Kshitij Chhajed

10th October 2021

1. **Assignment details:**
   Train using Neural notworks: Build a Neural Network with 1,2 or 3 hidden layers with different regularization methods (l2, l1), that takes the features as input and gives as output whether a person has diabetes or not. Compare the Regularization methods and explain it in your report.

2. **Dataset:**
   In order to implement logistic regression, dataset with data of 768 diabetic and non-diabetic patients was used. I have split the dataset manually and used 60% data for training, 20% for validation and 20% for testing. The dataset contains medical data of female patients above the age of 21 consisting of below 8 features.
   a. **Glucose (Blood Glucose level)**
   b. **Pregnancies (The number of pregnancies the patient has had)**
   c. **Blood Pressure(mm Hg)**
   d. **Skin Thickness(Triceps skin fold thickness (mm))**
   e. **Insulin level**
   f. **BMI (Body Mass Index : weight in kg/(height in m)2)**
   g. **Diabetes Pedigree Function**
   h. **Age (In years)**

3. **Languages/tools used:**
   I have used Python as a language and VScode as the editor.

4. **Methodology and Results:**
   a. **Extracting and pre-processing data:**
      The scaled data from logistic regression is used and split into:
      x_train – Data used for training the network
      x_val – Data used for validating the trained network
      x_test – Data used for testing and prediction
      y_train – Output data corresponding to x_train
      y_val – Output data corresponding to x_val
      y_test – Output data corresponding to x_test
      Since neural network methods used calculate bias automatically, extra column of ones which was added in logistic regression is not added here.

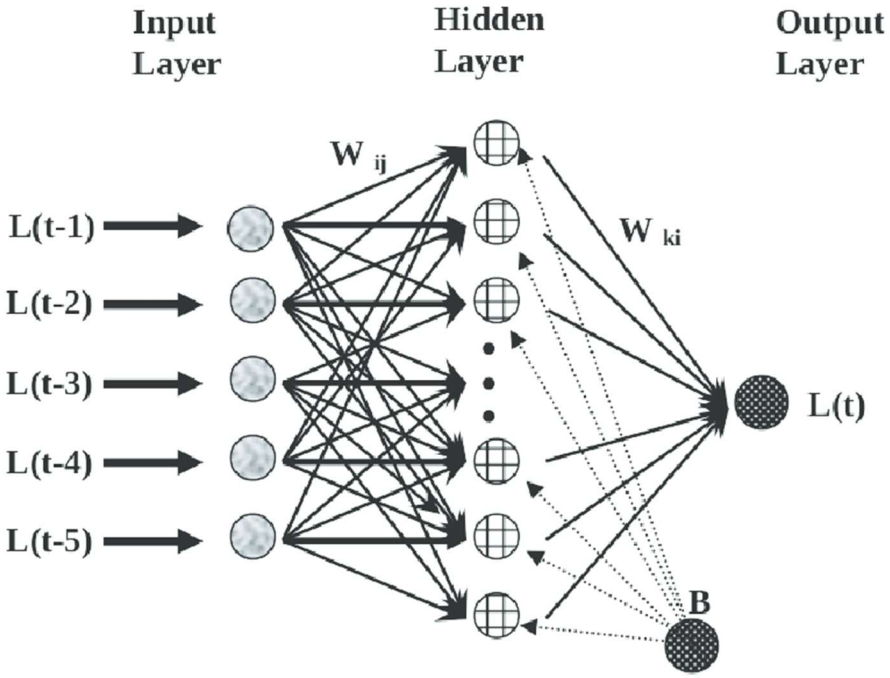
      For implementation of neural network, I have used tensorflow and keras library. I have selected sequential() model for this project. After creating an object for the sequential model, add() method was used to add multiple layers to the network. Some parameters can explained as below:
      Dense – denotes that the network is fully connected. All nodes from previous layer feed output to next layer
      Activation = <Activation_function_name> - The activation function to be used for the layer can be defined over here. Different layers can be assigned different activation functions.
      kernel_regularizer=tf.keras.regularizers.<regularizer_name>(<value_for_Regularizer>) – Different regularizers like L2, L1, L1_L2 can be defined using this method

      Different flags have been defined to choose different regularization methods.
      I have used SGD() i.e. Stochastic Gradient descent to reduce the cost/loss function. The parameters like alpha (learning rate) and decay rate have been passed to it. The values of such hyper parameters and epochs was selected after checking output at different values so that we can design the best-fit model.

Accuracies for the 3 different dataset parts was printed. Graphs were displayed to compare the rate of change of loss function and accuracy with respect to the no. of epochs.

b. Neural networks:

Neural networks or Artificial Neural Networks (ANNs) as they are known; are a network of computing units which are inspired from biological neural networks. The network is made up of multiple layers. First layer is called the input layer and last layer is called the output layer. All the layers in between input and output are called hidden layers. A layer can have multiple neurons (shown by circles in below image). The neurons are fed with input from previous layer along with the weights associated with them. Weights determine the importance of an input (feature) in predicting the output of the model. The layers also have an activation function which serves as a test whether the output of the neuron/node would be passed to the next layer or not. In other words, it decides whether the output of the node should be fired or not to the next layer. In our case, we have used relu() activation function which takes in the input (combination of weights and inputs) from previous layer and performs operation over it. If the outcome comes out to be negative, the neuron is disabled otherwise its output is fed forward. At the output layer, we have used sigmoid() activation function since it gives output between 0 and 1. Since we have to classify output in 2 classes (binary) this function is suitable for it.



Different regularization techniques like L1, L2 are used to add additional penalty to the error function so that the model generalizes better. Overuse of such techniques might cause the model to oscillate between converging and diverging. Hence, optimum values of lambda for these techniques helps to build a better model.

c. Validation and Testing:

The model was trained on initial 60% of data and the accuracy of the model was tested for both the validation and training parts of the dataset. Below were some of the results from a recent run:

Epochs = 200, learning rate (Stochastic gradient descent) = 0.08, decay (Stochastic gradient descent) = 0.008

Layer 1 = 25 Neurons, Activation function used = relu()

Layer 2 = 15 Neurons, Activation function used = relu()

Layer 3 = 10 Neurons, Activation function used = relu()

Layer 4 = 1 Neuron, Activation function used = sigmoid()

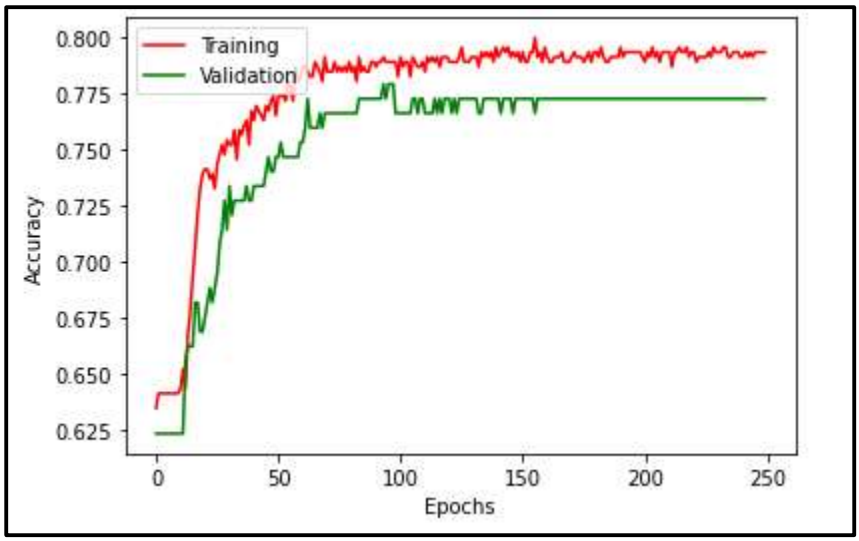
- **Neural networks without regularizer:**

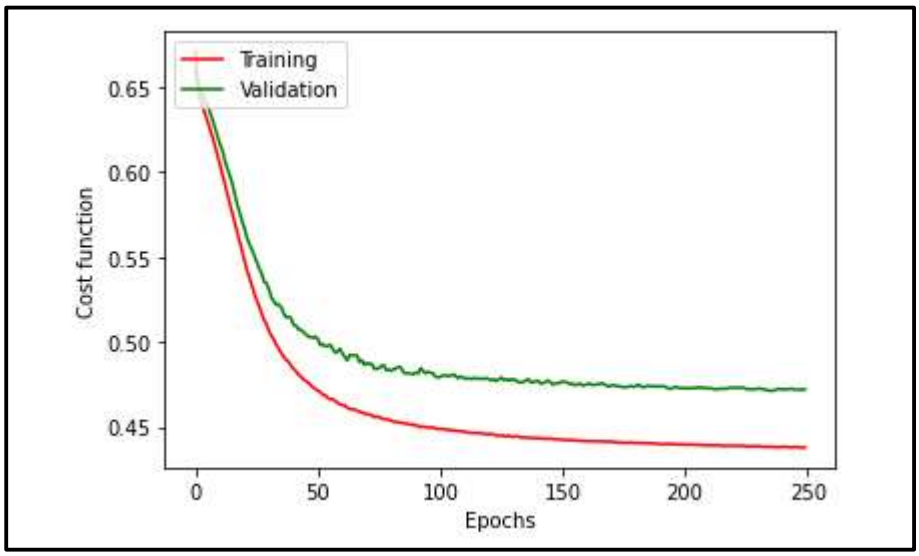
Set flag =1 to view results

```
Train data Loss: 0.4374596178531647    Train data Accuracy: 0.79347825050354

Validation data Loss: 0.47215351462364197      Validation data Accuracy: 0.7727272510528564

Test data Loss: 0.547768235206604      Test data Accuracy: 0.7337662577629089
```



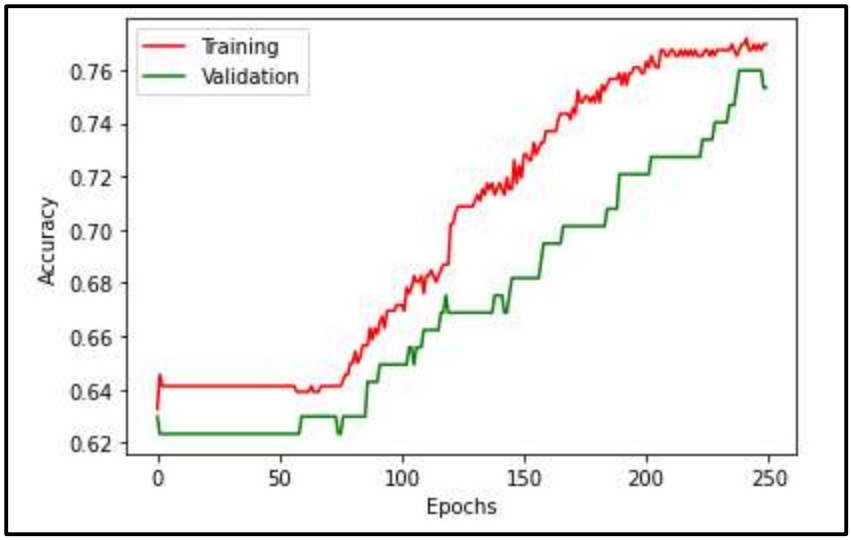


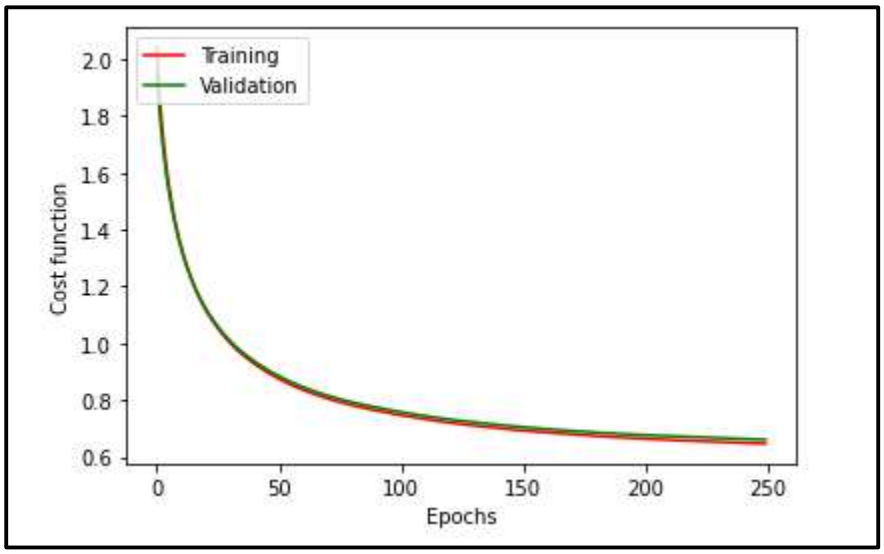
- **<u>Neural network with regularizer L1 (lambda = 0.01):</u>**
  All other hyperparameters remain same
  Set flag = 2 to view results

```
Train data Loss: 0.6464912295341492    Train data Accuracy: 0.769565224647522

Validation data Loss: 0.660682737827301        Validation data Accuracy: 0.7532467246055603

Test data Loss: 0.6586206555366516     Test data Accuracy: 0.7337662577629089
```





5. **Credits:**
   a. https://keras.io/api/layers/regularizers/
   b. **Google images**

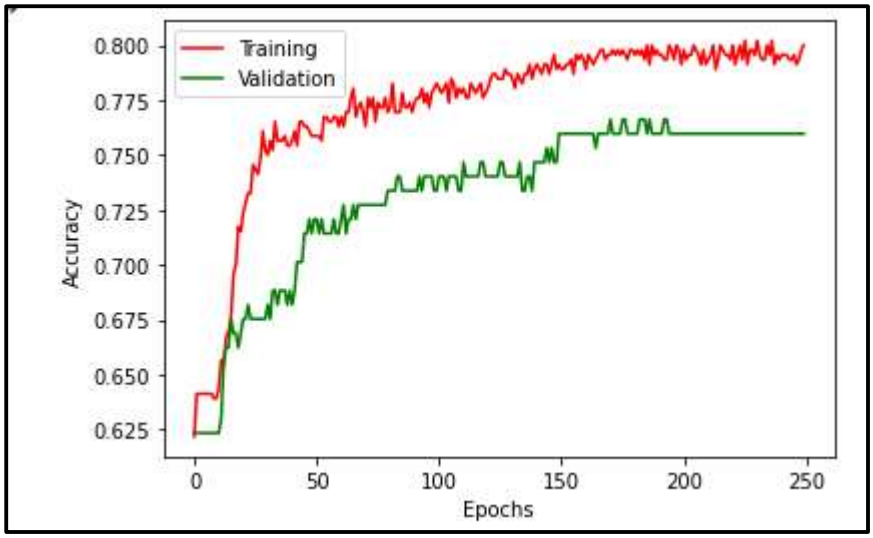
Comparison between different regularization techniques

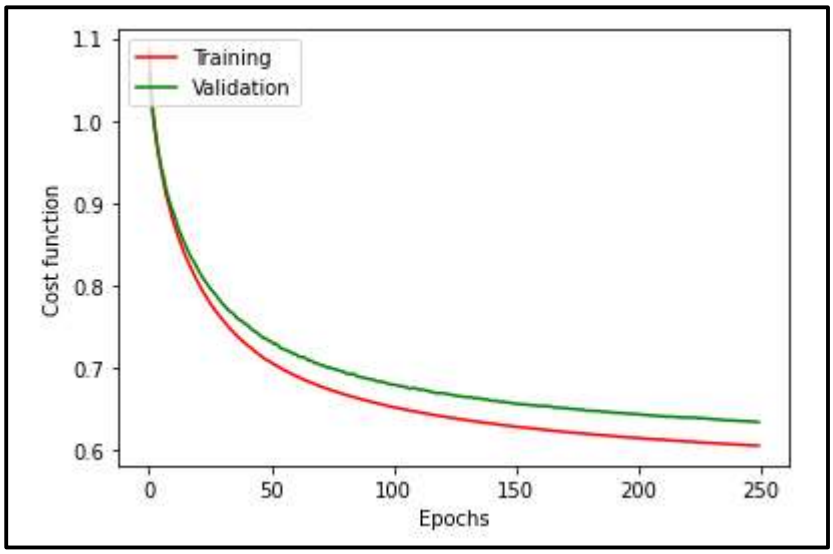
- **<u>Neural networks with L2 regularizer (Lambda = 0.01):</u>**
  All other parameters have been kept same
  Set flag =3 to view results

```
Train data Loss: 0.605335533618927     Train data Accuracy: 0.791304349899292

Validation data Loss: 0.6342328786849976       Validation data Accuracy: 0.7597402334213257

Test data Loss: 0.6733987331390381     Test data Accuracy: 0.7207792401313782
```



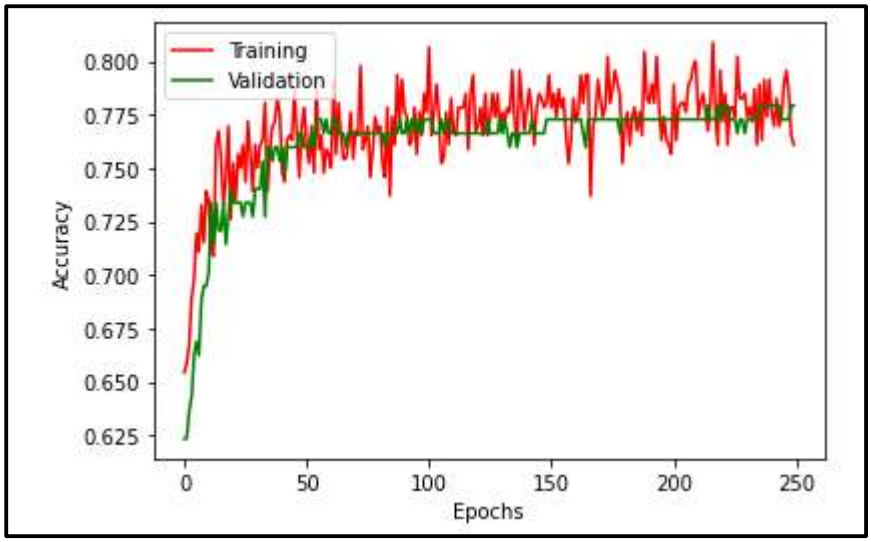


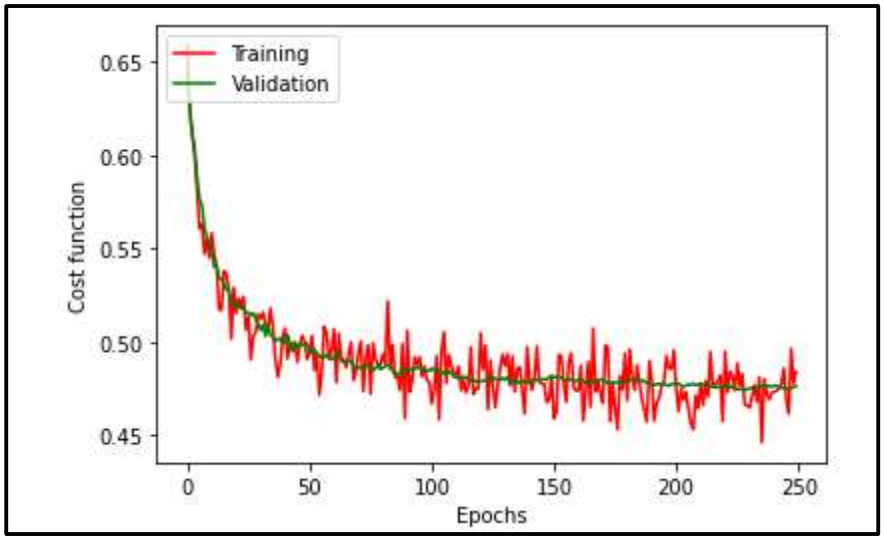
- **<u>Neural networks with Dropout regularizer (0.02):</u>**
  All other parameters have been kept same
  Set flag = 4 to view results

```
Train data Loss: 0.43931737542152405   Train data Accuracy: 0.7978261113166809

Validation data Loss: 0.47643423080444336      Validation data Accuracy: 0.7792207598686218

Test data Loss: 0.5387464165687561     Test data Accuracy: 0.7272727489471436
```





After checking above results, we can see that Dropout method of regularization gives slightly better accuracy. However, the fluctuations in accuracy and loss function in it are more than L2 regularization.