Homework 2

CS57700 Natural Language Processing, Spring 2023

Due Date: 24 March 2023

1 Coding Assignment

We will be using PyTorch for this part of the assignment. Please refer to the official PyTorch tutorials first if you are unfamiliar with their APIs. Furthermore, please refer to the notes towards the end of this section for the guidance on other external libraries.

1.1 Problem

Word-in-Context [3] is a binary classification task for identifying if the occurrences of the word w in the two contexts correspond to the same meaning or not. For this assignment, you are asked to build a series of classifiers using different combinations of word embeddings and neural architectures, and compare their performance.

1.2 Dataset Description

Please visit the WiC website and download the dataset files. Once you unzip the file, you will find three directories corresponding to train, dev, and test sets. (For your own code, use the files we've already uploaded to `/homes/cs577/WiC_dataset`.)

Label	Target	Context-1	Context-2
F	bed	There's a lot of trash on the bed of the river	I keep a glass of water next to my bed when I sleep
F	land	The pilot managed to land the airplane safely	The enemy landed several of our aircrafts
F	justify	Justify the margins	The end justifies the means
T	beat	We beat the competition	Agassi beat Becker in the tennis championship
T	air	Air pollution	Open a window and let in some air
T	window	The expanded window will give us time to catch the thieves	You have a two-hour window of clear weather to finish working on the lawn

Figure 1: Examples from the WiC dataset.

There are two files for each set, `(train/test/dev).gold.txt` and `(train/test/dev).data.txt`.

- `gold.txt` contains the gold labels for each example, `T` (True: same meaning) or `F` (False: different meaning).
- `data.txt` is a tab-separated file containing the following variables:
  1. Target word: the target word, which is present in both examples.
  2. POS: the Part-of-Speech tag of the target word (either `N`: noun or `V`: verb).
  3. Index 1, Index 2: indicates the index of the target word in example 1 and example 2, respectively.
  4. Example i: corresponds to the i-th example. In this version, all examples are tokenized.

1.3 Your Task

We will be providing three Python code files: `neural_archs.py`, which contains the definition of PyTorch `nn` modules, `utils.py`, which contains the definition of PyTorch dataset module, and `train.py`, the main script for training and testing classifiers. You need to fill in all the `TODO`s across the three files with your own code.

For the WiC dataset, use the ones we've already uploaded to `/homes/cs577/WiC_dataset`.

1.3.1 Core Components

Your code needs to support a classifier with any combination of options from the following categories.

NOTE: Before you submit your code, please change the `default` attributes in line 36 to 38 of `train.py` with the choices that achieved the best *accuracy*. Plus, hardcode your training hyperparameters – the parameters should be the ones that achieved such accuracy for these choices.

Neural architecture For obtaining representations of each input examples, your script must have the option to use any of the three neural architectures below.

1. Deep Averaging Network (DAN): See Lecture 14 slides and [1].
2. RNN: See torch.nn.RNN.
3. LSTM: See torch.nn.LSTM.
4. Bi-LSTM: You can enable bidirectionality by specifying `bidirectional=True` when declaring the LSTM module.

Note that you still need to figure out how to obtain the final representation out of RNN and LSTM.

Word embeddings For the word-level embeddings, please include the option to choose any of the following word embedding choices.

1. Randomly initialized embeddings: You initialize the embeddings of each word randomly and train them from scratch. See the documentation on `torch.nn.Embedding`.
2. GloVe [2]: Pretrained embeddings. Use `gensim` to obtain the `glove-wiki-gigaword-50` version. Please see this link for more details. Note that you still need to copy the embeddings from `gensim` to a `torch.nn.Embedding` instance using `from_pretrained()`.

1.3.2 Advanced Ideas

To achieve better classification performance, you are required to incorporate at least one "advanced" idea into your NN model and/or training. Choose one of the following:

- **Additional features**: In addition to the original features from the dataset, and the word and sentence representations out of word embeddings, what additional features could be used to improve the WiC performance? When feeding in the data to your NN models, you could include the extra feature of your choice in one way or another. Consider looking into the capabilities of `nltk` or using external databases such as `wordnet`. **NOTE: For this idea, you should NOT use any other publicly available pretrained embeddings (e.g. word2vec, fasttext, etc.) besides the GloVe embeddings we are using above.**
- **Multitask learning**: Word Sense Disambiguation (WSD) is a closely related task to WiC. Given a word and the sentence in which the word appears in, we need to determine the "sense" that the word is used within the sentence from a pre-defined sense inventory. Choose one of the datasets introduced in the link and devise a multitask learning scheme across WiC and WSD, to take the WiC performance further.

1.3.3 Model Training

To achieve more optimal performance with neural network, you need to appropriately tune hyperparameters and apply various training techniques such as alternative optimizers. Configure the best working configuration for your case in `train.py`.

1.3.4 Final Output

Your submitted code needs to output `test.pred.txt`, which should contain your *best* model's prediction for the test set. Write predictions (`F` or `T`) for each test example into `test.pred.txt`. It should be one line per each example, in the same order which the examples appear in `test.data.txt`.

NOTE: `test.pred.txt` should NOT be submitted via `turnin`; it should be produced by your submitted codes when we run them on our own.

Notes

Packages Your code will be run under the Anaconda Python environment to be set up using the following commands:

1. Download and install Miniconda to your system with the instructions on this link. If you already have a full Anaconda installation on your system, you can skip this step.
2. From your command prompt or terminal, please run the following: `conda create -n cs577hw2 python=3.7 pytorch=1.12.1 pandas=1.3.5 nltk=3.7 numpy=1.21.5 scipy=1.7.3 gensim=4.1.2`

You can use Python distributions other than Anaconda, but you CANNOT use any external libraries that are not included in the `conda` command above. Moreover, we won't be able to provide any assistance with the setups that were not created using the instructions above.

Running time The system for grading your submissions does not have a GPU installed, so please ensure that your codes could run on CPUs within a reasonable time (hours, not days.)

2 Written Answers (40 points)

2.1 General

1. Discuss the main advantages of LSTM compared to vanilla RNN. (3 points)
2. Assume a hypothetical corpus where it contains occurrences of all possible word senses for every single word. If we train Word2Vec embeddings on this corpus, could the resulting embeddings for each word be used to identify different word senses on their own? Please explain. (4 points)
3. In class, it was suggested that we could solve various NLP tasks by reformulating them as a textual entailment task. How would you approach solving the Word Sense Disambiguation (WSD) problem as an entailment task? You do not have to be too specific, but suggest at least how you would get premise and hypothesis sentences out of the original WSD examples. (4 points)

2.2 Your Experiment

1. Report the specific configuration with the best *accuracy*. Evaluate your best neural network configuration with learning curves. Plot your train, validation, and test set performance on the same plot as the epochs increase and training progresses. What could you tell about the model from this plot? (5 points)
2. Between DAN, RNN, LSTM, and Bi-LSTM, which worked the best in your setup? Was having the explicit ordering with RNN necessary, as opposed to DAN? Why or Why not? (5 points)
3. Was bidirectionality necessary to improve the performance? Please explain the most probable reason why it was or wasn't needed. (5 points)
4. With the neural architecture fixed (the one in your best configuration), how much of the performance difference there was between word embeddings trained from scratch and GloVe? Did GloVe benefit the performance? If so, why do you think it was useful? If it did not help, why do you think that was the case? (5 points)
5. Please answer just one of the following based on the idea you chose to implement in subsubsection 1.3.2 (5 points):
   - Discuss the extra feature(s) you added and the rationale for them. Report the performance improvements you got by adding in that feature.
   - Did multitask learning with WSD datasets improve the performance? Please explain the most probable reason why it was or wasn't needed.
6. Pick one of the hyperparameters you tuned. How did tuning these hyperparameters change the result? Explain with learning curves. (4 points)

Submission Instructions

Coding Assignment

You need to submit your codes via Turnin. Log into `data.cs.purdue.edu` (physically go to the lab or use ssh remotely) and follow these steps:

- Place the following in a directory named `(username)_hw2`. For example, your TA with username `bseoh` would name his directory for HW2 as `bseoh_hw2`.
  - `train.py`
  - `neural_archs.py`
  - `utils.py`
  - any artifacts used in subsubsection 1.3.2
- Execute the following command to `turnin` your code: `turnin -c cs577 -p hw2Spring2023 (username)_hw2`
- To overwrite an old submission, simply execute this command again.
- To verify the contents of your submission, execute this command: `turnin -v -c cs577 -p hw2Spring2023` Do not forget the `-v` option, else your submission will be overwritten with an empty submission.

Written Answers

Upload your written answers as a typed PDF to Gradescope.

- To make grading easier, please start a new page in your PDF file for each question. Hint: use the `newpage` command in LaTeX after every question ends.
- When submitting to Gradescope, match each page of your PDF with question numbers.

References

[1] Mohit Iyyer et al. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1681–1691. DOI: 10.3115/v1/P15-1162. URL: https://aclanthology.org/P15-1162.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[3] Mohammad Taher Pilehvar and Jose Camacho-Collados. *WiC: the Word-in-Context Dataset for Evaluating Context-Sensitive Meaning Representations*. 2018. DOI: 10.48550/ARXIV.1808.09121. URL: https://arxiv.org/abs/1808.09121.