Evaluation of Penn Treebank Dataset

Ali Akay [224414]
Trento University
Via Sommarive, 9, 38123 Povo,Trento, Italy
ali.akay@unitn.it

ABSTRACT

In this paper, I employed the Language Model, which is a word-level recurrent neural network (LM). In the first part of this project, I used the Penn Treebank Dataset for training one of the most popular RNN architectures, Long Sort Time Memory (LSTM). Then I try to make the model as efficient as possible. It was discovered that training with the appropriate hyper-parameters had an impact on evaluation.

1 INTRODUCTION

The development of probabilistic models capable of predicting the next word in a sequence given the words is known as language modeling. Based on text examples, a language model learns the probability of word occurrence. Many speech and language processing tasks, such as speech recognition and machine translation, rely on LM. RNN-based LMs are the current state of the art (Mikolov et al., 2010) RNN achieved 123 perplexity with the Penn Treebank [1], whereas LSTM achieved 78 perplexity with the same dataset [2]. With the same dataset, language models with transformers produce excellent results. In this paper, I tried to implement and evaluate LSTM models, as well as reduce perplexity values for test datasets.

2 LANGUAGE MODELLING

To start with N-gram models, based on counting the probability of observing each possible bi-gram. This is a really efficient way to make use of the data especially when you don't have a lot of text to train from. N-gram models can beat neural network models on small datasets. Estimating bigram probabilities formula can be define as:

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_i)} \tag{1}$$

There are a few advantages to neural networks that n-gram models do not have. If a recurrent neural network is used, it can take advantage of longer word histories. It can also share parameters with other n-grams of the same type. The gradient in a recurrent neural network explodes or decays exponentially over time. (Time steps correspond to word positions in a sentence from the perspective of language modeling.)[3] The major issue is that the RNN has a hard time learning to conserve information over a large number of time steps.

$$\text{RNN} : h_t^{l-1}, h_{t-1}^l \rightarrow h_t^l$$

A new type of RNN has been presented as a solution to the problem of vanishing gradients. There is a hidden state $h(t)$ and a cell state $c(t)$on step t. Both are n-dimensional vectors, and the cell

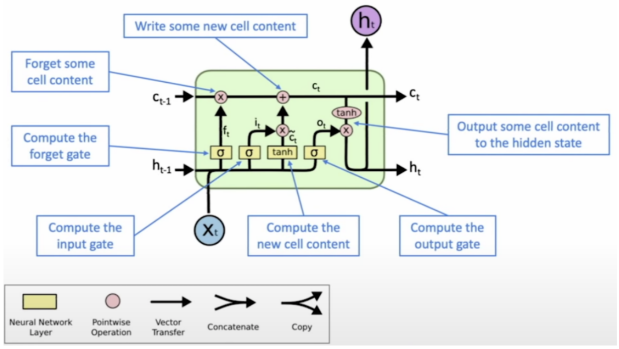

Figure 1: LSTM

stores long-term memory. The LSTM has the ability to erase, write, and read data from the cell. Three corresponding gates manage the selection of which information is erased/written/read. The gates are also vectors length n. On each time step, each element of gates can be open (1), closed(0), or somewhere in-between. gates value is computed based on the current information. We have a sequence of inputs $x(t)$, and we will compute a sequence of hidden states $h(t)$, and cell states $c(t)$.

The first step in our LSTM is to decide what information is going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

The LSTM architecture used in our experiments is given by the following equations [4]

$$\text{LSTM} : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$

2.1 Project Model Structure

I used word embeddings that provide a dense representation of words and their relative meanings.To begin, the dataset has 10K vocabulary, and torchtext adds 1 extra vocabulary, giving me 10001 input. I use embedding size **[10001,250]** based on general rule of thumb **Vocab*0.25** .Then I used LSTM layers with 400 hidden number **(250,400)**.After the LSTM layer I used dropout regularization to reduce overfitting. For the final layer I add fully connected layer

which shape with [batch-size,number of vocab] so in the prediction,we have output with shape of [batchsize*bpttlen,10001]. In my best case I use batch size is 45,bptt is 40 so my output shape is **[1800,10001]**.
At the output layer,multinomial distribution,which can be naturally parameterised by a softmax function to produce correctly normalized probability values. As training criterion the cross entropy error and perplexity is used. [5]

The cross entropy formula takes in two distributions, p(x) the true distribution, and q(x), the estimated distribution, defined over the discrete variable x and is given by

$$H(p,q) = -\sum_{\forall x} p(x)\log(q(x)) \tag{2}$$

$$H(\mathbf{y},\hat{\mathbf{y}}) = -\sum_{\mathbf{i}} \mathbf{y_i}\log_{\mathbf{e}}(\hat{\mathbf{y}}_{\mathbf{i}}) \tag{3}$$

Where *yhat* is the predicted probability vector (Softmax output), and *y* is the ground-truth vector.The reason we use natural log is because it is easy to differentiate and the reason we do not take log of ground truth vector is because it contains a lot of 0's which simplify the summation.

Perplexity metric is connected with cross entropy loss. We can define it given formulation.

$$2^{H(\mathrm{y},\hat{\mathrm{y}})} = \text{perplexity} \tag{4}$$

I use RMSprop for optimizer to adjust learning rate.Rmsprop is a gradient-based optimization technique propsed by G. Hinton his Coursera lecture.
Here is project model summary:

WordLSTM(
(dropout): Dropout(p=0.6, inplace=False)
(emblayer): Embedding(10001, 250)
(lstm): LSTM(250, 400, numlayers=2, batchfirst=True, dropout=0.6)
(fc): Linear(infeatures=400, outfeatures=10001, bias=True))

You can see more detailed of model graph in tensorboard by using **this link**.

3 DATASET

The University of Pennsylvania provides the Penn Treebank dataset. It's massive, with almost four million and eight hundred thousand annotated words, all of which have been manually corrected by humans. In total there are 10K vocabulary in the dataset.
In this project, I simply used a word-level data (with the exception of one tag — <unk> , which is used for rare words such as uncommon proper nouns) for our model. Also I added <eos> at the end of the sentences. In traning set, averege token for one sentence is 22.

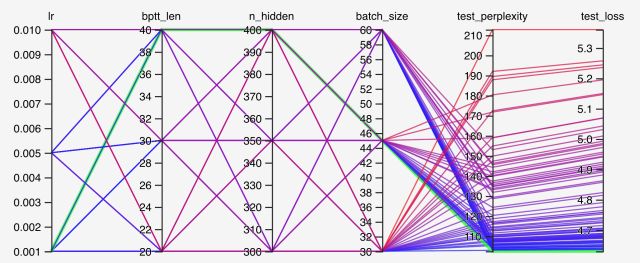

Figure 2: Hyper-parameter Tuning

LR	Bptt	Num Hidden	Batch size	Perplexity
0.001	40	400	45	102.50
0.001	40	400	30	102.96
0.005	30	400	60	103.10
0.005	40	350	60	103.31
0.001	30	400	60	103.58

Table 1: Perplexity Values of best 5 model

4 EXPERIMENTS AND HYPER-PARAMETER TUNING

In this project, I used pytorch to create LSTM model and torchtext to create dataloader and data pre-processing part of the project.

I experimented with 87 models with various batch sizes, bptt sizes, learning rates, and hidden layers. To display the models, I used tensorboard. Due to the computational power and time restrictions of colab, I conducted my tests using 30 epoch. Green model performs with the smallest cost of perplexity(bpttlen = 40 lr = 0.001 nhidden = 400 batchsize = 45.) [Figure 2]. The most important component of this graph is that the perpexity increases as the learning rate increases.

5 RESULTS

I run my best model with 100 epoch after the hyper parameter tuning. However, thanks to an early stopping algorithm, it was over at 83 epoch. Here are my train,validation and test perplexity values :

model saved.
Epoch: 83
Training loss 4.34766, Training perplexity 77.30
Validation loss 4.63159, Validation perplexity 102.68
————————————————————
model saved.
early stopped
After training:
model saved.
Validation loss 4.63004, Validation perplexity 102.52
Test loss 4.55423, Test perplexity 95.03

You can see more detailed of results in tensorboard by using **this link**.
You can see the project codes on colab with **this link**.

REFERENCES

[1] Mikolov, Tomas and Karafiát, Martin and Burget, Lukas and Cernockỳ, Jan and Khudanpur, Sanjeev *Recurrent neural network based language model.*. 2010.
[2] Shaojie Bai 1 J. Zico Kolter 2 Vladlen Koltun 3 *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. 2012.
[3] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney *LSTM Neural Networks for Language Modeling*. 2012.
[4] Graves, Alex and Mohamed, Abdel-rahman and Hinton, Geoffrey *Speech recognition with deep recurrent neural networks*. 2013.
[5] Huyen, Chip *Evaluation Metrics for Language Modeling*. 2019.