



MITx 6.86x

Machine Learning with Python-From Linear Models to Deep Learning[Course](#)[Progress](#)[Dates](#)[Discussion](#)[Resources](#)[Course](#) / [Unit 3. Neural networks \(2.5 weeks\)](#) / [Project 3: Digit recognition \(Part 2\)](#)[< Previous](#)

4. Training the Network

Bookmark this page

Project due Apr 5, 2023 08:59 -03 Completed

Forward propagation is simply the summation of the previous layer's output multiplied by the weights on the 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 cost function by 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 the cost function and an identity function as the output activation function. $f(x)$ is the activation function that we use for the output node to our final layer output node, and \hat{a} is the predicted value, while y is the actual value of the target.

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

$$f(x) = x$$

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

You'll notice that the `train` function inherits from `NeuralNetworkBase` in the codebox for grading purposes. In your local code, you implement the function directly in your `NeuralNetwork` file. The rest of the code in `NeuralNetworkBase` is the same as in the original `NeuralNetwork` file. You can have locally.

In this problem, you will see the network weights are initialized to 1. This is a bad set of initialization, 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: You need to use `output_layer_activation` at least once for the grader to correctly grade your solution.

15

```
hidden_layer_activation = relu_vec(hidden_layer_weighted_input)
```

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

Unanswered

Submit

< Previous

Next >

You have used 7 of 50 attempts



edX

[About](#)[Affiliates](#)[edX for Business](#)[Open edX](#)[Careers](#)[News](#)

Legal

[Terms of Service & Honor Code](#)[Privacy Policy](#)[Accessibility Policy](#)[Trademark Policy](#)[Sitemap](#)[Cookie Policy](#)[Do Not Sell My Personal Information](#)

Connect

[Blog](#)[Contact Us](#)[Help Center](#)[Security](#)[Media Kit](#)