TTIC 31210: Advanced Natural Language Processing
Assignment 2: Sequence Modeling

Kevin Gimpel
Assigned: May 1, 2017
Due: 11:00 pm, May 17, 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 "_hw2", and email the file to `kgimpel@ttic.edu` by 11:00 pm on the due date. For the report, use LaTeX and some standard style files, such as those prepared for ACL or ICLR. Your report should be a maximum of 8 pages.

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

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

In this assignment, you will experiment with sequence modeling with neural networks.

Data

You will use a simplified story dataset that I prepared from the ROC story corpus (Mostafazadeh et al., 2016). I converted most names to Bob and Sue and extracted a subset of sentences that only use words from a small vocabulary. The tarball posted on the course webpage contains the following files:

- `bobsue.lm.train.txt`: language modeling training data (LMTRAIN)
- `bobsue.lm.dev.txt`: language modeling development data (LMDEV)
- `bobsue.lm.test.txt`: language modeling test data (LMTEST)
- `bobsue.seq2seq.train.tsv`: sequence-to-sequence training data (SEQTRAIN)
- `bobsue.seq2seq.dev.tsv`: sequence-to-sequence development data (SEQDEV)
- `bobsue.seq2seq.test.tsv`: sequence-to-sequence test data (SEQTEST)
- `bobsue.voc.txt`: file containing vocabulary, with one word type per line

Each line in the `lm` files contains a sentence in a story. Each line in the `seq2seq` files contains a sentence in a story followed by a tab followed by the next sentence in the story. Note: the second field in each line of each `seq2seq` file is identical to the corresponding line in the corresponding `lm` file. (That is, `cut -f 2 bobsue.seq2seq.x.tsv` is the same as `bobsue.lm.x.txt`.) The complete vocabulary is contained in the file `bobsue.voc.txt`, one word per line. You do not have to worry about unknown words in this assignment.

Evaluation

You will train language models in this assignment. However, to evaluate, you should use **word prediction accuracy** as your primary evaluation metric rather than perplexity. We are doing this because perplexity is not helpful when you are trying to compare certain loss functions.

To be more specific, you should evaluate your model's ability to predict each token beyond the start-of-sentence symbol (⟨s⟩) in each line of the LMDEV (or LMTEST) file. When using the SEQ files, you should evaluate your model's ability to predict each token beyond the start-of-sentence symbol in the second tab-separated field of each line.

In the DEV files, there are 7957 predictions to make. In the TEST files, there are 8059 predictions to make. You can check your numbers of predictions against these to ensure you are computing these accuracies correctly. For each prediction, you should use the previous ground truth words in the sentence. You should not test your ability to predict the initial ⟨s⟩ at the start of each sentence, but you definitely should test your ability to predict the ⟨/s⟩ symbol at the end of each sentence!

When you have a softmax layer, the predicted word is the most-probable word in the softmax. When you are using hinge loss with negative sampling, the predicted word is the highest-scoring word.

Use the DEV files 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 and also to answer some of the questions below, you should compute the loss and word prediction accuracy on the relevant DEV set periodically during training. Do this at least once per epoch, and preferably 2 or more times per epoch.

Your Tasks (125 points total)

1. Language Modeling (65 points)

You will train LSTM language models on LMTRAIN. If you like, you can experiment with different dimensionalities and architectural variations. But by default, you can just use a single-layer LSTM with dimensionality 200 for both the word embeddings and the LSTM cell/hidden vectors.

Each experiment will take the following form: Train on LMTRAIN, use LMDEV for early stopping (using word prediction accuracy as the early stopping criterion), and report your final word prediction accuracy on LMTEST. Below we will refer to this evaluation procedure as EVALLM.

1.1 Implementation and Experimentation (15 points)

Implement your LSTM language model and training procedure. Use a softmax layer on the hidden vector at each time step to predict the word at that step. Train by minimizing log loss (cross entropy). 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.

Document in your write-up what you use along with the optimizer hyperparameter values you use. 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. With my implementation (using DyNet), running each experiment took less than 2 minutes per epoch on my laptop without any mini-batching.

Evaluate using the EVALLM protocol. Report your results.

1.2. Error Analysis (10 points)

Implement the capability to print the word prediction errors made by your model. Let's define an **error** as a pair $\langle y_g, y_p \rangle$ where y_g is the ground truth word, y_p is the model's prediction, and $y_g \neq y_p$.

- **(a)** List the top 20 most frequent errors made by the LSTM language model on LMDEV.
- **(b)** Inspect the common errors to identify common error categories, providing examples of each category as you define it. You do not have to categorize every error, but you should spend some time looking at the errors to determine if there are groups of related errors. To help yourself identify error categories, think about the following as you look through the errors: Why do you think the model might have predicted what it predicted? How could the model have done better? Was the model "close"? If so, in what way was the model close?

We will do more with these errors in Section 2 below.

1.3. Hinge Loss Implementation (10 points)

Now you'll experiment with a different loss function for training your language model. Before describing the loss, let's introduce some notation. We use $\boldsymbol{y}$ to refer to a sentence and we use y^t to refer to the word at position t in $\boldsymbol{y}$. We assume that $y^1 = \langle \mathrm{s} \rangle$, the special start-of-sentence token, and that $y^n = \langle / \mathrm{s} \rangle$, the end-of-sentence token where $n = |\boldsymbol{y}|$ is the length of $\boldsymbol{y}$. When feeding word y to the LSTM, we define the input embedding function that returns the embedding of y by $emb_i(y)$. We'll use $\boldsymbol{h}^t$ to denote the LSTM hidden vector at position t (which you used above to predict y^t using a softmax layer). Given the above notation, we will define our hinge loss as follows:

$$\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{y} \in D} \sum_{t=2}^{|\boldsymbol{y}|} \sum_{y' \in NEG} \max(0, 1 - \mathrm{score}(y^t, \boldsymbol{h}^t) + \mathrm{score}(y', \boldsymbol{h}^t)) \tag{1}$$

where $\boldsymbol{\theta}$ contains all model parameters, D is the training set of sentences, NEG is a set of negative examples, and $\mathrm{score}(y, \boldsymbol{h})$ outputs a scalar score under the model for outputting word y from hidden vector $\boldsymbol{h}$. Note that t begins at 2 because $y^1 = \langle \mathrm{s} \rangle$ always, so it doesn't need to be predicted. For the form of the score function, use the following:

$$\mathrm{score}(y, \boldsymbol{h}) = emb(y)^\top \boldsymbol{h} \tag{2}$$

where $emb(y)$ is some embedding function. This could the embedding function used for the inputs (emb_i), or it could be a different embedding function (be sure that your implementation can support both options!). Implement training for this loss and submit your code.

1.4. Hinge Loss Experiments (10 points)

- **(a)** Run EVALLM using the above hinge loss where NEG equals the entire vocabulary and using $emb = emb_i$. That is, use the same embeddings for both input and output. Compare your test accuracy to the LSTM trained with log loss.
- **(b)** Repeat (a) but let emb be a new "output" embedding function emb_o that is learned. Compare results to (a).
- **(c)** Using the configuration from (b) above, now experiment with two additional choices for NEG that sample r words uniformly at random from the vocabulary. Experiment with both $r = 100$ and $r = 10$. Compare the results to (b).

1.5. Loss Function Comparison and Analysis (20 points)

One of the main motivations for hinge loss and negative sampling is training time. But there are several ways to measure this. Consider the following three efficiency measures: (1) sentences processed per second ("#sents/sec"), (2) number of sentences processed in order to reach maximum LMDEV accuracy ("#sents for max acc"), and (3) wall-clock time needed to reach maximum LMDEV accuracy ("time for max acc"). You should compute the three efficiency measures above for log loss, hinge loss where NEG is the entire vocabulary, and hinge loss where NEG is random with $r = 10$. In particular, do the following (each is worth 2 points, except (b) and (c) which are each worth 3 points):

- **(a)** Compute the #sents/sec values for the three losses and report them in a table.
- **(b)** Create a plot showing numbers of sentences processed on the horizontal axis and LMDEV word prediction accuracy on the vertical axis. The plot should have three curves, one for each loss.
- **(c)** Create another plot where the horizontal axis shows wall-clock time and the vertical axis is again LMDEV accuracy. The plot should again have three curves.
- **(d)** Make observations about the above statistics and plots.

After generating those statistics/plots, answer the following questions (this may require additional reporting functionality in your code):

- **(e)** When training with log loss, does the best LMDEV log loss occur at the same time as the best LMDEV word prediction accuracy? Or do they occur at very different times during training?
- **(f)** When training with hinge loss, does the best LMDEV hinge loss occur with the best LMDEV word prediction accuracy? Or do they occur at very different times?
- **(g)** When training with log loss, as you train for longer and longer, what do you notice about the LMDEV log loss and LMDEV prediction accuracy?
- **(h)** When training with hinge loss, as you train for longer and longer, what do you notice about the LMDEV hinge loss and LMDEV prediction accuracy?
- **(i)** If you had a very large training set, which loss would you use?

2. Sequence-to-Sequence Models (60 points)

You will now train sequence-to-sequence (SEQ2SEQ) models on the same type of data. In particular, the words you need to predict in the SEQDEV and SEQTEST files are identical to those in the LMDEV and LMTEST files, so this will let you see the additional contribution to accuracy from having the previous sentence. Hopefully your results here will be better than in Section 1!

As the decoder in your model, use an LSTM just like you used in Section 1. Below you will experiment with different encoders.

When you run experiments in this section, do the following: Train on SEQTRAIN using log loss, use SEQDEV for early stopping, and report your final word prediction accuracy on SEQTEST. Below we will abbreviate this as EVALSEQ2SEQ.

2.1. Encoder Implementation and Empirical Comparison (30 points)

Implement and experiment with the following three variations for the encoder (submit your code), using log loss for training:

- **(a)** Use a forward LSTM as the encoder. Use the final cell and hidden vectors as the initial cell and hidden vectors for the decoder. Run EVALSEQ2SEQ and compare the accuracy to your log-loss-trained LSTM language model from Section 1.
- **(b)** Use a bidirectional LSTM as the encoder. Where the decoder's hidden state has dimension d, use $d/2$ as the dimensionality for the forward and backward LSTM in the encoder. Concatenate the final cell vectors from the forward and backward LSTMs to use as the initial cell vector for the decoder, and do the analogous procedure for the hidden vectors. Run EVALSEQ2SEQ and compare the accuracy to the setting from (a).
- **(c)** Use a bag-of-words encoder that simply averages the word embeddings of the previous sentence in the pair. There are multiple reasonable ways to handle the transition between the encoder and decoder. One way is to use the word embedding average as both the initial cell vector and the initial hidden vector of the decoder LSTM. Or you could add a layer between the word embedding average and the cell/hidden vectors used to initialize the decoder LSTM. Document what you do in your write-up. Run EVALSEQ2SEQ and compare to the above encoders.

2.2. Error Analysis (15 points)

- **(a)** Report the most frequent errors made by the forward LSTM SEQ2SEQ model.
- **(b)** Categorize the errors using similar categories as you used in Section 1.2 above, potentially adding categories as needed.
- **(c)** Calculate the largest differences in error counts between the LSTM and the SEQ2SEQ model. On what types of words is the SEQ2SEQ model doing better?

2.3. Sentence Embeddings and Story Generation (15 points)

Perform a qualitative evaluation of your SEQ2SEQ models by computing nearest neighbors and generating stories:

- **(a)** Select 10 sentences randomly from the LMDEV file. Let's call these the "query" sentences. Encode each query sentence using the bidirectional LSTM encoder, then find the 10 nearest neighbors

of each from among the sentences in the first field of SEQTRAIN. Use cosine similarity as the similarity metric when finding nearest neighbors. Repeat the process for the word averaging encoder, using the same set of query sentences for both encoders so that you can compare them more easily.

- **(b)** What kinds of similarity are being captured by each encoder? How do the encoders differ in terms of the nearest neighbors?
- **(c)** Run your forward LSTM or bidirectional LSTM SEQ2SEQ model in decoding mode to generate the next sentence in the story. Do this for 10 "previous" sentences in SEQDEV (i.e., the sentences from the first field in the file). Note: to get interesting next-sentences, you will need to train for many epochs. Include your best generated stories in your write-up and make observations about the kinds of things that are being captured in the generated stories.

2.4. Extra Credit (up to 10 extra points)

While doing this assignment, you have probably thought of new ideas or modifications that might be interesting to experiment with. If you would like extra credit, implement and experiment with one such new idea. If you're at a loss, here are a few ideas:

- change the encoder to be a convolutional network
- use character-level information in the encoder, e.g., use an RNN over characters (or character n-grams) instead of words; you could also use a convolutional network that operates on the character sequence
- add an attention mechanism to the decoder
- experiment with the use of scheduled sampling and evaluate qualitatively its impact on story generation
- experiment with using hierarchical softmax to speed up training

References

Mostafazadeh, N., Chambers, N., He, X., Parikh, D., Batra, D., Vanderwende, L., Kohli, P., and Allen, J. (2016). A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California. Association for Computational Linguistics. [1]