Deep Learning HW01

Siffi Singh – 0660819

Program to build a deep neural network model by using the backpropagation and stochastic gradient descent algorithm. In this report I'll be explaining:

Part 1: Regression

a) Performance of root-mean square
b) Showing:
    1. Network architecture
    2. Learning curve
    3. Training RMS error
    4. Testing RMS error
    5. Regression result with training labels
    6. Regression result with test labels
c) Feature selection procedure

Part 2: Classification

a) Classification Performance
b) Showing:
    1. Network architecture
    2. Learning curve
    3. Training error rate
    4. Testing error rate
c) Latent Feature distribution

Part 1: Regression

---

a) Performance of root-mean square

Regression predictive modeling is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). In our case, we use the **sum-of-squares** error function as the loss function and evaluate the **Root-mean square** error. **The heating load is predicted** by using the architecture as designed and shown below in the network architecture. With this designed architecture, I was able to achieve the following RMS values:

RMS training error: 3.6572
RMS testing error: 3.7713

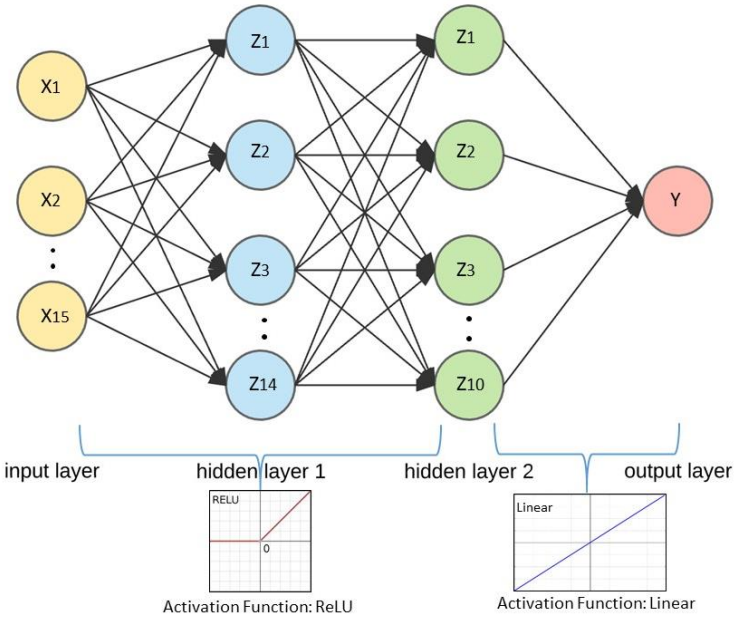
```
RMS error: 3.65723624697
RMS error: 3.77133432163
```

The first one is the training error, second is the testing error from the code based on the following explained network architecture.

---

b) 1. Network architecture

The network plays the most important role in determining the result of Regression on the given energy efficiency dataset. Therefore, the network architecture has been designed and checked on different combinations of number of neurons, the type of activation function used, the learning rate and the batch size.
Furthermore, the functions for backpropagation and stochastic gradient descent have all been written without the help of any library function. General details about the network architecture:
Total examples: 768
Training examples: 576 (75% of total samples)
Testing examples: 192 (25% of total samples)
Loss Function Used: Sum-of-squares error function
Evaluation Function: Root Mean Square Error
Input layer: The input layer had 15 entries, from which **8 features** were represented. A total of **15 input** entries because the feature “Orientation” has been one hot encoded as a vector size of 4, and the feature “Glazing Area Distribution” has been one hot encoded to get 5 entries in the input vector.



Activation functions used: **ReLU(for first L-1 layers), Linear(for the last layer)**
No. of layers**: 4-layer** architecture
Layer Dimensions: **[15, 14, 10, 1]**
Learning rate: **0.0095**
Optimization Strategy: **Stochastic Gradient descent**

The intuition for choosing the **activation function** as chosen was that the output had to be the heating value, and we used considerably more reliable activation function for the other hidden layers, i.e. "ReLU". The last layer uses "linear" than other activation functions.

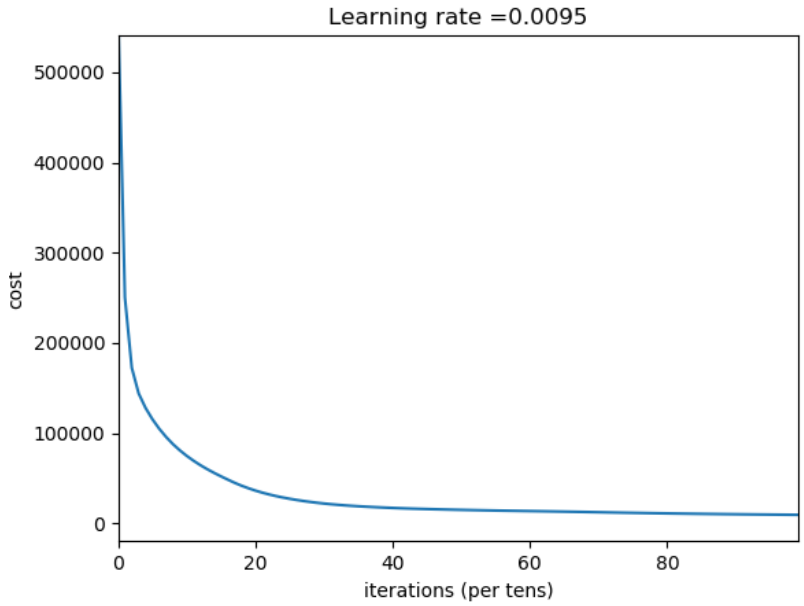
The initiation of choosing the **number of neurons** as chosen is again derived by **hit-n-trial** of multiple set of learning rates, iterations, number of neurons and finally the one which gave the least RMS error and had the training curve right, along with the training and testing curve showing minimum gap (showing no overfitting) was chosen.

The network begins with [15*576] matrix (input data) and the weight matrix and bias are initialized. Once the network undergoes the forward propagation, and reaches the last layer, it computes the loss and uses the same network in the reverse order computing the partial differentiation and proceeding by performing **backpropagation**. In every iteration, rather than doing the conventional gradient descent, we here use the **stochastic gradient descent** which is equivalent to mini batch gradient descent where each min-batch is just 1 example, hence, this means that we will be computing the gradients on **just one training example at a time**. With each iteration, we aim to move in the direction of reducing loss. After all the iterations are done, and we see the **loss value converged**, our model is ready for prediction.

b) 2. Learning curve

The learning curve for the designed neural network for Regression by using backpropagation and stochastic descent has been plotted after verifying multiple values of **learning rate ranging from** as small as 0.00001 to 0.001 and the result on which the best result was seen for the chosen network architecture was selected.

The expected curve is supposed to saturate after it has leaned from enough number of examples making the model's loss stable running the same process of forward and backward propagation with update of parameters for certain number of iterations and from the computed costs, the curve looks like as follows:



The algorithm seems to have converged after about 6000 iterations or so, for the given layer dimensions of [15, 14, 10, 1] and a learning rate of 0.0095. To ensure that the model has not over fitted, I plotted the loss curve for train and test and confirmed that the gap if nominal.

b) 3. Training RMS error

The training RMS error was determined by getting the labels after performing forward and **back propagation** and optimizing the weight values using the stochastic gradient approach in each iteration to minimize the loss function of sum-of-squares. After the number of iterations get over, we get the final predicted labels for the training samples and we check the accuracy of the true label with the predicted label based on the **training RMS error: 3.6572**

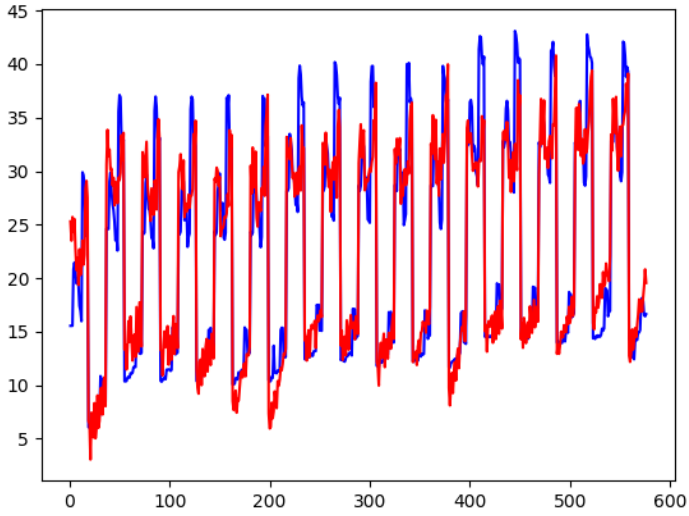
b) 4. Testing RMS error

We use the trained model trained on the 576 samples to test its performance on the 192 samples which have been shuffled. On predicting the heating load using the model derived from the above architecture, we get the following testing **RMS error: 3.7713**

The testing RMS error also helps us rectify the overfitting issues, if there would be a lot of difference in the RMS values of training and testing dataset.

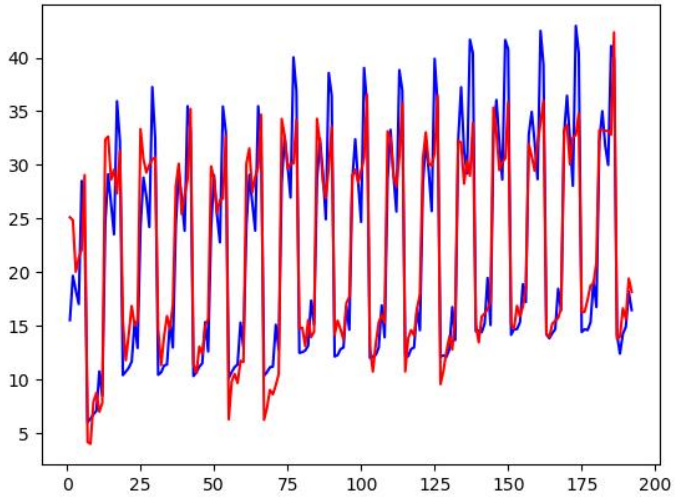
b) 5. Regression result with training labels

Following is the regression result with training labels:



b) 6. Regression result with test labels

Following is the regression result with testing labels:



c) Feature selection procedure

The feature selection process helps us in determining the quality features from the set of given features. Theoretically the features that are more distinct in the given samples and make give a sample its distinguishing properties are the

features that influences the result the most. For the current process of checking which feature affects the final result, I started by **removing one feature at a time** and see if the loss value is changed or not.
If by the removal of the feature the loss value is not affected, then it means that the feature doesn't have that much influence on the performance. We can also see this effect of influence when we are plotting the true labels and the predicted labels. I did this for every feature and came at the conclusion that the "Surface Area" influences the result the most, the other features that follow are "Relative Compactness" and "Overall height", since on removal of these features from the given features the loss value and the plot were severely impacted.

Part 2: Classification

a) Classification Performance

Classification predictive modeling is the task of predicting a discrete class label. In our case of binary classification for the given dataset of ionosphere, where each sample is being categorized as either 'good' as 1 or bad as '0'.
In our case, we try to classify the given dataset using the 34 feature set and cross-entropy function as the loss function. We train the deep neural network model by using the backpropagation and stochastic gradient descent without the use of library functions. The classification performance after a lot of changes in the no. of layers chosen and the layer dimensions was selected based on the best accuracy result in multiple runs. Using the designed model, whose architecture is explained in detail in the next section, we minimize the cross entropy loss and were able to achieve an accuracy of:
Training Accuracy: 0.7553
Testing Accuracy: 0.8342

```
Accuracy: 0.755362751718
Accuracy: 0.834271920122
```

The first one is the training accuracy, second is the testing accuracy from the code based on the following explained network architecture, where the accuracy is measured as the no. of matches between the true and predicted label.

b) 1. Network architecture

The process is similar to the above architecture except that the input is different and is designed for a different case, i.e. classification. For the purpose of classification, we need to customize the no. of neurons we select or the number of layers we select. Furthermore, the choice of the activation function also needed to keep in mind the classification criteria and choose the one which would intuitively give best performance. The following architecture as shown in the diagram below derived after several simulations is based on the following:

Total examples: 350

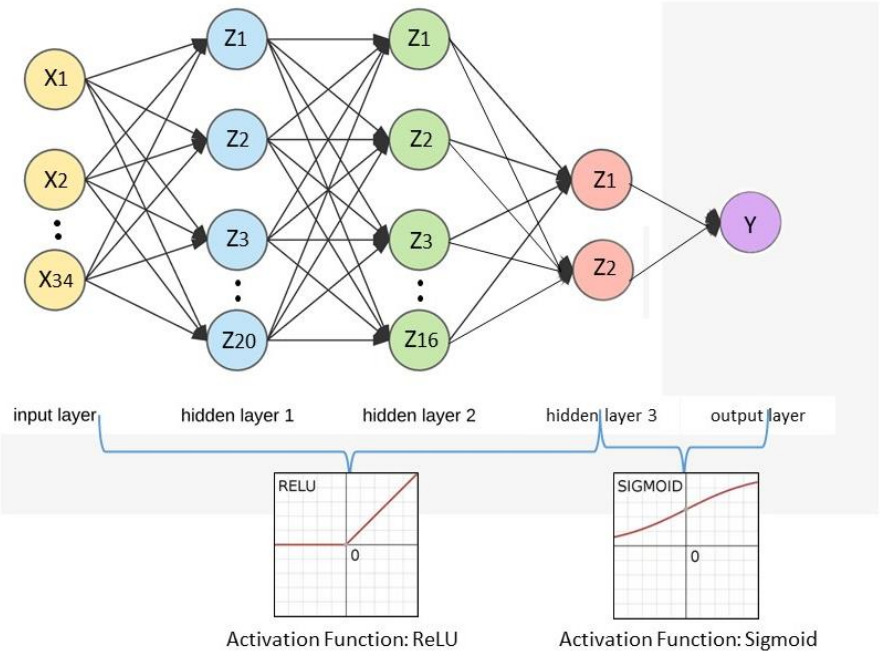
Training examples: 280 (80% of total samples)

Testing examples: 71 (20% of total samples)

Loss Function Used: **Cross-Entropy Error function**

Evaluation Function: **Accuracy**

Input layer: The input layer had **34 different features**.



The figure above shows the no. of neurons selected and the activation functions used while training the model for the classification problem of ionosphere dataset.

Activation functions used: **ReLU(for first L-1 layers), Sigmoid(for the last layer)**

No. of layers**: 5-layer** architecture

Layer Dimensions: **[34, 20, 16, 2, 1]**

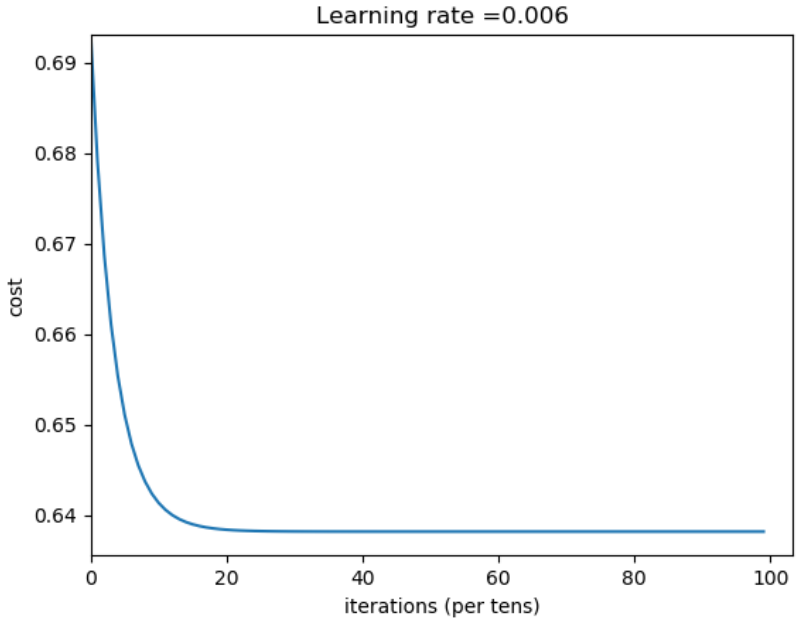
Learning rate: **0.006**

Optimization Strategy: **Stochastic Gradient descent**

Process: The input here had 280 examples to train the model, starting by shuffling the dataset.
The **intuition of selecting sigmoid** for the last layer was that the output needed a classification value, and once we get the probability value, we can label it as '0' or '1' based on a threshold value (which in our case is 0.5).
For the **other layers, ReLU** has been considering its good performance when checked for multiple epochs. The learning rate chosen here is 0.006, this value was chosen after performing **hit-'n'-trial** on several values ranging from 0.001 to 0.0001 and finally considering the shift in the loss value, and its stability, the value of 0.6 was chosen.
I tried multiple combinations of activation functions, with ReLU, linear and sigmoid. Considering the inability of Tanh to saturate, the activation function was not tried for the intermediate layers. There is also this possibility of model overfitting, for which the loss curve for training and testing has been verified.

b) 2. Learning curve

Here is the loss curve for the classification problem. The model seems to converge after only 4000 iterations, but for the sake of clarity some more iterations have been plotted.



The learning rate value is selected that gave the best accuracy results on both training and testing dataset.

b) 3. Training error rate

The training accuracy was determined by getting the labels after performing forward, cost estimation and back propagation and optimizing the weight values using the stochastic gradient approach in each iteration to minimize the loss function of cross entropy. After the number of iterations get over, we get the final predicted labels for the training samples and we check the accuracy of the true label with the predicted label based on the Training Accuracy: 0.7553. The error rate in this case is 0.2447. Hence the **Training Error Rate: 0.2447**

b) 4. Testing error rate

We use the trained model trained on the 280 samples to test its performance on the 71 samples which have been shuffled. On predicting the class using the model derived from the above architecture, we get the following Testing Accuracy: 0.8342. Based on this, we can conclude that the **Testing Error Rate: 0.1658**

c) Latent Feature distribution

The latent feature distribution here is the plot of the distribution of the features just a layer before the last layer. Since, in our architecture, we have chosen the second last layer's dimensions as having only 2 neurons (features), the 2D feature plots can be visualized in the following way:

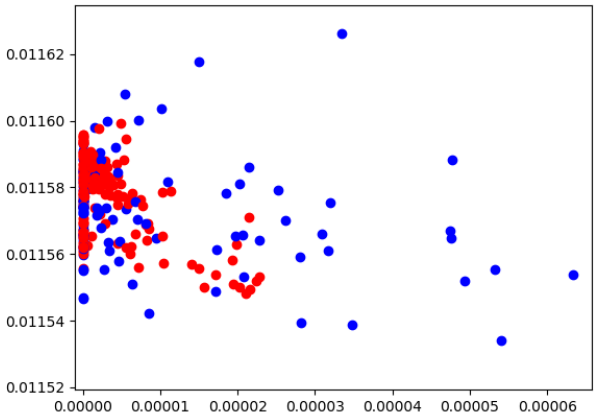


Figure 1: 20 epoch

The plot shows clearly that the red and blue labels have not separated at 20th epoch. To see at which epoch the data points started separating, I stored the intermediate epoch plots and following two results based on epoch 2500 and

epoch 4000(after convergence) shows the final separated data in the 2D feature space.

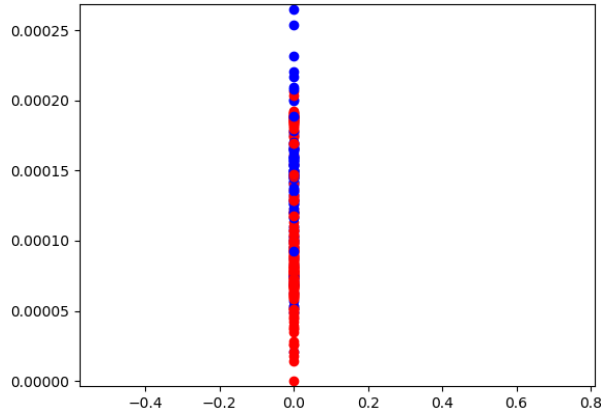


Figure 2: 2500 epoch

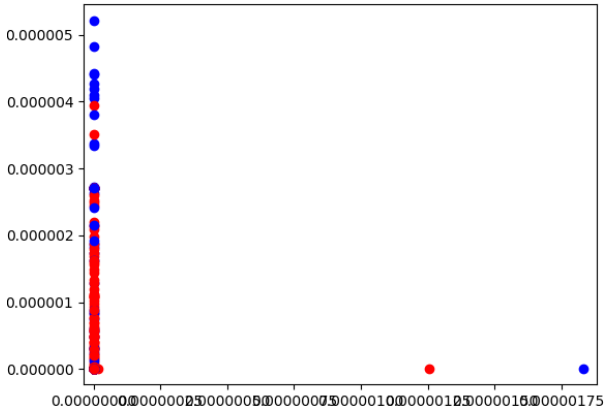


Fig 3: 4000 epoch

The above results have been zoomed to show the clear separation of data points, hence, it might look like the **axis is shifted**, but it is not, it has just been **zoomed to visualize better.**

Also from the above plot we learn that somewhere between 2500-4000 epoch the learning curve reaches its saturation point and the data is separated properly.

Inferences from the result

From this homework, I have learnt:

1) The use of forward and backpropagation algorithm to train the Neural Network successfully and the predictions convergence on the true values.
2) The importance of the selection of hyper parameters such as learning rate, the impact of which can lead us to having multiple layers or just a few layers to get the same result.
3) Selection of the activation function best suited for the problem at hand. Although there is no proper technique to find out which activation function will work for which case. The only way is to try and analyze the obtained accuracy/error values.
4) Implementation of basic function like sigmoid, ReLU by ourselves, which we could not have learnt if we had to use a library function.