Homework 4

Version: 1.0
Version Release Date: 2022-03-19
Deadline: April 1st, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3] with the tag `hw4`. *Note that 10% of the homework mark (worth 1 pt) may be removed for a lack of neatness.*

The teaching assistants for this assignment are Denny Wu and Emmy Fang.
`mailto:csc413-2022-01-tas@cs.toronto.edu`

[1] `https://markus.teach.cs.toronto.edu/2022-01/courses/16/`
[2] `https://uoft-csc413.github.io/2022/assets/misc/syllabus.pdf`
[3] `https://piazza.com/utoronto.ca/winter2022/csc4132516/`

1 RNNs and Self Attention

For any successful deep learning system, choosing the right network architecture is as important as choosing a good learning algorithm. In this question, we will explore how various architectural choices can have a significant impact on learning. We will analyze the learning performance from the perspective of vanishing /exploding gradients as they are backpropagated from the final layer to the first.

1.1 Warmup: A Single Neuron RNN

Consider an n layered fully connected network that has scalar inputs and outputs. For now, assume that all the hidden layers have a single unit, and that the weight matrices are set to 1 (because each hidden layer has a single unit, the weight matrices have a dimensionality of $\mathbb{R}^{1\times 1}$).

1.1.1 Effect of Activation - Sigmoid [0.5pt]

Lets say we're using the sigmoid activation. Let x be the input to the network and let $f : \mathbb{R}^1 \to \mathbb{R}^1$ be the function the network is computing. Do the gradients necessarily have to vanish or explode as they are backpropagated? Answer this by showing that $0 \le |\frac{\partial f(x)}{\partial x}| \le (\frac{1}{4})^n$, where n is the number of layers in the network.

1.1.2 Effect of Activation - Tanh [0 pt]

Instead of sigmoid, now lets say we're using the tanh activation (otherwise the same setup as in 1.1.1). Do the gradients necessarily have to vanish or explode this time? Answer this by deriving a similar bound as in Sec 1.1.1 for the magnitude of the gradient.

1.2 Matrices and RNN

We will now analyze the recurrent weight matrices under Singular Value Decomposition. SVD is one of the most important results in all of linear algebra. It says that any real matrix $M \in \mathbb{R}^{mxn}$ can be written as $M = U\Sigma V^T$ where $U \in \mathbb{R}^{mxm}$ and $V \in \mathbb{R}^{nxn}$ are square orthogonal matrices, and $\Sigma \in \mathbb{R}^{mxn}$ is a rectangular diagonal matrix with nonnegative entries on the diagonal (i.e. $\Sigma_{ii} \ge 0$ for $i \in \{1, \ldots, \min(m, n)\}$ and 0 otherwise). Geometrically, this means any linear transformation can be decomposed into a rotation/flip, followed by scaling along orthogonal directions, followed by another rotation/flip.

1.2.1 Gradient through RNN [0.5pt]

Let say we have a very simple RNN-like architecture that computes $x_{t+1} = tanh(Wx_t)$. You can view this architecture as a deep fully connected network that uses the same weight matrix at each layer. Suppose the largest singular value of the weight matrix is $\sigma_{max}(W) = \frac{1}{2}$. Show that the largest singular value of the input-output Jacobian has the following bound:

$$0 \le \sigma_{max}(\frac{\partial x_n}{\partial x_1}) \le (\frac{1}{2})^{n-1}$$

(Hint: if $C = AB$, then $\sigma_{max}(C) \le \sigma_{max}(A)\sigma_{max}(B)$. Also, the input-output Jacobian is the multiplication of layerwise Jacobians).

1.3 Self-Attention

In a self-attention layer (using scaled dot-product attention), the matrix of outputs is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where $Q, K, V \in \mathbb{R}^{n\times d}$ are the query, key, and value matrices, n is the sequence length, and d_m is the embedding dimension.

1.3.1 Complexity of Self-Attention [0.5pt]

Recal from Lecture 8, the total cost for scaled dot-product attention scales quadratically with the sequence length n, i.e., $\mathcal{O}(n^2)$. We can generalize the attention equation for any similarity function $sim()$ to the following:

$$\alpha_i = \frac{\sum_{j=1}^{n} sim(Q_i, K_j)V_j}{\sum_{j=1}^{n} sim(Q_i, K_j)} \tag{1.1}$$

where the subscript of a matrix represents the i-th row as a vector. This is equivalent to the Softmax attention if we substitute $sim(q, k) = exp(\frac{q^T k}{\sqrt{d_k}})$. Note that for this generalized equation to be a valid attention equation, the only constraint on $sim()$ is that it need to be non-negative, which is true for all kernel functions $k(x, y) = \phi(x)^T\phi(y)$, for some feature mapping $\phi()$. Show that by applying kernel functions, attention can be calculated with linear complexity (i.e., $\mathcal{O}(n)$).

Hint: Sub in the kernel function for the similarity function into Eq 1.1. Group the terms based on their subscript (i.e., i and j). You can find out more information about applying the kernel function in Katharopoulos et al. [2020]

1.3.2 Linear Attention with SVD [1pt]

It has been empirically shown in Transformer models that the context mapping matrix $P = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)$ often has a low rank. Show that if the rank of P is k and we already have access to the SVD of P, then it is possible to compute self-attention in $\mathcal{O}(nkd)$ time.

1.3.3 Linear Attention by Projecting [1pt]

Suppose we ignore the Softmax and scaling and let $P = QK^\top \in \mathbb{R}^{n\times n}$. Assume P is rank k. Show that there exist two linear projection matrices $C, D \in \mathbb{R}^{k\times n}$ such that $PV = Q(CK)^\top DV$ and the right hand side can be computed in $\mathcal{O}(nkd)$ time. *Hint: Consider using SVD in your proof.*

2 Policy gradients and black box optimization

Very often we have a function f that does not give us useful gradient information: input or output may be discrete; f may be piecewise constant, nowhere differentiable, or have pathological gradients (e.g., a discontinuous saw wave on an incline, whose gradient always points away from the global optimum); or f may be a black box that we cannot backpropagate through. For example, we may have a phone app that labels photos as cats or dogs. This situation is the default in Reinforcement Learning (RL), where we can execute the environment dynamics, but we cannot see or control their internals.

We still, however, want to optimize some score function $J[f] : X \to \mathbb{R}$. For example, in RL, we want to learn a policy that maximizes the non-differentiable environment reward.

When using the REINFORCE strategy, we replaced the θ optimization task with a Monte-Carlo approximation. One of the key factors for a successful REINFORCE application is the *variance*. The higher the variance, the more "noisy" the gradient estimates will be, which can slow down the optimization process. In this section we will derive the variance of the REINFORCE estimator for a simple toy task.

Consider a loss function, $f(\tilde{a})$ which is the zero-one loss of the logistic regression output, $p(a|\theta)$. The input vector has D *independent* scalar features, x_d. We evaluate the performance of the classifier by sampling from the output of the sigmoid μ. The loss function $J(\theta)$ can be written as:

$$\mu = \sigma\left(\sum_{d=1}^{D} \theta_d x_d\right), \tag{2.1}$$

$$p(a|\theta) = Bernoulli(\mu) = \begin{cases} \mu & a = 1 \\ 1-\mu & a = 0 \end{cases}, \tag{2.2}$$

$$\tilde{a} \sim p(a|\theta), \tag{2.3}$$

$$f(\tilde{a}) = \begin{cases} 1 & \tilde{a} = 1 \\ 0 & \tilde{a} = 0 \end{cases}, \tag{2.4}$$

$$J(\theta) = \mathbb{E}_{\tilde{a}\sim p(a|\theta)}[f(\tilde{a})]. \tag{2.5}$$

2.1 Closed form expression for REINFORCE estimator [1pt]

Recall from above that the expression for REINFORCE estimator is:

$$\nabla_\theta J[\theta] = \mathbb{E}_{\tilde{a}\sim p(a|\theta)}\left[f(\tilde{a})\frac{\partial}{\partial\theta}\log p(a=\tilde{a}|\theta)\right] \tag{2.6}$$

We can denote the expression inside the expectation as $g[\theta, \mathbf{x}]$:

$$g[\theta, \tilde{a}] = f(\tilde{a})\frac{\partial}{\partial\theta}\log p(a=\tilde{a}|\theta), \quad \tilde{a} \sim p(a|\theta) \tag{2.7}$$

For this question, derive a closed form for the $g[\theta, \tilde{a}]$ as a deterministic function of $\tilde{a}$, μ, θ, and x_d.

Hint: Substitute in the log likelihood of the Bernoulli distribution.

2.2 Variance of REINFORCE estimator [1pt]

We will derive the variance of the REINFORCE estimator above. Since the gradient is is D-dimensional, the covariance of the gradients will be $D \times D$ matrix. In this question, we will only consider the variance with respect to the first parameter, i.e. $\mathbb{V}[\hat{g}[\theta, \tilde{a}]_1]$ which scalar value corresponding to the first element in the diagonal of the covariance matrix. Derive the variance of the gradient estimator as a function of the first parameter vector: $\mathbb{V}[\hat{g}[\theta, \tilde{a}]_1]$, as a function of μ, θ, and x_d.

Hint: The second moment of a Bernoulli random variable is $\mu(1-\mu)$.

2.3 Convergence and variance of REINFORCE estimator [0 pt]

Comment on the variance in Part 2.2. When do we expect learning to converge slowly in terms of the output of the logistic regression model, μ?

3 Graphs and GNNs

3.1 Properties of the Graph Laplacian

Consider an unweighted and undirected graph $G = (V, E)$, where the vertex set is given as $V = [n] = \{1, 2, 3, ..., n\}$, and the edge set E contains tuples of integers (u, v) for $u, v \in V$, which indicate the presence of an edge between vertices u and v. Recall the definition of the adjacency matrix and the degree matrix,

$$\mathbf{A}(u,v) = \begin{cases} 1, & \text{if } (u,v) \in E. \\ 0, & \text{otherwise.} \end{cases} \qquad \mathbf{D}(u,v) = \begin{cases} deg(u), & u = v. \\ 0, & \text{otherwise.} \end{cases}$$

The graph Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A} \in \mathbb{R}^{n \times n}$.

3.1.1 [0.5pt]

Show that for any $\mathbf{x} \in \mathbb{R}^n$,

$$\mathbf{x}^\top \mathbf{L} \mathbf{x} = \sum_{(u,v) \in E} (\mathbf{x}(u) - \mathbf{x}(v))^2, \tag{3.1}$$

where $\mathbf{x}(u)$ denotes the u-th entry of $\mathbf{x}$.

3.1.2 [0.5pt]

Show that the smallest eigenvalue $\lambda_{\min}(\mathbf{L}) = 0$. Identify the corresponding eigenvector.
Hint: First argue that $\mathbf{L}$ is positive semi-definite. Then use properties of $\mathbf{A}, \mathbf{D}$ to find a vector $\mathbf{v}$ such that $\mathbf{L}\mathbf{v} = \mathbf{0}$.

3.1.3 [0pt]

Show that if G has k connected components, then the multiplicity of the smallest eigenvalue of $\mathbf{L}$ is exactly k.

3.2 Matrix Normalization

Observe that the eigenvalues of the matrices $\mathbf{A}, \mathbf{L}$ can scale with the degree of vertices in the graph (which may be very large in practical settings). To address this problem, we introduce the normalized adjacency matrix and the normalized Laplacian,

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}; \quad \tilde{\mathbf{L}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \tilde{\mathbf{A}}.$$

As we will see, this normalization ensures that the largest eigenvalue of the two matrices remain bounded. In the following, we take G to be a connected (unweighted and undirected) graph.

3.2.1 [0.5pt]

Show that the largest eigenvalue of the normalized Laplacian $\lambda_{\max}(\tilde{\mathbf{L}}) \leq 2$.
Hint: you may use the definition $\lambda_{\max}(\tilde{\mathbf{L}}) = \max_{\|\mathbf{x}\|=1} \mathbf{x}^\top \tilde{\mathbf{L}}\mathbf{x}$ *for* $\mathbf{x} \in \mathbb{R}^n$.

3.2.2 [0.5pt]

Show that for all $i \in [n]$, we have $|\lambda_i(\tilde{\mathbf{A}})| \leq 1$, that is, all the eigenvalues of $\tilde{\mathbf{A}}$ fall in the range between -1 and 1.
Hint: apply the conclusion in 3.2.1.

3.2.3 [0pt]

Recall that a connected graph $G = (V, E)$ is bipartite if and only if V can be partitioned into two *disjoint* sets V_1, V_2 such that every edge in E only connects one vertex in V_1 with one vertex in V_2 (i.e., there is no edge between two vertices in V_1, or two vertices in V_2).

Show that if G is bipartite, then the largest eigenvalue $\lambda_{\max}(\tilde{\mathbf{L}}) = 2$.
Hint: Take $\mathbf{v} \in \mathbb{R}^n$ *to be the leading eigenvector of* $\tilde{\mathbf{L}}$. *For any* $(u, v) \in E$, *argue that we must have* $\mathbf{v}(u) + \mathbf{v}(v) = 0$ *to achieve the equality in 3.2.1. Then show that this constraint can be satisfied when* G *is bipartite.*

3.2.4 [0pt]

Show that if $\lambda_{\max}(\tilde{\mathbf{L}}) = 2$, then G must be bipartite.

3.3 Simplified Graph Convolutional Networks

We now consider a linearized graph convolutional network (GCN) in Wu et al. [2019]. This simplified GCN omits the nonlinear transformation of the features, but instead performs *linear* feature propagation. Specifically, the feature matrix $\mathbf{H}$ at the k-th level is given as

$$\mathbf{H}^{(k)} = \tilde{\mathbf{A}}\mathbf{H}^{(k-1)}\mathbf{\Theta}^{(k)} = \tilde{\mathbf{A}}^k\mathbf{H}^{(0)}\prod_{i=1}^{k}\mathbf{\Theta}^{(i)}, \tag{3.2}$$

where $\mathbf{H}^{(0)} = \mathbf{X}$ is the input features, $\mathbf{\Theta}^{(i)}$ are the trainable parameters in the GCN, and $\tilde{\mathbf{A}}^k$ denotes the k-th power of the normalized adjacency matrix[4]. A softmax nonlinearity is applied to the output layer: $\mathbf{Y} = \mathrm{softmax}(\mathbf{H}^{(K)})$, where K is the total number of layers. Due to the linear

[4]The formulation in Wu et al. [2019] also includes a self-loop for all vertices; we ignore this feature for simplicity.

structure, we may also reparameterize the trainable parameters as $\prod_{i=1}^{K} \mathbf{\Theta}^{(i)} =: \mathbf{\Theta}$, and hence the prediction of the GCN is given as

$$\mathbf{Y} = \text{softmax}\Big(\tilde{\mathbf{A}}^K \mathbf{H}^{(0)} \mathbf{\Theta}\Big). \tag{3.3}$$

3.3.1 [0.5pt]

Using the eigendecomposition, show that the matrix power $\tilde{\mathbf{A}}^K = \prod_{i=1}^{K} \tilde{\mathbf{A}}$ can be computed without multiplying the matrix $\tilde{\mathbf{A}}$ for K times (which can be time-consuming when K is large).

3.3.2 [0.5pt]

For the linearized GCN (3.2), based on the eigenvalue properties in 3.2.2, argue in one or two sentences: is it always beneficial to use a deeper model, i.e., increase the number of layers K?

3.4 Graph Attention Networks (GATs)

Graph Attention Network (GAT) is a novel convolution-style neural network. It operates on graph-structured data and leverages masked self-attentional layers. In this question, we will look at its Graph Attentional Layer. The input to the attention layer is the set of node features $\mathbf{h} = \{\vec{h}_0, \vec{h}_1, ..., \vec{h}_{|V|}\}$, $\vec{h}_i \in \mathbb{R}^F$, where $|V|$ is the number of nodes, F is the number of features in each node. In order to obtain more expressive power, we transform the input features using a shared linear transformation, parameterized by a weight matrix $\mathbf{W} \in \mathbb{R}^{F' \times F}$,to higher-level features, $\mathbf{h}' = \{\vec{h}'_0, \vec{h}'_1, ..., \vec{h}'_{|V|}\}$, $\vec{h}'_i \in \mathbb{R}^{F'}$. We can then calculate the importance of node j's features to node i using some scoring function $e_{ij} = score(\vec{h}'_i, \vec{h}'_j)$. In this implementation, the attention mechanism is a single-layer feedforward neural network (parameterized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$) then followed by a nonlinear activation function, LeakyReLU, as shown in Figure 1. To make the coefficients comparable across different nodes, the attention scores e_{ij} need to be normalized across all choices of j using the softmax function. To inject the graph structure into the mechanism, we perform masked attention, i.e., we only compute e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some neighbourhood of node i in the graph.

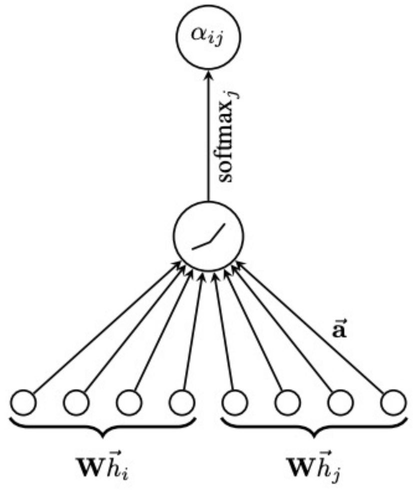

Figure 1: Attention Mechanism Veli**c**ković et al. [2018]

3.4.1 Attention coefficients and Advantages of GAT [0.5pt]

Write out the equation for the coefficients α_{ij} computed by this attention mechanism and list one advantage of GATs over vanilla Graph Convolution Networks.

Hint: You can use the notation $[\ \vec{x} \parallel \vec{y}\ ]$ *to denote concatenation between two vectors* $\vec{x}$ *and* $\vec{y}$.

References

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *CoRR*, abs/2006.16236, 2020. URL `https://arxiv.org/abs/2006.16236`.

Petar Velickoviċ, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`. accepted as poster.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.