TTIC 31210: Advanced Natural Language Processing
Assignment 1: Simple Neural NLP

Kevin Gimpel

Assigned: March 27, 2017

Due: 11:59 pm, April 10, 2017

Submission: email to `kgimpel@ttic.edu`

Submission Instructions

Package your report and code in a single zip file or tarball, name the file with your last name followed by "_hw1", and email the file to `kgimpel@ttic.edu` by 11:59 pm on the due date. To limit file size, you should NOT include any data files that we provided to you (we already have them!). For the report, please use LaTeX and some standard style files, such as those prepared for ACL or ICLR. Your report should be a maximum of 6 pages.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code must be written individually.

Lateness Policy

We want you to do the assignments, even if that means turning them in late (whether partially or fully). There will likely be a penalty assessed if assignments are turned in after the due dates, but we will continue to accept late submissions of assignments until the end of the quarter.

Overview

This is a short warm-up assignment to ensure that we're all on the same page, as it assumes knowledge of some essential material from TTIC 31190. If you didn't take 31190, this assignment is an opportunity for you to assess whether you're prepared to take this course.

The goal of this assignment is to develop skills in the following:

- formal definition and exposition of models
- implementation and experimentation using deep learning toolkits
- analysis of trained models
- brainstorming new model variations and analysis techniques

These are all fundamental skills in the research cycle. For those who took TTIC 31190, the implementation and experimentation will be relatively simple in this assignment, so please focus on strengthening your skills in formal exposition, model analysis, and developing new methods.

You will implement and experiment with ways of adding attention to a simple sentence classifier based on word averaging. You will use the binary sentiment analysis task developed from the Stanford sentiment treebank movie review dataset (Socher et al., 2013). There are three files (posted on the course webpage):

- `senti.binary.train`: training data (TRAIN)
- `senti.binary.dev`: development data (DEV)
- `senti.binary.test`: test data (TEST)

Each line in each file contains a textual input followed by a tab followed by an integer containing the gold standard label (0 or 1). For your evaluation metric, use classification accuracy, i.e., the percentage of sentences that were classified correctly.

Your Tasks (100 points total)

1. Word Averaging Sentence Classifier (40 points)

1.1 Exposition (10 points)

Formally describe the word averaging model for binary classification and the loss function for training. The model should average the vectors for the words in the input sequence, where d is the dimensionality of the word vectors. The model should then compute the dot produce between that d-dimensional average and a d-dimensional parameter vector, yielding a scalar. This scalar will be passed through a logistic sigmoid function to obtain the probability of one of the output classes (the probability of the other class is obtained by subtracting the first from 1). Use log loss as the objective function (log loss is often called "cross entropy" when training neural networks). During learning, learn both the word embeddings and the additional model parameters.

I deliberately described the above in words so that you can see how annoying this is. Describe your model in words *and* math! Develop notation that is simple and intuitive while still being accurate and precise. Make sure every variable is defined and that all of the equations "compile". Ensure that dimensions and types match. Be clear about what things are parameters to be learned and what things are from the data. These are important practices to follow when writing papers—many papers get rejected for notational inconsistencies or buggy equations. Getting this part right will also help you to more precisely define and compare model variations below.

1.2. Implementation (15 points)

Implement the model and learning procedure. Submit your code.

You are encouraged to use a deep learning toolkit for this assignment (though you are welcome to implement the model and backpropagation from scratch if you prefer). To simplify things, you don't need to use mini-batching, though of course you can if you want. In my implementation (using DyNet), running the experiments took just a few minutes without any mini-batching.

You should randomly initialize the word embeddings and learn them along with the other model parameters. For optimization, use stochastic gradient descent or any other optimizer you wish. Toolkits typically have many optimizers already implemented.

1.3. Experimentation (10 points)

Run experiments on the binary sentiment classification task. Train on TRAIN, perform tuning and early stopping on DEV, and report your final classification accuracy on TEST.

1.4. Analysis (5 points)

Compute the squared norms of the learned word vectors after training. What do you notice about the words with the largest norms? How about the smallest? Does anything surprise you?

2. Attention-Augmented Word Averaging (60 points)

2.1. Exposition (5 points)

Define a new model that replaces the average with an attention-weighted sum. Introduce a new parameter vector $\mathbf{w} \in \mathbb{R}^d$ for computing the attention weighting. Compute the unnormalized attention weight using a dot product between the word embedding and $\mathbf{w}$ followed by an exponential function. Normalize across all words in the sentence to get attention weights for each word. Multiply the attention weights by the word embeddings and sum the attention-weighted word embeddings. I described this in words above, but you should define it formally using words and math.

2.2. Implementation and Experimentation (15 points)

Implement the model (submit your code) and repeat the experiments on the dataset. Compare the results to the baseline.

2.3. Analysis (10 points)

Compute the attention weights of the words in the sentences in the development data. Report trends that you observe in the attention weights.

Compute aggregate statistics of attention weights for words and look at three categories of words: (1) those that tend to have large, low-variance attention weights, (2) small, low-variance attention weights, and (3) high-variance attention weights. What do you observe about the words that fall into these three categories? For the words with high-variance attention weights, you may find it helpful to look at the individual sentences that contain instances with low/high attention weight. You may also find it helpful to write your own short examples to more directly identify the causes of the high variance in the attention weights.

2.4. Enriching the Attention Function (30 points)

Enrich the attention function. Our simple attention function above only used the word at the current position, but you can enrich it to use any information about the sentence, including the relative/absolute position of the word in the sentence, the sentence length, nearby words, the presence of negation words before or after the word, information from a part-of-speech tagger or syntactic parse of the sentence, or anything else you can think of.

When making each change, how are the learned attentions on the development set affected? What is the impact on the test set accuracies? Explore three distinct ideas here for full credit (10 points for each). For each idea, write up the new attention function formally, implement it, run experiments, and analyze the results.

References

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*. [1]