Examining Approaches for escaping saddle points in non-convex optimization

Binoy Thomas
Department of ECE
Raleigh, NC
bthomas7@ncsu.edu

Rajaram Sivaramakrishnan
Department of ECE
Raleigh, NC
rthiruv@ncsu.edu

***Abstract*—A lot of research has been done on finding a global optimum value of a function. Many first order methods give very good results but their performance deteriorates because these methods get stuck in saddle points most of the time. Modern machine learning problems need local minimas in order to solve them. The main obstacle to solving these problems is that the loss function contains a lot of saddle points. Saddle points are surrounded by high error planes that can dramatically slow down learning and gives the idea of the existence of a local minimum. Another problem is the increase in dimensions. As the dimensions increase, the number of saddle points increase. In this paper we look at the various stochastic and deterministic methodologies to ensure that the algorithm is not stuck in a saddle point and is efficiently able to escape these points. We look at recent research being done on escaping saddle points and make a comparison on the efficiency. complexity and approach of these methodologies.**

***Index Terms*—Saddle points, Adaptive gradient methods, Second order approximation, PGD, SGD, NGD, Hessian, Gradient.**

I. Introduction

Non-convex optimization is an important part of modern machine learning problems. Finding the global minima is difficult but most problems just require the local minima for finding the solution. A simple and fundamental algorithm is the Gradient Descent algorithm. This algorithm converges to the first order stationary point within $(1/\epsilon^2)$. [4]

Dimensions do not have any effect on the convergence of the algorithm. Hence, gradient descent is a very useful algorithm for higher dimension problems. [3] Deep learning being on of them. The gradient descent algorithm only takes gradient information into consideration. Hence, it is very prone to getting stuck in saddle points. The region which satisfies the initial conditions is called as the attraction zone.

A saddle point is a point on a function that is a stationary point but is not a local extremum. Also called minimax points, saddle points are typically observed on surfaces in three-dimensional space but also occur in lower or higher dimensions. [10] The first and second derivative tests can often be used to distinguish between saddle points and other types of stationary points. We will look at critical points and their types and how to distinguish between them.

Optimization of a function in NP is very hard in general. There are two reasons for this. The first is that a non convex function can have many local minima and it is difficult to find the best one i.e. the global minima. [8] The other reason is that finding the local minima is also hard because of the large number of saddle points that are present in the optimization process. These points have zero gradient but are not local minimas. There is no algorithm that will guarantee the global minima in a fixed number of polynomial steps.

This problem is accentuated in deep learning problems. Most deep learning problems are highly dependent on gradient information since most of them are gradient based algorithms. The saddle point problem increases for second order problems that rely on the Hessian information.

In the course of this project, we worked on improving our knowledge of non-convex optimization. The concepts and theories we learnt in the 'Optimization and Algorithms' course helped us a lot in the completion of our project. We looked at Gradient based methods to escape saddle points. Saddle points are basically those points that cause a problem to finding the optimal point of functions. They can cause an issue whenever we want a function to converge to a local or global minima. Escaping saddle points is an ability that all of us need as machine learning engineers. This was one of our main objectives for this projects and we have successfully studied and learnt thoroughly the various methods and researches that have been done and although comparing all of them was not very easy, we now know which ones we can use depending on the kind of optimization, the speed requirements and the CPU/GPU constraints.

The paper contains information about saddle points and how to escape them. Firstly, we study what exactly saddle points are and their types. Saddle points are critical points which are neither a maxima nor a minima but the gradient at this point is 0. Then we look at the problems that are observed in non-convex algorithms and what role the number of dimensions have on the convergence of the algorithm.

A brief idea of the methodologies that are used to escape saddle points and a comparative study of Hessian and Gradient based algorithm follows. Hessian based solutions rely on computing the Hessian of the function and get the value of the second order derivative whereas gradient based methods rely on computing the gradient of the function.

Next we look at different research papers that are available for escaping saddle points. We look at a different way of looking at Newton method in which there is a slight change in the traditional Newton method so as to give a simple way of

escaping these saddle points. Stochastic gradient method for tensor decomposition is used for low dimension problems and uses the noisy stochastic gradient for the purpose of escaping saddle points.

Adaptive Gradient methods have long to be known as the fastest and the easiest methods to deal with saddle points. We look at the various Adaptive gradient methods and try to understand which one is better in finding the minima and make a comparison of their responses. The adaptive gradient methods are used for neural network models. The models were trained using the MNIST training data and were tested for various patterns.

II. Related Work

There has been much recent work on developing algorithms which provably avoid saddle points. Traditionally, only second-order optimization methods were known to converge to second-order stationary points. These algorithms rely on computing the Hessian to distinguish between first- and second-order stationary points. Nesterov and Polyak [1] designed a cubic regularization algorithm which converges to an ϵ-second-order stationary point in $l(f(x0) - f\star)$ / ϵ^2 iterations. Trust region algorithms curtis et al., [2] can also achieve the same performance if the parameters are chosen carefully. These algorithms typically require the computation of the inverse of the full Hessian per iteration, which can be very expensive. Second-order algorithms: The most popular second-order method is the Newton's method. Although Newton's method converges fast near a local minimum, its global convergence properties are less understood in the more general case. For non-convex functions, Frieze et al., [3] gave a concrete example where second-order method converges to the desired local minimum in polynomial number of steps. As Newton's method often converges also to saddle points, to avoid this behavior, different trusted-region algorithms are applied Dauphin et al., [4]. The recent work e.g. chen et al., [5] tries to understand and give theoretical guarantees for adaptive methods such as Adam and RMSProp. Second, the technical developments in using first-order algorithms to achieve non-convex second-order convergence e.g. Ge et al., [6].

Typically, to escape from saddles one has to use second-order methods. However, most works on second-order methods rely extensively on expensive Hessian-based computations, making them impractical in large-scale settings. The existing algorithms use either the perturbed/noisy gradient or partially or fully the Hessian information. Since the computation of the Hessian is often too expensive in practice, algorithms without second-order information are very desirable. Ge et al., [7] proved that stochastic gradient descent (SGD) can find local minima of strict saddle functions in polynomial time. Levy [8] showed that noisy normalized gradient descent can converge faster than SGD. Jin et al., [9] proposed the perturbed gradient descent (PGD) and showed that, with an additional cost depending on the problem dimension, PGD will converge to an second-order stationary point with high probability. We will discuss PGD in the later section. By combining PGD with the heavy ball method, Jin et al., [10] showed that PGD with momentum can converge faster. Du et al., [11] showed that PGD or noisy GD can escape saddle point faster.

III. Preliminary ideas for optimization

Convex problems are simple and usually have just a single critical point. Non-convex problems have a large number of critical points that are encountered. First we will talk about the classification of critical points.

A. Critical points

Here, we consider a function whose first three order derivatives exist. [2] We represent the derivatives as $\nabla f(x) \in R^n$, $\nabla^2 f(x) \in R^{n*n}$, $\nabla^3 f(x) \in R^{n*n*n}$, where

$$[\nabla f(x)]_i = \frac{\delta}{\delta x_i} f(x) \tag{1}$$

$$[\nabla^2 f(x)]_{i,j} = \frac{\delta^2}{\delta x_i \delta x_j} f(x) \tag{2}$$

$$[\nabla^3 f(x)]_{i,j,k} = \frac{\delta^3}{\delta x_i \delta x_j \delta x_k} f(x) \tag{3}$$

For such smooth function $f(x)$, we say x is a critical point if $\nabla f(x) = 0$. Traditionally, critical points are classified into four cases according to the Hessian matrix:

1) (Local Minimum) All eigenvalues of $\nabla^2 f(x)$ are positive.
2) (Local Maximum)All eigenvalues of $\nabla^2 f(x)$ are negative.
3) (Strict saddle) $\nabla^2 f(x)$ has at least one positive and one negative eigenvalues.
4) (Degenerate) $\nabla^2 f(x)$ has either non-negative or non-positive eigenvalues, with some eigenvalues equal to 0.

B. Second Derivative Test

Let $z = f(x, y)$ be a function of two variables for which the first- and second-order partial derivatives are continuous on some disk containing the point (x_0, y_0). Suppose $f_x(x_0, y_0)$=0 and $f_y(x_0, y_0)$=0. [20]

$$D = f_{xx}(x_0, y_0) f_{yy}(x_0, y_0) - (f_{xy}(x_0, y_0))^2 \tag{4}$$

Then

1) If D >0 and $f_{xx}(x_0, y_0)$ >0, then f has a local minimum at (x_0, y_0).
2) If D>0 and $f_{xx}(x_0, y_0)$ <0, then f has a local maximum at (x_0, y_0).
3) If D<0, then has a saddle point at (x_0, y_0).
4) If D=0, then the test is inconclusive

C. Stochastic Gradient Descent

The Stochastic gradient descent algorithm aims to solve the stochastic optimization problem. [9]

$$w = argmin f(w), where f(w) = E_{x \in D}[\phi(w, x)] \quad (5)$$

$\phi(w, x)$ denotes the loss function denoted for x at the point w. The algorithm follows a stochastic gradient

$$w_{t+1} = w_t - \eta \delta_{wt} \phi(w_t . x_t) \quad (6)$$

where x_t is a sample from the distribution and η is the learning rate.

Stochastic gradient can be viewed as given a stochastic gradient oracle, we are trying to optimize an arbitrary function $f(w)$

IV. Saddle points

A typical problem with saddle points is that they are surrounded by small plateaus of curvature in the error. Gradient descent is meant to follow the direction of negative descent, it could be slowed down due to these plateaus. The normal Newton methods rapidly descend these plateaus due to the multiplication of the slope with the inverse of the Hessian matrix. This causes the Gradient algorithms to attract the saddle points. In non convex optimization we cannot justify the theory of normal convex implementations because the Newton-Quasi algorithms get stuck in saddle points [1]

A. Types of Saddle points

Following are some examples of Saddle points for a few functions. [4]

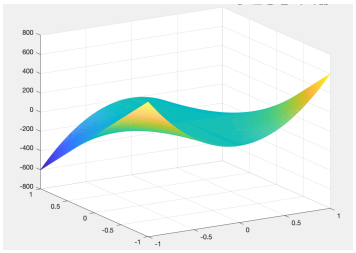

Fig. 1. Monkey Saddle$-3yx^2 + y^3$

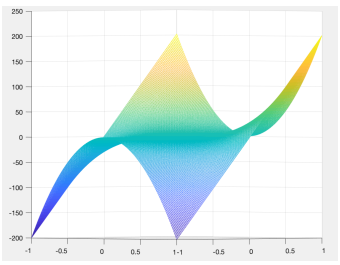

Fig. 2. $-yx^2 + y^2$

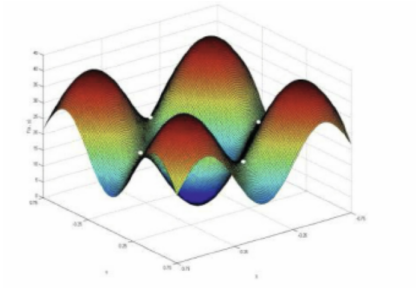

Fig. 3. Rastrigin function with highlighted saddle points

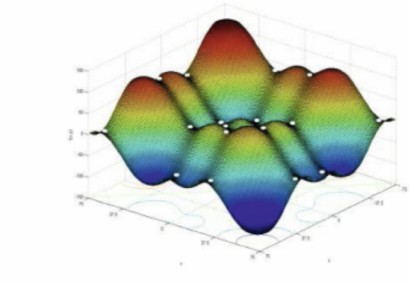

Fig. 4. Schwefel funtion with highlighted saddle points

B. Saddle points in higher dimensions

Saddle points are the main reason for slowing down high level non- convex optimization algorithms. On reviewing the result by Bray et al. [9], it shows how the critical points are distributed in the plane through the ϵ vs α graph, where ϵ is the error and α is the index of negative eigen values at a critical point. This graph shows that there is a strong correlation between α and ϵ. As α increases, the value of ϵ increases. Critical points with error much greater than that of the global minima are most probably saddle points. As the dimensions increase, the probability that the curvature of the graph around the critical points increase, exponentially decrease. Hence, it becomes extremely difficult to decide whether a point is a saddle point or not.

V. Hessian vs Gradient: A brief introduction

A. Hessian Based Solution

These algorithms rely on computing the Hessian, or a second-order derivative matrix, to find second-order stationary points as opposed to first-order stationary points. The idea is that second-order optimization methods converge to second-order stationary points. When the eigenvalues are positive for a given $\nabla * f(x)$ then x must be a local minimum. Otherwise, x may be a local maximum or a saddle point. But because these algorithms rely on computing the Hessian per iteration, they are often times too expensive to be put into practice, as many matrix operations would have to be applied onto a large data set per iteration. This is a major obstacle for many algorithms to be feasible on a larger scale, like in a deep Neural Networks.

B. Gradient Based Solution

Algorithms such as stochastic gradient descent and perturbed gradient descent, uses the gradient of a function and updates its direction of the search of functional minima. For example, the idea behind stochastic gradient descent is that, given only the gradient of the function, randomly adding noise to a point is enough to escape a saddle point. This is often efficient and less expensive to randomly calculate a noisy gradient than to calculate a true gradient, and it is obviously less expensive than computing the Hessian matrix.

VI. Methods to Deal with Saddle points

A. Adding Noise/Perturbations

This paper Rong Ge et al., [12] suggests a simple variant of stochastic gradient algorithm, where the only difference to the traditional algorithm is we add an extra noise term to the updates. The main benefit of this additional noise is that they guarantee there is noise in every direction, which allows the algorithm to effectively explore the local neighborhood around saddle points. This paper [4] shows that, as the dimension increases, the time taken to solve the given objective function also increases. Meaning that the time taken to process a function in high dimension eventually falls into the number of dimensions it has to tackle.

Algorithm 1: Noisy Stochastic Gradient

Result: Stochastic gradient oracle SG(w), initial point w_0, desired accuracy K.
for t =$o\rightarrow$ $T-1$ **do**
  Sample noise n uniformly from unit sphere
  $w_{t-1} \leftarrow w_t - \eta(SG(w)+n)$
end

In case of adding perturbations, the paper deals with adding perturbation to the input, before feeding it into the gradient descent. The idea is that by doing this the perturbations make sure that its spread even in all directions to make sure the algorithm explores all directions and escape the saddle point eventually.

Algorithm 2: Perturbed Gradient Descent(Meta algorithm)

for t =0,1,2... **do**
  if perturbations condition holds then
  $x_t \leftarrow x_t + \zeta_t$ where $\zeta_t$uniformly distributed
  $\mathrm{x}_t \leftarrow x_t - \eta\nabla f(x_t)$
end

At each time step, the algorithm checks for several conditions. When the norm of the current gradient is below a certain threshold (which would suggest that we might be close to a saddle point), the algorithm adds a small random perturbation to x_t, where the perturbation is uniformly sampled from $B_o(r)$. If the function value does not decrease enough after a certain threshold of iterations, then the algorithm returns the current value of x_t. This indicates that x_t should be a sufficiently "close" to a second-order stationary point and not a saddle point.

B. Momentum Based Optimization

A very popular technique that is used along with SGD is called Momentum. Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go.

Repeat Until Convergence:

$$V_j \leftarrow \eta * V_j - \alpha * \nabla_w \sum_{l}^{m} L_m(w)$$

$$w_j \leftarrow V_j + w_j$$

where η is the coefficient of momentum, while V_j is the retained gradient.

Here the main idea is to use the previous gradient as an additional parameter to guide the algorithm in a particular direction, hopefully towards minima. We see that the previous gradients are also included in subsequent updates, but the weight-age of the most recent previous gradients is more than the less recent ones. This also builds speed, and quickens convergence process. Essentially as the name suggests, it adds momentum to the gradient which minimizes the objective function.

For the purpose of testing this principle, we decided to test the performance of minimizing the objective function on a toy data-set and plotted the score of loss and the contour plot.

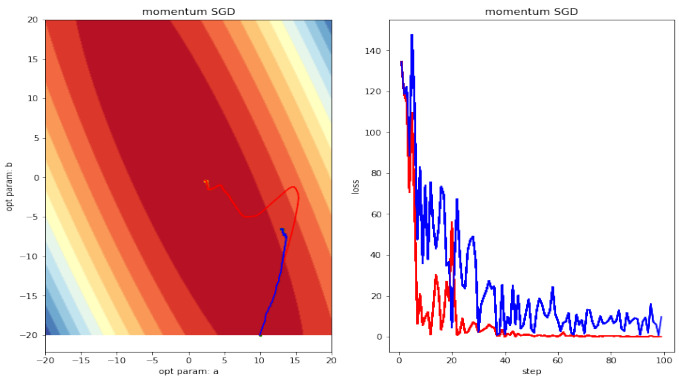

Fig. 5. Objective function and loss curve

From the plot it is clearly observable that, the traditional SGD algorithm, when reaches a saddle point, gets stuck at that point and fails to come out of it. But in case of momentum based implementation, we do not face that issue as the momentum guides the algorithm to reach the minima. This is cross verified by the fact that the loss function is minimized quickly in case of momentum based implementation.

C. Cyclical Learning Rate

The learning rate is the most important hyper-parameter to tune for training deep neural networks. This paper by Leslie Smith [13] describes a new method for setting the learning rate, named cyclical learning rates, which practically eliminates the need to experimentally find the best values and schedule for the global learning rates. Instead of monotonically

decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundary values.

The author demonstrates how CLR policies can provide quicker converge for some neural network tasks and architectures. One example from the paper compares validation accuracy for classification on the CIFAR-10 data-set. One reason this approach may work well is because increasing the learning rate is an effective way of escaping saddle points. By cycling the learning rate, we're guaranteeing that such an increase will take place if we end up in a saddle point.

Our implementation to test out the theory of having a cyclic learning rate is shown below. The paper goes through a exhaustive number of options in terms of choosing the parameter for cyclical learning rate. But we narrowed our focus to only the ones we were able to replicate.

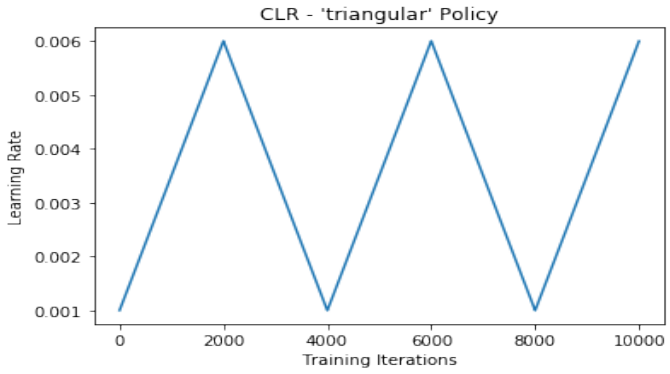

Fig. 6. Triangular cyclic lr

The above plot shows the choice of our learning rate and as the iterations go on, we increase and decrease the learning rate based on the triangular pattern and in cyclic nature.

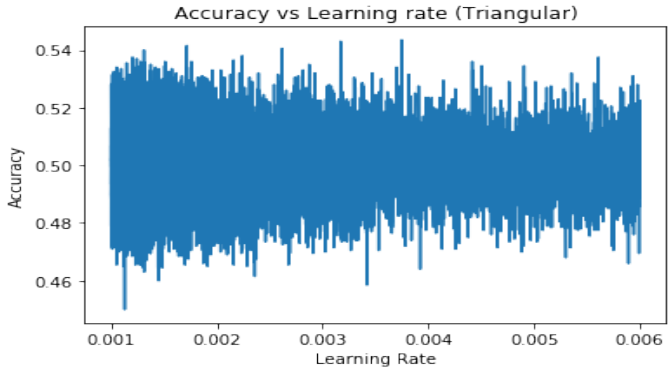

Fig. 7. Learning rate vs accuracy

The accuracy plot for different values of learning rate. Unique interpretation of this plot is that the accuracy value falls at different values of the cycle. Meaning, the second instance the learning rate falls to the same value, the accuracy value falls down.

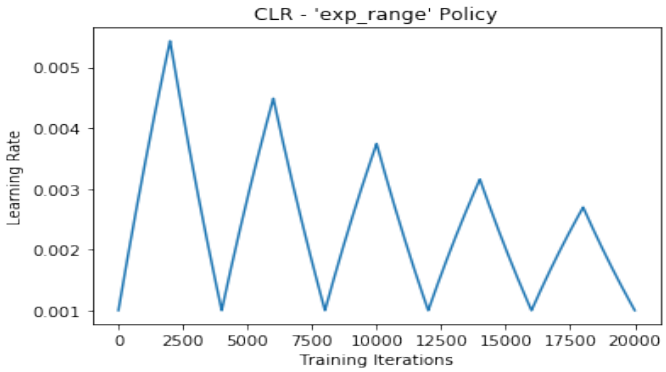

Fig. 8. Implementing Cyclical Learning rate

This plot above shows that by shuffling the learning rate in a pattern that is not recognizable by the learning algorithm, we make sure that there is no learning of the pattern which may yield a biased interpretation. By also incrementally reducing the max lr value of of each of the triangular peak, we test out that at which interval that the algorithm is capable of escaping the saddle point.

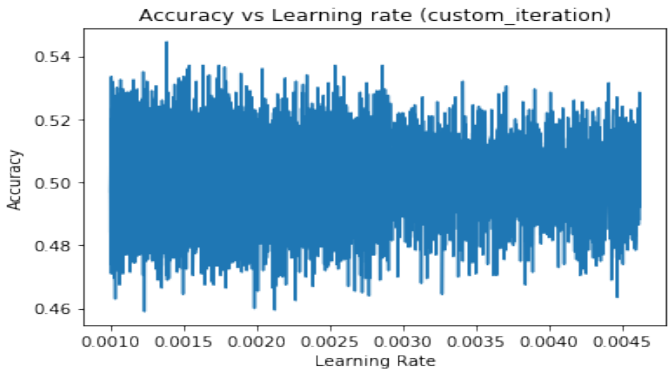

Fig. 9. Learning rate vs Accuracy for custom setting

From the plot, it is clear that there is a decreasing trend in terms of the values of accuracy as we increase the learning rate. This is common in training a machine learning models, but still the funnel shaped pattern shows that repeated implementation at the same learning rate still yields differing values, introducing a randomness in learning.

VII. Research on Escaping Saddle Points

A large amount of research has been done on escaping saddle points based on the methods mentioned in Section VI. We studied these papers and tried to understand and implement their ideas and a brief summary of our work in presented in the subsequent sections.

A. Saddle free Newton Method

This is the research done by Yane et al. [4] at University of Montreal in the year 2014. They argue based on theoretical physics, neural networks, random matrix theory and empirical formulation that there are a lot of issues when it comes to

finding a local minima in deep learning networks. Since these critical points are surrounded by high error plateaus, they have a very low learning rate and when we try to find what type of point it is, we face an issue. Because of these reasons, they tried to propose a new approach to the second order optimization known as the saddle free optimization.

This method can rapidly escape the saddle points unlike the gradient descent and the quasi-Newton methods. The algorithm was applied to a deep learning recurrent neural network and it was observed that this method provided a superior optimization results.

1) Algorithm dynamics near saddle points: Considering that saddle points are present in a function, we have to understand how algorithms behave near them. For the purpose of understanding we focus in saddle points for which the Hessain is singular. These saddle points are non degenerate. Morse's lemma can be used to re-parameterize the function. Morse's lemma is presented below

$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2}\sum_{i=1}^{n_\theta} \lambda \Delta v_i^2 \tag{7}$$

where λ_i represents the ith Eigen value of th e Hessian and Δv_i are the new parameters if the model corresponding to the motion along the eigen-vector e_i of the Hessain of f at θ^*

The theory says that if finding the minima is the aim of the optimization process then an optimal algorithm will move away from the saddle point at a rate that is inversely proportional to how flat the error surface is at the current position.

A step in the gradient descent algorithm always steps in a direction which is closer to the saddle point because of the flatness of the error surfaces surrounding the saddle point.

The main issue with the gradient descent is not the direction in which the algorithm moves, the issue is more related to the step size. Considering equation(7), any step in the direction of e_i is given by $-\lambda_i \Delta v_i$ and hence small steps are taken in the direction of eigenvalues of small absolute values.

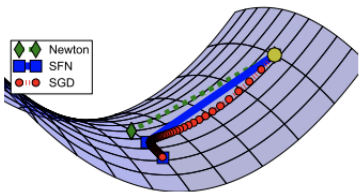

Fig. 10. Behavior of $5x^2 - y^2$ near a saddle point

The slowness problem is solved by the Newton method. The newton method re scales the gradient in each direction by the inverse of the eigen value and hence the new value of the step is given by $-\Delta v_i$

2) The trust region approach: This is an approach used mainly for scaling second order problems to non-convex problems. [2] To attack the problem mentioned above, a generalized trust region method is defined to search for the algorithm within the space. The Taylor series expansion is used for this purpose. $T_k(f, \theta, \Delta\theta)$ indicates the k-th order expansion of f around θ and evaluated at $(\theta + \Delta\theta)$. The generalized trust region method is given by

$$\Delta\theta = argmin T_k(f, \theta, \Delta\theta) with k \in [1, 2] s.t. d(\theta, \theta + \Delta\theta) < \Delta \tag{8}$$

3) Algorithm for saddle free Newton method: The saddle free Newton method is great for escaping saddle points for a given deep learning neural network. The algorithm for this method is given below

Algorithm 3: Approximate saddle free Newton algorithm

Result: Function f(θ) to $minimize$
for i =1$\rightarrow$ M **do**
  $V \leftarrow k$
  $s(\alpha) = f(\theta + V\alpha)$
  $\|H\| \leftarrow \|\frac{\delta^2 s}{\delta \alpha^2}\|$
  by using eigen decomposition of H
  for f =1$\rightarrow$ m **do**
    $g \leftarrow -\frac{\delta s}{\delta \alpha}$
    $\lambda \leftarrow argmin_\lambda s((s(\|H\| + \lambda I)^{-1} g))$
    $\theta \leftarrow \theta + V(\|H\| + \lambda I)^{-1} g$
  end
end

The algorithm known as the saddle free Newton method uses a different approach as expected. It uses the curvature information differently. It doesn't use the Taylor's series expansion as in the classical methods. Unlike the gradient descent, the escape is rapid even along weak negative curvatures.

The saddle free Newton method can be trained or larger networks with large number of hidden layers. This algorithm had previously remained heuristic and theoretically unjustified, as well as numerically unexplored within the context of deep and recurrent neural networks. The research proves that near saddle points rapid escape can be achieved by combining the best of gradient descent and Newton methods. The method avoids the issues related to both. The generalized trust region approach shows that this algorithm is sensible even far from saddle points. The research demonstrated an improved optimization on several neural network training problems.

B. Stochastic gradient for tensor decomposition

This is the research conducted by Rong et al [12]. in the year 2015. It basically looks at problems related to constrained optimization. It looks to solve problems with equality constrains. These constraints are of the form.

$$min f(w) where w \in R^d$$

s.t $c_i(w) = 0, i \in [m]$(9)

This method cant be used for high dimension saddle points since it will fail. The algorithm is presented below.

Algorithm 4: Projected Noisy Stochastic Gradient

Result: Stochastic gradient oracle SG(w), initial point w_0, desired accuracy K

for $t = o \rightarrow T - 1$ **do**

Sample noise n uniformly from unit sphere

$v_{t+1} \leftarrow w_t - \eta(SG(w) + n)$

$w_{t+1} = \prod_w (v_{t=1}$

end

The Projected Noisy Gradient algorithm was applied to orthogonal tensor decomposition. The results obtained in the paper shows that the algorithm converges efficiently and the reconstruction error is better based on this formulation. The problem was formulated for a function with 10 dimensions and a random input tensor T was selected at random with orthogonal decomposition form R^{10}. The reconstruction error is given by

$$\epsilon = (\|T - \sum_{i=1}^{d} u_i^4\|^2)/\|T\|^2 \tag{10}$$

Two ways were used to generate the array samples and these samples were used to do a sanity check on the function. The plots in figure() show the results of running the new objective function. They do not always converge to a low reconstruction error value.

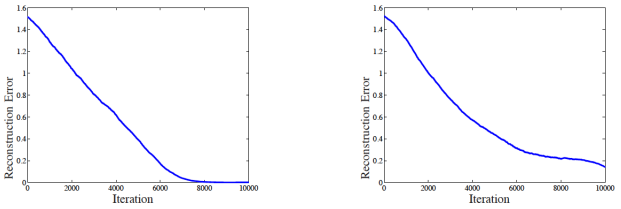

Fig. 11. Comparison of the objective function's reconstruction error

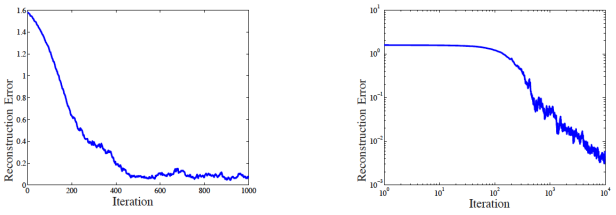

Fig. 12. Comparison with different learning rate

Learning rate also has an effect on the results. At a high learning rate, the error stays at a small value whereas as you decrease the learning rate, the error converges to 0.

C. Summary of Adaptive Gradient descent techniques

1) ADAM: Adaptive Moment Estimation [14] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.

Algorithm 1. Adam Optimization Algorithm

Require: Objective function $f(\theta)$, initial parameters θ_0, stepsize hyperparameter α, exponential decay rates β_1, β_2 for moment estimates, tolerance parameter $\lambda > 0$ for numerical stability, and decision rule for declaring convergence of θ_t in scheme.

procedure ADAM(f, θ_0 ; α, β_1, β_2)

m_0, v_0, $t \leftarrow [0, 0, 0]$ # Initialize moment estimates

and timestep to zero

Begin optimization procedure

while θ_t has not converged **do**

$t \leftarrow t + 1$ # Update timestep

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ # Compute gradient of objective

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ # Update first moment estimate

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (g_t \odot g_t)$ # Update second moment estimate

$\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ # Create unbiased estimate $\widehat{m}_t$

$\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ # Create unbiased estimate $\widehat{v}_t$

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \lambda)$ # Update objective parameters

return θ_t # Return final parameters

Fig. 13. Adam Algorithm

To test this out and see if we can test this algorithm against the SGD algorithm and see if it can tackle the scenario where the algorithms face a saddle point, We played with a toy dataset (linear regression data) and looked at the contour plot to observe the algorithms trying to minimize the objective function.

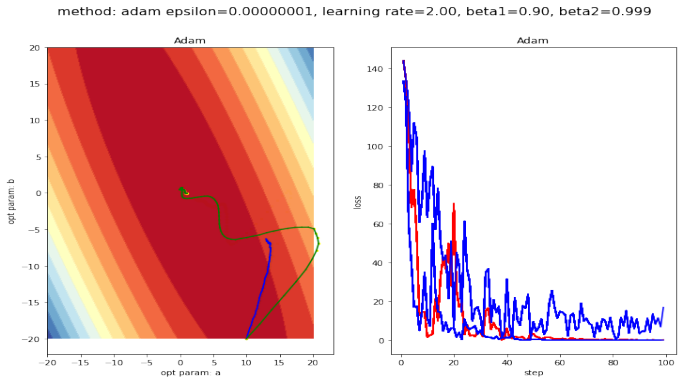

Fig. 14. Objective function and loss curve : Adam

From the looks of the plots, it is clear that the Adam algorithm manages to evade the saddle point based on its design and converge to its minima of the objective function.

2) RMSProp: RMSprop, or Root Mean Square Propagation [15] has an interesting history. It was devised by the Geoffrey Hinton. RMSProp also tries to dampen the oscillations, but in a different way than momentum. RMS prop also takes away the need to adjust learning rate, and does it automatically. More so, RMSProp chooses a different learning rate for each parameter.

$$v_t = \rho * v_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta * w_t = -\frac{\eta}{\sqrt{v_t + \epsilon}} * g_t$$

$$w_{t+1} = w_t + \Delta * w_t$$

In the first equation, we compute an exponential average of the square of the gradient. Then in the second equation, we decided our step size. We move in the direction of the gradient, but our step size is affected by the exponential average. We chose an initial learning rate eta, and then divide it by the average. The third equation is just the update step. The hyper-parameter p is generally chosen to be 0.9, but you might have to tune it. The epsilon is equation 2, is to ensure that we do not end up dividing by zero, and is generally chosen to be 1e-10.

Now continuing with the same procedure of testing this logic against a test data-set and comparing it against the SGD, We get the following plot.

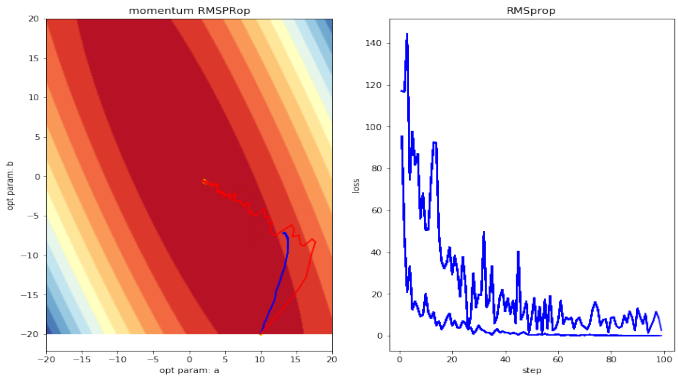

Fig. 15. Objective function and loss curve : RMSprop

The plot clearly shows that while the SGD algorithm fails to minimize the objective function, the RMSprop does this even without us defining the learning rate. Also observe that the algorithm has a zig-zag pattern which is understandable given the nature of the formulas mentioned above.

3) Adagrad: This algorithm [16] adaptively scaled the learning rate for each dimension. This algorithm performs best for sparse data because it decreases the learning rate faster for frequent parameters, and slower for parameters infrequent parameter. Adagrad greatly improved the robustness of SGD and used it for training large-scale neural nets at Google.

$$g_{t,i} = \nabla_\theta * J(\theta_{t,i})$$

$g_{t,i}$ is then the partial derivative of the objective function w.r.t. to the parameter θ_i at time step t. The SGD update for every parameter θ_i at each time step t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta * g_{t,i}$$

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i, based on the past gradients that have been computed for θ_i.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{g_{t,ii} + \epsilon}} * g_{t,i}$$

One of Adagrad's main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that.

Now let us look at the contour and loss plot, same as the other two algorithms compared above.

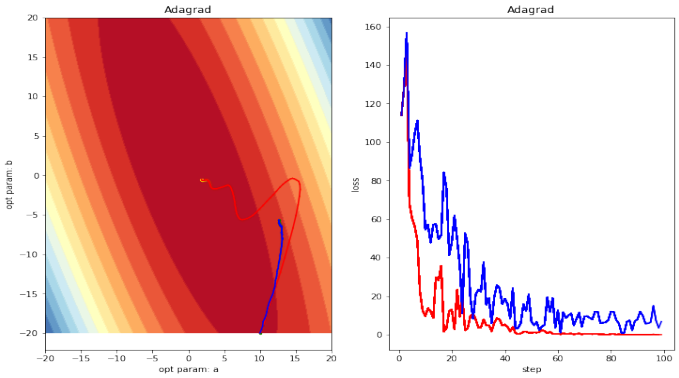

Fig. 16. Objective function and loss curve : Adagrad

Similar to the case of other Adaptive gradient techniques, it is clear that while the first order approximation technique such as SGD fails to escape the saddle point, the Adagrad does not face any difficulty in moving past that point. In fact it is common across all the adaptive algorithms is that it looks as if the algorithm first shoots away from the minimizing point, but then that helps in either the case of adding momentum, or using adaptive learning rates to adjust and get to the functional minima.

D. Comparing Optimizers in Neural Networks

The above described algorithms are predominantly used in training of neural network models. Let us look at the comparative performance analysis and see how it looks.

For this purpose, I used MNIST [17] training data, and use the tensorflow's inbuilt optimizer algorithms to test and see if we can recognize some patterns. For the purpose of keeping it simple, we decided to go with a 5 layer neural network and see how it performs.

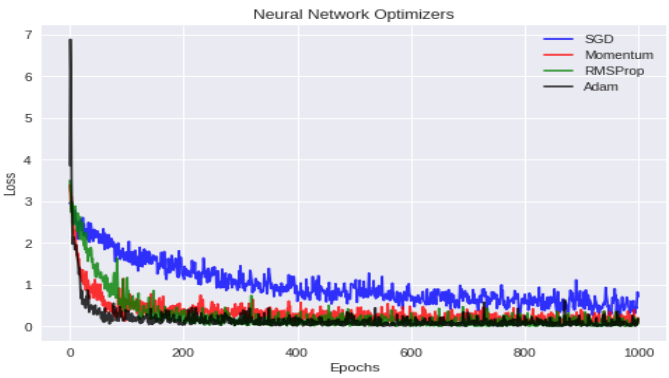

Fig. 17. Comparing Neural Network Architecture

One thing that is clearly visible is that all the optimizers other than SGD performs better at minimizing the loss functions relatively easily. This could also be the case where the function could be stuck at a local minima or a saddle point and

fails to come out of it. But the adaptive gradient techniques, Momentum based implementation performs better and have few tricks to tackle this scenario.

VIII. Comparison of different methods for escaping saddle points

Method	**Algorithm Complexity**	**Speed of convergence**	**Use in higher dimension problems**	**Limitations**
Newton free Saddle	High	Fast	Yes	Statistical information of high dimensional error surfaces needed
Stochastic gradient tensor decomposition	Low	Very fast	No	As dimensions increase, computatation increase
Adam	High	Fast	Yes	Better than other adaptive methods but difficult to implement
RMS Prop	Low	Fast	No	Difficult to tune parameters
Adagrad	Very low	Fast	Yes	Simpler parameter tuning but not reliable

TABLE I
HYPERPARAMETER SELECTION

IX. Conclusion

Saddle points are present in all functions and cause an impedance to the convergence of the function to the local optimum value. It becomes necessary to ensure that the algorithm does not get stuck in these points. We looked at various methods like adding noise/perturbations and momentum based optimizations to escape these points. Then we studied research papers which explained different methods for escaping these saddle points. A comparison between stochastic and deterministic methods were conducted and a great deal of understanding was gained from these studies. Even if saddle points can be avoided, there can be multiple local minima of widely different objective value. The performance of first-order methods would depend crucially on whether they converge for most initial conditions to nearly optimal global minima.

References

[1] Yurii Nesterov and Boris T Polyak. *Cubic regularization of newton method and its global performance. Mathematical Programming, 108(1):177–205, 2006.*

[2] Frank E Curtis, Daniel P Robinson, and Mohammadreza Samadi. *A trust region algorithm with a worst-case iteration complexity of mathcal O(epsilonˆ-3/2) for nonconvex optimization.* Mathematical Programming, pages 1–32, 2014.

[3] Frieze, A., Jerrum, M., and Kannan, R. (1996). Learning linear transformations. In 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, pages 359–359.

[4] Dauphin et al., 2014 Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*. In Advances in Neural Information Processing Systems, pages 2933–2941.

[5] [Chen et al., 2018 Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. *On the convergence of a class of adam-type algorithms for non-convex optimization.* arXiv preprint arXiv:1808.02941, 2018]

[6] Ge et al., 2015; Lee et al., 2016 Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. *Escaping from saddle points – online stochastic gradient for tensor decomposition.* In Peter Grnwald, Elad Hazan, and Satyen Kale, editors, Proceedings of The 28th Conference on Learning Theory, volume 40 of Proceedings of Machine Learning Research, pages 797-842, Paris, France, 03 06 Jul 2015. PMLR. URL http://proceedings.mlr.press/v40/Ge15.html

[7] R. Ge, F. Huang, C. Jin, and Y. Yuan. *Escaping from saddle points – online stochastic gradient for tensor decomposition.* In Conference on Learning Theory (COLT 2015), 2015

[8] K. Y. Levy. *The power of normalization: Faster evasion of saddle points.* ArXiv:1611.04831, 2016.

[9] C. Jin, R. Ge, P. Netrapalli, S. Kakade, and M. I. Jordan. *How to escape saddle points efficiently.* In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), 2017.

[10] C. Jin, P. Netrapalli, and M. I. Jordan. *Accelerated gradient descent escapes saddle points faster than gradient descent.* In Conference on Learning Theory (COLT 2018), 2018.

[11] Simon S Du, Chi Jin, Jason D Lee, Michael I Jordan, Aarti Singh, and Barnabas Poczos. *Gradient descent can take exponential time to escape saddle points.* In Advances in Neural Information Processing Systems, pp. 1067–1077, 2017

[12] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. *Escaping from saddle points - online stochastic gradient for tensor decomposition.* In Proceedings of The 28th Conference on Learning Theory, COLT 2015, pages 797–842, 2015.

[13] Leslie N Smith. *clical learning rates for training neural networks.* In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on, pp. 464–472. IEEE, 2017.

[14] D. Kingma and J. Ba. Adam: *A method for stochastic optimization.* In ICLR, 2015.

[15] Tijmen Tieleman and Geoffrey E. Hinton. Lecture 6.5-*rmsprop: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural Networks for Machine Learning, 4(2), 2012. URL https: www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[16] Duchi, J., Hazan, E., Singer, Y. (2011). *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* Journal of Machine Learning Research, 12, 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html

[17] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86, 2278–2324

[18] Yurii Nesterov and Boris T Polyak. Cubic regularization of Newton method and its global performance. Mathematical Programming, 108(1):177–205, 2006.

[19] Ioannis Panageas and Georgios Piliouras. Gradient descent only converges to minimizers: Non- isolated critical points and invariant regions. In Innovations of Theoretical Computer Science (ITCS), 2017.

[20] https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/maximums-minimums-and-saddle-points

[21] Tuo Zhao, Zhaoran Wang, and Han Liu. Nonconvex low rank matrix factorization via inexact first order oracle. Advances in Neural Information Processing Systems, 2015.

[22] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. arXiv preprint arXiv:1707.04926, 2017.

[23] Yurii Nesterov. Introductory Lectures on Convex Optimization, volume 87. Springer Science Business Media, 2004.

[24] Simon S Du, Chi Jin, Jason D Lee, Michael I Jordan, Barnabas Poczos, and Aarti Singh. Gradient descent can take exponential time to escape saddle points. arXiv preprint arXiv:1705.10412, 2017.

Appendix

Binoy Thomas

- Worked on the simulation of saddle points and completed the preliminary studies needed for the completion of the project.
- Researched about the problems related to simulating saddle points in higher dimension and which algorithm would be better for escaping saddle points.
- Researched about Saddle free Newton method and studied the algorithm.
- Worked on the stochastic gradient for tensor decomposition. Studied the noisy stochastic gradient algorithm and worked on its implementation.
- Did a comparitive study of the various methods

Rajaram Sivaramakrishnan

- Worked on studying and summarizing the related work to see what improvements were possible.
- Researched and implemented the subsections of Methods to deal with saddle points and generate relevant plots.
- Read and summarized the tricks to deal with saddle points including PGD and momentum methods.
- Worked on the comparative study of the different optimizers and studied relevant papers to it.