UMass CS690N: Advanced Natural Language Processing, Spring 2018

Assignment 2

Due: Feb 28

1 Log-linear models

We've talked about *conditional* log-linear models, $p(y \mid x) \propto \exp(\theta^\mathsf{T} f(x, y)$ where x is an input, y is an output, θ is the parameter vector, and $f(x, y)$ is the feature vector function. By contrast, a plain log-linear model $p(x)$ doesn't condition on any input; it takes the form

$$p(x) = \frac{\exp(\theta^\mathsf{T} f(x))}{\sum_{x' \in \mathcal{X}} \exp(\theta^\mathsf{T} f(x'))}$$

In lecture, we defined a conditional log-linear model for next-word (NW) modeling given a history:

$$p_{\text{NW}}(w_t = w \mid w_{1:t-1}) = \frac{\exp(\theta^\mathsf{T} f(w_{1:t-1}, w))}{\sum_{w' \in \mathcal{V}} \exp(\theta^\mathsf{T} f(w_{1:t-1}, w'))}$$

(Where $w_{1:t-1}$ simply means the sequence $(w_1, w_2, \ldots w_{t-1})$.)

Show that a conditional log-linear next-word model implies that the joint distribution of an entire text (that is, a sequence $w_1..w_n$) is itself log-linear.

Note: When proving this, we found it useful to add notation defining the next-word denominator $Z(w_{1:t-1}) = \sum_{w' \in \mathcal{V}} \exp(\theta^\mathsf{T} f(w_{1:t-1}, w'))$.

2 Markov linear embedding LM

In this question, you'll implement a much-simplified version of Bengio et al. (2003)'s Markovian word embedding-based model. As you recall, their model has both a direct linear layer and an MLP; to keep things simple, here we'll only consider the log-bilinear component, for an h-th order Markov window:

$$p(w_t = w \mid w_{t-h:t-1}) = \frac{\exp([Wx]_w)}{\sum_{w'} \exp([Wx]_{w'})}$$

where x is the concatenation of the h context words' embeddings, and $W \in \mathbb{R}^{V \times hK}$ are the weights connecting from all h context words' dimensions to the next word (where K is the number of word embedding dimensions), and thus Wx is the vector of unnormalized log-probabilities across the vocabulary for the next word.

Data: use the first 2000 sentences of the Brown corpus, lowercased, for training. This is ridiculously small, but large enough such that you'll be able to see the model improving during training. Using all seen words in that subset as the model vocabulary, we get $V = 7390$ (including START, END, and OOV symbols). Also use 100 sentences from elsewhere in the corpus as a validation set (this is the only reason OOV is needed).

Implementation: Implement this model using PyTorch or one of the other autodiff-based neural network frameworks like Tensorflow or DyNet. We will only be able to provide help for PyTorch. Look online for various PyTorch tutorials and examples to see how the library works – this model is pretty similar to a simple classifier.[1] We suggest using PyTorch's 'module' system, specifically with Embedding and Linear modules, with the CrossEntropyLoss loss function (which internally knows how to do a numerically stable, autodifferentiable log-softmax operation). We found that optim.Adam worked fine for learning (with learning rate and weight decay both 1e-3), though since the model is fairly simple, it hopefully won't matter much.

There are various ways to structure the data, but one reasonable method is to be sentence-centric: every sentence is a mini-batch, meaning that your training function loops through one sentence at a time, and for each, evaluates the loss and gradient, and takes an optimizer step after processing the sentence.
You'll have to design an appropriate representation for a sentence. If it's length N, one pytorch-friendly way to structure it is with a an 'output' vector, length N, with integer values encoding the words at each position, along with a 'history' or 'context' matrix, shape $N \times h$, again with integer values, where each row t is the sequence of words from positions $t - h$ through $t - 1$.
The history size and word embedding dimension should be controllable parameters; use history size 2 and 10-dimensional embeddings to start. Please implement at least:

- A training function, which initializes a new model, an optimizer, then for a fixed number of epochs, iterates through sentences, calculating the loss and stepping the optimizer along the gradients. (Eventually, you'll want to add in validation and sampling diagnostics as described next.)
- A sampling function that takes a model, and samples a sentence from it.
- A validation loss calculation function, which returns the cross-entropy (that is, negative average per-token log-likelihood) on the validation set.

Q 2.1: Show output from the training process for 10 epochs. At the end of each epoch, print:

- The training loss (as cross-entropy, or perplexity as you like). This should be going down.
- The validation set loss. This should be going down at least at first. (If it hasn't started going up yet, that indicates you're not yet overfitting.)
- Three sentence samples from the model.

Samples in later iterations should be better than earlier ones — do you find this here?

Q 2.2: Experiment with model hyperparameters—either embedding size or history length, say. Is it possible to improve prediction losses on the training and/or validation sets?

[1] For example, https://github.com/jcjohnson/pytorch-examples/tree/master/nn or https://gist.github.com/xmfbit/b27cdbff68870418bdb8cefa86a2d558

Q 2.3: Now, we'll investigate whether a particular hyperparameter setting gives rise to qualitatively different generative results. Take at least 10 sentence samples each from two different models with different hyperparameter settings (say, a short-history versus a long-history model). Manually assess the grammaticality of each sentence and give a binary judgment for each, as best you can. Report the results and comment on what you found.

Q 2.4: Now conduct a forced-choice evaluation. Pair each sample from one model with a sample from the other. (For example, if there are 10 sentences per model, this creates 10 pairs of sentences.) Now for each sentence pair, judge which one sounds more natural. What's the result?

Q 2.5: What is a possible advantage of the forced-choice evaluation, compared to independently assessing sentences, in trying to determine which model better captures grammaticality? What's a possible disadvantage?

Q 2.6: Conduct a significance test for the unpaired case: compare the grammaticality rates of the two different models with an appropriate two-sample test, like a t-test or chi-square test (e.g. in R, 'prop.test'; though the sample size is a little too small for those particular ones, it's OK for this problem). You only have 10 samples from each, so given the smallness of the sample sizes, you'd like to know if you can conclude anything real about the population-level quantities. Report the test's result and explain what it means; include 95% confidence interval for the difference in proportions and the p-value for whether the difference is compatible with the null hypothesis that the proportions are the same.

Q 2.7: Conduct a significance test for the paired case. The goal is to infer a theoretical population quantity: if you randomly match up a sentence from Model 1 against a sentence from Model 2, what fraction of the time is Model 1's sentence more natural or grammatical? You have just 10 samples of that matchup, and given the smallness of that sample size, you'd like to know if you can conclude anything real about that population-level quantity. Report the 95% confidence interval for that proportion, and the p-value whether that proportion is compatible with the null hypothesis the proportion is 50%. For example, you could use R's implementation of the one-sample exact binomial test ('binom.test'). Report the result and explain what it means.

3 Number prediction LSTM model

Introduction In this question, you will replicate the *number prediction task* from Linzen et al. (2016) "Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies."

Data Download data from http://tallinzen.net/media/rnn_agreement/rnn_agr_simple.tar.gz. After unzipping the file you should see a train, validation, and seven test files. Make sure to read the *README.txt* file. In each file, the first column is the singular/plural label and the second column is the text used for prediction. The text is already lowercased and tokens are split by whitespace. The size of the vocabulary is capped at 10,000 and infrequent words are replaced with their part of speech (i.e. NN or JJ). For training, take a random sample of 10% of the training data (this will decrease accuracy but also allow for reasonable training time on CPUs).

Q 3.1: Objective function Read the model description from Linzen et al. in Sec 3. They say “the state of the LSTM at the end of the sequence is fed into a logistic regression classifier." From this description, what is the training objective function? Describe mathematically how this objective function relates to logistic regression.

Q 3.2: Early stopping You will follow Linzen et al. and do “early stopping based on validation set error." What does early stopping mean here? How does this relate to the concept of overfitting?

Q 3.3: Model Implement the LSTM *number prediction task* model. Use the hyperparameters from the paper: 50-dimensional word embeddings, 50 hidden units, Adam optimizer and early stopping. Use a mini-batch size of 20. What are specifics you will have to implment when doing mini-batching with the examples you are given?

Q 3.4: Training on small subset Subsample to 100 training and 100 dev examples and run your model on 50 epochs. You should be able to drive the loss very close to 0 on this small dataset. Why is this the case? Provide a graph of epoch vs. training loss and epoch vs. dev accuracy for this small dataset.

Q 3.5: Hyperparameter tuning Now train on your 10% sampled training data. You will need to tune the learning rate of your optimizer. Provide of graph of learning rate vs. dev set accuracy. How should you decide how many epochs to run the model? Report the best dev set accuracy, the epoch that results in the best dev set accuracy, and the training loss for that epoch.

Q 3.6: Count of attractors graph Test on the files *numpred.test.k* for $k = 0, 1, \cdots 5$. For $k \leq 3$ sample 2000 of the examples to test (just to be faster) and for $k = 4, 5$ use all the examples. Provide a graph similar to figure $2c$ in the paper: count of attractors vs. error rate. What trend do you see and what does the trend mean?

Q 3.7: Error analysis Manually analyze at least ten examples that your model predicted in correctly and report them here. What patterns do you see? In the beginning of Sec 3, Linzen et al. introduce the ideas of “syntactic number" and “syntactic subjecthood." What do these concepts mean and how to they relate to the errors that you are seeing?

Q 3.8: Word embedding visualization; word lists Assemble lists of singular and plural nouns that are in the model vocabulary. Make at least 5 pairs (10 words) manually. What are they?
Also augment the lists with a few hundred words that can be identified as singular or plural nouns, according to the part-of-speech tags in the Brown corpus. What tags correspond to this (you’ll need to check a reference that defines the Brown tagset)? You may have to impose frequency filtering rules to identify these words; what did you do?

Q 3.9: Visualization Get the matrix of embeddings for these words. Use PCA to reduce them to two dimensions, and plot with color (or some visual element) indicating each word’s syntactic number. Do you get similar or different results than Linzen’s Figure 2f?