Due Date: November 14th, 2022 at 11:00 pm

Instructions

- *For all questions, show your work!*
- *Use LaTeX and the template we provide when writing your answers. You may reuse most of the notation shorthands, equations and/or tables. See the assignment policy on the course website for more details.*
- *Submit your answers electronically via Gradescope.*
- *TAs for this assignment are* ***Arian Khorasani and Nanda Harishankar Krishna****.*

Question 1 (2-5-5-5-3)**.** (**Backpropagation in RNNs**)

Consider the following RNN:

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1})$$
$$\boldsymbol{y}_t = \boldsymbol{V}\boldsymbol{h}_t$$

where σ denotes the logistic sigmoid function. Let $\boldsymbol{z}_t$ be the true target of the prediction $\boldsymbol{y}_t$ and consider the sum of squared loss $L = \sum_t L_t$ where $L_t = ||\boldsymbol{z}_t - \boldsymbol{y}_t||_2^2$.

In this question our goal is to obtain an expression for the gradients $\nabla_{\boldsymbol{W}} L$ and $\nabla_{\boldsymbol{U}} L$.

1. First, complete the following computational graph for this RNN, unrolled for 3 time steps (from $t = 1$ to $t = 3$). Label each node with the corresponding hidden unit and each edge with the corresponding weight.

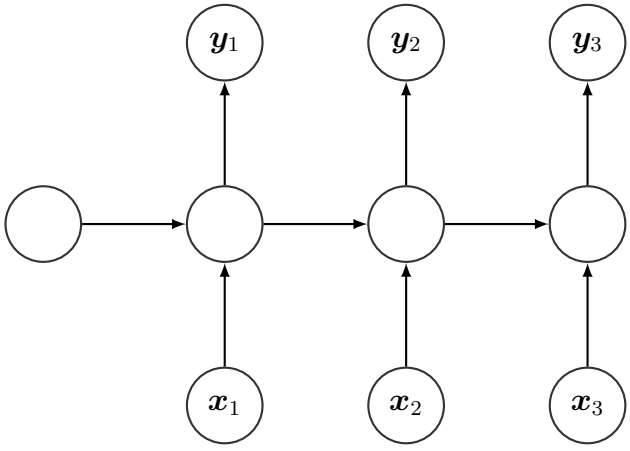

FIGURE 1 – Computational graph of the RNN unrolled for three timesteps.

2. Derive a recursive expression for the total gradient $\nabla_{\boldsymbol{h}_t} L$ in terms of $\nabla_{\boldsymbol{h}_t} L_t$ and $\nabla_{\boldsymbol{h}_{t+1}} L$.

3. Derive an expression for $\nabla_{\boldsymbol{h}_t} L_t$ and $\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$.
4. Now derive $\nabla_{\boldsymbol{W}} L$ and $\nabla_{\boldsymbol{U}} L$ as functions of $\nabla_{\boldsymbol{h}_t} L$, using the results from the previous 2 sub-questions.
   Hint: It might be useful to consider the contribution of the weight matrices when computing the recurrent hidden unit at a particular time t and how those contributions might be aggregated.
5. To avoid the problem of exploding or vanishing gradients, we can use Truncated BPTT to compute gradients for parameter updates in RNNs. What can you say about the gradient estimates from Truncated BPTT – are they unbiased or biased ? Why do you think so ? While Truncated BPTT can help mitigate exploding or vanishing gradients, what problems do you foresee with this approach, for example, in tasks involving language ?

Answer 1.

Question 2 (5-5-3-5-2)**.** (**Transformers as GNNs**)

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:

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

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:

$$h_i^{l+1} = \texttt{Attention}(Q^l h_i^l, K^l h_j^l, V^l h_j^l) \tag{2}$$

$$= \sum_{j \in S} \left(\texttt{softmax}_j(Q^l h_i^l \cdot K^l h_j^l) V^l h_j^l\right) \tag{3}$$

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:

(2.1) If the aggregation operation for $Agg(H'_{it})$ is the summation of representation of all adjacent vertices, rewrite the equation 1 by replacing $Agg(H'_{it})$) in terms of $\mathcal{N}$, f, and h.

(2.2) Consider the directed graph G in Fig 2. The values for the vertices at time step t are as

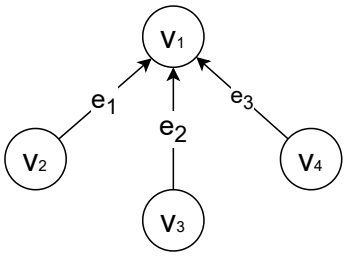

FIGURE 2 – Graph G for (2.2)

follows:

$$h_1^t = [1, -1] \quad h_2^t = [-1, 1] \quad h_3^t = [0, -1] \quad h_4^t = [1, 0] \tag{4}$$

(Note: these are row vectors.)
The aggregation function $Agg(H'_{1t})$ is:

$$Agg(H'_{1t}) = [0.6, 0.2, 0.2] \begin{bmatrix} f(h_2^t) \\ f(h_3^t) \\ f(h_4^t) \end{bmatrix} \tag{5}$$

And the function f on all the edges is:

$$f(x) = 2x \tag{6}$$

Now, given that

$$h_1^{t+1} = q(h_1^t, Agg(H'_{1t})) = W(h_1^t)^T + max\{Agg(H'_{1t}), 0\} \tag{7}$$

where $W = [1, 1]$, what is the updated value of h_1^{t+1} ?
(Note: All h_i^t and W are row vectors.)

(2.3) Consider the graph G in question (2.2). 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.

(2.4) Using equations 1 and 3, show that the Transformer model's single-head attention mechanism is equivalent to a special case of a GNN.

(2.5) 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 ?

Answer 2.

Question 3 (6-2-6-6)**.** (**Optimization and Regularization**)

(3.1) Batch normalization, layer normalization and instance normalization all involve calculating the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma^2}$ with respect to different subsets of the tensor dimensions. Given the following 3D tensor, calculate the corresponding mean and variance tensors for each normalization technique: $\boldsymbol{\mu}_{batch}$, $\boldsymbol{\mu}_{layer}$, $\boldsymbol{\mu}_{instance}$, $\boldsymbol{\sigma}^2_{batch}$, $\boldsymbol{\sigma}^2_{layer}$, and $\boldsymbol{\sigma}^2_{instance}$.

$$\left[\begin{bmatrix}4,3,2\\1,4,3\end{bmatrix},\begin{bmatrix}3,4,1\\4,2,2\end{bmatrix},\begin{bmatrix}3,2,3\\4,1,2\end{bmatrix},\begin{bmatrix}1,1,3\\4,1,2\end{bmatrix}\right]$$

The size of this tensor is 4 x 2 x 3 which corresponds to the batch size, number of channels, and number of features respectively.

(3.2) While BatchNorm is very common in Computer Vision models, it is significantly outperfotmed by LayerNorm in Natural Language Processing tasks. What could be some problems with using BatchNorm in NLP ? You may consider a naïve implementation of BatchNorm and also illustrate your point with an example if you wish.

(3.3) Consider a linear regression problem with input data $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, weights $\boldsymbol{w} \in \mathbb{R}^{d\times 1}$ and targets $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$. Suppose that dropout is applied to the input (with probability $1-p$ of dropping the unit i.e. setting it to 0). Let $\boldsymbol{R} \in \mathbb{R}^{n\times d}$ be the dropout mask such that $\boldsymbol{R}_{ij} \sim \text{Bern}(p)$ is sampled i.i.d. from the Bernoulli distribution.

For a squared error loss function with dropout, we then have:

$$L(\boldsymbol{w}) = ||\boldsymbol{y} - (\boldsymbol{X} \odot \boldsymbol{R})\boldsymbol{w}||^2$$

Let Γ be a diagonal matrix with $\Gamma_{ii} = (\boldsymbol{X}^\top \boldsymbol{X})^{1/2}_{ii}$. Show that the *expectation (over* $\boldsymbol{R}$*)* of the loss function can be rewritten as $\mathbb{E}[L(\boldsymbol{w})] = ||\boldsymbol{y} - p\boldsymbol{X}\boldsymbol{w}||^2 + p(1-p)||\Gamma\boldsymbol{w}||^2$. *Hint: Note we are trying to find the expectation over a squared term and use* $\text{Var}(Z) = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2$.

(3.4) Consider a linear regression problem with a d-dimensional feature vector $\boldsymbol{x} \in \mathbb{R}^d$ as input and $y_i \in \mathbb{R}$ as the output. The dataset consists of n training examples $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)$. Let $\boldsymbol{X}$ be the $n \times d$ data matrix formed by placing the feature vectors on the rows of this matrix, i.e.,

$$\boldsymbol{X}^T = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\right]$$

Let $\boldsymbol{y}$ be a column vector with elements $y_1, y_2 ... y_n$. We will operate in the setting where $d > n$, i.e., there are more feature dimensions than samples. Thus, the following linear system is under-constrained / under-determined:

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y} \tag{8}$$

To avoid the degenerate case, we will assume that $\boldsymbol{y}$ lies in the span of $\boldsymbol{X}$, i.e. this linear system has at least one solution.
Now, recall that the optimization problem in least-squares regression is the following:

$$\min_{\boldsymbol{w} \in \mathbf{R^d}} \quad f(\boldsymbol{w}) = \sum_{i=1}^{n} \underbrace{(y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2}_{\text{squared error on example } i} \tag{9}$$

We will optimize (9) via gradient descent. Specifically, let us initialize $\boldsymbol{w}^{(0)} = 0$. And repeatedly take a step in the direction of negative gradient with a sufficiently-small constant step-size η till convergence:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla f(\boldsymbol{w}^{(t)}) \tag{10}$$

Let us refer to the solution found by gradient descent at convergence as $\boldsymbol{w}^{gd}$.
Prove that the solution found by gradient descent for least-squares is equal to the result of the following *different* optimization problem:

$$\boldsymbol{w}^{gd} = \arg\min_{\boldsymbol{w} \in \mathbf{R^d}} \quad ||\boldsymbol{w}||_2^2 \tag{11}$$

$$\text{such that} \quad \boldsymbol{X}\boldsymbol{w} = \boldsymbol{y} \tag{12}$$

Hint: For this optimization problem, you must work with the Lagrangian $L(\boldsymbol{w}, \lambda)$, given by

$$L(\boldsymbol{w}, \lambda) = \|\boldsymbol{w}\|^2 + \lambda^\top (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

Obtain $\boldsymbol{w}^*$ and λ^* from the KKT conditions ($\frac{\partial L}{\partial w} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$) to arrive at the result.

Answer 3.

Question 4 (5-5-5-5). (**Paper Review: Vision Transformer**)

In this question, you are going to write a **one page review** of the vision transformer paper. Please structure your review into the following sections:

(4.1) **Summary:**

(a) What is this paper about ?

(b) What is the main contribution ?

(c) Describe the main approach and results. Just facts, no opinions yet.

(4.2) **Strengths:**

(a) Is there a new theoretical insight ?

(b) Or a significant empirical advance ? Did they solve a standing open problem ?

(c) Or a good formulation for a new problem ?

(d) Any good practical outcome (code, algorithm, etc) ?

(e) Are the experiments well executed ?

(f) Useful for the community in general ?

(4.3) **Weaknesses:**

(a) What can be done better ?

(b) Any missing baselines ? Missing datasets ?

(c) Any odd design choices in the algorithm not explained well ? Quality of writing ?

(d) Is there sufficient novelty in what they propose ? Minor variation of previous work ?

(e) Why should anyone care ? Is the problem interesting and significant ?

(4.4) **Reflections:**

(a) How does this relate to other concepts you have seen in the class ?

(b) What are the next research directions in this line of work ?

(c) What (directly or indirectly related) new ideas did this paper give you ? What would you be curious to try ?

Answer 4.