Spring 2022 CSE 354 - Natural Language Processing Assignment 2

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

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 and Sounak Mondal. 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.

Overview

This assignment is to learn (i) how to train sentence representations for a classification task and (ii) how to probe these representations for analysis. You will also explore (i) how the data size and training time affects your model (ii) how to perform error-analysis on your trained models.

You will implement two models for sentence representations for a task: (i) Deep Averaging Network and (ii) GRU based Model. You will also implement a linear probing model to understand what kind of information the learnt representations capture.

Installation

This assignment is implemented in python 3.6 and torch 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-hw2 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-hw2
```

4. Install the requirements:

```
pip3 install -r requirements.txt
```

5. Download GloVe wordvectors:

```
./download_glove.sh
```

6. Download Spacy package

```
python3 -m spacy download en_core_web_sm
```

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

We have used type annotations in the code to help you know the method signatures. Although, we wouldn't require you to type annotate your python code, knowing types might help know what is expected. You can know more about type annotations here.

Data

There are two classification datasets in this assignment stored in `data/` directory:

1. **IMDB Sentiment:**: It's a sample of the original dataset which has annotations on whether the imdb review is positive or negative. In our preprocessed version, positive is labeled 1 and negative is labeled 0.

- Following are the development and test sets: `imdb_sentiment_dev.jsonl` and `imdb_sentiment_test.jsonl`
- There are different sized samples of the training set : `imdb_sentiment_train_5k.jsonl`, `imdb_sentiment_train_10k.jsonl` etc.

2. **Bigram Order:**: It's a binary classification of wheter bigram is in correct order or reversed. For example, `New York` would be 1 and `York New` would be 0. It's in the same format as the imdb sentiment dataset.

- The train, dev, test files are `bigram_order_train.jsonl`, `bigram_order_dev.jsonl` and `bigram_order_test.jsonl`.

Code Overview

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

Data Reading File

Code dealing with reading the dataset, generating and managing vocabulary, indexing the dataset to tensorizing it and loading embedding files is present in `data.py`.

Modeling Files

The main code to build models is contained the following files:

- `main_model.py`
- `probing_model.py`
- `sequence_to_vector.py`

There are two kinds of models in this code: main and probing.

- The main model (`main_model.py`) is a simple classifier which can be instantiated using either DAN or GRU Sentence encoders defined (to be defined by you) in `sequence_to_vector.py`
- The probing model (`probing_model`) is built on top of a pretrained main model. It takes frozen representations from nth layer of a pretrained main model and then fits a linear model using those representations.

Expectations

Model Implementation - what to write in code:

There are only 3 placeholders in the code.

1. Deep Averaging Network: (of n layers and p dropout probability)

- in `sequence_to_vector.py`
- Please look at the details of Deep Averaging Network from this paper
  - Deep Unordered Composition Rivals Syntactic Methods for Text Classification by Iyyer et al.
  - Check out Figure 1 and equations 5-8 in particular.
  - Make sure you use ReLU activation for the first n - 1 hidden layers, but do not add any activation to the last hidden layer.
- Implement the word dropout correctly (and use it wisely).

2. GRU Encoder: (of n layers):

- in `sequence_to_vector.py`
- A GRU is a recurrent neural network that gates sequential information over time-steps.
- You have to use the built-in `torch.nn.GRU` rather than building from scratch. For an n-layered GRU based model, you will stack n layers of GRU layers unlike n layers of feedforward network in DAN. Note that you will be using a unidirectional GRU model.
- The sentence representation from each layer therefore will be the hidden state of the last time step of that layer. Consider the same from the last layer as the `combined_vector` in `sequence_to_vector.py`.
- To enable batch processing, padding is required to accomodate variable length texts in a minibatch. However, you do not want the GRU to process the trailing padding tokens - so use `torch.nn.utils.rnn.pack_padded_sequence`.
- This is trivial to mention, but you do not have to use any extra activation functions - the activation functions for the gates are provided in the `nn.GRU` module.

3. Probing Model

- in `probing_model.py`
- You will implement a linear probing model that takes vector representations from nth layer of the underlying (frozen) model and learns a linear classifier on these representations. For DAN, the probing

vector of nth layer is simply the average that you computed for that layer. For GRU, the probing vector is the hidden state of the last time step of that layer.

The code placeholders that you have to fill up are marked with :

```
# TODO(students): start
  ...
# TODO(students): end
```

Just fill the above place-holders, run the analysis scripts and write you observations/explanations in the report. DO NOT add or modify any code outside the code placeholders.

We would advise you the fill the above code in the same order and use the training commands as shown in previous section to check if it's running correctly. You should be able to see the loss decreasing gradually.

Operate on Model:

The following scripts help you operate on these models: `train.py`, `predict.py`, `evaluate.py`. To understand how to use them, simply run `python train.py -h`, `python predict.py -h` etc and it will show you how to use these scripts. We will give you a high-level overview below though:

Caveat: The command examples below demonstrate how to use them *after filling-up* the expected code in the model code files. They won't work before that.

Train:

The script `train.py` lets you train the `main` or `probing` models. To set it up rightly, the first argument of `train.py` must be model name: `main` or `probing`. The next two arguments need to be path to the training set and the development set. Next, based on what you model choose to train, you will be asked to pass extra configurations required by model. Try `python train.py main -h` to know about `main`'s command-line arguments.

The following command trains the `main` model using `dan` encoder:

```
python3 train.py main \
                  data/imdb_sentiment_train_5k.jsonl \
                  data/imdb_sentiment_dev.jsonl \
                  --seq2vec-choice dan \
                  --embedding-dim 50 \
                  --num-layers 4 \
                  --num-epochs 5 \
                  --suffix-name _dan_5k_with_emb \
                  --pretrained-embedding-file data/glove.6B.50d.txt
```

The output of this training is stored in its serialization directory, which includes all the model related files (weights, configs, vocab used, tensorboard logs). This serialization directory should be unique to each training to prevent clashes and its name can be adjusted using `suffix-name` argument. The training script automatically generates serialization directory at the path `"serialization_dirs/{model_name}_{suffix_name}"`. So in this case, the serialization directory is `serialization_dirs/main_dan_5k_with_emb`.

Similarly, to train `main` model with `gru` encoder, simply replace the occurrences of `dan` with `gru` in the above training command.

Predict:

Once the model is trained, you can use its serialization directory and any dataset to make predictions on it. For example, the following command:

```
python3 predict.py serialization_dirs/main_dan_5k_with_emb \
                  data/imdb_sentiment_test.jsonl \
                  --predictions-file my_predictions.txt
```

makes prediction on `data/imdb_sentiment_test.jsonl` using trained model at `serialization_dirs/main_dan_5k_with_emb` and stores the predicted labels in `my_predictions.txt`.

In case of the predict command, you do not need to specify what model type it is. This information is stored in the serialization directory.

Evaluate:

Once the predictions are generated you can evaluate the accuracy by passing the original dataset path and the predictions. For example:

```
python3 evaluate.py data/imdb_sentiment_test.jsonl my_predictions.txt
```

Probing

Once the `main` model is trained, we can use its frozen representations at certain layer and learn a linear classifier on it by *probing* it. This essentially checks if the representation in the given layer has enough information (extractable by linear model) for the specific task.

To train a probing model, we would again use `train.py`. For example, to train a probing model at layer 3, with base `main` model stored at `serialization_dirs/main_dan_5k_with_emb` on sentiment analysis task itself, you can use the following command:

```
python3 train.py probing \
                  data/imdb_sentiment_train_5k.jsonl \
                  data/imdb_sentiment_dev.jsonl \
                  --base-model-dir serialization_dirs/main_dan_5k_with_emb \
                  --num-epochs 5 \
                  --layer-num 3
```

Similarly, you can also probe the same model on bigram-classification task by just replacing the datasets in the above command.

Analysis & Probing Tasks:

There are four scripts in the code that will allow you to do analyses on the sentence representations:

1. `plot_performance_against_data_size.py`
2. `plot_probing_performances_on_sentiment_task.py`
3. `plot_probing_performances_on_bigram_order_task.py`
4. `plot_perturbation_analysis.py`

To run each of these, you would require to first train models in specific configurations. Each script has different requirements. Running these scripts would tell you these requirements are and what training/predicting commands need to be completed before generating the analysis plot. If you are half-done, it will tell you what commands are remaining yet.

Before you start with plot/analysis section make sure to clean-up your `serialization_dirs` directory, because the scripts identify what commands are to be run based on serialization directory names found in it. After a successful run, you should be able to see some corresponding plots in `plots/` directory.

Following is what you are supposed to do for your analysis -

1. Learning Curves

(a) Performance with respect to training dataset size

Compare the validation accuracy of DAN and GRU based models across 3 different training dataset sizes on sentiment analysis task. Analyze what changes you observe as we increase training data. Use `plot_performance_against_data_size.py` for this analysis.

(b) Performance with respect to training time

You are also supposed to train DAN for more epochs and say something about training and the validation losses from the tensorboard. For this, run the script:

```
./train_dan_for_long.sh
```

and check train/validation losses on the tensorboard (`http://localhost:6006/`) after running:

```
tensorboard --logdir serialization_dirs/main_dan_5k_with_emb_for_50k
```

Note: If you run training multiple times with same name, make sure to clean-up tensorboard directory. Or else, it will have multiple plots in same chart.

2. Error Analysis

- Explain one advantage of DAN over GRU and one advantage of GRU over DAN.

- Use the above to show and explain failure cases of GRU that DAN could get right and vice-versa. Specifically, run your trained models on example test cases (you are free to choose from the datasets or construct your own) and record the success and failure cases. You will see that DAN is able to predict some cases correctly where GRU fails to and vice-versa. Now use these specific cases to portray the one advantage of DAN and the one advantage of GRU that you mentioned.

3. Probing Performances on Sentiment Task

In this task, we are interested in what each layer of our models (both DAN and GRU) are capturing with respect to their ability to predict sentiment. The probing model you trained will be used for this. You will take the frozen trained DAN or GRU model, and apply the linear probing layer to the representations generated by each layer of the base DAN/GRU model for predicting sentiment. In your report add in your observations for each layer and its accuracy(generated on its associated plot). You are also expected to explain your observations based on what you know about the behaviors of DAN and GRU. Use `plot_probing_performances_on_sentiment_task.py` for this analysis.

4. Probing Performances on Bigram Order Task

In this task we are looking at a direct comparison of DAN vs GRU in their ability to predict order of words. Specifically, the task is to classify whether a bigram is in correct order or reversed. For example, `New York` is in correct order and `York New` is reversed. After training both your models you will test them on the reversed bi-gram dataset supplied in the assignment. For doing that, you will take the frozen trained models and use a linear probing layer to classify whether a bigram is in correct order or reversed. Add your plot of accuracy (results) and observations to your report. You are also expected to explain your observations based on what you know about the behaviors of DAN and GRU. Use `plot_probing_performances_on_bigram_order_task.py` for this analysis.

5. Perturbation Analysis

Lastly, we want you to track what happens to a specific input ("`the film performances were awesome`") when it is run through your DAN and GRU networks trained on sentiment task. The word `awesome` is changed to `worst`, `okay` and `cool` for each run. You are supposed to run the script `plot_perturbation_analysis.py` for this analysis. This script calculates and plots the L1 distances between layer representations of the original sentence with `awesome` in the end and the layer representations of the same sentence with the last word replaced by `worst`, `okay` and `cool` corresponding to both DAN and GRU models. Essentially, the plots portray how the representations (which are supposed to embody the sentiment) of the sentence change at every layer when the last word "`awesome`" is perturbed. You are supposed to generate these plots and add it to the report along with your observations. Again, you are also expected to explain these observations based on what you know about the behaviors of DAN and GRU.

What to turn in?

1. Following two files [on Blackboard]:

- `sequence_to_vector.py`
- `probing_model.py`

2. `plots` directory [on Blackboard]
3. `538_HW2_<SBUID>_report.pdf` [on Blackboard] - A PDF format file containing the detailed report with observations and plots(see details below). We encourage you to use LaTeX; but use whatever you are

comfortable with - as long as you stick to the format and the report is interpretable.
4. `serialization_dirs` directory [on Google Drive (publicly accessible)]
5. `gdrive_link.txt` [on Blackboard] - You should include the Google Drive link (publicly accessible) of your `serialization_dirs` directory in this file.
6. (Optional) `README.md` [on Blackboard] - README file containing any additional information you want to convey regarding the submission.

Zip your main submission folder and name it `538_HW2_<SBUID>.zip`. This is exactly what you should submit on Blackboard. Once unzipped, we should get a directory named `538_HW2_<SBUID>` with the following directory structure (again, README file is purely optional) -

```
538_HW2_<SBUID>
├── sequence_to_vector.py
├── probing_model.py
├── gdrive_link.txt
├── 538_HW2_<SBUID>_report.pdf
├── README.md
└── plots/
    ├── performance_against_data_size_dan_with_glove.png
    ├── performance_against_data_size_gru_with_glove.png
    ├── perturbation_response_dan.png
    ├── perturbation_response_gru.png
    ├── probing_performance_on_bigram_order_task.png
    ├── probing_performance_on_sentiment_task_dan.png
    └── probing_performance_on_sentiment_task_gru.png
```

Do not add any other directories like `serialization_dirs` or any other file to your Blackboard submission.

Note that `serialization_dirs` should contain all the serialization files required by the four plotting scripts. Running these scripts with your provided `sequence_to_vector.py`, `probing_model.py` and `serialization_dirs` should generate the `plots` that you submitted. As a sanity check, make sure none of the plotting scripts complain that something is incomplete.

What to write in report

The write up should be 3 to 4 pages of length. You should have the following 3 sections in your assignment report:

1. Model Implementation: Brief (1 page max) description of how you implemented the DAN, GRU and probing models.
2. Analysis: plots and observations/explanations for (i) Learning curves - (a) Performance with respect to training dataset size, (b) Performance with respect to training time; (ii) Error Analysis.
3. Probing Tasks: plots and observations/explanations for (i) Probing Performances on Sentiment Task, (ii) Probing Performances on Bigram Order Task, (iii) Perturbation Analysis.
4. (Optional) References

Please follow this format while writing your report. Keep it concise, but don't skip on important analysis and plots.

Due Date and Collaboration

- The assignment is due on April 8, 2022 at 5:00 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 between third and fourth day will be docked 20%. Submissions between fourth and fifth day will be docked 40%. Submissions after fifth day won't be evaluated.
- 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

This assignment involves a lot of plots and analysis which can be radically different based on what version of Python and other packages you are using. Additionally, you won't be able to do the Tensorboard portion of the assignment on Google Colab without considerable effort. So, we recommend you do this assignment on your local machine since you don't need to train for a long time (each training instance should take around 15 - 20 minutes max on CPU) or tune hyperparameters for this assignment. That being said, if you desperately want to use Colab, you can try changing Python version on Colab to 3.6, install the specified packages in `requirements.txt` and also bypass Colab's firewall to access its localhost ports. However, you will not receive extra credit or extra free late days for this effort.

Some Debugging Tips

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

If you don't know how to use a debugger, here are some basic examples:

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.

Sanity Check

Neural networks are generally over-parameterized. They can almost always fit few examples with a very high accuracy as long as modeling and losses are setup reasonably correctly. So one sanity check is to fit a model on a training dataset of 2-3 examples. If model is struggling to decrease training loss, then something is definitely wrong.

Nans

Nans are known to make people crazy. But most the times reason is one of the following: division by zero, exponent of very large numbers, potential infinities and log of negative numbers. Consider using torch.nan_to_num for instances where you are getting nan.