Assignment 3: RNN & Inference with LLMs

Due date: Thursday, October 23 at 11:59PM EST

Academic Honesty: Please see the course syllabus for information about collaboration in this course. While you may discuss the assignment with other students, **all work you submit must be your own!**

Part 1. Recurrent Neural Network (35pt)

In this problem, you will study the gradients for Recurrent Neural Network (RNN). While we have discussed the architecture in the class, we have not look into the loss term and its gradients carefully. Let's study this in this homework question.

We will first expand the gradient of the loss function with respect to the parameters using the chain rule. Then, we will bound the norm of each individual partial derivative with matrix norm inequalities. The last step will be to collect all of the partial derivative terms and show how repeated multiplication of a single weight matrix can lead to vanishing or exploding gradients.

RNN Derivatives

Let $S = (s_1, \cdots, s_T)$ be a sequence of input word tokens and T be the sequence length. For a particular token $s_t \in \mathcal{V}$ for $1 \leq t \leq T$, we can obtain its corresponding word embedding $x_t \in \mathbb{R}^d$ by applying equation (1), where $\phi_{\text{one-hot}}$ is the one-hot encoding function and W_e is the word embedding matrix.

The RNN forward pass computes the hidden state $h_t \in \mathbb{R}^{d'}$ using equation (2). Here $W_{\text{hh}} \in \mathbb{R}^{d' \times d'}$ is the recurrent weight matrix, $W_{\text{ih}} \in \mathbb{R}^{d' \times d}$ is the input-to-hidden weight matrix, $b_h \in \mathbb{R}^{d'}$ is the hidden states bias vector, and $\sigma : \mathbb{R}^{d'} \to [-1, 1]^{d'}$ is the tanh activation function. $W_{\text{hh}}, W_{\text{ih}}, b_h$ are shared across sequence index t. The output of RNN $o_t \in \mathbb{R}^k$ at each sequence index t is given by equation (3), where $W_{h_o} \in \mathbb{R}^{k \times d'}$ is the hidden-to-output weight matrix and $b_o \in \mathbb{R}^k$ is the output bias vector. For an input sequence $S = (s_1, \cdots, s_T)$, we have a corresponding sequence of RNN hidden states $H = (h_1, \cdots, h_T)$ and outputs $O = (o_1, \cdots, o_T)$.

$$x_t = W_e \phi_{\text{one-hot}}(s_t) \tag{1}$$
$$h_t = \sigma(W_{\text{hh}} h_{t-1} + W_{\text{ih}} x_t + b_h) \tag{2}$$
$$o_t = W_{h_o} h_t + b_o \tag{3}$$

Let's now use this RNN model for classification. In particular, we consider the last output o_T to be the logits (scores for each class), which we then convert to the class probability vector $p_T \in [0, 1]^k$ by $p_T = g(W_{h_o} h_T + b_o)$ where $g(\cdot)$ is the softmax function and $\|p_T\|_1 = 1$.

1. (4 point, written) Write down the per-example cross-entropy loss $\ell(y, p_T)$ for the classification task. Here $y \in \{0, 1\}^k$ is a one-hot vector of the label and p_T is the class probability vector where $p_T[i] = p(y[i] = 1 \mid S)$ for $i = 1, \ldots, k$. ($[i]$ denotes the i-th entry of the corresponding vector.)

2. To perform backpropagation, we need to compute the derivative of the loss with respect to each parameter. Without loss of generality, let's consider the derivative with respect to a single parameter $w = W_{\text{hh}}[i, j]$ where $[i, j]$ denotes the (i, j)-th entry of the matrix. By chain rule, we have

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial w} \tag{4}$$

   Note that the first two derivatives in the 4 are easy to compute, so let's focus on the last term $\frac{\partial h_t}{\partial w}$.

$$\frac{\partial h_t}{\partial w} = \sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i} \frac{\partial h_i^+}{\partial w} \tag{5}$$

   Here $\frac{\partial h_i^+}{\partial w}$ denotes the "immediate" gradient where h_{i-1} is taken as a constant.

   (a) (4 point, written) Give an expression for $\frac{\partial h_i^+}{\partial w}$.

   (b) (6 points, written) Expand the gradient vector $\frac{\partial h_t}{\partial h_i}$ using the chain rule as a product of partial derivatives of one hidden state with respect to the previous hidden state. You do not need to do differentiations beyond that.

3. (6 points, written) Now let's further expand one of the partial derivatives from the previous question. Write down the Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ by rules of differentiations. You can directly use σ' as the derivative of the activation function in the expression.

Bounding Gradient Norm

To determine if the gradient will vanish or explode, we need a notion of magnitude. For the Jacobian matrix, we can use the induced matrix norm (or operator norm). For this question, we use the spectral norm $\|A\|_2 = \sqrt{\lambda_{\max}(A^\top A)} = s_{\max}(A)$ for a matrix $A \in \mathbb{R}^{m\times n}$. Here $\lambda_{\max}(A^\top A)$ denotes the maximum eigenvalue of the matrix $A^\top A$ and $s_{\max}(A)$ denotes the maximum singular value of the matrix A. You can learn more about matrix norms at this Wikipedia entry.

Now, to determine if the gradient $\frac{\partial \ell}{\partial w}$ will vanish or explode, we can focus on $\|\frac{\partial h_t}{\partial h_i}\|$. Note that if $\|\frac{\partial h_t}{\partial h_i}\|$ vanishes or explodes, $\|\frac{\partial \ell}{\partial w}\|$ also vanishes or explodes based on (4) and (5).

1. (5 points, written) Given the mathematical form of the Jacobian matrix $\frac{\partial h_{i+1}}{\partial h_i}$ we derived earlier, we can now bound the norm of the Jacobian with the following matrix norm inequality

$$\|AB\|_2 \leq \|A\|_2 \cdot \|B\|_2 \tag{6}$$

for matrices A, B with matched shapes. Write down a bound for $\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\|_2$.

2. (10 points, written) Now we have all the pieces we need. Derive a bound on the gradient norm $\|\frac{\partial h_t}{\partial h_i}\|$. Explain how the magnitude of the maximum singular value of W_{hh} can lead to either vanishing or exploding gradient problems. You can use the fact that for the $\tanh$ activation function $\sigma(\cdot)$, the derivative $\sigma'(\cdot)$ is always less than or equal to 1.

Part 2. Prompt Engineering for Addition (65pt)

The goal of this coding problem is the following:

1. Give you hands-on experience with prompt engineering.
2. Understand the challenges involved in prompt engineering.

Specifically, we will use models from Hugging Face to teach the Llama-2-7B and Qwen3-8B model to add two positive 7-digit integers. First go through the file **README.md** and use the **requirements.txt** file to set up the environment required for the class. We recommend using lightning.ai for this assignment (You get free 15 credits!)

Q1. Zero-shot Addition (10pt)

We provided you with a notebook `addition_prompting.ipynb`. In the first section, there are two examples of zero-shot addition: ten instances of two 1-digit addition and ten instances of two 7-digit additions.

a. (2 points, written) Run the two examples. In your opinion, what are some factors that cause language model performance to deteriorate from 1 digit to 7 digits?

b. (3 points, written) Play around with the config parameters:
   - What does each parameter represent? The parameters to be probed are: Temperature, Max Tokens, Top P, Top K, and Repetition Penalty.
   - How does increasing each parameter change the generation?

c. (2 points, written) Do 7-digit addition with Qwen3-8B model.
   - How does the performance change?
   - What are some factors that cause this change?

d. (3 points, written) Previously we gave our language model the prior that the sum of two 7-digit numbers must have a maximum of 8 digits (by setting `max_token=8`). What if we remove this prior by increasing the `max_token` to 20?
   - Does the model still perform well?
   - What are some reasons why?

Q2. In Context Learning (25pt)

In this part, we will try to improve the performance of 7-digit addition via in-context learning. We will only use LLaMa-2-7B from here on.

a. (3 points) Using the baseline prompt ("Question: What is 3+7? Answer: 10 Question: What is a+b? Answer:"), check 7-digit addition for 10 pairs again.

   - Compared to zero-shot 7-digit additions with maximum 8 tokens, how does the performance change when we use the baseline in-context learning prompt?
   - What are some factors that cause this change?

b. (7 points) Now we will remove the prior on output length and re-evaluate the performance of our baseline one-shot learning prompt. We need to modify our post processing function to extract the answer from the output sequence.

   - Describe an approach to modify the post processing function.
   - Compared to 2a, How does the performance change when we relax the output length constraint?
   - What are some factors that cause this change?

c. (9 points) Let's change our one-shot learning example to something more "in-distribution". Previously we were using 1-digit addition as an example.
   Let's change it to 7-digit addition (1234567+1234567=2469134).

   - Evaluate the performance with max tokens = 8. Report the res, acc, mae, prompt length.
   - Evaluate the performance with max tokens = 50. Report the res, acc, mae, prompt length.
   - How does the performance change from 1-digit example to 7-digit example?
   - Take a closer look at test range. How was res calculated? What is its range? Does it make sense to you? Why or why not?

d. (6 points) Let's look at a specific example with large absolute error.

   - Run the cell at least 5 times. Does the error change with each time? Why?
   - Can you think of a prompt to reduce the error?
   - Why do you think it would work?
   - Does it work in practice? Why or why not?

Q3. Prompt-a-thon! (30pts)

In this part, you will teach Llama-2-7b model to add 2 7-digit numbers reliably! Note: You can utilize a `.txt` file to test a specific prompt, but make sure the final prompt is in **submission.py** to pass all test cases for full credit. **You will need to request access for downloading model weights from Hugging Face, so we highly recommend starting early.**

We will provide you with a `run_tests.py` file to test your submission. It will evaluate your prompt on 30 instances of 7-digit integer addition. Some instances are manually designed, and others are randomly generated using a random seed that is different from `addition_prompting.ipynb`.

We will run `submission.py` on our end with an autograder to verify your results. You must pass all test cases for full credit. Please keep `max_tokens` at least 50, so that you're not giving the model a prior on the output length. **Do not forget to include `your_hf_token` and `your_net_id` in submission.py; failure to do so will not allow your submission to be autograded.**

Also, do not try to hack the pre-processing and post-processing functions by including arithmetic operations.

The autograder can detect the use of arithmetic operations in these functions.

Here's a list of functions you can modify:

- `your_hf_token`
- `your_net_id`
- `your_prompt`
- `your_config`
- `your_pre_processing` (but no arithmetic operation. e.g., `f"a+b={a+b}"` is not allowed.)
- `your_post_processing` (but no arithmetic operation. e.g., `f"a+b={a+b}"` is not allowed.)

a. (coding, 30 points) Use prompt to improve test performance. For full credit, either
   - get average accuracy greater than 0.2, or
   - get average mean absolute error less than 1e5.

   For testing, we recommend using `python run_tests.py`.
   Please submit the files: submission.py and addition_prompting.ipynb

This homework was modified from previous offering of this course (offered by He He).