more noise than the batch or the mini-batch approach, the estimated gradient over each updated sample will not point to the optimum location. The updates seem to be random, but it iterated to going to the optimal region eventually.

More intuitively speaking, stochastic gradient descent scarifies a small portion of accuracy and iteration consumer, to earn the overall optimization efficiency. But the increasing of the number of iteration will be much smaller than the total number of samples.

Advantages:

(1) It provides a trade-off between local learning efficiency and global optimization effect.

(2) It is faster than the batch gradient descent when dataset is large.

Disadvantage:

(1) It will wander around the optimum.

(2) The stochastic gradient descent will suffer from the noise which increase the number of iterations, and the variance can be large since only one sample is used for each update.

Analysis: Reference [1] gives the idea of manipulation of gradient descent algorithms in practice, explains why SGD is a good learning algorithm when the training set is large, and provides useful recommendations. Gradient descent based algorithms in general are easy to implement, and it can guarantee that it will converge to a global optimum if the loss function is convex. But for non-convex problem, it have no ability to avoid to be stuck into a local optimum since the update will be stopped once the gradient equals to 0. By the way, for non-convex problem, we can convert that into a dual problem since the dual function can be convex, but the solution for this dual function can be the lower bound of the original function. Moreover, the learning rate in these techniques are fixed, and the truth is the learning performance are highly dependent on the choice of this hyperparameter. Too small learning rate gives slow convergence, but a large learning rate can make the update fluctuant over the optimal region, sometimes even divergence. An example which demonstrated about the learning trajectory towards to optimum of three gradient-based algorithms shows below.

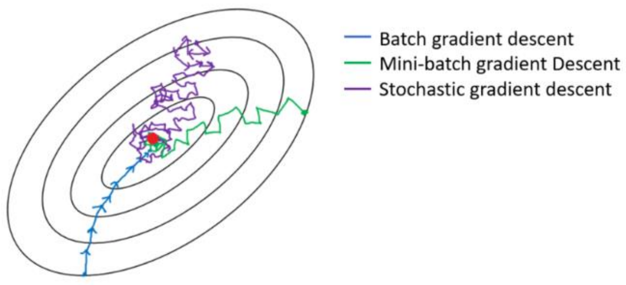

Figure 1. Learning trajectory towards to optimum of three gradient-based algorithms. Source: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3.

In practice the learning rate is designed to decay linearly till the iteration τ [4]:

$$\eta_k = (1-\alpha)\cdot\eta_0 + \alpha\cdot\eta_\tau \ , \quad \alpha = \frac{k}{\tau} \tag{9}$$

Where η_k is learning rate at iteration k. τ is usually set to the number of iterations needed for a large number of passes through the data, and η_τ should roughly be set to 1% of η_0.

Gradient descent based algorithm is not always the most effective method to solve optimization problems in practice since the drawbacks we just discussed. Next we will go through also widely-used and more advanced optimization algorithms used in machine learning.

4. Momentum Method

In practice, the gradient descent based method is not always the most effective way to solve optimization problems in machine learning. Since the noise in the stochastic gradient descent and mini-batch gradient descent method, the high variance in gradient descent method will suffer from the oscillation and it cause the network hard to converge. Researchers have proposed a technique which brought the momentum in.

The idea of this method is to focus on the optimization in a relevant direction while eliminate the oscillations along the irrelevant direction, hence the gradient descent method can be accelerated. The momentum is related to inertia in physics, which means the direction of the update will depend on not only the current gradient descent direction, but also the previous update direction.

So in every update using momentum, the optimizer is not 100% confidence at the current gradient, so it will keep a certain part of the previous learning step, and this update procedure can be expressed as [2]:

$$m_n = \mu\cdot m_{n-1} + \nabla(\theta)|_{J(\theta)} \tag{10}$$

$$\theta := \theta - \eta\cdot m_n \tag{11}$$

Where μ is another hyperparameter called momentum factor and it is set to smaller than 1, m denotes the momentum in each learning step. With the momentum, the variance in the learning procedures are effectively reduced, furthermore, the oscillations are also suppressed as the Figure 2 shows.

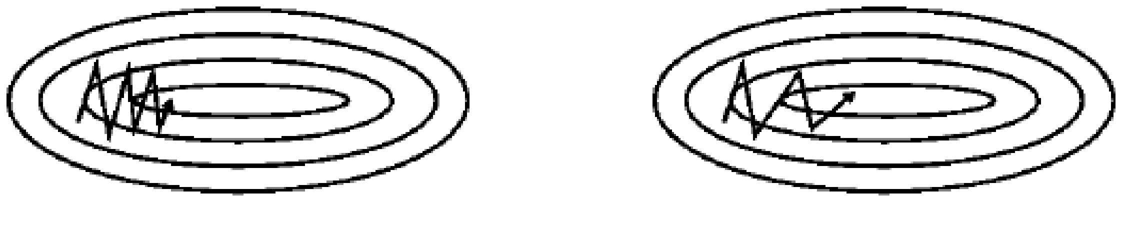

Figure 2. Stochastic gradient descent with momentum (left) and without momentum (right). Source: https://www.willamette.edu/~gorr/classes/cs449/momrate.html.

Advantages:

(1) Momentum method suppressed the problem that the gradient descent methods converge not stable when using batches, and accelerated the learning procedures, furthermore, it will have faster convergence.

(2) It has the ability to get rid of the local optimum, because it will keep updating a few steps after the current gradient is zero and the momentum term is still non-zero.

Disadvantages:

(1) It has one more hyper parameter, (momentum factor and learning rate), and their initialization will heavily influence the performance of optimization algorithm.

(2) Sometimes, with the accumulated momentum and increasing update speed, this optimizer have the probability to jump out of the optimum region.

5. Nesterov Method

Researcher Yurii Nesterov discovered the drawback of the momentum method: when the update reached the minimum, it has accumulated a relatively high momentum and the optimizer will miss the minimum, thus telling the optimizer when to decelerate is necessary. Nesterov proposed a method to solve this problem in 1983, and this algorithm was named with him.

The update using Nesterov momentum is expressed as [2]:

$$m_n = \mu \cdot m_{n-1} + \nabla\left(\theta_{n-1}\right)\big|_{(\theta_{n-1}-\eta\cdot\mu\cdot m_{n-1})} \tag{12}$$

$$\theta := \theta - \eta \cdot m_n \tag{13}$$

According to the update formula, the former momentum m_{n-1} has influenced the current gradient. Therefore, Nesterov method can be explained as the optimizer jumps ahead according to the previous momentum, and compute to calibrate the current gradient, the finally update the parameters. This procedure shows as Figure 3.

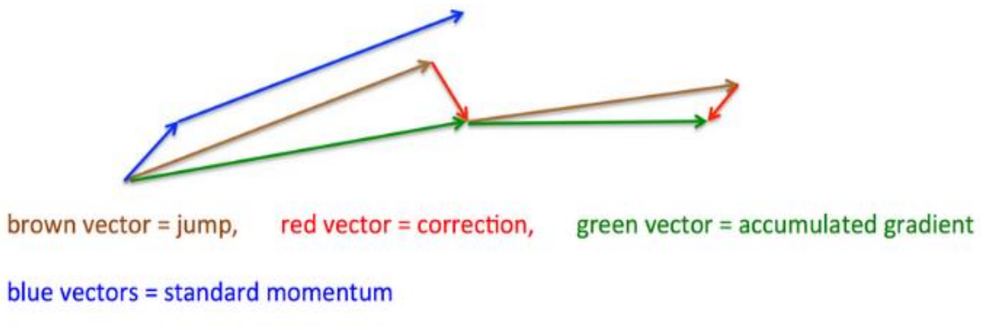
brown vector = jump,
red vector = correction,
green vector = accumulated gradient
blue vectors = standard momentum

Figure 3. Nesterov method update illustration. Source: https://www.coursera.org/learn/neural-network.

Momentum compute a gradient as the short blue arrow and make a significant jump along the direction where the update happens as the long blue arrow shows. Nesterov firstly make a jump along the previous gradient direction as the brown vector, and then revise the direction based on the current gradient.

Advantages:

(1) Inherit the advantages of momentum method.

(2) Nesterov method enabled the momentum with a prediction of approximate update, and it corrects faster.

Disadvantages:

(1) Like all gradient descent and momentum techniques, they can guarantee the global optimum if loss function is convex, but cannot if non-convex.

(2) The manually set of the hyperparameters such as learning rate and momentum factor could unpredictable influence the performance of the algorithm.

Analysis: Reference [2] find that both the initialization and the momentum are crucial when training a deep network, and they emphasized the importance of the initialization for momentum methods. The gradient descent algorithms and the momentum methods update their parameters with a fixed learning rate, but a huge amount of parameters with multi-dimensional are involved with the neural network and not every single parameter in that network is used frequently. For the frequently updated parameters, we hope they are not significantly affected by single sample, hence the lower learning speed is expected; but for that parameters which get updated infrequently we hope it can learning from the rarely shown samples, in other words, larger learning rate are expected.

So next we will step into a category of algorithms which can adaptively adjust their learning rate.

6. AdaGrad method

In this algorithm, the second-order momentum, which indicates the historical updated frequencies, has been employed. The second order momentum serves the fundamental evidence for adaptive learning rate.

The idea of AdaGrad algorithm is downscaling a model parameter by square-root of sum of squares of all its historical values in order to adjust the learning rate. Notice that the weights that receive high gradients will have their effective learning rate reduced, while weights that receive small or infrequent updates will have their effective learning rate increased. Thus, the AdaGrad algorithm are suitable for processing sparse dataset.

This algorithm adjust learning rates according to the recorded the historical gradient information for each parameter in every iteration, and its update equation can be [3]:

$$V_n = V_{n-1} + \left(\nabla\left(\theta_i\right)|_{J(\theta)}\right)^2 \tag{14}$$

$$\theta := \theta - \eta \cdot m_n / (\sqrt{V_n} + \varepsilon) \tag{15}$$

Where V_n denotes the second order momentum, and notice that V_n has size equal to the size of the gradient, and keeps track of per-parameter sum of squared gradients. If a parameter updated frequently, hence its second order momentum V_n can grow large.

This is then used to normalize the parameter update step, element-wise. Amusingly, the square root operation turns out to be very important and without it, the algorithm performs much worse. The smoothing term ε avoids division by zero.

As we can see, the learning rate is slightly reduced with the iteration goes. Intuitively, the function can be more flat and its gradient can be smaller while we are reaching the optimum, so we need take smaller and smaller steps to make sure the optimum cannot be passed through.

Advantages:

(1) It proposed a solution that the learning rate can adaptively adjusted, so no need to manual set the learning rate. It will augment the gradient at beginning, and restrain the gradient at the final steps to convergence.

(2) Sparse data are suitable to be processed using AdaGrad method.

(3) Performance is good when the objective is convex.

Disadvantages:

(1) The learning rate are always decreasing. Since the regularization terms are positive and the accumulated sum are keep increasing. This can decay the learning rate to be a tiny value at the final steps, which slows the convergence or even stop to learn

(2) Still needs human manipulation to set a global learning rate.

7. RMSProp

AdaGrad works well when the objective is convex and it can shrink the learning rate in a fixed decay procedure. The RMSProp update adjusts the AdaGrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate. In particular, it uses a moving average of squared gradients instead, and we can also manipulate the accumulation of an exponentially decaying average of the gradient. This is what RMSProp performed in many neural networks [3].

RMSProp combines the idea that keeping a moving average of the squared gradient for each weight, which only uses the sign of the gradient. This strategy effectively avoid the earlier stop of the learning procedure caused by accumulated second order momentum. Its update can be expressed as:

$$V_n = \rho \cdot V_{n-1} + (1-\rho) \cdot \left(\nabla \left(\theta_i \right) |_{J(\theta)} \right)^2 \tag{16}$$

$$\theta := \theta - \eta \cdot m_n / (\sqrt{V_n} + \varepsilon) \tag{17}$$

Where ρ is decay parameter. When ρ = 0.5, the RMSProp becomes to AdaGrad. In terms of some complex non-convex function in deep learning, the learning trajectory of parameters will pass through sorts of areas to reach a local convex area, and find a local minimum in this area. AdaGrad used in such networks may reduce the learning step too fast before it reaches this local convex region, causing it to fail to converge. But RMSProp overcame this drawback, and it can forget the previous information using the exponential decay so that the weight of the recent gradient information can be augmented. After the optimizer stepping into the local convex region, it will perform as AdaGrad does.

Adavantages:

(1) Overcome the shortcoming of AdaGrad method, which avoid the aggressive decay of the learning rate.

(2) Good for training big and redundant datasets.

Disadvantages: It still depends on a global learning rate.

RMSProp has experienced a lot of developments: it has collaborated with standard momentum but the performance had not been significantly improved. Also RMSProp has combined with Nesterov momentum method, but it only performed better if the root-mean-square of the recent gradients is used to divide the correction rather than the jump in the direction of accumulated corrections.

8. Adam (Specified)

Some explorations has been made on the RMSProp combined with the momentum, but the effects are not that remarkable. Adam is like RMSProp with momentum, but involved with the bias correction terms for the first and second momentum. Adam uses the first and second order moment estimations of the gradient to adaptively adjust the learning rate of each parameter. After the bias correction, the learning rate can be dynamically constrained into a certain slope, hence the parameters are updated stably.

According to the reference [3], the parameter update rule can be expressed as:

$$\hat{m}_n = \frac{m_n}{1-\beta_1^n} \tag{18}$$

$$\hat{V}_n = \frac{V_n}{1-\beta_2^n} \tag{19}$$

$$\theta := \theta - \frac{\eta}{\sqrt{\hat{V}_n} + \varepsilon}\hat{m}_n \tag{20}$$

Where β_1 and β_2 are exponential decay rates for moment estimates, and it suggested defaults by β_1 = 0.9, β_2 = 0.999. Again note that it used the first and second order moment estimation of the loss function to constrain the global learning rate.

Momentum and exponential decay of the weight are used for correcting the step size. In every iteration, these two terms are the weighted averages of the historical series and the initialization gives each term as zero.

Convergence analysis:

Given an arbitrary, unknown sequence of convex cost functions:

$$f_1(\theta), f_2(\theta), \ldots, f_T(\theta) \tag{21}$$

At any iteration step t, we want to predict the parameters θ_t, and evaluate it on a previously unknown cost function f_t. The reference evaluate Adam algorithm using the regret, that is the sum of all the previous difference between the online prediction $f_t(\theta_t)$, and the optimal parameter is denoted as θ^* [3]:

$$\theta^* = \arg\min_{\theta \subset \mathrm{K}} \sum_{t=1}^{T} f_t(\theta) \tag{22}$$

And the K is a feasible set. The regret bound has been proven to be:

$$O\left(\sqrt{T}\right) \tag{23}$$

Assume that the function f_t has bounded gradient for any m, n belongs to $\{1,2,\ldots,T\}$:

$$\left\|\nabla f_t(\theta)\right\|_2 \le G \tag{24}$$

$$\left\|\nabla f_t(\theta)\right\|_\infty \le G_\infty \tag{25}$$

Then the Adam algorithm can guarantee that for all $T \ge 1$:

$$\frac{R(T)}{T} = O\left(\frac{1}{\sqrt{T}}\right) \tag{26}$$

And we can also get:

$$\lim_{T \to \infty}\left(\frac{R(T)}{T}\right) = 0 \tag{27}$$

Advantages:

(1) Combines the advantages of processing sparse gradient (AdaGrad), and non-stable objects (RMSProp) [3].

(2) Adam converges faster since it is not required to store all scaled gradient, hence it is suitable for large-scale datasets and high-dimensional space (many non-convex optimization).

(3) Setting different adaptive learning rate for different parameters.

Analysis: For most of the optimization algorithms, they used different strategy, take different path, but reached a same goal. If no specific requirement for a precise optimization, Adam method can be a pretty good choice. But Adam cannot be always satisfied with all the situations. We can also make a better use of an optimization algorithm by fine-tuning their control hyperparamters as long as we have a better understand about the data.

For my best knowledge, even though Adam works well in lots of models, but scientists and engineers are still using stochastic gradient descent methods. I became curious about this phenomenon and did a few paper reviews on discussions about using Adam in all sorts of tasks in machine learning.

1. Adam may not converge:

   A conference paper in ICLR discussed the convergence of Adam [5], and they pointed out Adam cannot converge by listing some counter-examples. Stochastic gradient descent doesn't use second order momentum, so the learning rate can be fixed. But stochastic gradient descent always comes with learning rate decay strategies, hence the learning rate can be decreasing.

However, the Adam accumulate the momentum with some certain time slots. If the data are varying within a period of time, the moment can be vibrating instead of monotonically decreasing. This situation can make learning rate unstable so that the algorithm can be not converged.

This paper gives the solutions for this case, and it can manipulate the second order momentum to avoid learning rate oscillations [5]:

$$V_n = \max\left(\beta_2 \cdot V_{n-1} + (1-\beta_2) \cdot g_t^2,\ V_{n-1}\right) \tag{28}$$

After the correction, it can be guaranteed that the learning rate is monotonically decreasing.

2. Adam can miss the global optimum:

   Deep learning involves with a large number of parameters, which is in a pretty high-dimensional spaces. The non-convex objective function tends to rise and fall, with numerous uplands and depressions. There are lots of peaks, which can easily be crossed by the momentum terms. For some plateaus, algorithm cannot found after many steps and they just stop training.

   Different algorithms gives a large variety of optimum solutions. The adaptive learning rate method can over-fit the former shown-up features, but for the features observed later after, it can be too hard to completely eliminate the over-fit.

   Researchers tested these algorithms on CIFAR-10 dataset, they found that Adam converges very fast, but the optimum derived from Adam can be worse than that from stochastic gradient descent. This is because the learning rate can be really small at end which slowed the convergence speed. This problem can be solved by control the lower bound of the learning rate [6, 7].

   After review these papers, they all used some counter-examples to demonstrate the effectiveness of Adam, which really remind me that the algorithms have all sorts of advantages, but it is always better to get fully understand the data formation. As seen from the optimization history, all optimization algorithms are based on an initial assumption of the data. The performances can be determined if these assumptions are satisfied or not.

Before introducing the Newton method (second order), it is necessary to summarize the first order method we already discussed. For sparse datasets, it is better to use the adaptive optimization algorithms, and these method provide faster convergence when training deeper networks. Stochastic gradient descent requires longer training procedure, and it will perform reasonable when use proper initialization and learning rate adjustment. AdaGrad, RMSProp, and Adam are relatively similar method.

We are curious about why not use higher order optimization algorithm to training the networks. Next Newton method with second order derivative will be discussed.

9. Newton Method

Newton method approximately solve the function in real or complex domain, more specifically, this method use the front terms of the Tylor expansion of the function to search solutions. The second order Tylor expansion of the loss function can be:

$$J\theta \approx J(\theta_0)+(\theta-\theta_0)^T\nabla_\theta J(\theta_0)+\frac{1}{2}(\theta-\theta_0)^T H(\theta-\theta_0) \tag{29}$$

And the solution of this function for the optimum can be:

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0) \tag{30}$$

If the loss function is a convex function, the update procedure can be:

$$g=\nabla(\theta)|_{J(\theta)} \tag{31}$$

$$H=\nabla^2(\theta)|_{J(\theta)} \tag{32}$$

$$\theta := \theta - H^{-1}g \tag{33}$$

Where *H* is Hessian matrix, which is a square matrix of second-order partial derivatives of loss function. This method are suitable for the condition when the Hessian matrix is positive-definite. If the loss function is convex, we set its gradient to zero:

If the Hessian is positive-definite, eigenvalues are positive, which indicates the local minimum; if the Hessian is negative-definite, eigenvalues are negative which indicates the local maximum; if eigenvalues consist of both positive and negative values, it corresponds to the saddle point; if the Hessian is semi-definite, some eigenvalues can be zero. In this case, it cannot tell which point is minimum or saddle.

Using Newton method to figure out minimum, only if the Hessian is positive-definite. Newton method has second order convergence, and compared with first convergence methods, it will be much faster. Gradient descent methods chose the steepest step to update, and the Newton method not only take the current steepest direction, but also observe if the gradient would be larger after this step. Thus the Newton thinks more globally. Geometrically, Newton method is to use a quadric surface to fit the local surface of the current location, and the gradient descent method is to use a plane to fitting the current local curved surface. Usually, the quadric surface fitting is better than the flat plane, so Newton method's choice of descent path is more accord with the real optimal path.

Advantages:

(1) Second order properties ensure a faster convergence speed.

(2) Perform more stable training procedure than that in gradient descent method.

(3) No need for learning rate hyperparameters, can be better than what first order methods did.

Disadvantages: computing or inverting the Hessian matrix in its explicit form is a very costly process in both space and time.

In order to get rid of the massive computations in Hessian inversion, a large variety of quasi-Newton methods have been developed that seek to approximate the inverse Hessian. The most popular Quasi-Newton method is L-BFGS, which uses the information in the gradients over time to form the approximation implicitly, hence the full matrix is never computed.

In practice, it is currently not common to see L-BFGS or similar second-order methods applied to large-scale Deep Learning and Convolutional Neural Networks. Instead, gradient descent methods based on momentum are more standard because they are simpler and scale more easily. So after review some blogs and papers, I want to elaborate the reasons which resulted in this case:

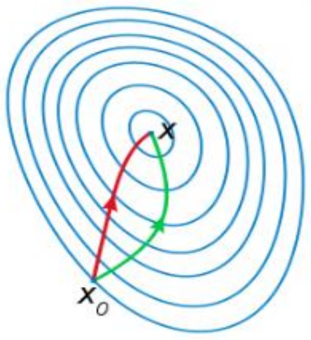

Figure 4. Iteration trajectory. Red is for Newton method, and green is for gradient descent. Source: https://blog.csdn.net/acelit/article/details/63685878

- Scientist and engineers have discussed about the feasibility of the second order methods utilized in the deep learning. Consider an n-dimensional optimization problem, the complexity of gradient descent method is $O(n)$, quasi-Newton is $O(n^2)$, and Newton is $O(n^3)$ in a single iteration [4]. Even though the Newton methods need less iteration steps, it cannot offset the computational expenses spend on each single iteration. Recently some research institutes have shifted their strategies from second to first order method with the popular of large-scale data, but some researchers are still trying to reduce the complexity of second order methods.
- In machine learning, especially in deep learning domain, higher precise solutions are not always necessary. Ignoring the constant coefficient, the generalization error in machine learning model can be decomposed into a statistical error which depends on the optimal solutions and an optimization error. The statistical error is determined by model selection, data distribution, and data size, hence this error are independent to the optimization part. When statistical error are much larger than the optimization error, the benefits brought by the optimization can be negligible, and this can be one part of the difference between machining learning and other applications. Second order method such as Newton can improve the optimization precision, but these superiority cannot be unfolded under machine learning applications.
- Last consideration is the robust. In general, the simpler, more robust for optimization algorithms. Gradient descent methods are easy to implement, and the results cannot be worse if properly initialized the learning rate. When using second order methods, we need to concern about the numerical stability, etc.

V. SIMULATIONS

In this part, I will implement some selected optimization algorithms from previous discussion. I designed the simulations to solve different problems so that I can intuitively understand and manipulate each optimization algorithm. I am not using the API functions from the library.

Case 1: Some gradient-based optimization algorithms are simulated and compared under the circumstance of solving one variable non-constraints optimization problem. Batch gradient descent, AdaGrad, RMSProp and Adam algorithm are implemented on a function.

Figures 5, 6, 7, 8 shows the data update trajectory along the function using each optimization algorithms. The initialization was set to be x = -1.2, and the global learning rate was set to be 0.001.

From the results, all the algorithm are successfully converged to a local minimum, and the iteration steps along with the optimum point derived from each algorithm are demonstrated in the Table 3. Please note that in order to present with a better visualized effect, I only plot a linearly spaced points within the whole converging trajectory.

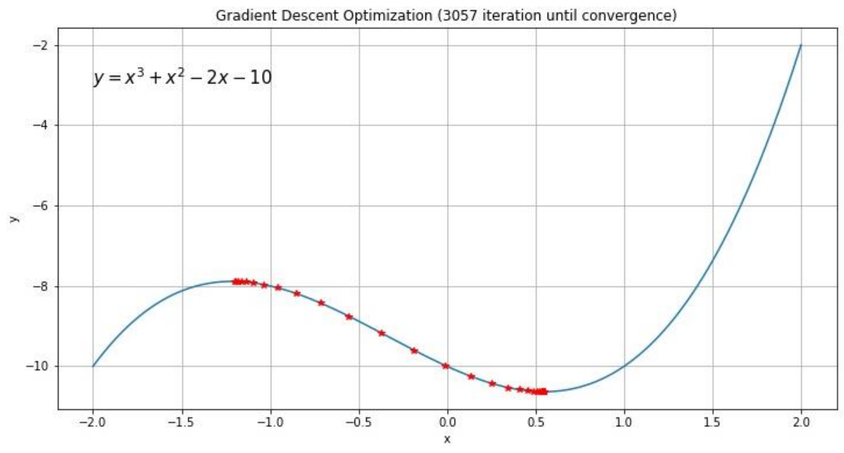

Figure 5. Gradient descent algorithm to optimize a linear function.

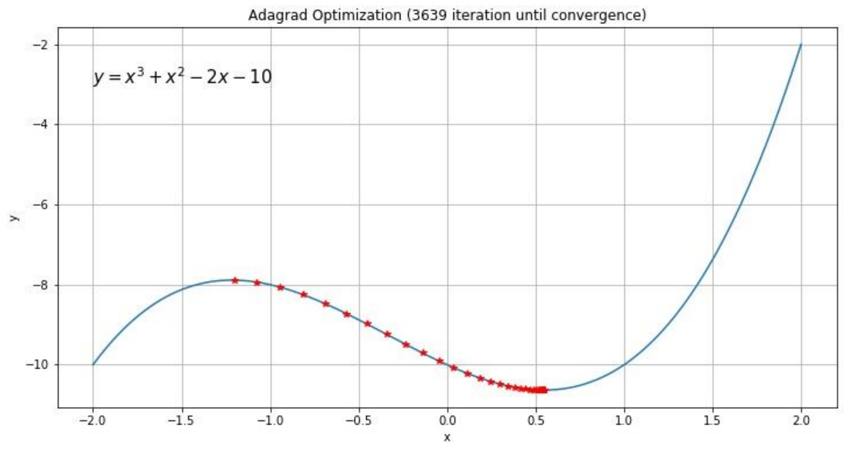

Figure 6. AdaGrad algorithm to optimize a linear function.

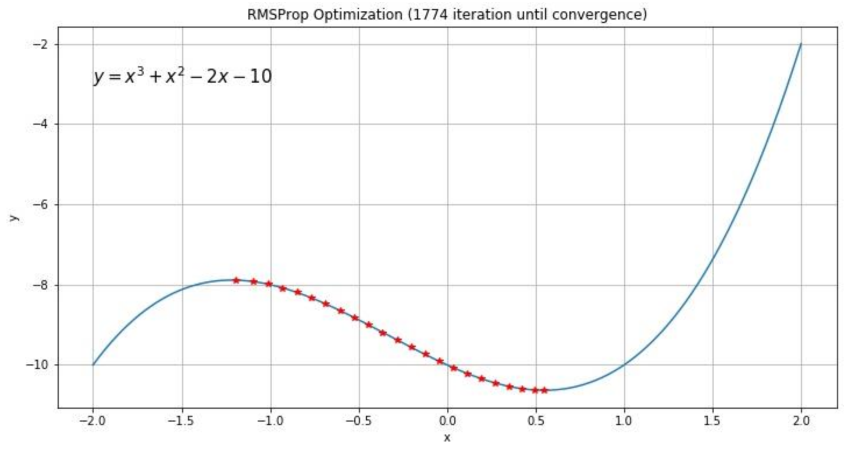

Figure 7. RMSProp algorithm to optimize a linear function.

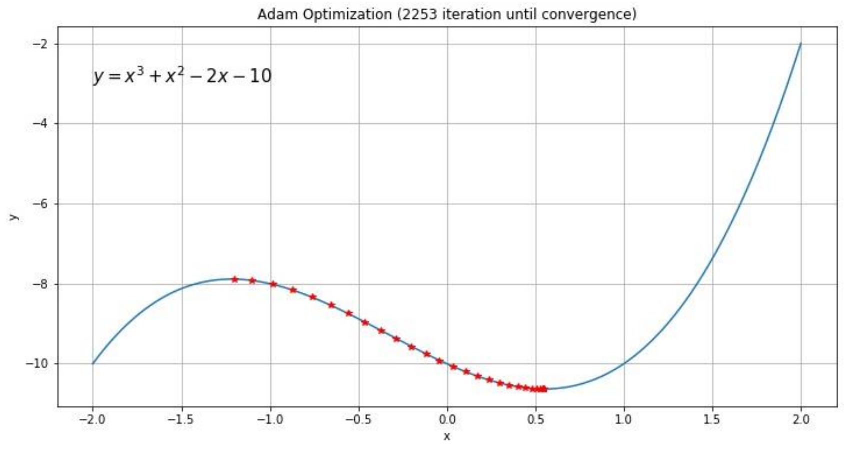

Figure 8. Adam algorithm to optimize a linear function.

To explore how learning rate can affect the learning convergence, it has also simulated with a more aggressive hyperparameter, with the same initialization location but the learning rate was set to 0.1 for this case. Table 4 recorded the simulation results in this case, but the learning trajectories are not presented.

These are low-dimensional problem. As can see, these algorithms are successfully converged to the local minimum with different iterations. In general, their performance are good at this case. The fastest convergence was given by RMSProp, but there are no significant gap between each algorithm. I tried to change the learning rate for each case, and they are all converged but the iteration number can vary a lot, which corresponds to what we discussed in the previous part.

When the initial learning rate is 0.1, the Adam algorithm will oscillate around the optimum which made the model not that confident to converge to that point, which results in more iteration steps to get the optimum.

	Iteration step	*Initialization*	*Optimum*
Gradient descent	3057	(-1.2, -7.9)	(0.54856, -10.6311)
AdaGrad	3639	(-1.2, -7.9)	(0.54856, -10.6311)
RMSProp	1774	(-1.2, -7.9)	(0.54857, -10.6311)
Adam	2253	(-1.2, -7.9)	(0.54856, -10.6311)

Table 3. Optimization results comparison for linear function case. Note that the global learning rate was 0.001.

	Iteration step	*Initialization*	*Optimum*
Gradient descent	28	(-1.2, -7.9)	(0.54857, -10.6311)
AdaGrad	33	(-1.2, -7.9)	(0.54857, -10.6311)
RMSProp	20	(-1.2, -7.9)	(0.54858, -10.6311)
Adam	160	(-1.2, -7.9)	(0.54849, -10.6311)

Table 4. Optimization results comparison for linear function case. Note that the global learning rate was 0.1.

Case 2: In this case, I designed to experiment on a two-variable function to verify the performance of the optimization algorithms: steepest descent, Adam, Newton method are simulated. The learning performance are presented as the form of parameter update trajectories.

The objective function I used for this case is Rosenbrock function, as an analytical form:

$$y = 100\left[\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2\right] \tag{33}$$

Obviously the optimum can be x_1 = 1, and x_2 = 1. The comparison can be illustrated in Figure 9 and Table 5.

Newton method gives significant convergence in this case, and the Adam provides a convergence which is better than the gradient descent provides in this case. In order to verify that these optimization algorithm can provide similar and robust solutions in different cases, I changed the initialization of the beginning position to check the learning trajectories, at meanwhile, the parameters are the same as Case1.

Case 3: In this case, I designed to apply the optimization algorithm for a more practical and more complicated model: hand-written number recognitions. I used to Tensorflow, which an open-source software library for machine learning tasks, to implement a neural network. In this case, the data is from the MNIST hand-written images with 28 by 28 pixels for each one. The MNIST contains 55000 images for training, 5000 for cross-validation and additional 10000 images for testing. For the input layer I chose the nodes of 784 in order to match the input data pixel-wise.

For the initialization of the hyperparameter for optimization procedure, I chose batch size to be 100, hidden layer of size 500, 0.001 for global learning rate, and 0.9 for the momentum factor. The mini-batch gradient descent, stochastic gradient descent with momentum, AdaGrad, RMSProp, and Adam have been implemented. I plot the curve of validation error over iterations to compare the optimization performance and the convergence. And the results are demonstrated in Figure 11, 12.

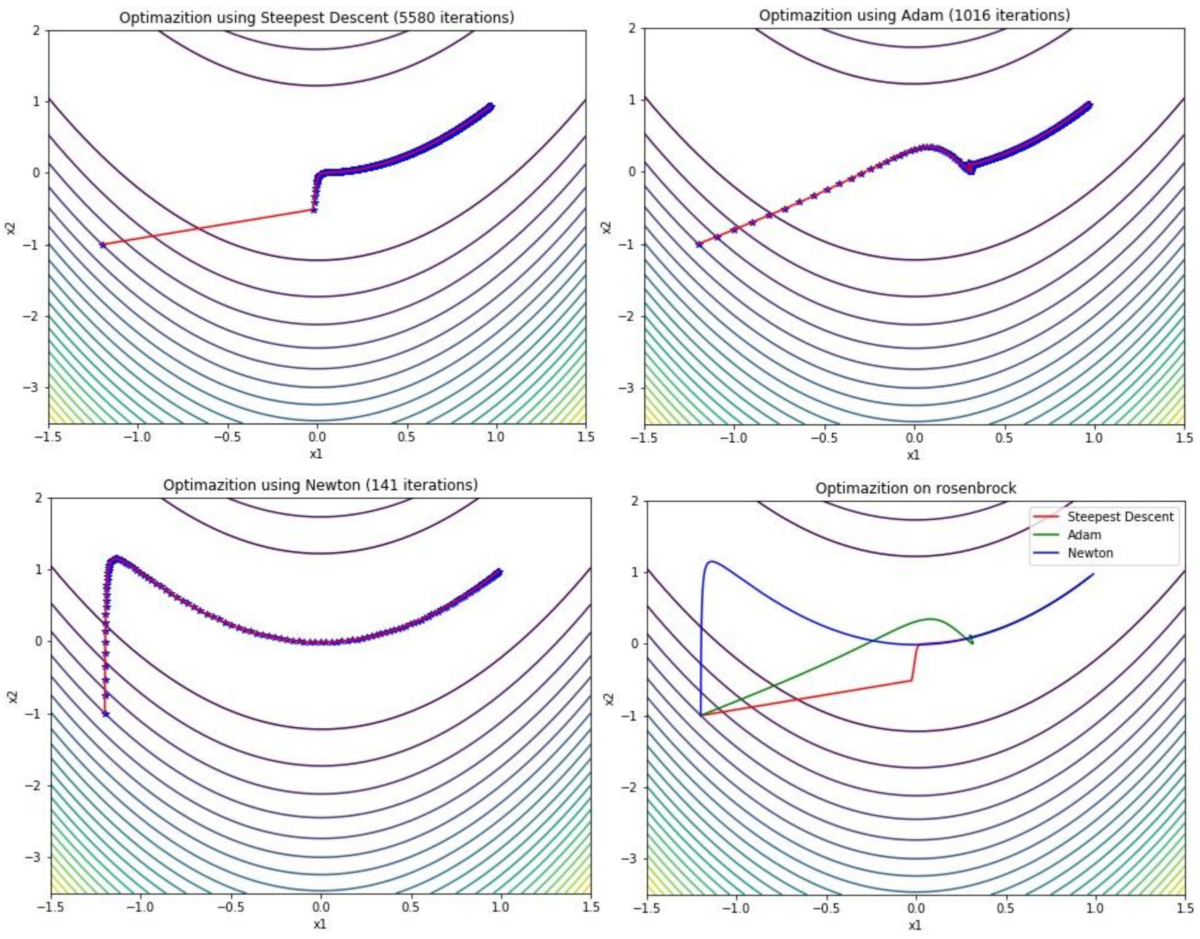

Figure 9. Learning trajectory of optimization algorithms tested on Rosenbrock function. Top-left is for gradient descent, top-right is Adam, bottom-left is for Newton method, and the bottom-right is the displayed trajectories of three. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01.

	Iteration step	***Initialization***	***Optimum***
Gradient descent	5580	(-1.2, -1)	(0.9656, 0.9322)
Adam	1016	(-1.2, -1)	(0.9674, 0.9359)
Newton	141	(-1.2, -1)	(0.9874, 0.9748)

Table 5. Optimization results comparison for Rosenbrock function. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01.

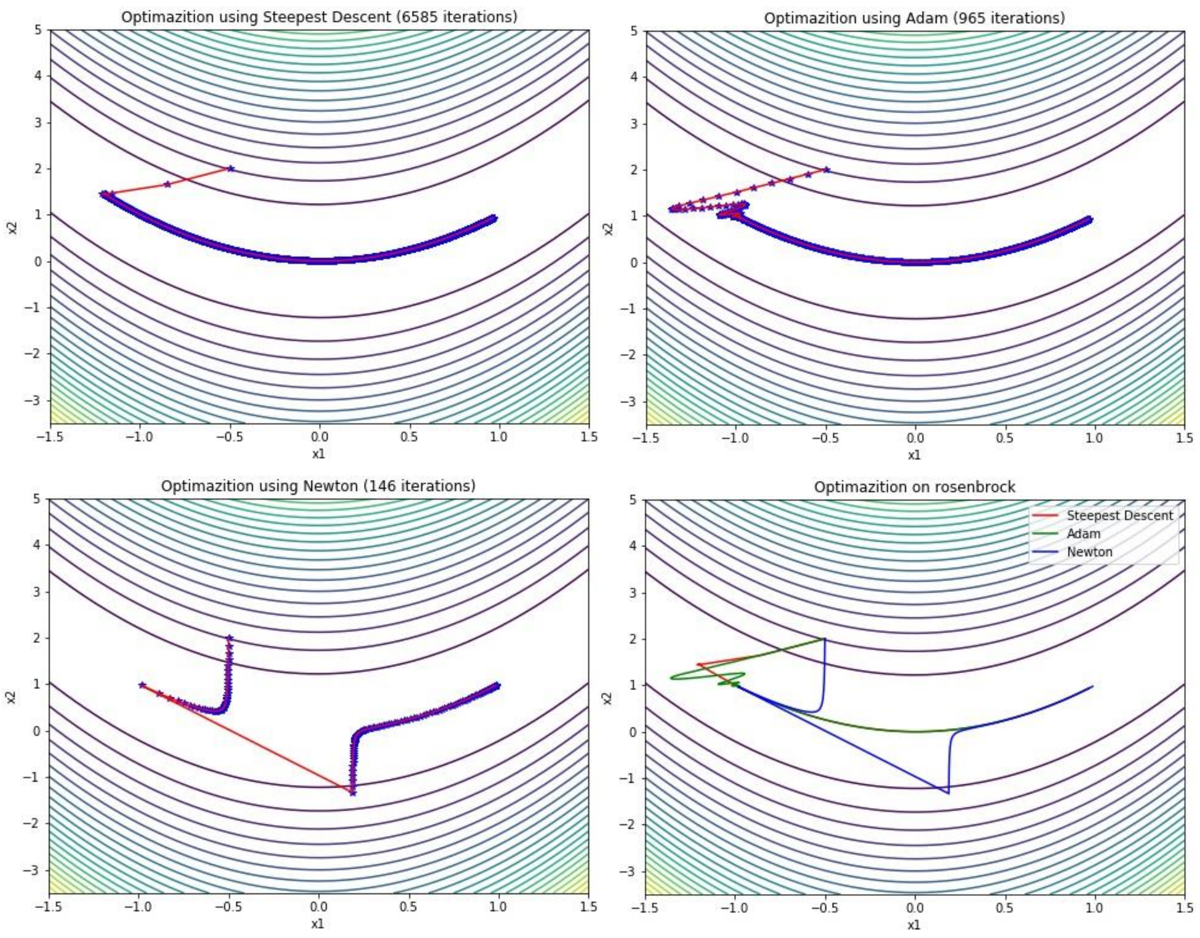

Figure 10. Learning trajectory of optimization algorithms tested on Rosenbrock function. Top-left is for gradient descent, top-right is Adam, bottom-left is for Newton method, and the bottom-right is the displayed trajectories of three. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01.

	Iteration step	***Initialization***	***Optimum***
Gradient descent	6585	(-0.5, -2)	(0.9656, 0.9322)
Adam	965	(-0.5, -2)	(0.9667, 0.9344)
Newton	146	(-0.5, -2)	(0.9873, 0.9746)

Table 6. Optimization results comparison for two variables function. Note that the initializations are different. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01.

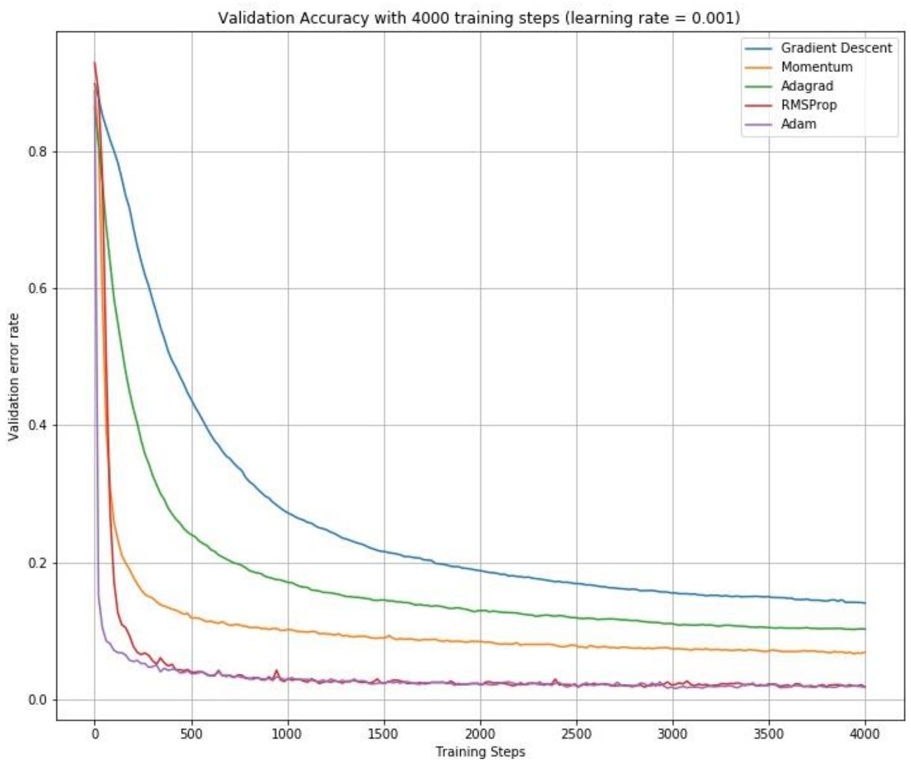

Figure 11. Optimization algorithms performance in a neural network, with MNIST dataset.

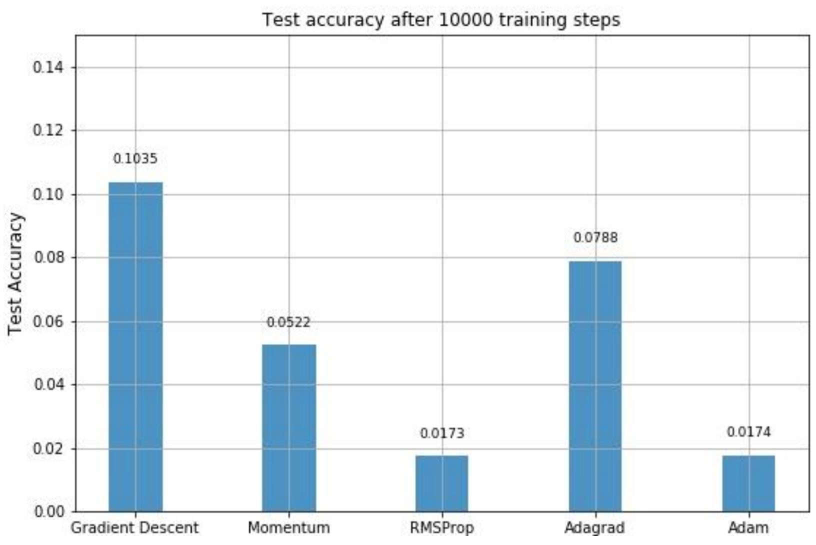

Figure 12. Error residue of optimization algorithms for training in MNIST dataset.

As we can see the performance in this case, Adam gives the best effects among these algorithms. Adam converged very fast and achieved a higher accuracy in validation and the curve is vibrating with a little magnitude. RMSProp can get the almost the same validation errors as Adam did, but this in case RMSProp reacted a little bit slower than Adam. Mini-batch gradient descent converged with a slower rate, and it leave a relative large validation error residual. This can be caused by the aggressively decay of the learning rate so that the small learning rate make the convergence slow or even stop. The AdaGrad and momentum method performed in the middle.

It is interesting that in machine learning, even though Adam works well in many cases, scientists and engineers are still using gradient descent methods in their model. So next, I will talk more about some details and researchers' experience for Adam algorithm.

VI. DISCUSSION & CONCLUSION

From my simulation results, the performance of the Adam is basically similar to what I reviewed in the references. Another optimization algorithms were also implemented to compare the performance. Expected performance have proven that my understanding and implementation were acceptable.

Optimization algorithms are differ from the direction to take gradient descent, and the descent directions can results in the situations that different local optimum are provided by different algorithms for a same problem.

Personal Contribution:

The pros and cons of different optimization algorithms are still a controversial topics. According to what I have concluded in papers and various communities, the adaptive learning rate algorithm, such as Adam, has the advantage of sparse data and have a fast convergence rate. However, the stochastic gradient descent (with momentum) often leads to better final results if the parameters are properly fine-tuned.

Here, I make an assumption that why not combine these algorithms together: by using Adam first to get a faster convergence, and then use the gradient descent method to find the better final results. In this way, we can minimize the disadvantages for both of them. I also stimulated this algorithm, and the results shows in Table 7 and 8. The iteration steps has been significantly dropped compared to the gradient descent method, but it guarantee to converge to a minimum as the property of gradient descent. The optimal results seems almost the same, and this may be our model is not that complicated.

Another problem has been popped out: what is the proper time to make algorithm handover? This cannot be ignored since the convergence and performance should be guaranteed while the resources consume needs to be limited or optimized. Right now, I just leave this problem behind, and it can be the issues covered in future works.

For machine learning tasks, in order to a model well, we do need to choose a proper optimization algorithm, get familiar with the data structures, experiment on small batch of data, have a proper learning rate decay strategy, and keep eye on the training and validation errors for monitoring the descent direction, learning rate and overfitting.

	Iteration step	***Initialization***	***Optimum***
Gradient descent	6585	(-1.2, -1)	(0.9656, 0.9322)
Adam	965	(-1.2, -1)	(0.9667, 0.9344)
Gradient descent + Adam	2467	(-1.2, -1)	(0.9656, 0.9324)

Table 7. Optimization results comparison for two variables function. Note that the initializations are different. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01. (Combination Method used 900 iterations of Adam first, and 1567 for gradient descent).

	Iteration step	***Initialization***	***Optimum***
Gradient descent	6585	(-0.5, -2)	(0.9656, 0.9322)
Adam	965	(-0.5, -2)	(0.9667, 0.9344)
Gradient descent + Adam	2049	(-0.5, -2)	(0.9656, 0.9322)

Table 8. Optimization results comparison for two variables function, with different initialization. Note that the learning rate of gradient descent was 0.001, and the initial learning rate for Adam was 0.01. (Combination Method used 900 iterations of Adam first, and 1149 for gradient descent).

After this Professor Segray Loyka’s lecture and project in optimization field, I did benefit a lot. I learned the fundamental principles used in convex optimization, how to recognize a problem is convex or not, and how to convert the constrained problem into unconstrained problem to solve by using Lagrange multiplier and KKT conditions. I also have a big picture in the optimization methods used in machine learning domain, which excited me since I have better understand the convergence of the learning model. Moreover, by simulating the problems, I read many source codes in optimizer library, and now I have my own version of a sort of optimization codes in Python which can be implemented in my future work.

MAIN REFERENCES

[1] Bottou, Léon. "Stochastic gradient descent tricks." *Neural networks: Tricks of the trade*. Springer, Berlin, Heidelberg, 2012. 421-436. (Cited by 483)

[2] Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013. 1139-1147 (Cited by 987)

[3] Kinga D, Adam J B. "Adam: A method for stochastic optimization." *International Conference on Learning Representations*. 2015. (Cited by 7550)

SUPPORTIVE REFERENCES

[4] Curtis, Frank E., and Katya Scheinberg. "Optimization Methods for Supervised Machine Learning: From Linear Models to Deep Learning." arXiv preprint arXiv:1706.10207 (2017).

[5] Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. "On the convergence of adam and beyond." International Conference on Learning Representations. 2018.

[6] Wilson, Ashia C., et al. “The marginal value of adaptive gradient methods in machine learning.” Advances in Neural Information Processing Systems. 2017.

[7] Keskar, Nitish Shirish, and Richard Socher. “Improving Generalization Performance by Switching from Adam to SGD.” ArXiv preprint arXiv:1712.07628 (2017).