LIGN 167: Problem Set 3

November 6, 2018

Collaboration policy: You may collaborate with up to two other students on this problem set. You must write up your own answers to the problems; do not just copy and paste from your collaborators. You must also submit your work individually. If you do not submit a copy of the problem set under your own name, you will not get credit. When you submit your work, you must indicate who you worked with, and what each of your individual contributions were.

Getting started: We will be uploading a file called pset3.py to Piazza. This file will contain some starter code for the problem set (some functions that you should call in your answers), as well as function signatures for your answers. Please use these function signatures for creating your functions.

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:

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

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

$$W^0 = \begin{bmatrix} w_0^0 \\ w_1^0 \\ w_2^0 \end{bmatrix} \tag{2}$$

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

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

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

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

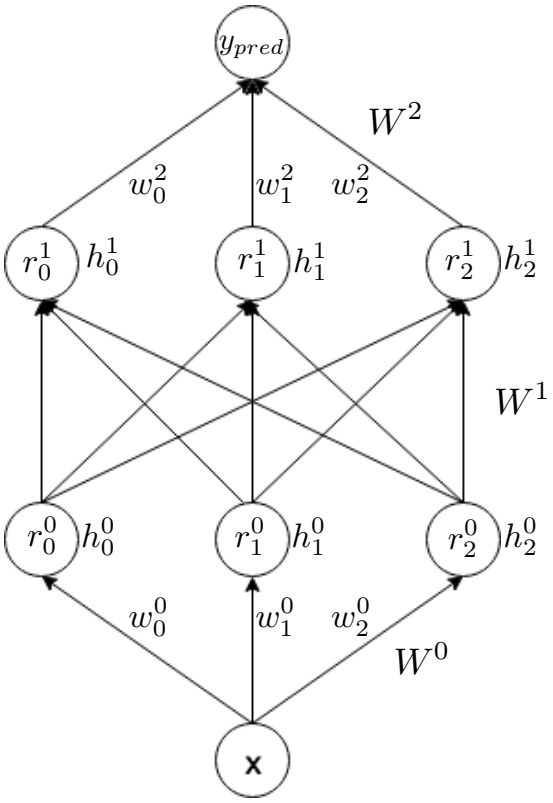

Figure 1: Our multi-layer perceptron.

The matrix in this equation is defined by:

$$W^1 = \begin{bmatrix} w^1_{0,0} & w^1_{0,1} & w^1_{0,2} \\ w^1_{1,0} & w^1_{1,1} & w^1_{1,2} \\ w^1_{2,0} & w^1_{2,1} & w^1_{2,2} \end{bmatrix} \tag{5}$$

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

$$\vec{h}^1 = \begin{bmatrix} h^1_0 \\ h^1_1 \\ h^1_2 \end{bmatrix} = \begin{bmatrix} ReLU(r^1_0) \\ ReLU(r^1_1) \\ ReLU(r^1_2) \end{bmatrix} = ReLU(\vec{r}^1) \tag{6}$$

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

$$y_{pred} = w^2_0 \cdot h^1_0 + w^2_1 \cdot h^1_1 + w^2_2 \cdot h^1_2 = W^2 \cdot \vec{h}^1 \tag{7}$$

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

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

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

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

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]emphbackwards 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 $\dfrac{\partial \ell_i}{\partial y_{pred}}$. 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:

$$\frac{\partial \ell_i}{\partial W^2} = \begin{bmatrix} \frac{\partial \ell_i}{\partial w_0^2} & \frac{\partial \ell_i}{\partial w_1^2} & \frac{\partial \ell_i}{\partial w_2^2} \end{bmatrix} \tag{10}$$

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 $\vec{h}^1$ 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:

$$\frac{\partial \ell_i}{\partial h^1} = \begin{bmatrix} \frac{\partial \ell_i}{\partial h_0^1} & \frac{\partial \ell_i}{\partial h_1^1} & \frac{\partial \ell_i}{\partial h_2^1} \end{bmatrix} \tag{11}$$

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 $\frac{\mathrm{d}ReLU}{\mathrm{d}x}(x)$, where the *ReLU* function is defined by:

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

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:

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

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 $\frac{\partial \ell_i}{\partial W^1}$:

$$\frac{\partial \ell_i}{\partial W^1} = \begin{bmatrix} \frac{\partial \ell_i}{w^1_{0,0}} & \frac{\partial \ell_i}{w^1_{0,1}} & \frac{\partial \ell_i}{w^1_{0,2}} \\ \frac{\partial \ell_i}{w^1_{1,0}} & \frac{\partial \ell_i}{w^1_{1,1}} & \frac{\partial \ell_i}{w^1_{1,2}} \\ \frac{\partial \ell_i}{w^1_{2,0}} & \frac{\partial \ell_i}{w^1_{2,1}} & \frac{\partial \ell_i}{w^1_{2,2}} \end{bmatrix} \tag{14}$$

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.

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:

$$\frac{\partial \ell_i}{\partial h^0} = \begin{bmatrix} \frac{\partial \ell_i}{\partial h^0_0} & \frac{\partial \ell_i}{\partial h^0_1} & \frac{\partial \ell_i}{\partial h^0_2} \end{bmatrix} \tag{15}$$

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:

$$\frac{\partial \ell_i}{\partial r^0} = \begin{bmatrix} \frac{\partial \ell_i}{\partial r^0_0} & \frac{\partial \ell_i}{\partial r^0_1} & \frac{\partial \ell_i}{\partial r^0_2} \end{bmatrix} \tag{16}$$

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:

$$\frac{\partial \ell_i}{\partial W^0} = \begin{bmatrix} \frac{\partial \ell_i}{\partial w^0_0} & \frac{\partial \ell_i}{\partial w^0_1} & \frac{\partial \ell_i}{\partial w^0_2} \end{bmatrix} \tag{17}$$

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

Do this by taking partial derivatives in Equation 1.

Comments on the problems: 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.

We have also included PyTorch code for the model in the starter code. Entirely optional: by slightly extending the starter code, you can compute gradients, and verify your solutions to the problems.