HW 3: Language Modeling

Parsa Mazaheri
UC, Santa Cruz pmazaher@ucsc.edu

Abstract

1 Introduction

The goal of this homework is to train a model to design and training a language model on the Penn Treebank Dataset.

As you know, (auto regressive) language modeling is the problem of predicting the next token in a sequence given the previous N tokens. and for example if we give the mode a sentence of `Today is going to be rainy <t>`, the model should be able to predict token `<t>` based on the other tokens previously seen in the sentence.

2 Problem Description

In this assignment unlike the previous problems, we want to work on a **Unsupervised Learning** in which the model should find the relation between the data itself. supervised learning is a machine learning approach where we have a dataset of labeled data and we train a model to predict the label of new data and Unsupervised Learning is the method to give the model data to predict for itself without giving it any labels or tags.

3 Dataset

For dataset, we're using the given data by the course instructors pdf which is Penn Treebank dataset. The Penn Treebank is a corpus of 1 million tokens. For getting the data we use `dataset` package to load the dataset super easy without any other work. In this assignment, the data is already in a good format for us to use. We don't need to do extensive data cleaning. However, we need to do some preprocessing to the data.

The preprocessing phase consist of different actions which has been explained below.

4 Preprocessing

Data preprocessing is a step in data analysis that takes raw data and transforms it into a format that can be understood and analyzed by computers and machine learning models. Good, preprocessed data is more important than the most powerful algorithms. By doing a good preprocessing on the data, we can save a lot on optimizing the model, hyperparamters, and trying to generalize the problem because of the diversity of the data. The preprocessing consists of different actions on the data which has been explained briefly in below.

4.1 Tokenization

In this step, we remove the punctuation which consists of `'!()-[]{};:'"\,<>./?@#$%^&*_~` symbols. After that we remove the stop words (such as "a", "an", "the", "of", etc). Again, since here we're dealing with a good data, we just need to **create a vocabulary** for our need based on the tokens. which we later need for mapping words and ids, and we're going to use two dicts for `word-to-id` and `id-to-word` in the code.

4.2 Embedding

For this task we're using Glove `6B` word embedding in which we're going to use for our task and don't bother ourselves with embedding and just import a state-of-the-art embedding. We can use more powerful and bigger embedding as well like `Glove 840B` or `Glove 42B` but it's overkill here to do so and I like 6B is good enough to do the job.
Since the data is also split into `train`, `test`, and `validation`, we don't need to do anything on this matter as well.

=pt

5 Hyper-parameters

One of the key challenges of every deep learning task is to determine the best hyperparameters. The hyperparameters are the parameters that are used to train the model. The hyperparameters are not learned by the model and therefore aren't updated during the training process. The hyperparameters that I've used for this task are shown in the following table.

Hyperparameter	Value
Learning Rate	0.001
Batch Size	64
Number of Epochs	10
Number of Layers	2
Dropout Rate	0.2

Table 1: Hyperparameters

6 Model Architecture

For the task, I've used different models from MLP, GRU (Chung et al., 2014) to LSTM (Hochreiter and Schmidhuber, 1997) and regular RNN. The best performance was achieved by the LSTM 1 hidden layer model. The model architecture is shown in the following figure. For each of the models, the model takes the utterance as an input and passes it through an embedding layer. The embedding layer converts the words in the utterance to vectors. The vectors are then passed through the rest of the model to be processed.

For more information on each model used, we've provided a little more detailed description in the following subsections.

6.1 LSTM Model

LSTM [1] is a type of recurrent neural network that is used for sequence modeling tasks. LSTM is a type of RNN that is able to remember information for a long time. LSTM has 3 gates: input gate, forget gate, and output gate which is shown on figure 1. The input gate decides which values from the input to update the cell state. The forget gate decides which values from the cell state to update. The output gate decides which values from the cell state to output. The output of the LSTM layer is a hidden state for each word in the utterance. The hidden states are then passed through a fully connected layer to get the output. The output is then passed through a sigmoid layer to get the probability of each relation.

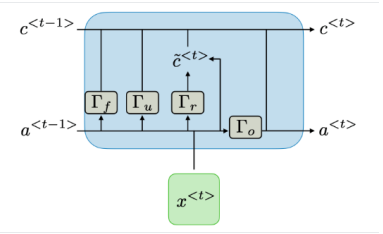

Figure 1: LSTM cell architecture

For this task, since we're going to generate a sequence of tokens and don't have the output, we're not gonna use bidirectional.

6.2 GRU

GRU [2] are very similar to Long Short Term Memory(LSTM). Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over them and have simpler architecture.

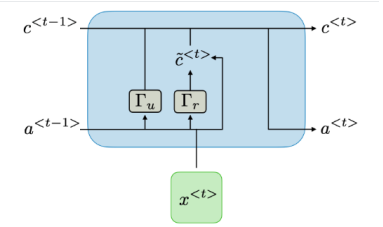

Figure 2: GRU cell architecture

7 Experiments

Since for this research project, there wasn't a lot of time to do experiments, I've only done a few experiments. The experiments that I've done are shown in the following subsections.

7.1 Hyper-parameters

For learning rate, a too high learning rate will make the learning jump over minima but a too low learning rate will either take too long to converge or get stuck in an undesirable local minimum. I've tested `learning_rate` from `0.01` to `0.001` and the best value is `0.001`. A sample comparison is shown for to of them in table 2.
For batch size, 64 here is pretty standard and based on the size of our dataset, it's good choice to use.

[1] Long Short-Term Memory

[2] Gated Recurrent Unit

=pt

Learning rate and Num of layers	Perplexity
lr=0.01, layers=2	112.39
lr=0.001, layers=2	107.67
lr=0.01, layers=4	110.31
lr=0.001, layers=4	105.54

Table 2: learning rate and batch size influence on score - All of them using LSTM

we can use 32 as well here.for the right learning rate, batch size 64 and 32 doesn't have much difference but 64 is a little bit higher.

For number of layers in this experiment, I tried different values from 1 to 4 hidden layers. The 4 layers had slightly better performance than 2 and 3 hidden layers and in the end we used 2 hidden layers for the model since more hidden layers increase the parameters and model size in which in this case wasn't with big difference.

For epochs, again since we're not handling a big data and a complex model architecture, 10 would suffice, but 20 can also be used but may cause the data to over-fit.

7.2 Optimizer

The optimizer that I've used is Adam optimizer. Adam optimizer is a type of stochastic gradient descent optimizer. Adam is a method that computes **adaptive learning rates** for each parameter. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients, similar to momentum.

7.3 Number of Hidden layers

When increasing the number of layers, the model is able to capture more complex relations. However, increasing the number of layers too much, also increases the number of parameters in the model and the context may be lost. As shown in Table 2, here that we have a small dataset, increasing the number of layers doesn't help much. **The best performance was achieved by the model with 3 hidden layer.**

7.4 Dropout

Dropout (Srivastava et al., 2014) is a technique for regularization. Dropout is a technique where randomly selected neurons are ignored during training. by using dropout in this problem, I didn't encountered much difference, since we're dealing with a small number of data and don't have very deep neural networks, that help with data loss.
Also, Dropout is used when we're trying to prevent overfitting. (When the model does very well on training but bad on test or evaluation set). But here the model isn't working as a perfect state-of-the-art and doesn't require dropout.

7.5 Model Architectures

As mentioned for model, I tried different models from LSTM, GRU, and RNN. Other than this, I used GRU and LSTM both which different settings such as different number of hidden layers, with or without dropout, etc. The model has been implemented in a way which is completely modular and every aspect of the models architecture can be determined by the hyperparameters and this design has been used for all 3 model type (RNN, LSTM, and GRU). This implementation helps on experiment and chaining the hyperparameters and not making mistake when doing so.

8 Result

Evaluation is scored based on perplexity. The less the better. Perplexity as it can be seen below, is weighted geometric average of the inverses of the probabilities which is used for evaluating language models. Here since we have a language model, we're using perplexity to evaluate the performance of the model.

$$Perpelexity = \exp\left(\sum_x p(x) \log_e \frac{1}{p(x)}\right)$$

As it can be seen on Table 3, I used different architectures with different number of hidden layers after getting the approximate best dropout and learning rate based on previous experiments and used that on different architectures and the results are as follows.
GRU didn't have a very good result and the perplexity increased for each epoch and in the end also got a high perplexity (high is bad here). RNN also was the second architecture which was tested and based on the test, it got

By using perplexity as the evaluation metric, the model is able to achieve a score of **107.19** on `learning_rate=0.001` and LSTM architecture. The reason for this is that, since we're doing a unsupervised learning, the model doesn't know which one is the best path and by giving it lower

=pt

Number of Layers	Perplexity
LSTM 4 layers	110.31
LSTM 3 layers	111.40
LSTM 2 layers	112.39
GRU 2 layers	304.78
RNN 2 layers	301.32

Table 3: Different architectures and their overall perplexity on the test set on lr=0.01

Dropout Rate	Perplexity
0.2	107.19
0.5	117.41
0.75	132.57

Table 4: Dropout influence on the perplexity for LSTM model, lr=0.001, with 2 hidden layers

learning rate, we can increase the chance of getting the best path.
This is a good score for a model that was trained on a small dataset (compared to terabyte data big models use today) without very intensive hyper parameter tuning and model optimization.

The result of using dropout on the model is shown on table 3. As mentioned in section 7.5, since here we don't have a deep network and a large dataset, dropout doesn't help very much and as seen in in the Table 3, with dropout `value=0.2`, the accuracy of the result improves but if we increase dropout rate to 0.5, the accuracy drops very quickly and noticeable.

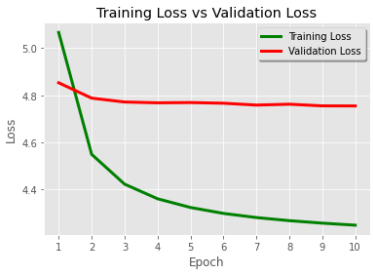

Figure 3: Training Loss vs Validation Loss

As shown in 3, we can see the training vs validation loss over the training time. and as shown in 4, we can see the Perplexity over the 10 epochs for the two datasets.

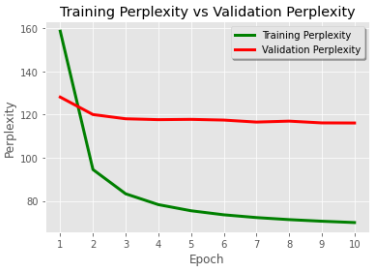

Figure 4: Perplexity

References

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

=pt