CSE 5526 - AUTUMN 2020 INTRODUCTION TO NEURAL NETWORKS PROGRAMMING ASSIGNMENT 1

A PREPRINT

Ibrahim M. Koc
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, OH 43201
koc.15@osu.edu

September 30, 2020

1 Discussion

Since we learned gradient descend algorithm in the class, we will apply this for our solution.

We know the weight update equation for the final layer is as in Eq. 1

$$w_{kj}(n+1) = w_{kj}(n) + \eta\delta_k(n)y_j(n), \quad \text{where} \tag{1}$$
$$\delta_k(n) = e_k(n)[y_k(n)(1-y_k(n))] \tag{2}$$

For other layers weight updates can be written as in Eq. 3

$$w_{ji}(n+1) = w_{ji}(n) + \eta\delta_j(n)x_i(n) \quad \text{where} \tag{3}$$
$$\delta_j(n) = [y_j(n)(1-y_j(n))][\delta_k w_{kj}(n)] \tag{4}$$

For faster calculation instead for loops using matrices with these equations would be beneficial. Let us rewrite the equations with matrices. Let our input vector be denoted as $\overline{x}$ and output vector be denoted as $\overline{y}$:

Forward:

$$\overline{y}_j = \sigma(W_j\overline{x}_i))$$
$$\overline{e} = (desired - \overline{y}_k)$$

Backpropagation:

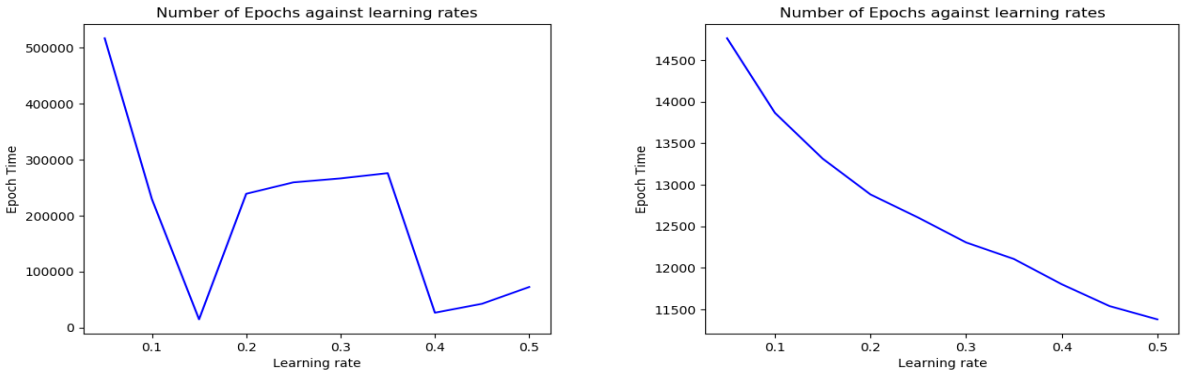
$$\overline{\delta_{last-layer}} = \overline{e} \circ [\overline{y}_k \circ (1-\overline{y}_k)]$$
$$\overline{\delta}_{inter-layer} = W^T\overline{\delta}_{inter-layer} \circ [\overline{y} \circ (1-\overline{y}_j)]$$

Weight update:

$$W_i = W_i + \Delta W_i, \quad \text{for any weight matrix}$$

2 Results

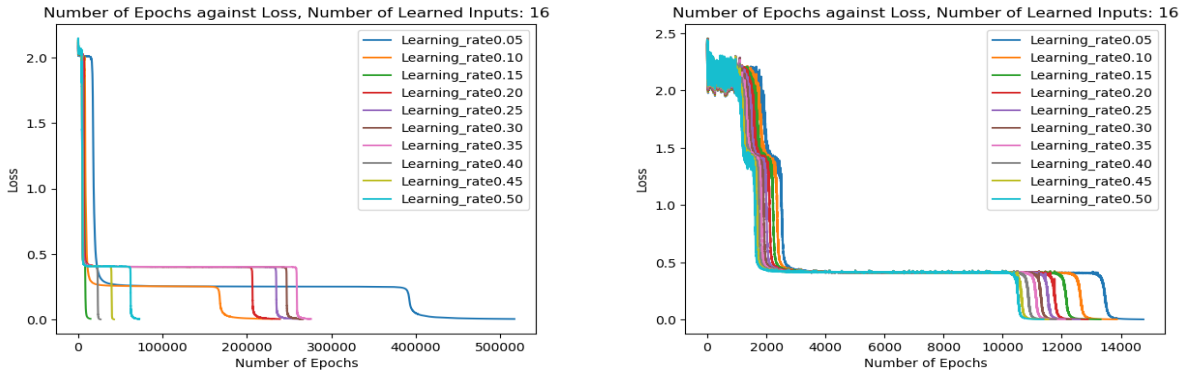
One can see in Fig. 1 the epoch time vs learning rate characteristics without the momentum term in Fig. 1a, and the epoch time vs learning rate characteristics with the momentum term in Fig. 1b. As can be seen from the figures, the momentum term speeds up the process massively for the same learning rate; furthermore, it also stabilizes the gradient descend so that increasing the learning rate actually helps to converge quickly and consistently.



(a) Epoch times vs. Learning Rates without momentum (b) Epoch times vs. Learning Rates with momentum

Figure 1: Epoch times vs. Learning Rate Characteristics

The reason for this could be easier to understand if we look at Fig. 2 in that one can easily see the reason from the loss vs. epoch time characteristics. In Fig. 2a the loss vs. epoch time without the momentum term characteristics demonstrates the unstable behaviour of loss term during the learning, whereas in Fig. 2b the loss vs. epoch time with the momentum term characteristics the loss graph for learning rate 0.05 and 0.5 is almost similar in terms of loss fluctuations albeit one is 10 times larger.



(a) Epoch times vs. Learning Rates without momentum (b) Epoch times vs. Learning Rates with momentum

Figure 2: Epoch times vs. Learning Rate Characteristics

Therefore, one can easily see that learning is slow and unstable without the momentum term whereas it is fast and stable with the momentum term. Please note that all learning process starts from the same random initialization in which it converges for even learning rate 0.05 without the momentum terms. In other words, after various trials, a good random initialization for weights are chosen for Fig. 1 and Fig. 2 above.

3 Conclusion

The constructed neural network is indeed solved a problem that is not linearly separable, due to its nonlinear nature. Furthermore, we have seen that there is a trade of between learning rate and the convergence time; however, if we increase the learning rate too much, it may cause our algorithm to be unstable. On the other hand, even for the smallest

learning rate, the network got stuck in a local minima several times, this could be due to the fact that modulo operation is a nonlinear function, albeit this is a simple 4 input 1 output, 16 possible input-output pair problem.

Another aspect is the momentum term α. Adding the momentum to the equation helped the network converge much faster.

For project's github link: `https://github.com/mertkoc/simpleNeuralNetwork`