Natural Language Processing

Chennai Mathematical Institute

Notebook Link

Task

Here the task was to design and implement a neural network to recognize 128 5-letter English words with one hidden layer. The model will try to predict the corresponding correct word when a misspelled word is provided as input. Here the main aim is to understand that the basic neural network model with basic input will not be able to learn much about the words and will fail for simple cases when we expect it to work. Though the result will highly depend on the test examples.

Model

Input:

Encoded word. Here we have 128 five length words in total. Each word is encoded based on the characters. First all the characters are converted to lower case. So we have 26 letters. Now we can create a one hot vector corresponding to each letter. Now for a given word, we just take the average of the one hot vectors of the letters present in the word.

Here the sequence of the letters doesn't matter. If two words have same letters with same frequency then irrespective of the position of the letters in the words both the words will have the same encoded vector. For e.g. - "heart" and "earth" will have the same encoding.

Output:

Output will be of 128 dimension as we have 128 words and we want to predict any given input as one of those words. Here we use the one hot encoding for the words.

So 26 dimensional input will be provided as input and an output of 128 dimension will be obtained which is necessarily a one hot vector for one of the 128 words.

Design of the network

One input layer. (With 26 nodes as input is a 26 dimensional vector)
One hidden layer. (with 10 nodes) — The no of nodes in hidden layer depends on our choice.
One output layer. (With 128 nodes as output is a 128 dimensional vector)

Activation Function:

We can choose our activation function. Sigmoid, softmax, tanh, Relu are some activation functions. Here,
In hidden layer: tanh
In output layer: Softmax

Cost Function: Categorical Cross-entropy

Procedure

Step 1: First we need to initialize the parameters based on the input, output and hidden layer dimension. We have two weight matrix and two bias vectors.
First weight matrix corresponding to the input(26 nodes) and hidden layer(10 nodes) is of size 26 x 10. — Effectively 260 parameters.

First bias vector corresponding to the hidden layer is a vector with 10 dimension. — Effectively 10 parameters.
Second weight matrix corresponding to the hidden(10 nodes) and output layer(128 nodes) is of size 10 x 128. — Effectively 1280 parameters.
Second bias vector corresponding to the output layer is a vector with 128 dimension. — Effectively 128 parameters.

Step 2: We calculate the cost or loss by forward propagation.
We got the predicted probabilities corresponding to each class(here word) and we have the actual values. So we calculate the categorical cross entropy loss.

Step 3: We calculate the gradients of the cost function w.r.t the parameters in backward propagation. Here we use the chain rule to find the exact form of gradients and we just use those closed form solutions for the gradients.

Step 4: We update the parameters based on the gradients and learning rate. (Gradient Descent algorithm). Here stochastic gradient descent is used and total no of epochs is 2000

Step 5: Based on the learned parameters we predict the output for a given test word.

Step 6: We check the test accuracy for a set of given test examples. If the predicted word for a given misspelled word matches the expected output then it contribute to increase in accuracy.

Mathematical Results and Formulas

Cost/Loss Function:

$$L(y,p) = -\sum_{i \in C} y_i \log(p_i)$$

This loss function calculates the loss for one input and C categories. Here p is the predicted values for given input and y is the actual output.

We have a input vector X. We can pass input as matrix but here in my implementation using Stochastic GD, X is a vector. Two weight matrices, W_1 and W_2. Two bias vectors, b_1 and b_2.

Forward Propagation:

$$\begin{aligned} Z_1 &= XW_1 + b_1 \\ A_1 &= \tanh(Z_1) \\ Z_2 &= A_1W_2 + b_2 \\ A_2 &= \text{softmax}(Z_2) \end{aligned}$$

Z_1 is the output in the hidden layer and A_1 is the output after applying tanh activation to it.Z_2 is the output in the output layer and A_2 is the output after applying sigmoid activation to it.A_2 is our predicted output. Here we treat X as a row input vector. So Z_1, Z_2, A_1, A_2 are also row vectors.

Backward Propagation:

$$\begin{aligned} dZ_2 &= A_2 - y \\ dZ_1 &= (1 - \tanh^2 Z_1) \circ dZ_2W_2^T \\ dW_2 &= A_1^T dZ_2 \\ db_2 &= dZ_2 \\ dW_1 &= X^T dZ_1 \\ db_1 &= dZ_1 \end{aligned}$$

dZ_1 is the gradient of the cost function w.r.t Z_1. Similar denotation for dZ_2, dW_1, dW_2, db_1, db_2.

Softmax Activation

For the i th node,

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$$

Results

Training accuracy is around 98%. The model failed in three following cases.

Actual :'heart' Predicted : 'earth'
Actual :'thing' Predicted : 'night'
Actual :'bread' Predicted : 'beard'

If we observe we can easily identify the problem.
"heart" is predicted as "earth". Both these words have same letters with same frequency. So when we pass them as input we actually pass the same encoded vector in each case. Same problem occurs for the other two training examples. So if we have inputs like this then the model will not be able to identify all of them correctly.

Test accuracy also highly depends on the words chosen. For e.g.- If we misspell "green" as "grean" then this model will give output as "anger" because "grean" and "anger" has same input representation.In these cases model fails miserably.
Here I have used 20 examples and accuracy is around 65-70%. The accuracy can be as low as 0% and high as 100% depending on the test data. So With this simple architecture and simple one hot encoded input the model doesn't learn much and will fail for lot of examples even with one change in letter of the word.