Adaline classifications

By:

Aviya Oren - 322273301

Maya Hayat - 322515669

Table of Content

Background

This exercise aims to use the Adaline algorithm to correctly classify the followings:

- “Bet” vs “Lamed”
- “Bet” vs “Mem”
- “Lamed” vs “Mem”

In the first part of the exercise we had to write the three letters 3 times each in handwriting on a 10x10 matrix, then turning each letter 15 degrees to the left and 15 to the write in order to create some noise and make it harder for the algorithm to classify them.
After having written the letters we had to first mark the first cell in the matrix (mat[0][0]) by 1, 2 or 3 for “Bet”, “Lamed” and “Mem” respectively. The rest of the grid is filled out as follows: 1 if there’s an entry, -1 otherwise.

The algorithm is trained on 80% of the data and then tested on the remaining 20%, this process repeats itself 5 times for each 20% of the data in order to make sure our data division isn’t biased.

Evaluation

The Adaline algorithm was used to classify the letters as explained above, in this section we'll evaluate our results and explain our hyped parameters.

Hyper Parameters

Epochs number -
the number of training epochs or iterations. We chose it according to the 'lr' hyperparameter so that the accuracy of the algorithm would be maximal, and in addition its running time would be reasonable.

Tolerance - To avoid overfitting we used a threshold of 0.5 for the n_epoch, if the difference between this round and the previous round's error is smaller than 0.5 , we will finish training the data and will move onto the next part which is testing our classification algorithm.

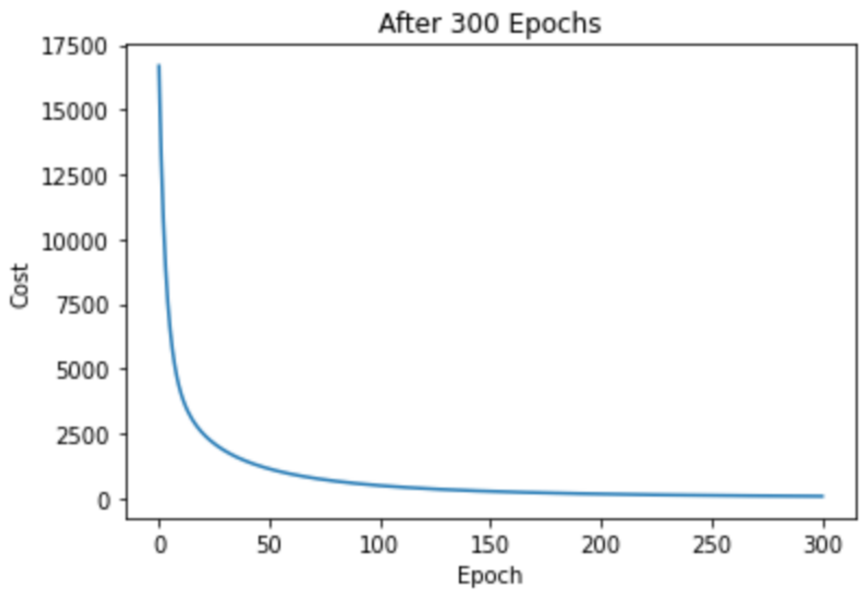
Learning rate -
Choosing an appropriate learning rate is crucial to ensure effective training of the Adaline model. We want to maximize the accuracy but without spending too much time running. We chose the 'lr' to be 0.0001 with appropriate and not too high 'n_epoch', such that We created a balance between the learning rate of the algorithm and its performance.

Important notes

Note that each time we run our code we get different accuracies, although they're all in about the same range (83%-89%) it may change as we shuffle our data which might affect the results as well as the initialization of the weights and bias at the beginning of the algorithm is taking place randomly therefore the random weights could be closer to farther away from what they should be. This randomness also affects the number of epochs taking place before converging.

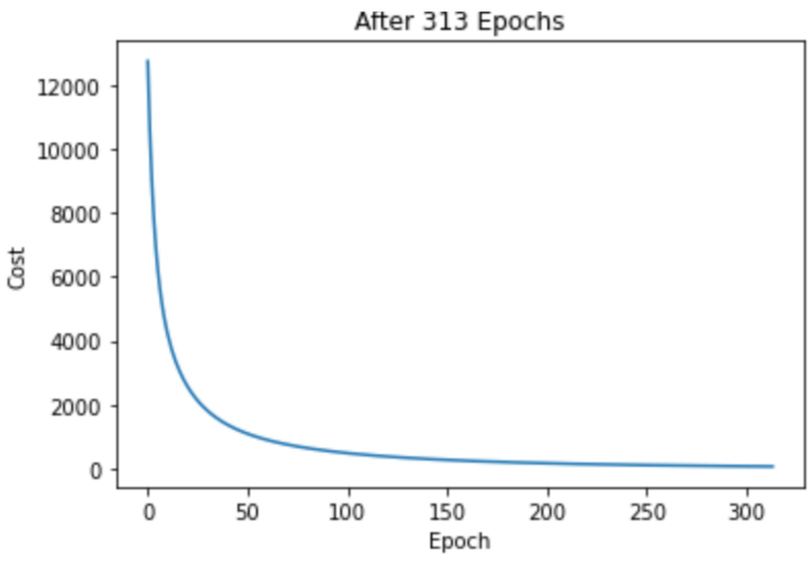
“Bet” vs “Lamed”

The first comparison was bet vs lamed, after having shuffled our data we’ve divided it into 5 equal sections and used 1 of them as the testing data and the remaining 4 as our training set.
After running the code using each section once as the testing set we got the following results: average accuracy is 87.23% and the standard deviation is 4.67%. Although the max number of epochs the algorithm could run is 1000 we noticed that it usually stops after ~290 epochs.



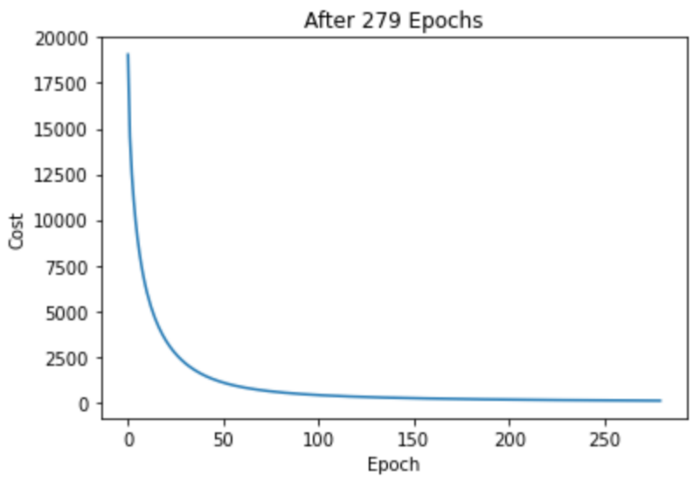
“Bet” vs “Mem”

Next we compared Bet and Mem, which in handwriting are pretty similar therefore, from the beginning we already expected the accuracy to be a little lower than the other two comparisons. As we anticipated the accuracy was 84.82% with std of 4.04% with approximately 270 epochs taking place before terminating.



“Lamed” vs “Mem”

The last two letters compared were Lamed and Mem, those two are fairly different, and we got accuracy of 86.59% and standard deviation of 4.63% and stopped after ~280 epochs.



The Adaline Algorithm

The Adaline algorithm is a linear classification algorithm that aims to minimize the mean square error between activation and target values by adjusting the weights of the linear neurons.
The algorithm works as follows:

1. Initial all weights randomly, of course all should be above 0.
2. Provide the input features X and corresponding target values y (1,2 or 3).
3. Calculate the net_input z by calculating the dot product of input features with the weights and adding the bias term.
4. Pass the net_input through an activation function. In the Adaline algorithm, the activation function is the identity function, so the output of the activation function is simply the net input.
5. Calculate the error which is the difference between the actual output and the predicted output.
6. Update the weights and bias using the error and the input features.
7. Repeat steps 3-6 for all the input features in the training set (the 80% of the data).
8. Convergence check: Repeat steps 3-7 for a fixed number of epochs or until the cost function converges (when the difference between the last two costs is less than 0.5).
9. When the weights have been learned , we use them to predict the output for new input features (the 20% data called the test).