Endterm Assignment

Mathematics for Deep Learning

Due: 31st December(Tentative) EOD

A Few Friendly Notes & Guidelines

We are excited to see how you tackle these challenges! This assignment is designed to help you bridge the gap between theory and practice. Here is how to get started:

- **The Path to Ratification:** Completing this assignment is necessary for your ratification, and it is somewhat long, so please start as quickly as possible!
- **Focus on Logic, Not Perfection:** We do not expect every solution to be a formal masterpiece. What we want to see is your thought process, show us your logical steps and how you arrived at your conclusions.
- **Use Your Resources Wisely:** Feel free to use AI as a personal tutor to help you grasp difficult concepts. However, to truly master the material, please ensure the final code and solutions you submit are your own original work.
- **Learn Together:** We highly encourage you to discuss these problems in your groups! Collaboration plays a significant role in the learning process. Just make sure the final submission reflects your individual understanding.
- **Integrity Matters:** To keep things fair and rewarding for everyone, please stick to these guidelines. Avoiding copied work ensures you get the most out of this experience.
- **Submission through GitHub :** Share your work by creating a GitHub repository! Please name it: `"Math for Deep Learning [Roll-No] [Name]"`. We'll share a submission form in due time.
- **Your Choice of Format:** Whether you prefer the classic feel of handwritten notes or the professional polish of LaTeX (which is a great skill to practice!), we welcome both.

Remember, the goal is to explore, learn, and have fun with the math. Please do not worry much about ratification; that will naturally be a by-product of you learning and enjoying the assignment! We can't wait to see your solutions!

Part 1: General Questions

Q1. Singular Value Decomposition (SVD)
Give a rough idea of how SVD could be used for the compression of information like images. Make your own example and show how it is possible.

Q2. Dimensionality Reduction
Reduce dimension from 2 to 1. You are free to use the internet for this problem but justify what you have done based on what was taught in class.

Table 1: Dataset for Dimensionality Reduction

Feature	**Example 1**	**Example 2**	**Example 3**	**Example 4**
X_1	4	8	13	7
X_2	11	4	5	14

Q3. Stochastic vs. Batch Gradient Descent
Mention the difference between Stochastic Gradient Descent (SGD) and Batch Gradient Descent. Which is better if a dataset is much more complex? Provide a mathematical proof as well.

Q4. Gradient Descent with Fixed Learning Rate
Mathematically write how gradient descent works with a fixed learning rate of 0.001. Create a training loop (pseudo-code is preferred, although a mathematical explanation would also suffice).

Q5. Adam Optimizer
Read about the Adam optimizer. What sort of models is it mostly used for? Mathematically analyze the process. Write a few insights about your research.

Q6. Cross Entropy Loss
Read about Cross Entropy Loss and try to work out why it is better for classification models (refer to class notes and the internet for your research). Mathematical proofs or insights are required.

Q7. Activation Layers
What are activation layers? Define a few types and what they are mostly used for. Mention a few mathematical requirements for the same (properties).

Part 2: Case Study

The Mathematical Engine of Neural Networks

In Deep Learning, activation functions introduce non-linearity, allowing networks to learn complex decision boundaries. However, their mathematical properties dictate how well the network learns via backpropagation.
Consider the Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

It squashes inputs between 0 and 1. While this mimics biological firing rates, it has two major flaws. First is **saturation**: at extremes, the curve flattens, causing derivatives to approach zero and killing gradients. Second is the **lack of zero-centering**: the output is always positive. This seemingly minor detail has massive implications for the dynamics of gradient descent.
To solve this, we often use ReLU:

$$f(x) = \max(0, x)$$

It is computationally efficient and does not saturate for positive values. However, researchers constantly seek better functions, such as Swish ($f(x) = x \cdot \sigma(x)$), which attempts to combine the smooth curve of sigmoid with the non-saturation of ReLU.

Case Study Questions

Q8. Consider a hidden neuron that receives its inputs $(x_1, x_2, \ldots, x_n)$ from a previous layer that utilized the Sigmoid activation function. During backpropagation, the gradient for any specific weight w_i connecting to this neuron is calculated using the chain rule term:

$$\frac{\partial \text{Loss}}{\partial w_i} = \delta \cdot x_i$$

where δ represents the error signal flowing back from deeper layers.

Explain how the strictly positive nature of the Sigmoid inputs ($x > 0$) mathematically constrains the direction in which all weights feeding into this neuron can be updated for a single data point. Discuss why this constraint makes the optimization inefficient (often resulting in a "zig-zagging" path toward the minimum).

Q9. A proposed "modern" activation function is the Swish function, defined as $f(x) = x \cdot \sigma(x)$, where $\sigma(x)$ is the sigmoid function.

Derive the analytical expression for the derivative $f'(x)$ exclusively in terms of $f(x)$ and $\sigma(x)$. Then, argue why computing this gradient during backpropagation is computationally more expensive (in terms of Floating Point Operations) than computing the gradient of the ReLU function.

Part 3: Mini-Project & Design

Q10. Mini Project: Convolutional Neural Networks from Scratch
Remember how in class we built a Deep Neural Network from the ground up? We started with the humble `Perceptron` class, stacked them together to create a `DenseLayer`, and finally assembled a full `Model` that could classify MNIST digits. We saw firsthand how

flattening a 28×28 image into a 784-dimensional vector loses all spatial information: the network treats pixels as unordered features.

Now it's your turn to go further.

Your Task: Build a **Convolutional Neural Network (CNN)** from scratch using PyTorch, following the same spirit as our class demonstration. Just like we created custom `Perceptron` and `DenseLayer` classes without using `nn.Linear`, you will now implement:

- **A Convolution Layer:** Implement a class (e.g., `ConvLayer`) that performs the 2D convolution operation manually using tensor operations. Your layer should support learnable kernels/filters (use `nn.Parameter`).
- **A Pooling Layer:** Implement a simple Max Pooling or Average Pooling layer to downsample feature maps.
- **The Full CNN Model:** Combine your convolutional layers, pooling layers, activation functions (ReLU), and the `DenseLayer`/`Perceptron` we built in class for the final classification head.

Requirements:

(a) Train your CNN on the MNIST dataset.

(b) Do **NOT** use `nn.Conv2d`, `nn.MaxPool2d`, or any other pre-built convolution/pooling modules. You must implement the core operations yourself.

(c) Document your code with comments explaining the mathematical operations (just like the class notebook).

Hints:

- Although the project may seem daunting, you only need to implement a single convolution and pooling function; the rest of the code closely follows what you already have.
- Backpropagation is handled automatically by PyTorch's `autograd`.
- You may explore `torch.nn.functional.unfold` and `torch.nn.functional.fold`; this is optional but potentially helpful.
- Think about input/output dimensions: if you have an input of size (H, W) and a kernel of size (k, k) with stride 1 and no padding, the output will be $(H - k + 1, W - k + 1)$.
- Start simple, get a single convolution layer working before building the full network.

Submission: Submit your Jupyter notebook (.ipynb) containing all code, training logs.

Bonus: Share a brief analysis comparing the CNN's performance with the DNN we trained in class.

Q11. Design Your Own Activation Function
Design your own activation function based on what was taught. You encouraged to research more about it and attach a paper of your own showing why it could be used.

Q12. Neural Network from Scratch (C++)
Write functions for a neural network in C++. Specifically:

- Training loop
- Loss calculator
- Gradient descent

- Testing loop

Note: Out of Q11 and Q12, both are to be done but completing one is sufficient for ratification (try completing one question first).