Fully Autonomous First Order Mathematical Optimizer

Anonymous Authors[1]

Abstract

This research paper describes a technique for performing mathematical optimization, which is free from any hyperparameters. The optimization requires only first order derivatives of the objective function with respect to the learning parameters. The step size, more commonly known as the learning rate, which is manually set for training is replaced by a heuristic which calculates an optimal step size relative to the instantaneous value of the objective function. Finally the results obtained on the MNIST dataset by using this technique are presented and discussed in brief while comparing to other optimization algorithms. The algorithm proves its stability on the MNIST classification and achieves performance comparable to Adam optimizer which is the current standard.

1. Introduction

Mathematical Optimization lies at the heart of Machine Learning. In order to statistically fit the function over the given data, the learnable parameters of the model are modified according to a certain objective function. This is true for both supervised and unsupervised learning. In case of the latter, the distinction is based only in the definition of the objective function, which in this case doesnt involve any label information. The optimization is performed on a mathematical function *f(x)* known as the **objective function** or **criterion** by obtaining the value of *x* such that the value of *f(x)* is optimal (either maximum or minimum). In general context, optimization refers to minimization of the criterion. The maximization of the same can be achieved by minimizing the negative criterion *-f(x)* (Goodfellow et al., 2016). When the minimization is considered, the objective function is also referred to as **loss function** or **error function** or **cost function**.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1.1. Related Works

The **gradient descent** algorithm (Cauchy, 1847) has been the most popular algorithm for performing mathematical optimization. It is an iterative algorithm which deduces the direction to move towards at any given particular instant by using the sign of the local gradient of *f(x)* wrt *x*. The update equation of gradient descent is given by:

$$x_{t+1} = x_t - \alpha \frac{df(x_t)}{dx}$$

where the α is known as the **learning rate** which specifies the step-size for any particular iteration of the algorithm. The most common method for choosing the value of α that has been applied is trial and run; and thereby the algorithm requires human attention.

Apart from its heavy reliance on a perfectly chosen learning rate, it was pointed out that the algorithm is susceptible to **ravines**, i.e. variable gradient rates along different dimensions (Sutton, 1986). Over the years, various attempts have been made in order to patch up these flaws of the gradient descent algorithm. The use of **Momentum** in the update equation of gradient descent was an attempt at tackling the latter problem (Qian, 1999). The update equations for the momentum are given by

$$v_t = \gamma v_{t-1} + (1-\gamma)\frac{df(x_t)}{dx}$$
$$x_{t+1} = x_t - \alpha v_t$$

where the γ is known as the momentum weightage or the running average factor. In some literature, the γ is also denoted by the symbol β, the meaning of both the symbols is however the same. The momentum modification takes into consideration the values of gradients of the previous iterations and accumulates them in the v term. Unfortunately, in order to evade the problem of ravines, the momentum makes a trade off with the addition of another hyperparameter γ.

The Nesterov Accelrated Gradient or **NAG** (Nesterov, 1983) builds upon the concept of momentum and adds a future lookup via the accumulated gradients in order to make a better estimate of the optimal direction to move towards. The **RMSprop** optimization scales the current gradient by the inverse square root of accumulated second moments of the previous gradients. The current standard and the most widely employed algorithm **Adam** (Kingma & Ba, 2015) combines the concept of Momentum and RMSprop coupled with learning rate decay. The **Nadam** variant of the same, which uses the Nesterov Momentum, is also sometimes used for optimization. The systematic update equations of Adam are given by:

$$
\begin{aligned}
g_t &= \frac{df(x_t)}{dx} \\
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{(1-\beta_1^t)} \\
\hat{v}_t &= \frac{v_t}{(1-\beta_2^t)} \\
\hat{\alpha} &= \alpha \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t} \\
x_{t+1} &= x_t - \hat{\alpha} \frac{\hat{m}_t}{(\sqrt{\hat{v}_t}+\epsilon)}
\end{aligned}
$$

There are in all four hyperparameters in the Adam update algorithm viz. α, β_1, β_2 and ϵ of which, the ϵ is used for the numerical stability of the algorithm and is rarely tweaked. The default value used for ϵ is *1e-8*.

There are various other algorithms that have been devised for the problem of Mathematical Optimization and have been utilised with varying rates of success; whereby, they all have their own advantages as well as shortcomings (Rudder, 2017). Some approaches based on statistical schedules such as (Darken et al., 2002) and using annealing (Robbins & Monro, 1951) have been published. Novel approaches such as the (Koolen et al., 2014) and (Schaul et al., 2013) for the **stochastic gradient setting**, where update is made per data example, prominently stand out, but unfortunately it is felt that they add up redundant complexity to the task of Mathematical Optimization. Works that involve the use of ***Higher Order Derivatives*** or ***Hessians*** in the update equations that follow the lines of **Newton's Method** are not relevant to the present research and hence not compared.

1.2. Comparative analysis

In almost all these gradient descent variants, it occurs that as much of efforts that have been put towards making the algorithm stabler and more performant, have entailed increment of hyperparameters to the update equations, resulting in lesser autonomy. The statistical approaches that guide the selection of these hyperparameters fit the relevant domains perfectly but require the same efforts to be reproduced for disparate domains. This brings to highlight the two key factors of the optimization algorithm to be presented here: **simplicity** due to the use of only first order derivatives and simplistic geometric approximations without any momentum accumulation; and **autonomy** because of not requiring any hyperparameters; while making only one assumption about the loss function being positive real valued [1].

2. Mathematical explanation of the Algorithm

This section provides a step by step description of the proposed algorithm for performing Mathematical Optimization. Following sub-sections would present the update equation of the algorithm and also explain the mathematics supporting it. Followed by, the analysis of the parameter update portion of the equation and primary simulations on common known mathematical functions.

2.1. Proposed Algorithm

The algorithm proposed by the present research has the same procedure as that of gradient descent. Precisely, the algorithm starts by initializing the trainable parameters x randomly [2] in order to make an initial guess and then proceeds iteratively by updating the parameters towards better values (Algorithm 1).

Algorithm 1 Proposed Optimization

```
Input: data $(x_i, y_i)$, parameters $W$
Initialize $W$ randomly
repeat
    Forward Pass $loss = l(f(x_i; W), y_i)$
    Backpropagation calculate $\frac{df(x_i)}{dW}$
    Update params $W = RU(W, loss, \frac{df(x_i)}{dW})$
until $epoch \leq totalEpochs$
```

There is a modification proposed in the update equation used for iteratively improving the parameters, termed *RU* i.e. Relative updation. The update equation is given by:

$$
\boxed{x_{t+1} = x_t - \frac{f(x_t)\dfrac{df(x_t)}{dx}}{f(x_t) + \left(\dfrac{df(x_t)}{dx}\right)^2}; \forall x_t \in \mathbb{R}, f(x_t) \geq 0}
$$

It is important to note that the $\left(\frac{df(x_t)}{dx}\right)^2$ corresponds to the squared derivative as opposed to the second order derivative denoted by $\frac{d^2f(x_t)}{dx^2}$. The $f(x_t)$ term corresponds to the value of the objective function at the t^{th} iteration of the algorithm. The only assumption made by the algorithm is

[1]The loss function for all the Machine Learning and Deep Learning tasks are almost always defined as positive real valued functions

[2]There are various distinct methods proposed for initialization. The main motive behind the random parameter initialization is breaking the symmetry among them.

that the cost function defined for this optimization task is a positive real valued function.

The intuition behind using the cost term in the update equation draws inspiration from the ***'General Theory of Relativity'*** put forth by Albert Einstein (Einstein, 1914 - 1917). The notion of unified existence of space and time, term coined as spacetime, is the driving motivation of the update term of this algorithm. As it was pronounced by Einstein, that space and time are not distinct but relative to each other, it is hypothesized that *'the dimensions of the objective function and its constituent parameters are not distinct, but relative to each other'*. It can be noted that in none of the prevailing algorithms, the cost term is used in the update equation; which have treated these dimensions as being absolute. The *'General Theory of Relativity'* explains motion by the existence of a curved or warped spacetime around the moving objects. This setting highly resonates with the curvature of the objective function being non-convex.

Another source of inspiration behind the update term of the proposed algorithm that needs to be mentioned is the ***Newton-Raphson*** method of finding roots of arbitrary polynomials (Akram & ul Ann, 2015). The *Newton-Raphson* algorithm is an iterative algorithm that updates the value x such that it moves towards the roots of the polynomial $p(x)$, i.e. the value of x where $p(x) = 0$. The update equation is given by:

$$x_{t+1} = x_t - \frac{p(x_t)}{p'(x_t)}$$
$$\text{where, } p'(x_t) = \frac{dp(x_t)}{dx}$$

For any given instant, the *Newton-Raphson* algorithm associates a linear line that corresponds to the tangent of the polynomial curve at the point $(x_t, p(x_t))$ to that point and extrapolates it till it cuts the x-axis, i.e. $y = 0$ line, and uses this length to make an update. This algorithm has been used tremendously for sloving non-square and non-linear mathematical problems for years. The influence of the *Newton-Raphson* method on the proposed equation is easily evident from the function value term in the update portion and the inverse proportional relation of the update portion with the gradient. It needs to be noted that although the *Newton-Raphson* method seems to be highly applicable in our context of optimization instead of the intended root-finding; due to the assumption of $p(x)$ ($f(x)$ in broader sense) being a positive real valued function, which implies that any root of the function must also correspond to the minima; there is still another subtle assumption that *Newton-Raphson* method imposes. This assumption is: the existence of a real root to the $p(x)$. This is a very stringent assumption and cannot be made in the context of Machine Learning optimization problems. The assumption of existence of a global minimum corresponding to a root implies that there would always exist a certain configuration of the parameters of model that fits the data perfectly for every problem. This is definitely not true and in fact, in practical settings, one has to settle at a local minimum that works decently and also generalises well to unseen data examples. From the $\frac{p(x_t)}{p'(x_t)}$ term, it is evident that the *Newton-Raphson* method is unstable at a point of local minimum. Precisely, at the point of local minimum, the gradient $p'(x_t) = 0$ while the $p(x_t)$ would have a real value leading to a fraction that has 0 in the denominator. This causes the update term to explode to ∞. Thus the *Newton-Raphson* algorithm cannot be directly used for optimization. The *Newton-Raphson* method however can be used to optimize a twice differentiable function by performing root-finding on the derivative of the original function. But, as mentioned earlier, since this invovlves second order derivatives, it cannot be compared to the proposed algorithm.

2.2. Update term gradient analysis

The update portion of the proposed algorithm can be isolated in order to gain better insights about it. Mathematically,

$$\text{Let, } z = u(x, y) = \frac{xy}{x + y^2}; \forall x \in \mathbb{R}, x \geq 0$$

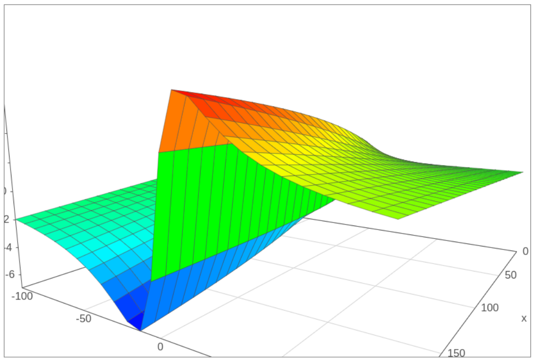

Figure 1. The 3D surface illustration of the update function $u(x, y)$

Where, x has the role of the instantaneous cost and y behaves as the gradient. The behaviour of this function can be analysed by considering the 3D surface formed by the function (Figure 1) [3]. The figure is a still shot of the surface from an explanatory angle. It is evident that the surface tapers toward the $z = 0$ plane as we go away in either directions in the $y - axis$. This means that the update slows down if the slope becomes too much, i.e. too steep; a behaviour that

[3]The figure is generated using the online tool https://academo.org/demos/3d-surface-plotter

is desired. Also, the surface cuts the $z = 0$ plane exactly at $y = 0$. This shows that the equation is stable and will converge when the derivative becomes 0. Finally, the most important feature is that the properties of these taped curves are controlled by the x dimension. As the cost becomes 0, the z approaches a singularity, i.e. when both $x = 0$ and $y = 0$, the function produces a $\frac{0}{0}$ form. Mathematically,

$$\lim_{x\to 0, y\to 0} (u(x,y)) = \frac{xy}{x+y^2} = \phi$$

The point where this singularity occurs, i.e. $(x = 0, y = 0)$, is the point of global minimum, where the cost is 0 (a root) and the derivative being 0 ensures a smooth touch of the cost function to the parameter dimensions' plane. The chances of reaching this point are very less. Through the experimental simulations, it was observed that in the context of computer based numerics, the updated parameters in spite of being ridiculously close, never exactly reach this point due to the round-off effect. The subsequent sub section sheds further light over this phenomenon. Although it is not required, but to ensure numerical safety, a very small value similar to the Adam optimizer can be added to the denominator. This modifies the function as,

$$z = u(x,y) = \frac{xy}{x+y^2+\epsilon}; \epsilon \leq 10^{-8}$$

The addition of the ϵ gives an oscillating nature to the algorithm. When the algorithm reaches very close to the point of the global minimum (the singularity), the values of the parameters would just oscillate in the ϵ delta range and would never yeild the $\frac{0}{0}$ value. This is the reason why the ϵ value has to be very small in order to achieve a good enough approximation.

Another way to visualize the update function $u(x,y)$ is by comparing it with the gradient descent update term. Upon equivalence comparison of the update terms of the two algoriths, it can be deduced that,

$$\alpha = \frac{f(x_t)}{f(x_t) + \left(\frac{df(x_t)}{dx}\right)^2}$$

This equation yeilds the geometric approximation of the *learning rate* of the gradient descent algorithm. At every iteration t of the proposed algorithm, it emulates the gradient descent behaviour by automatically approximating a value of the step-size (*learning rate*) and uses the sign and magnitude of the instantaneous slope (given by derivative) to make an update.

2.3. Simulation on single variable synthetic functions

The proposed algorithm was tested on numerous single variable mathematical functions. This section provides the results obtained on two of those functions. First function is the square function since it is very well behaved, i.e. continuous and differentiable function, corresponding to a preliminary *best-case*. The second function discussed here is the exponential function which represents the most skewed continuous function. It is important to test the algorithm on the e^x function due to its property of being immune to differentiation.

2.3.1. Square function

Let, $y = f(x) = x^2$ be the objective function

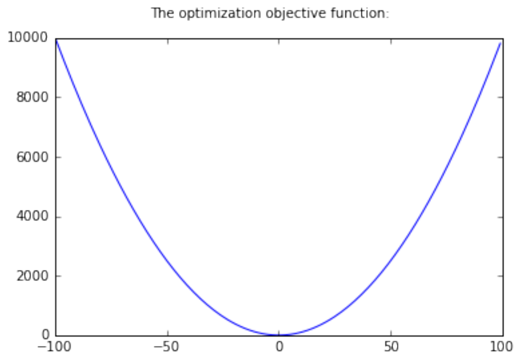

Figure 2. 2D y v/s x plot of the square objective function

Upon simulating the proposed algorithm on this function with an initial point $x = 1000$ for just 150 iterations, the following instantaneous cost plot was obtained (Figure 3).

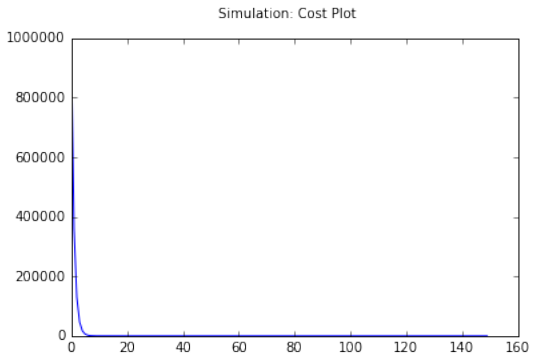

Figure 3. 2D Cost v/s Iteration plot of the simulation on square function

Final optimised position and cost were:
$x = 5.493336358815607$ x 10^{-16}
$y = 3.0176743$ x 10^{-31}

for the theoretical (actual) expected values of $x = 0$ and $y = 0$. While, the algorithm reached the x value of the order of 10^{-3} in just the initial 30 iterations.

2.3.2. Exponentiation function

Let, $y = f(x) = e^x$ be the objective function

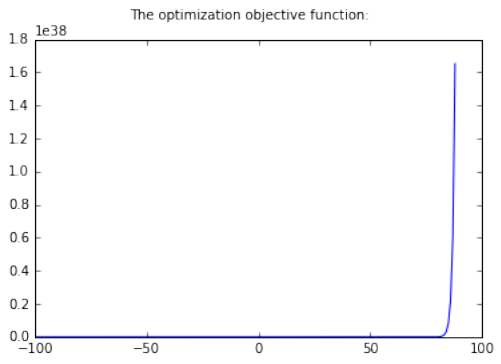

Figure 4. 2D y v/s x plot of the exponentiation objective function

Simulating the same algorithm on the exponentiation function this time, with an initial point $x = 10$ for 100 iterations, yielded the following instantaneous cost plot (Figure 5).

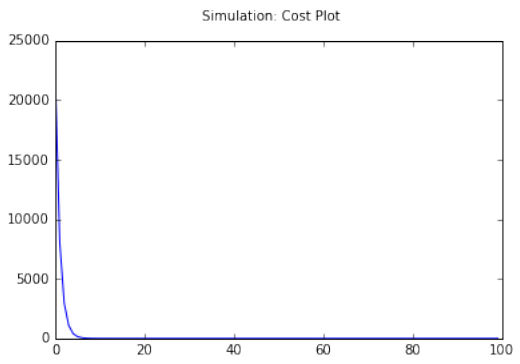

Figure 5. 2D Cost v/s Iteration plot of the simulation on exponentiation function

with the final optimised position and cost being:
$x = -4.47519719694$
$y = 0.0115184$

for the theoretical (actual) expected values of $x = -\infty$ and $y = 0$.

The function $f(x) = e^x$ has an oblique property of being unchanged by the differentation operator, i.e.

$$\frac{d(e^x)}{dx} = e^x$$

In spite of which, the algorithm proved it's stability during simulation and displayed consistent behaviour in the numerical calculations. Unfortunately the initial position for x could not be set higher, due to the inability of the programming framework to handle values of the order of e^{20}.

3. Experimentation

In order to test the practical soundness of the proposed algorithm, it was tested on the **MNIST** [4] dataset, which is regarded as the *'Drosophila'* of Machine Learning (Goodfellow et al., 2016). The classifier that is used is a 3 layers deep Neural Network having hidden representation size of 512 dimensions with ReLU activation function (Nair & Hinton, 2010). The use of batch normalization (Ioffe & Szegedy, 2015) is avoided since the network is quite shallow and the input data is normalised. The data distribution wouldnt shift too much in just three layers giving enough scope for ReLU to create non-linear boundaries. No regularization has been employed since the focus is on the optimization part and not the generalisation of the classification. The optimization was performed in the mini-batch gradient descent settings with a batch size of 64.

The programming implementation of this experiment and the simulations in the previous section are made using the TensorFlow framework (Abadi et al., 2016) and have been made open source at the repository [`link`] [5]. The loss plot shown in (Figure 6) is obtained after running the algorithm for 12 epochs. The important configuration of the algorithm that needs to be mentioned here is the initialization of the weights. This experiment uses the Xavier initializer (Glorot & Bengio, 2010).

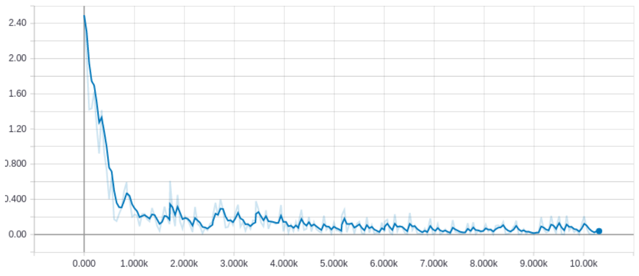

Figure 6. 2D Cost v/s Iteration plot of the described experiment

(Figure 7) denotes that the performance of the proposed algorithm is comparable to Adam Optimization. This brings to highlight that recently, a number of problems with the Adam optimizer have been highlighted (Anonymous, 2018), of which the most prominent being that adam is not guaran-

[4] `http://yann.lecun.com/exdb/mnist/`
[5] For anonymity, the link has been redacted.

teed to converge. Due to which, the analysis of the proposed algorithm becomes more pronounced.

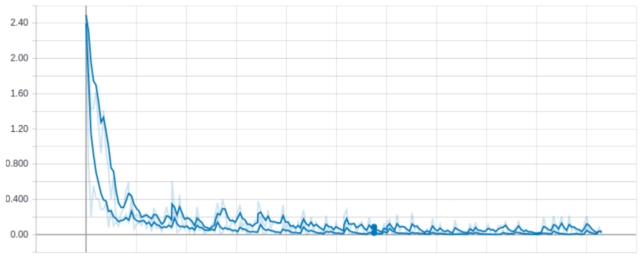

Figure 7. Comparison of Cost plots of the proposed algorithm and the Adam Optimizer

The proposed algorithm was compared alongside a number of other optimizers (more details in [`link`]), see (Figure 8) and it was observed that, majority of the other optimizers without careful tuning of their hyperparameters perfrom a lot worse than the proposed algorithm.

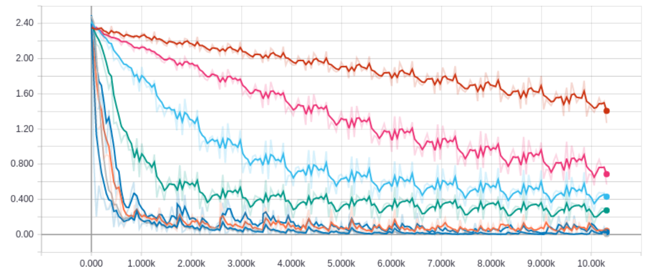

Figure 8. Comparison of Cost plots of the proposed algorithm and other prevailing optimizers

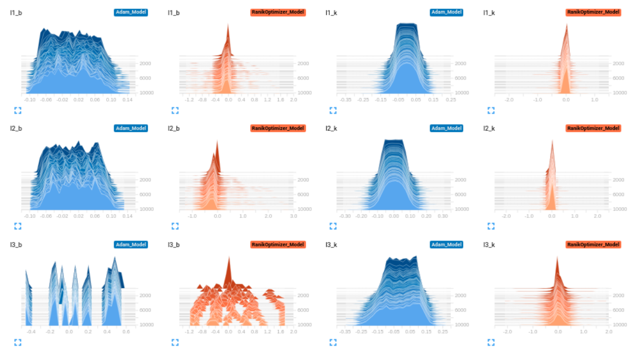

Figure 9. Histograms of the distributions of the trainable parameters in the Network

Another key distinction that is observed as described in (Figure 9) is that the distribution of the trained parameters for the proposed algorithm resemble the **Laplacian Distribution** as opposed to the ***Gaussian Distribution*** trained by the Adam Optimizer.

4. Conclusions

The research attempts to establish a hypothesis that the *'dimensions of the objective function and the constituent parameters are relative'*, and paves way for future analysis of this hypothesis. Given the promising results in the described experiments it would be interesting to see haw far can it go for bigger models such as CNN, ResNet, RNN etc. for other more robust Deep Learning datasets. The proposed algorithm without employing any *momentum*, or *RMSprop* mechanisms and utilising simple geometric approximations, achieves performance comparable to the optimizers that use *momentum* and *RMSprop*. With this, it is urged that the further research in this direction should give more importance to simplicity.

References

Abadi, Martn, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mane, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viegas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *ArXiv e-prints arXiv:1603.04467v2 [cs.DC]*, 2016.

Akram, Saba and ul Ann, Qurrat. Newton raphson method. *International Journal of Scientific and Engineering Research*, 6(7):1748 – 1752, July 2015.

Anonymous. On the convergence of adam and beyond. In *Under double blind review as a conference paper at ICLR 2018*, 2018.

Cauchy, Augustin-Louis. Methode generale pour la resolution de systemes d'equations simultanees. *Compute rendu des seances de l'academie des sciences*, pp. 536 – 538, 1847.

Darken, Christian, Chang, Joseph, and Moody, John. Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, Helsingoer, Denmark, August 2002. IEEE explore.

Einstein, Albert. The foundation of the general theory of relativity. *THE COLLECTED PAPERS OF Albert Einstein*, 6(30):146 – 200, 1914 - 1917.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, Saradina, Italy, 2010.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, Cambridge, Massachusetts, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv e-prints arXiv:1502.03167v3 [cs.LG]*, 2015.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *Proc. 3rd International Conference for Learning Representations, San Diego*, 2015.

Koolen, Wouter M, van Erven, Tim, and Grünwald, Peter. Learning the learning rate for prediction with expert advice. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2294–2302. Curran Associates, Inc., 2014.

Nair, Vinod and Hinton, Geoffrey. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27 th International Conference on Machine Learning 2010*, Haifa, Israel, 2010.

Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). *Doklady ANSSSR (translated as Soviet.Math.Docl.)*, 269: 543 – 547, 1983.

Qian, Ning. On the momentum term in gradient descent learning algorithms. *Neural Networks : The Official Journal of the International Neural Network Society, 12(1), 145151.*, 1999.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, September 1951.

Rudder, Sebastian. An overview of gradient descent optimization algorithms. *ArXiv e-prints arXiv:1609.04747 [cs.LG]*, 2017.

Schaul, Tom, Zhang, Sixin, and LeCun, Yann. No more pesky learning rates. *ArXiv e-prints arXiv:1206.1106v2[stat.ML]*, 2013.

Sutton, R S. Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. 8th Annual Conf.* Cognitive Science Society, 1986.