LIGN 167: Problem Set 5

December 3, 2018

Collaboration policy: You may collaborate with up to two other students on this problem set. You must write up your own answers to the problems; do not just copy and paste from your collaborators. You must also submit your work individually. If you do not submit a copy of the problem set under your own name, you will not get credit. When you submit your work, you must indicate who you worked with, and what each of your individual contributions were.

Additional dependencies: After you open your LIGN167 environment in the command line, you should type the following command: python -m spacy download en
This will download some dependencies that are necessary for running the code in this problem set.

Submitting your work: In this problem set you won't be submitting code. Instead, you're going to be writing up answers which include some math. When submitting your problem set, you can either write it up in LaTex/Word, or hand-write and take a photo/scan. In either case, your submissions must be done electronically through Gradescope.

This problem set will be different from the previous ones that we have had in this course. We will be providing you with all of the code. Your job will be to explain what the different parts of the code are doing.

All of the code is provided in elman_network.py. This file imports text from sample_corpus.txt, which is a small sample of text from Simple Wikipedia. It then trains a language model using this corpus.

The code provides an implementation of a simple RNN (an Elman network) from scratch in PyTorch. You would never write an RNN this way in practice (you would instead call built-in functions), but this shows how those build-in functions work.

The problem set should not require much knowledge of PyTorch, but the PyTorch tutorial from the syllabus may be helpful: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Problem 1. The function *load_corpus* loads the text from sample_corpus.txt, and returns it as a string.

The function *segment_and_tokenize* takes a single argument, *corpus*, which is assumed to be a string containing the entire corpus. Test this code on some simple examples, or on the corpus that is loaded by *load_corpus*. What does the function do to the string that it receives?

Problem 2. The function *get_data* first loads the corpus, then creates a variable *sents* by calling *segment_and_tokenize*. It then passes *sents* to a function *make_word_to_ix*. The variable *sents* is assumed to be a list of sentences, where each sentence is itself a list of words.

Explain what *make_word_to_ix* is doing.

Hint: Try this on some simple examples. For example, set *sents* equal to: [['The','dog','barked'],['The','cat','barked']].

Problem 3. After the function *get_data* calls *make_word_to_ix*, it then calls a function *vectorize_sents* with arguments *sents* and *word_to_ix*.

vectorize_sents turns the sentences into one-hot vectors. Explain how it does this.

Problem 4. The RNN defined in this code is being used for *language modeling.* As discussed in class, a language model is a probability distribution over sentences. If a sentence s consists of a sequence of words w_0,...,w_{n-1}, then the probability of the sentence is defined by:

$$\begin{aligned} P(s) &= P(<start>, w_0, ..., w_{n-1}, <end>) \\ &= P(w_0 | <start>) \cdot P(w_1 | <start>, w_0) \cdot ... \cdot P(<end> \mid <start>, w_0, ..., w_{n-1}) \end{aligned}$$

We will walk through the steps that the RNN is using to define a language model. The main function in this part of the code is *network_forward.* This function takes two arguments: *sent* and *param_dict.* The argument *sent* is a list of one-hot vectors, each representing a single word in the sentence. The argument *param_dict* is a dictionary containing all of the parameters needed to define the RNN. These are the parameters that will be learned later on.

One of the first things that this function does is call *embed_word*, given a word (*current_word*) from the sentence and the weight matrix W_e. The function *embed_word* returns a word embedding for the word. How does *embed_word* generate a word embedding for *current_word*?

Problem 5. After generating the word embedding for *current_word*, the function *network_forward* then calls the function *elman_unit.* This function will apply an Elman unit (as discussed in class) to two inputs: *current_word_embedding* and *h_previous* (the hidden state at the previous time point). The function *elman_unit* takes five arguments in total: *current_word_embedding*, *h_previous*, and three weight matrices.

Describe mathematically what function *elman_unit* computes on its inputs.

Hint: *torch.matmul* performs matrix multiplication: it multiplies a matrix by a vector. There is very good documentation for *torch.matmul*, and all other PyTorch functions, that can be found through Google.

Problem 6. After calling *elman_unit* and generating the value *h_current*, the function *network_forward* then calls the function *single_layer_perceptron.* The function *single_layer_perceptron* takes two arguments: *h_current* and the weight matrix W_p. Describe mathematically what the function *single_layer_perceptron* computes on its inputs.

Problem 7. The previous three problems have gone through the code describing the RNN. The function *train* is the code that is used for training the RNN. It first defines the dimensions for the different weight vectors, and then initializes the parameters. It uses PyTorch code for computing gradients automatically and optimizing the model weights.

When you run the code by calling *train*, what happens to the loss function over time? Name one problem that the current implementation of the RNN is likely to have. We discussed several in class.

Hint: What can go wrong with Elman networks?