Assignment 4

Version: 1.3
Version Release Date: November 19, 2024
Changes by Version:

- **(v1.1)**
  Q1.4: Removed extra softmax in α definition and fixed dimensions of W_Q, W_K, W_V matrices.
  Q2: Fixed missing $\frac{1}{8}$ factor in big Ω notation.
- **(v1.2)**
  Q2: Relaxed the question to allow for an arbitrary constant factor c factor, fixed matrix M's dimensions being reversed, and updated the hint.
  Q1.4.1: Updated the kernel $w = [-1, 0, 1]^T \rightarrow w = [2, 0, 1]^T$, making the question actually solvable.
- **(v1.3)**
  Q1.4: Included condition that the input sequence x is positive for all elements.

Deadline: Thursday, November 28th, at 11:59pm.

Submission: You must submit two files through MarkUs: (1) a PDF file containing your writeup, titled `a4-writeup.pdf`, and (2) your code file `a4-gnn.ipynb, a4-unet.ipynb`. There will be sections in the notebook for you to write your responses. Your writeup must be typed. Make sure that the relevant outputs (e.g. `print_gradients()` outputs, plots, etc.) are included and clearly visible.
See the syllabus on the course website for detailed policies. You may ask questions about the assignment on Piazza. *Note that 10% of the assignment mark (worth 2 pts) may be removed for lack of neatness.*

Written Assignment

What you have to submit for this part

For reference, here is everything you need to hand in for the first half of the PDF report `a4-writeup.pdf`.

- **Problem 1:** 1.1.2, 1.2.1, 1.3.1, 1.3.2, 1.4.1, 1.4.2
- **Problem 2:** 2.1.1

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 - ReLU [0pt]

Lets say we're using the ReLU 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 $0 \leq |\frac{\partial f(x)}{\partial x}| \leq 1$.

1.1.2 Effect of Activation - Different weights [0.5pt]

Solve the problem in 1.1.1 by assuming now the weights are not 1. You can assume that the i-th hidden layer has weight w_i. Do the gradients necessarily have to vanish or explode as they are backpropagated? 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. $\Sigma_{ii} \geq 0$ 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 $x_{t+1} = \text{sigmoid}(Wx_t)$. 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 $\sigma_{max}(W) = \frac{1}{4}$. Show that the largest singular value of the input-output Jacobian has the following bound:

$$0 \leq \sigma_{max}(\frac{\partial x_n}{\partial x_1}) \leq (\frac{1}{16})^{n-1}$$

(Hint: if $C = AB$, then $\sigma_{max}(C) \leq \sigma_{max}(A)\sigma_{max}(B)$. 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).

1.3.2 Linear Attention with SVD [0.5pt]

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 [0pt]

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.*

1.4 Relative Attention

Relative attention We implement relative attention, as in traditional Softmax attention, however we now include a new term that allows us to reweight the attention scores based on the distance between inputs. Now assume that we are working with a single-headed local self-attention mechanism on a one-dimensional sequence $x \in \mathbb{R}^n$ s.t. $x_i \geq 0$ for all i and $W_Q, W_K, W_V \in \mathbb{R}$.

$$\alpha_{i,j}(Q, K, p) = \left(\frac{Q_i K_j}{\sqrt{d_k}} + p_{i-j} \right)$$

where (p_k) is a sequence of scalars and $Q, K \in \mathbb{R}^n$. Note that here we allow the index k to take on negative values.

$$\text{ReltativeAttention}(Q, K, V, p) = \text{softmax}(\alpha(Q, K, p))V$$

And our attention layer is simply:

$$\text{RelAttnLayer}(x; W_Q, W_K, W_V, p) = \text{ReltativeAttention}(W_Q x, W_K x, W_V x, p)$$

1.4.1 Implement a 1D Convolution [0.5pts]

Implement the following $1D$-convolution with kernel $w = [2, 0, 1]^T$,as using the RelAttnLayer. Specifically, you need to implement:

$$\text{Conv1D}(x; w)_i = \sum_{j=-1}^{1} x_{i+j} w_{j+2}$$

You may ignore the behavior of your implementation in the edge cases when $i > n - 1$ and $i < 1$. Please specify the p, W_Q, W_K, and W_V you used and argue why they closely approximate this function in the relevant range.

1.4.2 Implement Max Pooling [0.5pts]

Now we will use RelAttnLayer to approximate 1d max-pooling with a stride of 1 and a window size of $2k + 1$ around the current input. Recall that max pooling with a stride 1 and width of 2k + 1 is simply:

$$\text{MaxPool}(x)_i = \max_{-k \leq m \leq k} x_{m+i}$$

Again you may ignore the edge cases in your solutions $i > n - k$ and $i < k + 1$. Please specify the p, W_Q, W_K, and W_V you used and argue why they approximately implement this function in the relevant range.

2 Johnson-Lindenstrauss Lemma and Almost Orthogonal Vectors

While a large network might train on billions of facts and word associations, it manages to store and retrieve this information from relatively low-dimensional embeddings. It is hypothesized that these sparse features of the data are stored in the form of many almost-orthogonal directions in the model's latent states [Elhage et al., 2022]. We will show that in addition to giving us insight into how the distance between two vectors can be preserved in a low-dimensional space, the Johnson-Lindenstrauss lemma suggests that a great many sparse features can be linearly encoded and retrieved from a relatively small continuous vector space!

Johnson-Lindenstrauss Lemma (**JL Lemma**) For any $\epsilon \in (0, 1)$ and any set of n points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ and $k \in \mathbb{N}$ if $k \geq 8(\ln n)/\epsilon^2$, then there exists a matrix $M \in \mathbb{R}^{k \times d}$ such that for all $x_i, x_j \in X$:

$$(1 - \epsilon)||x_i - x_j||_2^2 \leq ||Mx_i - Mx_j||_2^2 \leq (1 + \epsilon)||x_i - x_j||_2^2$$

2.1 Almost orthogonal vectors

We will consider two vectors δ almost-orthogonal iff $|x_i^\top x_j| \leq \delta$.

2.1.1 Show there are exponentially many almost-orthogonal vectors [1pts]

Show that for some $c > 0$ there are $\Omega(e^{\frac{n\delta^2}{c}})$ many almost-orthogonal unit vectors in $\mathbb{R}^n$.
In other words, show that for some arbitrary constant $c > 0$ and any $n \in \mathbb{N}$, $\delta > 0$ there is a finite set of vectors $U \subset \mathbb{R}^n$ s.t., $|U| \in \Omega(e^{\frac{n\delta^2}{c}})$, $\forall u \in U$, $||u||_2 = 1$ and $\forall u_i, u_j \in U$ if $u_i \neq u_j$ then $|u_i^\top u_j| \leq \delta$.

Hints: Take $\epsilon = \frac{1}{2}\delta$ and use the JL-lemma on the vectors $X = \{0, e_1, ..., e_d\}$ where $e_1, ..., e_d$ are the standard basis vectors. You will also need to use the fact that for two vectors x, y, $||x - y||_2^2 = ||x||_2^2 + ||y||_2^2 - 2x^\top y$.

Programming Assignment

What you have to submit for this part

For reference, here is everything you need to hand in:

- This is the second half of your PDF report `a4-writeup.pdf`. Please include the solutions to the following problems. You may choose to export `a4-gnn.ipynb, a4-unet.ipynb` as a PDF and attach it to the first half of `a4-writeup.pdf`.
  - **Question 3:** 3.1, 3.2, 3.3, 3.4, 3.5, 3.6
  - **Question 4:** 4.1, 4.2
- Your code file `a4-gnn.ipynb, a4-unet.ipynb`

3 Graph Convolution Networks[5pt]

For this part of the assignment, you will implement the vanilla version of Graph Convolution Networks (GCN) Kipf and Welling [2016] and Graph Attention Networks (GAT) Veli**c**ković et al. [2018].

Basics of GCN:

Recall from the lecture, the goal of a GCN is to learn a function of signals/features on a graph $G = (V, E)$, which takes as inputs:

1. the input features of each node, $x_i \in \mathcal{R}^F$ (in matrix form: $X \in \mathcal{R}^{|V| \times F}$)
2. some information about the graph structure, typically the adjacency matrix A

Each convolutional layer can be written as $H^{(l+1)} = f(H^{(l)}, A)$, for some function $f()$. The $f()$ we are using for this assignment is in the form of $f(H^{(l)}, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(l)}W^{(l)})$, where $\hat{A} = A + Identity$ and $\hat{D}$ is diagonal node degree matrix ($\hat{D}^{-1}\hat{A}$ normalizes $\hat{A}$ such that all rows sum to one). Let $\tilde{A} = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$. The GCN we will implement takes two convolution layers, $Z = f(X, A) = softmax(\tilde{A} \cdot Dropout(ReLU(\tilde{A}XW^{(0)})) \cdot W^{(1)})$

Basics of GAT:

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 assignment, we will implement the graph attention layer.

Dataset:

The dataset we used for this assignment is Cora Sen et al. [2008]. Cora is one of standard citation network benchmark dataset (just like MNIST dataset for computer vision tasks). It that consists of 2708 scientific publications and 5429 links. Each publication is classified into one of 7 classes. Each publication is described by a word vector (length 1433) that indicates the absence/presence of the corresponding word. This is used as the features of each node for our experiment. The task is to perform node classification (predict which class each node belongs to).

Experiments:

Open [GNN notebook link] on Colab and answer the following questions.

1. [1pt] **Implementation of Graph Convolution Layer**

   Complete the code for GraphConvolution() Class

2. [1pt] **Implementation of Graph Convolution Network**

   Complete the code for GCN() Class

3. [0.5pt] **Train your Graph Convolution Network**

   After implementing the required classes, now you can train your GCN. You can play with the hyperparameters in args.

4. [1.5pt] **Implementation of Graph Attention Layer**

   Complete the code for GraphAttentionLayer() Class

5. [0.5pt] **Train your Graph Attention Network**

   After implementing the required classes, now you can train your GAT. You can play with the hyperparameters in args.

6. [0.5pt] **Compare your models**

   Compare the evaluation results for Vanilla GCN and GAT. Comment on the discrepancy in their performance (if any) and briefly explain why you think it's the case (in 1-2 sentences).

Deliverables

Create a section in your report called **Graph Convolution Networks**. Add the following:

- Screenshots of your `GraphConvolution`, `GCN` implementations. Highlight the lines you've added.
- Screenshots of your GCN training output, you can just screenshot the last 10 epochs with test set results.
- Screenshots of your `GraphAttentionLayer` implementations. Highlight the lines you've added.
- Screenshots of your GAT training output, you can just screenshot the last 10 epochs with test set results.
- Your response to the written component of question 3.6. Your analysis should not exceed **3** sentences.

4 U-Nets [2.5 pts]

For this part of the assignment we will continue where we left off in assignment 2 and implement a more effective architecture for image segmentation. A U-Net Ronneberger et al. [2015] that consists of convolutional auto encoder with skip connections.

You can access the notebook here `https://colab.research.google.com/drive/1FoLWfyEmBbkCCf_a6Qw0mLmJN-8GkZp1?usp=sharing` as always create a local copy of the notebook in you're own drive and connect to a GPU instance to run.

Marks

- [1.5 pts] **Implement the U-Net** Complete the code in the `Up`, `Down` and `UNet` classes.
- [1 pts] **Train the U-Net**

  Train the U-net with and without its skip connections enabled and report on the results.

Deliverables

Create a section in your report called **U-Nets**. Add the following:

- Screenshots of your `UNet`, `Up` and `Down` implementations. Highlight the lines you've added.
- Screenshots of your final segmentation and your training curves with skip connections enabled and disabled.
- Your response to the written component of question 5.2. Your analysis should not exceed **3** sentences.

References

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL `http://arxiv.org/abs/1609.02907`.

Petar Veli**c**ković, 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.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008. URL `http://www.cs.iit.edu/~ml/pdfs/sen-aimag08.pdf`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.