TTIC 31210: Advanced Natural Language Processing
Assignment 2: Attention (70 points)

Instructor: Kevin Gimpel
Assigned: Wednesday, April 17, 2019
Due: 7:00 pm, Wednesday, May 1, 2019
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 first and last name followed by "_hw2", and email the file to `kgimpel@ttic.edu` by the due date and time.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code should be written individually. You may modify code you find online, but you must provide attribution and be sure that you understand it!

Overview

In this assignment, you will experiment with neural architectures for sentiment classification, specifically focusing on attention in sentence encoders.

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). The zip file posted on the course webpage contains the following files:

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

Evaluation

You will train binary sentiment classifiers in this assignment. For your evaluation metric, use classification accuracy, i.e., the percentage of inputs that were classified correctly.

Use DEV for early stopping. That is, when you report results, report the TEST accuracy for the model that achieves the best accuracy on DEV. You should also report the best accuracy achieved on DEV. In order to do early stopping, you should compute the classification accuracy on DEV periodically during training. Do this at least once per epoch, and preferably 2 or more times per epoch.

Below we will abbreviate this evaluation procedure as EVAL. To summarize, EVAL consists of training on TRAIN, using DEV for early stopping (using classification accuracy as the early stopping criterion), and reporting the best classification accuracy on DEV and the classification accuracy on TEST using the model that did best on DEV.

1. Word Averaging Binary Classifier (20 points)

We will denote a sentence by $\boldsymbol{x} = \langle x_1, x_2, ..., x_{|\boldsymbol{x}|}\rangle$ where x_t is the tth word in $\boldsymbol{x}$. We will use emb to denote the word embedding function, i.e., $emb(x)$ returns the embedding (a vector in $\mathbb{R}^d$) for word x. We will define our first encoder using simple word embedding averaging:

$$\mathbf{h}_{avg} = \frac{1}{|\boldsymbol{x}|}\sum_t emb(x_t)$$

Then, the probability of positive sentiment is given by

$$\sigma(\mathbf{w}^\top \mathbf{h}_{avg})$$

where σ is the logistic sigmoid function and $\mathbf{w} \in \mathbb{R}^d$ is a parameter vector. If $\sigma(\mathbf{w}^\top \mathbf{h}_{avg}) \geq 0.5$, the classifier should return positive sentiment; otherwise, it should return negative sentiment.

Use binary log loss as the loss function during training. The parameters of the model are those used in the embedding function emb and the vector $\mathbf{w}$. Note here that the dimensionality d of the word embeddings must equal the dimensionality of $\mathbf{w}$. Some words appear in DEV or TEST but do not appear in TRAIN. Randomly initialize embeddings for these words just like those that appear in TRAIN, but simply keep them fixed during training. Be careful when randomly initializing word embeddings; if you use too large of an initialization range, the unknown words will have large-norm embeddings that may negatively affect your results and analysis. For example, I found that initializing embedding parameters to be between -0.1 and 0.1 worked much better than using values between -1 and 1.

For optimization, use Adam, stochastic gradient descent, or any other optimizer you wish. Toolkits typically have many optimizers already implemented. Document what optimizer you used and the learning rate (and other relevant hyperparameters). Use a dimensionality d of at least 100 for your experiments and train for at least 5 epochs.

1.1. Implementation and Experimentation (15 points)

Implement the model and learning procedure. Submit your code. Run EVAL and report your DEV and TEST accuracies. You should be able to reach approximately 80% accuracy.

1.2. Analysis (5 points)

Compute the L_2 norms of the word embeddings after training, using the model with the highest accuracy on DEV. Print the 15 words with largest norms and the 15 words with smallest norms. What do you notice about the words with the largest/smallest norms?

2. Attention-Weighted Word Averaging (20 points)

We will now define an encoder that uses a simple attention function to produce a weight for each word in the sentence, followed by a sum of the attention-weighted word embeddings:

$$\alpha_t \propto \exp\{\cos(\mathbf{u}, emb(x_t))\}$$

$$\mathbf{h}_{att} = \sum_t \alpha_t emb(x_t)$$

Then, the probability of positive sentiment is given by

$$\sigma(\mathbf{w}^\top \mathbf{h}_{att})$$

where σ is the logistic sigmoid function and $\mathbf{w} \in \mathbb{R}^d$ is a parameter vector, just like what we used in Section 1. This model introduces a new parameter vector $\mathbf{u} \in \mathbb{R}^d$ used in the attention function.

In this model, the unnormalized attention weight for a word x is computed using the cosine similarity between $\mathbf{u}$ and the word embedding for x followed by exponentiation. To get normalized weights α_t, normalize across all words in the sentence. Then multiply the attention weights by the word embeddings and sum the attention-weighted embeddings. Unlike the model in Section 1, we don't need to divide the sum by $|\boldsymbol{x}|$ because the attention weights α_t are normalized such that they sum to 1 across all words in the sentence. (Note: using dot product is more common than cosine similarity when computing attentions like these, but I suggest using cosine similarity instead because it makes training more stable.)

2.1. Implementation and Experimentation (10 points)

Implement the model (submit your code) and run EVAL. Report your results.

2.2. Analysis: Word Embeddings and the Attention Vector (5 points)

Using the model with the highest DEV accuracy, compute the cosine similarity between $\mathbf{u}$ and all word embeddings and print the 15 words with highest cosine similarity to $\mathbf{u}$ and the 15 with lowest cosine similarity to $\mathbf{u}$. Why do you think those words have high/low cosine similarity to $\mathbf{u}$ (and therefore high/low attention weights on average)? Can you form a hypothesis to explain what you see?

2.3. Analysis: Variance of Attentions (5 points)

Using the model that does best on DEV, compute the attention weights for all words in the sentences in the training data. For each word that appears at least 100 times in the training data, compute the mean and standard deviation of the attention probabilities for that word. Sort the words according to the criterion "standard deviation divided by mean" and print the top 30 words under this criterion. These are words that show a large standard deviation in their attention probabilities relative to their average attention probabilities. How would you describe these words and/or why do you think they show these characteristics?

3. Simple Self-Attention (15 points)

We will now define an encoder that uses a simple form of self-attention when producing attention weights for each word in the sentence:

$$a_{ts} = emb(x_t)^\top emb(x_s)$$

$$\alpha_t \propto \exp\left\{\sum_s a_{ts}\right\}$$

$$\mathbf{h}_{self} = \sum_t \alpha_t emb(x_t)$$

Then, the probability of positive sentiment is given by

$$\sigma(\mathbf{w}^\top \mathbf{h}_{self}) \tag{1}$$

The unnormalized attention weight for a word x is computed using the dot product between its embedding and those for all other words in the sentence, followed by a summation and exponentiation. Unlike the model in Section 2, this model does not introduce any new parameters for computing the attention

function, simply using the same word embeddings for the attention. Therefore, this model has the same number of parameters as the model in Section 1.
For improved stability, we can also add a "residual connection", which would change Eq. 1 to

$$\sigma(\mathbf{w}^\top (\mathbf{h}_{self} + \mathbf{h}_{avg})) \tag{2}$$

where $\mathbf{h}_{avg}$ is computed as in Section 1 (though using the same word embeddings as in $\mathbf{h}_{self}$).

Implement this model (submit your code) and run EVAL both with and without the residual connection, i.e., using Eq. 2 and Eq. 1, respectively. Report your results.

4. Enriching the Attention Function (15 points)

Hopefully you've developed some intuition for using attention for this task. Now, come up with your own ways of modifying the attention function and experiment with them. Can you find an idea that outperforms your models from Sections 1-3?

Some potential ideas are below:

- Use transformation matrices to distinguish key, query, and value representations
- Use multiple attention heads
- Use positional encodings (either learned or fixed) in the attention function and/or in the word representations
- Add additional layers of self-attention before the attention-weighted sum of embeddings
- Compute features in the attention function based on characteristics of where the word is in the sentence, e.g., features of 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, etc.
- Use multiple word embedding spaces for when words are used as keys, queries, and values, or some subset of the three.
- Change the dot product/cosine similarity used in Sections 2-3 to a parameterized function, such as a bilinear function or feed-forward network

Describe your best new attention function formally in your report, along with the experimental results. Submit your code.

Extra credit may be rewarded for particularly creative, effective, and well-written solutions.

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*.