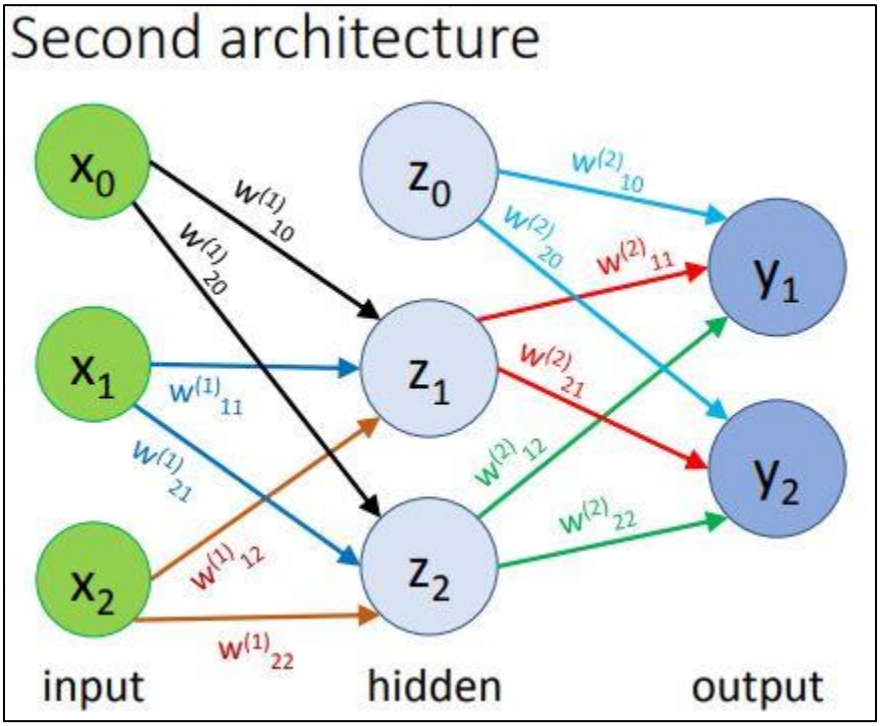
James Hahn
CS1675 Machine Learning
Dr. Adriana Kovashka

Computing weight updates by hand

In order to properly display the process of training a neural network, we will use the second network from recitation and train it on a single sample of x = [1 1], y = [0 0].



First, we need to compute the forward pass. If x_0 is the bias, then the input layer = [1 1 1] and all weights in the network are initialized to 0.05 with a learning rate of 0.3.

The activations of the hidden layer =
z_0 (bias) = 1
$z_1 = w^{(1)}_{10} x_0 + w^{(1)}_{11} x_1 + w^{(1)}_{12} x_2 = 0.05 * 1 + 0.05 * 1 + 0.05 * 1 = 0.15 \Rightarrow \tanh(0.15) = 0.1489$
$z_2 = w^{(1)}_{20} x_0 + w^{(1)}_{21} x_1 + w^{(1)}_{22} x_2 = 0.05 * 1 + 0.05 * 1 + 0.05 * 1 = 0.15 \Rightarrow \tanh(0.15) = 0.1489$

$y_1 = w^{(2)}_{10} z_0 + w^{(2)}_{11} z_1 + w^{(2)}_{12} z_2 = 0.05 * 1 + 0.05 * .1489 + 0.05 * .1489 = 0.0649$
$y_2 = w^{(2)}_{20} z_0 + w^{(2)}_{21} z_1 + w^{(2)}_{22} z_2 = 0.05 * 1 + 0.05 * .1489 + 0.05 * .1489 = 0.0649$

Now, we will begin with the backward pass, or the backpropagation, which is the weight update:

The errors of the output layers are as follows:
$\delta_k^1 = y_1 - t_1 = 0.0649 - 0 = 0.0649$
$\delta_k^2 = y_2 - t_2 = 0.0649 - 0 = 0.0649$

For the hidden layers, the error are as follows:
$\delta_j^1 = (1 - z_j^2)\sum w_{kj}\delta_k = (1 - 0.1489^2) * (0.05 * 0.0649 + 0.05 * 0.0649) = 0.00649$
$\delta_j^2 = (1 - z_j^2)\sum w_{kj}\delta_k = (1 - 0.1489^2) * (0.05 * 0.0649 + 0.05 * 0.0649) = 0.00649$

And finally, for the weight updates:
For the output units, the general weight update formula is $w_{kj} = w_{kj} - \eta * \delta_k * z_j$
$w^{(2)}_{20} = w^{(2)}_{20} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 1 = 0.0481$
$w^{(2)}_{21} = w^{(2)}_{21} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 0.1489 = 0.0497$
$w^{(2)}_{22} = w^{(2)}_{22} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 0.1489 = 0.0497$
$w^{(2)}_{10} = w^{(2)}_{10} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 1 = 0.048053$
$w^{(2)}_{11} = w^{(2)}_{11} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 0.1489 = 0.0497$
$w^{(2)}_{12} = w^{(2)}_{12} - \eta * \delta_k * z_j = 0.05 - 0.03 * 0.0649 * 0.1489 = 0.0497$

For the hidden units, the general weight update formula is $w_{ji} = w_{ji} - \eta * \delta_j * x_i$
$w^{(1)}_{20} = w^{(1)}_{20} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$
$w^{(1)}_{21} = w^{(1)}_{21} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$
$w^{(1)}_{22} = w^{(1)}_{22} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$
$w^{(1)}_{10} = w^{(1)}_{10} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$
$w^{(1)}_{11} = w^{(1)}_{11} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$
$w^{(1)}_{12} = w^{(1)}_{12} - \eta * \delta_j * x_j = 0.05 - 0.03 * 0.00649 * 1 = 0.0498$

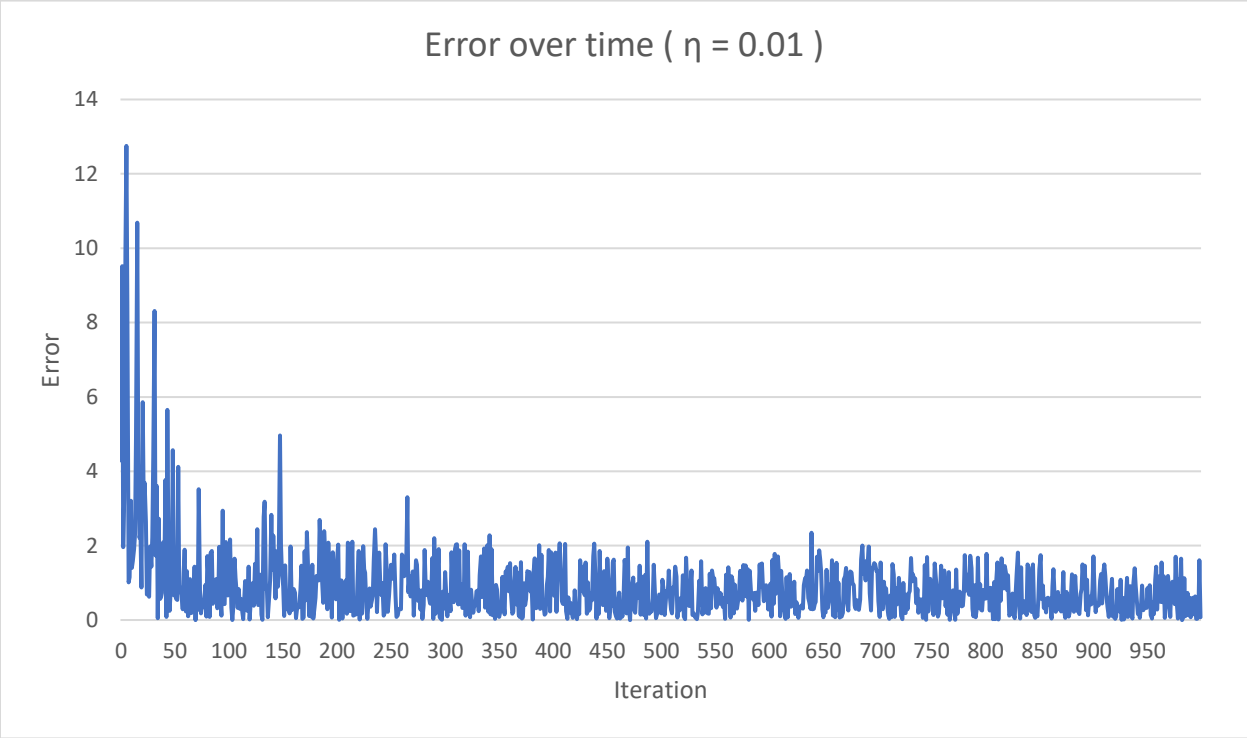
As such, we have performed both a forward and backward pass for our two-layer neural network with 3 input, 3 hidden, and 2 output neurons. Now, if we had more training samples, and especially training samples with less homogeneity in input and output vectors (i.e. not [1 1 1] or [0 0] respectively), the weights will begin to differ from each other more and more.

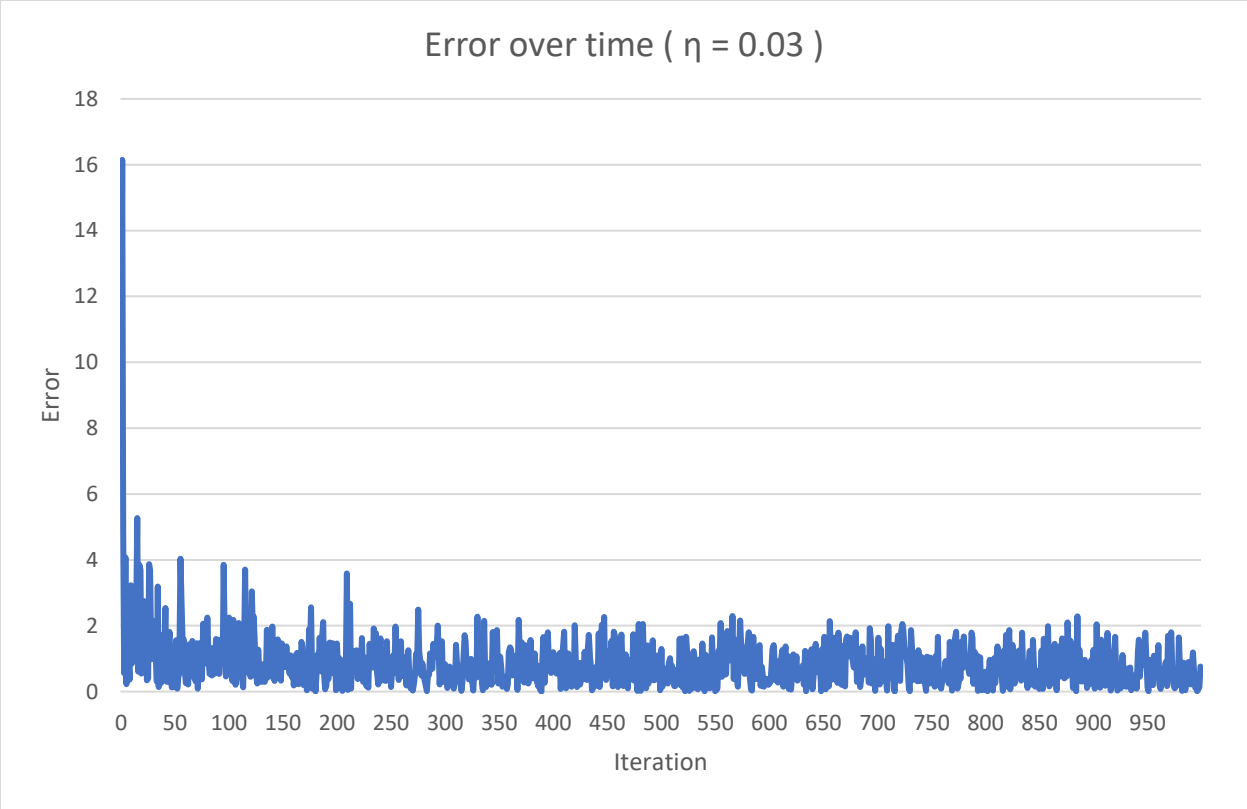
Results / Error

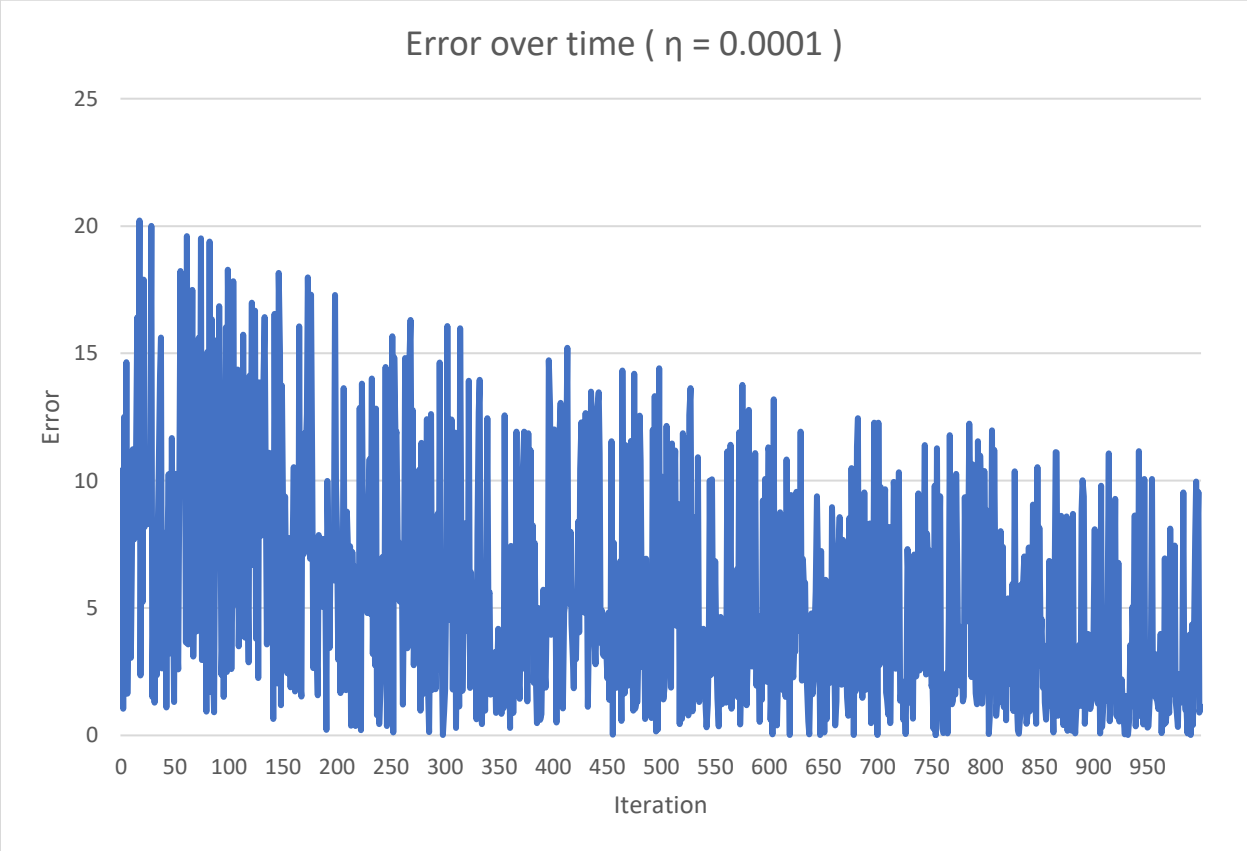
A two-layer neural network was constructed with 11 input features, 30 hidden neurons, and 1 output neuron. A square error loss function was utilized for the neural network. A series of different values of η were tested, such as $\eta = 0.03$, 0.01, and 0.0001. The model was trained with typical backpropagation with a tanh(…) activation function over 1000 iterations. During each iteration, one sample was randomly chosen from the training dataset to perform the weight update. To finally evaluate the performance of the network, the Root Mean Squared Error (RMSE) function was utilized. Analysis was performed on the red wine portion of the Wine Quality dataset (http://archive.ics.uci.edu/ml/datasets/Wine+Quality). Below, there are four different graphs. The first three figures represent the error over the 1000 iterations after updating the weights for a random sample for each learning rate. The fourth figure compares the RMSE across all three learning rates.

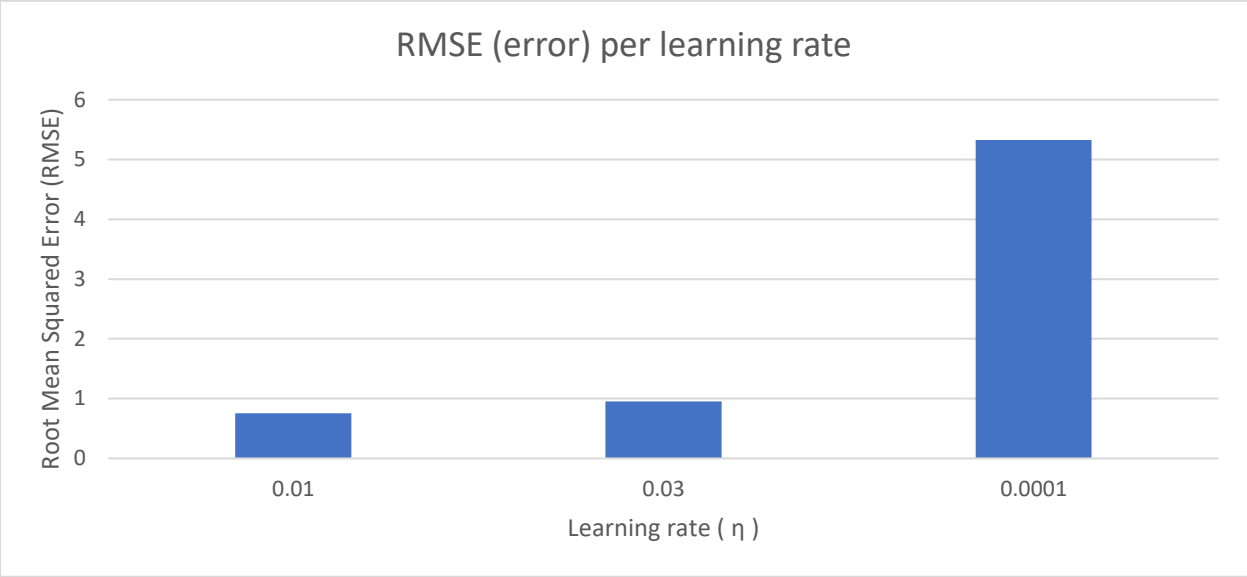
Clearly, as the learning rate increases from 0.0001 to 0.01 to 0.03, the error converges much quicker. This makes perfect sense since this is a convex optimization problem, so we're moving "down the hill" quicker. It is important to note a learning rate of 0.0001, after 1000 iterations, never truly converges. This implies more iterations are required for the weights to converge.

Finally, when comparing learning rates to their overall RMSE, a learning rate of 0.01 possesses the lowest error rate, 0.03 possesses the second highest, and 0.0001 possesses the highest error rate. To an extent, this makes sense. With a learning rate of 0.03, the weights might converge quicker, but the precision of the weights will not be as fine-grained as a learning rate of 0.01 would produce. As such, a slightly highly error is not a surprise and will be expected. In fact, if the neural network was trained on a higher number of iterations, such as 100k, a learning rate of 0.0001 should produce an even lower error rate. However, due to the relatively low number of iterations, a tiny learning rate will not converge quick enough to have reasonable enough weights. Therefore, when the learning rate of 0.0001 possesses the highest error, it is not a surprise. I hypothesize if the number of iterations was increased by a factor of 1k, the final error will be lower compared to the other learning rates.




Error over time (η = 0.03)
Error
Iteration
0
2
4
6
8
10
12
14
16
18
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950


Error over time (η = 0.0001)
Error
Iteration
0
5
10
15
20
25
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950


RMSE (error) per learning rate
Root Mean Squared Error (RMSE)
6
5
4
3
2
1
0
0.01
0.03
0.0001
Learning rate (η)