Report for Assignment_4

Introduction

Introduction of CNN & RNN

CNN: A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

RNN: A recurrent neural network (RNN) is a type of artificial neural network commonly used in speech recognition and natural language processing (NLP). RNNs are designed to recognize a data's sequential characteristics and use patterns to predict the next likely scenario.

Introduction of Assignment

In this assignment you are going to re-do the text classification task in assignment 2 with FastNLP and PyTroch with more powerful tools, RNN and CNN.

Part 1: CNN Text Classification

Introduction of CNN model

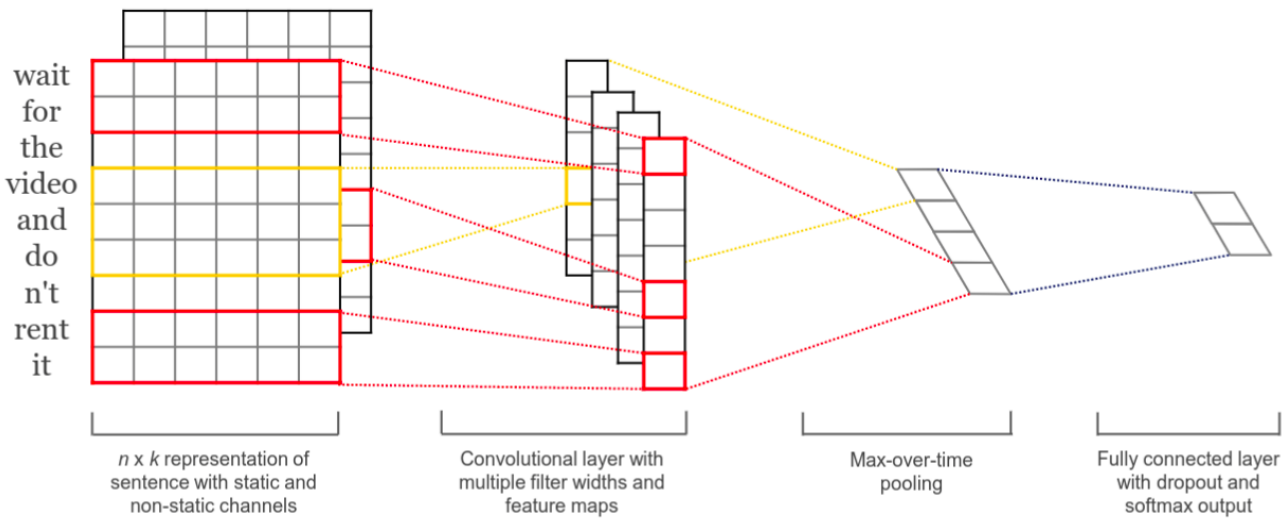


Figure 1 Model architecture with two channels for an example sentence

The model is shown in Figure 1. First, let $\mathbf{x}_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to the i-th word in the sentence (where k is the embedding size of the embedding layer). Then, for a sentence of length n (padded where necessary), we have a $n \times k$ matrix X. $X_{i:i+j}$ refers to the concentration of words $\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+j}$, which a are both represented in vector form. A convolution operation involves a *filter* $\mathbf{w} \in \mathbb{R}^{hk}$, which is applied to a window of h words to produce a new feature. For example, a feature c_i is generated from a window of words $X_i : i + h - 1$ by

$$c_i = f(\mathbf{w} \cdot X_{i:i+h-1} + b)$$

where b is a bias term and f is a non-linear function such as Relu or tanh. This filter is applied to each possible window of words in sentence $\{X_{1:h}, X_{2:h+1}, \ldots, X_{n-h+1:n}\}$ to produce a *feature* map

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$$

where $\mathbf{c} \in \mathbb{R}^{n-h+1}$. We then apply a max-over-time pooling operation over the feature map and take the maximum value $\hat{c} = \max\{\mathbf{c}\}$ as the feature corresponding to this particular filter. We have describe the process with *one* filter. The model uses multiple filters (with varying window sizes) to obtain multiple features. These features form the penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

For regularization, we employ dropout on the penultimate layer, which randomly dropping out a proportion p of the hidden units during forward-backpropagation.

Implementation

We implement the model with the help of FastNLP and PyTorch. I define a CNN with three filters to capture the feature. There're four layers in our model:

- The embedding layer: we use the function *torch.nn.Embedding()* which is provided by PyTorch
- The conv layer: we use the function *torch.nn.conv()* for convolution and *torch.nn.functional.relu()* for activation.
- The max-pooling layer: we use the function *torch.nn.functional.max_pool1d()* for extract the max value of every filter.
- The fully connected and softmax layer: we use a simply linear function *nn.Linear()*. Besides, we use *nn.Dropout()* to implement the dropout.

We train our model on *The 20 newsgroups text dataset*.

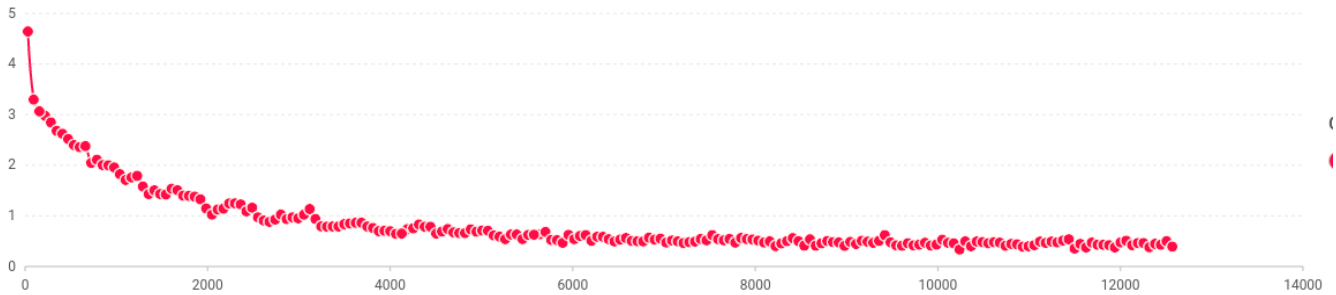
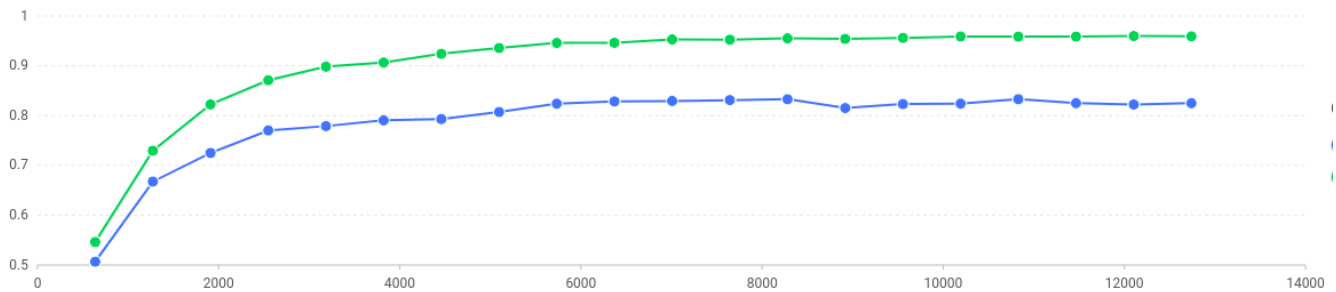


Figure 2 The loss curve on development set

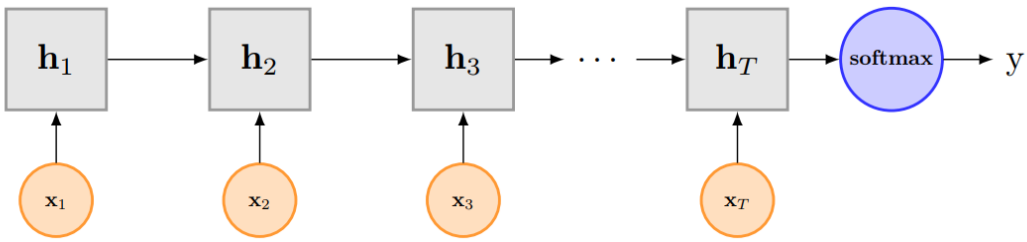


> Figure 3 The accuracy on test set and training set (Green curve: training set, Blue curve: dev set)

The figure 2 and 3 show that CNN model has a good performance on text classification assignment, which has the accuracy of 70.8% on test set. However, compared to the dev set and training set, which has the accuracy of 83.3% and 96.9%, the CNN model is a little bit over-fitting. Besides, We can find that the loss decrease quickly at the beginning, and the loss converge to 0.5 after nearly 4000 steps. We can conclude that CNN has a fair performance on text classification assignment.

Part 2: RNN Text Classification

Introduction of RNN model



> Figure 4 Recurrent Neural Network for Classification

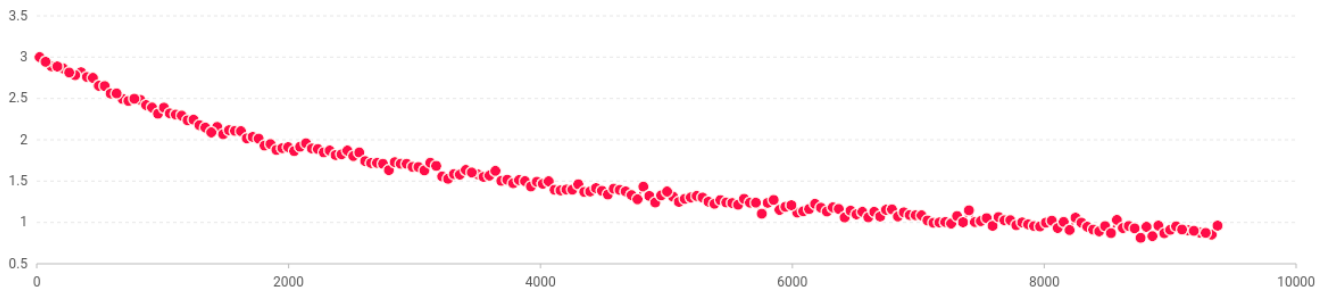
There's little difference between assignment 3 and 4. One is that we need every hidden state when we generate the tang poems, while we just need the last hidden state (output) when classify the text. Others is that we use LSTM cell in assignment 3 but use simple RNN cell in assignment 4. Like LSTM, RNN share the same weight in all cell. The model is shown in Figure 3.

Implementation

We implement the model with the use of FastNLP and Pytorch. There're three main layer in my *LSTM* model:

- The embedding layer: it is the same as CNN.
- The CNN layer: I use *nn.LSTM()* function
- The fully connected layer: it also use *nn.Linear()* mapping the output vector of CNN cell to prediction vector.

We train our model on *The 20 newsgroups text dataset*.



> Figure 5 The loss curve on development set

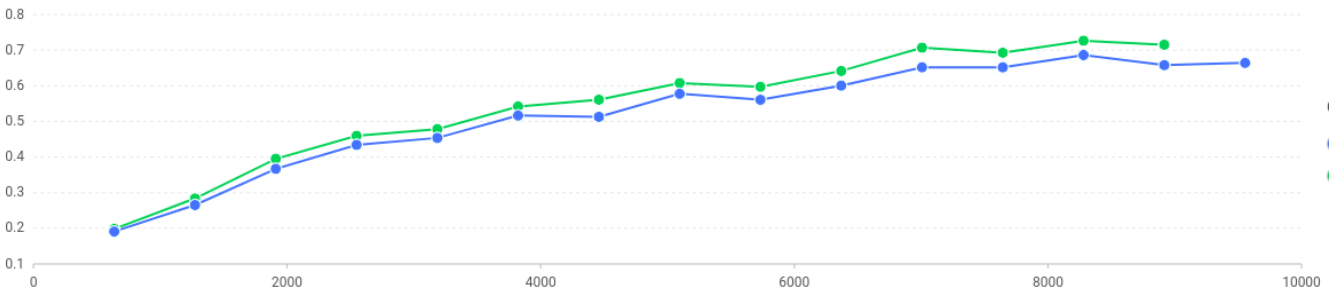


Figure 6 The accuracy on dev set and training set (Green curve: training set, Blue curve: dev set)

As the curve shown in Figure 5 that the loss decents slowly, the training of RNN costs much time, because of the long text in every training data. And its performance is worse than the CNN model. The best accuracy on test set is 63.8% and 68.6% on development set. The accuracy on training set is 73.0%, which indicate that the model isn't over fitting.

Part 3 : Thoughts about FastNLP

In general, the fastNLP is a useful tool for people who want to build a model structure for NLP problem. What impress me first is the inner class **Trainer**. It simplify our code, providing API for us to train our model conveniently. The template also visualize the train process. It calculates the training speed and predicts the train time, while gives a progress bar, which is easy for us to observe the train and check whether our model is bad or not. **Dataset** and **fitlog** class is also impressive. The former allows us to organize and process data easily, and the latter allows us to check the training result and the model. Also the **Vocabulary** class help me pre-process my data and build a dictionary for assignment.

However, there're something inconvenient when I use it.

- The tutorial is not helpful enough. It just simply introduces the function of the model, like where we need to use it, but not how we can use it. It lacks of many essential explanation of the class or function parameters. In most of the time, I have to see the original code to learn how it works so that I can use it properly. I used to learn Python library through reading their tutorial. Among them, I like Numpy and PyTorch most, because they list out all the detail about the parameters of their function and class. I think organizing the tutorial like them is a good idea.
- It hard to build a complete model without FastNLP. Except the Trainer, fitlog, Dataset, and Vocabulary, I need the help of PyTorch, such as LSTM and convolution layer, etc. Thinking that FastNLP is not a mature version and still has a long way to go, I consider perfecting the library and providing more useful models and modules is important.
- It isn't free enough for us to define our model and train process. Though *Callback()* in Trainer allow us to store the parameters with fitlog, or define our own behaviors, I often feel restricted when I use FastNLP. Sometimes I want to show more detail about my training process or design my training method like using different optimizer but fail. I think callback is a good idea for combining convenience and creativity together. I would appreciate it if I have more space to design my model and training process.
- Fitlog should show more detail about training process. The website is too simply and I can't get enough information about the training process. It would be better if it can provide more detail like the learning rate, batch size, etc.
- The last suggestion is that I always make some mistakes that the program went wrong after training. It would be annoying when I find I loss all the training statistic. When I want to make a prediction, I have to run it again and wait for a long time. Besides, due to the compute capability, my computer always halts when the neutral network is large, and I lost all the parameters again. I hope FastNLP can provide some useful method for us to reload our model and support the training after breakpoint.

In a nutshell, I hope FastNLP can be perfected day by day and become a popular tools for NLP one day.

Reference

1. Yoon Kim .Convolutional Neural Networks for Sentence Classification. $arXiv$ preprint $arXiv$: $1408.5882, 2014$
2. Pengfei Liu, Xipeng Qiu, Xuanjing Huang.Recurrent Neural Network for Text Classification with Multi-Task Learning. $arXiv$ preprint $arXiv$: $1605.05101, 2016$
3. Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification. $arXiv$ preprint $arXiv$: $1509.01626, 2015$