Comparing Training Methods for Neural Networks

Albert Parmenter

MAT 3338

Department of Mathematics, University of Dallas, Irving, Texas 75062

Neural networks are becoming an increasingly relevant topic in numerical analysis and one of the key aspects of this topic is how to optimally train these networks minimizing the computational cost while also maximizing the accuracy. In this report the basic structure of neural networks is discussed, as well as a comparison of three training methods: Momentum Based Gradient Optimizer, Root Mean Square Propagation, and Adaptive Moment Estimation.

Neural networks take inputs which act as activations for the first layer of the network. From here the next layer of neurons is given an activation based on the activations of the previous layer and the weights and biases for each neuron in the previous layer (3Blue1Brown).

$$a_i^{(l)} = \sigma\left(\mathrm{w}_1 a_1^{(l-1)} + \mathrm{w}_2 a_2^{(l-1)} + \ldots + \mathrm{w}_n a_n^{(l-1)} + \mathrm{b}\right) \tag{1}$$

Equation 1 describes the activation of a given neuron based on the activation of the previous neurons, denoted by $a_1^{(l-1)}$, where *(l-1)* represents that this neuron is in the previous layer, and the subscript 1 represents that this is the first neuron in that layer. *w* is used to represent the weight associated with the activation of that neuron in the previous layer and b is the bias. σ represents the sigmoid function, which “squishes” the values plugged into to a range to between 0 and 1 (3Blue1Brown).

The goal in training a neural network is to minimize the loss function for a given example. The loss function is given in Equation 2 and represents the mean squared difference for all neurons in the output layer.

$$L = \frac{1}{k+1}\sum_{j=0}^{k}(\hat{a}_j - a_j)^2 \tag{2}$$

Here $\hat{a}_j$ is the predicted value and a_j is the actual value for the activation of the *j*th neuron in the output layer (GeeksforGeeks, *“What is Gradient Descent?”*). Current training methods do not use loss function but instead use the cost function which represents the average of the mean squared errors for multiple training examples. This is given in Equation 3.

$$C(W, b) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=0}^{k}(\hat{a}_j - a_j)^2 \tag{3}$$

Here W is the matrix of weights and b is the matrix of biases. The next step is to find the gradient of the cost function in terms of each of the weights and each of the biases, $\frac{\partial C}{\partial w^{(l)}}$ and $\frac{\partial C}{\partial b^{(l)}}$. For a simple network with single neurons in each layer these are given in Equations 5 and 6 (3Blue1Brown).

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)} \tag{4}$$

$$\frac{\partial C}{\partial w^{(l)}} = \frac{\partial z^{(l)}}{\partial w^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial C}{\partial a^{(l)}} = a^{(l-1)}\sigma'(z^{(l)})2(a^{(l)} - \hat{a}^{(l)}) \tag{5}$$

$$\frac{\partial C}{\partial b^{(l)}} = \frac{\partial z^{(l)}}{\partial b^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial C}{\partial a^{(l)}} = \sigma'(z^{(l)})2(a^{(l)} - \hat{a}^{(l)}) \tag{6}$$

$\nabla C(W, b)$ is the matrix of all the partial derivatives of C is terms of all the weights and biases. The next step is to perform some form of gradient descent (3Blue1Brown).

Stochastic Gradient Descent, or SGD, is a basic version of gradient descent with a set learning rate. The formula for this is given in Equation 7. For SGD and the methods following the biases are updated the same way as the weights.

$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{7}$$

Here η is the learning rate, for this method usually around 0.1 or 0.01. This method has slow convergence and tends to get stuck at local minima (GeeksforGeeks, *"What is Gradient Descent?"*).

The Momentum Based Gradient Optimizer or ML builds off SGD by adding a momentum factor. In other words, the steps in the gradient descent are taken based on the sizes of the previous steps which smooths the trajectory of the descent.

$$v_{t+1} = \beta v_t + \nabla L(w_t) \tag{8}$$

$$w_{t+1} = w_t - \eta v_{t+1} \tag{9}$$

Equation 8 is the formula for updating the velocity and Equation 9 is the formula for updating the weight. Here, v_{t} is the velocity β is the momentum factor (controls how much previous velocity affect updated velocity, usually ≈ 0.9) and η is the learning rate (usually in 0.1 – 0.01 range). ML has faster and more reliable convergence than SGD and works well at avoiding oscillation and avoiding converging on local minima (GeeksforGeeks, *"ML"*).

Root Mean Square Propagation or RMSProp builds on SGD by adding adaptive learning rates for the parameters. This is shown in Equations 10 and 11.

$$s_{\mathrm{t+1}} = \beta s_{\mathrm{t}} + (1 - \beta)(\nabla C(w_{\mathrm{t}}))^2 \tag{10}$$

$$w_{\mathrm{t+1}} = w_{\mathrm{t}} - \left(\frac{\eta}{\sqrt{s_{\mathrm{t+1}} + \varepsilon}}\right)\nabla C(w_{\mathrm{t}}) \tag{11}$$

Here β is the decay rate, η is the learning rate (for this around 0.001), $s_{\mathrm{t+1}}$ is the updated running average of squared gradients. ε is a small constant to ensure stability. When the previous gradients have been large for a given parameter, the steps will be smaller to prevent overshooting. When the previous gradients have been small for a given parameter, the steps will be much larger to speed up convergence. The adaptive learning rate for each parameter allows for faster convergence than SGD. RMSProp also adjusts well to changing optimal parameters. One drawback of this algorithm is that it can perform poorly on sparse datasets (GeeksforGeeks, *"RMSProp Optimizer"*).

Adaptive Moment Estimation or Adam combines ML and RMSProp, such that there are adaptive learning rates as well as a momentum factor. Equation 12 gives the first moment (mean) estimate which is the momentum factor. This is very similar to Equation 8 for ML.

$$m_{\mathrm{t}} = \beta_1 m_{\mathrm{t-1}} + (1 - \beta_1)\, \partial L/\partial w_{\mathrm{t}} \tag{12}$$

Equation 13 gives the second moment (variance) estimate which allows for the adaptive learning rate for a given parameter. This is very similar to Equation 10 for RMSProp (GeeksforGeeks, *"Adam Optimizer"*).

$$v_{\mathrm{t}} = \beta_2 v_{\mathrm{t-1}} + (1 - \beta_2)(\partial L/\partial w_{\mathrm{t}})^2 \tag{13}$$

β_1 and β_2 are the decay rate constants, usually β_1 = 0.9 and β_2 = 0.999. Next m_{t} and v_{t} must be biased such that for the initial iterations of the method these values are not extremely small, since this would cause extremely slow convergence.

$$\hat{m}_t = \frac{m_t}{(1 - \beta_1{}^t)} \tag{14}$$

$$\hat{v}_t = v_t / (1 - \beta_2{}^t) \tag{15}$$

The biased values, $\hat{m}_t$ and $\hat{v}_t$ are then plugged into the weight update equation.

$$w_{t+1} = w_t - \frac{\eta \hat{m}_t}{(\sqrt{\hat{v}_t} + \varepsilon)} \tag{16}$$

Here, η is the learning rate (for this method usually 0.001) and ε is a constant to ensure stability and prevent division by 0 (usually 10^{-8}). Adam has faster convergence due to adaptive learning rate based on past gradients and magnitudes. The bias correction ensures stability. RMSProp has significantly better convergence than ML and RMSProp and is a standard method used in current neural network training (GeeksforGeeks, *"Adam Optimizer"*).

I chose this topic because I have been interested in neural networks for a while and wanted to explore how numerical analysis plays into machine learning. My research on this topic gave me a great introduction to this field.

References

3Blue1Brown. Neural Networks. YouTube, https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi. Accessed 13 May 2025.

"5 Algorithms to Train a Neural Network." GeeksforGeeks, https://www.geeksforgeeks.org/5-algorithms-to-train-a-neural-network/. Accessed 13 May 2025.

"Adam Optimizer." GeeksforGeeks, https://www.geeksforgeeks.org/adam-optimizer/. Accessed 13 May 2025.

ML | Momentum Based Gradient Optimizer – Introduction." GeeksforGeeks, https://www.geeksforgeeks.org/ml-momentum-based-gradient-optimizer introduction/. Accessed 13 May 2025.

"Neural Networks | A Beginner's Guide." GeeksforGeeks, https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/. Accessed 13 May 2025.

Quarteroni, Alfio, Fausto Saleri, and Paola Gervasio. Scientific Computing with MATLAB and Octave. Springer, 2010. PDF, https://hlevkin.com/hlevkin/92usefulBooks/Octave/Scientific%20Computing%20wit %20MATLAB%20and%20Octave%20Quarteroni,%20Saleri%20&%20Gervasio.pdf. Accessed 13 May 2025.

"RMSProp Optimizer in Deep Learning." GeeksforGeeks, https://www.geeksforgeeks.org/rmsprop-optimizer-in-deep-learning/. Accessed 13 May 2025.

"What Is Gradient Descent?" GeeksforGeeks, https://www.geeksforgeeks.org/what-is gradient-descent/. Accessed 13 May 2025.