Natural Language Processing

Subhasish Basak (MDS201803)

Assignment 2

March 6, 2020

The link to my **Colab Notebook** is as follows: Click to view Colab Notebook

Here we start with a Simple Neural Network framework with only 1 hidden layers. Our Neural network has the following configuration :

- Input Layer
  - Contains One-Hot vector representations of 5 letter English words
  - **Number of nodes** : 26
- Hidden Layer
  - Only one hidden layer
  - **Number of nodes** : 10
- Output Layer
  - Contains the One hot vector representation of 128 words in our vocabulary
  - **Number of nodes** : 128

Our aim is to recognize the words, i.e. given a 26×1 input vector (representing a 5 letter English word) our Neural Network should output the One hot vector (a 128×1 vector representing a single word in a vocabulary of 128 English word) corresponding to that word in the vocabulary.

Let us consider the following example :

- Consider the word **about**. The One hot vector corresponding the word **about** is,

```
In [1]:  one_hot_encode("about")
```

```
array([0.2, 0.2, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ,
       0. , 0.2, 0. , 0. , 0. , 0. , 0.2, 0.2, 0. , 0. , 0. , 0. , 0. ])
```

  The One hot vector representation of the word **about** is obtained by taking the average the One hot vector encoding of all the alphabets present in the word.

- Also the word **about** occurs in our vocabulary at the 0-th position. The One hot vector encoding representing the word in our vocabulary is

```
In [2]:  y[vocabulary.index("about")]
```

```
array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Given the 26×1 input one hot vector our Neural Network should be able to output the 128×1 one hot vector representing the position of the word in our vocabulary. The network should be trained in such a way that in case of mis-spelled words like **abouu** it should predict the correct word i.e. **about**.

1 Building the Neural Network

The Neural Network makes prediction using *Forward Propagation*, which is a set of matrix multiplication with weight matrices and adding bias vectors. Our input is a matrix of size 128×26 and output is a 128×128 matrix. In Our Neural Network with one hidden layer, we have used **tanh()** activation function in the transition from the first layer to the hidden layer. The hidden layer contains 10 nodes. We used **Softmax** activation in the final output layer. Our model uses the following equations for *Forward propagation*:

$$z_1 = X * W_1 + b_1 \tag{1}$$

$$a_1 = tanh(z_1) \tag{2}$$

$$z_2 = a_1 * W_2 + b_2 \tag{3}$$

$$a_2 = \hat{y} = softmax(z_2) \tag{4}$$

Now to estimate and optimize the weights and biases we use *Batch Gradient descent*. For that we define a loss function which is calculated based on the predicted and true output values. The loss function is then minimized to to estimate the weights and biases. Here we have used the ***Categorical Cross Entropy*** as the loss function, given by the following equation:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{n \in N} \sum_{i \in C} y_{n,i} \log(\hat{y_{n,i}}) \tag{5}$$

Where $y_{n,i}$ is the n-th observed belonging to the i-th class (here we have a total of 128 classes, since the training data consists of 128 words). $\hat{y_{n,i}}$ is the corresponding predicted output.
Next we use *Back-propagation* to update the weight and bias parameters. We compute the partial derivatives of the loss function **L** w.r.t weights and biases i.e. $\frac{\delta L}{\delta W_1}$, $\frac{\delta L}{\delta W_2}$, $\frac{\delta L}{\delta b_1}$ and $\frac{\delta L}{\delta b_2}$ as follows.

$$\delta_3 = \hat{y} - y \tag{6}$$

$$\delta_2 = (1 - tanh^2 z_1) * \delta_3 W_2^T \tag{7}$$

$$\frac{\delta L}{\delta W_2} = a_1^T \delta_3 \tag{8}$$

$$\frac{\delta L}{\delta b_2} = \delta_3 \tag{9}$$

$$\frac{\delta L}{\delta W_1} = x^T \delta_2 \tag{10}$$

$$\frac{\delta L}{\delta b_1} = \delta_2 \tag{11}$$

After building the Neural Network we train it with the 128 words in our vocabulary with 10 nodes in the hidden layer and for 500 iterations. We plot the training accuracies against the iteration sizes and from the graph we try to identify the best number of iterations. The training accuracy achieved after this was 97.65%.

```
In [3]:    model = Neural_network(10, 500)
```

```
In [4]:    accuracy(model, train = True)
```

```
input word : being, predicted word : begin, predicted index : 11, actual index : 12
input word : field, predicted word : filed, predicted index : 52, actual index : 49
input word : ideas, predicted word : ended, predicted index : 39, actual index : 73
0.9765625
```

From the above code snippet we can also see the mis-classified words in the training data set.

For the word **being**, the predicted word according to our model is **begin**. A plausible explanation to this may be as follows : The word **begin** and **being** both are permutations of the same alphabets and thus the input one hot vectors are the same for both the words. Since our model is returning the ***argmax*** of the softmax layer, so if there are multiple maximas it can return any of the values. Hence for both the word **begin** and **being** it gives the same prediction as **begin**.
Similar explanations can be provided for the word **filed** and **field**.
For the word **ideas** it predicts **ended**, which can be rectified by increasing the number of iterations from 500 to 1000.

```
In [5]:  model = Neural_network(10, 1000)
```

```
In [6]:  accuracy(model, train = True)
```

```
input word : begin, predicted word : being, predicted index : 12, actual index : 11
input word : field, predicted word : filed, predicted index : 52, actual index : 49

0.984375
```

In that case the training accuracy also increases to 98.43%
Once the model is trained used the training data we test the model on mis-spelled words. Here we have used the following dictionary which contains the mis-spelled words along with their expected predictions.

```
In [7]:  accuracy(model, train = False, test = test_dict)
```

```
input word : ajent, predicted word : paint, expected word : agent
input word : abouu, predicted word : about, expected word : about, CORRECT PREDICTION
input word : indes, predicted word : field, expected word : index
input word : abain, predicted word : again, expected word : again, CORRECT PREDICTION
input word : layer, predicted word : layer, expected word : later
input word : aappy, predicted word : happy, expected word : happy, CORRECT PREDICTION
input word : march, predicted word : match, expected word : match, CORRECT PREDICTION
input word : majik, predicted word : again, expected word : magic
input word : layre, predicted word : layer, expected word : layer, CORRECT PREDICTION
input word : gless, predicted word : girls, expected word : glass
input word : billo, predicted word : bills, expected word : bills, CORRECT PREDICTION
input word : other, predicted word : other, expected word : othre
input word : juict, predicted word : juice, expected word : juice, CORRECT PREDICTION
input word : feelz, predicted word : feels, expected word : feels, CORRECT PREDICTION
input word : knofe, predicted word : knife, expected word : knife, CORRECT PREDICTION

0.6
```

Thus we can see that for the test data set with 15 mis-spelled words our model can predict the correct word for 9 words, yielding the test accuracy of 60%

2 Observations

Our model learns the spellings of certain words in the vocabulary, so that even if some words are mis-spelled the model tries to predict the right word the user might have intended to type.
One drawback of using this Neural Network model to recognize words is the way one hot encoding is used. For example words like **beign** and **begin** are made up of the same alphabets with different permutations. These have the SAME One hot vectors, as a result the NN model tries to map the same One hot vector to different outputs.
In other words One hot vector representation of the input words do not take into consideration the sequence of the alphabets. This type of problem can be solved if sequential inputs are possible to the model. We can use **RNN**.