IIT BOMBAY

Lab Assignment #7
Points: 5

Main TAs: *Parshant Arora, Saurabh Kumar, Govind Saju, Vishal Tapase*

Course: *CS 335* – Instructor: *Preethi Jyothi*
Due date: *September 25, 2023*

General Instructions

1. Download lab7.tgz(`https://drive.google.com/file/d/1xtyK3UJdkQGK2J5SZ_UOmPKme5shxZkO/view?usp=sharing`) from Moodle, and extract the file to get `lab7`.
2. Your final submission is due on Moodle on or before **11.59 pm on Sept 25, 2023**.
3. For your final submission, you only need to submit `nn_template.py` with all the required functions fully implemented.
4. You will get 4 points if we are able to successfully run `nn_template.py` and obtain perfect test accuracy. You will get an additional 1 point if your test accuracies on a new unseen train/test set match our test accuracies. This is with keeping all the hyperparameters fixed as in the original template file. That is, `input_dim` = 2, `hidden_dim` = 4, `learning_rate` = 0.05, `num_epochs` = 100 and `np.random.seed(42)`.

Feedforward Neural Network and Backpropagation

For this problem, you will implement a single-hidden layer feedforward neural network training from scratch. The network has:

- An input layer and a single hidden layer, with dimensionalities `input_dim` and `hidden_dim`, respectively
- A single output node with a binary cross-entropy loss

`nn_template.py` contains skeleton code with the following functions within the class `NN` that should be completed:

- `__init__`: Initialize the weights and biases for the two affine layers between 1) the input and the hidden layer and, 2) the hidden layer and the output node. Use random initializations for all these weights by sampling from a Gaussian distribution with a mean of 0 and a standard deviation of 1.
- `forward`: Compute the activations for all the hidden nodes and the output node. Use the **sigmoid** activation function (denoted by σ) everywhere. Use separate variables to compute the weighted sum of inputs coming into each node (e.g., $\mathbf{z}_1 = \mathbf{w}_1\mathbf{x} + \mathbf{b}_1$), and the output after applying the activation function to the weighted sum (e.g., $\mathbf{a}_1 = \sigma(\mathbf{z}_1)$).
- `backward`: In this function, you will compute gradients of the loss function with respect to all the weights and biases. Once all the gradients have been computed, you will update the weights and biases of both affine layers using a gradient descent step. For example, $\mathbf{w}_1 = \mathbf{w}_1 - \eta \frac{\partial L}{\partial \mathbf{w}_1}$ where η is a learning rate and L is the binary cross-entropy loss function. Recall the expression for the binary cross-entropy loss:

  $$L(y, \hat{y}) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

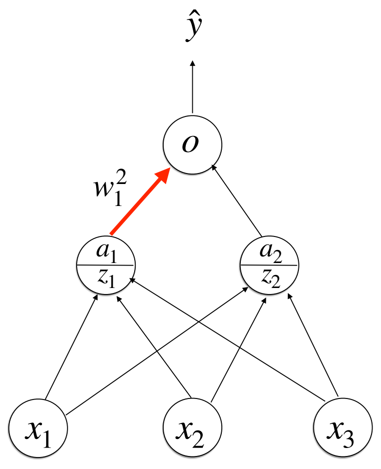
  The derivative of this loss with respect to $\hat{y}$ is:

  $$\frac{\partial L}{\partial \hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right)$$

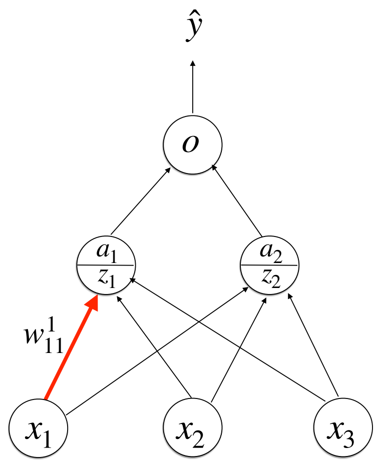
  To derive the other gradients, let's consider each layer at a time. First, the output layer. Consider a single weight w_1^2 in the output layer, highlighted in red in Figure (A) below.

  $$\frac{\partial L}{\partial w_1^2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o} \cdot \frac{\partial(w_1^2 a_1 + w_2^2 a_2 + b_2)}{\partial w_1^2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o} \cdot a_1$$

  where $\hat{y} = \sigma(o)$, hence $\frac{\partial \hat{y}}{\partial o} = \sigma(o)(1-\sigma(o))$ and b_2 is the (scalar) bias weight corresponding to the output node. Gradients for all the other weights in the output layer can be similarly computed.



(A)



(B)

Next, the hidden layer. Consider a single weight w_{11}^1 highlighted in red in Figure (B) above.

$$\begin{aligned}
\frac{\partial L}{\partial w_{11}^1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o} \cdot \frac{\partial (w_1^2 a_1 + w_2^2 a_2 + b_2)}{\partial w_{11}^1} \\
&= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o} \cdot w_1^2 \left(\frac{\partial a_1}{\partial w_{11}^1} \right) + 0 \\
&= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o} \cdot w_1^2 \cdot \sigma(z_1)(1 - \sigma(z_1)) \cdot x_1
\end{aligned}$$

since $a_1 = \sigma(z_1)$. Gradients for all the other weights in the hidden layer can be similarly computed.

Note that the above expressions are gradients for a single example. You will need to sum the gradients across all training examples.

- `train`: Over `num_epochs`, the function `train` calls the forward and backward passes and computes full gradients for all the training examples (stored in `X` and `y`).
- `pred`: This function has already been implemented for you. It writes the predicted output probabilities and predicted labels to an output file.

If your implementation is correct, your code will converge in less than 100 epochs and your test accuracy will be a perfect 1.0.