Group 4 Assignment 3

Summary

This paper tells about the **Feed Forward MultiLayer Perceptron Model** which will be used for **diagnosing Leishmaniasis using Cognitive Computing**. The genetic dataset of leishmaniasis is collected from the Gene Expression Omnibus database. The dataset has 33 different human samples with each sample having 45033 feature.80% of the data was used for training and left 20% for testing. It uses a feed-forward ANN with MLP as the bio-mimic neural network to develop a model for diagnosis where nodes of the ANN are used as the neurons of the human brain. The model has 4 hidden layers with 45 neurons in each layer. The learning rate of the model is 0.001 and the training is repeated for 500 epochs. The model is trained, checked for its accuracy, loss function and mean square error of the model at each iteration. The Final Accuracy, Loss Function value and Mean Square Error is **85.71%, 1.94** and **219.84** respectively after 500 epoch

Strengths:

1. Using an ANN helps capture the non-linear association of the multidimensional data with the output, as opposed to a regression model.
2. The use of back propagation in the network has decreased the loss function to the prescribed threshold value by adjusting the weights of the neuron connections.
3. The Gradient Descent Optimization used thus, helps reduce the error.
4. The randomization of weight input with the hidden layers and usage of hidden matrix makes a stable model, without affecting its convergence.

Weaknesses:

1. No pre-processing is done. A lot of columns are redundant, for eg.- around 19k features were just zero values for all samples. No feature selection/reduction techniques used.
2. Inconsistencies present in the types of each feature column. Numerical columns labelled as string.
3. Features are not scaled and normalized before processing.
4. Very high dimensionality and very low number of samples leading to overfitting.
5. Architecture is not suitable for the dataset used. We should use a deeper net, and average over multiple dropouts to get appropriate weights as described in this paper.
6. No hyperparameter tuning is done---learning rate, number of epochs.
7. Should perform cross-validation for the measure of accuracy. Basing on one split is highly inappropriate.

Discrepancy:

The paper’s implementation doesn’t account for normalizing the dataset, even though it claims to do so in theory. They specify the cost function to be calculated “by the prediction of deviations in the mean squared error between the actual and predicted output value” but in the code, they have used the “softmax_cross_entropy_with_logits” function as the cost function.

Methodology

API of the code

Class Neuron: This is the base class on which the neural network is built. The class has the following attributes:

1. Number of inputs or Input dimension.
2. Weights and bias
3. Activation function - Currently implemented: Sigmoid and ReLU. these can be extended by adding other.
4. *propagate()* :

    Parameters : input = 1-d array

    Returns : output = ‘float’

   Function which calculates the weighted sum of ***activation_fn((weights*inputs) + bias)***

Class Layer: This class forms a layer of Neuron objects. It has following functionality:

1. Number of neurons (n)
2. Input dimension (prev)
3. Activation function common to all neurons. (activation_fn)
4. *weight_mat():*

    Parameters : w = 2-d matrix

    Returns : None

   Takes a [n x prev] dimension matrix which assigns weights of [prev X1] array to each of the of the n neurons in the layer.
5. *assign_b():*

    Parameters : b = 1-d array

Returns : None

Takes a [n x 1] array and assigns bias to each of the n neurons.

6. *generate():*

    Parameters : input = 1-d array

    Returns : output = 1-d array

    Takes input as [prev x 1] array which is given as input to the n neurons and generates a [n x1] output.

Class Neural Network: This class is the main class which creates the whole Neural Network with the following functionality:

1. no_input: Dimension of the input layer. The default is set to 1
2. no_output: Number of outputs required. The default is set to 1
3. no_hidden_layers: Number of hidden layers required in the network. The default is set to 1.
4. hidden_dim: Takes a list of ‘int’ with the dimensions of the corresponding hidden layer. Default [40]
5. batch_siz: The batch size for weight update. If batch_size is 10 then, weight update occurs after every 10 inputs. The default is set to 1 i.e. will update weight after every input tuple.
6. h_act: Takes a list of strings containing activation functions for each corresponding hidden layer. The default is set to [‘relu’].
7. o_act: Activation function for the output layer. Default value set to ‘sig’.
8. learning_rate: Sets the learning rate for the weight update rule. Default set to 0.01.
9. Random_state: For reproducible results.

Functions:

1. *fit()* :

    Parameters: X, Y

    Returns: dict() type object of {input no. : mean rmse}

Takes the X as input matrix, Y as an array of the target output and no_epochs as the number of Epochs for which to train the network. Returns a dict() object for the rmse for each epoch.

2. *predict():*

    Parameters: X

    Returns : array_like_object of predicted value

   Takes X as the input matrix. Produces output for each input tuple based on the trained model and return an array of output values corresponding to each tuple.
3. *predict_class() :*

    Parameters: X

    Returns: array_like_object of predicted class

   Takes X as the input matrix. Produces output for each input tuple based on the trained model and return an array of output class corresponding to each tuple. The threshold is set to 0.5.

Description:

- Pre-processing:

  The dataset has to be pre-processed before it can go through any learning. Since the neural network takes only numerical inputs, the dataset needs to be appropriately processed.

1) Any feature containing non-numeric data type needs to be converted to numeric:

   If categorical then either using LabelEncoder or OneHotEncoding.
2) Handling NULL/Nan values.
   a) The tuples containing Null values can be dropped altogether.
   b) The Null values can be replaced by some central value (mode in case of the categorical/ median in case of continuous) present in that column.
3) Feature engineering: Create new features derived from existing features.

4) Feature reduction: Remove the features which are redundant or have little influence on the target variable. Any two features which are highly correlated can be reduced to one. Any feature which is known to be not causing any effect on the target variable from the domain knowledge is removed.
5) Data normalization. Data can be either scaled to a range of [0,1] using MinMaxScaling or it can be normalized using more sophisticated techniques. This makes the data homogenous so that any one particular feature does not high influence on the value of the target variable.

Model Training:

The neural network is initialised using the NeuralNetwork class with suitable parameters.
The preprocessed dataset is divided into a trai-validation split. The X_train dataset and Y_train (target variable) are given as parameters to the fit() function with no_epochs as the parameter.
The fit function does the following for each tuple:

- The row of X_train is given as the input to the neural_network in a FeedForward step.
- Backpropagation Step:
    a. Calculate Error for Output layer:

    $$\delta^L = \nabla_a C \odot \sigma'(z^L).$$

    Here C is the cost function used. The choice of cost function depends on the activation function of the layer. If it is ReLU then the cost function used is Mean Squared Error. If it is sigmoid then a logit type of cost function is used. Delta of C is with respect to output of the layer. The σ′ derivative of the cost function.

b. Calculate the error terms for Inner / Hidden layers:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Where w and δ are the weight matrix and errors of l+1th layer and z is for the layer l.

c. Calculate the update term for bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

d. Calculate the weight update term for each layer:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

These weights and the bias term calculated above are then multiplied with the learning rate and added to the weight and the bias terms respectively for each layer. The computation is carried out for a batch_size of the inputs and the update happens after calculating the mean of the computed results.
To take regularization into account, the regularization term is added. The L2 regularization is implemented by multiplying lambda with the weights of the layer and added to the delta w calculated in the backpropagation step.

Once the model is trained, the validation set is used as a parameter to the predict_class() function which predicts the class of the tuples which are finally matched against the validation target values to calculate the accuracy score.

Correspondingly the graph of RMSE is plotted against the number of input tuples.

The Dataset:

This dataset comes from a proof-of-concept study published in 1999 by Golub et al. It showed how new cases of cancer could be classified by gene expression monitoring (via DNA microarray) and thereby provided a general approach for identifying new cancer classes and assigning tumors to known classes. These data were used to classify patients with acute myeloid leukemia (AML) and acute lymphoblastic leukemia (ALL). We have selected this particular dataset because it is similar to the one used in the paper. It is a “high dimensionality, low sample space” dataset having 72 total number of samples and 7129 features. We have chosen a similar dataset so as to address the issues that training such a dataset imposes, like that of overfitting and high variance. We have currently incorporated measures such as:-

1) taking a deeper neural network with each layer having (two/third) the number of dimensions as the original input features
2) Included a regularization term in the cost function to reduce overfitting

Currently, we are trying to address this particular dataset by trying to average over multiple dropouts to get appropriate weights as described in this paper.

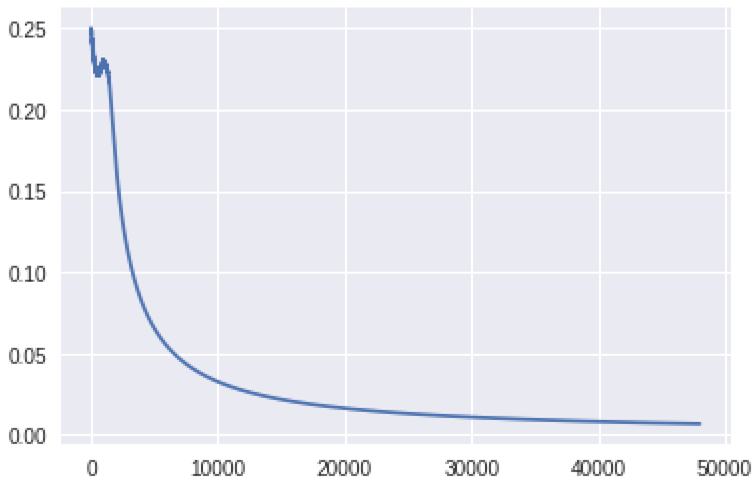
Results

The dataset was split into a train_test_split of 0.33 thus we got 48 training examples and 24 test examples. Data was normalized and regularization was performed for the weights by the methodology described above.

The results obtained gave an accuracy score of 83.33%. These results correspond to a neural network with :

2 Hidden layers of 50 neurons each with activation function as ‘ReLU’.
1 output layer with 1 neuron with activation function as ‘Sigmoid’.
Learning rate set to 0.01.
Lamba(Regularization parameter) set to 0.01.
Number of epochs = 1000.

The graph of RMSE vs input of epochs is as follows.



As it can be clearly seen the error is steadily decreasing as number of examples it is trained on increases until it reaches a steady state. The initial kink in the graph is because of the overfitting. But since the regularization is implemented, the error reduces back and avoids overfitting with the small dataset.

Hence it can be safely said that the model works well with a considerable accuracy rate of 83.33%.