**Summer research program report**

**What I leant from this program:**

In this summer, I joined Peking university’s deep learning summer research program. The whole program was split into two parts: learning and project. For the first half of this program, I leant a lot of useful and interesting deep leaning models. Here I list some models and algorithm I think is very important and useful for deep learning study.

**MLP:** MLP is the most classical neural network model. It contains input layers, hidden layers, output layers. Perceptron between different layers are fully connected and every connection has its unique weight to be trained. For every single perceptron, it contains a transfer function, activation function and bias. The transfer function is used to calculate the weighted average of every multiplication of weights and inputs. Activation functions are non-linear functions. We introduce activation functions to neural networks because the output of transfer function is always linear, it could not solve any non-linear problems.

**Backpropagation & gradient descent:** Backpropagation is chain rule based algorithm used to calculate the gradient and we could thus use gradient descent to optimize the parameters. Before backpropagation, we do forward propagation to calculate z and a for each perceptron.

**Convolutional neural network**: CNN including convolutional layer, pooling layer and fully connected layer. There is a benefit compared CNN with classical fully connected neural network. For fully connected neural networks, every connection has its own unique trained weights. However, the weight of CNN is shared: the whole image shares the same filters and thus the number of trained parameters is significantly lower than fully connected neural networks.

**RNN**: Different with other model such as CNN the size of input could be k-dimension, the input of language models should only be a one-dimensional sequence. For the very first step, we need to convert sentences into vectors just like we convert image into matrix. Classical RNN is a long-term memory network, the function of RNN is tanh(Whhht-1 + Wxhxt) where Xt stands for the new input, w stands for the weights and ht-1 stands for the previous output. This is very straight forward that we can see RNN is a long-term memory-based network: the output is affected not only by the new input but also by the previous output..

**Project 1:**

**LSTM based Chinese poetry generation**

Code: <https://github.com/zhukson/Peking-University-summer-program/tree/main/LSTM%20poetry%20generator>

**Introduction:**

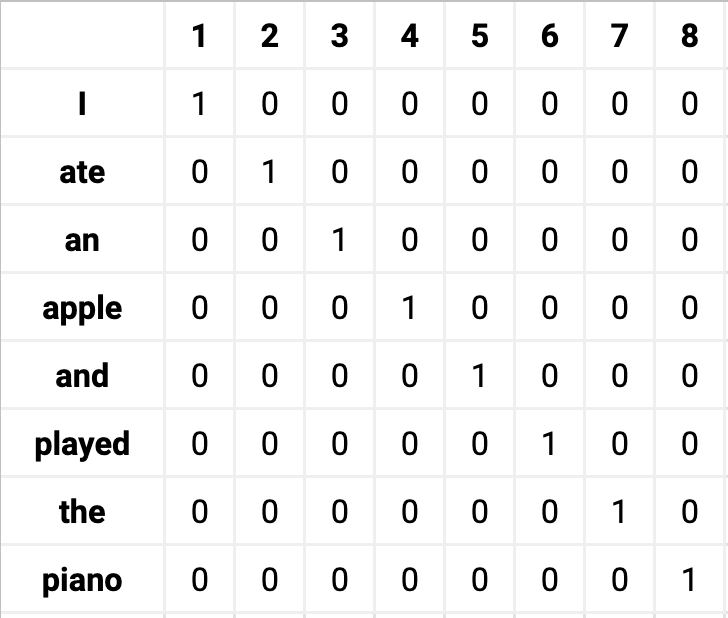
This is a LSTM based generator which could generate 5-words, 7-words poetry and acrostic (a verse in which the initial characters of the lines form a word/phrase) 5-words, 7-words poetry automatically.

**Data preprocessing and Word Embedding:**

Before putting data into the LSTM model, the conversion of words into one-dimensional sequence is required.

* **One-Hot encoding:**

One-Hot encoding is a discrete representation of a word. Suppose we have k words in total. It is a vector of size k with only one 1 and k-1 zeros.

****

**Figure 1.1: one-hot encoding representation**

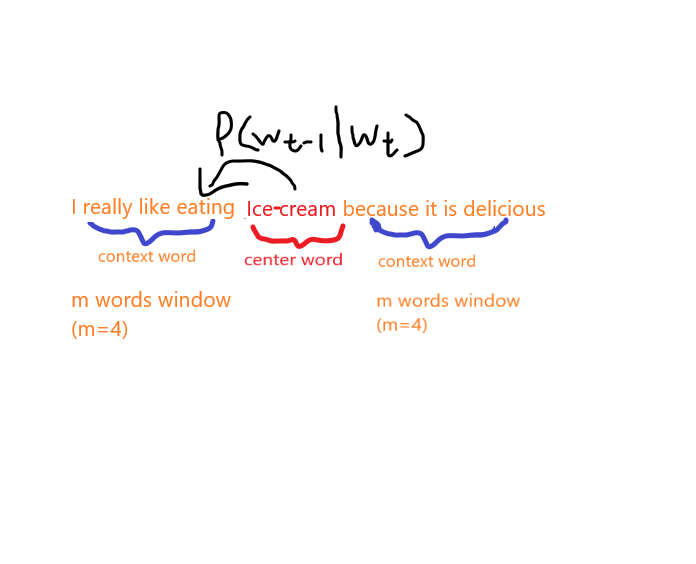
However, there is a big problem exist that if we have a lot of words in our datasets, the dimension of one-hot encoding vector could be very big. Since I have over 4000 poetries in total, the word number is too much and One-Hot encoding may not be a very good choice for me in this task.

* **Word2Vec:**

Word2Vec is a distributional representation of word and the dimension of this distributional representation vector is smaller (kind of dimensional reduction) than the discrete representation of One-Hot encoding.

* **Skip-gram:**

Word2Vec has two major models. One is called skip-gram which is to predict context word based on the center word and the other is called CBOW which is to predict the center word based on the context word.

****

**Figure1.2 the probability of Wt-1 to be the context word of Wt.**

Based on Figure1.2, We could build the loss function of skip gram:

Where the first ∏ represents we need to iterate the words from the whole list and let it to be the center word. The second  ∏ represents we need to iterate the context words in the m words window surrounding the center word. However, this function is very hard for us to do the partial derivative and thus to do the backprop and optimization. Here we use negative log Likelihood to simplify it and thus it become:

= = SoftMax()

refers to the vector representation of center word, refers to the vector representation of the context word, the transportation used here refers to the inner production of these two vectors. If the center word and the context word is more similar, would be bigger.

**图表

描述已自动生成**

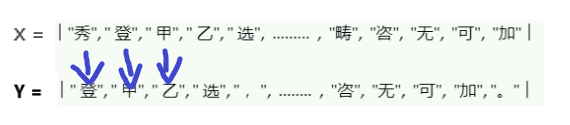
**Figure1.3 the main procedure of the skip-gram model.**

From figure1.3, we can see the main procedure of skip-gram model. The model first constructs a one-hot encoding vector of a center word and multiply it with the word embedding matrix to get the word embedding vector of this center word. The red word-embedding matrix is the leant parameters based on the loss function we discussed before.

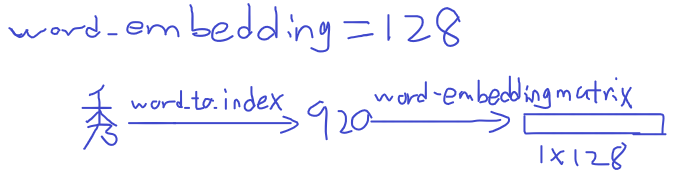
**Data Set Construction**

After we got the word-embedding matrix, we could also get the word\_to\_index (key is the word and value is the index) based on the word-embedding matrix. Hence we could initialize our dataset and prepare for the data loader.

First, we need to construct X and Y List, Y is the prediction of X.



**Figure 2.1 construction of the X and Y list**

We could then put these words in these lists into the word\_to\_index function to get the index for each word and based on this index to find the according word vector in the word-embedding matrix. 

**Figure2.2 The procedure of converting a Chinese kanji into a word vector when word\_embedding which is the column of the word\_embedding matrix is 128,**

**LSTM model:**

In the “What I learnt from this program” section, we discussed RNN model and explain how it realize the long-term memory function. However, RNN has a problem that since it is a long-term memory network, the more words it memorizes, the more part of earlier words would be forgotten. For example, an RNN model has already memorized a lot of words, the first eight words is “I have lived in China for five years”, the last three input is “I can speak”. Now based on the context, we definitely want the model to generate “Chinese” after “speak”, but the model has already forgotten the word “China” and end up with generating some bad result.

Based on this problem, LSTM was created. LSTM’s full name is called Long short-term memory. Unlike classical RNN which only have long term memory, LSTM have both short and long term memory.

图示

描述已自动生成

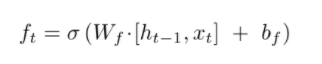
**Figure3.1 The LSTM model**

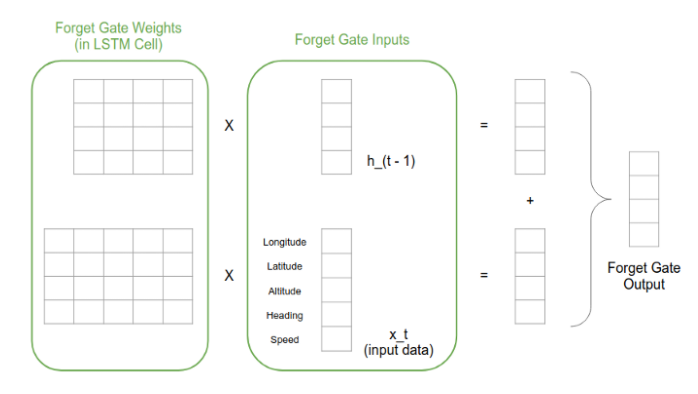
From figure3.1, Ct is the cell state (long-term memory) of LSTM, which means to Encode data from **all** previous time-steps. Ht is the hidden state (short-term memory) of LSTM, which means to Encode data **only** from the data from previous time-step t-1. However, Hidden state is not equal to the prediction or output, because hidden state could still be processed to obtain more meaningful data.

LSTM has 3 gates. A forgot gate () which apply a sigmoid function(Range(0,1), 0 means completely forget, 1 means keep) to control what words we want to forgot in the long term memory.

We also get input gate（）, which also apply a sigmoid function to decide which words we want to add from the candidate ()into the long-term memory. Finally, we get an output gate（）, which also apply a sigmoid function to control our output.

There are some hyperparameters of LSTM I’d like to mention. One is called the Hidden size(or num units). Take forget gate as an example. Here is the formula of how forget gate works.



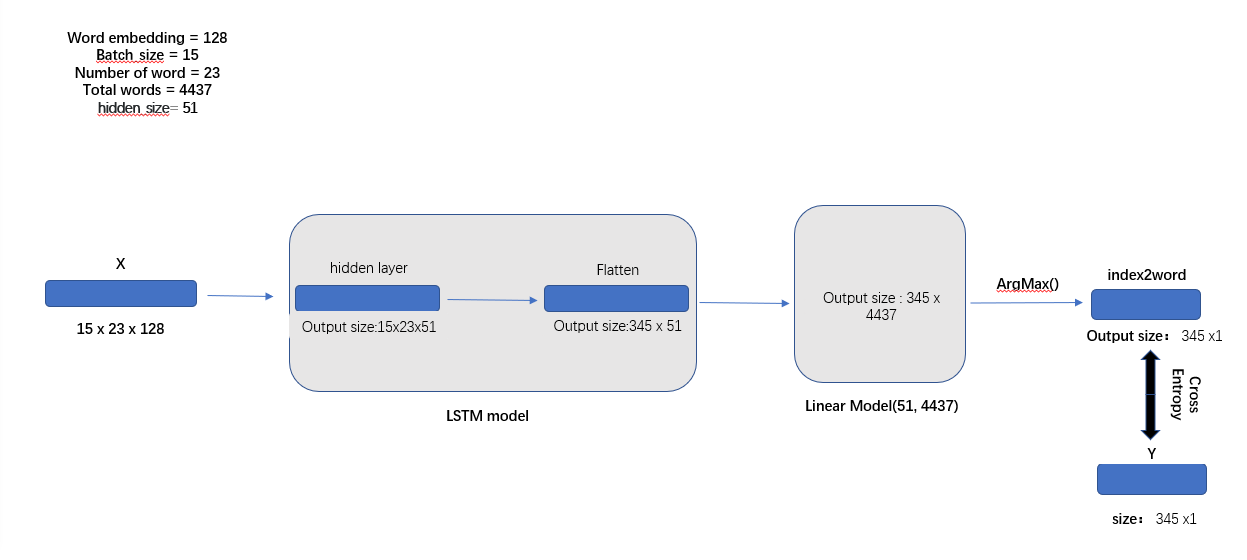
We can see before we put our input and hidden state into a sigmoid function, we need to multiply it with the weights. Hidden size means the dimension of this weight.

**Figure3.2 the relationship between hidden size and weights**

From Figure3.2, We can see the size of the weight for hidden state is (hidden size \* hidden size) and the size of the weights for input data is (hidden size \* dimension of input data).

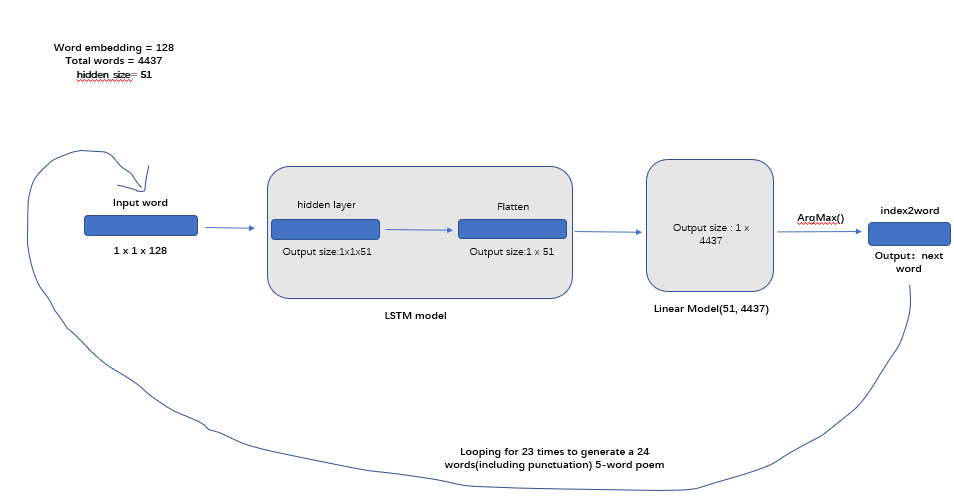
Another hyperparameter is called number of hidden layers which means how many LSTM model we stacked together.

**Poetry generation process:**

****

**Figure3.1 the training process of auto generator**

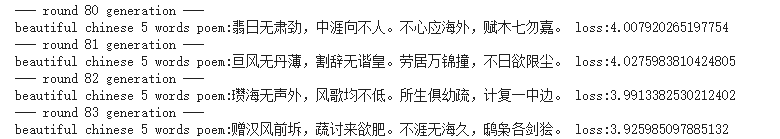
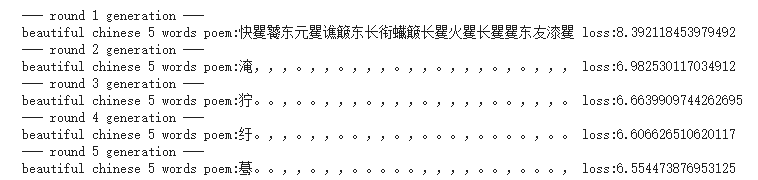
X and Y are lists we constructed and discussed in the previous part. Linear model + ArgMax() here is to find out which word in our whole datasets is the most relevant to the prediction result got from LSTM model. Then we could get the index list of these relevant words and calculate the cross-entropy Loss with Y lists to do further optimization.

****

**Figure3.2 the generation process of auto generator**

There are only a few differences between generation and training. One is the input size, when we do generation, the input is a single word with size (1 x 1 x word embedding). The other different is after we find out the most relevant word’s index, we need to put it into the index2word dictionary to get the word and then use the index to find the according word embedding representation in the word embedding matrix as the new input word.

**Generation result:**



We can see it successfully generate 5 words poem with correct form after some epochs.

**Project 2:**

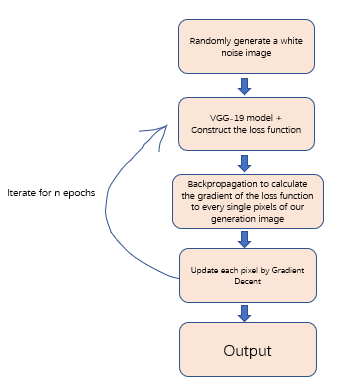
**VGG-19 based Style Transformer**

**Code:** **https://github.com/zhukson/Peking-University-summer-program/tree/main/Style%20Transformer**

**Introduction:**

This is a VGG-19 based Style Transformer which could generate image which combine the style from one image and the content from another image.

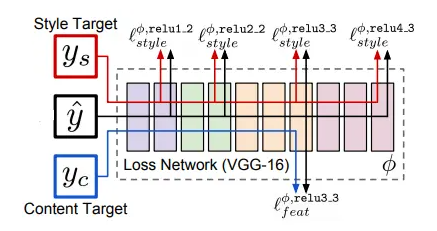
**Main Procedure:**

****

**Figure 1.1: The main process of style transformation.**

From figure 1.1, we can see our primary goal for this generation process is to construct our Loss function to continuously optimize our generated white noise image.

**Loss function:**

****

**Figure 2.1: The main process of style transformation.**

For our task, we used pretrained VGG-19 model with pretrained weights and freeze these weights during our training process. This is because the only parameter we need to train is the pixels of generation image. From figure1.1, we can see we need to put both style image, content image and randomly generated white noise image into the VGG model to compare their feature outputs from certain layers to do further parameter optimization. Hence, we could construct the Loss function for this task:



It is very straight forward that the whole Loss function is constructed with two parts: the Content Loss and the Style Loss. Where are the weights of style Loss and content Loss.

**图示

描述已自动生成**

**Figure 2.2: image reconstruction based on feature maps of different layers**

**Content Loss:**

For content loss, we only need to compute the differences between the feature maps of generation image and the content image. We use squared-error loss to represent this difference. Therefore, The Loss function could be like:



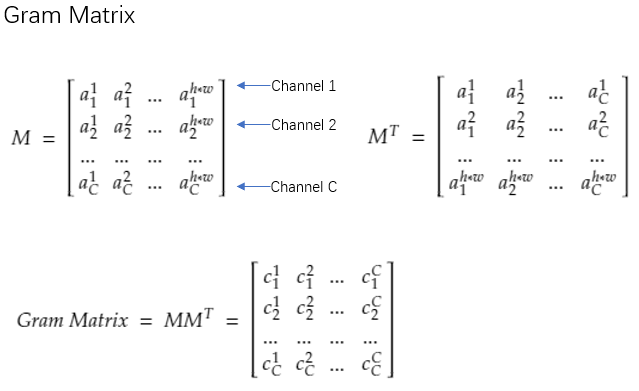
Where refers to the content image, x refers the generation image, and are their feature maps respectively, refers to the layer and the 1/2 is used to simplify the partial derivative calculation. It is very important to know what layer we should used to construct this loss function. From Figure 1.2, we can see the deeper the convolutional layer, the more the content get lost. Thus, we used the 3rd convolutional layer of VGG-19(16 convolutional layer in total) to construct our content loss function.

**Style Loss:**

For Style Loss we used a Gram Matrix to represent one image’s style.

**Gram matrix:**

Suppose we have a feature map M with size (C, w, h), and we put this feature map into a flatten layer to get a new size (C, w\*h). Then we multiply it with M’s transpose (w\*h, C) and finally we could get a matrix MM^T with size (C, C). Each element in this matrix is the multiplication of two feature map channels.



MM^T is called the Gram matrix which could represent the linear correlation between two different features. If two features are more correlated, the multiplication result would be bigger. If the two features are not correlated, the multiplication result would be a negative number. Based on this Gram Matrix, we could happily represent different feature maps by style and no longer need to consider any pixel information. Therefore, the loss function could be the differences between the Gram matrix representation of generation image’s feature maps and style image’s feature maps. This the Style loss function for layer l:

图片包含 文本

描述已自动生成

Where N is the number of channels, M is width x height. G is the gram matrix for generation image, A is the gram matrix for style image.

We need to determine which layer we should used to calculate the loss function. From Figure1.2, we can see for relatively shallow layer, the extracted feature is very basic features like the line and edge whereas for deeper layer, the extracted feature is more abstract. Therefore, we want to combine all these features together and the Total style loss function could be:

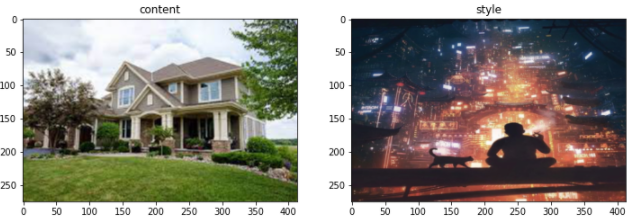
文本

描述已自动生成

Where w refers to the weight for different layers.

Now, since we have already constructed the most important Loss function and thus we could base on the main procedure from Figure1.1 to train our generation image.

**Generation Result:**

****

**Figure3.1 the content image(left) and the style image(right)**

For the total loss function, I set the content weight into 0.5 and style weight into 0.5. For the style loss function, I put more weight for the deeper convolutional layer to consider more of the abstract features instead of some basic line or edge features.

****

**Figure3.2 generation result after 500 epochs**

This is the final result after 500 epochs .

**Acknowledgement:**

Thanks to Professor Lin and TA Wang for directing me with the right direction and offering me with a lot of instructions. Before I join this program, Deep Learning is almost a new world to me. With your great help, Now I have already gained my own understanding to this field and became more confident for my future deep learning study.