

LIGN 167: Problem Set 3

October 31, 2023

Collaboration policy: You may collaborate with up to three other students on this problem set (that is, total group size of 4). You should submit **one response** per group, using group submissions on Gradescope. When you submit your work, you must indicate who was in the group, and what each of your individual contributions were.

Starter code: We are providing starter code for the problem set. This starter code contains function signatures for all of the functions that you are defining. You **must** use this starter code when writing answers to the problems. If you do not follow the format of the starter code, **you will not receive credit**.

IMPORTANT: For each problem, you are expected to do 6 things:

1. Provide the correct answer to the problem.
2. Provide the solution that GPT-4 gives. This will be GPT-4 Solution 1.
3. After each solution that GPT-4 gives, respond by saying, "I think you made a mistake. I want you to find it and fix it." Provide the new solution (if any) that GPT-4 generates. This will be GPT-4 Solution 2.
4. If there are errors in Solution 1 from GPT-4, explain where they occur.
5. If there are errors in Solution 2 from GPT-4, explain where they occur.
6. For each problem, include links to your conversations with GPT-4.

Please submit this work in **three** files. One file should be named **ps3.py**. This should contain the correct answer to the problem. The second file should be named **gpt_solution1.py**. This should contain Solution 1 generated by GPT-4, as well as your explanations of the mistakes that occurred. The third file should be **gpt_solution2.py**. This should contain Solution 2 generated by GPT-4, as well as your explanations of the mistakes that occurred.

For **gpt_solution1.py** and **gpt_solution2.py**, please use the same format as the starter code.

In this problem set we will be implementing backpropagation for a multi-layer perceptron. This network is illustrated in Figure 1, and has the following mathematical definition. The vector \vec{r}^0 is defined in terms of the input x , which is a scalar, and the weight matrix W^0 :

$$\vec{r}^0 = \begin{bmatrix} r_0^0 \\ r_1^0 \\ r_2^0 \end{bmatrix} = \begin{bmatrix} w_0^0 \cdot x \\ w_1^0 \cdot x \\ w_2^0 \cdot x \end{bmatrix} = W^0 \cdot x \quad (1)$$

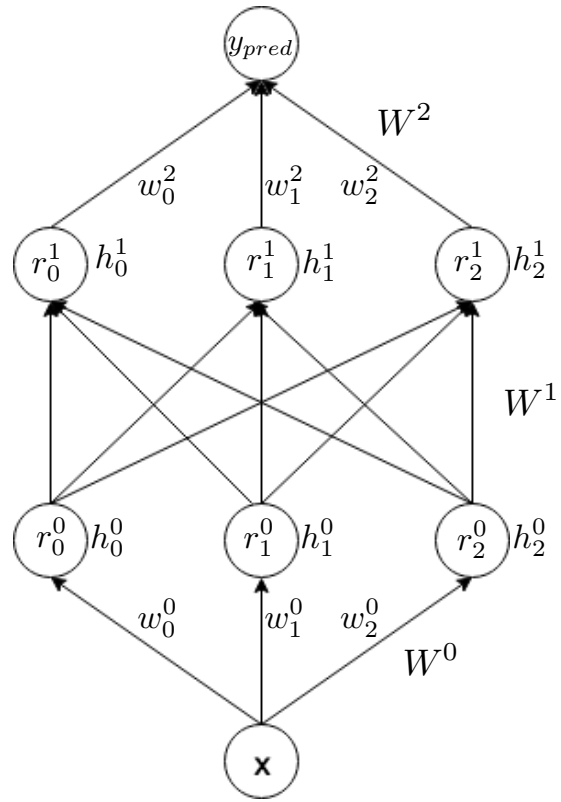


Figure 1: Our multi-layer perceptron.

Here we are using the following definition of W^0 :

$$W^0 = \begin{bmatrix} w_0^0 \\ w_1^0 \\ w_2^0 \end{bmatrix} \quad (2)$$

The first hidden layer \vec{h}^0 is defined by applying a non-linearity (ReLU) to \vec{r}^0 :

$$\vec{h}^0 = \begin{bmatrix} h_0^0 \\ h_1^0 \\ h_2^0 \end{bmatrix} = \begin{bmatrix} ReLU(r_0^0) \\ ReLU(r_1^0) \\ ReLU(r_2^0) \end{bmatrix} = ReLU(\vec{r}^0) \quad (3)$$

The next layer, \vec{r}^1 , is defined as follows:

$$\vec{r}^1 = \begin{bmatrix} r_0^1 \\ r_1^1 \\ r_2^1 \end{bmatrix} = \begin{bmatrix} w_{0,0}^1 \cdot h_0^0 + w_{0,1}^1 \cdot h_1^0 + w_{0,2}^1 \cdot h_2^0 \\ w_{1,0}^1 \cdot h_0^0 + w_{1,1}^1 \cdot h_1^0 + w_{1,2}^1 \cdot h_2^0 \\ w_{2,0}^1 \cdot h_0^0 + w_{2,1}^1 \cdot h_1^0 + w_{2,2}^1 \cdot h_2^0 \end{bmatrix} = W^1 \cdot \vec{h}^0 \quad (4)$$

The matrix in this equation is defined by:

$$W^1 = \begin{bmatrix} w_{0,0}^1 & w_{0,1}^1 & w_{0,2}^1 \\ w_{1,0}^1 & w_{1,1}^1 & w_{1,2}^1 \\ w_{2,0}^1 & w_{2,1}^1 & w_{2,2}^1 \end{bmatrix} \quad (5)$$

The second hidden layer \vec{h}^1 is defined by applying a ReLU to \vec{r}^1 :

$$\vec{h}^1 = \begin{bmatrix} h_0^1 \\ h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} ReLU(r_0^1) \\ ReLU(r_1^1) \\ ReLU(r_2^1) \end{bmatrix} = ReLU(\vec{r}^1) \quad (6)$$

Finally, the output y_{pred} , which is a scalar value, is defined by:

$$y_{pred} = w_0^2 \cdot h_0^1 + w_1^2 \cdot h_1^1 + w_2^2 \cdot h_2^1 = W^2 \cdot \vec{h}^1 \quad (7)$$

We have a dataset that consists of two parts: $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_n\}$. Each x_i and y_i is a scalar. The loss associated with a datapoint x_i, y_i is defined by:

$$\ell_i = (y_{pred,i} - y_i)^2 \quad (8)$$

Here we are writing $y_{pred,i}$ for the neural network's prediction given input x_i . The total loss L can be written:

$$L(\theta|X, Y) = \frac{1}{n} \sum_{i=1}^n \ell_i \quad (9)$$

The parameter term θ captures all of the model parameters that are being learned, in this case: W^0 , W^1 , and W^2 .

In the starter code that we have provided, we have given you an implementation of the *forward direction* of the neural network. That is, the provided function `mlp` will compute the output y_{pred} of the network given a particular input x . In the problems, you will be implementing the *backwards direction* for the network, calculating the partial derivatives of the loss function with respect to the weight parameters.

The function `mlp` in the starter code returns a Python dictionary called `variable_dict`. The dictionary contains the value of all of the nodes in the network, after giving the network a particular input value x_i . We will be using this `variable_dict` throughout the rest of the problem set. You should spend some time reading through the code for `mlp`, to understand how it is constructed.

Problem 1. In this problem we will begin implementing the backpropagation algorithm, starting from the top of the network. You should write a function `d_loss_d_ypredicted`, which calculates the partial derivative $\frac{\partial \ell_i}{\partial y_{pred}}$. The loss ℓ_i is defined by Equation 8.

The function should take two arguments: `variable_dict` and `y_observed`. `variable_dict` is a dictionary containing the values of all of the nodes of the network, for a particular input value x_i (as discussed above). `y_observed` is a real number, which equals the value y_i observed for the input x_i .

Hint: retrieve the network's predicted value y_{pred} by calling `variable_dict['y_predicted']`.

Problem 2. Write a function `d_loss_d_W2` which takes two arguments, `variable_dict` and `y_observed`. `variable_dict` is a dictionary of network node values, and `y_observed` is a real number, as in the previous problem.

The function should compute the partial derivative $\frac{\partial \ell_i}{\partial W^2}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial W^2} = \left[\frac{\partial \ell_i}{\partial w_0^2} \quad \frac{\partial \ell_i}{\partial w_1^2} \quad \frac{\partial \ell_i}{\partial w_2^2} \right] \quad (10)$$

These three partial derivatives should be returned as a 1×3 NumPy array, in the same order as shown in the equation above.

Hint: call `d_loss_d_ypredicted` from the previous problem, and retrieve the network's value for the layer \tilde{h}^1 from `variable_dict`. Then take partial derivatives of Equation 7.

Problem 3. Write a function `d_loss_d_h1`, which takes three arguments: `variable_dict`, `W2`, and `y_observed`. The arguments `variable_dict` and `y_observed` are the same as previous problems. The argument `W2` is a 1×3 NumPy array, which represents the weight matrix W^2 from Equation 7.

The function should compute the partial derivative $\frac{\partial \ell_i}{\partial h^1}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial h^1} = \left[\frac{\partial \ell_i}{\partial h_0^1} \quad \frac{\partial \ell_i}{\partial h_1^1} \quad \frac{\partial \ell_i}{\partial h_2^1} \right] \quad (11)$$

These three partial derivatives should be returned as a 1×3 NumPy array, in the same order as the equation above. (For the remainder of the problems, when a NumPy array is being returned, it should be in the same order as the corresponding equation.)

Problem 4. Write a function `relu_derivative`, which takes a single argument x . The value x is a real number.

It should return the derivative $\frac{dReLU}{dx}(x)$, where the *ReLU* function is defined by:

$$\begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } otherwise \end{cases} \quad (12)$$

Problem 5. Write a function `d_loss_d_r1`, which takes three arguments: `variable_dict`, `W2`, and `y_observed`. These arguments should be the same as in Problem 3. The function should compute the partial derivative $\frac{\partial \ell_i}{\partial r^1}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial r^1} = \begin{bmatrix} \frac{\partial \ell_i}{\partial r_0^1} & \frac{\partial \ell_i}{\partial r_1^1} & \frac{\partial \ell_i}{\partial r_2^1} \end{bmatrix} \quad (13)$$

These values should be returned as a 1×3 NumPy array.

Hint: Take partial derivatives in Equation 6, and use the function `relu_derivative` that you defined in Problem 4.

Problem 6. Write a function `d_loss_d_W1`, which takes three arguments: `variable_dict`, `W2`, and `y_observed`. These arguments should be the same as in Problem 3. The function should compute a matrix of partial derivatives $\frac{\partial \ell_i}{\partial W^1}$:

$$\frac{\partial \ell_i}{\partial W^1} = \begin{bmatrix} \frac{\partial \ell_i}{\partial w_{0,0}^1} & \frac{\partial \ell_i}{\partial w_{0,1}^1} & \frac{\partial \ell_i}{\partial w_{0,2}^1} \\ \frac{\partial \ell_i}{\partial w_{1,0}^1} & \frac{\partial \ell_i}{\partial w_{1,1}^1} & \frac{\partial \ell_i}{\partial w_{1,2}^1} \\ \frac{\partial \ell_i}{\partial w_{2,0}^1} & \frac{\partial \ell_i}{\partial w_{2,1}^1} & \frac{\partial \ell_i}{\partial w_{2,2}^1} \end{bmatrix} \quad (14)$$

These partial derivatives should be returned as a NumPy array of dimension 3×3 .

To do this you should take partial derivatives in Equation 4.

Hint: This is not necessary, but it may be convenient to use the NumPy function `np.outer`, which computes the outer product of two (one-dimensional) arrays. You can also use `Einsum`.

Problem 7. Write a function `d_loss_d_h0`, which takes four arguments: `variable_dict`, `W1`, `W2`, and `y_observed`. The arguments `variable_dict`, `W2`, and `y_observed` should be the same as in previous problems. The argument `W1` is a 3×3 matrix which represents the weight matrix W^1 .

The function should compute the partial derivative $\frac{\partial \ell_i}{\partial h^0}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial h^0} = \begin{bmatrix} \frac{\partial \ell_i}{\partial h_0^0} & \frac{\partial \ell_i}{\partial h_1^0} & \frac{\partial \ell_i}{\partial h_2^0} \end{bmatrix} \quad (15)$$

These partial derivatives should be returned as a 1×3 NumPy array.

Do this by taking partial derivatives in Equation 4.

Problem 8. Write a function `d_loss_d_r0`, which takes four arguments: `variable_dict`, `W1`, `W2`, and `y_observed`. These four arguments should be the same as in Problem 7. The function should compute the partial derivative $\frac{\partial \ell_i}{\partial r^0}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial r^0} = \left[\frac{\partial \ell_i}{\partial r_0^0} \quad \frac{\partial \ell_i}{\partial r_1^0} \quad \frac{\partial \ell_i}{\partial r_2^0} \right] \quad (16)$$

These partial derivatives should be returned as a 1×3 NumPy array.

Do this by taking partial derivatives in Equation 3.

Problem 9. Write a function `d_loss_d_W0`, which takes four arguments: `variable_dict`, `W1`, `W2`, and `y_observed`. These four arguments should be the same as in Problems 6 and 8.

The function should compute the partial derivative $\frac{\partial \ell_i}{\partial W^0}$, which is defined as follows:

$$\frac{\partial \ell_i}{\partial W^0} = \left[\frac{\partial \ell_i}{\partial w_0^0} \quad \frac{\partial \ell_i}{\partial w_1^0} \quad \frac{\partial \ell_i}{\partial w_2^0} \right] \quad (17)$$

These three partial derivatives should be returned as a 1×3 NumPy array

Do this by taking partial derivatives in Equation 1.

Problem 10. You have now computed the partial derivatives $\frac{\partial \ell_i}{\partial W^0}$, $\frac{\partial \ell_i}{\partial W^1}$, and $\frac{\partial \ell_i}{\partial W^2}$. This is all that you need in order to perform gradient descent and optimize the weight parameters.

In the next three problems, you will implement this gradient computation in PyTorch (it will be much easier!).

Write a class `TorchMLP` (method signatures are given to you in the starter code). This class should be a PyTorch implementation of the MLP defined in this problem set.

Problem 11. Write a function `torch_loss`, which takes two arguments: `y_predicted` and `y_observed`. These are the predicted/observed values of y for a single datapoint. You can assume that they are both PyTorch tensors of length 1.

The function should return the loss for this datapoint.

Problem 12. Write a function `torch_compute_gradient`, which takes three arguments `x`, `y_observed`, and `model`. `x` and `y_observed` are the input/observed values for a single datapoint. Both are PyTorch tensors of length 1. `model` is an instance of the `TorchMLP` class that you wrote in Problem 10.

The function should compute the gradient of the model on the single datapoint. It should then return the model.

Comment

You now have the ability to check your solutions to Problems 1-9 using the PyTorch code that you wrote. Take advantage of that!