Homework 1

Deep Learning (Fall’25) Undergraduate version

Title: Getting some familiarity with the Artificial Neural Networks from Scratch

Objective:

- Gain hands-on experience building and training a basic neural network.
- Understand fundamental concepts of forward propagation, backpropagation, and gradient descent.
- Learn the importance of hyperparameter tuning and debugging in deep learning.
- Harboring the “Keep it simple” attitude during the deep learning experiments!

Part 1: Conceptual Warm-Up (No Coding)

Goal: Ensure students understand the foundational concepts through explanation and derivation.

Tasks:

1. **Short Answer Questions**:
    a) Derive the gradient update rule for the weight(s) for a single neuron with one input feature (i.e., field, or attribute) and the sigmoid activation function, i.e., $\sigma(x) = \frac{1}{1+e^{-x}}$.
    b) Explain the role of the activation function in a neural network. Please guess without searching textbook/the Internet.
    c) What is overfitting, and how can it be mitigated? *List couple of techniques to reduce overfitting.*
    d) Can you guess what might be an underfitting problem? How can that be mitigated? *List couple of techniques to reduce underfitting*
2. **Math Problem**:
    a) For the following small binary classification dataset (4 samples each with two input features (x_1, x_2) and one target feature (y)). Compute the forward passes and the gradients manually for a neural network with 1 hidden layer (with 2 neurons) and 1 output neuron.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1. A Small dataset

Part 2: Build a Neural Network from Scratch (Coding)

Goal: Students implement a simple neural network <u>without relying on deep learning libraries like TensorFlow or PyTorch.</u>

Tasks:

3. Implement a neural network with:
    a) **1 input layer**, **1 hidden layer (with 2 neurons)**, and **1 output layer**.
    b) Use a non-linear activation function (e.g., sigmoid, ReLU, tanh, etc) for each of the neuron.
    c) Please do not forget to introduce the bias inputs for the two neural layers (i.e., hidden layer and output layer).
    d) Consider mean squared error as the loss/error function.
4. Train it on the small dataset of 4 samples (denoted by rows in Fig. 1), each having two input features (x_1, x_2) and one target feature (y).

Requirements:

- Implement **forward propagation**, **backpropagation** algorithm manually. Please refer to the boilerplate code given to find placeholders where you'll put your implementations.
- Train the network for a fixed number of epochs (10,000) and plot the loss over time.

Specific Guidelines:

- **Do not** use any pre-built neural network libraries. Only use libraries for basic operations like NumPy.
- Write detailed comments in your code to explain each step of your implementation.

Part 3: Experimentation and Analysis

Goal: Encourage critical thinking.

Tasks:

5. **Hyperparameter Tuning**:
    - Experiment with different learning rates and hidden layer sizes.
    - Analyze how these changes impact the convergence and performance of your model.
6. **Visualization**:
    - Plot decision boundaries after training your model.
    - Provide insights into how the model separates the data.

Part 4: Reflective Questions

Goal: Make students reflect on the learning process.

Tasks:

7. What challenges did you face in implementing backpropagation? How did you overcome them?
8. Explain the importance of debugging in neural network training and provide one strategy that helped you debug effectively.
9. Discuss how the training process would change if you used a different activation function (e.g., tanh instead of ReLU, or vice versa).

Deliverables:

- ✓ **Code**: A well-documented Python script (or jupyter notebook) implementing the neural network.
- ✓ **Report**: A PDF file containing --
    - Answers to the conceptual and reflective questions.
    - Plots (e.g., loss over epochs, decision boundaries).
        a) **Loss Curve**: Show how the loss decreases over epochs.
        b) **Decision Boundary**: Visualize how the trained model separates the small dataset.
    - Observations from experimentation.

Grading Criteria:

- **Correctness** (40%): Does the neural network work as intended?
- **Clarity** (20%): Is the code well-documented and easy to follow?
- **Analysis** (20%): Are the experiments and reflections thoughtful and insightful?
- **Effort** (20%): Evidence of effort in debugging, experimentation, and explanation.

Template Code

Attached in Canvas a boilerplate code: `hw1-boilerplate.py`. You need to work on the places throughout the boilerplate code to complete the said tasks.

Hints for Students

- **Debugging**: Print intermediate outputs (e.g., gradients) and ensure they make sense. Check for exploding/vanishing gradients if the model fails to train.
- **Visualization**: Use a library like Matplotlib to visualize decision boundaries and loss curves. Decision boundary plot can look like the following:

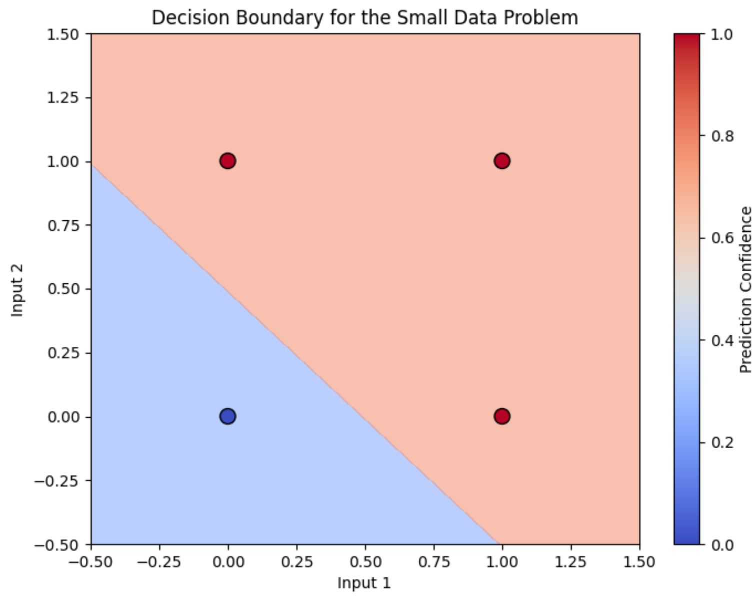

Figure 2. Decision boundary learned for the binary classification problem

- **Hyperparameters**: Experiment with learning rates (0.01, 0.1, etc.) and hidden layer sizes (e.g., 1–10 neurons).