CSCI 1051 Homework 2

January 20, 2023

Submission Instructions

Please upload your solutions by **5pm Friday January 20, 2023.** Remember you have 24 hours no-questions-asked *combined* lateness across all assignments.

- You are encouraged to discuss ideas and work with your classmates. However, you **must acknowledge** your collaborators at the top of each solution on which you collaborated with others and you **must write** your solutions independently.
- Your solutions to theory questions must be typeset in LaTeX or markdown. I strongly recommend uploading the source LaTeX (found here) to Overleaf for editing.
- Your solutions to coding questions must be written in a Jupyter notebook. I strongly suggest working with colab as we do in the demos.
- You should submit your solutions as a **single PDF** via the assignment on Canvas.

Problem 1 (from January 17)

Part 1

Consider the small recurrent neural network drawn in Figure 1. Suppose we apply the network to a sequence of **even** length T. What values could the final output y_T take? And when does it take each value?

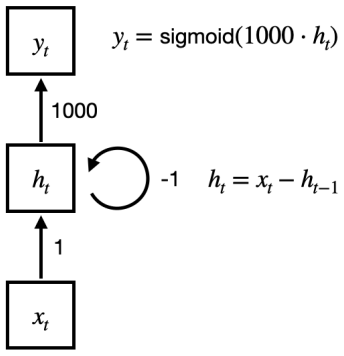

Figure 1: A small recurrent neural network.

Hint: It may help to enroll the network through time on a test input of your choice.

Part 2

Consider the demo on recurrent neural networks. We used a fancy model called an LSTM that we didn't actually implement. In this problem, your job is to replace the LSTM architecture with the simple recurrent neural network from class. In particular, put in the following architecture

$$\mathbf{h}^t = \text{sigmoid}(\mathbf{U}\mathbf{x}^t + \mathbf{W}\mathbf{h}^{t-1})$$
$$\hat{\mathbf{y}}^t = \text{softmax}(\mathbf{V}\mathbf{h}^t).$$

You may not use the Pytorch recurrent neural network function. Instead, you should be using the following functions: `nn.Linear`, `F.sigmoid`, and `F.softmax`. In addition, `torch.zeros` and `torch.stack` may be helpful.

Hint: The only code block you should change in the demo is the `RNN` class.

Once you have implemented your own recurrent neural network and run the rest of the code, comment on why you think the performance of your RNN differs from the performance of the LSTM.

Problem 2

Let us assume the basic definition of self-attention (without any weight matrices) where all the queries, keys, and values are the data points themselves. That is, $\mathbf{x}_i = \mathbf{q}_i = \mathbf{k}_i = \mathbf{v}_i$.

Consider four orthogonal vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$. Suppose the ℓ_2-norm of each vector is β, a very large number.

We will define the following tokens

$$\mathbf{x}_1 = \mathbf{b} + \mathbf{d} \qquad \mathbf{x}_2 = \mathbf{a} \qquad \mathbf{x}_3 = \mathbf{c} + \mathbf{b}.$$

1. What are the norms of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$?
2. Compute $(\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3) = \text{self-attention}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$. Identify which tokens (or combinations of tokens) are approximated by the outputs $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$.
3. Using the above example, describe in a couple of sentences how self-attention allows networks to "copy" an input value to the output.

Problem 3

In this problem, you will code the the GloVe model from scratch. Consider a vocabulary with n words. By processing a bunch of text (perhaps all Harry Potter books?), you can build a co-occurrence matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ where $\mathbf{X}_{ij}$ is the number of times the ith word in your vocabulary appears next to the jth word.

The goal of GloVe is to approximate this co-occurrence matrix as as the product of two low-rank matrices. These low-rank matrices, and some additional vectors, are the trainable parameters of the model: $\mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{c}$. It will be helpful to figure out the dimension of each of these.

Then the loss function is

$$\mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{c}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} f(\mathbf{X}_{ij}) \left(\mathbf{u}_i^\top \mathbf{v}_j + \mathbf{b}_i + \mathbf{c}_j - \log(\mathbf{X}_{ij})\right)^2$$

where $f : \mathbb{R} \to [0, 1]$ is a fixed function which maps large numbers to 1 and small numbers to 0, $\mathbf{u}_i$ is the ith column of $\mathbf{U}$, and $\mathbf{v}_j$ is the jth column of $\mathbf{V}$. Note in the original paper, they use $f(z) = (z/100)^{3/4}$ if $z < 100$ and 1 otherwise.

Hint: There will be many 0 entries in $\mathbf{X}$, so you should be clever so you only iterate over the *nonzero* entries.

You can choose to use the Pytorch functionality for the linear layers and the loss. However, I found it easier to instantiate matrices/vectors and directly update their weights.

Once you have a trained GloVe model, you should repeat the embedding exploration we did in the demo for Word2Vec. In particular,

- find the top ten "similar" words to a word of your choice
- and plot the embeddings of some words of your choice.

You are welcome to re-use code from the demo for any part of this assignment.