CS291K Report

---Poem Writer

Group Member:

Xin Liu

Shenghan Gao

Kai Yan

Introduction

Classical ancient poems are a significant part of China’s cultural heritage. Their popularity manifests itself in many aspects of everyday life, e.g., as a means of expressing personal emotions, political views or communicating messages at festive occasions.

The problem that we are trying to solve is how to make use of neural network to generate a Chinese poem. The Chinese poems are organized in certain formats, and quatrain is one of the best known ones. A poem must meet some structural, phonological and semantic requirements. If we take a quatrain as an example, the final characters of the second and the fourth line must be rhyme. On the other hand, the poems must follow a tonal pattern. Every Chinese character has a tone, which is Ping or Ze, and a poem should follow a certain tonal pattern.

The Chinese poem generation program we are trying to build works this way: the user choose some of the keywords as the first word of the poem, and the neural network should be able to finish the rest of the poem based on this word. The generated poem is expected to exhibit accurate use of the phrases and express a certain kind of meaning which can be conveyed to the reader.

Implementation

We used TensorFlow to implement our neural network. TensorFlow is a very convenient tool as we can use their well packaged models without bothering building our own. So it is indeed very efficient and saved us a lot of time.

And the language we used is python.

Training Data

Our corpus contains about 140,000 quatrains from Quan Tangshi (Complete Tang Poems) and Quan Songshi (Complete Song Poems).

Data preprocessing

To use the data we currently have, we have to do some work on them first. We firstly have to process the data so that we can have a poem every line. We have to read these poems and generating two maps that are useful for us during the training process. We need to allocate a unique number to every Chinese character. Considering the fact that a lot of characters that appear in the poems are actually not common and may appear in only in one or two poems. To keep the size of the maps generated, we count the number of times that a character appears and pick top 2000 most common characters in all poems and replace all other characters with an asterisk in the poems. By using this strategy, when we sample the sequence, we will only choose characters from these 2000 characters. To use these poems to train the model, we want to keep the length of the poems the same so that we can feed the batches of training data into the model with the same size. The way we use to keep the length of different poems the same is to put a caret in the front of every poem to indicate that this is the start of the poem and put a certain number of dollar sign at the end of a poem to indicate that this is the end of a poem. The number of dollar signs that we would put at the end of the poem is determined by the longest poem in the dataset and we add only one dollar sign at the end of it, so the size of each line in the dataset is the longest poem length + 2, including one caret at the front and one dollar sign at the end. After we have added caret, dollar sign and replace uncommon characters with asterisks in the dataset, we can use this preprocessed data to train the model.

The most frequent 20 characters in our corpus:

[0] ， [1] 。 [2] 不 [3] 人 [4] 一 [5] 风 [6] 山 [7] 无 [8] 有 [9] 来

[10] 日 [11] 花 [12] 春 [13] 何 [14] 天 [15] 云 [16] 年 [17] 时 [18] 中 [19] 自

Here, ‘，’ and ‘。’ are traded as ordinary characters which appears most frequently.

Architecture

For the architecture of our neural network, we used LSTM network. LSTM network is very special kind of the Recurrent Neural Networks. LSTM works much much better than the standard RNN. Almost all exciting results based on recurrent neural networks are achieved with them. The advantage of LSTM is that they can avoid long-term dependency problem. Remembering information for long periods of time is their strong suit. And more amazingly, they can selectively forget something which won’t be needed any more.

Since the goal of our project is to make the neural network learn to write poems, in which each word is created depending on all the previous words, it is obvious that we need the neural network have a long memory. So that is why LSTM particularly suits our project.

And to pursue better performance, instead of using just one LSTM cell, we used two.

Sampling

During the training process, we use the saver in TensorFlow to save our trained model on disk so that when we want to sample a sequence, we can restore the model and use the it to do the prediction. The way we feed the data into the model during sampling is different from the one we use for training. During training, the batch of data we feed into the graph is of size batch\_size \* num\_steps, however, the batch of data we feed into the graph is of size 1 \* 1 when we are sampling. When the user enter the character which he or she would like to start the poem with, we will add a caret in the front of the input. After that, we will first start with the zero state and calculate the state of the cells based on the characters provided by the user and this calculated state is used together with the last character in the user’s input to sample the next character. We choose the character with the highest probability. If the character chosen is an asterisk, which means that it has sampled an uncommon character, we choose another character instead according to the probabilities we calculated before. If the character chosen is a dollar sign, which means that the model predicts that this poem comes to the end, but if we need a longer sequence, we will do the same thing as when the asterisk is chosen. Then according to the previous state and the character, we can update the cell states and then sample the next character.

Model Building

Building the training model maybe the most important part of any neural network project, because how to train and training result directly determines the quality of the outcome of the neural network.

Since we are using TensorFlow, so graph flow is created beforehand, and then we run session in a train function.

The flow is like this: first, we define the input data and targets placeholders, and then build MultiRNNCell consisting of two LSTM cells, initialize the state of the cell, then we are ready to feed the data into the cell. Feeding data is a tricky part. What we do is to get a batch, which is a matrix of batch size rows and number of steps columns. Then get their word embedding, feed batch size words and the initial state into the cell, get the output and outcome state, feed next batch size words and the outcome state into cell again, continue doing this for number of steps times, and then compute loss, cost, gradients, and update parameters. During the feeding part, each step of cell deals with batch size words. When compute loss, we also need to feed in the targets, which are the next words of the input words. Then this process can be repeated over and over in order to improve the performance.

When we are deciding how the data should be fed into the graph, we have though about several different possibilities:

1. To split a line according to the number of steps to unroll. In this case, when we get a batch of data, we will feed two non-overlapping sequences to the model for training. For example, if the number of steps is set to 20, if in one row we have a sequence x consisting of character indexed in [0, 20), so the predicted sequence y consisting of character indexed in [1, 21). After we have fed this x and y into the graph, we choose another sequence x from index 20, which does not overlap with the previous x. We may carefully choose the number of steps so that we can make most use of a row of data.
2. To choose the sequence in a row sequentially so that we can use many sequences in a single row. For example, if the first x consists of character indexed in [0, 20), the second x would consist of character indexed in [1, 21) and so on. However, it is not hard to realize the fact that it would take a much longer time to train the model because the number of sequences used in a row is much larger than the one if the previous method is used.

Results:

The following is table showing the relationship between training time and training perplexity.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Begin | 1 min | 5 min | 30 min | 1 hr | 5 hr |
| Perplexity | 1.463 | 1.398 | 1.350 | 1.340 | 1.311 | 1.284 |

And below is the critical parameters.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Hidden  Size | Num  Steps | Batch  Size | Dropout | Layers | Vacab  Size | Learning  Decay |
| 200 | 13 | 20 | 0.5 | 2 | 2000 | 0.5 |

Here is an example of the sampled sequence. As we can see, when this model is used, during sampling, it knows what a correct line in a poem looks like so that it knows when to put in a comma or a period.

“秋游未鹃一回难，离知自雨难春寺。四牧尤云盛一棱，还有春华诗出早。”

On the other hand, if the training data are fed using the second way mentioned in Model Building section (to choose the sequence in a row sequentially), the results of sampling is shown. The advantage of this problem is the contents of the line in the poem make more sense. The reason for that is because we are feeding many character sequences into the graph more than once because the overlapping sequences. However, this method does not ensure the format of the line of the poem sampled.

So it can produce a poem like, “山林，解觉，人间免与浮丘贤。绿云渐有问天下，沙石，绿云两树省历云”. Clearly this poem’s format does not match the requirement of a quatrain.

Challenges:

1. The working mechanism of RNN and LSTM can be pretty complex, and to soak it all in, process them and come up an idea of how to use them takes us a long time.
2. RNN is very hard to train. It takes a long time to train the model well.
3. Deciding how to train the neural network can be tough, since we do not have any previous experience of predicting word sequences.
4. TensorFlow is new compared to other libraries that can be used for machine learning research. The APIs for RNN is not easy to use because documentations are not complete for them.

Possible Improvements:

1. Improve the model to take the meaning of the characters and words into consideration. Let the user give a topic, and the main content , emotion, and style of the generated poem will convey to this topic.
2. Use some extra data to control Ping or Ze of the character chosen for the next character instead of choosing them only according to the calculated probabilities.
3. The computational power for us is limited for us, one possible way we can improve the training is that we can modify our code so that we can take advantage of GPU to accelerate the training process.