**BPNN**

Required Python libraries : Pandas, Numpy, Scikit-learn, Tensorflow, Matplotlib

You will also need to have software installed to run and execute the Jupyter notebook.

Template code is provided in the 4A-BPNN.ipynb notebook file.

**DATA LOADING AND PREPROCESSING**

First, training data that is stored as .csv is loaded using pandas. Next, labels are stored as a column vector of shape (100, 1). 100 being the number of training samples and 1 is the label for each sample that is either 0 or 1.

0 for Simple Substitution and 1 for Vignere.

After that, training features are created using CountVectorizer function of Scikit-learn that converts each training sample into a feature vector of length 500.

We have also used train\_test\_split() to split our data into training and validating data, and our data is ready to be loaded into neural network model.

Shape of training features: (500, 90)

Shape pf training labels: (1, 90)

Shape of validation features: (500, 10)

Shape of validation labels: (1, 10)

**CREATING MODEL**

Steps involved in creation of model are as follows:

1. Initializing parameters: Our model is a 3 layer(2 hidden and 1 output) neural network, hence W1, b1 are weights and biases corresponding to layer 1, W2, b2 for layer 2 and W3,b3 for layer 3. Weights are initialized using tf.contrib.layers.xavier\_initializer() and biases are initialized with zeros.
2. Forward propagation: Z is the mapping function that is calculated as Z = W.X + b and A is the output of relu activation function which can be calculated as A = max(0, Z).

Note that, Z for the output layer i.e. Z3 is not passed through relu activation since for output we are using sigmoid activation function and hence function\_propagation() is returning Z3.

1. Compute cost: Cross-entropy loss is calculated as −(ylog(p)+(1−y)log(1−p)) where y is the true label and p is the predicted label. We have used tensorflow function to do the same.

Since, model is created using Tensorflow we need a function that compiles everything discussed till now into a single computational graph and create the session to run the model.

Hence, model(parameters\_shape, X\_train, Y\_train, learning\_rate = 0.001, num\_epochs = 50) is the function that does the same.

Parameters\_shape is a dictionary that stores the shape of all the parameters used and is defined using function shape\_of\_parameters().

X\_train and Y\_train are Training features and labels respectively.

Learning\_rate is a hyper parameter that is used in Optimizer function(Here Adam optimizer).

Num\_epochs is the number of forward\_passes through the whole training set.

The function returns the model\_cost and set of final parameters.

**MAKING PREDICTIONS**

We used model1 for making our predictions of validation and test data. (reason and shape of parameters for model1 are discussed momentarily). 3 utility functions are created for making predictions:

1. forward\_propagation\_for\_predict(X, parameters): This function calculates Z3 for the input X using trained parameters of model1.
2. predict(X, parameters): This function passes Z3 through sigmoid activation function to make the prediction.
3. print\_prediction(predictions, label, data\_type): Function for printing predicted value and actual value of label.

**COMPARISON OF THREE MODELS**

1. Model1: shape of parameters:

W1: (128, 500), b1: (128, 1),

W2: (64, 128), b2: (64,1),

W3: (1, 64), b3: (1, 1)

Cost after 45 epochs: 0.007102

1. Model2: shape of parameters:

W1: (64, 500), b1: (64, 1),

W2: (16, 64), b2: (16,1),

W3: (1, 16), b3: (1, 1)

Cost after 45 epochs: 0.060803

1. Model2: shape of parameters:

W1: (32, 500), b1: (32, 1),

W2: (16, 32), b2: (16,1),

W3: (1, 16), b3: (1, 1)

Cost after 45 epochs: 0.082088

Since cost in model1 is much lower as compared to model2 and model3 hence we used model1 for making our predictions for which we managed to get test accuracy of 95%.