Lab4 CNN based Sentiment Analysis and RNN based Language Models

SHINE-MING WU SCHOOL OF INTELLIGENT ENGINEERING

Spring 2022

1 Prerequisites

- You need to install keras and tensorflow packages:

```
pip install tensorflow
```

- or,

```
conda install tensorflow
```

2 Assignment

Make sure you have the following file(s): `NLP_lab4.zip`, including:

```
NLP_lab4
 |__nlp_lab4.pdf (this file)
 |__CNN
 |   |__data_helper.py
 |   |__lab4_cnn_skeleton.py
 |   |__data
 |       |__glove_50d.txt
 |       |__test.csv
 |       |__train.csv
 |__RNN
     |__data_helper.py
     |__lab4_rnn_skeleton.py
     |__data
         |__ptb.test.txt
         |__ptb.train.txt
         |__ptb.valid.txt
```

Q1 Using text CNN to finish sentiment classification task. Note that, everyone should implement a unified model architecture (**check section 4**) rather than designing a new one.

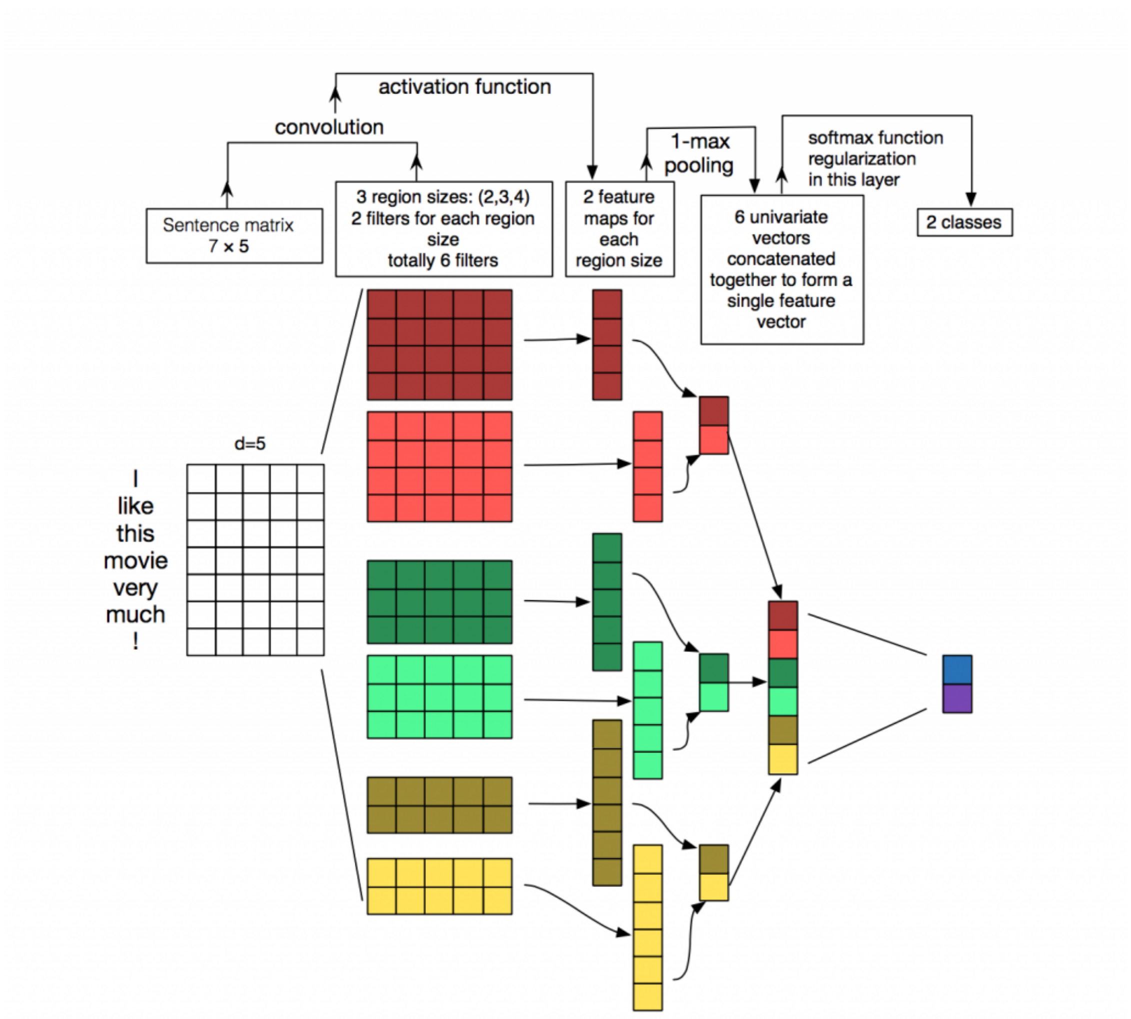
activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 1: Illustration of a CNN architecture for text classification. We depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1D max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states. Image Source: [1]

Q2 Using RNN to finish language model task, you can design a novel model architecture, or select an existing approach (**check section 5**).

Q3 (*Optional*) CUDA is NVIDIA's parallel computing platform and programming model for general-purpose computing on graphics processing units (GPUs). With CUDA, developers can harness the power of GPUs to dramatically accelerate computing applications.

You can install CUDA on your computer for faster training of deep learning tasks (Q1 and Q2), or, try using free GPU resources, such as Kaggle and Google Colab [2].

3 Submission

If you miss onsite assessment, you need to submit two files (program output, python script.) to SCUT-mail (202110190459@mail.scut.edu.cn). After you finished the assignments, make sure you include the header information in the beginning of your code

```
# author: Your_name
# student_id: Your_student_ID
```

4 CNN

Suppose the number of words in each document is `seq_len`, and each word is associated with an embedding where the dimension of the embedding `emb_dim`. If the length of each document is variate, we need to do padding to ensure the length of each document is the same.
CNN mainly contains two kinds of layers, namely, convolutional layer and pooling layer. Sepcially, in text classification, we will use `Conv1D` and `GlobalMaxPooling1D` in `Keras.layers`.

- `Conv1D(filters, kernel_size, strides, activation, ...)`

  `filters`: Integer, specifying the number of output filters in the convolution.
  `kernel_size`: Integer, specifying the length of the 1D convolution window.
  `strides`: Integer, specifying the stride length of the convolution.
  `activation`: String, specifying the activation function to use.
  `...`: other arguments. In this lab, we just set it default. For more details, please read `https://keras.io/layers/convolutional/`.
  Input shape

  3D tensor with shape: (`batch_size, seq_len, emb_dim`)
  Output shape

3D tensor with shape: (`batch_size, new_steps, filters`)
where `new_steps = seq_len - kernel_size + stride`
Example

The red filters in Figure 1 should be implemented as:

```
Conv1D(filters=2, kernel_size=4, strides=1,activation='relu')
```

The green filters should be implemented as:

```
Conv1D(filters=2, kernel_size=3, strides=1,activation='relu')
```

- `GlobalMaxPooling1D(...)`
  `...`: other arguments. In this lab, we just set it default. For more details, please read `https://keras.io/layers/pooling/`.
  Input shape

  3D tensor with shape: (`batch_size, new_steps, filters`)
  Output shape

  2D tensor with shape: (`batch_size, filters`)
  Example
  The max pooling on the outputs of red/green/yellow filters in Figure 1 should be implemented as:

  `GlobalMaxPooling1D()`
  Note that, Global max pooling operation for 1D temporal data. `https://keras.io/api/layers/pooling_layers/global_max_pooling1d/`

- `Dense(...)`
  Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True). These are all attributes of Dense. `https://keras.io/api/layers/core_layers/dense/`
  Input shape

  N-D tensor with shape: (batch_size, ..., input_dim). The most common situation would be a 2D input with shape (batch_size, input_dim).

  Output shape

  N-D tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, units).
  Example
  The final outputs in Figure 1 should be implemented as:

  ```
  Dense(units=2, activation='softmax')
  ```

5 RNN

Please find the slides **NLP_lec9** for futher understanding of RNN (see Figure 2), or these blogs

- Introduction: `https://zhuanlan.zhihu.com/p/32755043`
- Code: `https://zhuanlan.zhihu.com/p/32881783`

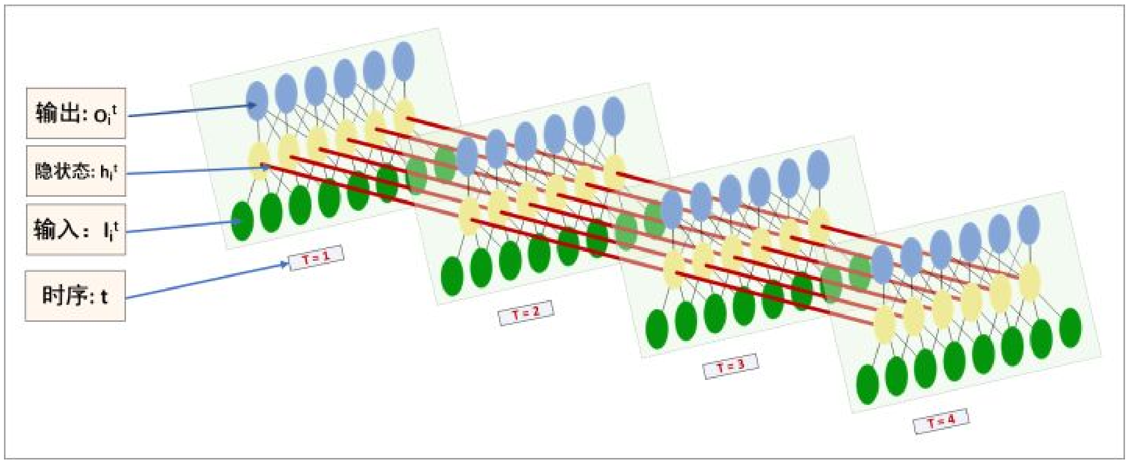

Figure 2: Illustration of a RNN architecture. Image Source: `https://www.jianshu.com/p/b9cd38804ac6/`

In this assignment, you can add a RNN by `SimpleRNN(...)`, please read `https://keras.io/api/layers/recurrent_layers/simple_rnn/`.

Example

inputs: A 3D tensor denotes some sentences, with shape [batch, timesteps, feature], e.g,[32, 10, 8], i.e., The number of sentences is 32, each sentence contains 10 words, each word is represented as a 8-dimensional vector.

```
SimpleRNN(units = 10, activation='relu',
use_bias=True, return_sequences=True)(inputs)
```

Advanced Reading Materials (CNN)

- Keras: `https://keras.io/`.
- CNN: Convolutional Neural Networks for Sentence Classification [3].
- Tuning hyper-parameters: A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification [1].

- Word embeddings tutorial: `http://ruder.io/word-embeddings-1/`
- GloVe embeddings [4]: `https://nlp.stanford.edu/projects/glove/`

Advanced Reading Materials (RNN)

- Keras: `https://keras.io/`.
- RNN: `http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/`
- LSTM: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.
- The state-of-the-arts: `https://arxiv.org/pdf/1707.05589.pdf`

References

[1] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[2] `https://medium.com/aiplusoau/how-to-use-kaggle-and-google-colab-notebooks-with-gpu-enabled-c2a1512cd4f`.

[3] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.