LIGN 167: Problem Set 4

November 16, 2023

Collaboration policy: You may collaborate with up to three other students on this problem set (that is, total group size of 4). You should submit **one response** per group, using group submissions on Gradescope. When you submit your work, you must indicate who was in the group, and what each of your individual contributions were.

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.

In this problem set, 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 Recurrent Neural Network (an Elman network) from scratch in PyTorch. RNNs are the major alternative to transformers when processing text. There are many types of RNNs; Elman networks are the simplest. You would never write an RNN this way in practice (you would instead call built-in functions), but this shows how those built-in functions work.

IMPORTANT: For each problem, you are expected to do 6 things:

1. Provide the correct answer to the problem.
2. Provide the solution that GPT-4 gives. This will be GPT-4 Solution 1.
3. After each solution that GPT-4 gives, respond by saying, "I think you made a mistake. I want you to find it and fix it." Provide the new solution (if any) that GPT-4 generates. This will be GPT-4 Solution 2.
4. If there are errors in Solution 1 from GPT-4, explain where they occur.
5. If there are errors in Solution 2 from GPT-4, explain where they occur.
6. For each problem, include links to your conversations with GPT-4.

Please submit this work in **three** files. One file should be named **ps4**. This should contain the correct answer to the problem. The second file should be named **gpt_solution1**. This should contain Solution 1 generated by GPT-4, as well as your explanations of the mistakes that occurred. The third file should be **gpt_solution2**. This should contain Solution 2 generated by GPT-4, as well as your explanations of the mistakes that occurred.

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:

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

We will walk through the steps that the RNN is using to define a language model. Most of the logic in this part of the code is contained in the $forward$ function of the class $ElmanNetwork$. This function takes one argument: *sent*. The argument *sent* is a list of one-hot vectors, each representing a single word in the sentence.

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 $forward$ then calls the function *elman_unit*. This function will apply an Elman unit to two inputs: *current_word_embedding* and *h_previous* (the hidden state at the previous time point). The function *elman_unit* takes two arguments: *current_word_embedding* and *h_previous*.

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 one argument: *h_current*. Describe mathematically what the function *single_layer_perceptron* computes on its inputs.

Problem 7. You now have described all of the individual steps of the Elman network. A single step in the $forward$ function's for-loop processes a single word. By running the full for-loop, it processes an entire sentence.

In class, we have emphasized the importance of maintaining causality in language modeling (preventing future words from having an influence on previous words). How does the Elman network enforce causality?

Problem 8. Describe how the first word in a sentence influences the representation of the second word in the sentence. Describe how the first word influences the representation of the third word.

Problem 9. The previous 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 over time? What does this imply about the probability distribution that is learned by the model?

Problem 10. Try to identify some hyperparameters that result in good model performance on the training set. Report the results of your experiments, and any patterns that you find.

Problem 11. Imagine an Elman network processing a long piece of text (1000s of words long). Suppose that the correct interpretation of the 950th word in the text depends on the 3rd word in the text. Will the Elman network be able to model this dependency? What kinds of problems might arise here?