Assignment -1
Pima Indians Diabetes Classification using Logistic Regression and Neural Network

Nikhil Bandari 10 October 2021

1 Assignment Overview

The goal of the assignment is to work with logistic regression and Neural network Classifying whether a patient has diabetes (class 1) or not (class 0), based on the diagnostic measurements provided in the dataset. In the first part of the assignment, I have implemented logistic regression using gradient descent. In the second part I have implemented Neural network using Tensor Flow for the same dataset.

2 Dataset

For implementing Logistic regression and Neural models with diabetes dataset of 768 samples and 8 features. I have split data samples comprises of training, validation and testing data, each constituting 60%,20% and 20% of overall data. Training dataset has 460 samples, Validation has 154 samples whereas testing dataset has 154 samples. The diagnostic measurements of 8 features are:

```
norm=MinMaxScaler()
features=norm.fit_transform(features)
# Splitting  dataframes into 80% into training , 20 % testing and 20% Validation
x_train, x_test, y_train, y_test = train_test_split(features, outcome, test_size = 0.4, random_state = 0)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size = 0.5, random_state = 0)
```

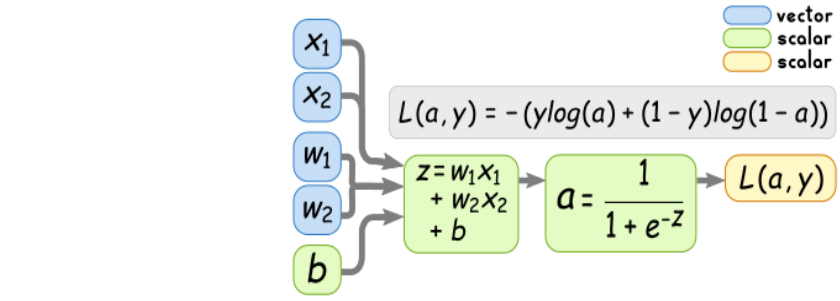
1 Glucose (Blood Glucose level)
2 Pregnancies (The number of pregnancies the patient has had)
3 Blood Pressure (mm Hg)
4 Skin Thickness (Triceps skin fold thickness (mm))
5 Insulin level
6 BMI (Body Mass Index: weight in kg/(height in m)2)
7 Diabetes Pedigree Function
8 Age (In years)

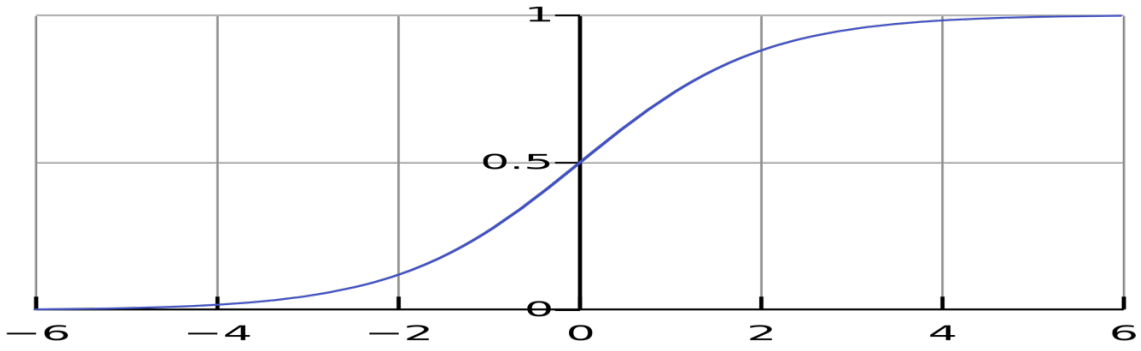
3 Python Editor

I have used Google colab IDE for implementation and Comparing results.

4 **Part 1: Logistic Regression: Architecture**

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, **logistic regression**[1] (or **logit regression**) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".






```
def sigmoid(z):
    return  1 / (1 + np.exp(-z))
```

4.1 Loss and Cost function(Gradient Descent)

Loss function is the loss for a training example Cost is the loss for whole training set.P is our prediction and y is correct value. Sigmoid function to produce value between 0 and 1. Updating weights and biases, p is our prediction and y is correct value. We can change the speed at which we reach the optimal minimum by adjusting the learning rate. Finding db and dw & Derivative wrt p->Derivative wrt z.

$$b := b - \alpha db$$
$$w := w - \alpha dw$$

After Knowing B and w, From the code, you find p(Prediction). It will be between 0 and 1.

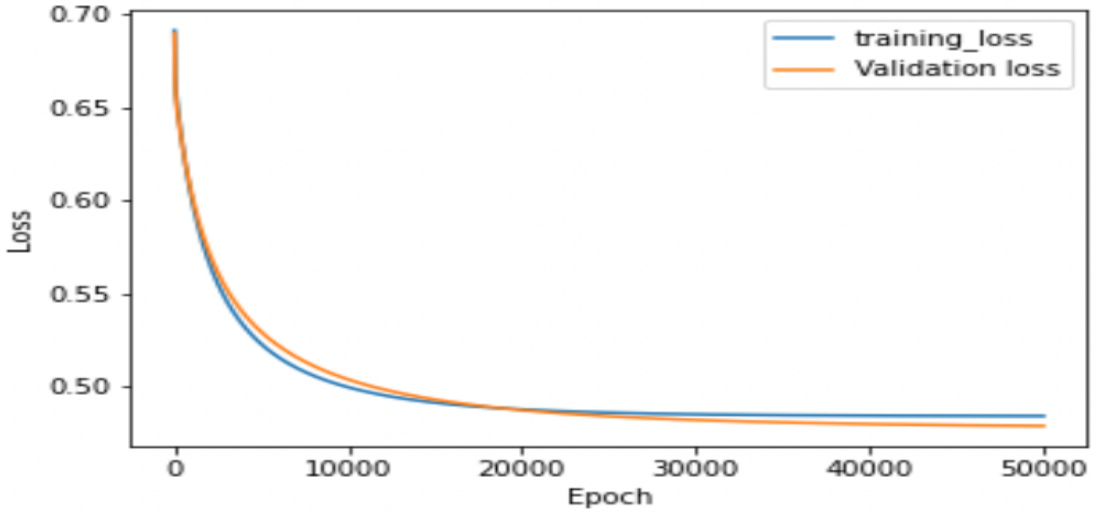
4.2 Tuning the hyperparameters and training the model:

To tune Model, following hyper-parameters (epochs, learning rate α) values were used Epochs =50,000 and α = 0.05.

Using Gradient Descent Approach, an optimization algorithm approach used to find the weights and bias and when it reaches the minimum value, by fixing the values of weight we can test the model using different sets.

4.3 Graphs

Below figure shows Training and Validation Loss for given epochs and Loss.



4.4 Test your logistic regression model on the testing set:
After finishing the above steps, tested the model's performance on the testing and Validation set. This shows the effectiveness of your model's generalization power gained by learning.

With the above parameters, 76% and 79% accuracy were found on validation and testing sets.

```
LR Accuracy of test set%:  79.87012987012987
LR Accuracy of validation set%:  77.92207792207793
```

5. Part 2: Implement Neural Networks

A Neural Network consists of an input later some hidden layer and an output layer. Each layer consists of a neuron which compute the result of a linear equation (figure 4) and is followed by an activation function. The input enters from the input layers and goes through a hidden layers and arrives at the output layers as predictions.

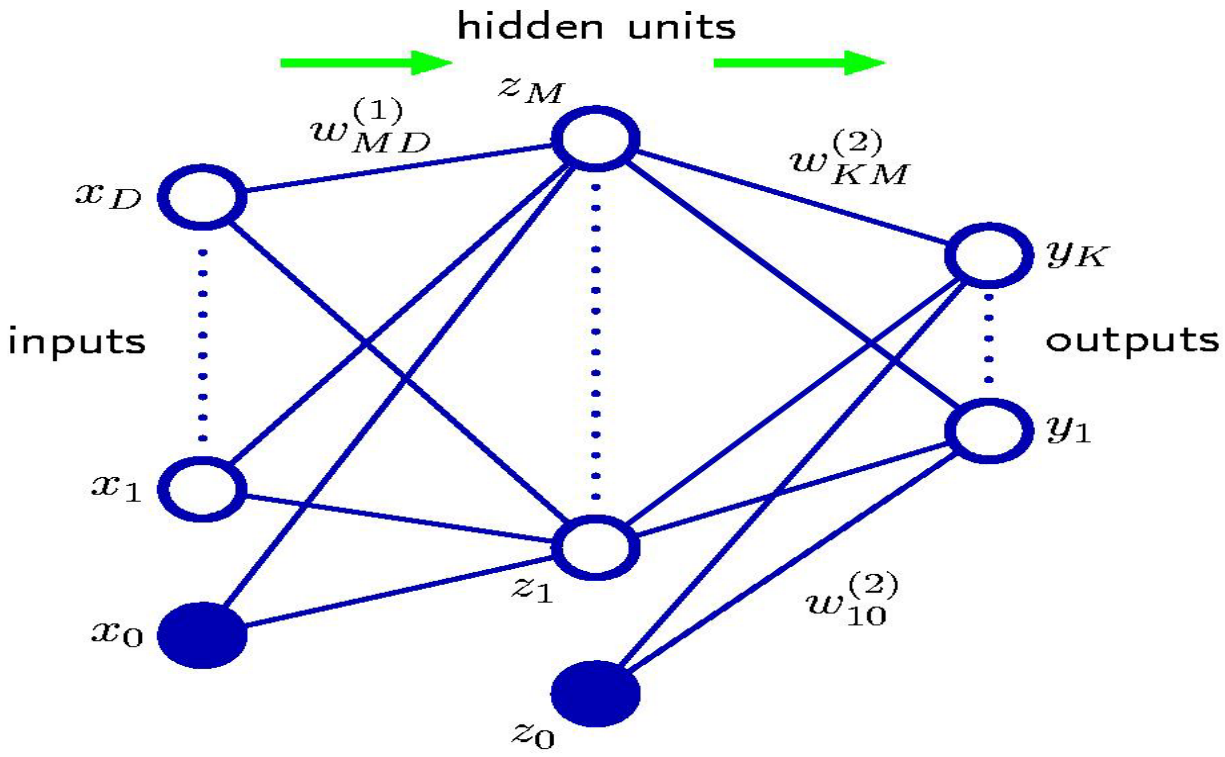


Figure : Typical Neural network

5.1 **Classification of Neural Network:**

By using Tensor Flow + Keras Machine Learning Model, we can classify the given Set as :

1. Define the model.
2. Compile the model.
3. Fit the model.
4. Evaluate the model.
5. Make predictions.

5.2 Define the Model

Neural Network consists of different layers in the architecture and adding weights to each layer will be input to another layer.

- The model expects rows of data with 8 variables (the input_dim=8 argument)
- The first hidden layer has 8 nodes and uses the relu activation function.
- The second hidden layer has 4 nodes and uses the relu activation function.
- The output layer has one node and uses the sigmoid activation function.

```
] # create model, add dense layers one by one specifying activation function
  #Create the model using the Sequential API
  model = Sequential()
  model.add(Dense(4, kernel_regularizer=l2(0.001),activation='relu',input_dim=8)) # Hidden layer 1 with L2 regularization.
  model.add(Dense(8, activation='relu')) # Hidden Layer 2
  model.add(Dense(1, activation='sigmoid')) #Since there is one Class between 0 and 1, we can use sigmoid Function
```

5.3 Compile Model:

In this case, we will use cross entropy as the **loss** argument. This loss is for a binary classification problems and is defined in Keras as "**binary_crossentropy**". We will define the optimizer as the efficient stochastic gradient descent ". This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.

```
model.compile(loss='binary_crossentropy', optimizer=SGD(), metrics=['accuracy'])
model.summary()
```

5.4 Fit the Model :

We can train or fit our model on our loaded data by calling the **fit()** function on the model. Training occurs over epochs and each epoch is split into batches.

- **Epoch**: One pass through all of the rows in the training dataset.- 200
- **Batch**: One or more samples considered by the model within an epoch before weights are updated. -10

```
history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=200,verbose=0, batch_size=10)
```

5.5 Evaluate the Model :

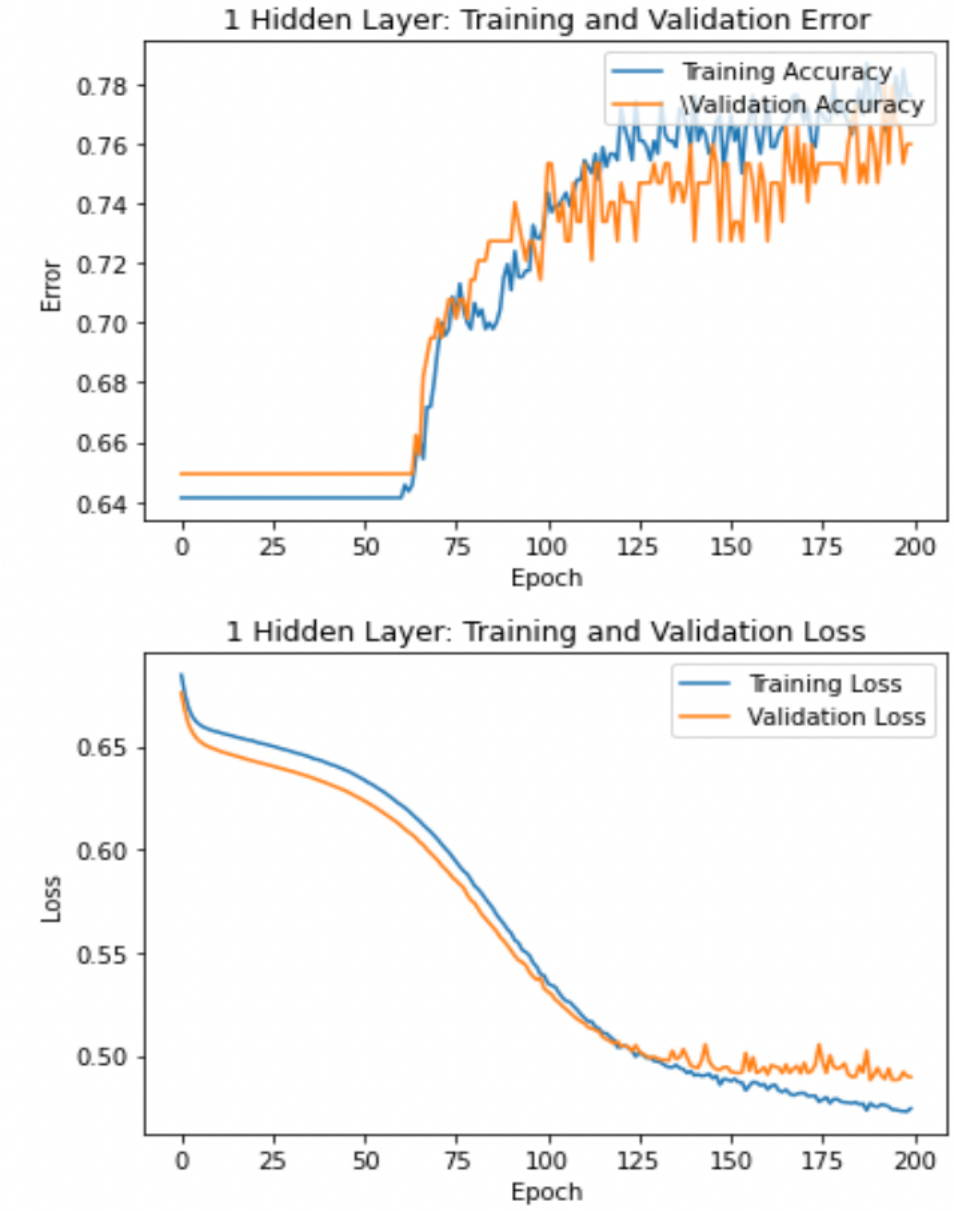
This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.

The **evaluate()** function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset. We are only interested in reporting the accuracy, so we will ignore the loss value.

```
# Evaluating Model with testing data set.
loss, accuracy = model.evaluate(x_test, y_test)
training_loss, training_accuracy = model.evaluate(x_train, y_train)
Val_loss, val_accuracy = model.evaluate(x_val, y_val)
```

5.6 Test your Neural Network model on the testing set:
After fixing the Hyperparameters with above mentioned values, Tested my model with test parameters.
With the above parameters, 78- 80% accuracy were found on testing sets with L1 Regularization.

5.7 Graphs Observed between Validation and Training loss and Accuracy .



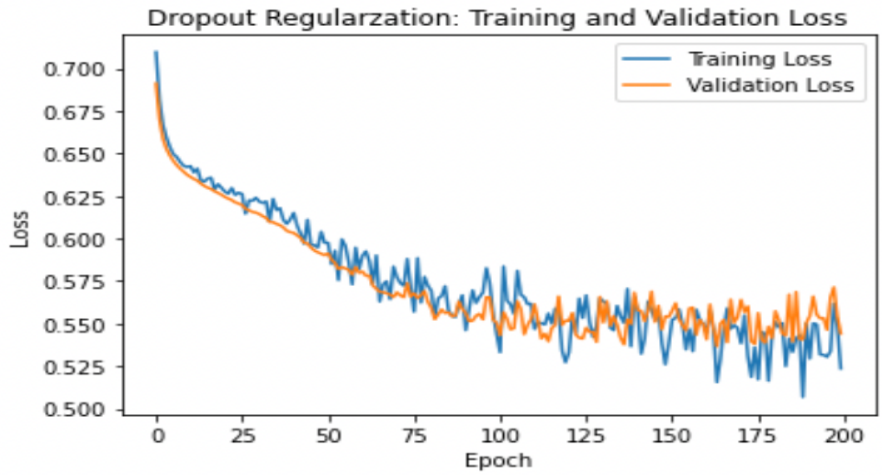
6 Part 3- Implement different regularization methods for the Neural Networks

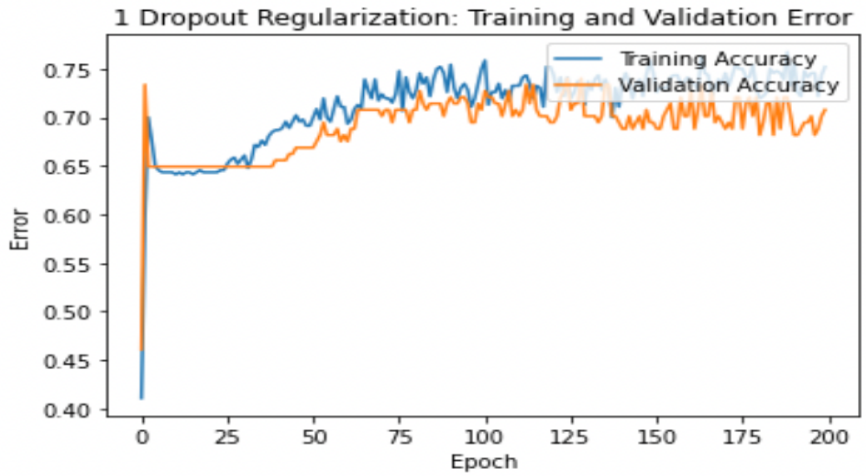
With Model implemented in Part 2, By Fixing all Neurons and Layers with same epoch 200 and batch size of 10, L2 regularization is removed. Dropout is added to the model and tested on test set.

```
 Drop Loss 0.47775799036026
DropOut Testing Accuracy: 77.92207598686218
```

6.1 Graphs:

With Dropout added to the model, the accuracy has been nearly equal or decreasing.
Below figures can be compared to with L2 regularization.





Results:

- Since the validation loss was lower than the training loss, it can infer that the

model is neither overfitting and not underfitting in both logistic regression and neural network models.

- Accuracy was
- 77-79 % in Logistic regression
- 78-80 % in L2 Regularization -Neural Network
- 77.92 % in Dropout Regularization – Neural network

Hence the derived model can be used for any data set to predict whether Patient has diabetes or not.

References:

1. https://towardsdatascience.com/logistic-regression-fromvery-scratch-ea914961f320
2. https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/
3. Professor and TA Notes.