PROBLEM SET 1

CMSC 828L

Deep Learning

Harika Pendli

117501421

University of Maryland, College Park

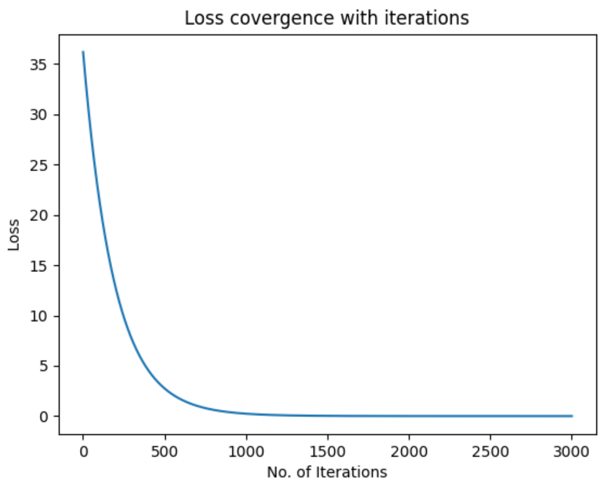
Email: hpendli@umd.edu

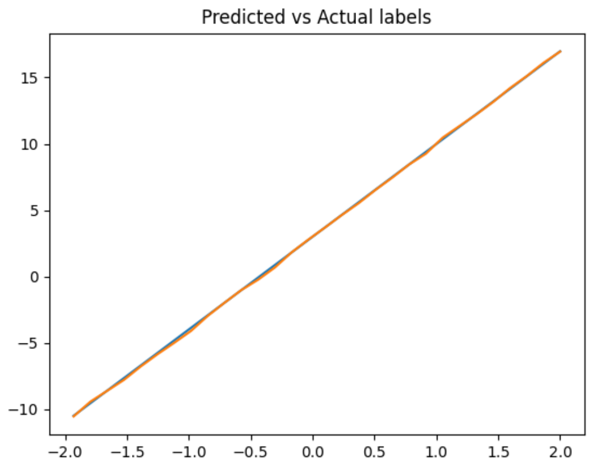
((for all the problems, weights have been initialized with random gaussian weights divided by a factor of the square root of the number of inputs))

1. Problem Statement:

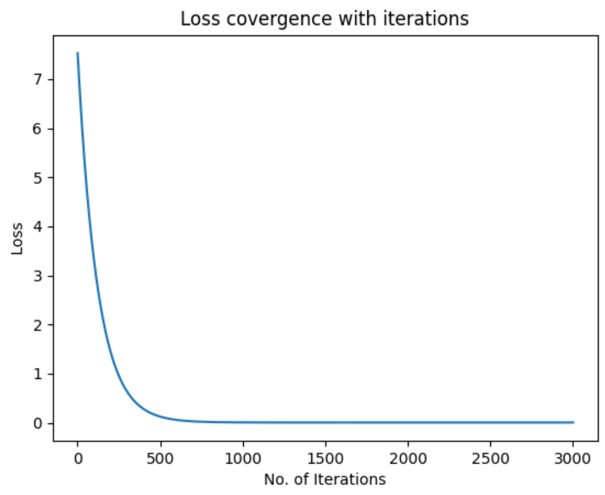
Network: 1 input layer, no hidden layer, and 1 output layer

1.a) 1D Data: the function, y=2 is used to generate the data set. Further, a gaussian noise with a variation of 0.2 is used to add noise for target points, that is y=7x + noise. Network: 1 neuron in each; input layer and output layer Activation Function Used: No Activation Function Used Loss function: Regression Loss.





1.b) Multidimensional Data: Considered einsum NumPy function to generate data with data points lying in linspace of -2 to 2. Network: Here n neurons in the input layer, no hidden layer, and 1 neuron in the output layer. The number of neurons used in the input layer is based on the generated input data or data layer. Activation Function Used: No Activation Function Used Loss function: Regression Loss.



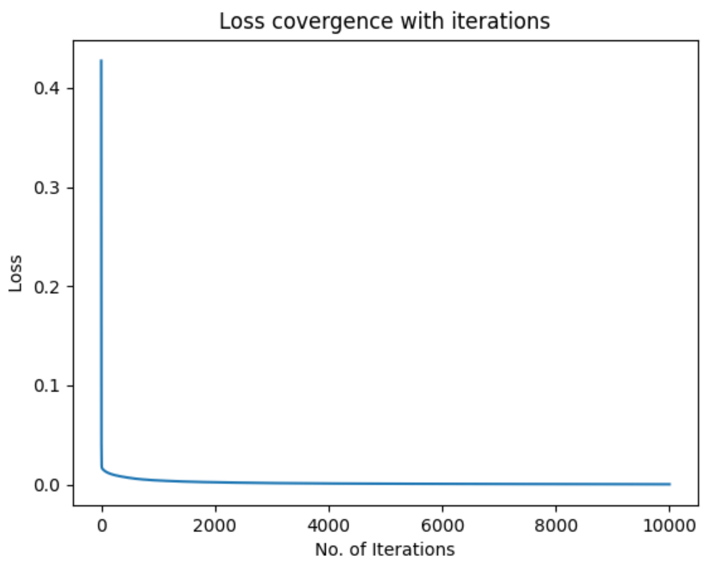
This was also tested with data_generators provided data which gave a loss of MSE = 0.00495205767667192.

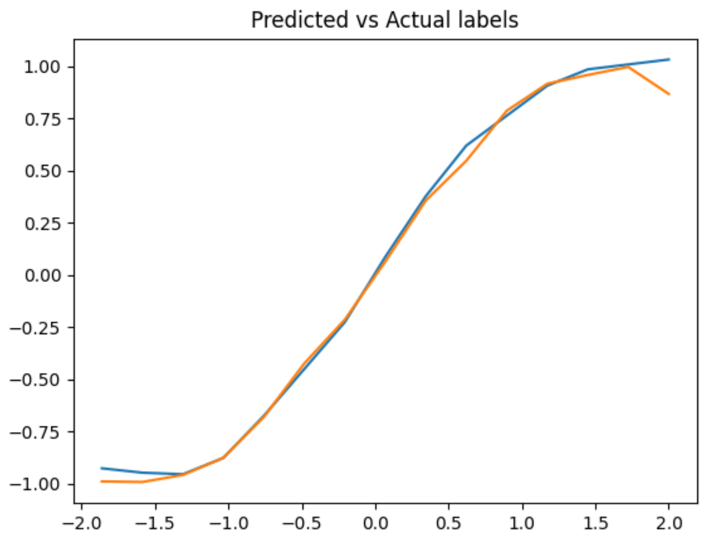
1. c) It was difficult to set the hyperparameters for the neural network despite it being simple without hidden layers. The difficulty arises due to various factors. From the plot, we know how the loss converges with several iterations, but further information is not given. There is as such no trend to follow while setting parameters except trying to tune various hyperparameters such as the learning rate which can be estimated from the above plot, in the sense, such a plot can answer whether the network is running or not. We can also tune the initialization of weights, and iterations too. There are other ways to gauge but being a simple network only the above steps were implemented.

2. Problem Statement 2: A Shallow Network

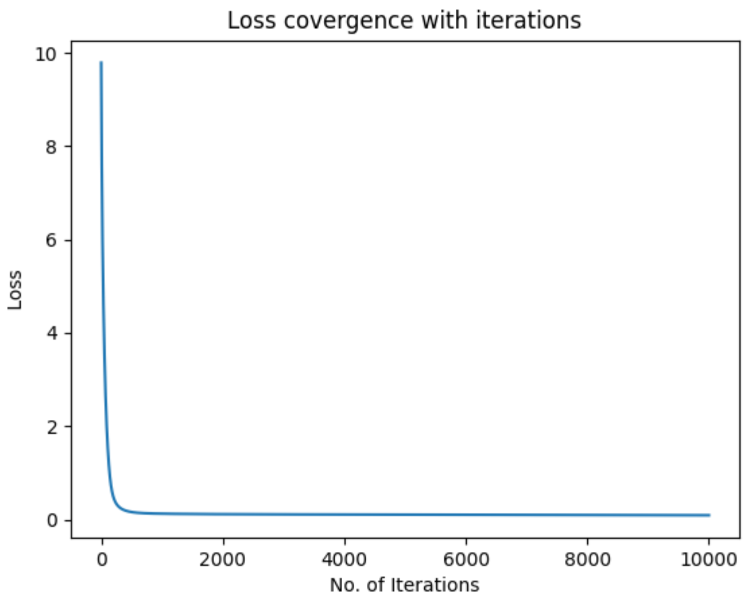
Network: 1 input layer, 1 hidden layer and 1 output layer

2.a) 1D Data: y=sin(x) is the function used to generate the dataset. Network: 1 neuron in each; input layer, hidden layer and output layer Activation Function Used: ReLU Loss function: Regression Loss.

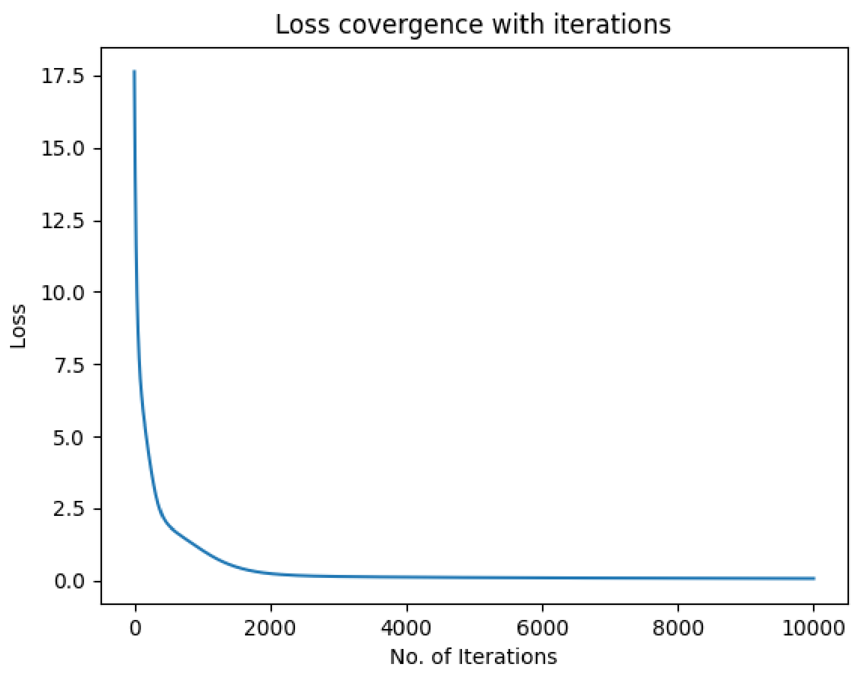




2.b) Multidimensional Data: Considered data similar to the data generators but then checked with 2 features, using seed() , function used is: y=0.1x1 + 0.1x2 + ... + 0.1*xi + 1. But ultimately ran with the data generators 2b function. Network: 2 neurons in input layer, 2 neurons in hidden layer and 1 neuron in output layer Activation Function Used: ReLU Loss function: Regression Loss.



The above diagram represents loss convergence at a high learning rate, which is 0.0058. Whereas the below plot uses a learning rate of 0.002 which gives the smooth curve of transition with a number of iterations. These help in choosing the hyperparameters.

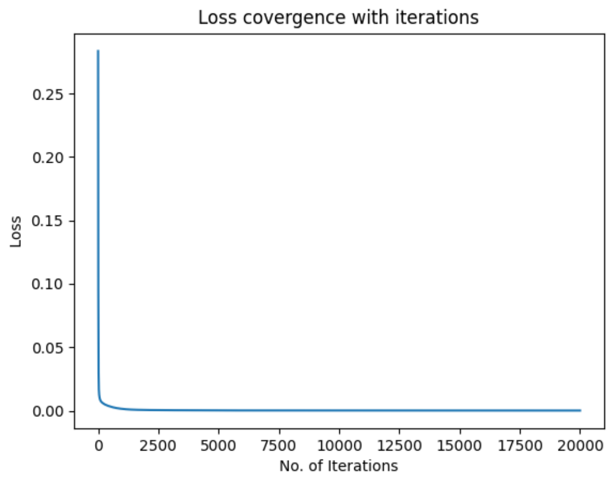


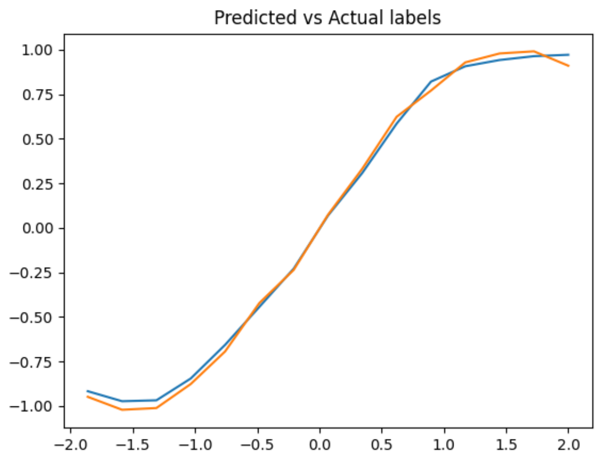
2.c) Setting hyperparameters for a higher dimension network was more difficult than for 1 Dimension data. It is easier to visualize 1D data but that's not the case with higher dimension data. It's more difficult to set parameters just by observing the loss vs iteration plot and MSE. Although it does tell us about the kind of initialization for weights that can be chosen or the learning rate.

3. Problem Statement

Network: 1 input layer, n hidden layers and 1 output layer

3.a) 1D Data: y=sin(x) is the function used to generate the dataset. Network: 1 neuron in each; input layer, all hidden layers, and output layer Activation Function Used: ReLU Loss function: Regression Loss

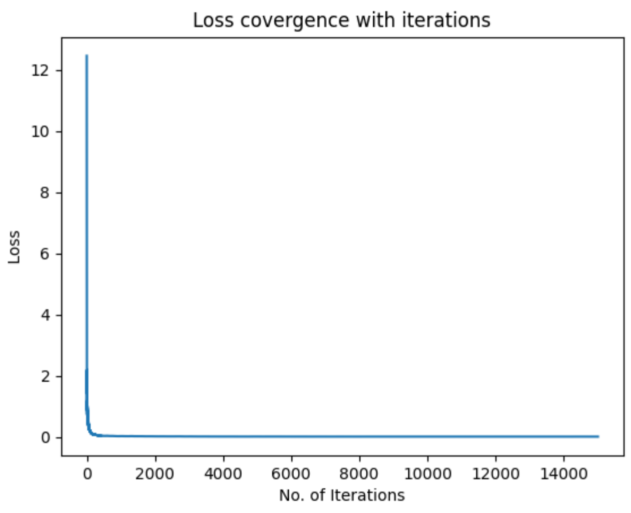




MSE = 0.0006310976613721724

By observing the plots and the mean squared error, we can see that the network works well.

3.b) Multidimensional Data: Considered: p features of x; x1 and x2, ..., xp. Using seed(), function used is: y=0.4*x1 + 0.4*x2 + ... + 0.4*xi + 1 is the function used to generate the dataset. But the most recent implementation is of the data_generator . Network: p neurons in input layer, 2 hidden layers, 12 neurons in each hidden layer and 1 neuron in output layer Activation Function Used: ReLU Loss function: Regression Loss



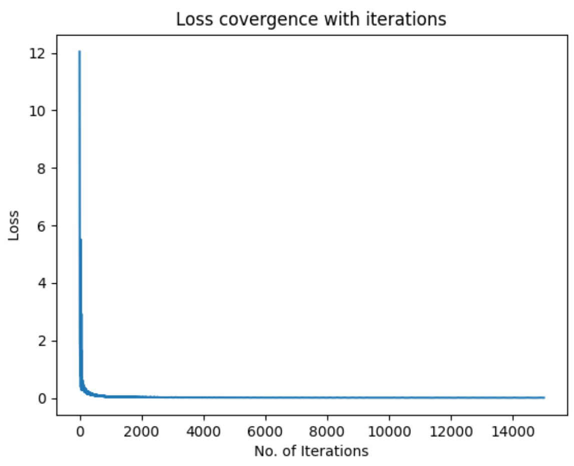
MSE = 0.006818407657524787

3.c) It is harder to choose hyperparameters for deeper networks, as the change of cost with respect to bias and weights changes recursively multiple times. In my case learning rate and the number of iterations caused loss to oscillate and sometimes it stayed high, but by changing the number of neurons, it helped in finding the hyperparameters which bought minimum.

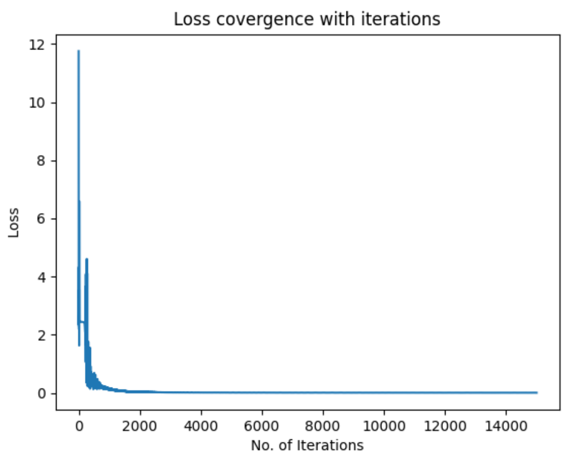
It is easier to tune hyperparameters for 1D problems as the gradient change with respect to bias and weights has to be analyzed for only a few neurons. Unlike multi-dimension problems, the complexity is smaller and analysis can be performed more efficiently.

Another problem is the MSE fluctuates sometimes during different runs even for the same hyperparameters which caused the public tests to pass and sometimes fail with the same parameters.

The above plots were for 2 hidden layers. Keeping the hyperparameters the same, except increasing hidden layers gives the following effect.



Number of layers = 3



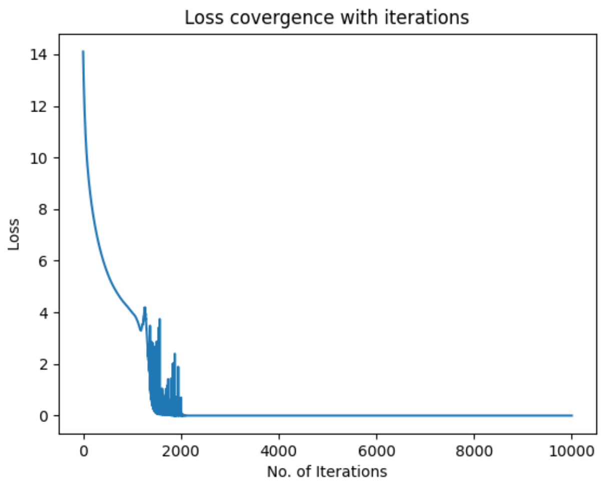
Number of layers = 9

4. Problem Statement

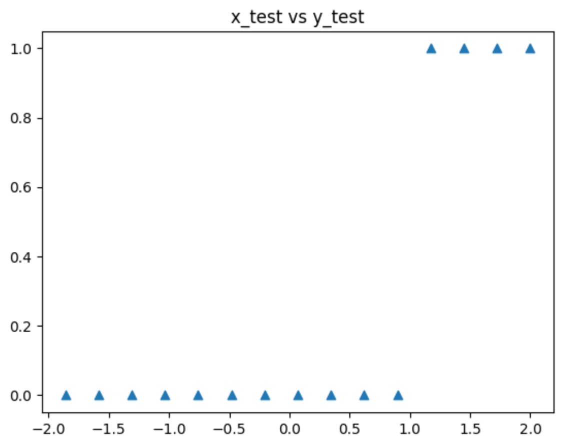
Network: 1 input layer, n hidden layers, and 1 output layer

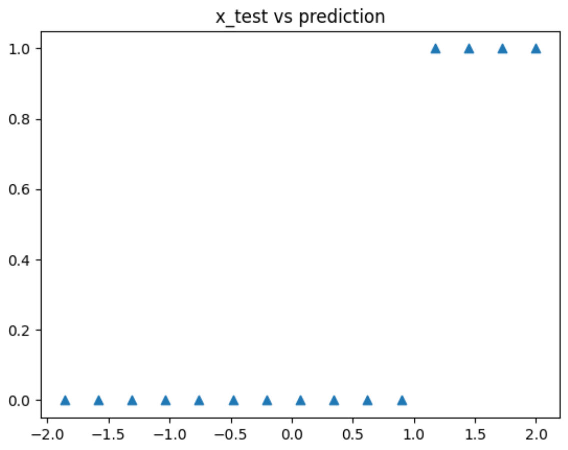
4.a) 1D Data: linspace is the function used to generate the dataset between two fixed points such that initial 6 points where classified to 0 and the rest were 1. But the final implementation has been done as per the data generators function to test compatibility with public tests. Network: input layer, all hidden layers, and Activation Function Used: ReLU for hidden layers and for the output layer: Sigmoid Activation, Loss function: Cross-Entropy Loss.

((hidden layers=2, hidden units = 11 in each)



On observation, it is seen from the plots below that the neural network classifies the scalar x quantities provided in the dataset with an error of less than 0.001





In the above data set, the data is linearly separable. For linearly non-separable dataset can be generated by randomizing indexes between the same two points on the same generated array as before and assigning 1 and 0 labels.

4. b) Multidimensional Data: mesh grid function has been used to generate the dataset between two fixed points such that the initial 6 points were classified to 0 and the rest were 1. But the final implementation has been done as per the data generators function to test compatibility with public tests. Network: 1 neuron input layer, all hidden layers with Activation Function Used: ReLU and for the output layer: Sigmoid Activation, Loss function: Cross-Entropy Loss.

5. Network: Datalayer as input layer, n hidden layers with ReLU, and output layer with activation function of sigmoid and loss function: SoftmaxCrossEntropy. Couldn't finish the last problem but completed the layers.py.