October 7, 2022

UNIVERSITY OF ARKANSAS

CSCE 5563: Introduction to Deep Learning

Homework Assignment #2

Submission Deadline: 11:59PM, 10/24/2022

What to submit:

- **1 pdf file** contains your solution and your name. Please don't submit the screenshot/scan of your handwritten solution. Exception is made for hand-drawn graphs. Grading rule: If it's unreadable, I won't grade. Confusing answer gets no point.
- **source code folder** that contains your code. The source code contains readme file on how to run the code.
- Put all the to-be-submitted files into a folder with the name format as:

  {*LASTNAME*}_{*FIRSTNAME*}_homework1

What else to note:

A.. If you don't understand the question, ask the TA immediately at khoavoho@uark.edu or visit AICV Lab (JBHT #447) on Tuesday 1:45PM-2:45PM.

B.. You can discuss with your classmates, DO NOT COPY and please submit your own work.

C.. Solution for this homework is provided within 1-2 weeks after submission deadline or extended deadline, whichever comes later.

Question 1. (10 points): RNNs vs CNNs

(5 points) What is the difference between CNNs and RNNs ?

(5 points) What tasks that CNNs can do but RNNs ? Please explain with examples.

Question 2. (30 points): Analysis of RNNs

(5 points) What is the limitations of RNNs ?

(5 points) If we have a GPU to accelerate speed by parallel computations, can it parallelize the computations of RNN at timestep-level during training phase? If so, does it also hold true at the inference phase of RNN? Please justify your answer.

(5 points) Explain the problems of vanishing gradient and exploding gradient, in which cases will any of those problems happen ?

(5 points) In the figure below, we have 4 schemes of RNNs, please explain what types are they, and list one or more problems that each scheme of those can solve. Please justify why you think a RNN scheme can solve the problem you listed.

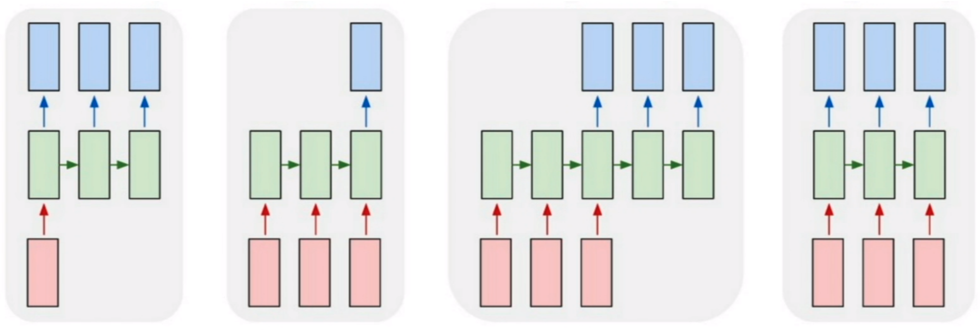

Figure 1: Typical schemes of RNNs.

(10 points) Compute the back-propagation through time of an LSTM cell. At a time-step t, an LSTM cell takes $x^{(t)}$ as input, hidden state $h^{(t-1)}$, cell state $c^{(t-1)}$, it progresses through the following equations:

$$
\begin{aligned}
f^{(t)} &= \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)}) && \longrightarrow \text{Forget gate} \\
i^{(t)} &= \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)}) && \longrightarrow \text{Input gate} \\
o^{(t)} &= \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)}) && \longrightarrow \text{Output gate} \\
g^{(t)} &= \tanh(W_{gx}x^{(t)} + W_{gh}h^{(t-1)}) && \\
c^{(t)} &= f^{(t)}c^{(t-1)} + i^{(t)}g^{(t)} && \longrightarrow \text{Update cell state} \\
h^{(t)} &= o^{(t)}\tanh(c^{(t)}) && \longrightarrow \text{Final output}
\end{aligned}
$$

Now, considering the gradient that back-propagated from loss function applied at $h^{(t)}$ to any state s as $\frac{dL}{ds}$, which will be re-written as δs for concise. Please derive the back-propagation equation to $\delta h^{(t)}$, $\delta c^{(t)}$, $\delta o^{(t)}$, $\delta g^{(t)}$, $\delta f^{(t)}$, $\delta i^{(t)}$.

Next, derive the back-propagation equation of δW_{ix} (the other weight matrices follow the same equation).

Question 3. (20 points): Types of RNNs and Transformer

(10 points) Distinguish between Seq2Seq, attention, self-attention, transformer.

(10 points) What is multiple head, positional encoding used in transformer.

Question 4. (40 points): Binary Addition

In this question, you will be implementing a recurrent neural network which operates binary addition. The inputs are two arbitrary binary sequences, starting with the **least** significant binary digit. Two sequences are padded by zeros (at least one zero) at the end, two have the same length. At each time step, the recurrent neural network takes in two binary digits from both inputs, and outputs a result digit.

Example:
- Equation: 10110100 + 10111 = 11001011
- Input 1: 0,0,1,0,1,1,0,1,0
- Input 2: 1,1,1,0,1,0,0,0,0
- Output: 1,1,0,1,0,0,1,1,0

Loss function: as the output at each time step is binary (0 or 1), you can apply Binary Cross Entropy to train the model.

What to report:

- A plot of training loss values after every epoch for each part.
- Show 5 examples of deployment procedure. Every example includes two input binary sequences and an output sequence.

(20 points) Part 1:
You are expected to write a complete system that trains the RNN model on this task. The system expects you to include the following modules:

- A pytorch Dataset, that has a __getitem__() function, which, for each call, samples two binary sequences of arbitrary length and at most 16 digits, and the exact solution when we add these two binary numbers. The binary sequences and the solution are pre-processed so that they follow least-to-most-significance order, and is padded with at least one zero so that two input sequences have the same length (as shown in the above example).
- A pytorch Dataloader to process Dataset, and creates mini-batches of samples, every sample as a pair of binary sequences, to train the model. Batch size is set to 128 samples, and an epoch is set to 100 iterations on Dataloader.
- A pytorch RNN model that has two input units, one output units, a hidden layer of 10 perceptrons, and use ReLU activation function.
- A training procedure that trains the model through 100 epochs. During training process, save the best performed model. The performance of the model is represented as the average loss of the model through an epoch.
- A deployment procedure. In this procedure, you will load the best performed model from training procedure. Then, the system takes as input two arbitrary integers, and pre-processes them into binary sequences with the order of least-to-most-significant-digits, and feeds them to the model to obtain the result. Also, you are required to program a checking function to check how many percent of the predicted digits matched with the exact solution.

(20 points) Part 2:
In the above part, we just tried the fourth scheme of RNN as shown in Fig. 1, which is a straightforward way to solve the problem of binary addition, just like how we learned to perform such computation. However, RNNs also have the ability of memorizing inputs.

In this part, we will do an investigation on such ability of RNN, using the third RNN scheme in Fig. 1. To do that, you are required to:

- Modify the Dataset class to return all the input binary sequences and the result binary sequence in the typical most-to-least-significant-digit order.
- The new RNN model will be fed by every pair of digits from the input sequences without producing any output. After processing through all input digits, it starts predicting the one-by-one digit of the output.
- As the RNN model now will need to memorize much more information, it will need a bigger hidden layer, try increasing hidden layer to 100 perceptrons.
- Keeping the training and deployment procedure as in Part 1.

Note: This part is meant to investigate the capability of RNN in memorizing things, therefore, it does not necessarily have to give an exact result.

Important Notice

1. You should use ipython notebook and Google Colab for **Question 4** to accelerate model training by GPU and have a clean, well-organized submission.

2. If you have any question, please feel free to contact TA (Khoa Vo) through email khoavoho@uark.edu, or visit AICV Lab (JBHT #447) on Tuesday 1:45PM-2:45PM.