Mohammad Shahriar Hooshmand
Ph.D. Candidate
Department of Materials Science and Engineering
The Ohio State University

Neural Network
Project 1: Parity Learning

The purpose of this project is to learn the parity problem using two-layer perceptron structure. The code is written in Python (v.3). Initial weights and bias terms are generated using the random function in Python. The Numpy library is used for the array operations. The code is generalized to make the desired number of nodes in the hidden layer. To better compare results, "seed" function is used in the randomized weights. This helps to directly compare the effect of momentum and learning rates in convergence without dealing with various initial weight initiation.

In general learning algorithms, we feed in a wisely chosen inputs to learn and then predict the other sets of inputs without already observing them. However, since the total number of 16 input samples are present in this case, we let the code learn all of the samples without considering the cross-validation error. Based on the project instruction, the absolute error of 0.05 is considered as the convergence criterion instead of mean squared error. The input array for the parity problem has 4 elements. The bias multiplier (± 1) is considered as the fifth element of input vector. The weights for hidden layer are presented by 5X4 matrix (named "layer1" array), which stores the weights corresponding to each of the 5 inputs (row of the matrix) connected to each of the 4 hidden nodes (columns of the matrix). The weights of hidden layer connected to the one output layer is presented by the 4X1 vector named "layer2".

The scope of this project is the online learning pattern. Each epoch is done when the code sees all the 16 input samples and accordingly updated the weights following the backpropagation adjustment for these samples. At the end of each epoch, maximum absolute error over all the updated 16 samples is calculated and checked to see if the convergence achieved. This process is repeated until the absolute error for the epoch is less than 0.05. The same procedure is pursued for all of the learning rates given in the instruction. The details of how the code is performing the learning and backpropagation are commented in the .py file.

Email: hooshmand.1@osu.edu
Website: http://u.osu.edu/hooshmand.1

Mohammad Shahriar Hooshmand
Ph.D. Candidate
Department of Materials Science and Engineering
The Ohio State University

The learning rate parameter η plays an important role in the convergence speed by determining how much the error from desired value would contribute to the weight update. The **number of epochs** performed in each trial is considered as the convergence speed criterion. Figure 1 shows that the increase in learning rate parameter lead to the faster convergence. The learning rate parameter can be interpreted as the step size of each weight update in the gradient descent method. It is important noting that in case of reaching the proximity of minimum well, high learning rate can produce oscillation over the local minimum resulting the slow convergence. To solve this, we introduce momentum term α , that basically acts as a driving force to push the weight adjustment toward the local minimum well in each step of the weight update. Implementing this parameter can highly contribute to the fast convergence in the opposite direction of gradient. As it is depicted in the Figure1 and Figure2, introducing the momentum enhances the convergence speed by the factor of 3.5 to 7.3 from lower learning rates to the higher learning rates, respectively. It is worth mentioning that, the general trend is of our interest since the influence of these parameters greatly depends on how far/close we are to the local minima (Occurrence of few outliers is consistent with this). Figure 3-5 compares the effect of momentum in the rate of convergence for three chosen values of $\eta = 0.05,0.25,0.5$ as a representative of low-high learning rates, respectively. As it is expected, the general trend shows that the algorithm minimizes the error faster as the momentum parameter enhances for either the lower learning rate of 0.05 to the higher learning rate of 0.5. However, the efficiency of higher speed of convergence can be different depending on the choice of learning rate. So, it is highly recommended to optimize both learning rate and momentum parameters accordingly to achieve the efficient and fast convergence.

Email: hooshmand.1@osu.edu
Website: http://u.osu.edu/hooshmand.1

Mohammad Shahriar Hooshmand
Ph.D. Candidate
Department of Materials Science and Engineering
The Ohio State University

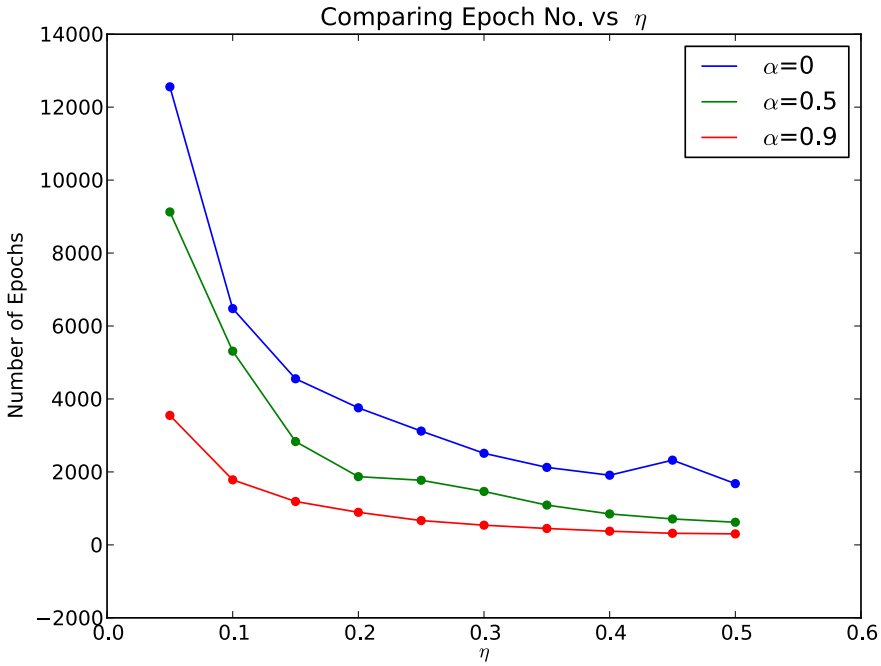

Figure 1: The number of epochs performed to reach the convergence versus the learning rate parameter. Implementing the momentum enhances the speed of convergence.

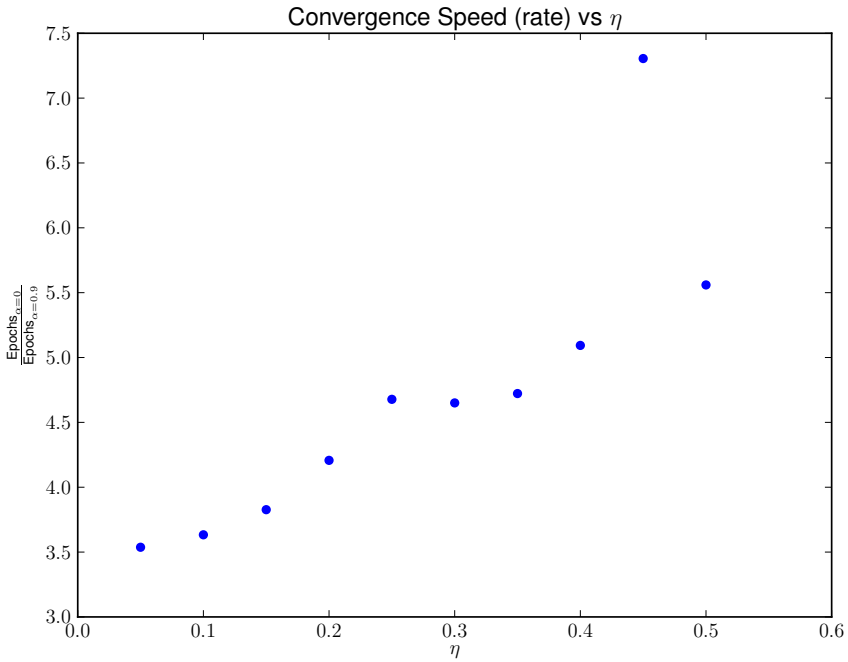

Figure 2: The effect of introducing the momentum on the convergence rate. The convergence speed for each learning rate is defined as the rate of the total number of epochs without momentum to the number of epochs with $\alpha = 0.9$.

Email: hooshmand.1@osu.edu
Website: http://u.osu.edu/hooshmand.1

Mohammad Shahriar Hooshmand
Ph.D. Candidate
Department of Materials Science and Engineering
The Ohio State University

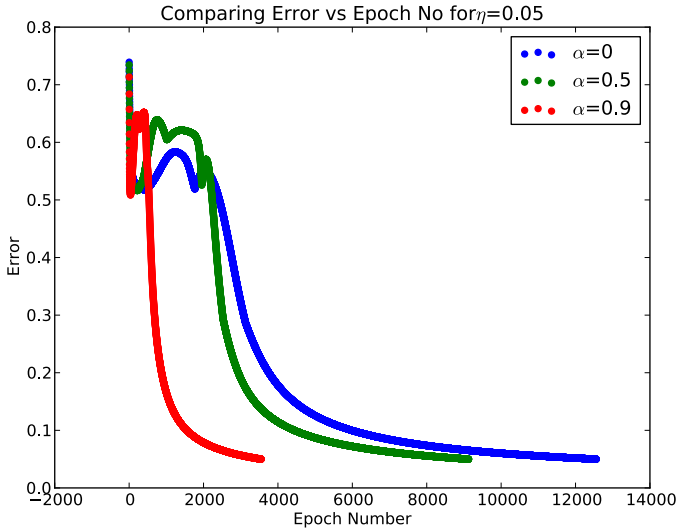

Figure 3: The rate of error reduction as a function of total number of epochs for a particular learning rate $\eta = 0.05$. Introducing the momentum leads to the faster error reduction and as a result, quick convergence

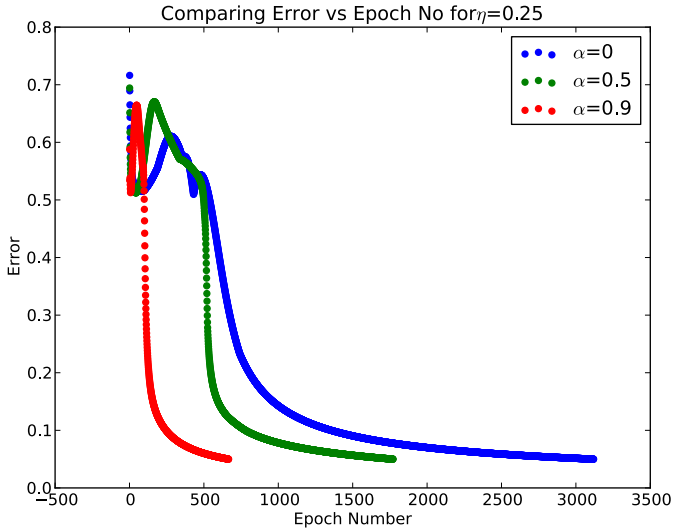

Figure 4: The rate of error reduction as a function of total number of epochs for a particular learning rate η=0.25 . Introducing the momentum leads to the faster error reduction and as a result, quick convergence

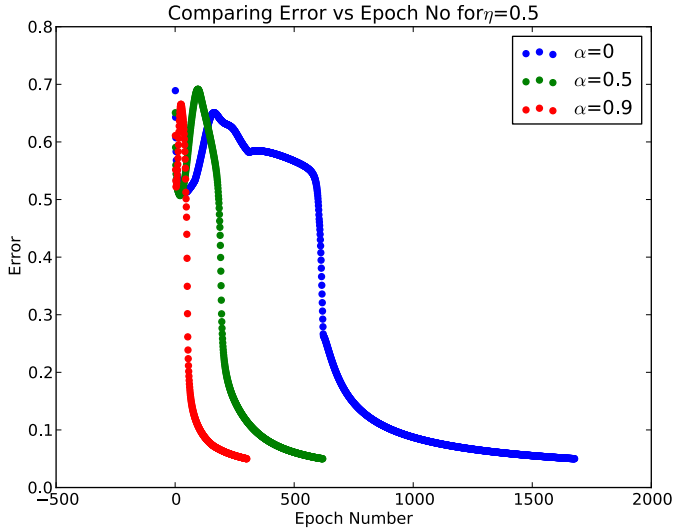

Figure 5: The rate of error reduction as a function of total number of epochs for a particular learning rate η=0.5 . Introducing the momentum leads to the faster error reduction and as a result, quick convergence

Email: hooshmand.1@osu.edu
Website: http://u.osu.edu/hooshmand.1