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. $f\left(x\right)$ 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.

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

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

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

```
class NeuralNetwork(NeuralNetworkBase):

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

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

        # Calculate the input and activation of the hidden layer
        hidden_layer_weighted_input = self.input_to_hidden_weights * in
        hidden_layer_activation = np.vectorize(rectified_linear_unit)(h

        output = self.hidden_to_output_weights * hidden_layer_activatio
        activated_output = output_layer_activation(output)  # scalar
```

```
15
```

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
        hidden_layer_activation = vec_relu(hidden_layer_weighted_input) # 3 by

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

        # Compute gradients
        output_layer_error =  (activated_output - y) * output_layer_activation_
        hidden_layer_error = np.multiply((np.transpose(self.hidden_to_output_we

        bias_gradients = hidden_layer_error
        hidden_to_output_weight_gradients = np.transpose(hidden_layer_activatio
        input_to_hidden_weight_gradients = np.transpose(input_values * np.trans

        # 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.lear
        self.hidden_to_output_weights = self.hidden_to_output_weights - self.le
```

Test results

See full output

CORRECT

See full output

Submit You have used 4 of 25 attempts

Answers are displayed within the problem

Discussion

Hide Discussion

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

Add a Post

Show all posts by recent activity

- [Tips from Staff] in "Implementing Train" — 14
  Hi all, In this post, we will try to summarize some typical errors you may incur solving this qu...
  Pinned Staff
- [STAFF] d(C)/d(a) error in my solution — 2
  Hi! Everything else is right in my solution, but I forgot the (-1) in the derivative for the output_l...
- If all nodes start with exactly the same weight, how do the weights ever diverge? — 4
  I'm having a hard time with this exercise, but one thing among many I don't understand is: if ...
- Wrong function of rectified_linear_unit — 5
  The builtin function of rectified_linear_unit is using np.max, which does not make sense. Inst...
- Hints On Training the Network — 2
  Hi, I spent a ridiculous amount of time on this question breaking my brain, so I felt it would b...
- ? [Staff] A question about "bias_gradients" — 4
- Errors not under 0.15 but code correct — 4
  I'm not seeing output layer error drop but my code works right....anyone else seeing this? -1 -...
- ? [Staff] Please assist — 8
  Greetings from COVID-19 Central, Directly in python I get a "Test Passed" but the grader isn't ...
- Need urgent hint!!!1 — 3
  Hi Guys Can someone from staff look at my code for backprop and drop me a hint where i'm ...

[STAFF] HELP WITH GRADER Hi, Staff I 've sent my submition and the awnser is ok. I've read others post, but if I delete in...	1
[STAFF] grader issue Can you please check my submitted response? The grader say that that both test are failing b...	7
Wow this was a pain Anyone else feels the same?	3