Implementation and Empirical Analysis of Modifications to the Stochastic Gradient Descent Optimizer

Kenan Karavoussanos

September 7, 2018

Abstract The choice of Optimizer for an Artificial Neural Networks requires multiple considerations, including length of training time, reliance on initial training constants and ability converge to a global minimum rather than a local minimum. This article aims to introduce improvements to the Stochastic Gradient Descent Optimizer and evaluate the resulting performance on various metrics. These Optimizers will be compared in the context of an Artificial Neural Network classifier.The evaluation of each optimizer on a Standard Dataset allows the results obtained to maintain a level of generality such that readers from various fields can make use of the comparison. This comparison is not universally applicable due to the stochastic nature of different datasets and problems. The results contained within this article are specifically applicable to Multiclass classification on balanced datasets.

Contents

1 Introduction

The purpose of an Optimizer in the context of Artificial Neural Networks is to minimize the objective function describing the classification error of the ANN. Stochastic Gradient Descent forms the base for many cutting edge Optimizers such as the Adam Optimizer. Certain strategies are employed in these optimizers to overcome the limitations SGD however these improvements typically come at a price. This paper seeks to elucidate the benefits and trade-offs of each of these strategies with respect to the time to convergence and the level of generalization of the network once it has converged.

The specific limitations of SGD are as follows:

1. SGD has slower performance on areas of the surface where the gradient in one dimension is drastically different from the gradient in other dimensions.

2. Stochastic Gradient Descent has a constant learning rate for the entire training epoch which can cause the ANN to overshoot minima and potentially fail to converge.

3. The learning rate is common across all parameters. This is suboptimal as we may want to perform different sized updates to different parameters at certain points.

4. The affinity to stay at a local minimum instead of searching for a better and or global minimum.

5. Stochastic Gradient Descent performs poorly around saddle points.

Due to the large amount of computation and calculation required to train an ANN, it is infeasible to trace the training algorithm for more than a few training examples. As such various metrics are used to evaluate the performance of the ANN and its Optimizer. Slow optimizers become prohibitive to training for large datasets so time to convergence has been chosen as a comparison metric. The level of generalization of the network has been chosen as a broad metric to be tested with specific metrics such as accuracy and precision, amongst others.

Significance of problem The current body of research shows the benefits of each new optimizer and how it overcomes specific limitations. However, the trade-offs are rarely specifically studied and as such this paper serves as a means to correct the selection bias in the body of research.

For specific or niche problem domains and datasets, knowing the performance enhancements and trade offs of each optimizer allows for the reader to select and appropriate Optimizer to fit within various real world constraints.

The evaluation of the each optimizer on a Standard Dataset allows for empirical validation of previous theoretical study on Gradient Descent based Optimizers with specific metrics.

Application of the problem solution to other domains Due to the generality provided by evaluation on a Standard Dataset the problem solution is applicable across many domains. However, this generality also makes the comparison less accurate on niche datasets with imbalanced classes or noisy data as the EMNIST dataset we are using has perfectly balanced classes as well as exceptionally clean data.

Organisation of the rest of the proposal The Methodology 1 which describes analytical aspects of work, MEthodology 2 which describes implementation of work and new algorithms and describes comparison order. Experimental setup/Computational Setup describe my machine, the development environment and packages also describe piecewise individual testing then combination into named optimizers and combined unnamed optimisers. Describe Dataset in Detail Discuss use of GIT, OO Design, public access to repo and how to run code.

preliminary results before moving to full research plan. validation of results describe limitation of results and how to extend current results.

2 Literature Survey

The following literature review seeks to critically evaluate the current evaluations of various prolific optimizers used in Artificial Neural Networks. In particular, the benefits of each optimizer in comparison to others as well as the discussed trade-offs of the optimizer. The time to convergence as well as level of generalization will be the comparison metrics discussed.

(Kingma et al 2015) provides an unbiased empirical evaluation of the Adam Optimizer with respect to other optimizers. Adam can be described as a combination of RMSProp, Momentum and a bias-correction term. Multiple statistics are used for the comparison and shows that Adam is faster to converge than all the other tested optimizers, however it is admitted that Adam tends to over-fit to the training data so regularization and dropout strategies were used to improve generalization. For a Multi-layer Neural Network with dropout stochastic regularization, the training cost versus iteration number is reported for multiple Optimizers. However, the generalization of these optimizers, along our chosen metrics is not reported.

(S Ruder 2016) provides an overview of various gradient based optimizers and their specific limitations. However the overview is wholly theoretical and does not report any empirical statistics in its analysis. Provided derivations of each optimizer have been used in the current work for the development of the algorithms used in each A visualization of some of the discussed optimizers is provided but this does not specifically pertain to the domain of our problem as the visualized objective functions are the Beale Function and a hyperparaboloid. It is important to note the poor performance around saddle points of the optimizers with static learning rates.

(C Darken et all, 199x) discusses learning rate schedules as a solution to a static learning rate as well as introduces the Search-Then-Converge (STC) Method. A major limitation to learning rate schedules discussed is that the learning rate decreases too quickly so convergence is not guaranteed. STC attempts to solve this problem by slowing down this reduction in learning rate and then increasing the reduction once a certain time threshold has been reached. This gives the optimizer time to *search* for a local minimum and then *converge*, hopefully, once it is found. The paper describes that this method is *guaranteed* to converge, however there is no proven guarantee that the optimizer will find a local minimum within the "search" period. This bias warrants that further investigation be reported in the current work to validate the claims of this paper.

(J Duchi et al, 2011) proposes the Adaptive Subgradient Method ,or ADAGRAD.The performance of this method is gauged along two metrics. The online error as well as the test set performance of the resultant classifier after one learning epoch. Various other adaptive gradient methods are tested for comparison and it is shown that ADAGRAD is the best optimizer to use when the training data is sparse. However, for small datasets, ADAGRAD is beat by the AROW algorithm. It is also shown that, on MNIST classification, the error rate for ADAGRAD is 0.4% greater than that of the Passive-Aggressive Algorithm. The version of ADAGRAD tested in this paper involves the use of Regularised Dual Averaging and thus differs from the ADAGRAD version to be tested in the current work.

(L smith 2017) proposes the use of cyclical learning rates as opposed to adaptive learning rates. Cyclical learning rates (CLR) are shown to have less computational cost than other adaptive methods. The method involves sampling a periodic function which is bounded by a maximum and minimum learning rate. This prevents the learning rate from monotonically decreasing as in ADAGRAD or other learning rate scheduling techniques. This improves the performance of the optimizer around saddle points. Selecting a minimum and maximum learning rate as well as the cycle length of the periodic function is covered by a selection of tests contained within the paper.

An interesting result in the paper shows that when using CLR with a gradient acceleration method, The Nesterov Method, it took only 25000 training iterations to achieve accuracy similar to the regular Nesterov Method after 70000 iterations. Similarly, RMSProp with CLR achieves the same improvement. However, when CLR is used in conjunction with the Adam Optimizer, the performance is worse than the regular Adam Optimizer. This suggests some conflict with either the Momentum Method or the bias correction term in the Adam Optimizer.

The only generalization metric tested is the accuracy. This does not provide a holistic view of the generalization of the method. As such the current work will report the generalization along multiple metrics.

3 Methodology

This section will provide a detailed explanation of the concepts being used as well as the experimental process to be followed in this work. This will be split into two sections:

- Methodology-I
- Methodology-II

3.1 Methodology-I

This section provides an overview of the concepts involved in the preliminary work, a detailed explanation of each concept, the equations to be used throughout the preliminary work and the pseudo-code of the learning process with the SGD optimizer.

3.1.1 Detailed Concepts

Artificial Neural Network [?] defines an Artificial Neural Network as a massively distributed and parallelized processor made up of small computational units called Neurons that acquires knowledge from its environment through a learning process and stores this knowledge as synaptic weights between neurons.

A Neuron takes external data or data from other neurons as input. Each input is multiplied by its respective synaptic weight and then all of these are added up to produce a single value called the activation[need reference] this activation is then passed as input into a non-linear activation function which produces an output which either goes to another Neuron or goes to the output of the network.

Neurons are arranged in layers:

- An input layer
- An output layer
- Zero or more hidden layers. So called as the inputs and outputs to and from the neurons in these layers are typically unobserved.

In a feed-forward network, output is passed exclusively to the next layer of neurons. The strength of the connection between two neurons is determined by their synaptic weight. Each neuron in a layer is connected to every neuron in the previous layer and every neuron in the next layer. These values of all the synaptic weights can easily be stored in matrix from called a Weight Matrix. Each layer of a network has its own respective weight matrix.

The learning process is typically modeled by the backpropogation algorithm combined with an update rule. The backpropogation algorithm is responsible for assigning blame for error in the network's output to the neurons of the network. Once this blame has been assigned, the update rule i.e The Optimizer updates the synaptic weights of each neuron in an attempt to minimize the error function.

Gradient Descent Gradient Descent is classified as a first-order iterative optimization algorithm. It works by exploiting the principle that given a multivariable function $J(\underline{\theta})$, that is differentiable at a point $\underline{a}$, then $J(\underline{\theta})$ decreases fastest along the direction of the negative gradient of $J(\underline{\theta})$ at a i.e $-\nabla J(\underline{a})$

Given a weight matrix $\underline{\Theta}$, the gradient descent update rule for the weight θ_{ji} is given by:

$$\theta_j i \leftarrow \theta_{ji} + \Delta\theta_{ji} \tag{1}$$

where the weight update, $\Delta\theta_{ji}$, is given by:

$$\Delta\theta_{ji} := -\alpha \frac{\partial J(\theta)}{\partial \theta_{ji}} \tag{2}$$

α is the learning rate and $\frac{\partial J(\theta)}{\partial \theta_{ji}}$ is the gradient of the error function with respect to θ_{ji}. $\frac{\partial J(\theta)}{\partial \theta_{ji}}$ is calculated using the backpropagation algorithm [?].

Learning Rate The Learning Rate of an ANN can be defined as the rate at which parameters change in a network. That is, given a new update on θ_{ji}, the proportion of $-\frac{\partial J(\theta)}{\partial \theta_{ji}}$ that is added to the current θ_{ji} is given by, α, our learning rate.

The value of the Learning Rate is crucial to the convergence of the network. Too low of a learning rate and the time to convergence becomes exceedingly high. Too high of a learning rate and the network may fail to converge due to continually stepping over the minimum. The Learning Rate can be global to all weights in the network or local to individual or groups of weights. Additionally, the learning rate can be static for the entire learning process or can be adaptive throughout the learning process.

Momentum The Momentum Method is an acceleration method [**?**] used to overcome sudden changes in gradient, that are common around minima, and reduce the time to convergence by setting the next weight update equal to a linear combination of:

1. $\frac{\partial J(\theta)}{\partial \theta_{ji}}$ the gradient of the error function with respect to θ_{ji}
2. $\Delta\theta_{ji}$ the previous weight update.

$$\Delta\theta_{ji_{t+1}} := -\alpha \frac{\partial J(\theta)}{\partial \theta_{ji}} + \beta \Delta\theta_{ji} \tag{3}$$

Where $\beta < 1$ is the momentum, or exponential decay, constant

Doing this improves the ability of the optimizer to avoid local minima by allowing it to follow the overall trend in gradient rather than the instantaneous gradient. This also speeds up convergence as once the trend in the gradient is found, the optimizer accelerates along this trend towards the minimum.

3.1.2 Dataset

The Dataset the ANN will be tested on is the Extended MNIST dataset. It is a dataset of 18800 28x28 pixel images of handwritten letters a-z and digts 0-9. The classes are evenly distributed thus the dataset is considered balanced. The full dataset is available on the github page for this work.

The data shall be randomly split as follows:

- 60% Training Data: This data is used during the learning process.
- 20% Validation Data: This data is used to tune and test for hyperparameter values.
- 20% Test Data: This data is used for unbiased evaluation of the networks performance. This data is not seen by thee network until the testing phase. The performance metrics for each Optimizer will be based solely on the performance on this set.

3.1.3 Architecture

Input Layer The Architecture of our ANN will be a Multilayer Feed Forward Network with 2 hidden layers.

The data takes the form of 28x28 pixel images. With each pixel being an individual input we have, $\boldsymbol{N}_i$, the number of inputs equal to 784.

Due to the fact that all Optimizers will be run on the same network architecture the number of neurons in each hidden layer will be determined beforehand rather than via cross-validation, this prevents the architecture from being tailored to any one optimizer as well as controlling a variable in the experiment.

(F Panchal et al 2014) suggest a simple approach to selecting the number of neurons in a network by providing some rules of thumb. Specifically,

1. $\boldsymbol{N}_h \in [\boldsymbol{N}_o, \boldsymbol{N}_i]$
2. $\boldsymbol{N}_h < 2 \times \boldsymbol{N}_i$
3. $\boldsymbol{N}_h = \frac{2}{3}\boldsymbol{N}_i + \boldsymbol{N}_o$

However, guideline 3 results in a prohibitively large number of neurons. Thus guideline 1 and 2 will be followed and a custom strategy adopted. To minimize the number of neurons in the network so as to minimize computation time the following strategy has been selected:

$$\boldsymbol{N}_{h1} = \boldsymbol{N}_{h2} = \boldsymbol{N}_o = 36 \tag{4}$$

These numbers satisfy both guideline 1 and guideline 2.

3.1.4 Learning Process Algorithms

This section provides the pseudocoded algorithms for the learning process with the SGD optimizer, the SGD optimizer with momentum. The pseudocoded algorithms for the adaptive learning rate methods can be found in Methodology-II.

Learning Process With SGD The algorithm below shows the full learning process involving the backpropagation algorithm and the SGD optimizer.

```
   Input: Training Symbols $\{(\underline{x}_0, \underline{y}_0), ..., (\underline{x}_n, \underline{y}_n)\}$
   Output: None
1  Initialize all weights in all weight matrices to small non-zero gaussian
     random numbers.
2  Initialize all update matrices $\boldsymbol{\Delta\theta}^{(l)}$ to zero matrices.
3  foreach $(\underline{\boldsymbol{x}}_i, \underline{y}_i)$ in Symbols do
4      Set $\underline{\boldsymbol{a}}^{(1)} = \underline{\boldsymbol{x}}_i$;
5      foreach Layer $l \in \{1, ..., L-1\}$ do
6          Set $\underline{\boldsymbol{z}}^{(l+1)} = \boldsymbol{\Theta}^{(l)}\underline{\boldsymbol{a}}^{(l)}$
7          Set $\underline{\boldsymbol{a}}^{(l+1)} = g(\underline{\boldsymbol{z}}^{(l+1)})$
8      end
9      Set $\underline{\boldsymbol{\delta}}^{(L)} = \underline{\boldsymbol{a}}^{(L)} - \underline{y}_i$
10     foreach Layer $l \in \{L-1, ..., 2\}$ do
11         Set $\underline{\boldsymbol{\delta}}^{(l)} = (\Theta^{(l)})\underline{\boldsymbol{\delta}}^{(l+1)} \times g^{'}(\underline{\boldsymbol{z}}^{(l)})$
12     end
13     foreach Layer $l \in \{1, ..., L-1\}$ do
14         Set $\boldsymbol{\Delta\theta}^{(l)} = -\alpha[\underline{\boldsymbol{\delta}}^{(l+1)}\underline{\boldsymbol{a}}^{(l)^{\boldsymbol{T}}}]$
15         Set $\boldsymbol{\Theta}^{(l)} = \boldsymbol{\Theta}^{(l)} + \boldsymbol{\Delta\theta}^{(l)}$
16     end
17 end
```

SGD with Momentum The algorithm shall be modified by adding a proportion, β, of the previous update matrix i.e $\boldsymbol{\Delta\theta}_i^{(l)}$ to the latest update matrix.

```
   Input: Training Symbols $\{(\underline{x}_0, \underline{y}_0), ..., (\underline{x}_n, \underline{y}_n)\}$
   Output: None
1  Initialize all weights in all weight matrices to small non-zero gaussian
     random numbers.
2  Initialize all update matrices $\boldsymbol{\Delta\theta}_0^{(l)}$ to zero matrices.
3  foreach $(\underline{\boldsymbol{x}}_i, \underline{y}_i)$ in Symbols do
4      Set $\underline{\boldsymbol{a}}^{(1)} = \underline{\boldsymbol{x}}_i$;
5      foreach Layer $l \in \{1, ..., L-1\}$ do
6          Set $\underline{\boldsymbol{z}}^{(l+1)} = \boldsymbol{\Theta}^{(l)} \underline{\boldsymbol{a}}^{(l)}$
7          Set $\underline{\boldsymbol{a}}^{(l+1)} = g(\underline{\boldsymbol{z}}^{(l+1)})$
8      end
9      Set $\underline{\boldsymbol{\delta}}^{(L)} = \underline{\boldsymbol{a}}^{(L)} - \underline{y}_i$
10     foreach Layer $l \in \{L-1, ..., 2\}$ do
11         Set $\underline{\boldsymbol{\delta}}^{(l)} = (\Theta^{(l)}) \underline{\boldsymbol{\delta}}^{(l+1)} \times g^{'}(\underline{\boldsymbol{z}}^{(l)})$
12     end
13     foreach Layer $l \in \{1, ..., L-1\}$ do
14         Set $\boldsymbol{\Delta\theta}_{i+1}^{(l)} = -\alpha[\underline{\boldsymbol{\delta}}^{(l+1)} \underline{\boldsymbol{a}}^{(l)^{\boldsymbol{T}}}] + \beta \boldsymbol{\Delta\theta}_i^{(l)}$
15         Set $\boldsymbol{\Theta}^{(l)} = \boldsymbol{\Theta}^{(l)} + \boldsymbol{\Delta\theta}_{i+1}^{(l)}$
16     end
17 end
```

3.1.5 Learning rate schedules

This section discusses the Search-Then-Converge and Cyclical Learning Rate schedules and provides the governing equations for said schedules.

Search-Then-Converge This strategy involves decreasing the learning rate as time passes. The learning rate is given by the following equation:

$$\alpha = \frac{\alpha_0}{1 + \frac{i}{T}} \tag{5}$$

Where i is the index of the current training symbol, α_0 and T are selected hyperparameters.

Cyclical Learning Rates There are 3 hyperparameters that need selecting for this method:

- Stepsize - The number of iterations taken in a half-cycle of the periodic function.
- α_{min} - The infimum of our periodic function.
- α_{max} - The supremum of our periodic function.

It is recommended that the stepsize ranges from 2 to 10 times the number of data points in your training set. It is noted that there is no significant difference in performance along this range. Thus in our case:

$$stepsize = 2 \times |Trainingset| = 25300 \tag{6}$$

The learning rate boundaries are estimated with a test. The test is to run the algorithm for a number of epochs with linearly increasing learning rates. The learning rate versus validation accuracy graph is plotted. α_{min} is the learning rate at which the accuracy starts to increase and α_{max} is the point at which the validation accuracy starts to become jagged or decrease.

The learning rate, using a triangular periodic function, is given by the following equation:

$$\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) \times Max\{0, (1 - x)\} \tag{7}$$

where

$$x = |\frac{iteration}{stepsize} - 2 \times (CycleLength) + 1| \tag{8}$$

and

$$CycleLength = \lfloor 1 + \frac{iteration}{2 \times stepsize} \rfloor \tag{9}$$

3.2 Methodology-II

3.2.1 Adaptive Learning Rate Optimizers

This section proposes various vectorized update rules for the ADAGRAD, RMSProp and Adam Optimizers.

Adaptive Gradient Method The vectorized update rule for ADAGRAD is given by:

$$\boldsymbol{\Delta\theta}_{t+1} = -\alpha \boldsymbol{g}_{t+1} \oslash [\epsilon \boldsymbol{I} + \boldsymbol{G}_{t+1}]^{\circ \frac{1}{2}} \tag{10}$$

where

- $\boldsymbol{g}_t$ is the gradient matrix $\underline{\boldsymbol{\delta}}^{(l+1)} \underline{\boldsymbol{a}}^{(l)^{\boldsymbol{T}}}$
- $\boldsymbol{G}_{t+1}$, termed the normalization matrix is the sum of squares of gradients before time t+1 and is iteratively defined as:

$$\boldsymbol{G}_{t+1} = \boldsymbol{G}_t + \boldsymbol{g}_t^{\circ 2} \tag{11}$$

- $\epsilon \boldsymbol{I}$ is a smoothing matrix where all elements are a small number ϵ to prevent division by zero.
- $\circ$ is the Hadamard power operator

RMSProp The main problem with ADAGRAD is that the elements of the normalization matrix are monotonically increasing thus the learning rate is monotonically decreasing at each iteration. The vectorized update rule is identical to that of ADAGRAD except in this case $\boldsymbol{G}_t$ is now defined as matrix of the the running average of past squared gradients.

$$\boldsymbol{G}_{t+1} = \frac{1}{t+1}[t\boldsymbol{G}_t + \boldsymbol{g}_t^{\circ 2}] \tag{12}$$

Adam Adam combines the benefits of keeping the RMS of the past gradients, the variance, as well as a running average of past gradients, the mean. The first moment term acts as a momentum term as it is a direction preserving exponentially decaying average of previous gradients.

The vectorized update rule for Adam is given by:

$$\boldsymbol{\Delta\theta}_{t+1} = -\alpha\hat{\boldsymbol{m}}_{t+1} \oslash [\epsilon\boldsymbol{I} + \hat{\boldsymbol{v}}_{t+1}^{\circ\frac{1}{2}}] \tag{13}$$

where $\hat{\boldsymbol{m}}_{t+1}$ and $\hat{\boldsymbol{v}}_{t+1}$ are the bias corrected first and second moments of the gradients. This bias correction term is introduced as $\boldsymbol{m}_0$ and $\boldsymbol{v}_0$ are zero matrices and thus the moments are biased towards zero.

$\hat{\boldsymbol{m}}_{t+1}$ is given by the equation:

$$\hat{\boldsymbol{m}}_{t+1} = \frac{1}{1-\beta_1^{t+1}}\boldsymbol{m}_{t+1} \tag{14}$$

$\hat{\boldsymbol{v}}_{t+1}$ is given by the equation:

$$\hat{\boldsymbol{v}}_{t+1} = \frac{1}{1-\beta_2^{t+1}}\boldsymbol{v}_{t+1} \tag{15}$$

where,

$$\boldsymbol{m}_{t+1} = \beta_1\boldsymbol{m}_t + (1-\beta_1)\boldsymbol{g}_{t+1} \tag{16}$$

and,

$$\boldsymbol{v}_{t+1} = \beta_2\boldsymbol{v}_t + (1-\beta_2)\boldsymbol{g}_{t+1}^{\circ 2} \tag{17}$$

3.2.2 Contrast with previous work

The current work differs from previous work mainly due to the aggregation of various Optimizers that have not been compared in a single source. Additionally, the comparison metrics used in this work extend the comparisons made in previous research. The use of the EMNIST database extends previous comparisons made with the standard MNIST database.

4 Analytic Results to be used

This Section shall provide derivations for the newly vectorized update rules for the ADAGRAD, RMSPROP and Adam Optimizers. It will be shown that given the $M \times N$ update matrix, each element, $\boldsymbol{\theta}_{i,j}$, in the matrix is the single parameter update rule for that optimizer.

4.1 ADAGRAD and RMSProp

4.1.1 Derivation

Given the vectorized update rule:

$$\boldsymbol{\Delta\theta} = -\alpha \boldsymbol{g} \oslash [\epsilon \boldsymbol{I} + \boldsymbol{G}]^{\circ \frac{1}{2}}$$

$$\iff \boldsymbol{\Delta\theta} = -\alpha \begin{bmatrix} g_{11} & \dots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{m1} & \dots & g_{mn} \end{bmatrix} \oslash \left(\begin{bmatrix} \epsilon & \dots & \epsilon \\ \vdots & \ddots & \vdots \\ \epsilon & \dots & \epsilon \end{bmatrix} + \begin{bmatrix} G_{11} & \dots & G_{1n} \\ \vdots & \ddots & \vdots \\ G_{m1} & \dots & G_{mn} \end{bmatrix} \right)^{\circ \frac{1}{2}}$$

$$\iff \boldsymbol{\Delta\theta} = -\alpha \begin{bmatrix} g_{11} & \dots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{m1} & \dots & g_{mn} \end{bmatrix} \oslash \begin{bmatrix} \sqrt{\epsilon + G_{11}} & \dots & \sqrt{\epsilon + G_{1n}} \\ \vdots & \ddots & \vdots \\ \sqrt{\epsilon + G_{m1}} & \dots & \sqrt{\epsilon + G_{mn}} \end{bmatrix}$$

$$\iff \boldsymbol{\Delta\theta} = \begin{bmatrix} -\alpha \frac{g_{11}}{\sqrt{\epsilon + G_{11}}} & \dots & -\alpha \frac{g_{1n}}{\sqrt{\epsilon + G_{1n}}} \\ \vdots & \ddots & \vdots \\ -\alpha \frac{g_{m1}}{\sqrt{\epsilon + G_{m1}}} & \dots & -\alpha \frac{g_{mn}}{\sqrt{\epsilon + G_{mn}}} \end{bmatrix}$$

Thus $\boldsymbol{\Delta\theta}_{ij} = -\alpha \frac{g_{i,j}}{\sqrt{\epsilon + G_{ij}}}$

4.1.2 Analysis

The benefit of using this vectorized equation is that all operations are element-wise algebraic operations. As such, the equation is simple to understand for the reader and programmatic implementation is also simple. The presence of an interchangeable normalization matrix, $\boldsymbol{G}_{t+1}$, allows various other normalizations (other than Sum of Squared Gradients and the RMS of Gradients) to be easily implemented without having to derive a new vectorized equation.

4.2 Adam

4.2.1 Derivation

Given the vectorized update rule:

$$\boldsymbol{\Delta\theta} = -\alpha \hat{\boldsymbol{m}} \oslash [\epsilon \boldsymbol{I} + \hat{\boldsymbol{v}}^{\circ \frac{1}{2}}]$$

$$\iff \boldsymbol{\Delta\theta} = -\alpha \begin{bmatrix} \hat{m}_{11} & \dots & \hat{m}_{1n} \\ \vdots & \ddots & \vdots \\ \hat{m}_{m1} & \dots & \hat{m}_{mn} \end{bmatrix} \oslash \left(\begin{bmatrix} \epsilon & \dots & \epsilon \\ \vdots & \ddots & \vdots \\ \epsilon & \dots & \epsilon \end{bmatrix} + \begin{bmatrix} \hat{v}_{11} & \dots & \hat{v}_{1n} \\ \vdots & \ddots & \vdots \\ \hat{v}_{m1} & \dots & \hat{v}_{mn} \end{bmatrix}^{\circ \frac{1}{2}} \right)$$

$$\iff \boldsymbol{\Delta\theta} = -\alpha \begin{bmatrix} \hat{m}_{11} & \dots & \hat{m}_{1n} \\ \vdots & \ddots & \vdots \\ \hat{m}_{m1} & \dots & \hat{m}_{mn} \end{bmatrix} \oslash \begin{bmatrix} \epsilon + \sqrt{\hat{v}_{11}} & \dots & \epsilon + \sqrt{\hat{v}_{1n}} \\ \vdots & \ddots & \vdots \\ \epsilon + \sqrt{\hat{v}_{m1}} & \dots & \epsilon + \sqrt{\hat{v}_{mn}} \end{bmatrix}$$

$$\iff \boldsymbol{\Delta\theta} = \begin{bmatrix} -\alpha\frac{\hat{m}_{11}}{\epsilon+\sqrt{\hat{v}_{11}}} & \cdots & -\alpha\frac{\hat{m}_{1n}}{\epsilon+\sqrt{\hat{v}_{1n}}} \\ \vdots & \ddots & \vdots \\ -\alpha\frac{\hat{m}_{m1}}{\epsilon+\sqrt{\hat{v}_{m1}}} & \cdots & -\alpha\frac{\hat{m}_{mn}}{\epsilon+\sqrt{\hat{v}_{mn}}} \end{bmatrix}$$

Thus $\boldsymbol{\Delta\theta}_{ij} = -\alpha\frac{\hat{m}_{ij}}{\epsilon+\sqrt{\hat{v}_{ij}}}$

4.2.2 Analysis

Given the fact that $\epsilon < 1$ and that it is no longer square rooted. It can be deduced that the magnitude of smoothing used in this method is less than that of the other Adaptive Gradient Methods in this work. This is possibly due to the fact that, there is already a bias correction to prevent the $\hat{v}_{ij}$ term from being zero. So having the usual magnitude of smoothing may further bias the term away from zero, which is undesirable.

5 Experimental Setup and Computational Model

This section will describe various components of the experimental setup such that the experiment can be repeated and the validity of claims made in future components of this work can be tested.

5.1 Computational Environment

5.1.1 Hardware Specification

- : CPU: Intel(R) Core(TM) i5-3570k CPU @ 3.4-3.8 GHz
- GPU: AMD Radeon HD 5700 Series
- RAM: 4GB

5.2 Computational Model

- Artificial Neural Network

5.2.1 Software and Versions

- Operating System: Windows 10
- Python v3.6.0
- numpy v1.14.3
- sympy v1.1.1

5.3 Data Sources

The data can be obtained from

6 Preliminary Results

7 Further Work to be completed

8 Validation, Expected Results and Exceptions

9 Summary

10 References

11 Appendices