Name Student : Raja Naseer Ahmed Khan.

Project : Language Recognition Training Using Neural Network Encog API.

Module : B.Sc. (Hons) Artificial Intelligence.

<Note>

- **Please use the case sensitive letters mentioned in Menu, while selecting options i.e. F, Y,N not f,y,n. Also, the test language detection file needs full path. Willi dataset is at ./ so, can be put within the folder with jar file.**
- **Also, Train the model first, as it generates the files needed for network to perform it's tasks.**

About the Neural Network

- Neural Networks are used to train machines to and have an artificial brain of their own.
- In order to train a NN to perform any task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (EW). In other words, it must calculate how the error changes as each weight is increased or decreased slightly.

About Ngram

An n-gram is a sequence of n characters within a given piece of text.

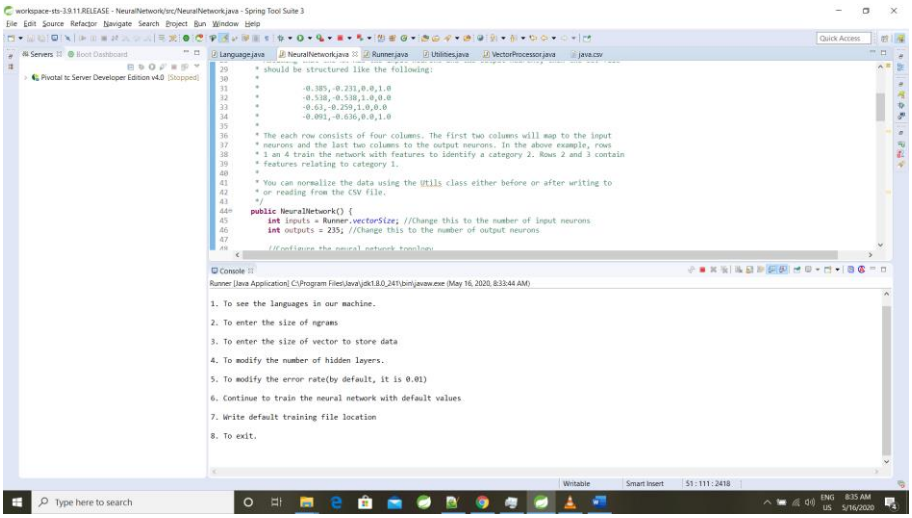
i.e. "abcdef"

- **3-grams:** abc bcd cde def
- **4-grams:** abcd bcde cdef

all you have to do is iterate through the string with a window size n. In each iteration, the new substring, or n-gram, will be added to the ngrams list.

Construction of our Project:

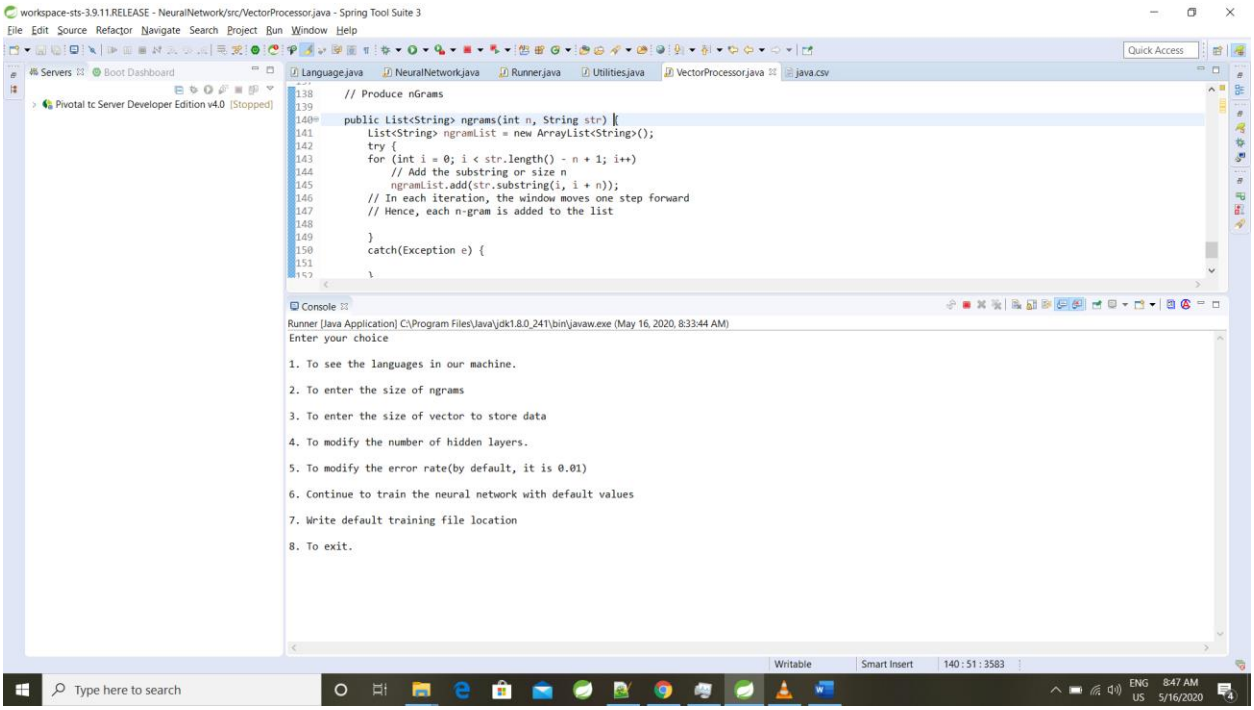
- Initially we have our main screen, where we have multiple options for the user to select like have a user input for vector size, length of each ngram , show all languages, training file location, string for language detection, file for language detection.



- Once a user inputs the value or goes forward with default values, we call “go()” method of VectorProcessor Class.

- Here in Vector Processor class, we have an array of type double called vector in which we are going to store ngrams . This vector[] will contain our input elements. Bigger the size of vector , longer it will take to train though precision can become better.

- We read each line of input data and split it based on “@” character with first part being the training data and another part being the language.



- The above figure shows the method ngrams where we split create our ngram list based on the size of ngram.

- After splitting the data we use “hashcode %n” on each vector value to compute the vector index for each ngram.

- Since Neural networks are designed to accept floating-point numbers as their input , we needed to normalize the vector values in range of either 0 to 1 or -1 to -1 for better efficacy. **Here we have normalized the value between 0 to 1 to keep in positive values**

- The vector array we have is an input to neural network and for output values we have columns from (vector.length()+1) till additional 235(which is the size of recognizable languages).

- We have all output values as 0 except for the language which we get after splitting each of text from input line.

- Then we save the csv file as java.csv with normalized training data for neural network.

- After this we call constructor of NeuralNetwork class, here we will build our own neural network.

- We need to have number of input neurons, number of output neurons, number of hidden layers and their number of neurons, Activation Function for each layer.

- For this particular network topology , if we increase the number of hidden layer neurons we experienced that the network took more time to train and same thing happened when we increased the size of input vector . So we started it with 200 input neurons and 1 hidden layer with 200 neurons and increased further using for loop. As we know that number of neurons in hidden layer should always be less than the twice of input neurons for best case results. After some saturation point I then increase the size of vector and again started with 200 neurons in hidden layer.

- Number of input neurons will be always equal to the size of our vector and output will be length of our defined languages for which neural network can be trained .i.e 235 in our case.

- **Activation functions are used to scale the output from a layer**. Here we have used ActivationSoftMax for output layer and `default hyperbolic tangent for input layer.`

    1. **ActivationSoftMax** is used to compute probability distribution from a vector of real numbers, i.e positive values. This function produces an output in range of values between 0 and 1. The sum of probabilities being equal to 1. This is generally used for output layers and after doing lot of research and testing, I implemented it for output layer because we have output in range 0 to 1.

    2. **`ActivationReLU,`** `Rectifies Linear (ReLu) is a faster learning Activation function.` It offers the better performance and generalization in deep learning compared to the Sigmoid and tanh activation function. The ReLU presents a almost nearly linear function and thus preserves the properties of linear models which made them easy to optimize, with gradient-descent method. These properties combined with the fact that this processed data faster than sigmoid or tanh activation Function.

    3. **Hyperbolic tangent function,** This function is a very common choice for feedforwardand simple recurrent neural networks. The hyperbolic tangent function has a derivative so it can be used with propagation training. The hyperbolic tangent function is the most commonly used activation function due to its efficacy to work with negative and positive both type of numbers.

- There are many other like ActivationBiPolar but it made the learning very slow and this could have applied to the output layer if we had value either -1 or +1.

- **ROLE OF HIDDEN LAYER**
    1. Generally, two or less than two should be the number of hidden layers in any architecture unless it's a very complex one. Here we initially tried with no hidden layer but had results come to saturation to around 50.
    2. So I increased the size to one and took results and the as we increased the number of neurons in hidden layer the efficiency increased. Below are some values computed.

Ngram size	**vectorsize**	**hidden layer neurons range**	**Efficiency in %(Normally)**
4	**200**	**1 to 200**	**10-12**
4	**250**	**1 to 200**	**10-13**
4	**300**	**1 to 200**	**10-15**
4	**1000**	**1 to 200**	**15-20**

Below the is the data for ngram = 4 vector size = 8500 and hidden layer between 1-125

No of neuron in hidden layer	**Time taken(seconds)**	**Efficiency**
1	3	0.0
7	7	2.06
10	9	8.56
25	35	18
49	86	36
103	162	57.7
121	**195**	**70.37**
124	222	67

Snapshot for above configuration output

```
67-----Testing complte     51.94650310929381      in     137    seconds
2
70-----Testing complte     48.23238776727149      in     147    seconds
2
73-----Testing complte     45.84717607973422      in     147    seconds
2
76-----Testing complte     42.69528920691712      in     152    seconds
2
79-----Testing complte     56.998040718970955      in     158    seconds
2
82-----Testing complte     55.80543487520232      in     163    seconds
2
85-----Testing complte     55.43913450890196      in     154    seconds
2
88-----Testing complte     55.41357866939263      in     153    seconds
2
91-----Testing complte     47.80645710878269      in     168    seconds
2
94-----Testing complte     61.15512394582162      in     176    seconds
2
97-----Testing complte     59.71547832012948      in     195    seconds
2
100-----Testing complte     58.02027429934407      in     166    seconds
2
103-----Testing complte     57.534713348666834      in     162    seconds
2
106-----Testing complte     64.90331374052303      in     171    seconds
2
109-----Testing complte     67.73149331288866      in     178    seconds
2
112-----Testing complte     66.75185279836442      in     181    seconds
2
115-----Testing complte     57.30471079308289      in     184    seconds
2
118-----Testing complte     62.5436578924951      in     195    seconds
2
121-----Testing complte     70.37226339551921      in     195    seconds
2
124-----Testing complte     67.34815572024874      in     222    seconds
```

Snapshot for configuration with Ngram size 2 vector size = 800 and number of neurons in hidden layer being 500-600

```
Training finished .....

Saving the network

Network Saved....

Starting network testing.....
538-----Testing complte    53.88874691200273     in    153   second
Starting training.......

Training finished .....

Saving the network

Network Saved....

Starting network testing.....
544-----Testing complte    59.1788056904336     in    159   seconds
Starting training.......

Training finished .....

Saving the network

Network Saved....

Starting network testing.....
550-----Testing complte    46.613851265014056     in    164   secon
Starting training.......

Training finished .....

Saving the network

Network Saved....

Starting network testing.....
556-----Testing complte    47.40608228980322     in    159   second
Starting training.......
```

- Using these, we created our network, now the next step is to feed training data to our network i.e data from java.csv file, So we read the file and store it into MLDataSet object called trainingSet.

- Now we use `ResilientPropagation algorithm` training to train our network and let the network change weights until it has error less than 0.0015 or as defined by user.

- **Resilient propagation,**is one of the fastest training algorithms available. The RPROP algorithm just refers to the direction of the gradient. It is a supervised learning method. It works similarly to back propagation, except that the weight updates is done in a different manner.

- After this we cross validate our network for its efficacy by using procedure in which by

    1. We traverse the row of data that we have in csv and we calculate the number of correct evaluations by, determining the column number of the ideal set,(i.e the input set values used to train network) which has value 1 and see if that particular column number in output set(i.e value provided by network) is the highest among the output array, if both the are same then we can say that the output has been predicted correctly.
    2. In simple words I am just checking whether the neural network provides the highest value in that particular column number for which the data in ideal set column number is 1. For example, If ideal data set for any row has 1 for column number 50 , and if the output data array has the highest value for that column number 50 then we can say network predicted it correct.

- Next process is to predict the language , So here we provide an input string and we make an array of vector similar to the one we made when we provided the training set. And then similar procedure is used to find the highest column index in output array . When we find the index with highest value, we try to get that particular index Language from our enum .

- **We also calculated**

    1. True Positive: Correct Language identified.(correctly identified)
    2. False Positive : Other language identified as the asked language(incorrectly identified)
    3. True Negative: Other language identified as false or 0.(correctly rejected)
    4. False Negative : Asked language identified as other language . (incorrectly rejected)

EXTRAS:

Tried to make the ngrams, Vector Size flexible by asking user to input and override the default values of those.

Few other extra menus in addition, i.e. checking all languages are feed into system using Wilis Dataset, which would be checked rate of accuracy against the data file we provide.

REFERENCES

- J. Clark, I. Koprinska, and J. Poon, “A neural network based approach to automated email classification,” In Proceedings of IEEE/WIC international conference on web intelligence, Halifax, Canada, 2003, pp. 702–705.

- https://www.researchgate.net/publication/272294880_Programming_Neural_Networks_with_Encog3_in_Java