CME 216, ME 343 - Winter 2021

Eric Darve, ICME

As we have mentioned before, training a neural network is a difficult.

In some cases, we may get a DNN that fits the training set very well but still have very poor accuracy on the validation set.

This problem is also known as generalization error. That is we do well on the data we are given but fail on unseen data (the "generalization").

This behavior is typically caused when the neural network tends to oscillate wildly as we move away from the training points.

This problem is called "overfitting".

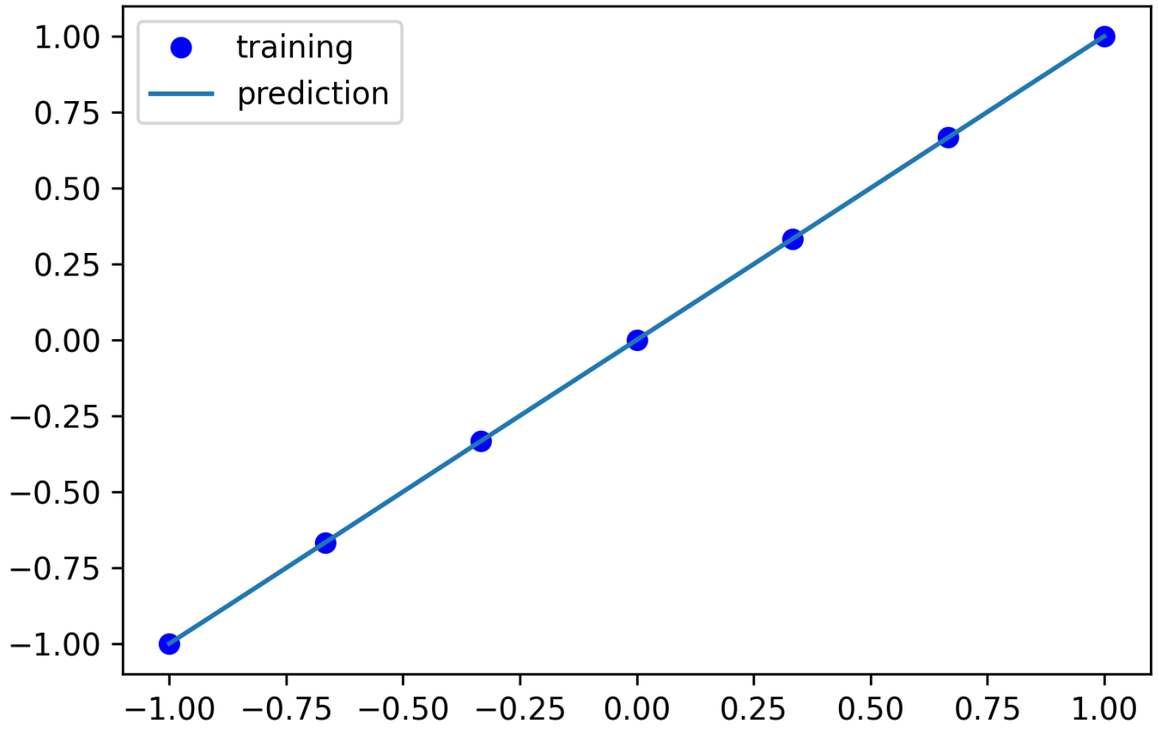
This is similar to the problem of fitting a polynomial of order n to points (x_i, y_i), $1 \leq i \leq n$, where x_i are uniformly distributed.

The polynomial will go through each (x_i, y_i) but will have wild oscillations in between.

DNNs suffer from similar problems. There are different techniques that can mitigate overfitting:

- Early training termination using the validation error
- Controlling the size of the DNN
- Regularization

Let us consider the following example:



For reference, the DNN we use is

- fully connected
- `tanh` activation
- 1 hidden layer with 8 nodes

The exact solution is just the line $y = x$.

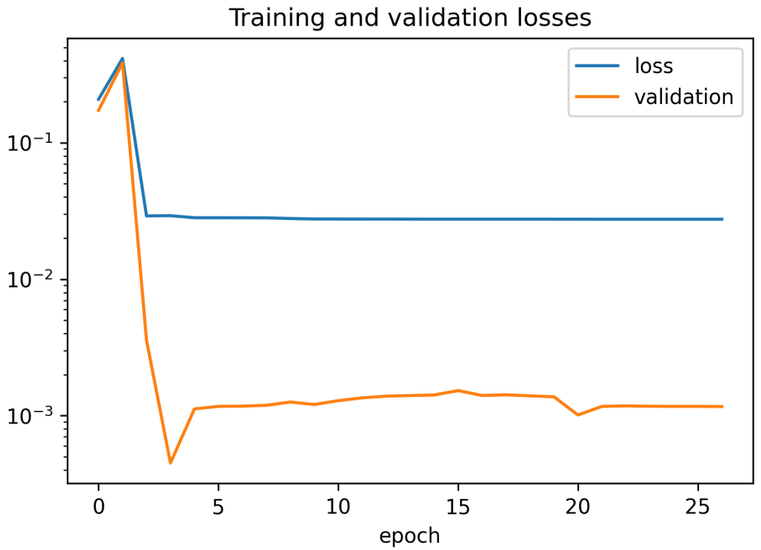
Our answer is very accurate.

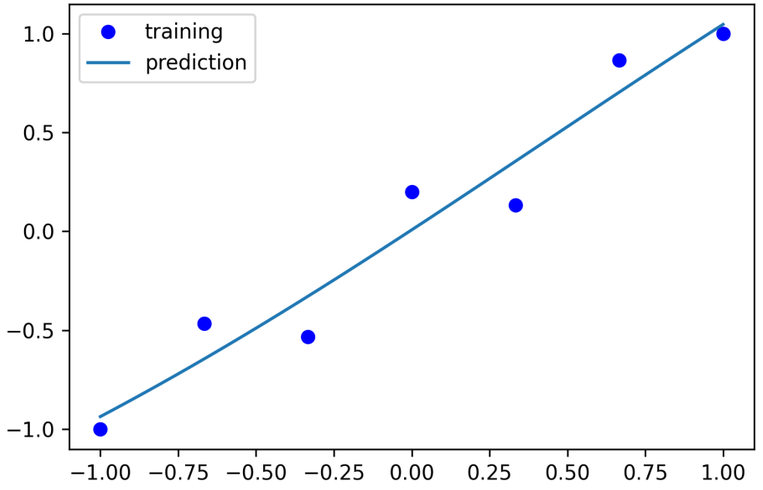
Let's add 10% of noise to the training data.

What do we get?

The DNN is now making a seriously wrong prediction.

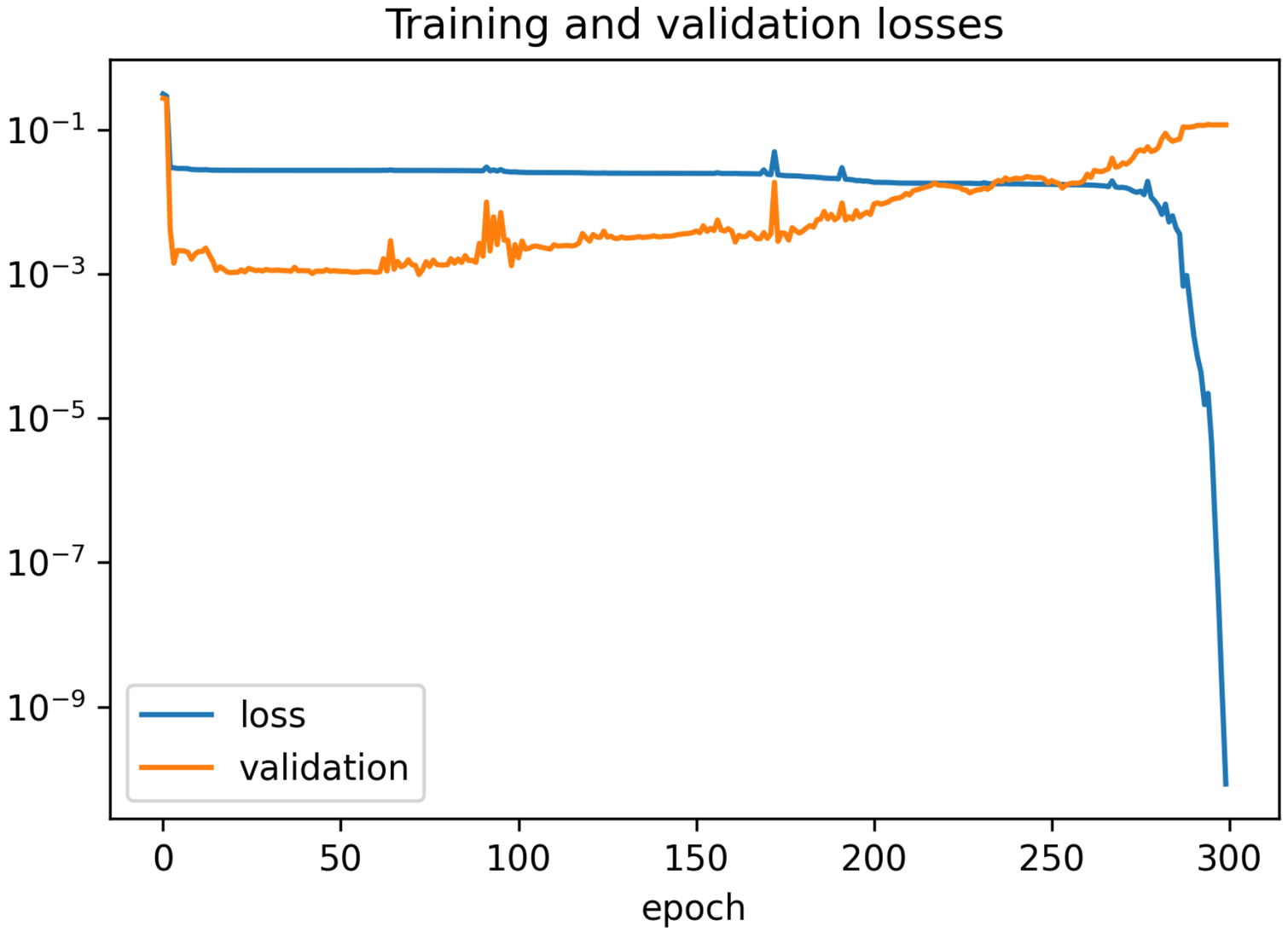
Let's stop the convergence early after 20 steps.





The validation error is small initially.

But as we keep iterating, the training error decreases (the DNN gets closer to the data) but the validation error keeps increasing.


Training and validation losses
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
loss
validation
0
50
100
150
200
250
300
epoch

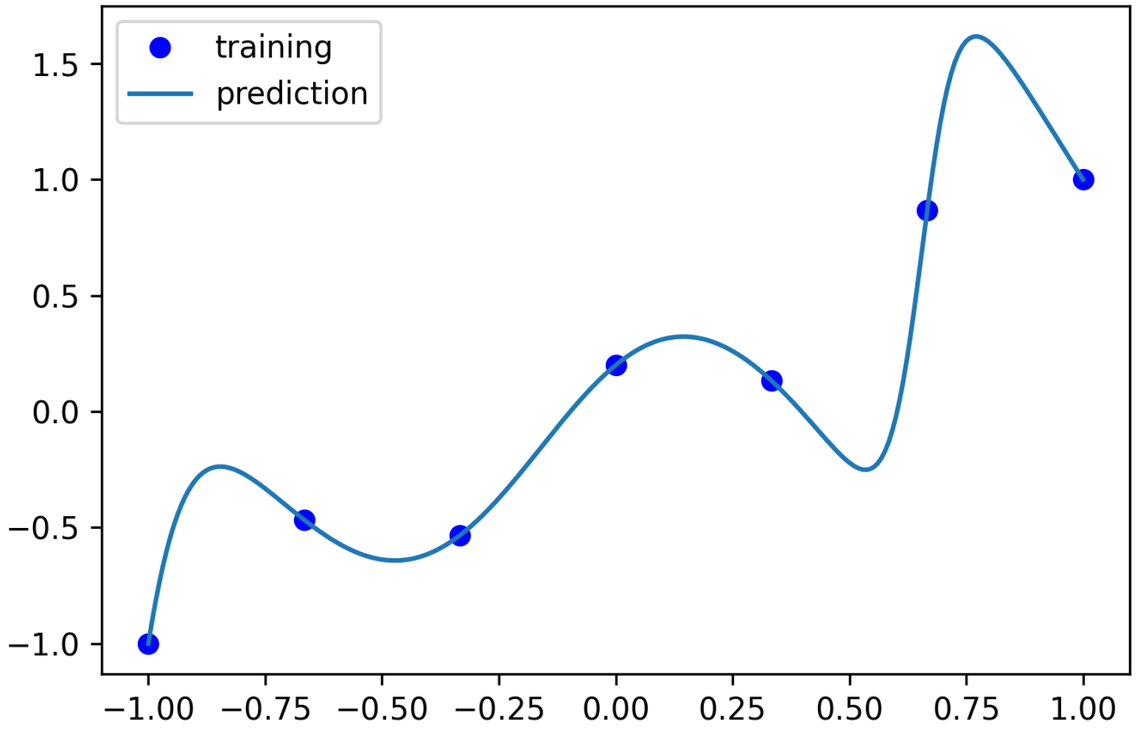
This is because our initial guess for our DNN is quite good in this case.

So our initial choice gives us good accuracy.

But as we train and get closer to the data, our model becomes less accurate.

This is because of the noise we added to the data.

Eventually, we fit the data perfectly:



But our error on the validation set is very high.

This is because the true solution is $y = x$.


Training and validation losses
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
loss
validation
0
50
100
150
200
250
300
epoch

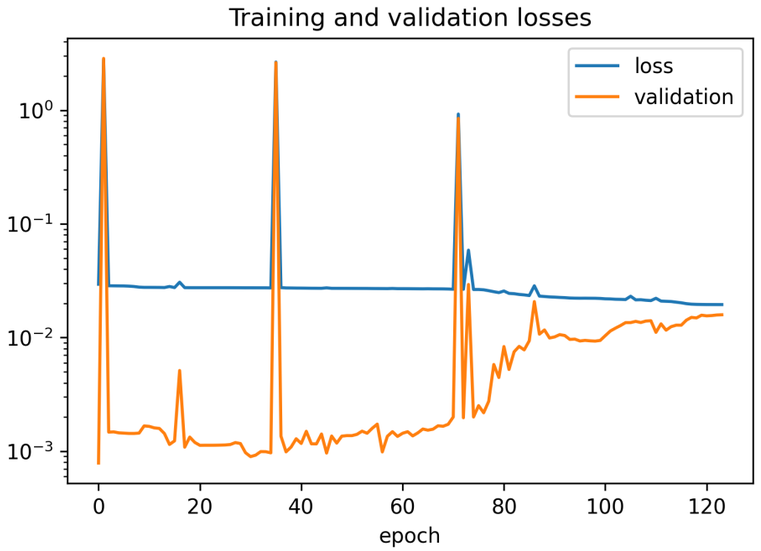
The data cannot be fully trusted in this example.

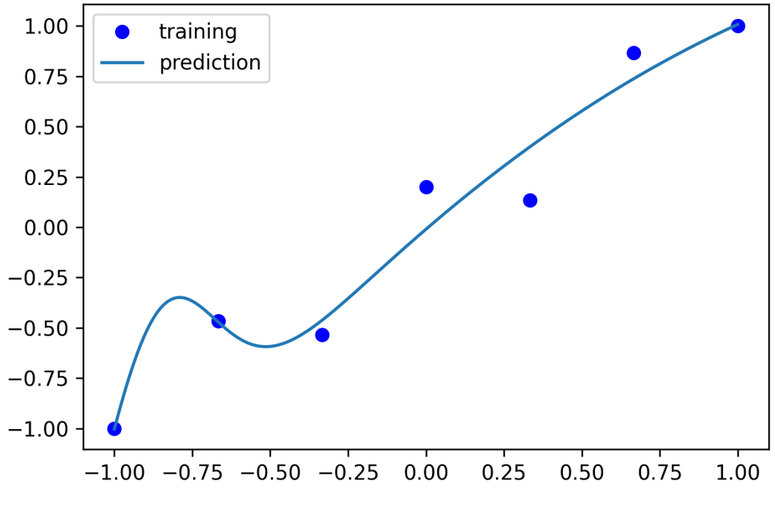
We are overfitting.

But if we look at the validation error, there is a minimum.

The solution is quite accurate at that point.

In this run, the minimum is around 30.





So a simple learning strategy is to train until the validation error starts increasing.

Then we stop the iteration, and use the solution we have obtained at that point.

Solution after 20 steps: