Title:

Assignment 0,

Deep Neural Network from scratch

Department of Computer and Data Sciences

Advisor:

Dr. Kheradpisheh

Student:

Fatemeh Saberi Khomami

Student Number:

400422114

1. Introduction

The main purpose of this exercise is implementing a dynamic deep neural network and compare its results in two tasks with a same network derived by Pytorch/Tensorflow.

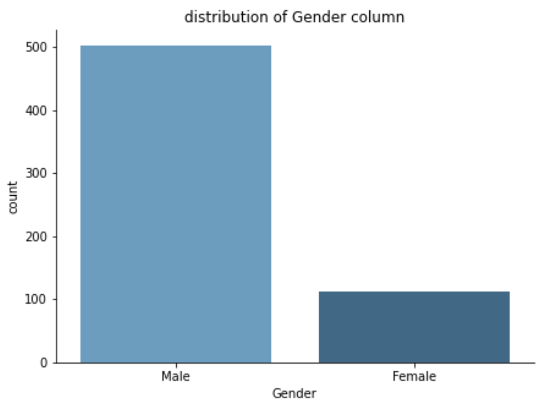
The tasks considered in this assignment are as follows: First, we should accomplish a classification algorithm on the loan-eligible dataset and estimate whether a costumer can get a loan or not. Second, we should perform a regression to predict LoanAmount values for the test dataset.

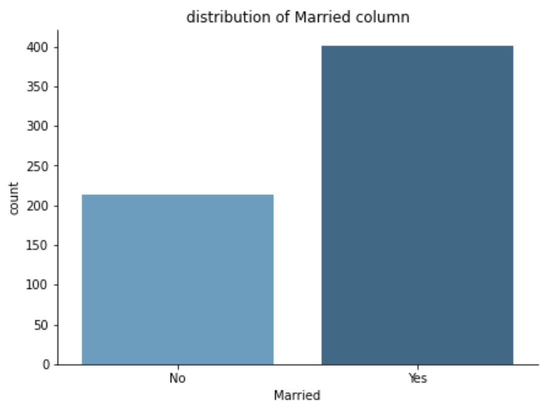
According to the hint, in the assignment, the proposed deep neural network should be implemented using inheritance which means we should first define a base layer class, and then implement Fully Connected layer and Activation Layer inherited form the base layer.

2. Getting to know the Data and Data Cleaning

We have 614 training samples and 367 data for the test set.

First, we dropped the Loan_ID columns from both datasets. Regarding that the amount of the training data is not sufficient, we filled the missing values by the mode and average based on the type of the variable. In below figures, you can see the distribution of the categorical variables in the training dataset.





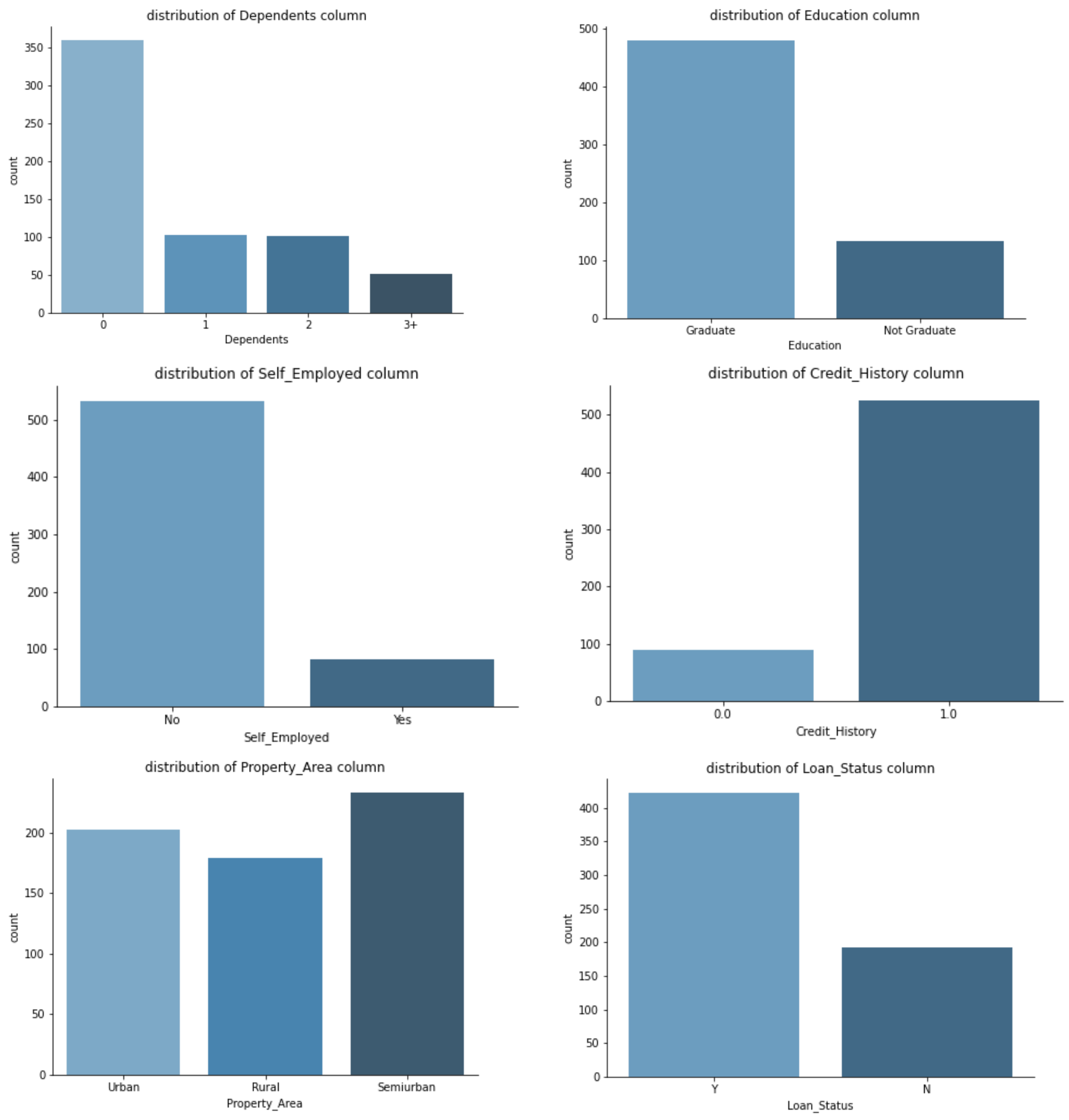


Figure (1): Distribution of categorical variables in the training dataset

Then we replaced these categorical variables with their corresponding one hot columns.

At last, we normalized the numeric columns by dividing them by their maximum values.

Notice that the test set does not contain Loan_Status column, so we can not use it for model evaluation in the classification task.

3. Implementing the deep neural network

3.1. Base layer

In this part, we start to develop a neural network with the aid of the hint. The base layer is a class containing input and output variables, forward_prop and backward_prop functions. In fact, this layer will apply forward propagation on its input and backward propagation on its output, depending on whether the layer is the fully connected layer or the activation layer.

3.2. Fully-connected layer

The fully-connected layer (FCLayer) is a class that inherits from the Layer class. At first, this layer considers random values for its weight matrix and bias vector. These matrices' dimensions are based on the output and input of the layer. In the forward_prop function, the FCLayer gets the input from the previous layer (or input layer) and then computes "Wa+b".

For the back_prop, this layer computes gradients of its weights and biases according to the error of its next layer, and then updates their values with gradient descent. Finally, the layer calculates the da for the next layer in the backward procedure. For instance, in Figure (1), FCLayer1 gets input X and returns WX+b to ActivationLayer1. Moreover, FCLayer2 gets dz2 (output of ActivationLayer2 back_prop function), updates its weights, and pass W^T.dz2 to ActivationLayer1.

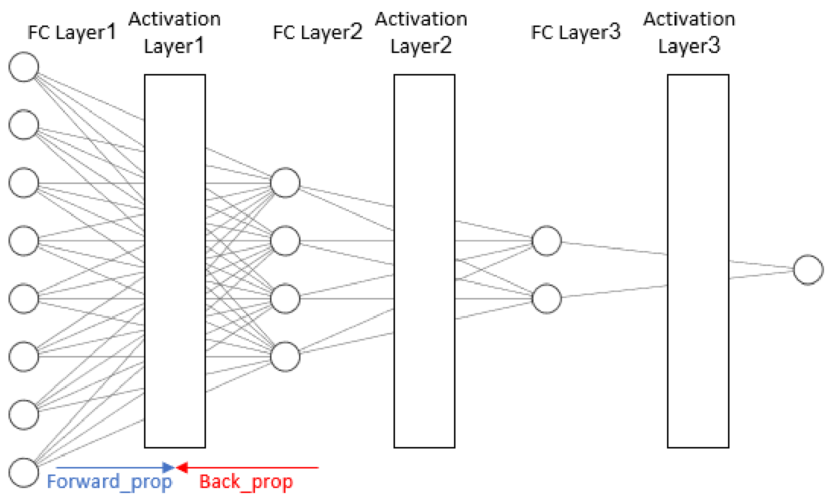


Figure (2): An example of a neural network and its layers' objects

3.3. Activation layer

This class also inherits from the Layer class. Its forward_prop function gets an input, applies the corresponding activation function (Tanh or Sigmoid) on it, and returns the result to the next layer. In the back_prop function, this class gets da and compute da$\oplus$g'(z) and pass it to the previous layer in the network.

3.4. Train function

This function performs the training of the network. It takes a network with a specific sequence of FCLayers and ActivationLayers, loss function and its derivative, training data and their corresponding labels, number of epochs, and the learning rate as its inputs. Firstly, the function applies forward procedure to the entire network with the training inputs and calculates the loss for different samples. Forward_prop functions are depended on the layer type of the corresponding layer. Then, the gradient of the loss function is considered as the input of the back propagation procedure and updates the weights' values of each FCLayer. The train function repeats this procedure at each epoch. Finally, the network is trained and the final values of weight matrices are obtained.

3.5. Test function

The test function takes the trained network and performs the forward procedure to the test data, and returns the network's output.

3.6. Activation and loss functions

We define Tanh and Sigmoid functions as the activation functions, and mean square error and cross entropy functions as losses. We choose one of the loss functions as the network loss function and one of the activation functions for each ActivationLayer.

4. Training and evaluation of the constructed model

Now, we design a deep neural network to estimate Loan_status and predict Loan_Amount.

4.1. Classification

We used the below architecture for training on the loan data.

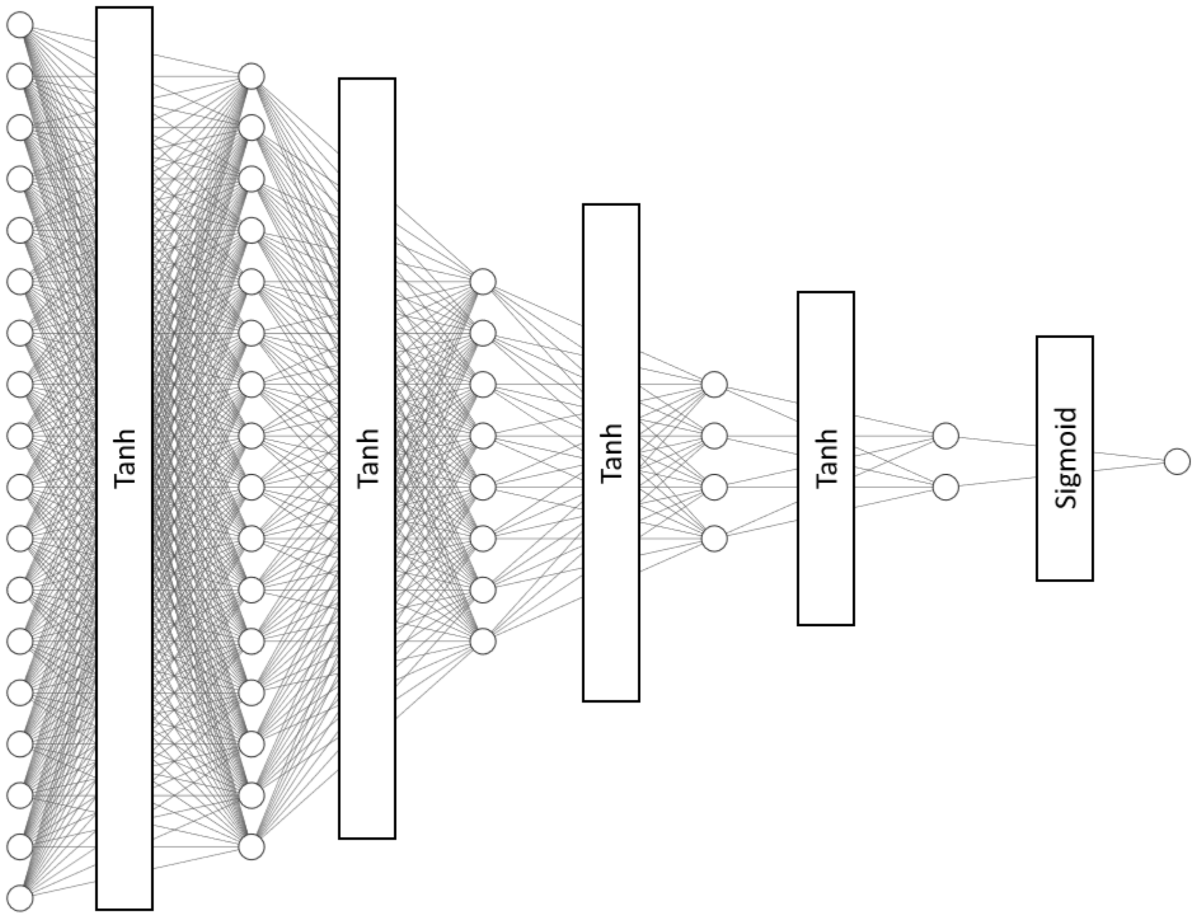


Figure (3): Architecture of the developed neural network for classification

We used a 5-layer structure with, 16, 8, 4, 2, 1 neuron in each layer respectively. The training was 100 epochs long using Gradient Descent as the optimizer with learning rate 0.001.

The result of the last epoch is as follows:

error=0.48331863785347084

We manually saved 10% of the training set for testing. The performance of the above model on the unseen data is 83.6%.

4.2. Regression

We used the same architecture for the regression with one difference. The last layer doesn't have an activation, and it's the linear product of the previous layer.

The result of the last epoch is as follows:

error=0.015788464157109196

We manually saved 10% of the training set for testing. The performance of the above model on the unseen data is 0.014516528719760741.

5. Training with the same architecture by TensorFlow

5.1. Classification

The result of the last epoch of TensorFlow model for the classification task is as follows:

loss: 0.5137 - binary_accuracy: 0.7754 - val_loss: 0.5159 - val_binary_accuracy: 0.7903

Ass you can see, our model performed better!!

You can see the performance of the model in the below figure.

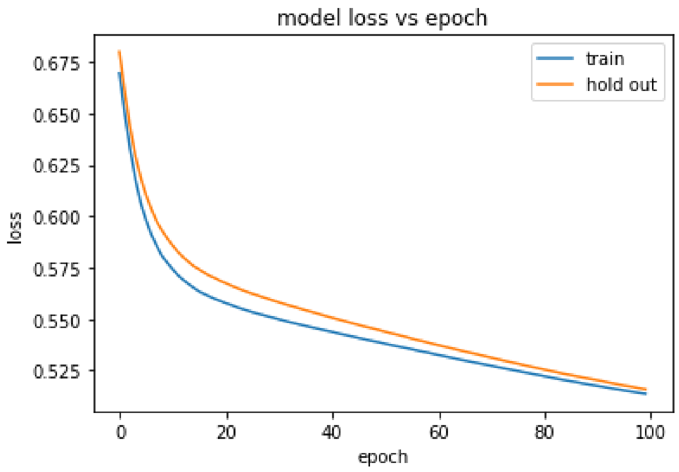


Figure (4): Loss of the TensorFlow model on the training and validation dataset

5.2. Regression

The result of the last epoch of TensorFlow model for the regression task is as follows:

loss: 0.0114 - val_loss: 0.0175

Loss of the model on the test set is: 0.011589603498578072

As you can see, this model's efficiency is similar to our network.

You can see the performance of the model in the below figure.

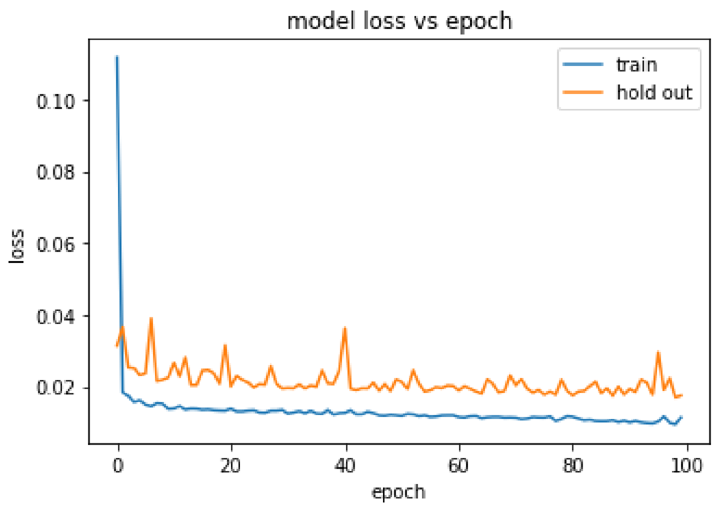


Figure (5): Loss of the TensorFlow model on the training and validation dataset