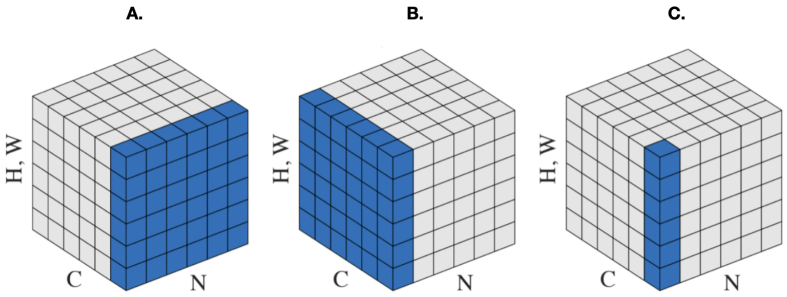
EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Homework 7

This homework is due on Wednesday, October 21, 2026, at 10:59PM.

1. Batch Normalization for CNN

(a) Consider the following diagram where the shaded blocks are the entries participating in one normalization step for a CNN-type architecture. N represents the mini-batch, H, W represent the different pixels of the "image" at this layer, and C represents different channels.



- **Which one denotes the process of batch normalization?** Please use ■ for your selections.

  ☐ A ☐ B ☐ C

- **Which one denotes layer normalization?** Please use ■ for your selections.

  ☐ A ☐ B ☐ C

(b) Consider a simplified BN where we do not divide by the standard deviation of the data batch. Instead, we just de-mean our data batch before applying the scaling factor γ and shifting factor β. For simplicity, consider scalar data in an n-sized batch: $[x_1, x_2, \ldots, x_n]$. Specifically, we let $\hat{x}_i = x_i - \mu$ where μ is the average $\frac{1}{n}\sum_{j=1}^{n} x_j$ across the batch and output $[y_1, y_2, \ldots, y_n]$ where $y_i = \gamma\hat{x}_i + \beta$ to the next layer. Assume we have a final loss L somewhere downstream. **Calculate $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$ for $j = 1, \ldots, n$ as well as γ and β as needed.**

Numerically, **what is $\frac{\partial L}{\partial x_1}$ when $n = 1$ and our input batch just consists of $[x_1]$ with an output batch of $[y_1]$?** (Your answer should be a real number. No need to justify.)

What happens when $n \to \infty$? (Feel free to assume here that all relevant quantities are bounded.)

2. Batchnorm, Dropout and Convolutions (Coding Question)

In this assignment, you will implement batch normalization, dropout, and convolutions using NumPy and PyTorch. For this assignment, we recommend using Google Colab as some PCs or laptops may not support GPU-based PyTorch. You can sign up for a free trial of Colab Pro for students and teachers by following this link.

 All Rights Reserved. This may not be publicly shared without explicit permission.

Attention: This coding task will take approximately 4 to 6 hours to complete, taking into account the time required for training the model. Please plan accordingly and start early to ensure adequate time to finish.

The assignment consists of three parts:

Implementing Batch Norm and Dropout. Open bn_drop.ipynb and follow the instructions in the notebook. You will implement the forward and backward propagation of dropout and batch normalization in NumPy. A GPU runtime is not required for this part.

Implement convolution and spatial batch norm. Open cnn.ipynb and follow the instructions in the notebook. You will implement the forward and backward propagation of convolutional layers and spatial dropout in NumPy. A GPU runtime is not required for this part.

Use deep learning framework. Open pytorch_cnn.ipynb and follow the instructions in the notebook. For this part, you will need to switch to a GPU runtime (details can be found in the notebook). You will implement a convolutional neural network with convolution layers, batch normalization, and dropout using PyTorch and train it on a GPU. You also have the opportunity to improve or design your own neural network.

Please answer the following questions in your submission:

(a) **Draw the computational graph of training-time batch normalization.** The input of the computational graph should be $\mathbf{X}, \gamma, \beta$, the output of the computational graph should be $\mathbf{Y}$, and the intermediate nodes are $\mu, \sigma^2, \mathbf{Z}$.

(b) **Derive the closed-form back-propagation of a batch normalization layer (during training).** Include the answer in your written assignment.

Specifically, given $\mathrm{dy}_{i,j} = \dfrac{\partial \mathcal{L}}{\partial Y_{i,j}}$ for every i, j, Please derive $\dfrac{\partial \mathcal{L}}{\partial X_{i,j}}$ for every i, j as a function of $\mathrm{dy}, \mathbf{X}, \mu, \sigma^2, \epsilon, \gamma, \beta$.

(c) **Explain what you see in this experiment. What does it suggest about dropout?**

(d) **Briefly describe your neural network design and the procedure of hyperparameter tuning.**

3. 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:

 All Rights Reserved. This may not be publicly shared without explicit permission.

- 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 $(0, \cdots, 0, \underbrace{1}_{i\text{-th entry}}, 0, \cdots, 0)$.
- 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?**

 All Rights Reserved. This may not be publicly shared without explicit permission.

(iii) **How would you encode the node representation for the input to the GNN?**

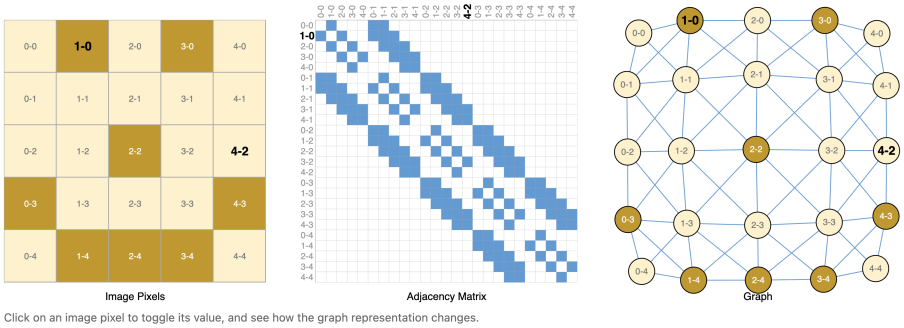


Figure 1: Images as Graphs

(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 then 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.

 All Rights Reserved. This may not be publicly shared without explicit permission.

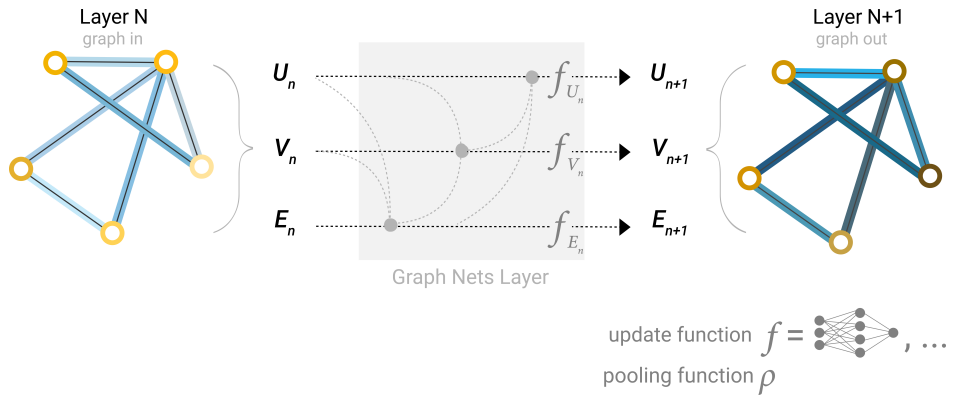


Figure 2: GNN architecture

(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?**

4. Zachary's Karate Club (Coding)

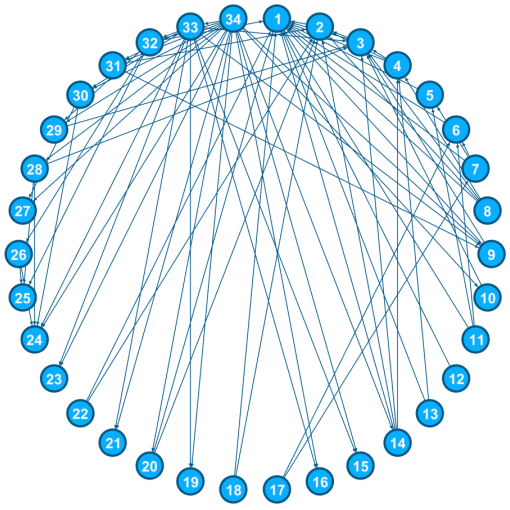


Figure 3: Zachary's Karate Club Graph

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.

 All Rights Reserved. This may not be publicly shared without explicit permission.

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 beween 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) **Report the final training and test accuracies and list any misclassified node IDs. Does a node lying near the other class in the embedding plot necessarily mean that the classifier misclassified it? Explain using the predicted classes. Is 100% test accuracy guaranteed for this experiment?**

5. Implementing RNNs (Coding)

This problem involves filling out this notebook.

(a) **Implement Section 1A in the notebook**, which constructs a vanilla RNN layer. This layer implements the function

$$h_t = \sigma(W^h h_{t-1} + W^x x_t + b)$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

where W^h, W^x, and b are learned parameter matrices, x is the input sequence, and σ is a nonlinearity such as tanh. The RNN layer "unrolls" across a sequence, passing a hidden state between timesteps and returning an array of hidden states at all timesteps.

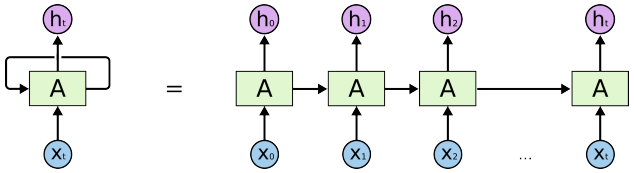


Figure 4: Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Copy the outputs of the *"Test Cases"* code cell and paste it into your submission of the written assignment.

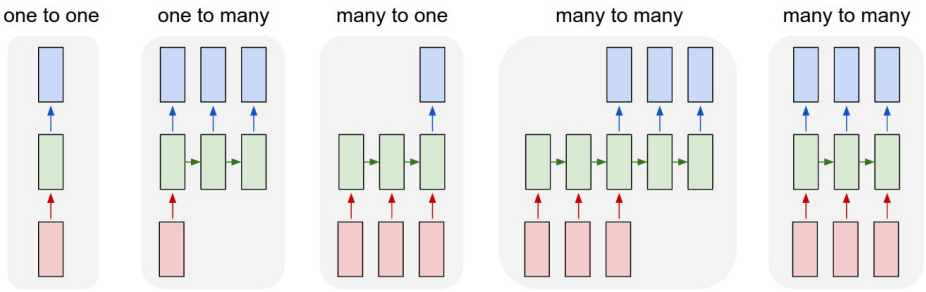
(b) **Implement Section 1.B of the notebook**, in which you'll use this RNN layer in a regression model by adding a final linear layer on top of the RNN outputs.

$$\hat{y}_t = W^f h_t + b^f$$

We'll compute one prediction for each timestep.

Copy the outputs of the *"Tests"* code cell and paste it into your submission of the written assignment.

(c) RNNs can be used for many kinds of prediction problems, as shown below. In this notebook we will look at many-to-one prediction and aligned many-to-many prediction.



We will use a simple averaging task. The input X consists of a sequence of numbers, and the label y is a running average of all numbers seen so far.

We will consider two tasks with this dataset:

- Task 1: predict the running average at all timesteps
- Task 2: predict the average at the last timestep only

Implement Section 1.C in the notebook, in which you'll look at the synthetic dataset shown and implement a loss function for the two problem variants.

Copy the outputs of the *"Tests"* code cell and paste it into your submission of the written assignment.

 All Rights Reserved. This may not be publicly shared without explicit permission.

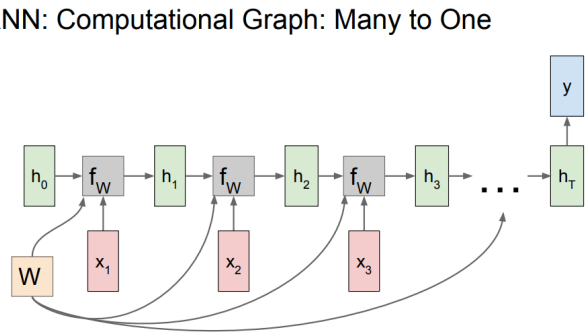


Figure 5: Image source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

(d) Consider an RNN which outputs a single prediction at timestep T. As shown in Figure 5, each weight matrix W influences the loss by multiple paths. As a result, the gradient is also summed over multiple paths:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial h_T}\frac{\partial h_T}{\partial W} + \frac{\partial \mathcal{L}}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial W} + \ldots + \frac{\partial \mathcal{L}}{\partial h_1}\frac{\partial h_1}{\partial W} \tag{1}$$

When you backpropagate a loss through many timesteps, the later terms in this sum often end up with either very small or very large magnitude - called vanishing or exploding gradients respectively. Either problem can make learning with long sequences difficult.

Implement Notebook Section 1.D, which plots the magnitude at each timestep of $\frac{\partial \mathcal{L}}{\partial h_t}$. Play around with this visualization tool and try to generate exploding and vanishing gradients.

Include a screenshot of your visualization in the written assignment submission.

(e) **If the network has no nonlinearities, under what conditions would you expect the exploding or vanishing gradients with for long sequences? Why?** (Hint: it might be helpful to write out the formula for $\frac{\partial \mathcal{L}}{\partial h_t}$ and analyze how this changes with different t). **Do you see this pattern empirically using the visualization tool in Section 1.D in the notebook** with last_target_only=True?

(f) Compare the magnitude of hidden states and gradients when using ReLU and tanh nonlinearities in Section 1.D in the notebook. **Which activation results in more vanishing and exploding gradients? Why?** (This does not have to be a rigorous mathematical explanation.)

(g) **What happens if you set last_target_only = False in Section 1.D in the notebook? Explain why this change affects vanishing gradients. Does it help the network's ability to learn dependencies across long sequences?** (The explanation can be intuitive, not mathematically rigorous.)

(h) (Optional) We can create multi-layer recurrent networks by stacking layers as shown in Figure 6. The hidden state outputs from one layer become the inputs to the layer above.

 All Rights Reserved. This may not be publicly shared without explicit permission.

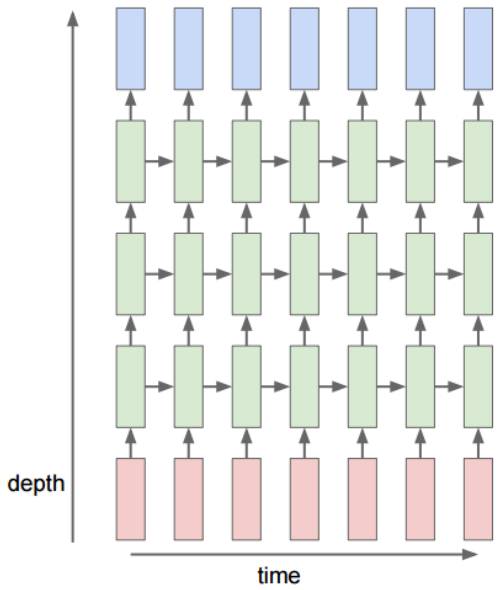


Figure 6: Image source: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Implement notebook Section 1.K and run the last cell to train your network. You should be able to reach training loss < 0.001 for the 2-layer networks, and <.01 for the 1-layer networks.

6. RNNs for Last Name Classification (Coding)

Please follow the instructions in this notebook. You will train a neural network to predict the probable language of origin for a given last name / family name in Latin alphabets.

(a) Although the neural network you have trained is intended to predict the language of origin for a given last name, it could potentially be misused. **In what ways do you think this could be problematic in real-world applications**?

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: Matteo Guarrera, Mert Cemri, Sizhe Chen, Kevin Li, Saagar Sanghavi, Sheng Shen, Suhong Moon, Jake Austin, Anant Sahai, Linyuan Gong, Jerome Quenum, Olivia Watkins, Anrui Gu, Matthew Lacayo, Past EECS 282 and 227 Staff, Romil Bhardwaj, Dhruv Shah.

 All Rights Reserved. This may not be publicly shared without explicit permission.