Assignment 2
CSE 447/547M: Natural Language Processing

University of Washington

Due: February 7, 2019, 11:59pm

Your final writeup for this assignment, including the problem and the report of the experimental findings of the programming part, should be no more than four pages long. You should submit it as a PDF. We *strongly* recommend typesetting your scientific writing using LaTeX. Some free tools that might help: TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Overleaf (online).

Task: Sentiment Analysis

In this assignment, you will be building text classifiers for the task of *sentiment analysis*. One goal of sentiment analysis is to determine if a text (e.g., a sentence) has *positive* or *negative* sentiment. You will be using the Stanford Sentiment Treebank (SST; Socher et al., 2013); this is a standard sentiment analysis dataset based on movie reviews. We have provided a starter configuration (`skeleton.jsonnet`), which points to the SST dataset files and AllenNLP SST `DatasetReader`.[1]

1 Programming: Text Classification with RNNs (35%)

In this programming problem, you will use AllenNLP to build a RNN-based text classifier. More specifically, you will implement and provide configuration for an AllenNLP `Model` which performs the following:

1. Embed the input text as a sequence of vectors.
2. Transform the sequence of embeddings into a vector using a single-layer, simple RNN.
3. Apply a feed-forward layer on that vector to obtain a label.

Though this seems like a detailed specification, you will still need to make several hyperparameter decisions, including but not limited to:[2]

- number of training epochs
- training batch size
- choice of nonlinearity
- choice of optimization method
- dropout rate

[1]You may notice that the dataset is in a rather unusual format: each line contains a syntactic parse tree of a sentence, and each node of the tree has a label on a 1–5 scale (where 1 is most negative and 5 is most positive). The aptly-named `StanfordSentimentTreeBankDatasetReader` handles transforming this format into the expected list of sentences and a positive/negative label per sentence. The task we are interested in in this assignment doesn't care about the parse trees, which we haven't discussed in lecture yet.

[2]Other config files are a good place to see what other hyperparameters are set in practice: `https://github.com/allenai/allennlp/tree/master/training_config`

Vary a single hyperparameter and use the development data to select the best value; you may keep the others fixed for the purposes of this assignment (but recognize that, in general, hyperparameter choices interact).

You can develop and run your code on the course server, `aziak.cs.washington.edu`. You will turn in your source code along with the PDF writeup. Your submission will not be evaluated for efficiency, but we recommend keeping such issues in mind to better streamline the experiments. You may find that writing unit tests will help speed up the development process; details on how to do so are provided in the AllenNLP tutorials.[3] We have also provided a `Predictor` class that you can use to test out arbitrary input sentences. The later sections of this assignment will build off of the model you create here, so we recommend (as always) that you start early.

Deliverables: In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report the accuracy of your model on the training and development sets across the hyperparameter values you tested (at most 5).
2. Take the best-performing model from (1) and report its accuracy on the test set.

Note: For ease of grading, register your model with the name `sentiment_classifier` and provide your configuration in `rnn-simple.jsonnet`.

2 Experimentation: Recurrent Units (30%)

As discussed in lecture, the simple RNN suffers from two related problems:

- "Vanishing gradients" during learning make it hard to propagate error into the distant past.
- State tends to change a lot on each iteration; the model "forgets" too much.

In practice, we often use more complex recurrent units, like long short-term memories (LSTMs; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs; Cho et al., 2014).[4] Update your model to use one of these units instead, and compare its performance to the simple-RNN-based one from section 1.

Deliverables: In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Report your LSTM/GRU-based model's training, development, and test set accuracy (following the same experimental procedure as in section 1).
2. We might expect the LSTM/GRU model to have better accuracy than the simple RNN model on longer sequences. Does this hold for your models? Provide evidence and/or examples to demonstrate.

Note: For ease of grading, provide the configuration for your updated model as `rnn-complex.jsonnet`.

3 Experimentation: Pretrained Word Embeddings (35%)

Pretrained word embeddings are word embeddings that are already constructed in advance of your NLP project (whether your project is a neural language model, a text classifier, or a class assignment). The advantage of pretraining is that it simplifies learning your model, because the embedding parameters are

[3]Guide for creating a test class for your model: `https://github.com/allenai/allennlp/blob/master/tutorials/getting_started/predicting_paper_venues/predicting_paper_venues_pt1.md#step-four-write-your-model`. Once you have created a test class for your model, you can run it using `pytest` from the command line.

[4]Sections 15.1-3 of Goldberg [2017] provide a more detailed comparison of simple RNNs, LSTMs, and GRUs.

fixed in advance. The disadvantage is that, if your pretrained embeddings happen to be bad for your task, you're stuck with them.[5]

In the previous sections, we gave you the configuration for 300-dimensional word2vec embeddings [Mikolov et al., 2013a,b] trained on a large Google News corpus.[6] For this problem, select another set of pretrained word embeddings and update your LSTM/GRU model to use those instead. You are free to use any pretrained embeddings you can find, as long as you cite their papers in the writeup. Here are some alternatives:

- GloVe [Pennington et al., 2014]: `https://nlp.stanford.edu/projects/glove`
- FastText [Mikolov et al., 2018]: `https://fasttext.cc/docs/en/english-vectors.html`
- Dependency-based word embeddings [Levy and Goldberg, 2014]: `https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/`
- Paraphrastic word embeddings [Wieting et al., 2016]: `https://www.cs.cmu.edu/~jwieting/`

(Warning: not all pretrained embeddings may be in a format supported by AllenNLP out of the box, and most of these files are large (1 GB+). We have pre-downloaded GloVe embeddings to aziak in case you run into issues; refer to the skeleton config.)

Note that embeddings vary not only in the method of construction from data, but also in dimensionality, amount and type of text used, and more. When you report on your choices, be precise!

Deliverables: In the writeup, be sure to fully describe your models and experimental procedure. Provide graphs, tables, charts or other summary evidence to support any claims you make.

1. Briefly describe the differences between the two sets of pretrained embeddings you used.
2. Report the training, development, and test set accuracy of your updated model (following the same experimental procedure as in section 1). How does performance change when you use different embeddings?
3. One known characteristic of word embeddings is that antonyms (words with meanings considered to be opposites) often have similar embeddings. You can verify this by searching for the top 10 most similar words to a few words, like *increase* or *good*, that have clear antonyms (e.g., *decrease* and *bad*, respectively) using cosine similarity.[7] Discuss why embeddings might have this tendency.
4. Select a word w that you think may be informative to a sentiment classifier, along with its antonym w'. How similar are their word embeddings? Create some input example sentences that contain w, and report if swapping w out for w' in those sentences changes your classifier's prediction.

Note: For ease of grading, provide the configuration for your alternate embedding model as `embed.jsonnet`.

4 Bonus: Contextual Word Embeddings (10%)

Recently, *contextual* word embeddings have become popular due to state-of-the-art performance gains on many tasks. These embeddings are described as "contextual" because the embedding of a word is not just dependent on the word itself, but also the context in which it appears.

For bonus points, update your model configuration to use ELMo [Peters et al., 2018], one such method for obtaining contextual embeddings; you may find the AllenNLP ELMo tutorial helpful for doing so.[8]

[5]Not to make things too complicated, but there is a version of pretraining where you first pretrain the embeddings, then use them to *initialize* the embeddings in your model, but allow the learner to "fine-tune" the embeddings while it's learning the other parameters. Whether and how to do this is an active area of research.

[6]`https://code.google.com/archive/p/word2vec/`

[7]It may help to use other libraries for finding similar words or computing similarity between two words, e.g., `spacy` (used by AllenNLP under the covers) or `gensim`.

[8]Pretrained ELMo models: `https://allennlp.org/elmo`. ELMo tutorial: `https://github.com/allenai/`

Train and evaluate this following the same experimental procedure as before, and in your report, describe how performance differs. Furthermore, note particular types of input for which including ELMo is helpful, and provide examples which support this.

Submission Instructions

Submit a single gzipped tarfile (`A2.tgz`) on Canvas, with the following:

- **Code**: You will submit your model code and configuration (with filenames specified in the previous sections). You are also welcome to submit other code you write over the course of this assignment (e.g., for error analysis); if so, explain what it does in a README. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. However, please provide well-commented code if you want partial credit.
- **Report** (use the filename `A2.pdf` and include in the tarfile): Your writeup should be at most four pages long, in PDF format (one-inch margins, reasonable font sizes, LaTeX-typeset). Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

References

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

Yoav Goldberg. *Neural network methods for natural language processing*. Morgan Claypool (Synthesis Lectures on Human Language Technologies), 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *ACL*, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013a.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *NAACL-HLT*, 2013b.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *LREC*, 2018.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *EMNLP*, 2014.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL*, 2018.

`allennlp/blob/master/tutorials/how_to/elmo.md`

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 2013.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. In *ICLR*, 2016.