Dogan Aykas
6227725

ACML Assignment 1: Neural Networks

Explanation of the code

For the implementation of the homework, I have chosen Python to program the steps necessary to apply gradient descent on the neural network model. The libraries used are numpy and matplotlib for dealing with vectors and some graph depictions, respectively. By setting the random seed to a stationary value to I was able to observe the parameters to optimize the model. Since there is no built-in function for python, one was created in order to calculate the activation values of neurons which we will explain in detail later. The assignment starts with 8 input nodes for the initial layer, each having one value of 1 and the rest zero, corresponding to each neuron. Since there are 8 training examples to optimize the model, we created a list with all training examples represented as vectors.

Functions

• Feedforward function: Although the content of the feedforward function is to be repeated for the implementation function, it became handy when trying to check if the activations of the output neuron have actually converged to the intended values exactly the same as the input variables. The feedforward function takes the input as x parameter, and calculates the z values of the neurons in the hidden layer and the output layer as variables named as z2 and z3. For the activations of the neurons in the hidden layer and output layer, previously defined sigmoid function is used to transform the Z values to activation values of nodes. The function returns the calculated z and a-values in the end. For clarification, the returned z and a values are calculated as follows: o z2 = (np.matmul(W1, x) + b1) o z3 = (np.matmul(W2, a2) + b2) o a2 = sigmoid(z2) o a3 = sigmoid(z3)

• Backpropagation function: This is the function where the model learns and adjusts the weights and biases of the matrix. It takes the iteration amount, alpha and lambda values as inputs and returns the adjusted weight matrices and bias vectors. The function starts with initializing the matrix containing weights that connects inputs to hidden layer as W1 and the weights that connects hidden layer to output layer as W2. W1 is of shape 3*8 since there are 3 hidden neurons and 8 input neurons and W2 is of shape 8*3 since there are also 8 output nodes. Biases are initialized as vectors which has 3 rows for b1 and 8 rows for b2, each corresponding to the right neuron. The matrices and vectors are initialized with random values between 0 and 1. After that, delta W and delta b are initialized as 0 vectors and then the derivations of both are calculated for each layer. For each training example, the derivations are calculated and added cumulatively. The average of these derived weight matrices and bias vectors are calculated for each iteration and and subtracted from the first initialized weights and matrices with parameters alpha and regularization term lambda is added to control the boundaries of weights and biases. The function returns the adjusted W1, W2, b1 and b2 matrices at the end. After receiving adjusted weights and biases, we run the feedforward function once again to check if the results of the output neurons are close to our assumptions.

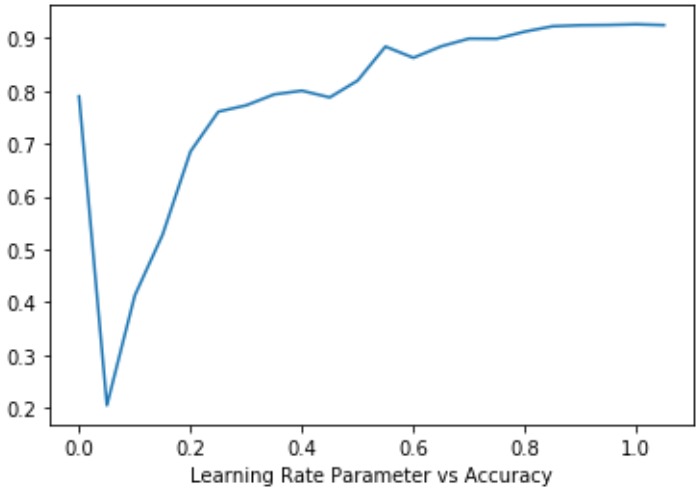
Brief description of the learning performance of your network

The network requires 3 hyperparameters to be chosen manually, which are number of iterations, and the values for learning rate and weight decay parameters.

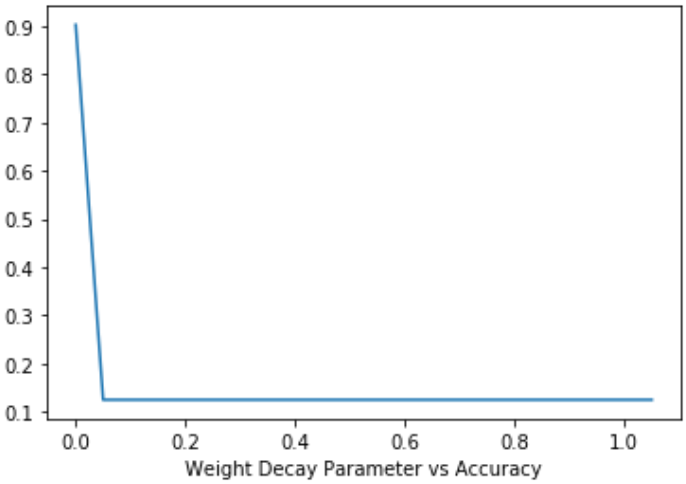
Dogan Aykas
6227725

During my experimentation regarding the number of iterations, usually 100,000 iterations gave the necessary convergence that is needed for the model. Although the related output nodes did not converge exactly to giving values of 1, it was close to something like 0.97-0.98 when the number of iterations is indicated as above.

I have done a small experimentation on which value to take for learning rate. For this experiment, I have taken the number of iterations as 10,000 for convenience and time sake. Below, you can see the graph of an average accuracy vs the value taken as learning rate. The average accuracy depicted in the y-axis is not an official metric but something I have represented as averages of the correct output nodes activated after the weights are learned.



A similar experiment was conducted on which value to take the weight decay as. Interestingly, the same average accuracy calculated above was the highest during when the weight decay was taken as zero. In fact, when it is taken as something other than, the model did not converge to the intended outcomes.



Dogan Aykas
6227725

According to the experimentations for the selection of hyperparameters, I have taken the learning rate as 0.85 and weight decay as 0 for the final version of the neural network. The number of iterations were increased to 1,000,000 for the final neural network for best performance, which showed the best performance that the desired output nodes' activations converged to 0.99.

Interpretation of the learned weights

The learned weights of a neural network is usually a black box and thus hard to interpret with human eye. Still, we can inspect the values for the weights and biases after the backpropagation, below.

```
W1 = ([[ 4.86170246, -2.83199226,  6.31077546, -5.43253138, -3.22927088,
         1.59847906, -4.57978743,  4.67220003],
       [-1.17833605,  4.6307884 ,  6.71036666,  4.39668447, -3.70609407,
         1.33551276, -4.63427531, -5.35851288],
       [ 6.41428751,  6.16604059,  0.81400597, -0.86868162, -3.59076653,
        -6.64995992,  2.11260624, -0.62901002]])

W2 = ([[  9.10062021,  -5.82245087,  15.16866961],
       [ -6.26991442,   8.88540071,  13.63002566],
       [ 11.55967384,  13.00624992,   3.08086252],
       [-13.14656556,  13.70357729, -13.27716444],
       [ -8.49434189,  -9.14893879, -13.1984179 ],
       [  8.70240328,   7.49175148, -25.81023761],
       [-11.12444605, -12.47007147,  12.16225687],
       [ 13.93226474, -15.42140252, -11.60718592]])

b1 = ([[-0.68854839],
       [-0.71978381],
       [-0.29632253]])

b2 = ([[-18.40539703],
       [-17.10305714],
       [-21.61693866],
       [ -5.17054947],
       [  5.79128491],
       [ -6.29223379],
       [ -5.27384002],
       [ -5.41481104]])
```

The magnitude of the weights depict how much of an impact the particular activation from the source node to the target node. Weights with values that are close to 0 are examples of signals that are not transferred. Furthermore, when the biases are checked, it can be seen that the biases for the output layer has more of an impact than the biases for the hidden layer. This might be due to the fact that when the information/input coming from 8 input layers are suppressed to 3 hidden nodes, the biases for the output layer plays a more critical role to retrieve some of the possibly lost information in the hidden layer.