Neural Language Models

Swapnil Gupta
M.Tech, IISC Bangalore
swapnilgupta.229@gmail.com

1 Introduction

Language Models are probabilistic models of word sequences which assign a likelihood score to a sequence of words on the legitimacy of that sentence in any language. Such models are widely used in the applications such as speech recognition, handwriting Recognition, spelling correction and machine translation.

Suppose we have a corpus, which is a set of sentences in any language. We define **V** as the vocabulary of the corpus. Based on these given a sequence of words $(w_1, w_2, ..., w_n)$ we want to estimate

$$P(w_1, w_2, ..., w_n) \text{ such that,}$$
$$\sum_{(w_1,w_2,...,w_n)\epsilon\mathbf{V}} P(w_1, w_2, ..., w_n) = 1$$

where $(w_1, w_2, ..., w_n)\epsilon\mathbf{V}$ implies all possible sentences using the vocabulary.

Using **chain rule**

$$P(w_1, w_2, ..., w_n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

where $w_1^{k-1} = (w_1, w_2, ..., w_{k-1})$ the last $K-1$ word history.

Initial works in the field used **N-gram** models for language generation. But they suffered from a major limitation of capturing a very small and local context/history for each word. This gave rise to what are called **Neural Language Models** which treats Language Modelling as an application of sequence to sequence modelling using **Recurrent Neural Networks**. These models have taken a big leap forward as they allow us to capture large amount of context/history. But these models have their own challenges and the major one of them is curse of dimensionality. Neural Language models like any other deep learning models include a lot of parameters and hyper-parameters and tuning them require a lot of data training data, computational resources and care.

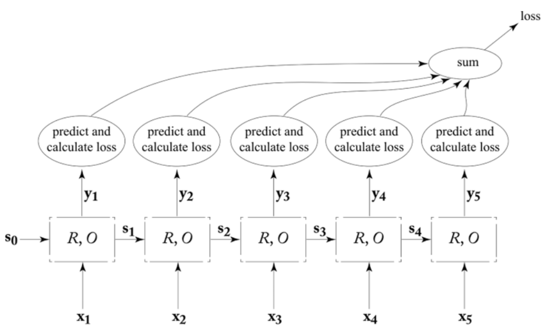

loss
sum
predict and calculate loss
predict and calculate loss
predict and calculate loss
predict and calculate loss
predict and calculate loss
y1
y2
y3
y4
y5
s0
R, O
s1
R, O
s2
R, O
s3
R, O
s4
R, O
x1
x2
x3
x4
x5

Fig 1: Transducer Training Model for RNN.
Image Credits: Taken from the book Neural Network Methods for Language Processing by Yoav Goldberg

Fig 1, shows a schematic of a transducer model of RNN which produces an output for each input it reads in. Such a model can be used for language modelling where for each input token we would want our model to capture the next token and state variables captures the context. Two different Language models are built and compared. One, a character level Language Model and the other at the word level.

The report is organized as follows. In section(2) model and implementation details are discussed. Then in Section(3) some emperical results are presented and the report is concluded with a discussion section.

2 Model Design

A four layer neural transducer model is designed which is capable of handling varying size of input sequences and outputs a sequence of size equal to the size of of the input sequence. A block diagram of the architecture of the model is shown in Fig 2. Below is a brief description of the functions of each layer.

Embedding Layer: The layer enables the model to learn task dependent embeddings for all the elements. This layer takes a sequence of tokens as input in the form of an array with each token represented by its index no. in a predefined dictionary. Since here the embeddings are tuned for the task hence it outperforms using predefined embeddings using word2vec or Glove models. The layer has single hyper parameter which is the size of the embedding for each token which needs to be tuned

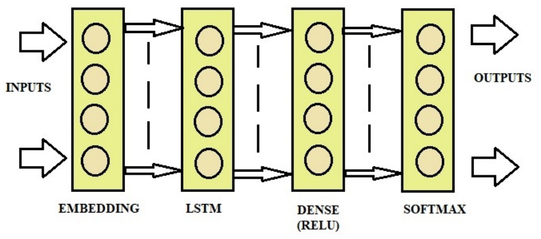


Fig 2:Model Design Architecture

LSTM Layer : LSTMs are very effective in capturing long-range structural dependencies which are essential to the task of Language models and which gives Neural Language Models advantage over N-gram based Models. This layer operates in a many-many transducer mode. This layer outputs the states at each time step. Here the size of the state vector in the layer is treated as the hyper parameter.

Dense Layer: This layer is added to increase the representational power of the model. It takes a linear combination of all the state outputs from the LSTM layer and pass them through a non linear activation function. In this layer we have to make two choices, one is the no. of computational units and the other the type of activation function. For this project all the experiments are done using 'RELU' activation because of its fast convergence and the no. of hidden units is treated as a hyper parameter.

Softmax Layer: This layer outputs a matrix having same no. of rows as the no. of elements in the input sequence and each row is a probability distribution over the entire vocabulary which signifies the probability of each element in the vocabulary to be the next element in the sequence. Hence our model has the following hyper parameters:

- Embedding Size
- State Vector Size in the LSTM layer
- No. of computation units in the Dense layer

3 Experiments and Results

The publically available Project Gutenberg corpus is used for this project. **Gutenberg Corpus** is a collection of 18 English books of several different authors containing around 49,000 unique words and 2,650,000 total words. Due to limited computational resources 6 text documents are used for experimentation.

A major difference between parameter training of character and word level models is the staggering difference in the Vocab size. Vocab size in character level model remains between 55-65 whereas in word level it can go upto 16,000 (after converting the entire corpus to lower). Fig 3 shows the dominating impact of vocab size on the no. of training parameters in the first Embedding and final Softmax layer. Also generating the training data as discussed above we can generate a lot more training examples for character level model then word level model. Hence training word level Language Model posed several challenges. So in order to keep a check on the no. of training parameters in the word level more conservative values for hyper parameters is used as compared to character level model.

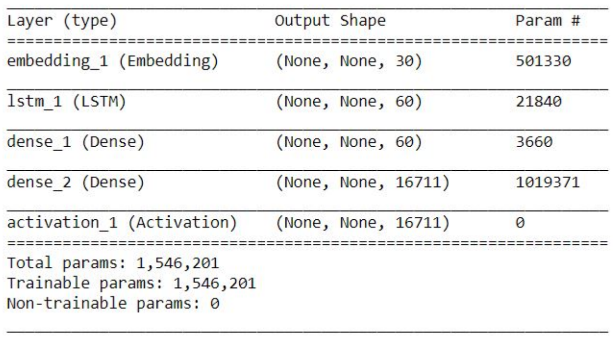
Due to limited computational resources without much validation the following values of the hyper parameter is used.

Word Level Model

- Embedding Size: 30
- State vector size in LSTM: 60
- Computation units in Dense Layer: 60

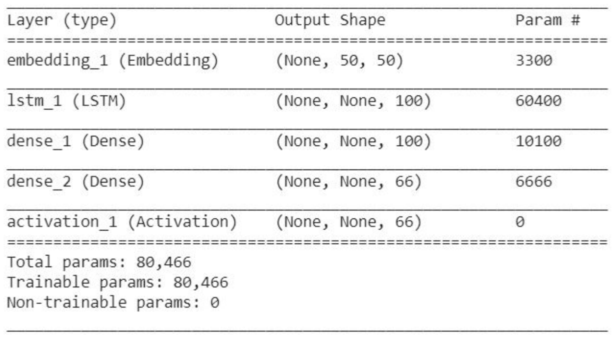
Char Level Model

- Embedding Size: 50
- State vector size in LSTM: 100
- Computation units in Dense Layer: 100



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 30)          501330
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 60)          21840
_________________________________________________________________
dense_1 (Dense)              (None, None, 60)          3660
_________________________________________________________________
dense_2 (Dense)              (None, None, 16711)       1019371
_________________________________________________________________
activation_1 (Activation)    (None, None, 16711)       0
=================================================================
Total params: 1,546,201
Trainable params: 1,546,201
Non-trainable params: 0
_________________________________________________________________
```

Fig 3a: Word Language Model



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 50)            3300
_________________________________________________________________
lstm_1 (LSTM)                (None, None, 100)         60400
_________________________________________________________________
dense_1 (Dense)              (None, None, 100)         10100
_________________________________________________________________
dense_2 (Dense)              (None, None, 66)          6666
_________________________________________________________________
activation_1 (Activation)    (None, None, 66)          0
=================================================================
Total params: 80,466
Trainable params: 80,466
Non-trainable params: 0
_________________________________________________________________
```

Fig 3b: Character Language Model

The training data is prepared by sampling 50 token sequences from the entire corpus using a moving window. And an 80:20 split is used to get the train and test dataset. For both Neural models **ADAM** optimizer is used with **cross-entropy loss** for training the models.

Table 1 contains a comparison of different models on the test data. Bigram Kneser Ney model has also been included to compare the N gram models with Neural models. Perplexity on the test data is used as the evaluation parameter of the different models.

Model type	**Char LM**	**Word LM**	**Bigram Kneser Ney**
Perplexity	6.45	83.24	141.99

T1: Table showing Perplexity values on the test data.

In the case of Word Level Language model there were still some instances occurring when the probability of correct class was very close to zero and in the above results such cases have been removed. The above results are discussed in the following section.

4 Discussions

The perplexity values clearly indicates towards the effectiveness of the Neural models over N-gram models. This is on expected lines due to the nature of language modelling where long term dependencies capture a lot of information. Also in N-gram models inherently are not capable of handling the cases when a particular N-gram in the test data is not seen in the training data though all the individual tokens are present in the vocabulary. We are then required to several approximate smoothing techniques. But this problem doesn't really occur in any of the Neural Models. Both Word Language model and N-gram models are suffer from the out of vocabulary problem quite frequently whereas it is very rare to encounter an out of vocabulary token in Character Language model.

A more interesting observation and discussion point is the comparison between Word and Character Language models. As expected the results clearly depict that the Word Language model suffers from curse of dimensionality. As shown in Fig 3. there are almost 20 times the no. of parameters to be tuned in the Word model than the character model. This difference is attributed to vast difference in the Vocabulary size of the two models. Also, the amount of training data of word model is almost a third of the parameters. Hence, it is expected that the word model actually overfits the available training data. This also explains why while finding test data perplexity at some predictions the model assign close to zero probability to the true word.

5 Sentence Generation

In this task we have to generate a random sentence by giving an initial token to the system. Here the Neural models didn't perform as well as expected, whereas N-gram models which essentially chooses sequences directly from the training data itself were really effective. This can also be due to the fact that we are only generating a 10 token sentence where N-gram models can also be effective as there we are anyways capturing small context very well. Neural Language models are expected to work better when generating longer sentences where there would be a need to capture longer dependencies.

This do reflect some shortcomings in our training procedure. For final sentence generation we are using the Word Language Models because the Character Language model was generating some non-words.

Random sentences generated through Word Language model.

> ”the most valued by his manners i was hardly see an invitation”

> ”the loss of some very high change of your love so”

> ”the absurdity of congratulation which were without all its recommendations”

Random sentences generated through N-gram model.

> ”We are worried about their machinery beyond mechanical details.”
>
> ”(2) slow down and permit the passage.”
>
> ”For the prevention of anaplasmosis, feed 100-200 grams.”

Clearly the N-gram sentences are more syntactical and semantically correct.

6 Conclusion

The biggest take away from this project is to get some initial exposure in applying deep networks and to get aware about the challenges of training these models. Also through this project I could develop some incites about the functioning of the different layer like Embedding, LSTM and Dense layers.

7 Github Link

All the code files can be found at the below link:

https://github.com/229Swapnil/Neural-Language-Model