11-741/11-641/11-441 – Machine Learning for Text Mining **Due: Oct 8, 2019, 11:59pm**

Instructor: Yiming Yang

Homework 2: Deep Learning for Text Classification

In this assignment, you will be asked to implement text classifiers using Recurrent Neural Network(RNN) and Convolutional Neural Network(CNN), to train word embeddings on large corpus and try to use the pre-trained embeddings to improve the classification performance.

1 Deep Learning library

There are many deep learning libraries. To complete this assignment, we asked you to choose from Tensorflow, Keras, PyTorch, or any other auto differentiation framework that you prefer. A useful comparison of deep learning libraries can be found on Wikipedia. We recommend Keras for this assignment for students who are new to deep learning, because of its simplicity.

2 Preprocessing

The input for most neural network text classifiers is a sequence of word indexes. For example, if we have vocabulary:

`{"are":1, "cat":2, "dog":2, "is":3, "now":4, "on":5, "table":6, "the":7}`

and the input document:

`the dog is on the table`

The bog-of-word representation would be

$$[0, 0, 1, 1, 0, 1, 1, 1].$$

The sequence of word indexes would be

$$[7, 2, 3, 5, 7, 6].$$

Now you can see that by using sequence of word indexes, the word order matters! For a similar but semantically different document:

`the table is on the dog`

For the two documents, clearly the bag-of-word representations are the same while the indexes sequences are different.

A obvious challenge for sequence of indexes representation is that documents are of different length. A simple solution is to specify a threshold of sequence length l_{max}. If a document is longer than this length, we keep the first l_{max} words; if is is shorter than l_{max}, we pad the empty positions with a special index(usually 0).

3 Word Embedding

3.1 Embedding Lookup Layer

The first component for neural text classifier (and some other tasks like machine translation, question answering, etc) is the embedding lookup layer. In Keras this component can be implemented by a convenient layer object **Embedding**(https://keras.io/layers/embeddings/). For example, if the input is:

$$[7, 2, 3, 5, 7, 6, 0, 0, 0, 0]$$

which is of length 10. And we have a word embedding of dimension 100 for each word, the resulting matrix after the word embedding lookup layer is of dimension 10×100.

3.2 Pre-trained word embedding

As discussed in class, the word embedding look up layer can be learned from scratch, or initialized by some pre-trained word embedding. Please check the document of your favorite deep learning library to figure out how to use a pre-trained word embedding. In later experiments we are going to compare the effect of using pre-trained word embedding versus learning it from scratch.

4 Recurrent Neural Network(RNN)

We asked you to implement a LSTM model for text classification. The final softmax layer should have the final state of your LSTM as input. In Keras, all you need to build this model is simply as follows:

```
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

The `Sequential` model is a linear stack of layers. We construct the whole pipeline by passing a list of layer instances to the constructor. Check out the Keras documentation to find the meanings of arguments for `Embedding`, `LSTM`, `Dense` layers.

Before training a model, you need to configure the learning process, which is done via the `compile` method. It receives three arguments: an optimizer, a loss function and a list of metrics.

5 Convolutional Neural Network(CNN)

In this part you are required to implement a simple version of a CNN text classifier[1]. The original paper uses filters of different lengths(3, 4 and 5). For simplicity we only require you to implement a single filter length. However, a better performance might be achieved by merging outputs from filters of different sizes. If you are using Keras, you can look at its examples (link). If you are using Tensorflow, there is a nice tutorial (link).

6 Evaluation

We ask you to evaluate your implementation on a small sentiment classification dataset. Classifiers should be trained on 2000 training documents and tested on 2000 testing documents, with sentiment labels(positive/negative). A pre-trained word embedding on a large set of Amazon review is also provided.

Due to the possible shortage of GPU resource, we will not ask for the optimal performance in this homework. All you need to do is training the correct model on training set and report the performance on the test set. We will provide suggested hyper-parameters for each model and you don't need to tune it for a better performance. Please refer to the report template for more detail.

By default, the code will be tested on Ubuntu Linux 18.04 and Python 3.6.7, please make sure your code can run successfully under such environment. If you plan to use a different version please document the choice in the README.txt. If you plan to use other programming languages, please discuss with TAs in advance. Make sure to include the package dependency files such as requirements.txt or environment.yml for documenting the packages used.

7 Submission Checklist

The report and the source code should be submitted separately.

For the source code, please compress all the following files into a .zip file for submission.

1. All source code in **src** folder
2. README.txt on how to execute your code in **command line**

Example folder structure:

```
HW2.zip
 ├── src
 │    ├── CNN.py
 │    └── LSTM.py
 └── README.txt
```

References

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

Last updated on August 28, 2019