CS 4644/7643: Deep Learning
Fall 2023
Problem Set 2

Instructor: Zsolt Kira
TAs: Krishanu Agarwal, Manav Agrawal, Aditya Akula, Will Held, Vikranth Keerthipati, Pranay Mathur, Avinash Prabhu, Katie Stevo, Wei Zhou, Bowen Zuo
Discussions: https://piazza.com/gatech/spring2024/cs4644acs7643a

Due: Saturday, March 09, 11:59pm

Instructions

1. We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!
   - Please submit one comprehensive PDF for the **HW3 Writeup** on Gradescope. This PDF should include responses to all theory questions, the report addressing coding problems, and the Jupyter notebooks: containing tests and explanations for code implementations.
   - When submitting your assignment to Gradescope, ensure that each problem or subproblem solution is on a separate page, clearly marked with the corresponding problem or subproblem. For the coding part, only include files collected by the `collect_submission` script to avoid receiving a zero from the auto-grader. Regrade requests will not be accepted for submissions that do not follow this requirement due to the size of the class.
   - To complete the coding task, use the provided `collect_submission.py` script. Submit the generated file to the **HW3 Code** assignment on Gradescope. Although your code won't undergo explicit grading, its submission remains mandatory. Ensure you've saved the latest version of your code before executing the script.
   - Note: This is a large class and Gradescope's assignment segmentation features are essential. Failure to follow these instructions may result in parts of your assignment not being graded. We will not entertain regrading requests for failure to follow instructions. Please read https://stats200.stanford.edu/gradescope_tips.pdf for additional information on submitting to Gradescope.
2. LaTeX'd solutions are strongly encouraged (solution template available on Canvas—Assignment 3 HW), but scanned handwritten copies are acceptable. Hard copies are **not** accepted.
3. We generally encourage you to collaborate with other students. You may talk to a friend, and discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and *not* as a group activity. Please list the students you collaborated with.

1 Collaborators [0.5 points]

Please list your collaborators and assign this list to the corresponding section of the outline on Gradescope. If you don't have any collaborators, please write 'None' and assign it to the corresponding section of the Gradescope submission regardless.

2 Logic and XOR

1. **[1 point]** Implement AND and OR for pairs of binary inputs using a single linear threshold neuron with weights $\mathbf{w} \in \mathbb{R}^2$, bias $b \in \mathbb{R}$, and $\mathbf{x} \in \{0,1\}^2$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{1}$$

That is, find $\mathbf{w}_{\texttt{AND}}$ and $b_{\texttt{AND}}$ such that

x_1	x_2	$f_{\texttt{AND}}(\mathbf{x})$
0	0	0
0	1	0
1	0	0
1	1	1

Also find $\mathbf{w}_{\texttt{OR}}$ and $b_{\texttt{OR}}$ such that

x_1	x_2	$f_{\texttt{OR}}(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	1

2. **[1 point]** Consider the XOR function

x_1	x_2	$f_{\texttt{XOR}}(x)$
0	0	0
0	1	1
1	0	1
1	1	0

Prove that XOR can NOT be represented using a linear model with the same form as (1).

[*Hint:* To see why, plot the examples from above in a plane and think about drawing a linear boundary that separates them.]

3 Recurrent Neural Networks and Transformers

3. **[LSTM for parity function: 5 points]** Let us recall different gates in an LSTM Network. First gate is the "forget gate layer":

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2}$$

where f_t is the output of forget gate, W_f is the weight matrix, h_{t-1} is the hidden state of step *t-1*, x_t is the current input and b_t is the bias value.
Next we have "input gate layer":

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{3}$$

$$\tilde{C}_t = \tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{4}$$

where i_t decides which values we will update and $\tilde{C}_t$ are the new candidate values that could be added to the cell state. Next we have new cell state candidate values:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{5}$$

Finally, we have the output and hidden state

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{6}$$

$$h_t = o_t * \tanh(C_t) \tag{7}$$

Design an LSTM Network for the bit parity problem mentioned in Question 1. Specifically, provide values for W_f, b_f, W_i, b_i, W_C, b_C, W_o and b_o such that the cell state C_t stores the parity of bit string. Please mention any assumptions you make. For this problem, you can assume below for Sigmoid and tanh function:

$$\sigma(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

$$\tanh(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & x = 0 \\ -1, & \text{if } x < 0 \end{cases} \tag{9}$$

Hint: Recall that XOR of x and y can be represented as $(x \wedge \bar{y}) \vee (\bar{x} \wedge y)$ which you implemented above. Think about how you can leverage this information for equation (5).

4. **[When to Stop in Beam Search: 5 points]** Beam Search is a widely used technique for decoding the most likely sequence from sequence models. However, it is difficult to decide when to stop the beam search to obtain optimality because hypotheses can be finished in different steps. In this question, we will develop a formal understanding of the stopping criteria in beam search.

   Let $\mathbf{x}$ denote the input upon which we condition our sequence model. Let $\mathbf{y}$ denote the output sequence. Let $\mathbf{y}_{<t}$ be a shorthand notation for the sub-sequence $(y_0, y_1, \ldots, y_{t-1})$. We say that a sequence (or hypothesis as they are sometimes referred to in this literature) $\mathbf{y}$ is *completed* ($\texttt{comp}(\mathbf{y}) = \textit{true}$), if its last token is </s>, *i.e.*,

$$\texttt{comp}(\mathbf{y}) = \textit{true} \leftrightarrow (\mathbf{y}_{|\mathbf{y}|} = \texttt{</s>})$$

   in which case it will not be further expanded in beam search.

With this notation, we can write down the maximum a-posteriori inference problem as:

$$\mathbf{y}* = \underset{\mathbf{y}}{\text{argmax}} \quad \prod_{t \le |\mathbf{y}|} p(y_t|\mathbf{x}, \mathbf{y}_{<t}) \tag{10}$$

$$s.t. \quad \texttt{comp}(\mathbf{y}) = true \tag{11}$$

We use beam search to find the (approximate) best output $\mathbf{y}^*$ At time t, let B_{t-1} denote the beams so far. Thus, B_{t-1} is a b-length list consisting of $\langle \mathbf{y}, s \rangle$ pairs, *i.e.*, $B_{t-1} = \big(\langle \mathbf{y}^1, s^1 \rangle, \ldots, \langle \mathbf{y}^i, s^i \rangle \big)$, where $\mathbf{y}^i$ is a $(t-1)$-length sequence (a beam) and s^i is its associated score (sum of log-conditional probabilities), *i.e.* $s^i = \sum_{j=1}^{t-1} \log p(y_j^i \mid \mathbf{x}, \mathbf{y}_{<j}^i)$.

Let $\circ$ denote a concatenation operation, *i.e.* $\mathbf{y} \circ y_t$ represents a beam expansion where $\mathbf{y}$ is concatenated with y_t. Beam search can then be formalized via a $\mathbf{top}^b$ operator that selects (quite literally) the top-b scoring items in an expanded list of beams:

$$B_t = \overset{b}{\mathbf{top}} \left\{ \langle \mathbf{y} \circ y_t, \ s + \log p(y_t|\mathbf{x}, \mathbf{y}_{<t}) \rangle \mid \langle \mathbf{y}, s \rangle \in B_{t-1} \right\} \tag{12}$$

Let $\texttt{best}_{\le t}$ be the best-completed hypothesis so far (up to step t), *i.e.*

$$\texttt{best}_{\le t} \quad \overset{\Delta}{=} \quad \max \Big\{ s \mid \langle \mathbf{y}, s \rangle \in \cup_{j \le t} B_t, \texttt{comp}(\mathbf{y}) = true \Big\} \tag{13}$$

Notice that if there is no completed beam till time t, $\texttt{best}_{\le t}$ is undefined/empty.

Now, for the proof.

Assuming that $\texttt{best}_{\le t}$ is defined at time t and the current highest scoring beam in B_t (*i.e.* $\mathbf{y}^1$) scores worse than or equal to $\texttt{best}_{\le t}$, *i.e.* $s^1 \le \texttt{best}_{\le t}$, prove that there is no need to run beam search out further. That is, prove that the current best-completed beam (corresponding to $\texttt{best}_{\le t}$) is the overall highest-probability completed beam and future steps will be no better.

5. **[Exploding Gradients: 5 points]** Learning long-term dependencies in recurrent networks suffers from a particular numerical challenge – gradients propagated over many time steps tend to either 'vanish' (i.e. converge to 0, frequently) or 'explode' (i.e. diverge to infinity; rarely, but with more damage to the optimization). To study this problem in a simple setting, consider the following recurrence relation without any nonlinear activation function or input x:

$$h_t = W^\top h_{t-1} \tag{14}$$

where W is a weight-sharing matrix for recurrent relation at any time t. Let $\lambda_1, ..., \lambda_n$ be the eigenvalues of the weight matrix $W \in \mathbb{C}^{n \times n}$. Its spectral radius $\rho(W)$ is defined as:

$$\rho(W) = \max\{|\lambda_1|, ..., |\lambda_n|\} \tag{15}$$

Assuming the initial hidden state is h_0, write the relation between h_t and h_0 and explain the role of the eigenvalues of W in determining the 'vanishing' or 'exploding' property as $t \gg 0$.

6. **[Transformer as GNN: 5 points; Extra credit for both 4644 and 7643]** Learning representations of inputs is the bedrock of all neural networks.

In recent years, Transformer models have been widely adapted to sequence modeling tasks in the vision and language domains, while Graph Neural Networks (GNNs) have been effective in constructing representations of nodes and edges in graph data. In the following questions, we will explore both Transformers and GNNs and draw some connections between them.

Background:

Let us first take a look at a graph model. We define a directed graph $G = \{V, E\}$ where V is the set of all vertices and E is the set of all edges. For $\forall v_i \in V$, let us define $\mathcal{N}(v_i)$ as the set of all of v_i's neighbors with outgoing edges towards v_i. v_i has a state representation h_i^t at each time step t.

The values of h_i^t are updated in parallel, using the same snapshot of the graph at a given time step. The procedures are as follows: We first need to aggregate the incoming data $H'_{it} = \{f_{ji}(h_j^t) | \forall j, v_j \in \mathcal{N}(v_i)\}$ from neighbors using the function $Agg(H'_{it})$. Note that the incoming data from each neighbor is a transformed version of its representation using function f_{ji}. The aggregation function $Agg(H'_{it})$ can be something like the summation or the mean of elements in H'_{it}.

Say the initial state at time step 0 is h_i^0. Now let us define the update rule for h_i^t at time step $t+1$ as the following:

$$h_i^{t+1} = q(h_i^t, Agg(H'_{it})) \tag{16}$$

where q is a function - $Q_t : \{H_t, Agg(H'_t)\} \to H_t$, where $H_t = \{h_n^t | \forall n, v_n \in V\}$.

Now, let us take a look at Transformer models. Recall that Transformer models build features for each word as a function of the features of all the other words with an attention mechanism over them, while RNNs update features in a sequential fashion.

To represent a Transformer model's attention mechanism, let us define a feature representation h_i for word i in sentence S. We have the standard equation for the attention update at layer l as a function of the each other word j as follows:

$$\begin{aligned} h_i^{l+1} &= \texttt{Attention}(Q^l h_i^l, K^l h_j^l, V^l h_j^l) && (17) \\ &= \sum_{j \in S} \left(\texttt{softmax}_j (Q^l h_i^l \cdot K^l h_j^l) V^l h_j^l \right) && (18) \end{aligned}$$

where Q^l, K^l, V^l are weight matrices for "Query", "Key", and "Value". Q is a matrix that contains vector representations of one word in the sentence, while K is a matrix containing representations for all the words in the sentence. V is another matrix similar to K that has representations for all words in the sentence. As a refresher for your knowledge about the Transformer model, you can refer to this paper.

Based on the above background information, answer the following questions:

(a) If the aggregation operation for $Agg(H'_{it})$ is the summation of representation of all adjacent vertices, rewrite the equation 16 by replacing $Agg(H'_{it}))$ in terms of $\mathcal{N}$, f, and h.

(b) Consider the directed graph G in Fig 1. The values for the vertices at time step t are as follows:

$$h_1^t = [1, -1] \quad h_2^t = [-1, 1] \quad h_3^t = [0, -1] \quad h_4^t = [1, 0] \tag{19}$$

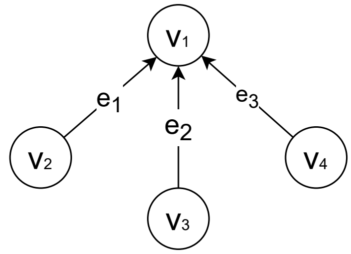

Figure 1

The aggregation function $Agg(H'_{1t})$ is:

$$Agg(H'_{1t}) = [0.6, 0.2, 0.2] \begin{bmatrix} f(h_2^t) \\ f(h_3^t) \\ f(h_4^t) \end{bmatrix} \tag{20}$$

And the function f on all the edges is:

$$f(x) = 2x \tag{21}$$

Now, given that

$$h_1^{t+1} = q(h_1^t, Agg(H'_{1t})) = W(h_1^t)^T + max\{Agg(H'_{1t}), 0\} \tag{22}$$

where $W = [1, 1]$, what is the updated value of h_1^{t+1}?

(c) Consider the graph G in question (b). We want to alter it to represent the sentence "I eat red apples" (4 word tokens) as a fully connected graph. Each vertex represents one word token, and the edges represent the relationships among the tokens. How many edges in all would graph G contain? Notice that the edges are directed and a bi-directional edge counts as two edges.

(d) Using equations 16 and 18, show that the Transformer model's single-head attention mechanism is equivalent to a special case of a GNN.

(e) An ongoing area of research in Transformer models for NLP is the challenge of learning very-long-term dependencies among entities in a sequence. Based on this connection with GNNs, why do you think this could be problematic?

4 Paper Review [Extra credit for 4644, regular credit for 7643]

For this homework's paper review section, we turn to the interesting and increasingly important field of scaling laws.

The following paper **BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension**, which was presented at ACL 2020, introduces a method which unifies prior pretrained language models into a single denoising task with an Encoder-Decoder network. This model enormously improved the flexibility of existing pretrained language models, achieving state-of-the-art results in a diverse range of language tasks. This work also highlights the advantages of the different architectures which you will complete in the coding section of the homework later.

The paper can be viewed here. The evaluation rubric for this section is as follows:

7. **[2 points]** Briefly summarize the key contributions, strengths and weaknesses of this paper.

8. **[2 points]** What is your personal takeaway from this paper? This could be expressed either in terms of relating the approaches adopted in this paper to your traditional understanding of explainability techniques, or potential future directions of research in the area which the authors haven't addressed, or anything else that struck you as being noteworthy.

Guidelines: Please restrict your reviews to at least one paragraph per question, but no more than 350 words (total length for answers to both of the above questions). Please write separate answers for each question to assist in Gradescope grading.

5 Coding:

5.1 Part 1: Transformer Architectures and Language Modeling

[11 points coding + 5 points writeup] The first coding part of this assignment will consist of the implementation of multiple architectures a Transformer models. To get started, download the code zip file from Canvas. Also, be sure to append the completed report (PPT template available in the zip folder) *and* the Jupyter notebook `Transformer_Architectures.ipynb` (as mentioned in the instructions) to the PS3 theory solutions and upload them as one PDF on Gradescope under the HW3 writeup.

5.2 Part 2: Diffusion Model

[14 points coding + 7 points writeup] The second part of coding for this assignment is a Diffusion Model (mostly) from scratch. Get started early on the coding as this is a long assignment. To get started, download the code zip file from Canvas. Also, be sure to append the completed Jupyter notebook `Diffusion_CV.ipynb` (as mentioned in the instructions) to the HW3 theory solutions and upload them as one PDF on Gradescope under HW3 writeup.

Points breakdown:

- HW3 Theory
  - Theory Answers [17.5 points for 4803/ 21.5 points for 7643]
- HW3 Code
  - Part 1 [11 points]
  - Part 2 [14 points]
- HW3 Writeup
  - Report PDF part 1 [5 points]
  - Jupyter Notebook (writeup) as PDF part 2 [7 points]
  - Jupyter Notebook as PDF part 1