E1-246 Natural Language Understanding Assignment-2

Sayambhu Sen
M.Tech (1st Year)
SE (Department of EE)
SR No. 14351
sayambhusen@iisc.ac.in

Abstract

This is a report for the second assignment of Natural Language Understanding. In the first assignment , we used n-gram models to try and generate language models for our corpus and used them to test perplexity on separate test data. Further, our language model which was a generative model was used to generate sentences according the n-gram distributions of our training corpus. This time we are using neural network models to create our language model. In particular, we will be using the Recurrent neural network structure to create our language models which are able to capture sequence information as well.

1 Language Modelling tools

The language model implemented in this assignment is the LSTM Model. We are using 'tensorflow' to implement our language model. The package 're' is the regular expression package which is used to preprocess our data. The 'nltk' package has been mainly used for the tokenization of our data.

2 Introduction

RNN is the basic architecture of neural language model used for generating the language model. While all values shown will be mainly related to using LSTM(Long Short Term Memory) model , it is to be noted that we can use other models like basic RNNs or GRUs(Gated Recurrent units) . Our implementation uses as input a 'one hot encoding' with one input for each word in the vocabulary for our word level language model and one character for each character in our vocabulary for our character model.

3 Neural Language model

For both our Language models that we used, (word level and char level) we are using the same Neural network model for training. This means the inputs will be the one hot input , the size of the vocabulary of each. The outputs for both are also one hot encodings. This is done by making the output a softmax layer. This helps in making the outputs a distribution of the word vocabulary. Hence, during the testing of the model for perplexity, we can directly get the outputs for the future words given the current words as a probability.

The loss function being used here is sparse softmax cross entropy with logits. The optimizer being used is the Adam Optimizer. Also, for the training model we have used learning rate with decay in order to improve the convergence of the function.

Thus, for both the character and word level we have as input a one hot vector $X_{onehotvector} \in R^v$ where v is the size of the vocabulary. This is the embedding layer and multiplyting this vector by W_{embed} forms the input to the First LSTM layer. This LSTM then gives an output to the next layer of the LSTM. The state outputs of each LSTM are fed back into itself by way of the multiplying with $W_{statetostate}$ Finally The output of the final layer is finally multiplied with W_{out} to provide the logits output for the softmax layer. . Therefore,

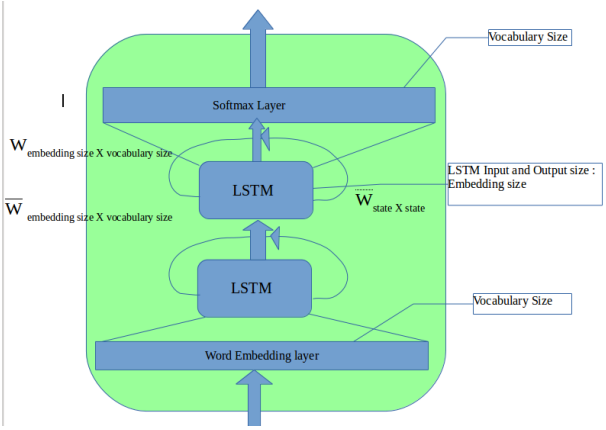
$$X_{LSTMInput1} = W_{embed} * X_{onehotvector}$$

$$Softmax(Logits) = W_{Output} * Y_{LSTM2output} + b$$

$$Softmax_{Output} = Softmax(Logits)$$

$$Loss = \frac{-1}{N} \sum_{i=1}^{n} p_{yi} log(P_{yi}) + (1 - p_{yi}) log(1 - p_{yi}))$$

Figure 1: 2 layer LSTM model with word embeddings.

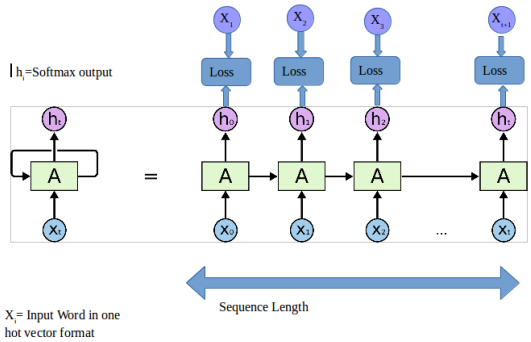


3.1 Training of the model

The model is made in such a way that the training of the model takes place in batches with which each batch has a certain fixed length of sequences. For a default, value we are using a sequence length of 50 and batch size of 50. Thus each batch has 50 sequences. Hence, we are essentially taking a word history(or character history) of 50.

The model of taking in 50 batches each of 50 sequences length is only present during the training of the model and during the perplexity calculation. During the actual sentence generation part we are taking sequence length of one and a batch size of one. Thus, for sentence generation, we are sending in one word/character at a time and giving out one word/character at a time on the trained model.

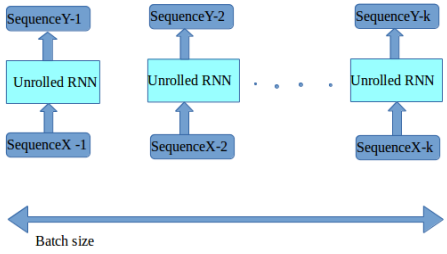
Figure 2: Basic training layers[1]



As can be seen in the figure the above model is what is used to train the model as well as find the perplexity. A sequence of 50 inputs (words or characters) and 50 outputs are used to train the model. Thus, the backpropogation takes place through a sequence of 50 input, output pairs. Essentially, this follows the transducer model of the LSTM.

Now, to speed up training size, since, the sequences in sizes of 50 are independent of each other. We can train a whole batch of sequences one after the other. Thus, we take batch sizes of 50 for our training data.Thus, 50 sequences are trained at each instant.

Figure 3: Batch Training



Here, it is to be noted, that the same model can be used for calculating perplexity. Thus, this helps in faster calculation of perplexity as well.

3.2 Preprocessing

The preprocessing of the text data is a crucial step for language modelling in the neural model. Essentially, the most important step is the inclusion or exclusion of characters in the text corpus. The main issue is that without the preprocessing step, the size of vocabulary grows larger. The larger the vocabulary, the harder it becomes to capture information regarding sequence information.

Hence, what we have done is remove a certain number of characters which do not add anything new to the words or are present in extremely few words. Also, for unknown characters I am replacing a few of those token characters with '$' sign while for the word level model, I am changing all words with frequency count equal to one to be replaced by '<unk >'. The most important step however, is conversion of all characters to the lowercase. This tremendously helps in reducing the vocabulary size and in reducing the training error and achieve faster convergence.

4 Model Implementation

The following steps are taken for the Language modelling:

1. We have first taken our text corpus and divided it into training, test and dev data. We have not shuffled our data (in language modelling, this will be sentences).

2. Now, we have word tokenized the text and divided it into batches of 50 each containing 50 characters/words. This is done for the input and the output.

3. Now we begin the training of the data and run for a few epochs. We fix our vocabulary using the training data and keep it sorted.

4. Using the model that we obtain, we test the perplexity by training the input and then for the corresponding output word, we choose the softmax output as the probability.

Table 1: Small corpus trained model-Character level (One textfile)

$VocabSize$	$testperplexity$	$TrainAccuracy$
46	3.6177	61.2 %

Table 2: Small corpus trained model-Word level (One textfile)

$VocabSize$	$testperplexity$	$TrainAccuracy$
3355	307.618	26 %

Table 3: Large corpus trained model-Character level (Full Gutenberg)

$VocabSize$	$testperplexity$	$TrainAccuracy$
60	6.079654	55 %

Table 4: Large corpus trained model-Character level (Full Gutenberg)

$VocabSize$	$testperplexity$	$TrainAccuracy$
19816	441.1976	22 %

5 Sentence Generation

Using the neural language generation model, we can now generate sentences. Using almost the same procedures as earlier, we generate the sentences. The only difference would be that, earlier we chose the next word according to the distribution of the bigram or trigram probabilities. Now, since our vocabulary was sorted, we find the cumulative distribution of the vocabulary and find a random place to insert using the search sorted function of numpy. This way it is somewhat similar to the beam search methods. Also, while we have a fixed length of number of characters or words to train, we try to stop within the first appearance of '.' or '?' or '!' as long as we have a minimum number of tokens.

Table 5: Equivalent Bigram and Trigram probabilities with full Gutenberg

$Perplexity(Bigram)$	$Perplexity(Trigram)$
173.866	46.844

5.1 Example Sentences

Some example sentences generated are shown below:

1. With small text character level: the had been so surprised and judgment with the most present of one countenance of london,$and the same more in every day, that he saw with elinor, which was the continual unable to be sure, and i have ha

2. With small text word level: the exerted poor again that mr. his to . been ; settled thing gentility have together divide , anything her i who felt not general more any ¡unk¿ mother most not in , spirit handed she staying all and a-year be if my the and in beyond not every , see could much by sight to of a may woman she .

3. With full gutenberg word level:the sun by the water the delight that are old , bear it will hear him

4. With full gutenberg character level: the will rest with such out of the sea unto his little kind had that make up a few all that was the lord like some thing of his boys that come before very boy like unto the boat else upon the days of his

6 Discussion

The character level perplexity was always found to be smaller than the word level perplexity. This is mainly because of the fact that the vocabulary size of character level LSTM was always found to be much smaller than the vocabulary size of word level LSTM model. A smaller vocabulary is very well generalized over a large number of training epochs.

On the other hand, for the word level RNN, the larger the training set, the larger the vocabulary. Hence, in the softmax and the embedding layer there would be a much larger number of parameters to train. This means there must a larger number of examples to train. This may again lead to the introduction of new words. Hence, for the word level model, there can almost never be enough training data. In fact, for our case, we had to remove all words with frequency count of 1 so that our vocabulary size can be drastically.

One important reason why larger vocabulary size leads to rise in perplexity is because the probability distribution over past words/characters is now more widely distributed. This wider distribution leads to very small probabilities of next words/characters even if it is the largest probability among the vocabulary. This is why bigram and trigram probabilities are also lesser than the word level RNN model even though bigram and trigrams have much larger set sizes.

Bigrams and trigrams, on the other hand have very small distributions over other words given previous words. Also, if training and test are the same type of corpus, then they both have similiar distributions during perplexity calculation for their short word histories. But for RNN models which have large history, the prediction of next words is very evenly distributed within the vocabulary.

References

- https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial
- https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767
- https://github.com/sherjilozair/char-rnn-tensorflow