Spring 2022 CSE 354 - Natural Language Processing Assignment 1

This file is best viewed in a Markdown reader (eg. https://jbt.github.io/markdown-editor/)

Overview

In this assignment, you will be asked to:

- generate batch for skip-gram model (data.py)
- implement two loss functions to train word embeddings (model.py)
- tune the parameters for word embeddings
- calculate bias score on your best models (eval_bias.py)

Setup

This assignment is implemented in Python 3.6 and PyTorch 1.9.0. Follow these steps to setup your environment:

1. Download and install Conda
2. Create a Conda environment with Python 3.6: `conda create -n nlp-hw1 Python=3.6`
3. Activate the Conda environment. You will need to activate the Conda environment in each terminal in which you want to use this code: `conda activate nlp-hw1`
4. Install the requirements: `pip install -r requirements.txt`. You do not need anything else, so please do not install other packages.

NOTE: We will be using this environment to check your code, so please don't work in your default or any other Python environment.

Boilerplate Code Organisation

When you download the assignment from Blackboard, it will unzip into the following directory structure:

```
538_HW1
├── .gitignore
└── data/
  ├── text8
  ├── weat.json
├── data.py
├── main.py
├── model.py
├── README.md
├── requirements.txt
├── train.py
```

Please only make your code edits in the TODO(students) blocks in the codebase. The exact files you need to work are listed later.

Generating data

To train word vectors, you need to generate training instances from the given data. You will implement a method that will generate training instances in batches.

For skip-gram model, you will slide a window and sample training instances from the data inside the window.

[Example] Suppose that we have a text: "The quick brown fox jumps over the lazy dog."

And batch_size = 8, window_size = 3

"[The quick brown] fox jumps over the lazy dog"

Context word would be 'quick' and predicting words are 'The' and 'brown'.

This will generate training examples:

context(x), predicted_word(y)

(quick , The)

(quick , brown)

And then move the sliding window.

"The [quick brown fox] jumps over the lazy dog"

In the same way, we have two more examples

(brown, quick)

(brown, fox)

Moving the window again:

"The quick [brown fox jumps] over the lazy dog"

We get,

(fox, brown)

(fox, jumps)

Finally we get two more instances from the moved window,

"The quick brown [fox jumps over] the lazy dog"

(jumps, fox)

(jumps, over)

Since now we have 8 training instances, which is the batch size, stop generating this batch and return batch data.

data_index is the index of a word.

You can access a word using data[data_index].

batch_size is the number of instances in one batch.

num_skips is the number of samples you want to draw in a window (in example, it was 2).

skip_windows decides how many words to consider left and right from a context word(so, skip_windows*2+1 = window_size).

batch will contains word ids for context words. Dimension is [batch_size].

labels will contains word ids for predicting words. Dimension is [batch_size, 1].

You need to fill in `data.py`.

Loss Functions

You will implement two loss functions to train word vectors:

(1) negative log likelihood - recall that we discussed the log likelihood function in class, and

(2) negative sampling - an alternate function that uses a set of k negative words.

You need to fill in `model.py`.

Negative log likelihood

We discussed the log likelihood function in class. This is the negative of the same. These are called “loss” functions since they measure how bad the current model is from the expected behavior.

Refer to the class notes on this topic.

The equation of negative log likelihood which you would be implementing is given here

To delve deep and understand it better, you may also refer to Section 4.3 here.

Expeccted running time: ~30 mins on Google Colab with GPU accelarator.

Negative Sampling

The negative sampling formulates a slightly different classification task and a corresponding loss.

The idea here is to build a classifier that can give high probabilities to words that are the correct target words and low probabilities to words that are incorrect target words. As with negative log likelihood loss, here we define the classifier using a function that uses the word vectors of the context and target as free parameters. The key difference however is that instead of using the entire vocabulary, here we sample a set of k negative words for each instance, and create an augmented instance which is a collection of the true target word and k negative words. Now the vectors are trained to maximize the probability of this augmented instance.

The equation of negative sampling which you would be implementing is given here

To understand it better, you may also refer to Section 4.4 here. Additionally, This paper describes the method in detail.

Expected running time: ~1h10 mins on Google Colab with GPU accelarator.

Relevant code files:

- main.py: **You aren't required to change anything here**

  This file is the main script for training word2vec model. Your trained model will be saved as word2vec.model.

- data.py: **Fill in the TO-DO block in this file**

  This file contains various operations related to reading data and other files. It encapsulates functions relevant to model training into a Dataset class. You should fill out the part that generates batch from training data.

- train.py: **You aren't required to change anything here**

  This file contains a Trainer class to train a model. It is a similar formulation as seen in the logistic regression demo in class.

- model.py:**Fill in the TO-DO block in this file**

  This file contains code for the word2vec model as well as the two loss function implementations, that need to be filled in.

  - negative_log_likelihood_loss
  - negative_sampling

After completing the #TODO(students) portions in data.py and model.py, run using main,py

In order to learn how to use main.py and its arguments, you can use the command **"python main.py --help"**

Hyperparameter Experiments

You will conduct some hyper-parameter tuning experiments to figure out the best way to learn the word vectors. Your goal is to understand how the different hyperparameters affect the embeddings.

The hyperparameters experimented in this code are:

1. Number of iterations (Max_num_steps)
2. The number of pairs generated for each center word (num_skips)
3. Size of skip window (skip_window)
4. The batch size taken during training (batch_size)
5. The size of the word embeddings (Embedding_size)
6. The number of negative samples (k)

Run these below mentioned configurations for each model.

For the **Negative Log Likelihood model**, try these configurations:

Max_num_steps	num_skips	skip_window	batch_size	Embedding Size
100000	2	1	64	128
100000	8	4	64	128
50000	2	1	64	128
100000	2	1	128	128
100000	2	1	64	64

For the **negative sampling model**, try these configurations:

Max_num_steps	num_skips	batch_size	Embedding Size	K
100000	2	64	128	1
100000	4	64	128	1
50000	2	128	128	1
100000	2	128	64	1
100000	2	64	128	3

For each of these configurations , report the average loss obtained. Report the best negative log likelihood and negative sampling model configurations that gave the minimal loss.

Bias Evaluation using WEAT Test

Word embeddings, when trained on datasets, also capture the social biases that exist on human text.

For example, Word2Vec trained on US newspapers might associate more negative words to African Americans due to the data it is trained upon.

In order to measure this bias, we perform WEAT test. WEAT test takes two pairs of sets of words and tries to calculate how much the words in each pair are similar to each other, thereby the bias.

A higher WEAT score means that the bias is much more stronger while a negative WEAT score indicates anti-bias or disimillarity.

In this task, we will be looking at how to evaluate whether the embeddings are biased or not.

The WEAT test provides a way to measure quantifiably the bias in the word embeddings. This paper describes the method in detail.

The basic idea is to examine the associations in word embeddings between concepts. It measures the degree to which a model associates sets of target words (e.g., African American names, European American names, flowers, insects) with sets of attribute words (e.g., ”stable”, ”pleasant” or ”unpleasant”). The association between two given words is defined as the cosine similarity between the embedding vectors for the words.

In order to run bias estimation on generated models:

```
python eval_bias.py
    --model_path [full_model_path]
    --weat_file_path [custom_weat_file_path]
```

This will generate the bias scores as evaluated on 5 different tasks with different sets of attributes (A and B) and targets (X and Y) as defined in the file pointed to in the `weat_file_path` (`weat.json` for the given data). This will print and dump the output in the filepath you specify.

To see the scores on tasks as defined in `weat.json`, run the following command:

```
python eval_bias.py --model_path [full_model_path] --weat_file_path
[weat_file_path]
    --out_file [filepath_for_bias_output]
```

Example usage: `python eval_bias.py --model_path best_nll_model/word2vec_nll.model --weat_file_path data/weat.json --out_file nll_bias_output.json`

Your submission bias output files should be named `nll_bias_output.json` and `neg_bias_output.json`.

Blackboard Submission Code Organisation

PAY ATTENTION TO THE FILENAMES AND STRUCTURE!!!

Submit the following files:

- `data.py` - With the completed TO-DO block for estimating context word,focus word pairs
- `model.py` - With the completed TO-DO block for computing negative log loss and negative sampling.
- `nll_bias_output.json` - Results for the WEAT task on `weat.json` using your best NLL model
- `neg_bias_output.json` - Results for the WEAT task on `weat.json` using your best negative sampling model
- `gdrive_link.txt` - Should contain a `wget`able to a folder that contains your best models. The model files should be named `word2vec_nll.model` and `word2vec_neg.model`, and the folder should be named `354-hw1-<SBUID>-models`. Please make sure you provide the necessary permissions.
- `<SBUID>_Report.pdf` - A PDF report as detailed below.

Your PDF report should contain your answer to the following questions.

1. Explanation of your code implementations for each TO-DO tasks.
2. Brief description of the Hyper-parameters explored for each of the loss functions.
3. Discuss the hyperparameters experimented and how the average loss varied across each combination.
4. Summary of the comparison in bias scores (for the weat.json output) obtained for the best model as obtained with loss function of NLL and negative sampling. Justification for the bias score along with the attributes and targets used must be mentioned.

Due Date and Collaboration

- The assignment is due on Mar 5, 2022 at 11:59 pm EDT. You have a total of three extra days for late submission across all the assignments. You can use all three days for a single assignment, one day for each assignment – whatever works best for you. Submissions after the third day will be docked 20% per every additional day.
- You can collaborate to discuss ideas and to help each other for better understanding of concepts and math.
- You should NOT collaborate on the code level. This includes all implementation activities: design, coding, and debugging.
- You should NOT not use any code that you did not write to complete the assignment.
- The homework will be **cross-checked**. Do not cheat at all! It's worth doing the homework partially instead of cheating and copying your code and get 0 for the whole homework. In previous years, students have faced harsh disciplinary action as a result of the same.

Google Colab

You should code your assignment on your local machine. However, due to the size of the dataset, you will need GPUs in order to accelerate the training. You can use your own GPU if you have any or you can use Google Colab.

Google Colab is a free resource available to everyone. You should use a GPU.. It allows you to reserve a CPU, GPU or TPU, depending on your use case. You can run commands in a cell if you add `!` at the start of the line of the command in the cell. To use it to train your models, all you need to do is upload all the relevant files and make sure you mimic the directory structure of the assignment.

The abovementioned setup is ephemeral.

You will lose all the files when the **runtime shuts down (due to inactivity or otherwise)** . So, save your models when they are trained.

You can link it to your Google Drive (using your stonybrook.edu or cs.stonybrook.edu) account, and the trained models will be saved in your drive.

We recommend using Google Colab for model training, and doing the weat test for evaluating bias on your local machine.

Some Debugging Tips

You can use a debugger anywhere in your code. You can also put plain python print statements if you don't like debuggers for some reason.

Here are some examples of using pdb debugger:

1. Put `import pdb; pdb.set_trace()` anywhere in your python code. When the control flow reaches there, the execution will stop and you will be provided python prompt in the terminal. You can now print and interact with variables to see what is causing problem or whether your idea to fix it will work or not.

2. If your code gives error on nth run of some routine, you can wrap it with try and except to catch it with debugger when it occurrs.

```
try:
        erroneous code
except:
        import pdb; pdb.set_trace()
```

You can, of course, use your favorite IDE to debug, that might be better. But this bare-bone debugging would itself make your debugging much more efficient than simple print statements.

Extra Notes

- If you add any code apart from the TODOs in the codebase (note that you don't need to), please mark it by commenting in the code itself. An example of the same could be:

  ```
  # Adding some_global_var for XXX
  some_global_var
  ```

- General tips when you work on tensor computations:
  - Break the whole list of operations into smaller ones.
  - Write down the shapes of the tensors

Credits and Disclaimer

Credits: This code is part of the starter package of the assignment/s used in NLP course at Stony Brook University. This assignment has been designed, implemented and revamped as required by many NLP TAs to varying degrees. In chronological order of TAship they include Heeyoung Kwon, Jun Kang, Mohaddeseh Bastan, Harsh Trivedi, Matthew Matero, Nikita Soni, Sharvil Katariya, Yash Kumar Lal, Adithya V. Ganesan, Sounak Mondal, Dhruv Verma, Aditi Kandoi, Saqib Md.Hasan and Shyam Jayashankar. Thanks to all of them!

Disclaimer/License: This code is only for school assignment purpose, and **any version of this should NOT be shared publicly on github or otherwise even after semester ends**. Public availability of answers devalues usability of the assignment and work of several TAs who have contributed to this. We hope you'll respect this restriction.