
EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade Homework 6

This homework is due on Oct 17, at 10:59PM.

1. Memory considerations when using GPUs (Coding Question)

In this homework, you will run [GPUMemory.ipynb](#) to train a ResNet model on CIFAR-10 using PyTorch and explore its implications on GPU memory.

We will explore various systems considerations, such as the effect of batch size on memory usage and how different optimizers (SGD, SGD with momentum, Adam) vary in their memory requirements.

It is strongly recommended that you start early on this question, since colab daily GPU limits may require you to complete this question over a few days with breaks in between.

- (a) Managing GPU memory for training neural networks (Notebook Section 1).
 - (i) **How many trainable parameters does ResNet-152 have?** What is the **estimated size of the model in MB?**
 - (ii) Which GPU are you using? **How much total memory does it have?**
 - (iii) After you load the model into memory, **what is the memory overhead (MB) of the CUDA context loaded with the model?**
- (b) Optimizer memory usage (Notebook Section 2).
 - (i) **What is the total memory utilization during training with SGD, SGD with momentum and Adam optimizers?** Report in MB individually for each optimizer.
 - (ii) **Which optimizer consumes the most memory?** Why?
- (c) Batch size, learning rates and memory utilization (Notebook Section 3)
 - (i) **What is the memory utilization for different batch sizes (4, 16, 64, 256)? What is the largest batch size you were able to train?**
 - (ii) **Which batch size gave you the highest accuracy at the end of 10 epochs?**
 - (iii) **Which batch size completed 10 epochs the fastest (least wall clock time)? Why?**
 - (iv) Attach your **training accuracy vs wall time plots** with your written submission.

2. Graph Dynamics and GNN Concepts

In this problem, we will explore connections between graph dynamics and graph neural networks (GNNs).

Diagrams in this question were taken from <https://distill.pub/2021/gnn-intro>. This blog post is an excellent resource for understanding GNNs and contains interactive diagrams. Some graph neural network methods operate on the full adjacency matrix. Others, such as those discussed in <https://distill.pub/2021/gnn-intro/>, at each layer apply the same local operation to each node based on inputs from its neighbors.

This problem is designed to:

- Show connections between these methods.

- Show that for a positive integer k , the matrix A^k has an interesting interpretation. That is, the entry in row i and column j gives the number of walks of length k (i.e., a collection of k edges) leading from vertex i to vertex j .

To do this, let's consider a very simple deep linear network, defined as follows:

- Its underlying graph has n vertices, with adjacency matrix A . That is, $A_{i,j} = 1$ if vertices i and j are connected in the graph and 0 otherwise.
- It has n vertices in each layer, corresponding to the n vertices of the underlying graph.
- Each vertex has n channels.
- The input to each node in the 0-th layer is a one-hot encoding of own identity. That is, the node i in the graph has input $(0, \dots, 0, \underbrace{1}_{i\text{-th entry}}, 0, \dots, 0)$.
- The weight connecting node i in layer k to node j in layer $k+1$ is $A_{i,j}$.
- At each layer, the operation at each node is simply to sum up the weighted sum of its inputs and to output the resulting n -dim vector to the next layer. You can think of these as being depth-wise operations if you'd like.

- (a) **Write the output of the j -th node at layer k in this network in terms of the matrix A .**

(Hint: This output is an n -dimensional vector since there are n output channels at each layer.)

- (b) Recall that a path from i to j in a graph is a sequence of vertices that starts with i , ends with j , and every successive vertex in the sequence is connected by an edge in the graph. The length of a path is the number of edges in it.

Here is some helpful notation:

- $V(i)$ is the set of vertices that are connected to vertex i in the graph.
- $L_k(i, j)$ is the number of distinct paths that go from vertex i to vertex j in the graph where the number of edges traversed in the path is exactly k .
- By convention, there is exactly 1 path of length 0 that starts at each node and ends up at itself. That is, $L_0(i, j) = 1_{i=j}$.

Prove that the i -th output of node j at layer k in the network above is the count of how many paths there are from i to j of length k .

(Hint: Induct on k .)

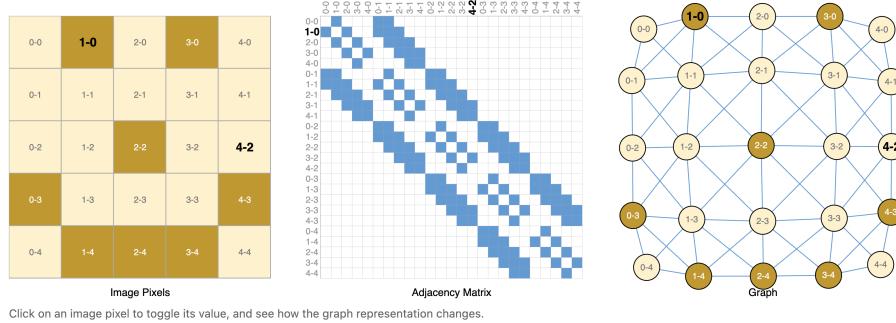
- (c) The GNN we have worked on so far is essentially linear, since the operations performed at each layer are permutation-invariant locally at each node, and can be viewed as essentially doing the exact same thing at each vertex in the graph based on inputs coming from its neighbors. This is called "aggregation" in the language of graph neural nets.

If we represent the graph as a matrix, with the activations of the i -th node in the i -th row, **what is the update function?**

In the case of the computations in previous parts, **what is the update function that takes the aggregated inputs from neighbors and results in the output for this node?**

- (d) The simple GNN described in the previous parts counts paths in the graph. If we were to replace sum aggregation with max aggregation, **what is the interpretation of the outputs of node j at layer k ?**
- (e) You are studying how organic molecules break down when heated. For each molecule, you know the element for each atom, which other atoms it is connected to, the length of the bonds, and the type of molecule it is (carbohydrate, protein, etc.) You are trying to predict which bond, if any, will break first if the molecule is heated.

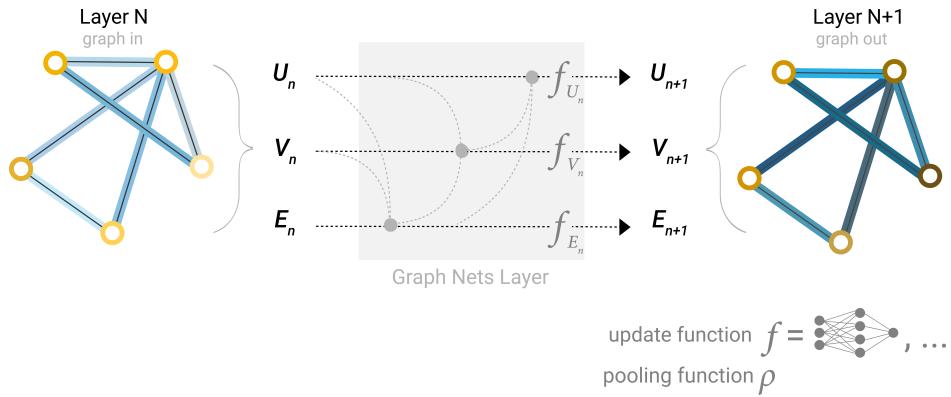
- (i) **How would you represent this as a graph?** (What are the nodes, edges, and global state representations? Is it directed or undirected?)
- (ii) **How would you use the outputs of the last GNN layer to make the prediction?**
- (iii) **How would you encode the node representation for the input to the GNN?**

**Figure 1:** Images as Graphs

- (f) There are analogs of many ConvNet operations which can be done with GNNs. As Figure 1 illustrates, we can think of pixels as nodes and pixel adjacencies as similar to edges. Graph-level classification tasks, for instance, are analogous to image classification, since both produce a single, global prediction. **Fill out the rest of the table.** (Not all rows have a perfect answer. The goal is to think about the role an operation serves in one architecture and whether you could use a technique which serves a similar role in the other architecture.)

CNN	GNN
Image classification	Graph-level prediction problem
	Node-level prediction problem
Color jitter data augmentation (adjusting the color or brightness of an image)	
Image flip data augmentation	
Channel dropout	
Zero padding edges	
ResNet skip connections	
Blurring an image	Predicting missing values of nodes

- (g) **If you're doing a graph-level classification problem, but node values are missing for some of your graph nodes, how would you use this graph for prediction?**
- (h) Consider the graph neural net architecture shown in Figure 2. It includes representations of nodes (V_n), edges (E_n), and global state (U_n). At each timestep, each node and edge is updated by aggregating neighboring nodes/edges, as well as global state. The global state is updated by getting information from all nodes and edges. For more details on the architecture setup, see <https://distill.pub/2021/gnn-intro/#passing-messages-between-parts-of-the-graph>.

**Figure 2:** GNN architecture

- (i) If we double the number of nodes in a graph which only has node representations, how does this change the number of learned weights in the graph? How does it change the amount of computation used for this graph if the average node degree remains the same? What if the graph is fully connected? (Assume you are not using a global state representation).
- (ii) Where in this network are learned weights/parameters?
- (iii) The diagram provided shows undirected edges. How would you incorporate directed edges?

3. Graph Neural Networks

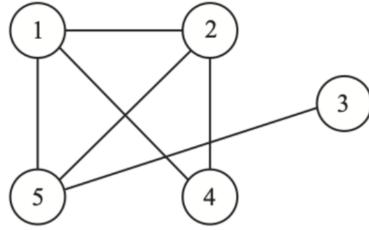
For an undirected graph with no labels on edges, the function that we compute at each layer of a Graph Neural Network must respect certain properties so that the same function (with weight-sharing) can be used at different nodes in the graph. Let's focus on a single particular "layer" ℓ . For a given node i in the graph, let $s_i^{\ell-1}$ be the self-message (i.e. the state computed at the previous layer for this node) for this node from the preceding layer, while the preceding layer messages from the n_i neighbors of node i are denoted by $m_{i,j}^{\ell-1}$ where j ranges from 1 to n_i . We will use w with subscripts and superscripts to denote learnable scalar weights. If there's no superscript, the weights are shared across layers. Assume that all dimensions work out.

- (a) Tell which of these are valid functions for this node's computation of the next self-message s_i^ℓ .
For any choices that are not valid, briefly point out why.

Note: we are *not* asking you to judge whether these are useful or will have well behaved gradients. Validity means that they respect the invariances and equivariances that we need to be able to deploy as a GNN on an undirected graph.

- (i) $s_i^\ell = w_1 s_i^{\ell-1} + w_2 \frac{1}{n_i} \sum_{j=1}^{n_i} m_{i,j}^{\ell-1}$
- (ii) $s_i^\ell = \max(w_1 s_i^{\ell-1}, w_2 m_{i,1}^{\ell-1}, w_3 m_{i,2}^{\ell-1}, \dots, w_{n_i+1} m_{i,n_i}^{\ell-1})$ where the max acts component-wise on the vectors.
- (iii) $s_i^\ell = \max(w_1 s_i^{\ell-1}, w_2 m_{i,1}^{\ell-1}, w_2 m_{i,2}^{\ell-1}, \dots, w_2 m_{i,n_i}^{\ell-1})$ where the max acts component-wise on the vectors.

- (b) We are given the following simple graph on which we want to train a GNN. The goal is binary node classification (i.e. classifying the nodes as belonging to type 1 or 0) and we want to hold back nodes 1 and 4 to evaluate performance at the end while using the rest for training. We decide that the surrogate loss to be used for training is the average binary cross-entropy loss.

**Figure 3:** Simple Undirected Graph

nodes	1	2	3	4	5
y_i	0	1	1	1	0
\hat{y}_i	a	b	c	d	e

Table 1: y_i is the ground truth label, while \hat{y}_i is the predicted probability of node i belonging to class 1 after training.

Table 1 gives you relevant information about the situation.

Compute the training loss at the end of training.

Remember that with n training points, the formula for average binary cross-entropy loss is

$$\frac{1}{n} \sum_x \left(y(x) \log \frac{1}{\hat{y}(x)} + (1 - y(x)) \log \left(\frac{1}{1 - \hat{y}(x)} \right) \right)$$

where the x in the sum ranges over the training points and $\hat{y}(x)$ is the network's predicted probability that the label for point x is 1.

- (c) Suppose we decide to use the following update rule for the internal state of the nodes at layer ℓ .

$$\mathbf{s}_i^\ell = \mathbf{s}_i^{\ell-1} + W_1 \frac{\sum_{j=1}^{n_i} \tanh(W_2 \mathbf{m}_{i,j}^{\ell-1})}{n_i} \quad (1)$$

where the \tanh nonlinearity acts element-wise.

For a given node i in the graph, let $\mathbf{s}_i^{\ell-1}$ be the self-message for this node from the preceding layer, while the preceding layer messages from the n_i neighbors of node i are denoted by $\mathbf{m}_{i,j}^{\ell-1}$ where j ranges from 1 to n_i . We will use W with subscripts and superscripts to denote learnable weights in matrix form. If there's no superscript, the weights are shared across layers.

- (i) **Which of the following design patterns does this update rule have?**
 - Residual connection
 - Batch normalization
- (ii) **If the dimension of the state s is d -dimensional and W_2 has k rows, what are the dimensions of the matrix W_1 ?**
- (iii) **If we choose to use the state $\mathbf{s}_i^{\ell-1}$ itself as the message $\mathbf{m}^{\ell-1}$ going to all of node i 's neighbors, please write out the update rules corresponding to (1) giving \mathbf{s}_i^ℓ for the graph in Figure 3 for nodes $i = 2$ and $i = 3$ in terms of information from earlier layers. Expand out all sums.**

4. Exploring Deep Learning Tooling

Deep learning in practice often requires the use of various tooling in order make the learning workflow efficient. Among these include tools for creating graphs and organizing experiments. These kinds of tools

can aid in careful ablation studies, and can accelerate research. We will explore the use of two different tools in this question: tensorboard and wandb.

- (a) Complete the first notebook: [tensorboard.ipynb](#). Provide your graphs here. What is easy about tensorboard? What do you dislike? Would it still be easy to use when we need to run massive amounts of experiments? How organized is it?
- (b) Complete the second notebook: [wandb.ipynb](#). No need to provide your graphs. What does wandb have that tensorboard does not?

5. Zachary's Karate Club (Coding)

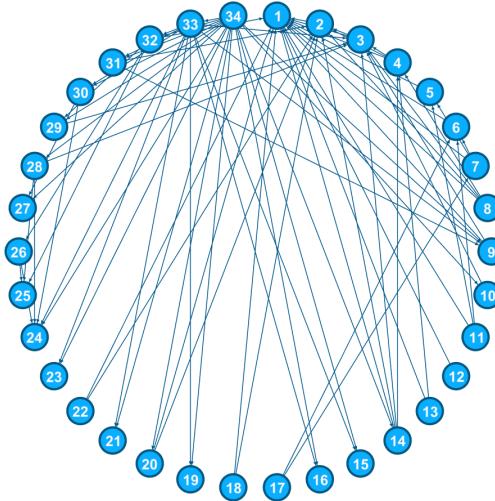


Figure 4: Zachary's Karate Club Graph

Zachary's Karate Club (ZKC) is a social network of a university karate club, described in the paper "An Information Flow Model for Conflict and Fission in Small Groups" by Wayne W. Zachary.

A social network captures 34 members of a karate club, documenting links between pairs of members who interacted outside the club.

During the study a conflict arose between the officer/ administrator ("John A") and the instructor "Mr. Hi", which led to the split of the club into two.

Half of the members formed a new club around Mr. Hi; members from the other part found a new instructor or gave up karate.

Based on collected data Zachary correctly assigned all but one member of the club to the groups they actually joined after the split. You could read more about it here <https://www.jstor.org/stable/3629752>, and here https://commons.wikimedia.org/wiki/File:Social_Network_Model_of_Relationships_in_the_Karate_Club.png

We will train a GNN to cluster people in the karate club in such that people who are more likely to associate with either the officer or Mr. Hi will be close together, while the distance between the 2 classes will be far.

In the original paper titled "Semi-Supervised Classification with Graph Convolutional Networks" that can be found here <https://arxiv.org/pdf/1609.02907.pdf>, the authors framed this as a node-level

classification problem on a graph. We will pretend that we only know the affiliation labels for some of the nodes (which we'll call our training set) and we'll predict the affiliation labels for the rest of the nodes (our test set).

Implement all the TODOs in `zkc.ipynb` and include your notebook with your submission.

- (a) Go through `q_zkc.ipynb`. We want our network to be aware of information about the nodes themselves instead of only the neighborhood, so we add self loops our adjacency matrix. The [paper](#) called this \tilde{A} . **Compute \tilde{A} to add self loops to your adjacency matrix.**
- (b) **Write a function that takes in \tilde{A} as argument and returns the $\tilde{A}^{SymNorm}$ adjacency matrix.**
- (c) The other input to our GNN is the graph node matrix X which contains node features. For simplicity, we set X to be the identity matrix because we don't have any node features in this example. **Generate the feature input matrix X .**
- (d) We will now implement a single layer GNN. **Implement the forward and backward pass functions for `GNN_Layer` class.** Details can be found in the notebook.
- (e) **Run the forward and backward passes and ensure the checks pass.**
- (f) We are now ready to setup our classification network! **Use the GNN and Softmax layers to setup the network.**
- (g) **Instantiate the GNN model with the correct input and output dimensions.**
- (h) With the model, data and optimizer ready, **fill in the todos in the training loop function and train your model. Plot the clustered data.**
- (i) **Explain why we obtain 100% on accuracy on our test set, yet we see in the plot that 2 samples seem to be misclassified.**

6. Implementing Muon (Coding Question)

In this assignment, you will implement key components of a Muon optimizer in PyTorch. For this assignment, we recommend using Google Colab as some PCs or laptops may not support GPU-based PyTorch. You will be implementing the code at [q_coding_muon.ipynb](#), then answering the questions in the notebook in your submission. The notebook itself does not need to be submitted.

- (a) Notice that in this Newton-Schulz implementation, we initially scale the Frobenius norm to be at most $\sqrt{3}$. **Can you explain why we choose this particular scaling?** Comment on why this is better than using 1.

(Hint: Inspect the roots of the cubic polynomial. What is the connection between the roots and the convergence properties of the singular values? You can refer to Discussion 4 for the answer)

- (b) Note that Muon requires that parameters are 2D matrices of shape $d_{out} \times d_{in}$. However, we know that parameters that are convolutional kernels have shape $c_{out} \times c_{in} \times k \times k$, where c denotes number of channels and k is kernel size.

Modern implementations of convolutional layers will (implicitly) transform an input image \mathbf{x} of shape $c_{in} \times h \times w$ to \mathbf{x}' such that each column of \mathbf{x}' has size $c_{in} \cdot k \cdot k$ and corresponds to one flattened "receptive field" of the image (or one patch of the image that a filter passes over to compute one output value). This allows the GPU to use heavily optimized matrix-multiplication capabilities directly. You may look up `im2col` for more details and take a look at <https://arxiv.org/abs/2110.03901> if you are interested.

Given this fact, **how do we modify the convolutional kernel into a $d_{out} \times d_{in}$ matrix C such that the output of the convolutional layer can be expressed as $C\mathbf{x}'$?**

- (c) **Which optimizer performed best between Muon, SGD, and AdamW? Also copy the resulting plots into the submission as well.**
- (d) Compare the loss curves between Muon and MuonSVD. **Are the results expected? Explain why.**
- (e) (Optional) Our implementation of Newton-Schulz is suboptimal in the polynomial used for convergence. The community has developed quintic polynomials that converge faster while still being efficient. **Implement an improved Newton-Schulz and compare. Comment on the speed advantage of the improved Muon relative to the MuonSVD.**
(Hint: You can modify the number of iterations by setting the ns_iters parameter in the Muon optimizer)
- (f) (Optional) **What were the best choices for hyperparameters for Muon? What about for AdamW?**

7. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student! We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

- (a) **What sources (if any) did you use as you worked through the homework?**
- (b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)
- (c) **Roughly how many total hours did you work on this homework?**

Contributors:

- Romil Bhardwaj.
- Linyuan Gong.
- Jerome Quenum.
- Olivia Watkins.
- Anant Sahai.
- Anrui Gu.
- Matthew Lacayo.
- Past EECS 282 and 227 Staff.
- Naman Jain.
- Liam Tan.
- Joey Hong.
- Gireeja Ranade.