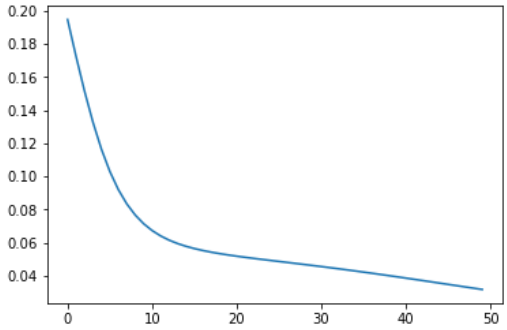
Donna, I did it! I spent a few hours trying to figure out why am I not getting the same results as you when I have done everything according to the book. It turns out the log that you uploaded on the blackboard had a small mistake in it, it was not multiplying learning weight while updating the Weight matrix 1. Anyhow, I am so relieved that I figured it out.
This is what the RMSE looks like after training for 50 epochs: Just the most beautiful thing I have seen today.



I also created a random testing set, which looks like this: trained on the same number of hidden layers, epochs and learning rate.

```
[[-0.3  0.2  0.4 -0.3]
 [ 0.4 -0.2  0.8 -0.3]
 [ 0.6  0.3 -0.7  0. ]]

 (3, 4)
```

```
1 y_test = pd.read_excel(r'Traini
2 print(y_train, "\n\n", y_train.
```

```
[[0.4 0.6 0.5 0.7]
 [0.1 0.3 0.2 0.9]
 [0.7 0.1 0.2 0.1]]

 (3, 4)
```

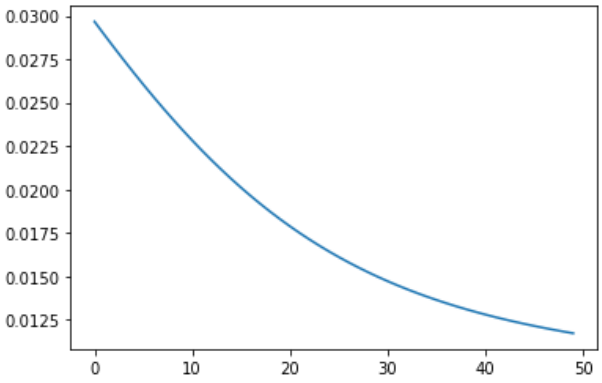


Figure 1 testing

Changing the hidden layers from 5 to 10:

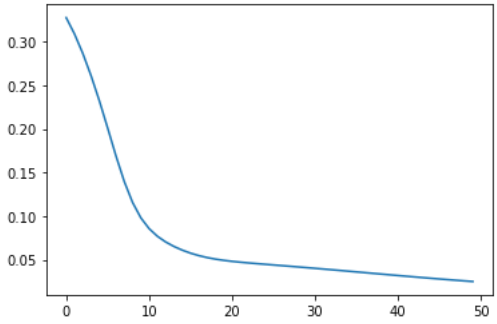


Figure 2 training with 10 hidden layers 50 epochs

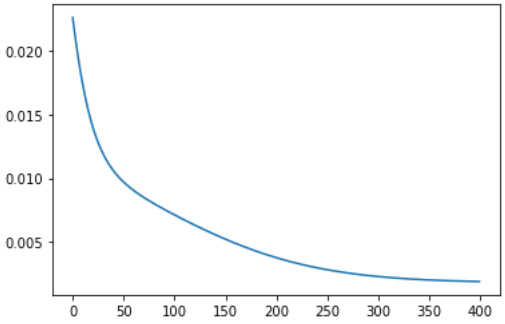


Figure 3 testing with 10 hidden layers

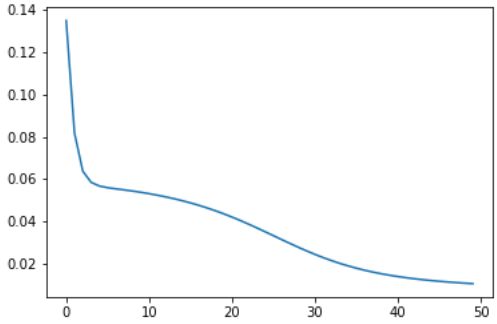


Figure 4 learning rate from 0.5 to 2

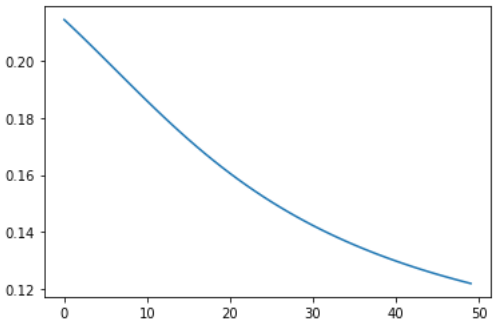


Figure 5 learning rate from 0.5 to 0.01

Changing the learning rate to a high value may skip the optimal solution because it's taking long steps and not really paying attention to small changes, whereas changing the learning rate to a small value takes longer iterations to find the optimal solution, so basically learning late should be chosen in a way it final the optimal solution but doesn't end up taking a lot of resources. Whereas overfitting and underfitting of the model are based on the number of neurons in the hidden layer, a small number of neurons can lead to underfitting, and a large number of neurons can lead to overfitting, so basically should be choosing the number of neurons that generalizes data well. It's all about balance, isn't it?

Thank you for giving us the opportunity to make it right!