Assignment 04

RNN and CNN

Anonymous

Department of Computer Science, Fudan University

May 28, 2019

1 Overview

1.1 Introduction

In Lecture 7&8, we have discussed Recurrent Neural Network(RNN) and Convolutional Neural Network(CNN).

Convolutional Neural Network (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.

Recurrent Neural Networks (RNN) are a type of artificial neural network designed to recognize patterns in sequences of data, such as text, genomes, handwriting, the spoken word, or numerical times series data emanating from sensors, stock markets and government agencies. These algorithms take time and sequence into account, they have a temporal dimension.

In this assignment, we will apply them to a text classification problem.

1.2 Assignment Description

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

1.3 Outline

The rest of this report is organized as follow. Sec. 2 introduces two powerful models, CNN and RNN. After that, the text classification methods with CNN and RNN will be presented in Sec. 3 and Sec. 4. We outline the basic architecture of the network and then conduct experiments on dataset. In Sec. 5, we will present our thoughts and experience about using FastNLP. Hope this will be useful for the future development of FastNLP.

2 Introduction to CNN and RNN

This introduction section is referred to [4].

2.1 CNN

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers.

In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting a property of translation invariance.[1]

For details, I recommend you to read this course material[2].

2.2 RNN

A recurrent neural network (RNN) [1] is able to process a sequence of arbitrary length by recursively applying a transition function to its internal hidden state vector $\boldsymbol{h}_t$ of the input sequence. The activation of the hidden state $\boldsymbol{h}_t$ at time-step t is computed as a function f of the current input symbol $\boldsymbol{x}_t$ and the previous hidden state

$$\boldsymbol{h}_t = \begin{cases} 0 & t = 0 \\ f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) & \text{otherwise} \end{cases}. \tag{1}$$

It is common to use the state-to-state transition function f as the composition of an element-wise nonlinearity with an affine transformation of both $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. Traditionally, a simple strategy for modeling sequence is to map the input sequence to a fixed-sized vector using one RNN, and then to feed the vector to a softmax layer for classification or other tasks [2].

The gradient exploding and vanishing problem encountered during the training process of conventional RNN is addressed by invoking the gate mechanism.

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

[2] https://cs231n.github.io/convolutional-networks/

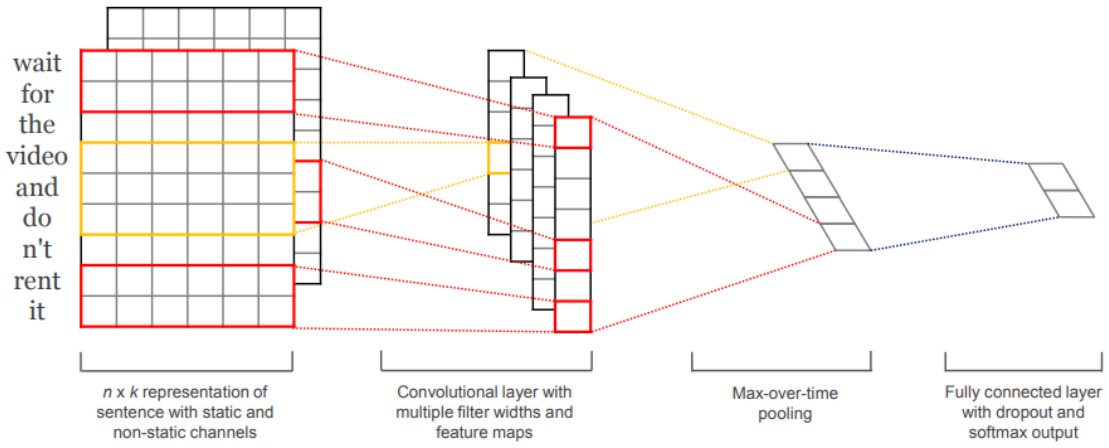

Figure 1: Model architecture with two channels for an example sentence from [3]

3 Model I: CNN Text Classification

We referred to the CNN proposed in [3] and reproduced the model for learning purpose. In this section, we give a brief introduction.

3.1 Theory

The model in [3] is shown in Fig. 1. First, they use a embedding layer to map a word to a k-dimensional word vector and use $\boldsymbol{x}_i$ to denote the vector corresponding to i-th word in the sentence. For a sentence of length n, they concatenate n word vectors on rows and thus get a $n \times k$ matrix $\boldsymbol{X}$. Then, a convolution operation is applied on this matrix. This operation involves a filter $\boldsymbol{w} \in \mathbb{R}^{h \times k}$, which is applied to a window of h to produce a new feature. That is, a feature c_i is generated from a window of words $\boldsymbol{x}_{i:i+h-1}$ by

$$c_i = f(\boldsymbol{w} \cdot \boldsymbol{x}_{i:i+h-1} + b), \tag{2}$$

where b is a bias term and f is a non-linear activation function. They apply this filter to all possible windows to get a feature map

$$\boldsymbol{c} = [c_1, c_2, ..., c_{n-h+1}], \tag{3}$$

with $\boldsymbol{c} \in \mathbb{R}^{n-h+1}$. Several output channels with different convolution filters and window sizes will be produced in this layer.

After that, they employ a max-over-time pooling operation over the feature map and take the maximum value $\hat{c} = \max\{\boldsymbol{c}\}$ as the feature corresponding to this channel. The idea behind this operation is to capture the most important feature for each feature map.

Finally, a fully connected layer and softmax output will be added after the last layer.

As for regularization, we add a dropout layer after the max-pooling layer, that is, randomly setting to zero a portion p of the hidden units during forward-back propagation.

3.2 Experiments

We have implemented this model with FastNLP. Since the *ConvMaxpool* is provided in FastNLP, the implementation is rather simple. We just need to describe our four layers in the model definition. And for embedding layer, the authors use word2vec in their methods, while we use randomized initialization in this task. The implementation source code is given in CNN.py. We train our model on *The 20 newsgroups text dataset*[3].

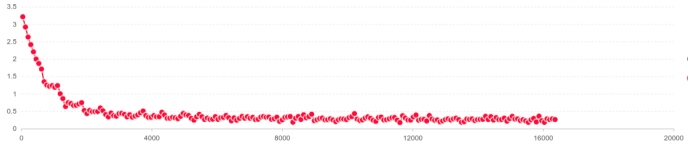

Figure 2: Cross Entropy Loss on trainning set in 25 epochs

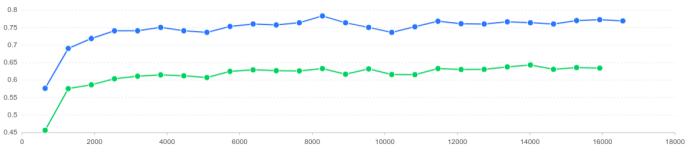

Figure 3: Accuracy of CNN model on the development set and test set during training process

Results are reported in Fig. 2 and Fig. 3. Fig. 2 illustrates the loss function variation on the training set. Notice that in the first 4,000 steps, the loss decreases rapidly whereas it seems to oscillate around a local optima after 4,000 steps. And the accuracy metric is invoked to measure the model performance on the test set. In Fig. 3, the blue curve depicts the accuracy on development set, while the green one depicts that on test set. Since the accuracy on development set does not improve, we use early stop and pick up the best model with accuracy 78.3% at 13-th epoch. Then, we apply this model on the training set and test set to measure its performance. It gets a 96.9% accuracy on the training set and gets a 63.3% accuracy on the test set.

[3] https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

4 Model II: RNN Classification

4.1 Theory

Since we have discussed the LSTM model in last assignment, we omit it here. The only difference between poems generating and text classification is that the latter only has one output after the last layer. A traditional recurrent neural network for text classification is illustrated in Fig. 4, where we omit the gates in each layer.

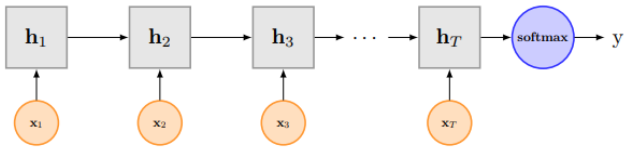

Figure 4: Recurrent Neural Network for Classification from [4]

4.2 Experiments

We undertake experiments on the same dataset with Sec. 3.2. In our algorithms, we resort to the LSTM cell and Linear layer supplied by FastNLP. The training loss curve is depicted in Fig. 5.

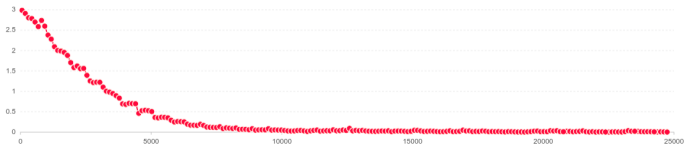

Figure 5: Cross Entropy Loss on trainning set in 35 epochs

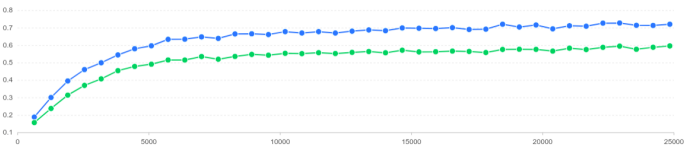

Figure 6: Accuracy of RNN model on the development set and test set during training process

The training process of RNN is extremely slow due to occurrences of long sentences. Furthermore, its performance is far poorer than the CNN model. The best accuracy achieved on development set is only 72.7% and is 58.9% on test set. The accuracy on training set is 99.9%.

5 Thoughts about FastNLP

为了使得表述更加清楚，我将使用中文来书写这一部分。深度学习方面，我的开发经验比较少，除了 PyTorch 和 FastNLP 之外，没有使用过别的框架，所以很多观点难免不成熟，希望多多包涵。

5.1 使用体验

便利 首先，在这次的作业中，FastNLP 给我带来的第一个便利就是内置的 Trainer 和 Tester。因为之前用 PyTorch 手写过 LSTM 的训练过程，代码也不是很少，且容易出错，而内置的 Trainer 在这一方面可以大大减少代码量，是比较便利的。但与之相应的一个问题是，Trainer 的灵活性变低了，假如我想在 Trainer 中加入自定义的操作，例如用 matplotlib 绘制 loss function 曲线，就不是很方便。虽然 0.4.1 版本中增加了 CallBack 的功能，但是书写会比较繁杂，相比起学习复杂的使用方法，我可能选择自己书写 Trainer。

然后，第二个便利是 fitlog 的使用。fitlog 是很良心的工具，实时更新 loss function，绘制 loss 曲线。同时，还可以查看过往的训练记录，简直就是调参小助手！一个小问题是，目前好像还没有导出功能，如果能导出.eps 或.pdf 类型的文件，那么对 LaTeX 选手会友好很多。

不便之处 之后是一些使用上不便的地方。一是文档不全，这一点让我在最初使用的时候十分头疼。像最开始版本的 trainer 和 tester 都是没有文档的，所以必须自己去看源码来理解，而源码又往往有比较复杂的调用关系，这样的递归关系大大地增加了我的工作量。然后像 Padder 这部分，默认使用的是 AutoPadder，但我发现在我的模型上效果并不好，所以果断选择自己写一个 Padder。然而，在文档中没有找到自定义 Padder 的方法，最后只好仿照着 AutoPadder 自定义了一个，中间又夹杂着许多小问题，花了比较长的时间。

二是在使用时，我时常会出现 torch 和 FastNLP 混用的情况。这主要是因为 FastNLP 还处于开发阶段，很多 PyTorch 已有的单元还没有完全搬过来，就我自己的使用而言，可能直接调用 PyTorch 会更方便些。希望以后开放更完善之后，FastNLP 能够做到 self-contained，这样就能避免书写一些“杂交”的代码。

5.2 建议

Padder 的设置 在使用 LSTM 的时候，我发现自动设置的 Padder 会在同一 batch 的所有句子后面填 0 补至同一长度。但在本问题中，若同一 batch 中的句子长度差异较大的话，会导致 0 数目过多，影响模型性能。我的解决方法是，自己仿照 AutoPadder 重写了一个 Padder，在句子前面补 0，才解决了这一问题。所以，一是希望能添加自定义 Padder 的功能，二是希望能修改 LSTM 的实现，其实对于长度不一致的句子，也是可以同时通过 LSTM 层的。

文档检索方式 往后 FastNLP 中可能会复现更多 NLP 领域已有的模型，与之相应的一个问题是如何系统地为这些模型的模块命名并检索。要知道，大部分人是没有耐心读完整个文档的，而如果对模型按字典序排列，那么其实很难找到自己想要的模型，或者确认模型在 FastNLP 中是否有实现。

我的想法是可以建立模型与论文之间的映射，按照论文检索相应的模型。并且可以提供按照论文的时间对模型排序的功能，这样也从一方面放映出 NLP 领域模型演变的进程。

模型分析 对于每个模型的性能，原理，优劣，具体应用，可在与文档配套的描述中简单地整理，这样无论是对学习还是开发都会有很大的帮助。典型的方式可以是，引用论文说明原理，配套上检验模型时用到的实验结果来说明模型的性能。

最后，希望 FastNLP 未来可以有更好的发展！造轮子不易，且行且珍惜！

References

[1] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[4] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.