Arjun Prakash

Student Id: 21239525

Course: MSc. AI (1MAI)

Assignment 2 - CT5165 Principles of Machine Learning

Description: Neural Network

The developed Neural network consists of two hidden layers with 6 neurons in each hidden layer and one output layer. The model takes input of n-dimension and classifies it into 2 classes, 0 or 1. Language used: Python. The model consists of 7 functions as mentioned below.

1. __init__(x,y)

This function takes in 2 parameters: (input and output labels) and initializes it to the object's input and output variable. It initializes random weights (for layer 1 to 2) of dimension (n,6) where n value depends on the input data dimension, random weights of dimension (6,6) for layer 2 to 3, random weights of dimension (6,1) for the layer 3 to 4. It also initializes loss variable for calculating the loss for each epoch.

2. feed_forward()

Feed forward is done by summing the products of each neuron value in the layer with its weights and passing it through an activation function, sigmoid in this case. The same process is done for every neuron of every layer in the network except the input layer.
Sigmoid function: $\boldsymbol{f(x) = \frac{1}{1+e^{-x}}}$, where (x) =$\sum w \times a^{(L-1)} + bias$, (This is done for all neurons in the layer). Here bias is set to 0.

3. back_propagate()

Back propagation[3] is the algorithm where the model learns. This functions first calculates the loss between the actual 'Y' (label value) and the output layer (predicted value) using Mean squared error.

Mean Squared Error (cost)= $\boldsymbol{\frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y})^2}$

After calculating the error, we backpropagate and change the weights with respect to the error. To do this, we need to compute the ratio of change in cost and change in weights $\left(\frac{\partial C_0}{\partial w^{(L)}}\right)$

$\boldsymbol{C_0=\left(a^{(L)} - y\right)^2}$, where a = activation of output layer and y= actual class label
$\boldsymbol{Z^{(L)} = w^{(L)}a^{(L-1)} + b}$, where w= weights of layer 'L' and 'a' is the activation (previous layer), b= bias
$\boldsymbol{a^{(L)}=\sigma\left(Z^{(L)}\right)}$, where a= activation at layer 'L' and σ = sigmoid function as stated above.

To find change in cost w.r.t (With respect to) weights (or how sensitive the cost is with change in weights) in a Layer, we use chain rule which includes the change in all the intermediate variables like 'Z', activation, and weights. The change in weights $w^{(L)}$, causes some change in $Z^{(L)}$ which in-turn causes some changes in the activation $a^{(L)}$ which influences the cost C_0. To find this [2][3][4]:

$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$ (Using chain rule)

Where, in general:

Arjun Prakash

Student Id: 21239525

Course: MSc. AI (1MAI)

$\frac{\partial z^{(L)}}{\partial w^{(L)}} = \boldsymbol{a^{(L-1)}}$: $\quad Activation\ of\ previous\ layer.$

$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \boldsymbol{\sigma}'(z^{(L)})$: $\quad Derivative\ of\ sigmoid\ function, which\ is\ \boldsymbol{\sigma}'(\mathbf{x}) = \boldsymbol{x}(\mathbf{1} - \boldsymbol{x}).$

$\frac{\partial C_0}{\partial a^{(L)}} = 2\left(\boldsymbol{a^{(L)}} - \boldsymbol{y}\right)$: $\quad Derivative\ of\ cost\ function, where\ a^{(L)}\ is\ the\ predicted\ output\ layer.$

The same is applied for all the layer and each neuron.

The Change in last layer weights w.r.t cost is calculated by (Applying chain rule as mentioned above):

$\boldsymbol{\partial w_3^*} = \mathbf{2}\left(\boldsymbol{a^{(L^3)}} - \boldsymbol{y}\right)\ \boldsymbol{\sigma}'\left(\mathbf{L^3}\right), \quad where\ \boldsymbol{L^3} is\ output\ layer,\ \ '\boldsymbol{a}'\ is\ activation, \boldsymbol{y}\ is\ actual\ label$

$\boldsymbol{\partial w_3}\ = \boldsymbol{a^{(L^3-1)}}.\boldsymbol{\partial w_3^*}, \quad These\ change\ in\ weights\ are\ added\ with\ the\ network's\ weight\ 3$

<u>For all other weights, the loss will be calculated by:</u>

$\boldsymbol{\partial w_2^*} = \left(\boldsymbol{\partial w_3^*}.\boldsymbol{w_3}\right)\ \boldsymbol{\sigma}'(\boldsymbol{L^2}),$ where $\boldsymbol{L^2}\ is\ layer\ 2,\ \ '\boldsymbol{w_3}'\ \boldsymbol{is\ weights\ of\ 3-4th\ layer}$

$\boldsymbol{\partial w_2}\ = \boldsymbol{a^{(L^2-1)}}.\boldsymbol{\partial w_2^*}, \qquad These\ change\ in\ weights\ are\ added\ with\ the\ network's\ weight\ 2$

$\boldsymbol{\partial w_1^*} = (\boldsymbol{\partial w_2^*}.\boldsymbol{w_2})\boldsymbol{\sigma}'\left(\mathbf{L^1}\right), \qquad where\ \boldsymbol{L^1} = Layer\ 1,\ \ '\boldsymbol{w_1}'\ is\ \boldsymbol{weights\ of\ 2-3th\ layer}$

$\boldsymbol{\partial w_1} = \boldsymbol{a^{(inp)}}.\boldsymbol{\partial w_1^*}, \qquad These\ change\ in\ weights\ are\ added\ with\ the\ network's\ weight\ 1$

All the above changes in weights are then added with the actual weights of respectively layer. These changes in weights are done in every epoch. Updated loss is then saved in the loss (list), every epoch for further visualization. Loss function used for visualization: **Mean Squared Logarithmic Error Loss.**

4. fit_nn(epochs)

This function takes in epochs as parameter and calls the function feed_forward and back_propagate till the range of given epochs. The back_propagate function updates the weights whenever its called.

5. predict_nn(test_input)

The predict function takes in test input as a parameter and performs similar steps as feedforward function but using the test label as input and using the trained weights. Then it returns the predicted class label.

6. getweights()

Returns the latest trained weights of the model.

7. __str__()

The str() method prints the current network shape of each layer in below format:

```
Network is of shape:(136, 9),(6,),(6,),(1,)
```

Arjun Prakash

Student Id: 21239525

Course: MSc. AI (1MAI)

Preprocessing :

- Removed inconsistency from the dataset by removing spaces/tabs and converting the txt file to csv file.
- Replaced labels in dataframe from yes/no to 1/0.
- Dropped the label column from the data for training and testing set. Split the data into 2/3rd training and 1/3rd testing.
- Normalized the data using StandardScalar() [6], which scales the data and makes the mean of data as 0 and standard deviation as 1.

Training:

Own Model from Scratch:

Implemented a neural network with 2 hidden layers, with number of neurons as 6 in both the layers and activation function as Sigmoid.

Trained the classifier (using fit() function) using backpropagation with normalized training data for 1800 epochs. Stored the loss of each epoch and divided the number with the size of training data (completely optional) to make the loss value more normalized. Stored the F1 score for 10 different training samples. Also added an option to pickle the trained weights[5].

Sklearn Implementation:

Implemented MLPClassifier using same parameters. Hidden layers = 2 with same number of neurons as in scratch implementation and activation function as "tanh"(sigmoid).

Trained the classifier using the same normalized data that was used for the scratch implementation with the same epoch (1800) range. Stored the F1 score for 10 different training samples.

Testing:

Own Model from Scratch, F1 score:

Achieved F1 score in the range (testing data) = 0.8 – 0.94+

Average F1 score over 10 different training = 0.87

Sklearn Implementation F1 score:

Achieved F1 score in the range (testing data) = 0.78 – 0.94+

Average F1 score over 10 different training = 0.85

Observation:

Could see that Sklearn and the own implemented model had similar F1 Scores, for example, at any given point, if the sklearn model classified the data at 0.78 f1 score then the own implemented model also had the score around the same range (0.77-0.82) and when the score of sklearn was around 0.92, then the

Arjun Prakash

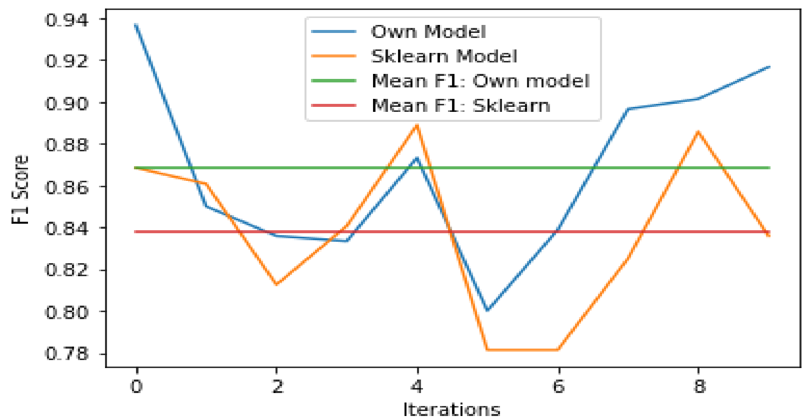
Student Id: 21239525

Course: MSc. AI (1MAI)

implemented model had the score in range (0.91-0.94). On average, most of the time, the sklearn model had an average f1 score of 2-3% below the f1 score of built from scratch model (For 10 distinct training samples).

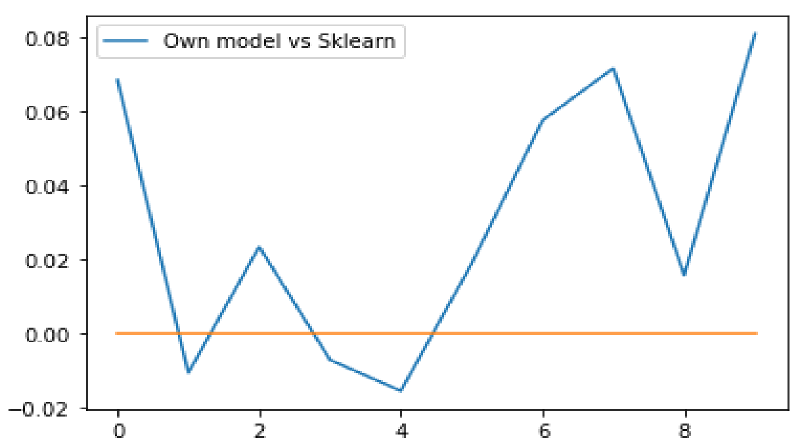
This can be seen in the below visualizations.

1. Model comparison



Here the Blue line represents the f1 score of the own built from scratch model and the orange line represents the f1 score of sklearn MLPClassifier over 10 different trainings. The Green line is the mean f1 score of the built from scratch model and the red line is the mean f1 score of sklearn MLPClassifier. Could see similar results every time the model was tested.

2. Difference in the accuracy/score



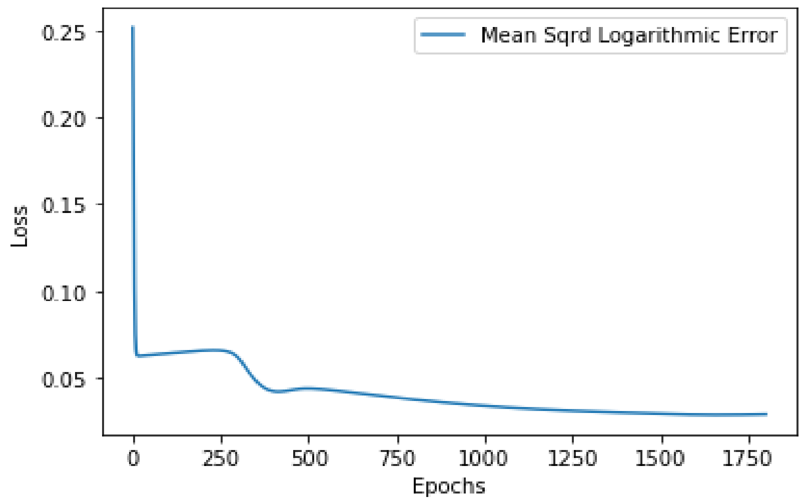
In this graph, the difference in both the models are calculated. When both the model classifies with the same accuracy/f1-score then their difference is 0, which means, if the datapoint touches the yellow line then both the models have same f1 score. For points below the yellow line, Sklearn classifier had better accuracy. Also, farther the data point is from the yellow line, higher the magnitude of difference in the f1 score of both the models.

Arjun Prakash

Student Id: 21239525

Course: MSc. AI (1MAI)

3. Loss function:



With the above loss function graph, we see that the loss is highest at first few epochs as the model has random weights assigned to it, thus the classification accuracy is not good initially. As the model gets trained, we see the loss function (or cost function) dropping till it saturates. (The loss is summed at axis=1 and multiplied with -1 to make the loss value positive)

Additional

- Exported classified data to file 'Predictions_x.csv' for all the 10 training iterations.
- To get the pickled weights data, uncomment the pickling line in training module, after the prediction file export. Pickle File name: 'model_x.pkl'.

References:

[1] **https://www.youtube.com/watch?v=WUvTyaaNkzM** **–** 3blue1brown - Calculus

[2] **https://www.youtube.com/watch?v=tIeHLnjs5U8** **–** 3blue1brown - Backpropagation calculus

[3] **https://www.youtube.com/watch?v=Ilg3gGewQ5U** **-** 3blue1brown -Backpropagation

[4] **https://www.youtube.com/watch?v=aircAruvnKk** **-** 3blue1brown – Neural network

[5] **https://docs.python.org/3/library/pickle.html** **-** Pickling

[6] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html-scalar transformations

Appendix (code):

```
# Importing and Class implementation

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
import pickle as pk

#Implementing sigmoid function
def sig(x):
    return 1/(1+ np.exp(-x))

#implementing NeuralNetwork class
class NeuralNet:
    def __init__(self, x, y):
        self.input    = x                 #Initializing Input layer
        self.y        = y                 # Initializing actual outputs
        # Initializing Weights for input to 1st layer of dimension (n,6) where n is the input dimension
        self.weights_l1   = np.random.rand(self.input.shape[1],6)
        # Initializing Weights for layer 2 to 3
        self.weights_l2 = np.random.rand(len(self.weights_l1[0]),6)
        # Initializing weights for layer 3-4 with dimension (6,1)
        self.weights_l3 = np.random.rand(len(self.weights_l2[0]),1)
        # Initializing predicted output neuron
        self.output     = np.zeros(y.shape)

        # Initializing layers
        self.layer1=0
        self.layer2=0
        self.output=0

        # List for storing the decrease in loss with respect to epochs
        self.loss=[]

    # Feed forward implementation without using bias variable
    def feed_forward(self):
        self.layer1=sig(np.dot(self.input,self.weights_l1))
        self.layer2=sig(np.dot(self.layer1,self.weights_l2))
        self.output=sig(np.dot(self.layer2,self.weights_l3))

    # Backpropagation for training the model and reducing the loss
    def back_propagate(self):
        #storing loss of each epochs in a list
        self.loss.append(1/len(self.input)*np.log((self.y+1)/(self.output+1))) #Mean Squared Logarithmic Error Loss.

        # Updating weights of 3rd to 4th layer and storing it in der_weights
        change_w3=2*(self.y - self.output)*self.output*(1-self.output)   #
        der_weights_3 = (1/len(self.input))*np.dot(self.layer2.T,change_w3)

        # Updating weights of 2nd to 3rd layer and storing it in der_weights
        change_w2=np.dot(change_w3,self.weights_l3.T) * self.layer2*(1-self.layer2)
```

```
        der_weights_2 = (1/len(self.input))*np.dot(self.layer1.T,change_w2)

        # Updating weights of 1st to 2nd layer and storing it in der_weights
        change_w1=np.dot(change_w2,self.weights_l2.T)*self.layer1*(1-self.layer1)
        der_weights_1= (1/len(self.input))*np.dot(self.input.T,change_w1)

        # Updating all the weights
        self.weights_l1+=der_weights_1
        self.weights_l2+=der_weights_2
        self.weights_l3+=der_weights_3

    # Function for training the model
    def fit_nn(self,epochs):
        for i in range(epochs):
            self.feed_forward()
            self.back_propagate()

    # Function for predicting the model
    def predict_nn(self,test_input):
        test=sig(np.dot(test_input,self.weights_l1))
        test1=sig(np.dot(test,self.weights_l2))
        testout=sig(np.dot(test1,self.weights_l3))
        return testout


    # For returning weights
    def getweights(self):
        return self.weights_l1, self.weights_l2,self.weights_l3

    def __str__(self):
        return "Network is of shape:{0},{1},{2},{3}".format(self.input.shape,self.weights_l1[0].shape,self.weights_l2[
0].shape,
                                                          self.weights_l3[0].shape)



# Preprocessing

# Naming all the columns of the data file. Input own column names by changing this variable
col=["fire", "year", "temp", "humidity", "rainfall", "drought_code", "buildup_index", "day", "month", "wind_speed"]


# Reading the data file, change this path depending on the file location
data=pd.read_csv("D:\StudyMaterial - MSc AI\Semister 1\ML\Assignment-2\wildfire.txt",names=col)

# Preprocessing all the texts of output column to 0/1
data['fire']=data['fire'].replace("no",0)
data['fire']=data['fire'].replace("yes",1)


# Training

# Variable for storing F1 score of the model
own=[]
# List for storing F1 score of sklearn classifier
```

```
skf1=[]

# Running the train/test and prediction 10 times
for itr in range(10):
    # splitting train/test data with random shuffles in evry iteration
    X_train, X_test, y_train, y_test = train_test_split(data, data["fire"],test_size=0.33333)

    # Dropping fire column
    X_train=np.array(X_train.drop(labels=["fire"],axis=1))
    y_train=np.array([[i] for i in y_train.tolist()])

    y_test=np.array([[i] for i in y_test.tolist()])
    X_test=np.array(X_test.drop(labels=["fire"],axis=1))

    # Standarizing data, this makes mean as 0 and standard deviation as 1.
    scl_x=StandardScaler()
    normalised_x=scl_x.fit_transform(X_train)
    normalised_y=scl_x.transform(X_test)

    # Initializing neuralnetwork object with training data at 1800 epochs
    nn = NeuralNet(normalised_x,y_train)
    # Adjust this value for changing epochs
    epochs=1800
    nn.fit_nn(epochs)

    # List for storing classified data
    predicted=[]
    for i,j in zip(normalised_y,y_test):
        predicted.append(np.round(nn.predict_nn(i)).tolist())
     # List for storing F1 score of the model
    own.append(f1_score(predicted,y_test))

    # Exporting predicted files.
    file=pd.DataFrame()
    file['predicted']=predicted
    file['actual']=y_test
    file['predicted']=file['predicted'].apply(lambda x: int(np.array(x)))
    file['Correct_Classification']=file['predicted']==file['actual']
    file.to_csv('Predictions_{}.csv'.format(itr+1),index=False)

    # Pickling the weights - Uncomment the below line for pickling weights
    #pk.dump(nn,open("model_{}.pkl".format(itr+1), "wb"))

    #-------------------------------------------------------------------------------#

    # Implementing SKlearn Multi Layer Perceptron model with same number of neurons as above.

    mlp = MLPClassifier(hidden_layer_sizes=(6,6), activation='tanh', max_iter=epochs)
    mlp.fit(normalised_x,np.ravel(y_train))

    predict_train = mlp.predict(normalised_x)
    predict_test = mlp.predict(normalised_y)
    skf1.append(f1_score(predict_test,y_test))

    # Printing F1 scores for both the models.
```

```
    print("Itr:",itr+1)
    print("F1 score (Own Model)",":",own[-1])
    print("\nTrain data prediction F1 score (Sklearn):",f1_score(predict_train,y_train))
    print("Test data prediciton F1 score (sklearn):",skf1[-1])
    print()

# Printing data details
print("\nTrained with:", len(normalised_x)*len(y_train)/len(normalised_x)," samples")
print("Tested with :",len(normalised_y)*len(y_test)/len(normalised_y)," samples")
print('Total data size:',len(data))

# Visualization

# Dataframe containing F1 Scores of both the models and the difference in their F1 scores
df=pd.DataFrame()
df['own_model']=own
df['sklearn_model']=skf1
df['difference']=df['own_model']-df['sklearn_model']

#df.head()




# Plotting the scores for each shuffle (10 times)
plt.xlabel("Iterations")
plt.ylabel("F1 Score")
plt.plot(df['own_model'],label="Own Model")
plt.plot(df['sklearn_model'],label="Sklearn Model")
plt.plot([df['own_model'].mean() for i in range(10)], label='Mean F1: Own model')
plt.plot([df['sklearn_model'].mean() for i in range(10)], label='Mean F1: Sklearn')

plt.legend()




# Plotting the difference. For values <0, Sklearn had better classification accuracy.
plt.plot(df['difference'],label="Own model vs Sklearn")
plt.plot([0 for i in range(10)])
plt.legend()


# calculating loss
loss=np.sum(nn.loss,axis=1)*-1

#Loss/Cost Function

# Plotting loss for each Epoch
plt.plot(loss,label="Mean Sqrd Logarithmic Error")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
```