Assignment 1

Rishi Jay Joshi
Department of Computer Science and Engineering
University at Buffalo, SUNY
New York, United States
rishijay@buffalo.edu

ABSTRACT

Part 1 of the assignment implements Gradient descent for Logistic Regression. [1] Logistic Regression is an algorithm that is suited for discovering the link between features or cues and some particular outcome. It is a supervised machine learning algorithm that is used to model the probability of a certain class or event existing like pass/fail, win/lose, alive/dead, healthy/sick etc. In its basic form, Logistic Regression uses a logistic function to model a binary dependent variable. Gradient Descent is an optimization algorithm that minimizes a cost/loss function and converges to a local minimum. As a result we find optimal weights. Gradient descent is an iterative algorithm. It takes repeated directions in the opposite direction of the gradient of the function at the current point because this is the direction of the steepest descent. Part 2 of the assignment implements Neural Networks for classification. A Neural Network or an Artificial Neural Network is a network or circuit of neurons composed of nodes. These are implemented by using Tensorflow and Keras. Part 3 of the assignment adds regularization to the neural network. Regularization is added to prevent overfitting.

1 Introduction

1.1 Logistic Regression

[2] Logistic Regression is a process of modelling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome something that can take two values. Logistic Regression is a classification which is used for binary and linear classification problems. [3] Some examples of classification problems include:-

- Email: Spam / Not Spam ?
- Online Transactions: Fraudulent (Yes / No)?
- Tumor: Malignant / Benign

In all these problems the variable ‘y’ we are trying to predict takes two values ‘0’ and ‘1’.

$$y \in \{0,1\}$$
0: “Negative Class” (e.g., benign tumor)
1: “Positive Class” (e.g., malignant tumor)

Hence, we want an hypothesis that satisfies $\mathbf{0 <= h_0(x) <= 1}$. Modifying the hypothesis of linear regression, we get $\mathbf{h_0(x) = g(\theta^T X)}$ where g can be defined as S(x) which is a sigmoid function:-

$$S(x) = \frac{1}{1 + e^{-x}}$$

To get the hypothesis of logistic regression we simply plug in $\boldsymbol{\theta^T X}$ to get

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

[4] The cost function represents optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error. The linear regression cost function is

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Using this will lead us to a non-convex function with many local minimums in which it would be very difficult to minimize the cost value and find the global minimum. Hence, for logistic regression we define the cost function as:-

$-log(h\theta(x))\ if y = 1$

$-log(1 - h\theta(x))\ if y = 0$

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) & \text{if } y = 1 \\ -log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

These two functions can be combined to form:-

$$J(\theta) = -\frac{1}{m}\sum \left[y^{(i)} \log(h\theta(x(i))) + \left(1 - y^{(i)}\right) \log(1 - h\theta(x(i))) \right]$$

1.2 Gradient Descent

Gradient descent is an algorithm that minimizes the cost function and converges to a local minimum. In this process we try different values and update them to reach optimal ones, minimizing the output. [4] We apply gradient descent to the cost function of logistic regression. This way, we can find an optimal solution minimizing the cost over model parameters:

$$\min_\theta J(\theta)$$

For n features we have n parameters for the θ vector. We run the gradient descent on each parameter θ_j.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

To update each parameter simultaneously, we loop through the parameters θ_0, θ_1,…, θ_n in vector $\theta = [\theta_0, \theta_1, \ldots, \theta_n]$.

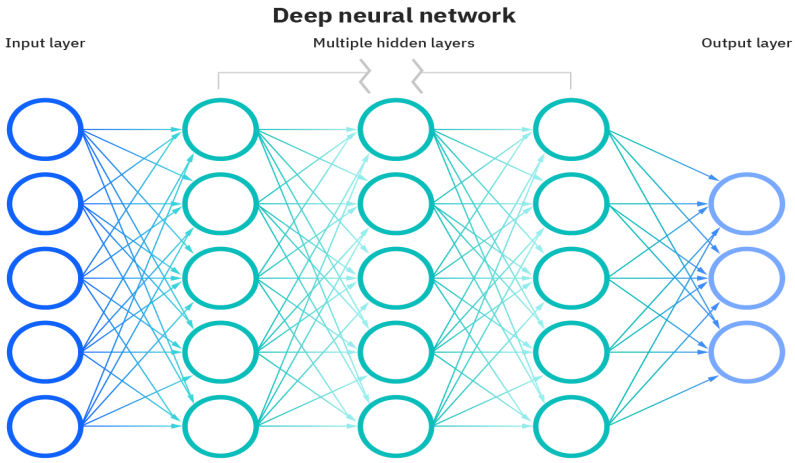
$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Plugging this into the gradient descent function leads to the update rule:

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

1.3 Neural Network

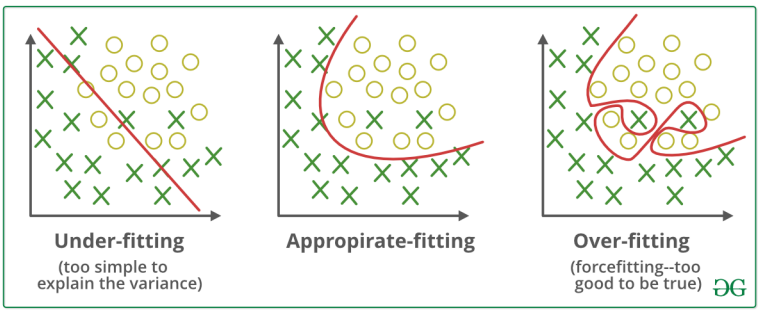
Neural Networks for machine learning and artificial intelligence are inspired by the biological neural networks found in the brain. They are based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection can transmit a signal to other neurons. The signal is a real number and the output of each neuron is computed by some non-linear function of the sum of its inputs. Neural Networks typically have some weights and biases. When the inputs are transmitted between neurons, the weights are applied to the inputs and passed into an activation function along with the bias. [5] The figure below shows the structure of a neural network.



1.4 Regularization

[7] In statistical terminology a fit refers to how well you approximate a target function. We use this concept in machine learning to approximate the unknown underlying mapping function for the output variables given the input variables.

Overfitting refers to a model that models the training data too well. This happens when a model learns the details and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This creates a problem as these concepts do not apply to the new data. Underfitting refers to a model that can neither model the training data nor generalize to new data. It will have poor performance on the training data. [6] The following image illustrates the concept of underfitting and overfitting.



Regularization is a technique used to reduce the error by fitting a function appropriately on a given training set and avoid overfitting. [8] Regularizers allow you to apply penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes. Regularization penalties are applied on a per-layer basis.

These layers expose 3 keyword arguments:

- kernel_regularizer: Regularizer to apply a penalty on the layer's kernel
- bias_regularizer: Regularizer to apply a penalty on the layer's bias
- activity_regularizer: Regularizer to apply a penalty on the layer's output

2. Experiment

2.1 Dataset

The dataset used for the experiment is the Pima Indians Diabetes Dataset. This dataset will be used for training and testing. The dataset contains 768 samples and 8 features. 60% of the dataset is used for training, 20% for testing and 20% for validation.

2.2 Python Editor

Google Colab based on Jupyter Notebook is used for implementation

2.3 Data Preparation

Sklearn's train_test_split is used to split the data into 60%, 20% and 20% for training, testing and validating respectively.

Sklearn's MinMaxScaler is used for the normalization of the dataset.

2.4 Logistic Regression Using Stochastic Gradient Descent

[1] For part 1 the stochastic gradient descent algorithm is applied for optimization. Stochastic gradient descent minimizes the loss function by computing its gradient after each training example, and nudging θ in the right direction (the opposite direction of the gradient).

The learning rate η is a hyperparameter that must be adjusted. If it's too high, the learner will take steps that are too large, overshooting the minimum of the loss function. If it's too low, the learner will take steps that are too small, and take too long to get to the minimum. It is common to start with a higher learning rate and then slowly decrease it, so that it is a function of the iteration k of training; the notation η_k can be used to mean the value of the learning rate at iteration k.

2.5 Neural Networks

For part 2 we implement neural networks using tensorflow and keras. First we create a sequential model. For designing the neural network we add one input layer, two hidden layers and one output layer. The activation function relu is used for each of our layers except the output layer for which sigmoid is used. For optimization we use the adam optimizer. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower order moments. Binary Crossentropy is used as our loss function. Finally we use the fit function to pass our training set, 200 epochs and our validation set.

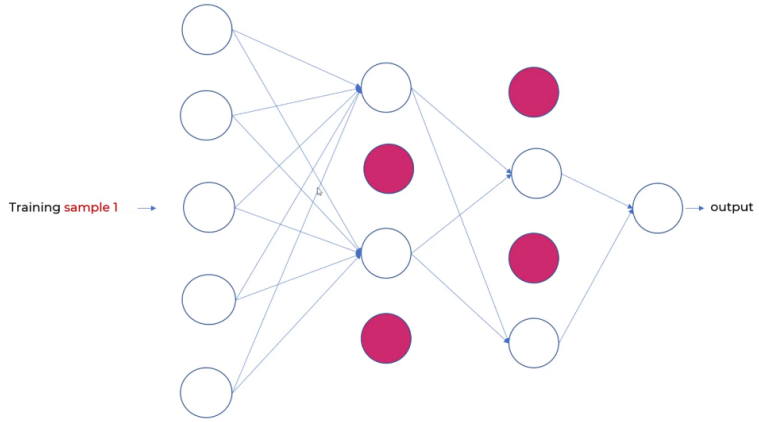
2.6 Regularization

We use the L2 regularization also called Ridge Regularization to prevent over-fitting in part 2. L2 regularization adds "squared magnitude" of coefficient as penalty term to the loss

function.. We use L2 as we do not have a huge number of features. The cost function for L2 regularization is

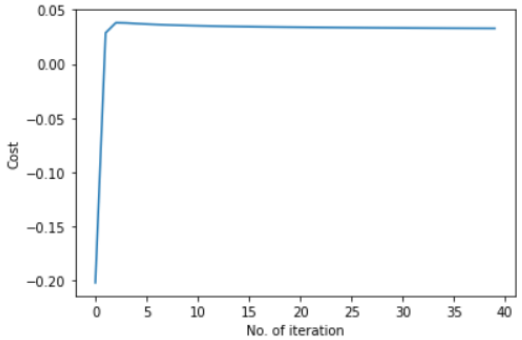
$$\text{L2 Regularization}$$
$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

We use dropout regularization to prevent over-fitting in part 3. [10] The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units 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. This significantly reduces overfitting and gives major improvements over other regularization methods. [11] The following image illustrates this phenomenon of randomly dropping the neurons which are colored.



3. Results

For part 1 we set the learning rate to 0.06 and the number of iterations to 4000. We get the following visualization of the cost function with respect to the number of iterations:-



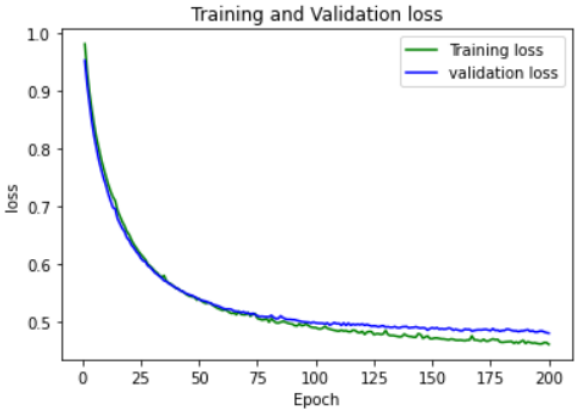
Finally, we get an accuracy of **74.34%** on the training set and **75.32%** on the testing set.

```
Training set accuracy =  74.34782608695652
Test set accuracy =  75.32467532467533
```

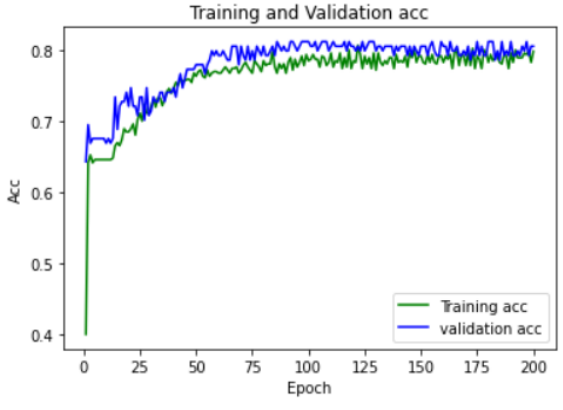
For part 2 we create the neural network using tensorflow. use the L2 regularizer from Keras and add it to three of our hidden layers of the neural network. Our neural network consists of one input layer, three hidden layers and one output layer. The input layer consists of 24 neurons. The two hidden layers consist of 12 neurons each. The Output layer consists of 1 neuron. We use the adam optimizer. Mean Squared Loss Error is used as our loss function. We then pass the training set and the validation set and train on 200 epochs. The model summary is as follows:-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_84 (Dense)             (None, 24)                216
_________________________________________________________________
dense_85 (Dense)             (None, 12)                300
_________________________________________________________________
dense_86 (Dense)             (None, 12)                156
_________________________________________________________________
dense_87 (Dense)             (None, 1)                 13
=================================================================
Total params: 685
Trainable params: 685
Non-trainable params: 0
_________________________________________________________________
```

We get the following chart of training loss vs validation after training



We get the following chart of training accuracy vs validation accuracy after training



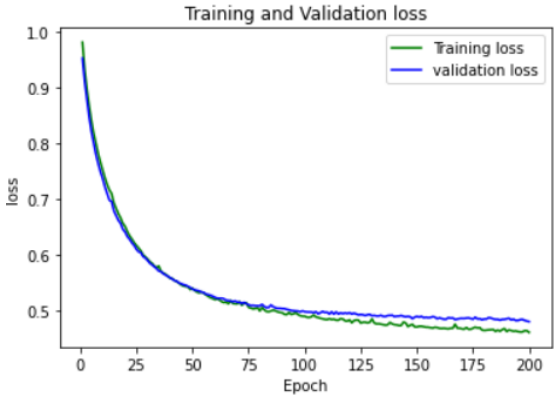
Finally we get the total accuracy of **80.52%** on the test set

```
6/6 [==============================] - 0s 2ms/step - loss: 0.4808 - accuracy: 0.8052
```

For part 3 we use the same model we created for part 2 (which uses L2 regularizer) and create another model for comparison that uses dropout regularization. The input layer consists of 12 neurons. The two hidden layers consist of 12 neurons each. The Output layer consists of 1 neuron. The model summary is as follows:-

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_88 (Dense)             (None, 24)                216
_________________________________________________________________
dense_89 (Dense)             (None, 12)                300
_________________________________________________________________
dropout_14 (Dropout)         (None, 12)                0
_________________________________________________________________
dense_90 (Dense)             (None, 12)                156
_________________________________________________________________
dropout_15 (Dropout)         (None, 12)                0
_________________________________________________________________
dense_91 (Dense)             (None, 1)                 13
=================================================================
Total params: 685
Trainable params: 685
Non-trainable params: 0
_________________________________________________________________
None
```

We get the following chart of training loss vs validation loss after training



We get the following chart of training accuracy vs validation accuracy after training

Finally we get the accuracy of **78.57%** on the test set

```
6/6 [==============================] - 0s 2ms/step - loss: 0.4525 - accuracy: 0.7857
```

We can see that the L2 regularization method performs better. This is because L2 regularization achieves higher predictive accuracy than dropout in a small network since averaging the learning model will enhance the overall performance when the number of sub-model is large and each of them must be different from each other.

4. References

[1] Daniel Jurafsky & James H. Martin - Speech and Language Processing

[2] Thomas W. Edgar, David O. Manz - Research Methods for Cyber Security, 2017

[3] Andrew Ng – Machine Learning

[4] A. Aylin Tokuç - Gradient Descent Equation in Logistic Regression

[5] IBM - Neural Networks

[6] GeeksforGeeks - Underfitting and Overfitting

[7] Machine Learning Mastery - Overfitting and Underfitting With Machine Learning Algorithms

[8] Tensorflow Documentation - tf.keras.regularizers.Regularizer

[9] Diederik P. Kingma, Jimmy Ba - Adam: A Method for Stochastic Optimization

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov - Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[11] Codebasics - Dropout Regularization