4. Training the Network

Forward propagation is simply the summation of the previous layer's output multiplied by the weight of each wire, while back-propagation works by computing the partial derivatives of the cost function with respect to **every** weight or bias in the network. In back propagation, the network gets better at minimizing the error and predicting the output of the data being used for training by incrementally updating their weights and biases using stochastic gradient descent.

We are trying to estimate a continuous-valued function, thus we will use squared loss as our cost function and an identity function as the output activation function. $f(x)$ is the activation function that is called on the input to our final layer output node, and $\hat{a}$ is the predicted value, while y is the actual value of the input.

$$C = \frac{1}{2}(y - \hat{a})^2 \tag{5.1}$$

$$f(x) = x \tag{5.2}$$

When you're done implementing the function `train` (below and in your local repository), run the script and see if the errors are decreasing. If your errors are all under 0.15 after the last training iteration then you have implemented the neural network training correctly.

You'll notice that the `train` functin inherits from `NeuralNetworkBase` in the codebox below; this is done for grading purposes. In your local code, you implement the function directly in your Neural Network class all in one file. The rest of the code in `NeuralNetworkBase` is the same as in the original `NeuralNetwork` class you have locally.

In this problem, you will see the network weights are initialized to 1. This is a bad setting in practice, but we do so for simplicity and grading here.

You will be working in the file `part2-nn/neural_nets.py` in this problem

Implementing Train

5.0/5.0 points (graded)

Available Functions: You have access to the NumPy python library as `np`, `rectified_linear_unit`, `output_layer_activation`, `rectified_linear_unit_derivative`, and `output_layer_activation_derivative`

Note: Functions `rectified_linear_unit_derivative`, and `output_layer_activation_derivative` can only handle scalar input. You will need to use `np.vectorize` to use them

```
        activated_output = linear(output)
17
18      ### Backpropagation ###
19      # Compute gradients
20      output_layer_error = -(y - activated_output) * linear_derivative(output)
21      hidden_layer_error = np.multiply(ReLU_derivative(hidden_layer_weighted_input), self.hidden_to_output_weights.T) @ outpu
22
23      bias_gradients = hidden_layer_error * 1
24      hidden_to_output_weight_gradients = output_layer_error @ hidden_layer_activation.T
25      input_to_hidden_weight_gradients = hidden_layer_error @ input_values.T
26
27      # Use gradients to adjust weights and biases using gradient descent
28      self.biases = self.biases - self.learning_rate * bias_gradients
29      self.input_to_hidden_weights = self.input_to_hidden_weights - self.learning_rate * input_to_hidden_weight_gradients
30      self.hidden_to_output_weights = self.hidden_to_output_weights - self.learning_rate * hidden_to_output_weight_gradients
31
```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
class NeuralNetwork(NeuralNetworkBase):

    def train(self, x1, x2, y):

        vec_relu = np.vectorize(rectified_linear_unit)
        vec_relu_derivative = np.vectorize(rectified_linear_unit_derivative)

        # Forward propagation
        input_values = np.matrix([[x1],[x2]]) # 2 by 1

        hidden_layer_weighted_input = self.input_to_hidden_weights*input_values  + self.biases #should be 3 by 1
        hidden_layer_activation = vec_relu(hidden_layer_weighted_input) # 3 by 1

        output = self.hidden_to_output_weights * hidden_layer_activation # 1 by 1
        activated_output = output_layer_activation(output) # 1 by 1

        # Compute gradients
        output_layer_error =  (activated_output - y) * output_layer_activation_derivative(output)  # 1 by 1
        hidden_layer_error = np.multiply((np.transpose(self.hidden_to_output_weights) * output_layer_error), vec_relu_derivative(hidden_layer_w
        bias_gradients = hidden_layer_error
        hidden_to_output_weight_gradients = np.transpose(hidden_layer_activation * output_layer_error)# [3 by 1] * [1 by 1] = [3 by 1]
        input_to_hidden_weight_gradients = np.transpose(input_values * np.transpose(hidden_layer_error)) #  = [2 by 1] * [1 by 3] = [2 by 3]

        # Use gradients to adjust weights and biases
        self.biases = self.biases - self.learning_rate * bias_gradients
        self.input_to_hidden_weights = self.input_to_hidden_weights - self.learning_rate * input_to_hidden_weight_gradients
        self.hidden_to_output_weights = self.hidden_to_output_weights - self.learning_rate * hidden_to_output_weight_gradients
```

Test results

See full output

CORRECT

See full output

Submit You have used 4 of 20 attempts

Answers are displayed within the problem

Discussion

Show Discussion

Topic: Unit 3 Neural networks (2.5 weeks):Project 3: Digit recognition (Part 2) / 4. Training the Network