CS7.401: Introduction to NLP — Assignment 4

Instructor: Manish Shrivastava
Deadline: April 20, 2023 — 23:55

1 General Instructions

1. The assignment must be implemented in Python.
2. Ensure that the submitted assignment is your original work. Please do not copy any part from any source, including your friends, seniors, and/or the internet. If any such attempt is caught, serious actions, including an F grade in the course, are possible.
3. A single .zip file needs to be uploaded to the Moodle course portal.
4. Your grade will depend on the correctness of answers and output. In addition, due consideration will be given to the clarity and details of your answers and the legibility and structure of your code.
5. Please start early since no extension to the announced deadline would be possible.

2 ELMo

ELMo (Embeddings from Language Models) is a deep contextualized word representation model developed by researchers at Allen Institute for Artificial Intelligence (AI2) in 2018. Unlike traditional word embedding models, ELMo is capable of capturing the contextual meaning of words by taking into account the entire sentence in which they appear. ELMo's contextualized embeddings have been shown to improve performance on a range of natural language processing tasks, including sentiment analysis, named entity recognition, and question answering.

The ELMo architecture contains forward and backward language models trained through its Bi-LSTMs. The stacked Bi-LSTM-based langauge model has a layer of non-contextual word embeddings (like word2vec, or a character convolutional network as in the original ELMo) at the input. While a forward language model can be trained by the next word prediction task, for a backward language model - to put it simply - it's nothing but previous word prediction task. In either of the cases, the LSTM model makes the word prediction based on the context. By context, we mean all the words preceeding (forward LM) or succeeding (backward LM) the word to be predicted in the sentence.
By training these Bi-LSTMs using the language modeling objective, we essentially pretrain the weights of the ELMo architecture, making it suitable for application on downstream tasks. To break this down, when we pretrain the

model, we make the model learn the nature of the language itself. The model does not really learn any useful task in this process - instead it captures the general dependencies in the language, which makes it smarter to be able to solve any task thrown at it later quickly. This task is what we refer to as the downstream task. On pretraining the model, we expect it to give good performance on a given NLP task with lesser training and data than it would take without pretraining.
Specific to ELMo, a stacked Bi-LSTM is used in order to capture the complexities in natural language in a layered manner - with the lower layers focusing on the syntactic aspects and the higher ones on more complex, semantic aspects. Also, using a Bi-LSTM enables us to capture the context for a given word in the sentence based on all the surrounding words in that sentence. At each layer, we get this bidirectional context of increasing complexity by simply concatenating the representations obtained from the forward and backward LSTMs at each layer. Finally, we add up the representations from each layer of the model (including the non-contextual input embeddings) to have the final contextual word representation of our ELMo architecture. In this assignment, we will implement ELMo from scratch and fine tune it for Sentiment Analysis and Natural Language Inference.

3 Implementation

1. **Preprocessing**: Preprocess the datasets by performing tasks such as tokenization, stemming, and stop-word removal. You may also need to split the dataset into train, validation, and test sets.

2. **Build the Elmo model**: Build the Elmo model using pytorch. It should contain a 2 layered stacked Bi-LSTM each of which gives the embedding for a word in a sentence, and a trainable parameter for weighing the word embeddings obtained at each layer of the ELMo network. Use GloVe or FastText or word2vec as pre-trained static embeddings followed by an Elmo embedding layer which will be trained as a part of the model.

3. **PreTraining the Elmo model**: Train the Elmo model on the preprocessed datasets separately. You can use the datasets as an unlabeled corpus for the purpose of this word prediction step. You should have two models at the end of this task each trained on one corpus.

4. **Evaluate the Elmo model**: Build a pipeline by adding classifiers on top of the pre-trained models and Evaluate the performance of the model on the respective test dataset by measuring metrics such as accuracy, precision, recall, and F1 score. You can use sklearn classification report to get these scores

5. **Visualize the results**: Visualize the results of the models by generating visualizations such as confusion matrices and ROC curves.

Note: You need to perform steps 3-6 for both the datasets separately and report their results

4 Datasets

1. **Sentiment Analysis**:
   Stanford Sentiment Treebank (`https://huggingface.co/datasets/sst`)
2. **Natural Languagle Inference**:
   Multi-Genre NLI Corpus (`https://huggingface.co/datasets/multi_nli`)

5 Grading

Evaluation will be individual and will be based on your viva, report, and code review. You will be expected to walk us through your code and explain your results during your evaluation. You will be graded based on the correctness of your code, accuracy of your results and quality of the code.

- Dataset Preprocessing: 10 Marks
- Building the model: 25 Marks
- Training the Embeddings for both downstream tasks: 15 Marks each
- Results and analysis: 20 marks
- Report: 5 Marks
- Viva: 10 Marks

6 Submission format

Zip the following into one file and submit it on the Moodle course portal:

1. Source code (Should include .py files only.)
2. Pre-trained model and the generated embeddings
3. Readme file (should contain instructions on how to execute the code, restore the pre-trained model, and any other information necessary for clarity)
4. Report containing the graphs and scores

Name the zipped file as 'roll number'_assignment4.zip,
e.g.: 2022xxxxxx_assignment4.zip.
Note: If the pre-trained models and embeddings cross the file size limits, upload them to a OneDrive/GDrive and share the read-only accessible links in the readme file

7 FAQ

1. **Can I submit my code in Jupyter Notebook?**
   No, the final submission should be a Python script. You may work using Jupyter Notebooks, but make sure to convert them to .py files before submitting.

2. **Do I need to code everything from scratch?**
   No. You are free to use neural and LSTM layers from deep learning frameworks like PyTorch / Keras etc.

3. **My model takes too long for train/test. What should I do?**
   Ensure you are using a GPU environment with proper batch sizes for training. Look into optimisations that can be made in the forward function or elsewhere.

4. **My model size is greater than the limit allowed on moodle. What should I do?**
   Place the read-only link in the readme file after uploading the model to any cloud storage. Mention every specific step involved in running the model.

5. **Can I use libraries for automatic preprocessing?** Yes, you can use libraries like torchtext and NLTK for this (including word tokenization).