Assignment 1 - Logistic Regression and Neural Networks

Baasit Sharief

October 11, 2021

1 Data-set and Understanding the problem statement

1.1 Introduction

The goal of the first part of the assignment is to understand how a logistic regression model is created from scratch to gain a better understanding of the intricacies of the algorithm. With the help of python libraries like NumPy and Pandas along with JupyTer Notebook, I have implemented a logistic classifier that works on gradient descent for optimization.

The second part of the assignment is based on building a neural network model with either L1 or L2 regularisation to solve the given problem. I have made use of tensorflow and keras to build the model and used several optimizers such as SGD, RMSProp, Adadelta and Adam along with different activations for each layer i.e. Sigmoid and ReLU.

1.2 Data Loading and Pre-processing

The Dataset consists of 768 instances of 8 features each as described below of several patients and their Diabetes diagnosis.

	Feature Name
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)

I have used pandas library to read the csv files, and then used sklearn library to split data into training, validation and testing set with a ratio of 60:20:20. I used a custom defined a utility function to normalize the data between 0-1 using standardization technique.

$$\mathrm{X} = \frac{x - mu}{\sigma}$$

2 Logistic Regression

2.1 Designing the Logistic Regression Model

Logistic regression model is based upon Bernoulli distribution with a sigmoid activation and is used as a base to calculate probabilities for a 2-class classification problem which can be extended to multiclass using a softmax calculation.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Designing a logistic regression requires a cross-entropy loss function, an optimization technique, in this case, Gradient Descent.

$$\text{cost} = -(y \log(p) + (1 - y) \log(1 - p))$$

$$w = w - lr * dw$$

2.2 Hyperparameter Tuning - Experimental Setup

For hyper-parameter tuning, I performed a grid search on 4 different learning rates and 4 different number of epochs.

Below are the observed results on the validation dataset.

Learning Rate	100 epochs	500 epochs	1000 epochs	5000 epochs
0.1	0.78	0.62	0.51	0.51
0.01	0.79	0.75	0.53	0.51
0.001	0.79	0.78	0.62	0.51
0.0001	0.79	0.79	0.75	0.53

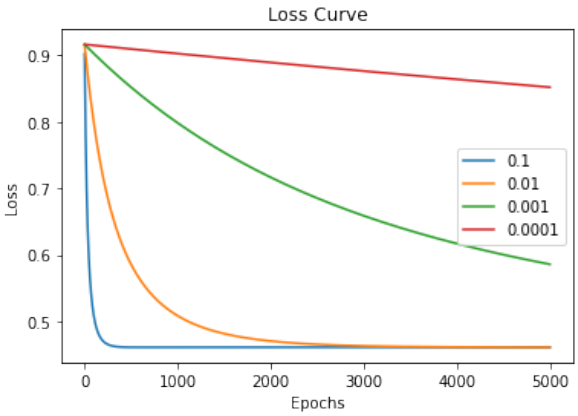

Figure 1: Validation Loss Curve for Tuning

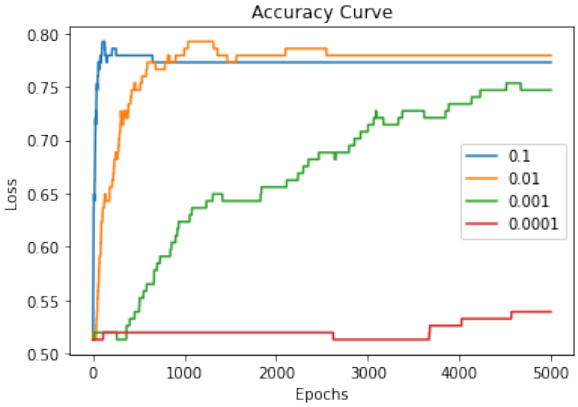

Figure 2: Validation Accuracy Curve for Tuning

2.3 Final Logistic Regression Model Training and Results

From the results of tuning, we can observe the best model is given by a learning rate of 0.1 and for around 100-200 epochs. I train the Final Model for 200 epochs and 0.1 learning rate and achieve an accuracy of 77.92 percent and a 66.62 F1-score. Loss curve and accuracy curve given below.

Confusion Matrix		
	0	1
0	92	8
1	26	28

Classification Report				
	Precision	Recall	F1-score	Support
0	0.78	0.92	0.84	100
1	0.78	0.52	0.62	54
Accuracy = 0.78				

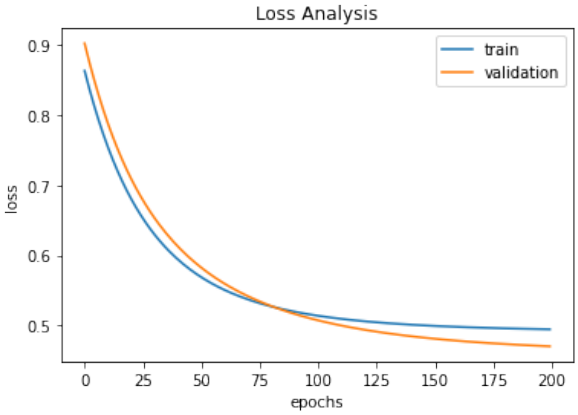

Figure 3: Final Model Loss Curve

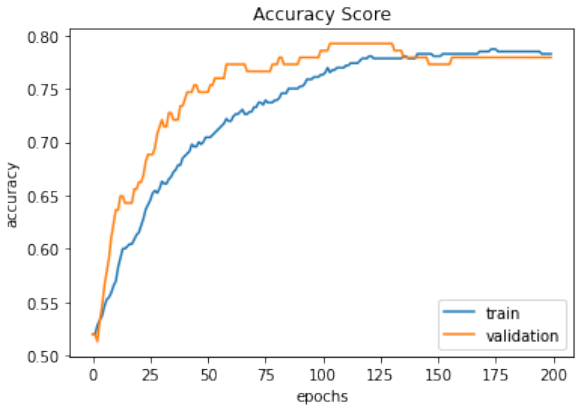

Figure 4: Final Model Accuracy Curve

3 Neural Network

3.1 Neural Network Design and Experimental Setup

For the second part of the assignment, I based my architecture decisions on experimenting with 1, 2 and 3 hidden layers, and decided to go with 1 hidden layer since the improvement from 1 layer to 2 hidden layer was not significant for 5000 epochs.

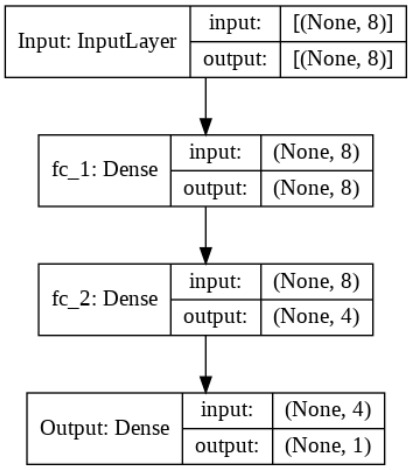

Figure 5: Neural Network Base Architecture

The first hidden layer (fc 1) has 8 neurons, and the second (fc 2) has 4 neurons and all the weights are glorot initialized, i.e. Uniform Xavier Initialization. The decision for the number of neurons was based upon a thumb rule that the number of neurons should be in between the number of features of the input and the output.

For tuning and training, I have added certain call backs, i.e. saving the logs and model checkpoints along with early stopping and reducing learning rate on plateau with a factor of 0.1. For this, I start with 0.01 as learning rate with Adam optimizer with default α and β values of 0.9 and 0.999.

3.1.1 Choosing the right hyperparameters: Activation functions

The decision of choosing what activation functions to use for hidden layers what was done through experimentation with ReLU and Sigmoid for a 100 epochs.

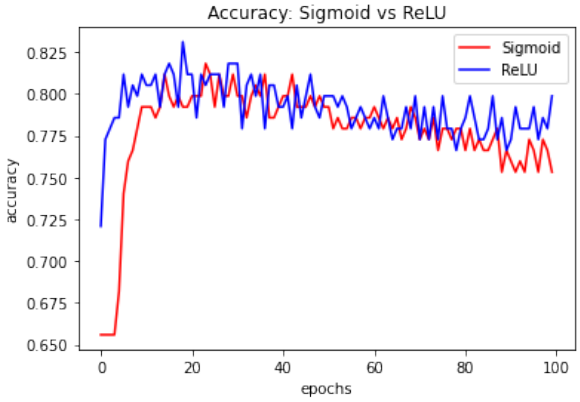

Figure 6: Comparison between ReLU and Sigmoid

From the above observations, I decide to go forward with ReLU as the hidden layer activation function for both the layers.

3.1.2 Choosing the right hyperparameters: Regularization Constant λ

I tested the model with different λ for both L1 and L2 on the test set. Both L1 and L2 penalize the model from having exploding gradients and thus, help in reducing over-fitting. The equations for both regularization's are given below:

$$Loss_{L1} = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} |w_i|$$

$$Loss_{L2} = Error(Y - \widehat{Y}) + \lambda \sum_{1}^{n} w_i^2$$

For each regularization type, models were trained on 3 different values of λ i.e 0.01, 0.005, 0.001. The following was the result obtained.

L1 Regularization Tuning			
Metric	$\lambda = 0.01$	$\lambda = 0.005$	$\lambda = 0.001$
Accuracy	0.7662	0.7532	0.7402
F1-Score	0.6326	0.6041	0.5918

L2 Regularization Tuning			
Metric	$\lambda = 0.01$	$\lambda = 0.005$	$\lambda = 0.001$
Accuracy	0.7792	0.7467	0.7337
F1-Score	0.6530	0.5894	0.5591

From the observations we can see that applying L2 regularization improved the F1 and Accuracy score for given architecture and we move forward with training our final model.

3.2 Final Neural Network Model Training

From the above observations we can see that the best model parameters are obtained using L2-regularization with $\lambda = 0.01$. The following are the observation on the test set.

Confusion Matrix		
	0	1
0	88	12
1	22	32

Classification Report				
	Precision	Recall	F1-score	Support
0	0.7662	0.7532	0.7402	100
1	0.6326	0.6041	0.5918	54
Accuracy = 0.78				

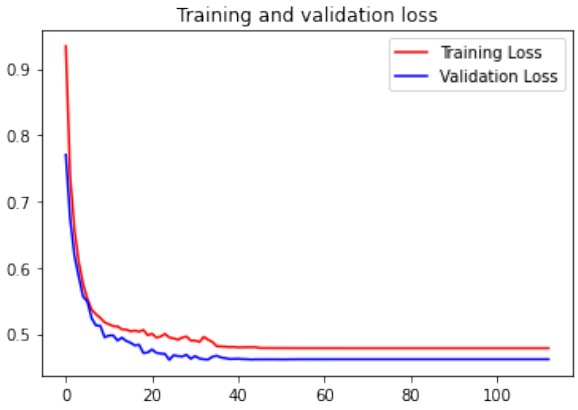

Figure 7: Neural Network Loss Curve

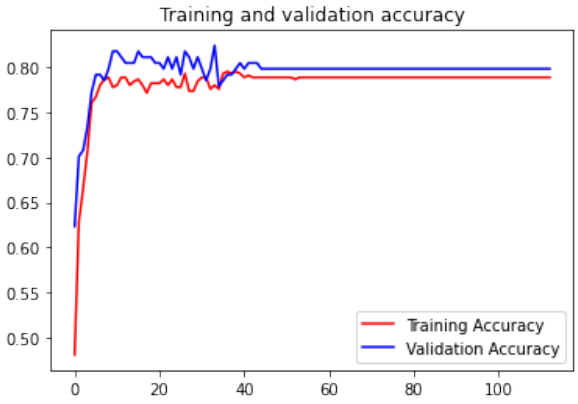

Figure 8: Neural Network Accuracy Curve

3.3 Dropout regularization

3.3.1 What is Dropout Regularization?

Dropout is a technique to deal with over-fitting by combining the predictions of many different large neural nets at test time. The key idea is to randomly drop units (along with their connections) from the neural network during training. The dropping of units is done layer wise and is done with a dropout parameter p which ranges from 0-1. Basically it is a term which denotes what fraction of neurons would be dropped at each epoch.

This prevents neurons from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single un-thinned network that has smaller weights.

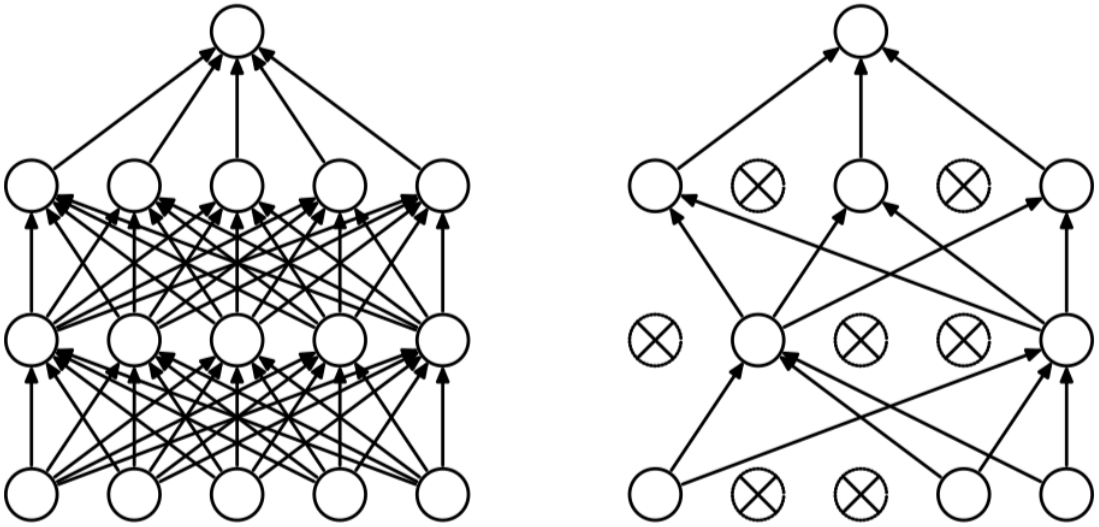

Figure 9: The Basic Idea of Dropout

3.3.2 Dropout: Model Training and results

I build a model using dropout regularization where I add 2 dropout layers after the first input layer as well as the hidden layer with rate as 0.125 and 0.25.

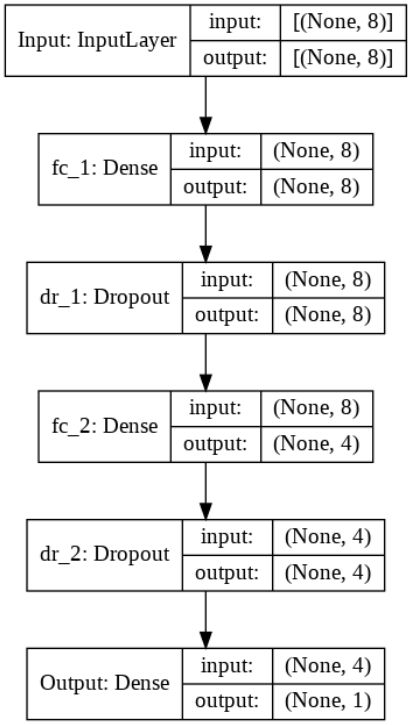

Figure 10: Architecture of the Dropout Regularised Model

The rest of the training hyper-parameters are the same as the base model i.e. Reduce Learning rate on plateau starting from 0.01 with an Adam optimizer. With those, I obtain the following results on the test set.

Confusion Matrix		
	0	1
0	90	10
1	23	31

Classification Report				
	Precision	Recall	F1-score	Support
0	0.80	0.90	0.85	100
1	0.76	0.57	0.65	54
Accuracy = 0.79				

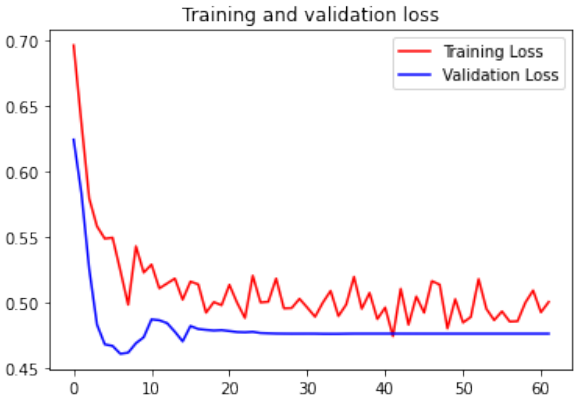

Figure 11: Dropout Model Loss Curve

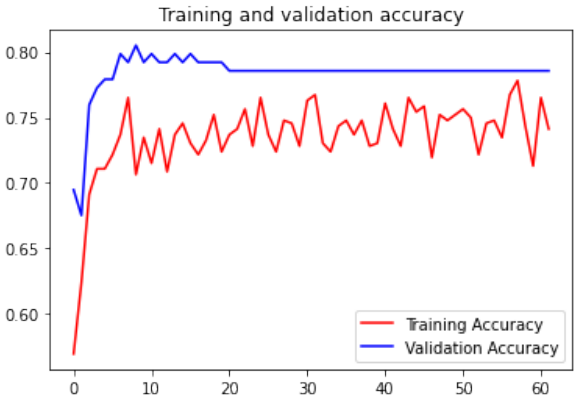

Figure 12: Dropout Model Accuracy Curve

4 Conclusion

Given are the observations on the test set from all the models built.

Model	Accuracy	Weighted F1-score
Logistic Regression	0.78	0.62
L1	0.76	0.6326
L2	0.78	0.65
Dropout	0.79	0.65

From the all the following models, I observed that the dropout model and the L2 regularized worked the best as both had similar performance. The assignment gave an overview of how we can create a logistic regression model and introduced us to neural networks and the related concepts, for example, optimizers like Adam, Adadelta, SGD etc.