A Method to Accelerate Optimization in Convolutional Neural Network

Venkateshwaran P, V.Naveen Sreeram, Vamsi Krishna.M, Mouli K, Bala Tripura Sundari B[a]
vamkri259@gmail.com, venkateshwaran.p@yahoo.com, naveensreeram555@gmail.com,
kmouliswaran@gmail.com, [a]b_bala@cb.amrita.edu
Department of Electronics and Communication Engineering
Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India.

Abstract— Optimizer play an important role in training of the network model. The optimizer's algorithm for updating of the weights determines the performance of the network model. Traditionally training of the network model takes longer duration of time. Each optimizers employs its own unique algorithms which takes its corresponding time for the training phase. In this paper we have tried to compare the efficiency of our proposed Error-based decay algorithm with No decay and Time-based decay algorithm for an Adam optimizer. The metric used here to evaluate the optimizer performance is loss function.

1. INTRODUCTION

In recent times, neural networks are playing a major role in fields like computer vision, natural language processing and are delivering good results. In the process of modelling a neural network for a particular classification problem, a number of parameters should be set. Optimization algorithm is one of the parameters that are most influential. Tuning of such parameters improves the performance of the model like in terms of the number of iterations for training and prediction accuracy.

2. CONVOLUTIONAL NEURAL NETWORK:

Convolution neural network is widely used in recognition applications because of its specialty in recognition of patterns with minimal pre-processing directly from pixel images. A traditional CNN is composed of a convolution and pooling layer followed by the fully-connected layer as shown in Fig 1. Here, in this paper we take the example of digit recognition network model for analyzing the optimization process. The CNN model here is loaded with the input grey-scale image of dimension 28x28. The convolution layer consists of 32 filters of each dimension 5x5 along with a pooling layer having both stride and pool size as 2. Following is the flatten layer and two fully-connected layers. The 2nd fully-connected layer which is the output layer consisting of 10 neurons corresponding to the predicted digit in the input image.

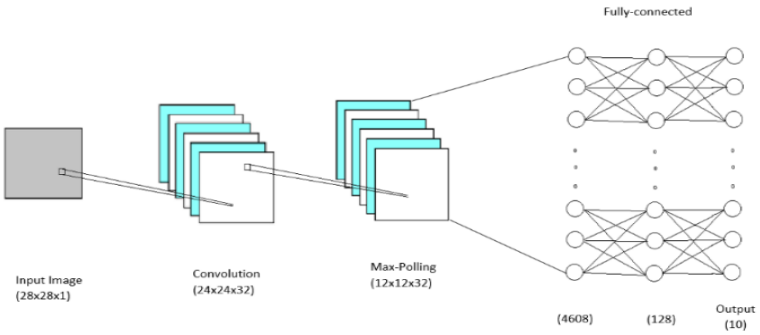

Fig 1: Architecture of CNN model

In a CNN, the whole recognition process is divided into two main stages namely feature extraction and prediction. In the feature extraction stage there is convolution layer and the pooling layer that extracts the features from the input image loaded. The prediction part is done by the fully-connected layers which predicts the information present in the image based on the features extracted from convolving. [2]

2.1 Convolution Layer:

This layer is the primary building block of CNN. It consists of kernels or filters which are responsible for the retrieval of features from the input images. During each forward pass the input image is convolved with each of the filters to form its corresponding feature map. These feature map are stacked to form the output of the convolution layer. This output obtained is further passed through an activation function. [3]

2.2 Pooling Layer:

Here in this layer, the output from convolution layer is given as its input. This layer's function is to reduce the spatial size of representation is a progressive manner. It reduces the parameters and computation in the network. The commonly used pooling technique is max-pooling where the maximum value among the pool is considered as the whole value of that particular pool. [2]

2.3 Fully-Connected Layer:

This layer performs high-layer of reasoning by connecting all neurons in the previous layer with every neuron in the current layer. These layers have weights that get updated through training of the network. Using the training data set, the network improves the accuracy and minimizes the loss in prediction of the input images. [2]

3. FUNCTIONING OF CNN

In the process of creation of a network model, there is two phases which is training and testing. In both of these phases the input data is first pre-processed in the form suitable for our network model. Feature extraction is the next stage in both the phases where vital information from the data is extracted using which the model classifies in the later period of process. In CNN convolution method is used for extraction of features. The third stage is where the training and testing phase differs. In training phase after feature extraction the estimation of model is done and weights of the network model is updated. For testing phase the extracted feature is compared with known actual model to find the best suiting class for it. [5]

4. PARAMETERS OF CNN

4.1 Weights:

In a network the neurons are connected with other neurons by means of communication links which are associated with the weights. The output of the neuron is obtained by taking the dot product of the inputs and their corresponding weights.

4.2 Bias:

Bias is an offset value which generally has a value 1. It always makes the neuron a non-null value.

4.3 Activation Function:

A threshold value is initialized and then compared with the calculated value through which the network's output is obtained. There are two states for an activation function namely inhibition state and exhibition state. In inhibition state; if the condition is not satisfied after comparing the calculated value with threshold then the neuron doesn't fire which means it produce a value zero and exhibition state is the vice versa of the inhibition state.

4.4 Learning Rate:

It determines how fast and accurate a network can reach its minima. It controls the amount of weight adjustments at each step during training. The learning rate value depends on the network but its value lies between 0 and 1 always.

5. TRAINING OF CNN

In CNN, the Training phase is when the model gains knowledge. The training data set consists of both the input and its corresponding output with it. The output value from the network model when passing a training data set is compared with the target output and weights are updated based of it. There are various techniques used to adjust the weights of a neuron in a CNN.

5.1 Feedforward Backpropogation:

Feedforward Backpropagation is an error based weight updation algorithm. In this algorithm we assign the values at the input layer, and the output is calculated. The error is obtained by comparing the derived output and the actual output. The error is fed backwards to the hidden layers in order to update the weights. This process continues until it reached the specified error rate. [1]

The algorithm is as follows:

- The input values, weights and learning rate are initialized.
- The output is calculated at the last node and then compared with the actual value to get the error rate.
- Consideration of incremental values, which shows us how to change the weight of the neuron.
- Modifying the weight in form of every neuron to minimize the local error.
- Setting the learning rate which is a hyper parameter that controls how much we are adjusting the weights of our network with respect to the loss gradient. [1]

5.2 Optimizer:

Whenever training of a network model happens the focus is on updation of weights in network model to minimize the loss function. Doing so makes our predicted output matched to the desired output. Optimizer here function in updating these weights. They compute the loss function and employ a particular algorithm to update these weights. The updation of weights and minimization of loss function happens after each epoch. The training continues until the local minimum is reached. All optimizers are based on the principle of Back-propagation. Gradient descent is an algorithm conventionally used for optimization. It computes the gradient of the cost function and updates the weights accordingly. The high variance oscillation in gradient descent lead to the development of many new optimizers like Particle Swarm Optimizer [9].

5.2.1 Adam Optimizer:

An Adam optimizer is a combination of RMSProp [4] and Stochastic Gradient Descent with momentum. This optimizer uses the squared gradients to scale the learning rate. Adam is an adaptive learning rate method where it computes individual learning rates for different parameters. Its name is derived from adaptive moment estimation. Adam uses first and second moments of gradient to adapt the learning rate for each weight in the network. [4]

5.2.2 Decay:

Learning rate is a parameter which determines the step size which guides us to the path of local minima. When the step size is small it takes more time to reach the minima and when the step size is large it can reach the minima faster but it may result in overshooting. Ideally the learning rate should decay for faster convergence. [10]

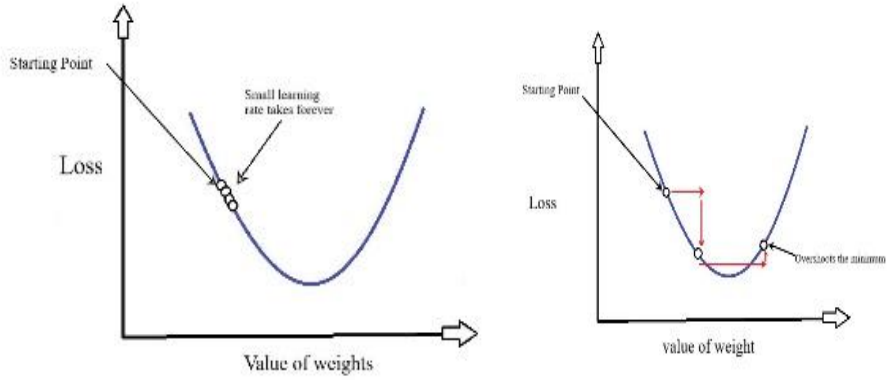

Fig 2: comparison of small & large learning rate

1) Time-based:

The learning rate is constantly reduced by a factor of time with respect to the number of iterations.

2) Error based:

The learning rate is reduced by time. At the beginning of the optimization process, the step size is made bigger until it reaches a point which is closer to local minima and from there on the step size is gradually decreased until it reaches the minima.

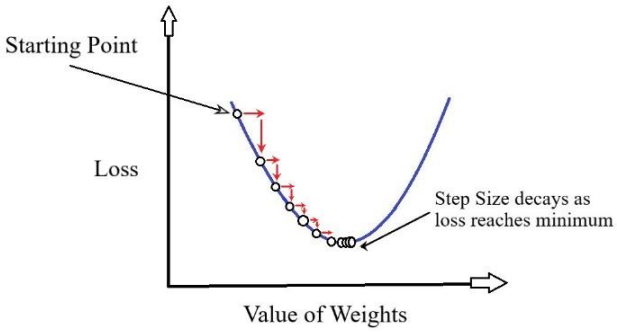

Fig 3: Decaying Error-based learning rate

6. RESULTS AND DISCUSSIONS

The dataset used for evaluation is in Table1. Here we have taken two optimizers namely, Adam and Adam with Time- based decay and compared their performance with our proposed algorithm model for optimization. In our algorithm model we have used the error based technique to update the weights.

Table 1: Performance of Adam Optimizers for Sample Data (with error rate:1e-6)

Sample Data Set(Y=mX+c)	Adam(no decay)	Adam(TIME BASED DECAY)	Adam(ERROR BASED DECAY)		
X= [1,2,3,4,5] Y=[5,7,9,11,13]	m=2.0014 c=2.9962 iteration=10088 Error=$3.105*10^{-6}$ Accuracy=99.9968	m=2.00083 c=2.9975 iteration=12952 Error=$1.23*10^{-6}$ Accuracy=99.9987	m=2.00157 c=2.9941 iteration=9293 Error=$7.09*10^{-6}$ Accuracy=99.9929	m=2 c=3	Y=2X+3
X= [1,2,3,4,5] Y=[3,4,5,6,7]	m=1.0015 c=1.9942 iteration=6921 Error=$6.98*10^{-6}$ Accuracy=99.9993	m=1.0085 c=1.9978 iteration=7597 Error=$1.025*10^{-6}$ Accuracy=99.9998	m=1.0011 c=1.9968 iteration=6868 Error=$2.08*10^{-6}$ Accuracy=99.9997	m=1 c=2	Y=X+2
X= [1,2,3,4,5] Y=[6,9,12,15,18]	m=3.0003 c=3.0003 iteration=10099 Error=$4.9933*10^{-10}$ Accuracy=99.9999	m=3.0000020 c=3.0000014 iteration=12968 Error=$3.25*10^{-7}$ Accuracy=99.99996	m=2.99966 c=2.99966 iteration=8840 Error=$1.41*10^{-7}$ Accuracy=99.99998	m=3 c=3	Y=3X+3
X= [1,2,3,4,5] Y=[-1,0,1,2,3]	m=0.9983 c= -1.9938 iteration=8502 Error=$7.73*10^{-6}$ Accuracy=99.9992	m=0.9990 c= -1.9963 iteration=9913 Error=$2.8*10^{-6}$ Accuracy=99.9997	m=0.99913 c= -1.9977 iteration=9682 Error=$1.12*10^{-6}$ Accuracy=99.99989	m=1 c= -2	Y=X-2
X= [1,2,3,4,5] Y=[1,-1,-3,-5,-7]	m= -1.9985 c= 2.99435 iteration=14828 Error=$6.71*10^{-6}$ Accuracy=99.9993	m= -1.8011 c=2.2708 iteration=38541 Error=0.09666 Accuracy=90.33	m= -1.999 c=2.9979 iteration=15403 Error=$9.005*10^{-7}$ Accuracy=99.9999	m=-2 c=3	Y= -2X+3

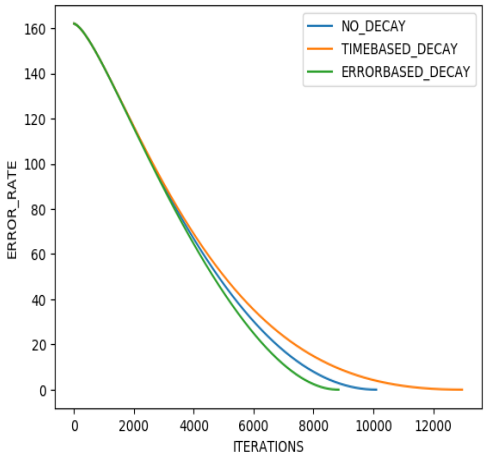

Fig 4: Error rate

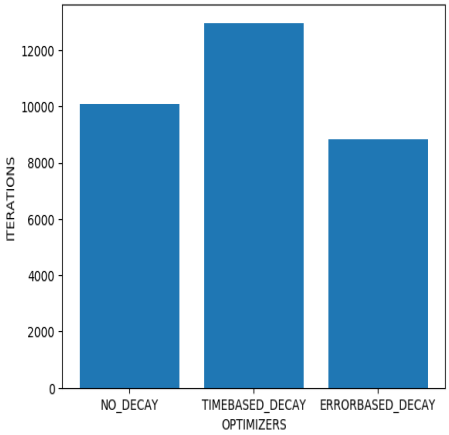

Fig 5: No of iterations

The above graphs in figure 4 and figure 5 are the calculated error rate and number of iterations respectively for an Adam optimizer with No decay, Time-based decay and our proposed Error-based decay for our sample data set 3 from Table:1.

From the above graphs we can interpret that for a given set of data:

a) Adam optimizer with Error-based decay takes less number of iteration than the Adam with No decay and Adam with Time-based decay.
b) Adam optimizer with Error-based decay has tolerable less accuracy compared to Adam with No decay.

Adam with our Error-based decay is accurate enough as well as time efficient. Therefore, it is evident that, there is a trade-off between accuracy and time. In our Error-based decay we were able to balance this trade-off and obtain optimum results.

REFERENCES

[1] R. Arnold and P. Milkos, "Character recognition using neural networks", *2010 11th International Symposium on Computational Intelligence and Informatics (CINTI)"*, Budapest, pp. 311-314, 2010.

[2] A. Bhandare, M. Bhide, P. Gokhale and R. Chandavarkar "*Applications of Convolutional Neural Networks*", in International Journal of Computer Science and Information Technologies, Vol. 7, no. 5, pp. 2206-2215, 2016.

[3] A. Ardakani, C. Condo, M. Ahmadi and W. J. Gross, "An Architecture to accelerate Convolution in Deep Neural Netwroks", in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 4, pp. 1349-1362, April 2018.

[4] A. M. Taqi, A. Awad, F. Al-Azzo and M. Milanova, "The Impact of Multi-Optimizers and Data Augmentation on TensorFlow Convolutional Neural Network Performance", 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), pp. 140-145, Miami, FL, 2018.

[5] Anil R., Manjusha K., Kumar S.S., Soman K.P. "Convolutional Neural Networks for the Recognition of Malayalam Characters". In: Satapathy S., Biswal B., Udgata S., Mandal J. (eds) Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014. Advances in Intelligent Systems and Computing, vol 328. Springer, Cham, 2015

[6] Neena A., Geetha M. "Image Classification Using an Ensemble-Based Deep CNN". In: Sa P., Bakshi S., Hatzilygeroudis I., Sahoo M. (eds) Recent Findings in Intelligent Computing Techniques. Advances in Intelligent Systems and Computing, vol 709. Springer, Singapore, 2018.

[7] Y. Hou and H. Zhao, "Handwritten digit recognition based on depth neural network," *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, pp. 35-38, Okinawa, 2017.

[8] J. Dong, H. Zheng and L. Lian, "Dynamic Facial Expression Recognition Based on Convolutional Neural Networks with Dense Connections," *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 3433-3438, Beijing, 2018.

[9] F. van den Bergh and A. P. Engelbrecht, "Training product unit networks using cooperative particle swarm optimisers," *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, Washington, DC, USA, 2001, pp. 126-131 vol.1.

[10] J. Zhang, F. Hu, L. Li, X. Xu and Z. Yang, "AEDR: An Adaptive Mechanism to Achieve Online Learning Rate Dynamically," *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, Boston, MA, 2017, pp. 30-36.7.