

---

# Sentence-level Sentiment Classification with PyTorch

---

## Homework 5 for Deep Learning, Spring 2020

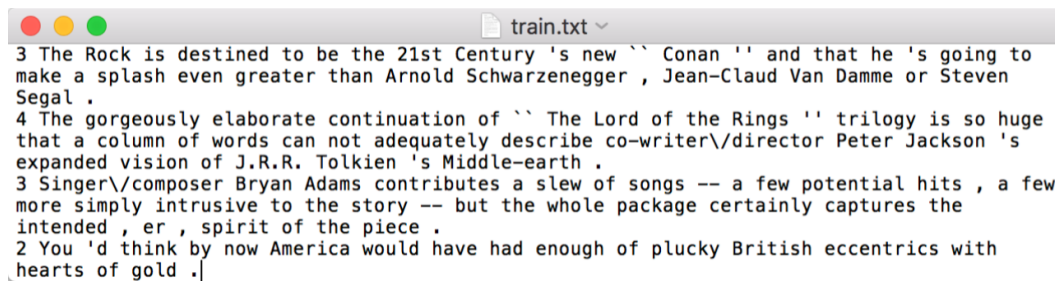
Deadline: 2020.04.21 11:59:00 AM

### 1 Introduction

**Stanford Sentiment Treebank** (SST) is a dataset for sentiment classification. It contains 11855 sentences, and has been split into the training / validation / test parts, respectively containing 8,544 / 1,101 / 2,210 sentences. The test accuracy in this case will not be very high (about 50% is acceptable).

#### 1.1 Data Format

Every line in SST: Label(Sentiment) + Data(Sentence) There are five kinds of annotations in label: 0-“very negative”; 1-“negative”; 2-“neutral” 3-“positive”; 4-“very positive”. Digits in MNIST range from 0 to 9. Some examples are shown below.



```
3 The Rock is destined to be the 21st Century 's new `` Conan '' and that he 's going to
make a splash even greater than Arnold Schwarzenegger , Jean-Claud Van Damme or Steven
Segal .
4 The gorgeously elaborate continuation of `` The Lord of the Rings '' trilogy is so huge
that a column of words can not adequately describe co-writer\director Peter Jackson 's
expanded vision of J.R.R. Tolkien 's Middle-earth .
3 Singer\composer Bryan Adams contributes a slew of songs -- a few potential hits , a few
more simply intrusive to the story -- but the whole package certainly captures the
intended , er , spirit of the piece .
2 You 'd think by now America would have had enough of plucky British eccentrics with
hearts of gold .|
```

Figure 1: Examples of data in SST

#### 1.2 Data Preprocessing

Torchtext is recommended for loading and preprocessing SST data. To install torchtext, you can use

- **pip install torchtext**

We provide some start codes for SST DataLoader, which are included in **tips\_code.py**.

To learn more about Torchtext, you can read some documents about TorchText: [Torchtext Doc](#), [SST Dataset Source Code](#), and [Another Code for Preprocessing SST](#)

#### 1.3 Introduction for Word Embedding

Word Embedding is used in our DataLoader code. The embedding layer is used to transform the word into a dense embedding vector. This embedding layer is simply a single fully connected layer. You can see [torch.nn.Embedding](#) for more details. The input is firstly passed through the embedding layer to get embedded, which gives us a dense vector representation of our sentences. Embedding is then fed into the RNN. For simplicity, we use pre-trained word embeddings. Codes for pre-trained embeddings are provided. You can also use other pre-trained embeddings in this task. Figure 2 shows the basic pipeline for sentiment classification.

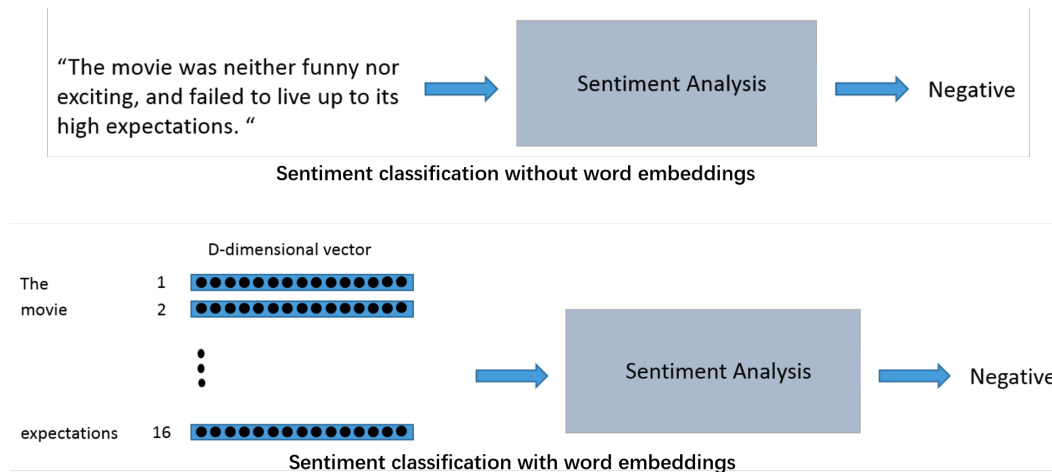


Figure 2: Basic process for sentiment classification

## 1.4 Introduction of RNN in PyTorch

RNN is a basic network for sequence processing. PyTorch provides many kinds of RNN such as RNN, LSTM and GRU. (<https://pytorch.org/docs/stable/nn.html#recurrent-layers>)

## 1.5 Example Architecture for Sentiment Classification

Here are two examples of network architecture: Figure 3 and Figure 4.

## 2 Requirements

You are required to perform Sentence-level Sentiment Classification with PyTorch. All parts of implementation depend on you (e.g. types of rnn, number of layers/units, loss, optimizer...). You are encouraged to use techniques such as bidirectional, dropout and attention, to improve the accuracy. **You need to submit all codes and a report** with the following requirements:

- Illustrate your network architecture with words and figures in your report.
- Show your best results in your report.
- Show your hyper-parameters. Plot the loss curve and accuracy curve in the report.

## 3 Attention

- You need to submit all codes and a report (at least one pages **in PDF format**). Delete the dataset before submit.
- **Plagiarism (from the internet) is not permitted.**

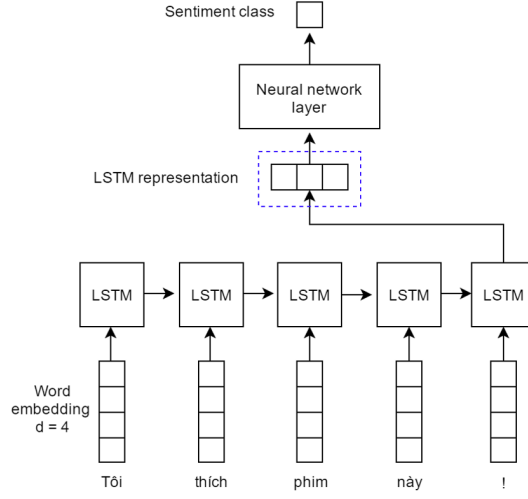


Figure 3: Model architecture example 1

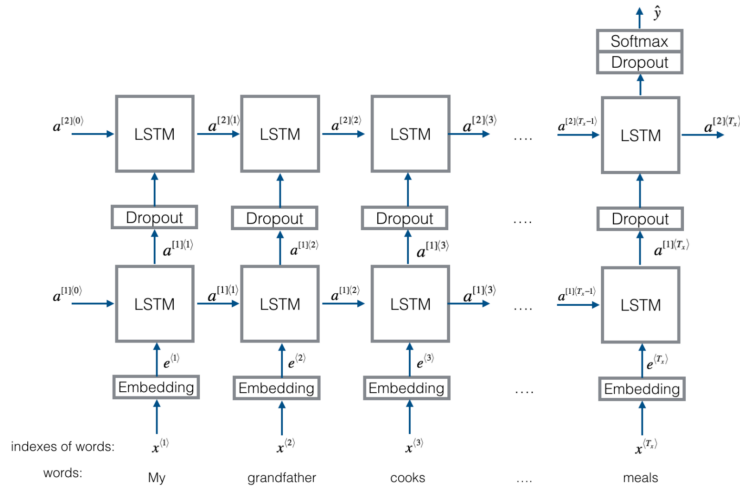


Figure 4: Model architecture example 2