Machine learning Final Project : Captcha Image Recognition

SeokHyeon Jeong
Seoul-national university
sh102201@snu.ac.kr

Abstract

CNN and LSTM are mainstream of Deep Learning. We want to solve the Captcha problem with CNN and LSTM. Captcha is OCR image with different length of word. The data type is image, so you may use CNN. If you only use CNN, you can not categorize characters well because the length of training datasets and test datasets are different. Otherwise, LSTM can handle variable length data. For these reasons, we used a method of combining CNN and LSTM. From data processing to training, test, we have gone through quite a few trials and errors in forecasting. Various methods were used to increase. My test results for char_correct and word_correct is follows.
char_correct : 96.9% word_correct: 81.7%

1. Introduction

Deep Neural Network(deep learning) has revolutionized engineering as a whole. Business, Society and our lives are becoming digital transformation, and Deep Neural Network is the main role. Computation problem was serious for Deep Learning in the past due to hardware. Current, with the help of GPU, we can use it for our research and business.

Image recognition and Natural Language Processing are two applications to lead the renaissance of Deep Learning. CNN(Convolution Neural Network) significantly increased the performance of image recognition. RNN(Recurrent Neural Network) and LSTM(Long Short-Term Memory Models) also explosively increased the performance of NLP. If we use these tools, performance is better than human.

In captcha recognition problem, we should classify each characters of captcha image. The data type is image, therefore we would use CNN(Resnet or assign3 model) and LSTM(LSTMcell in Pytorch).

1.1. CNN

It is a type of Neural Net and has very good performance for image recognition. The CNN consists of a convolution layer, a full connected layer, a ReLu layer, and so on. It requires a lot of calculation. The CNN used in this paper used a ResNet and self-produced CNN model consisting of inception module.

1.2. LSTM

The longer the data is, the more interaction information between cells disappears when training is performed in the RNN technique. It is called a gradient vanish. There is an intuitive solution, but changing the structure is a more common solution. It is to separate the hidden state from the cell state and let them learn. That's the LSTM technique.

The core of the LSTM is the cell state. The cell state is passed through the whole chain like a conveyor belt. LSTM can remove and add information to cell state with gates, sigmoid layers. Standard LSTM has 3 gates which called forget, input, output. Forget gates decide what information to throw away from cell state. Output gates decide what parts of cell state to output. Input gates decides which values to update.

LSTM has two advantages over RNN. The first is the element-wise multiplication, i.e. the matrix multiplication is efficient. And because backpropagation is done through cell state, the gradient is super highway. It is similar to the effect of skip connections in Resnet. GRU(Gated recurrent unit) is simplification version of LSTM. It combines forget and input gates into “update gate”. Cell state and hidden state are combined into one state.

1.3. Captcha recognition problem

Our main problem is Captcha recognition problem. Each captcha image has a variable length of characters. Therefore we have to consider the variable length captcha image. If you look at the captcha image dataset, there are 10000 training images with lengths of 2 to 5 and 1000 test images with lengths of 2 to 7. In other words, because the length of the training image and test image are different, it cannot be classified with only CNN.

Therefore, it is necessary to use LSTM that can also be learned about variable lengths. I solved the above problem with a combined model of CNN and LSTM. First, we designed LSTM forward structure and connected Resnet and LSTM to test train. Next, instead of Resnet, it connected its own CNN and LSTM.

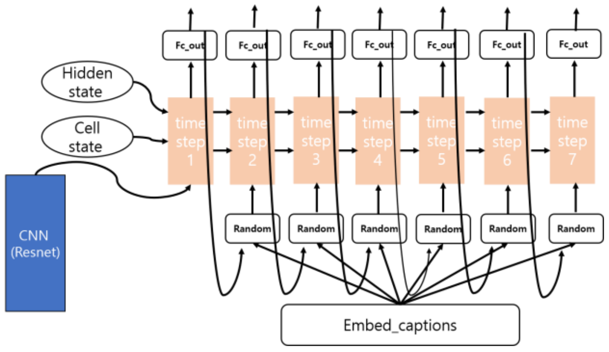

Figure 1 CNN-LSTM architecture

2. Method

To solve problem 1,2-1, 2-2, 3, we used various methods. The sections related to problem 1, 2-1 and 2-2 are explained in this "Method" section, and the problem 3 related to hyperparameters are explained in the "Experiments & Discussion" section. I made the model like Figure 1.

2.1. Data processing

When using the Dataset function, the training set data were not sorted, so the **library natsort** was used to sort it. If you look at the size of the label_oh, it is in the form of batch-size * 259. The argument **captions** of the lstm_cell shall be entered as the ground truth label_oh. Therefore, reshape and argmax the label_oh when the model is training. The code is as follows.

label_oh1 = torch.reshape(label_oh, (batch_size, 7,37))
label_oh2 = torch.argmax(label_oh1, dim=2)

Likewise, it is treated in the same form when testing.

2.2. LSTM cell forward

Let's take a look at it in a Figure 1.

We cannot use ground truth of test set. Thus, to distinguish between train forward and test forward, a factor called flagship was defined.

Therefore, if the flagship is train, the input of the LSTM_cell is randomly inputted either the embedded caption or the previous step output as shown in Figure 1.

In the case of a test, the structure simply inserted the previous step output into the next step input. Finally, the output for each step was saved in **"output".**

2.3. Use Resnet, own CNN, Connect CNN and LSTM

Let's take a look at own CNN architecture in a table.

CNN (epoch : 10)
Conv2d(1,64, kernel = 5, padding =2)
Maxpool2d(2,2)
Batchnorm(64)
Relu layer
Maxpool2d(2,2)
Inception(64,56,40,32,60,32,8)
Maxpool2d(2,2)
Inception(128,56,40,32,60,32,8)
Maxpool2d(2,2)
Inception(128,200,160,140,240,120,52)
Maxpool2d(2,2)
Linear(512*5, 512)

Table 1 CNN architecture

The last block in the CNN structure was resized to 512 to be inserted as the first input to the LSTM cell. The way to connect CNN and LSTM is simple. The output of CNN is the **feature** argument of LSTM forward.

In Resnet case, we can get two arguments from forwarding, **out and features**. We take **feature** argument and reshape it to become input of LSTM_cell. The finished model is the same as **figure 1**.

2.4. loss function and optimization

To combine CNN and LSTM, we added the LSTM optimize function and LSTM step function respectively. The code is as follows.

lstm_optim = torch.optim.Adam(lstm.parameters(),lr = 0.001)
lstm_optim.zero_grad()
lstm_optim.step()

3. Experiments & Discussion

To increase accuracy, the hyperparameter must be adjusted.

3.1. Learning rate

When experimenting with reducing the learning rate in the optimize function, the more epoch, the higher the accuracy. But it takes a long time to run a lot of epoch. It's called trading off. And even if the leading rate became too small or too large, the learning did not go well. Thus I applied the leading rate 0.001 to the final model.

3.2. CNN feature(batch size, fc layer output)

Reducing or increasing the number of batch sizes or outputs and inputs of layers only changed the memory of the model, but did not have a significant impact on accuracy itself. Therefore, the conditions of batch size = 8, fc_in, fc_out given initially were used.

3.3. Input ground truth and teacher forcing train

The original LSTM architecture uses the output from the previous phase again as the input from the next phase. But it takes a lot of time to learn.

Therefore, we took the method of give input as training ground truth. That way, learning works well. However, it lacks performance because it uses original LSTM method when testing. Therefore, the method we used is **teacher focusing.** With a certain probability, I chose to insert the input ground thrust or input the output of the shear meter. Therefore, performance has improved slightly. The code is as follows.

teacher_forcing_ratio = 0.4
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

3.4. Output Soft max processing

When designing the lstm forward function, I tried to normalize the last output by using softmax. The training speed was slow to use. However It affect the accuracy very much. When we test the result, we don’t use softmax.
The code used is as follows.
outputs = self.softmax(outputs)

3.5. Changing loss function

I tried to use cc loss function, cross entropy function, and so on, but I couldn't use it because there were continuous errors. So I used a given loss function.

4. Conclusion

Test results for char_correct and word_correct is follows.
char_correct : 96.9%
word_correct: 81.7%

If you look at the results, you can see some first letters were not correctly predicted. Therefore char_correct is high, but word_correct is low. If you think about other ways, you can see that CNN is not learning properly. Therefore, it is possible to try to learn not only the loss of the entire structure but also the CNN and LSTM respectively when training.

It can be seen that the predictions for the remaining values are almost accurate. After this project, I want to study image captioning a little more.

References

[1] C, Szegedy, W , Liu Y Jia P Sermanet, S Reed, D. Anguelov, D Erhan, V. Vanhoucke, and A.Rabinovich, “Going deeper with convolutions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun “Deep Residual Learning for Image Recognition, 2015