Language Modelling

Sree Latha Chopparapu
UC Santa Cruz
schoppar@ucsc.edu

1 Introduction

A language model in Natural Language Processing is a probabilistic statistical model that calculates the probability of a given sequence of words occurring in a sentence based on the previous words. It helps to predict which word is more likely to appear next in the sentence. Typically, the input to a language model is a training set of example sentences. The output is a probability distribution over sequences of words.

1.1 Dataset

The Dataset used for this language model is Penn Treebank (**PTB**) dataset. As per my research through internet, the PTB dataset is a collection of numerous stories from a three year Wall Street Journal (WSJ) collection and the dataset is maintained by *University of Pennsylvania*. The dataset is downloded by **load_dataset** function huggingface's **datset** library. The PTB dataset does not contain capital letters, numbers (numbers are represented by N) and punctuation, and the vocabulary is capped at 9743 unique words (as per my tokenzation method). The downloded the ptb object is a dictionary with three splits i.e. train, validation, and test, and each example in each split is a dictionary with one key: sentence. Train set, test set and Validation set consists of 42068, 3761, 3370 of such sentences respectively. For example, *ptb['train'][25]['sentence'] -> 'in july the environmental protection agency imposed a gradual ban on virtually all uses of asbestos*'. To indicate the start of the sequence a <sep> token is added at the beginning of each sentence of training and validation sets. The entire corpus is tokenized using split(), with delimiter as ' ' (space), few hyphenated words are seperated with split('-'). After the tokenization process train set has 935556, validation set has 74258 and test has 78669 tokens. Out of all the tokens 9743 tokens are found out to be unique from which the vocab for this model is created.

1.2 Creating Data buckets

The tokenized data is enumerated to create a vocab dictionary, through which all the tokens are encoded. All the encoded tokens are divided into small data chunks (with a fixed chunk size, per se 10, 50 etc). Seperate data buckets are created for input sequence and target sequence like, if input sequence is ['<sep>', 'in', 'july', 'the', 'environmental', 'protection', 'agency', 'imposed', 'a', 'gradual'], then the target sequence is ['in', 'july', 'the', 'environmental', 'protection', 'agency', 'imposed', 'a', 'gradual', 'ban']. For the test data, the data has been tokenized and encoded but did not create separate vocab for test data as we use this for the inference process.

The primary goal of the task is to build a language model with the possible lowest perplexity. **Perplexity:** Perplexity is a measure of a probability model's ability to accurately forecast a sample. Perplexity is one technique to assess language models in the context of natural language processing.

$$PP(W) = P(w_1, w_2, w_3....w_N)^{(-1/N)} \quad (1)$$

Where N - is the number of words in the corpus

2 Models

The language models can be built by either probabilistic methods or neural network models. The probabilistic language models are built using n-gram models in which the probability of n^{th} is the conditional probability that the n-gram's last word follows the a particular $n-1^{th}$ gram. These probabilistic methods have evident drawbacks. So for this task I have used a fixed window neural network model. Using the same word to ID method for vocab, I have created an embedding layer of

vocab_sizexembedding_dimension, i.e. 9743x300. As language modelling is a context based problem, recurrent neural network (RNN) would best suit for modelling. For this task I have used both Gated Recurrent Unit (GRU) and Long Shot Term Memory (LSTM) architectures.

RNN is a unique kind of artificial neural network called a recurrent neural network (RNN) is designed to cope with time series data or data that contains sequences. Standard feed forward neural networks are only suitable for independent data points. To include the dependencies between these data points, we must change the neural network if the data are organized in a sequence where each data point depends on the one before it. RNNs have the idea of "memory," which enables them to save the states or details of prior inputs in order to produce the subsequent output in the sequence.

2.1 LSTM

LSTM Long Short Term Memory network is one kind of RNNs which are designed to learn long-term dependencies. In general, LSTMs are created to prevent the long-term reliance issue. Remembering information for extended periods of time is basically their default behavior. LSTMs generally have the cell states which stores or discard the information based on sequence of the context by structures called gates. An LSTM has three of these gates, to protect and control the cell state namely input, forget and output gates. Input gate adds the data to the cell state and forget gate layer is the sigmoid layer which maintains the cell data i.e. either add or discard the information. In this task, the cell states and the hidden states are initialized with zeros as the language models require some initial value to predict the following occurring word.

2.2 GRU

GRU Gated recurrent units, are improved version of LSTM, The update gate and reset gate mechanisms are used by GRU. In essence, these two vectors determine what data should be sent to the output. They have the unique ability to be trained to retain information from the past without having it fade away over time or to discard information that is unrelated to the forecast.

The models were trained using the training data provided, keeping the hyper parameters initialized with some random but basic values. The hyper parameters used are hidden size = 128, 256, number of epochs = (10- 50), learning rate = (0.01 - 0.05), number of layers = 2, embedding size = 300, vocab size of 9743 for the probability distribution of predicted word.

In the training process of neural network, it is very important to optimize the model parameters (weights) for minimizing the prediction error which is a highly complicated task. Certain algorithms are available for optimizing these weights and in this task SGD(Stochastic gradient descent) optimizers are used for optimizing the weights. While training the model, loss is calculated which the difference between predicted output and the ground truth. In the training process, we mainly focus on reducing the loss. For this task, I used, “CrossEntropyLoss” function from PyTorch framework, for the analysis of the loss during training.

3 Experiments

While training the model the hyper parameters were tuned based on the gap between training loss and validation loss. I have conducted many experiments throughout the model training. The hyper parameters were constantly tuned to yield better results. In this task, I have tuned the learning rate to yield better results.

In this specific task I found batch size hyper param to make very significant changes in the training process. With a larger batch size of 64 the perplexity observed was very huge around 9000+, but as the batch size is decreased the model performed far better relatively. While experimenting, batch size 1 seemed very ideal for me to train the model that I built. With batch size 1 the test perplexity seems to be 224. I have experimented with both LSTM and GRU, for LSTM the data chunks were made with a size of 50 and for GRU the chunk size was made to be 10. In both the models smaller the batch size better the model performance. But computational wise I found LSTM to be better, as both the networks were giving approximately same results, but LSTM data chunk size was 50 and was performing almost equally with GRU.

EXP#	Model	BS	PP (Approx)
Exp1	LSTM	1	300
Exp2	LSTM	64	700
Exp3	GRU	1	225
Exp4	GRU	64	336

Table 1: Experiments with LSTM and GRU

In the above table:*BS -Batch size, PP -Perplexity*

4 Results

After numerous experiments with hyper parameter tuning and model dimensions etc, the Model trained was settling mostly on a testing perplexity of (Approx.)225. Through out the training the complicated task was to choose a correct batch size, which impacted the results drastically. Below plots shall illustrate the experiments graphically.

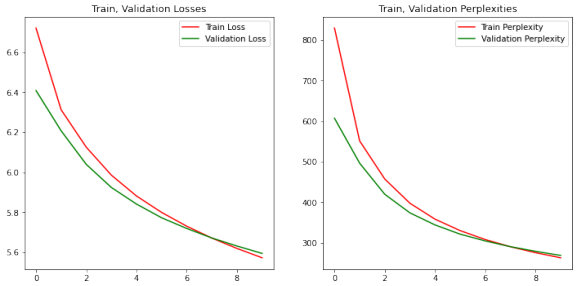

Figure 1: LSTM BS-1, Train loss, Validation Loss and perplexity graph

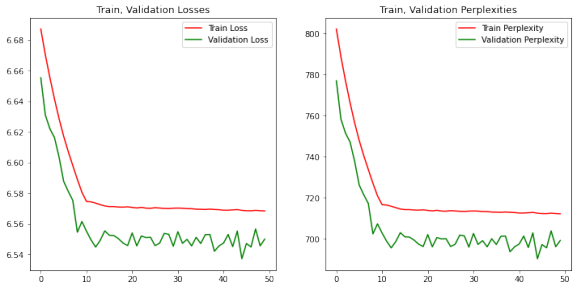

Figure 2: LSTM BS-64, Train loss, Validation Loss and perplexity graph

Figure 3: GRU BS-64, Train loss, Validation Loss and perplexity graph

5 References

References

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/

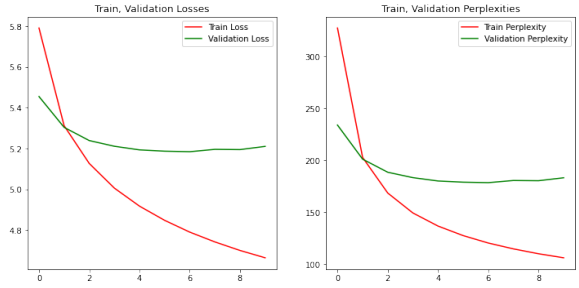

Figure 4: GRU BS-1, Train loss, Validation Loss and perplexity graph

[2] https://towardsdatascience.com/the-beginners-guide-to-language-models-aa47165b57f9

[3] https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

[4] Smith, N. A. (2017). Probabilistic Language Models 1.0. Technical report.

[5] Bahdanau, D., Cho, K., Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[6] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.