Assignment 2: Exploring Word Representations

1 Introduction

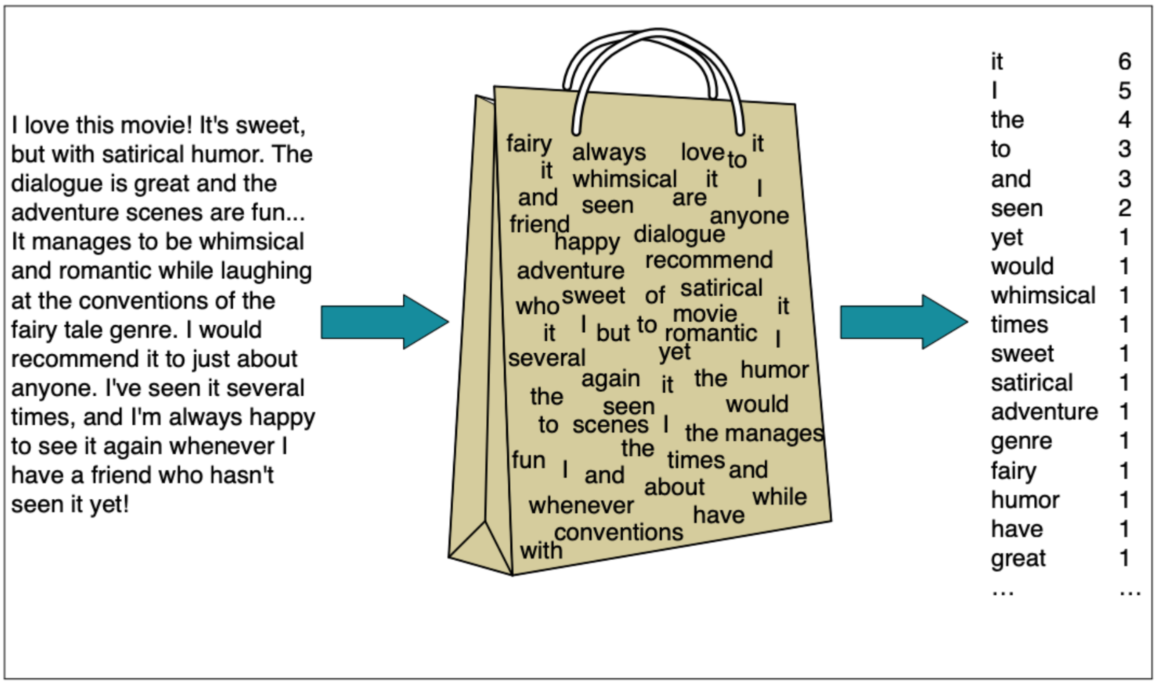
In this assignment, you will write a word vector representation project in python (using NumPy and PyTorch). Word vector representations are often used as a fundamental component for NLP tasks, e.g., text classification, question answering, text generation, translation, etc., so it is important to develop some intuitions as to their strengths and weaknesses. Here, you will explore two types of word vector representations: BOW (Bag-of-words) representation (high dimension vector semantics) and pre-trained word embeddings derived via *GloVe* (low dimension vector semantics). You will test them in a text classification task.

The only file you need to (or should) modify is *model.py*. You will implement or modify the code for some functions (see functions in Section 4). In this text classification task, the dataset contains 10000 reviews from the internet. It has two files, one is a text file (*review.txt*) and the other is the label file (*label.txt*). You need to conduct experiments on this dataset, and to answer some questions based on your results.

2 Model

2.1 Bag-Of-Words (BOW) Model

The intuition of the classifier is shown in the following figure. We represent a text as if it were a bag-of-words, that is, an unordered set of words (ignoring their positions), and noting only their frequency in the document. In the example in the figure, instead of representing the word order in phrases like “I love this movie” and “I would recommend it”, we simply note that the word 'I' occurred 5 times in the entire excerpt, the word 'it' 6 times, the words 'love', 'recommend', and 'movie' once, and so on.

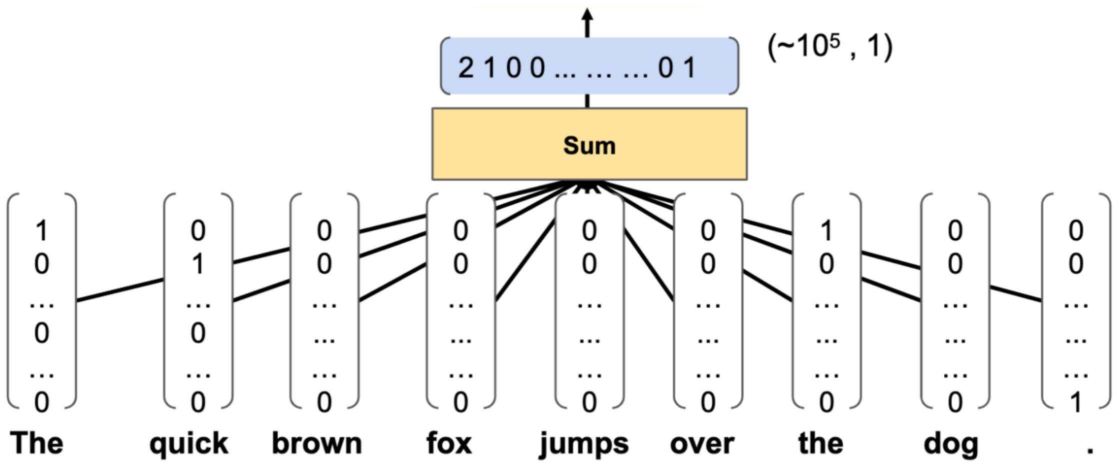


I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

A traditional approach to create BOW vectors is to build bag-of-words features:

1. build a vocabulary of frequent words (use training data only)
2. for each training sample, count the number of times a word occurs
3. consider this count a feature for the classifier, as illustrated below

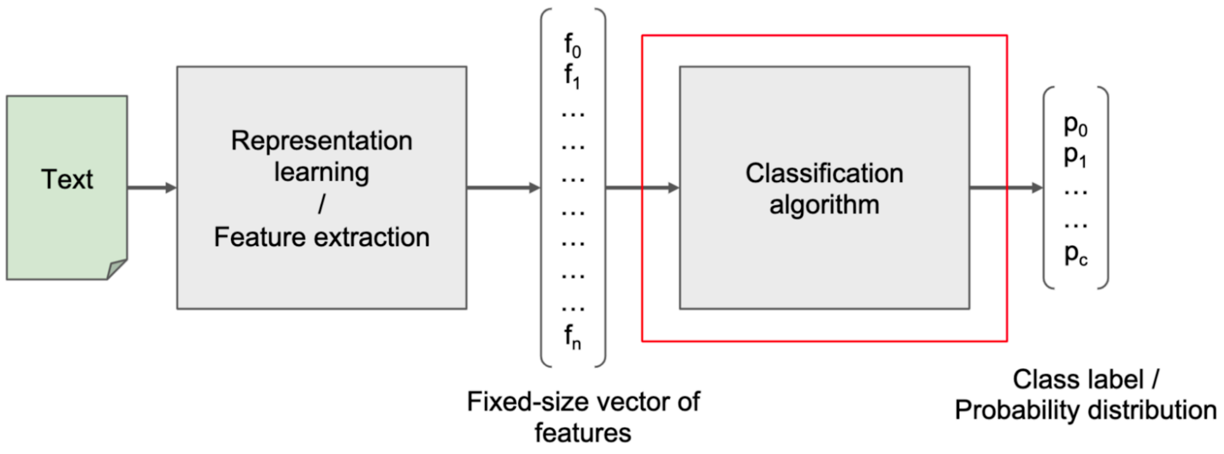


2.2 Latent Semantic Word Vectors

Document representation using word vectors, instead of actual words, has demonstrated better performance, such as word2vec and GloVe. Here, we shall explore the embeddings produced by GloVe. Read the Pennington et al. (2014) paper for details on the GloVe algorithm.

2.3 Text Classification

The figure below shows the whole story of this text classification task, where:


Text
Representation learning / Feature extraction
f0
f1
fn
Fixed-size vector of features
Classification algorithm
p0
p1
pc
Class label / Probability distribution

- **Input**: Some text ***x*** (e.g., sentence, document)
- **Output**: The Probability of each label ***y*** (from a finite label set)
- **Goal**: learn a mapping function ***f*** from ***x*** to ***y***:

In this assignment, you will implement such a text classification model to test the two kinds of word vectors. You need to implement a simple binary classifier. Since the amount of data is small, it does not need a very complicated algorithm. Here we use a three layer neural network as an example.

3 Requirements for The Functions

You need to modify or complete some functions in ***model.py:***

`def __init__(self, vocab, pos_data, neg_data)`: This is the place where you should define the neural network classifier model. For instance, you should implement a three layer neural network for binary classification.

`def sentence2vec(self, sentence, dictionary)`: This function can convert a sentence with an array of tokens into a sentence representation vector. You should modify the code to define two methods to convert sentence text to vector representations. One is for BOW and another is for GloVe. The return value is the average of the sum of all word vectors in this sentence.

`def load_glove(self)`: This function loads the pre-trained GloVe vectors from an existing file.

`def training(self)`: This function specifies the whole training process of the experiment which includes training and testing. You can modify the code to define a cross entropy loss function and an Adam optimizer via pytorch functions, but it is not necessary.

You are encouraged to define other helper functions to modularize your code. And note that you should not modify the ***run.py*** file ！

4 Doing the Project

Perform the following tasks to complete the project.

4.1 Download Tarball

The tarball for the assignment is available from Canvas at *Files/Homework templates, instructions, data/hw2*. This tarball includes the python project files and the dataset.

4.2 Project Tasks

You will then perform the following steps.

Task 0: Use Anaconda to create a ***python 3.5.6*** environment, and use ***pip*** to install all the dependent packages (***torch, nltk and numpy***). Make sure your code can run in this virtual environment.

Task 1: 1) Write the functions to get ***GloVe*** and **B*OW*** word representations. 2) Load the GloVe pre-trained word embeddings available from the Stanford GloVe website. You can choose any kind of GloVe embeddings in using the Wikipedia/Gigaword corpus (glove.6b.zip) (e.g. ***glove.6B.300d.txt***). Note that, if you change the dimension of the GloVe embedding file, you also need to modify the dimension parameters (***embed_size***) in your model definition part.

Task 2: Develop classifier functions and complete at least one training process. Aim to get the best performance for each model. You may need to change the parameters of the classifier, like the ***learning rate, hidden_size.*** Note that, you might need different learning rates for the same classifier, depending on the input vectors (BOW and GloVe).

Task 3: According to your experimental results, answer the questions in **section 6**.

5 Testing

We will test your submission on python 3.5.6. We highly recommend you test your code using the Anaconda 3 virtual environment (e.g. conda create -n assign_env python=3.5.6). After that you can test your code by running python scripts. For example, if you want to run the BOW experiment, you can use this command:

```
python run.py --algo BOW --lr 0.001 --hidden_size 5 \
        --review_file ./data/reviews.txt --label_file ./data/labels.txt
```

For the GloVe experiment, you need to create a directory called glove in the same location. Copy a glove.6B.<N>d.txt file to that location, which you can download from the GloVe website, and then test your code:

```
Python run.py --algo GLOVE --lr 0.001 --embed_size 50 --hidden_size 5 \
      --emb_file ./glove/glove.6B.50d.txt --review_file ./data/reviews.txt \ --
      label_file ./data/labels.txt
```

For every training round, it may take from 5 to 30 mins to train and test each model (depending on your machine). You do not need to wait for every training process to end automatically: if its loss and accuracy have **converged** to some point then it can be killed. For each model, you need to make sure the loss of your model can converge and the accuracy is no less than ***0.80***.

Points for BOW: 30 if Acc > 88%

Points for GloVe: 40 if Acc > 82%

6 Questions

1. **(10 Points)** What is the greatest advantage of BOW in this setting, if any?

2. **(10 Points)** What is the greatest advantage of GloVe in this setting, if any?

3. **(10 Points)** Explain any performance differences between BOW and GloVe in this setting. Note: Restrict your answers to about 2 sentences each.

7 Deliverables

Submit a single zip file called <PSUID>.zip where PSUID is your psu email id (drop the “@psu.edu” portion.)

1) Your completed model.py file and any other supporting .py files, with no environment dependencies, and no subfolders. We will run your code locally to verify the accuracy of the results you report. If we cannot run your code, your grade might be zero.

2) A csv file called ***<PSUID>.csv*** file with at least 2 lines for your runs and results. First put your command line for a run, then after the field-separator ',', put the accuracy your model achieved. We will run your code and expect the same accuracy, within some epsilon error.

Report at least your best performing BOW, and your best performing GloVe. You can have as many as 6 runs total in this file, if you want to refer to your non-optimal results in the answers you provide to the questions below.

3) A pdf file named ***<PSUID>.pdf*** providing answers to the questions above.