NLU Assignment 2 Report

Lokesh (14355)
lokeshn@iisc.ac.in

Abstract

Below presented is the report of the assignment 2 which is about building a language model using neural networks. The assignment is done using keras and tensorflow APIS. A language model is about assigning probabilities to the words that could potentially follow a given context. There are two variants in this assignment.
1. Building a language model of words
2. Building a language model using characters
Hence we assign probabilities to words and characters respectively. Once we have modelled the probability distribution we evaluate the performance of the language model by checking how well suited this pobability distribution is againgst an unseen test corpus. We use perplexity as a measure of this. And we also try to generate a random sentence from the language model in compliance with the learnt distribution

1 How can neural methods model language

A neural network is abstractly an algorithm that learns functions. This is done by the algorithm exploring the relationship among the domain and codomain from the given many examples. Because of presence of nonlinearity in the network, it can theoritically approximate any function given the examples in an appropriate format that the network can understand. In our case since the examples are in text format, we need to convert them into meaningful numbers also called as representations. Naively there are methods like one hot encoding that can solve this problem. Hence we can formulate the examples like {(x,y)} pairs where x is the equivalent representation for the text in numbers and y is the probability distribution we are trying to achieve. We can handle OOV words etc as done in the assignment 1.

2 Incorporating contexts is an overhead

In a traditional neural network, it is difficult to enforce the context. To do so using a neural network, the dimensionality of the input would explode exponentially as now we would like to learn unique representations for every possible context which we would ever encounter. Therefore we have Recurrent Neural Networks that can learn the representations effeciently incorporating the context also.

3 Models that also encode time (time is context here)

Since RNN also handles context its output and input would be time dependent now. Hence our input would be now {(x(t), y(t)} pairs. Simply put RNN is a connection of many neural network units connected in sequence with feedback edges from past into the present unit. These feedback edges help to incorporate contyext in the learning process. Also to control the explosion of the parameters that need to be learnt, the individual neural network units share the parameters.

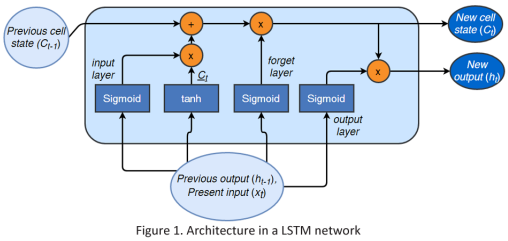

Figure 1. Architecture in a LSTM network

Figure 1: LSTM Cell.

Shown above is a single LSTM cell.

Input to each LSTM cell is
c(t-1) cell state so far.
x(t) the input at time t, or t word in the context
h(t-1) it is an encoding of what has happened in the past. This essentially helps LSTM to model Long term dependencies

Output from each LSTM cell
c(t) The current cell state
h(t) the updated context

First we concatenate input with the past (x(t).h(t-1))to obtain the new context.

LSTM has a cell state C(t) that runs horizontally through the entire LSTM. It does only linear interaction. what passes through and what gets filtered away is decided by each of the individual LSTM cell. What gets passed through is determined through x(t) and h(t-1). We apply sigmoid non linearity, which squashes them to a number between [0,1], for that pair and value of 0 implies nothing from c(t-1) gets propogated further and value of 1 implies we propogate everything of c(t-1) further. Now that we have forgotten some of the irrelevant state from c(t-1), we need to add new information from that is generated from x(t). This is done by the sigmoid and tanh layers. Both of these layers together update c(t) through an add gate. Finally the output h(t) we generat from the LSTM cell is a combination of some of the cell state c(t) and the input x(t)

Shown below is the entire RNN, a collection of LSTM cells connected in sequence.

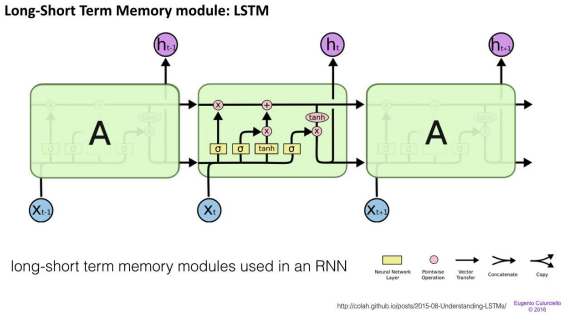

Figure 2: RNN

4 Output of the neural model is likelihood distribution and not prob

The output obtained from the final layer of the RNN is n dimensional vector. Because we are modeling a probability distribution we can interpret each component in the n dimensional vector as likelihood values. ie. each component xi tells us how likely it is to follow the context x1x2....xt-1. To convert into a probability distribution we can apply smoothing transformation called softmax. Once we have obtained the probability distribution we can easily generate the sentence and evaluate the perplexity.

5 How to obtain the character/word from the likelihood

Naively we can choose the component that has the highest probability from the output distribution. This in some sense, selects the word/character deterministically. But from what i have observed, we get good sentences when we incorporate randomness exactly as dictated by the network. To do so, we should model the selected word as a multinomial Random variable which can take n values (the dimension of output obtained from the RNN). The pmf of the random variable should follow the distribution as predicted by the RNN. Hence we sample a multinomial random variable according the pmf as obtained by the RNN.

6 Influence of Diversity Parameter

Because the output of the RNN is only likelihood values, there are many ways to obtain probability distributions from them like
1. softmax
2. $\frac{xi}{\sum xi}$
3. etc.

All of them are correct and yield different results. But we have chosen softmax because it is a smooth approximation and has some nice properties. There is a trick to control the relative impact of the likelihood values. We can define a diversity parameter. We can divide each of the likelihood values with this diversity parameter before applyting softmax. The impact of the diversity parameter is as follows.

Case 1 : diversity <1
It makes large values much larger
It makes smaller values less larger

Case 2 : diversity >1
It makes larger values much smaller
It makes smaller values less smaller.

It is easy to see that this diversity transformation does not change the distribution as is but only affects the relative importance of each of the components in the output.

Example sentences from character lstm with different diversity parameters.
Random seed given to the model is : "t of her hair , she was almost sure of b"
diversity: 0.2

t of her hair , she was almost sure of be as the dressing and the distression of the dressing

diversity: 0.5
t of her hair , she was almost sure of better than the rest she was made often in the course of her years

diversity: 1.0
t of her hair , she was almost sure of be , came . He had no lord of yeinpble clouse impossiblise

diversity: 1.2
t of her hair , she was almost sure of berered so jumg asow thought without a year ospome account ?

For character model it is easily observed that lesser diversity values yield better sentences. It is observed empirically that with diversity 0.2 atleast there was not even a single meaningless word. We can try to atleast interpret this result as follows For the character level LSTM we should not give more diversity because among the 26 characters in the english language letters like vowels are more prominent in the words. Hence giving more diversity attempts to shift the language away from the vowels and yields wrong words. We can also see the significance of this diversity parameter itself. As we can observe when the diversity is 1.0 we blindly encode the output of RNN as multinomial random variable and hence the results are not as promising as they are with diversity of 0.2.

7 LSTM Configuration

The character and the Word LSTM configuration are as shown in the diagrams. For Character LSTM we had 81 unique characters and each of those characters are one hot encoded. For word LSTM each of the the word has an index. Hence representations are replaced by a look up table. I have taken a batch size of 128 and context window of 40 characters for char-RNN and context of 40 words for the word-LSTM.

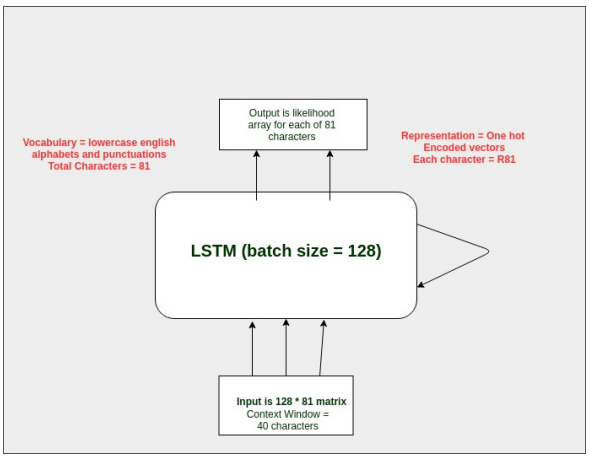

Figure 3: Character Model

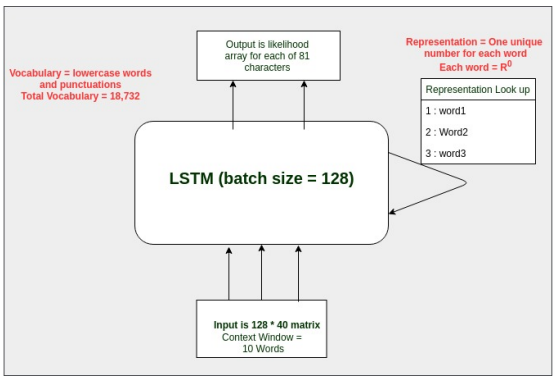

Figure 4: Word Model

Results and Observations

The perplexities of the older triGram Language model using Kneser-Ney smoothing is 220.58.
Using neural models the perplexity obtained is
For Char-LSTM : 3
For word-LSTM : 104.9

For LSTM we had 554796 examples and we had a batch size of 128 and hence each epoch has 4334 runs. Below presented graph reports the trend of how the loss and perplexity is varying as the model is getting trained.

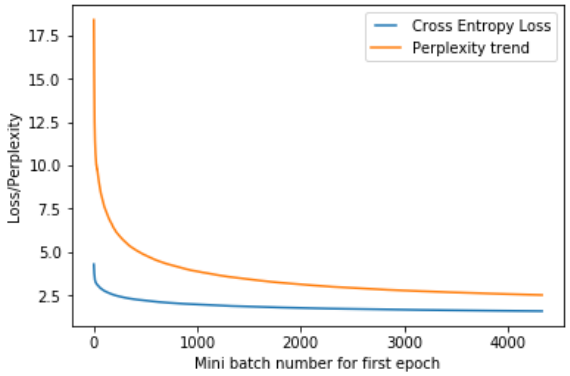

Figure 5: Loss and Perplexity trend for the first epoch

As we can see from the model, because we had many batches the loss is behaving very well from the very first epoch.

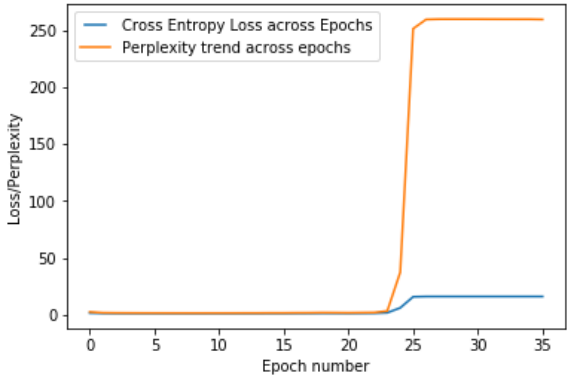

Figure 6: Loss and Perplexity trend across epochs

The above image shows a surprising result. We are able to observe that the perplexity is shooting up after a certain epoch. for my character LSTM this is observed from the epoch number 25. Hence i am using the model that is saved after epoch 24 for generating the sentences. Though the actual reason for this behavior is unsure. I can think that maybe the model is overfitting and hence going for a toss. This is also validated from the quality of the sentence generated. For the model that is obtained after 30 epoch the sentence is as follows

YYKYKYYYYKYKYYYYYYYYYYKYKY YYYYYxYYoYYda?r ;er dY ; MtKjeK YhKYhKfiYi oY Kqje YoYY;o' oYaT" - ;"Y s' ; YoBirrKghYKgYKf KrpY !YYYw YYK YKteY:te eqeKY' H ny l ,eKY CoYYKY KY Ydof teYTY'

We can observe that the sentence is completely gibberish.

8 References

LSTM
Keras Example Code