CME 216, ME 343 - Winter 2021

Eric Darve, ICME

The way we initialize the DNN weights is important.

There is a big danger in DNN.

When we stack several layers one after the other, we need to make sure that the outputs at each layer do not keep increasing.

Consider the linear transformation:

$$z_i = \sum_{j=1}^{n} w_{ij} a_j$$

a_j: activation at previous layer; w_{ij}: weights of current layer.

Assume that the activations are initially somewhat random.

For our simple analysis we assume that a_j are independent and identically distributed.

Their mean is 0 and their variance is assumed to be σ.

Assume that $w_{ij} = 1$ for simplicity. We have:

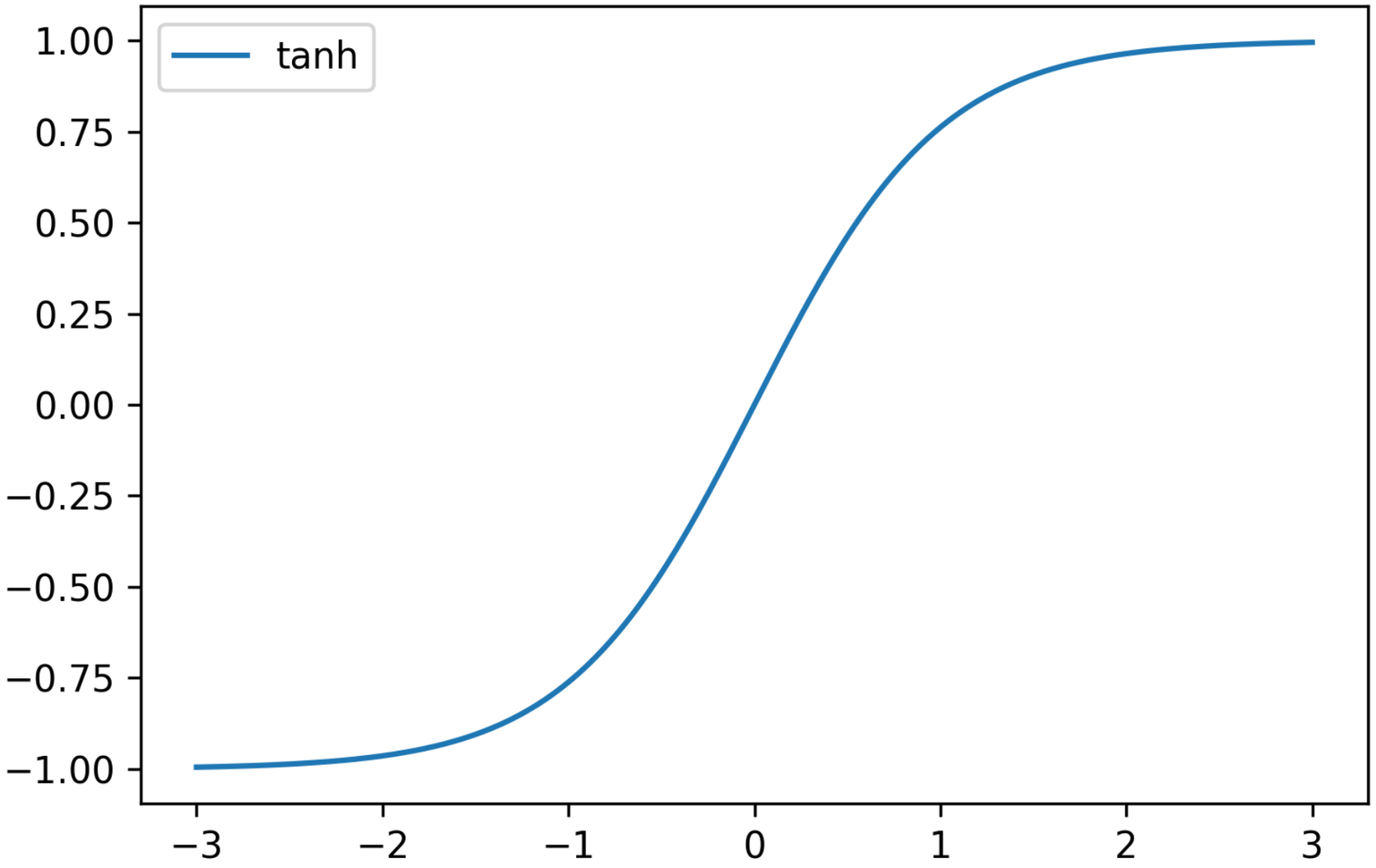
$$z_i = \sum_{j=1}^{n} w_{ij} a_j = \sum_{j=1}^{n} a_j$$

The variance of z_i is $n\sigma^2$.

If we consider the function `tanh` we see that it becomes very flat when the argument is large.

If all weights are of order 1, the output z_i will tend to be large as n increases.

This will push the argument `tanh` to large values, and saturate `tanh`.


tanh
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3

The consequence of this is that training becomes very difficult. The gradient becomes very small.

This is known as the **vanishing gradient problem.**

In addition, instabilities may creep in the model.

So it is important that we make sure that the outputs stay nicely bounded.

This can be done by making sure that the variance of z_j is 1 in our example.

This idea has led to different techniques to initialize the weights of DNNs.

A well-known one is the Glorot-Bengio initialization which considers the number of incoming and outgoing edges in the model (in our case it was n) and defines

$$\mathbf{fan}_{\mathbf{avg}} = \frac{\mathbf{fan}_{\mathbf{in}} + \mathbf{fan}_{\mathbf{out}}}{2}$$

Then the weights are initialized with variance

$$\sigma^2 = \frac{1}{\mathbf{fan}_{\mathbf{avg}}}$$

In our analysis this makes sure that the output of each layer stays of order 1.

Many variants have been proposed for this idea.

See the TF documentation on initializers.

Some of the common choices are for different activation functions:

Initialization	Choice of activation function	Variance
Glorot	default, tanh, logistic, softmax	$1/\mathrm{fan}_{\mathrm{avg}}$
He	ReLU	$2/\mathrm{fan}_{\mathrm{in}}$
LeCun	SELU	$1/\mathrm{fan}_{\mathrm{in}}$

We can test this in our simple example.

Since the solution is $y = x$, we expect the weights to be order 1 (slope 1).

We will use a large network with many coefficients in this example.

We have 4 hidden layers of size 128.

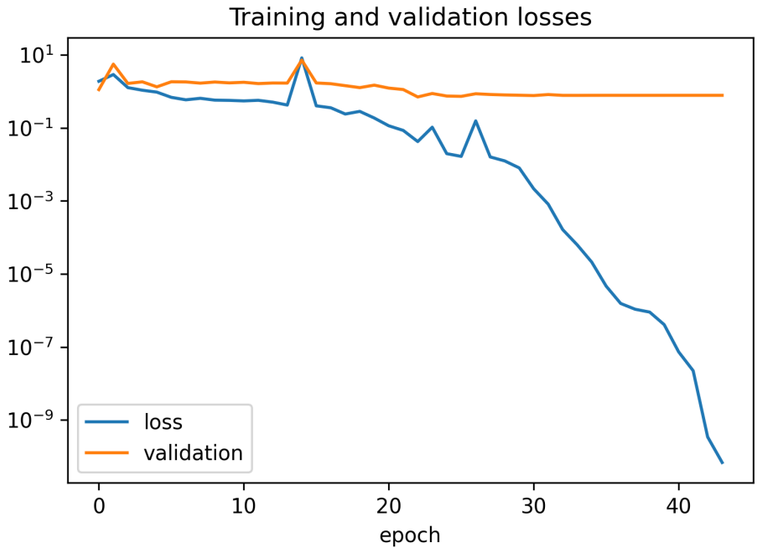
Let's initialize with large weights using a random uniform initialization:

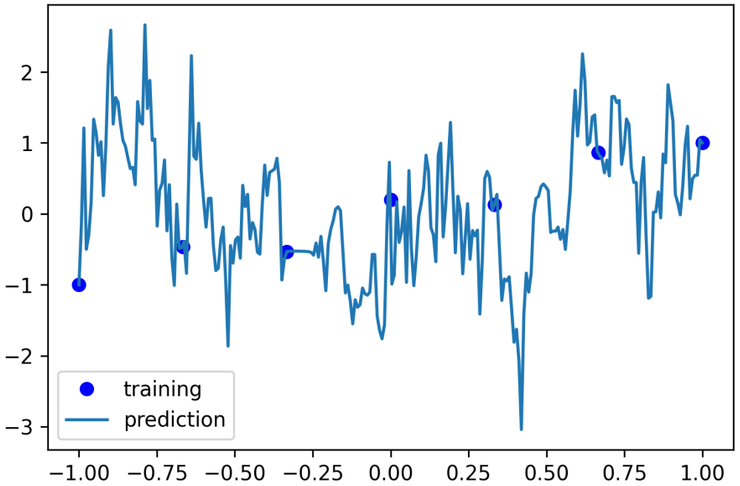
To initialize each layer we use

```
m_val = 10
kernel_initializer=tf.random_uniform_initializer(minval=-m_val, maxval=m_val)
bias_initializer=tf.random_uniform_initializer(minval=-m_val, maxval=m_val)
```

This corresponds to a random uniform initialization using the interval $[-10, 10]$.

The results are absolutely awful.

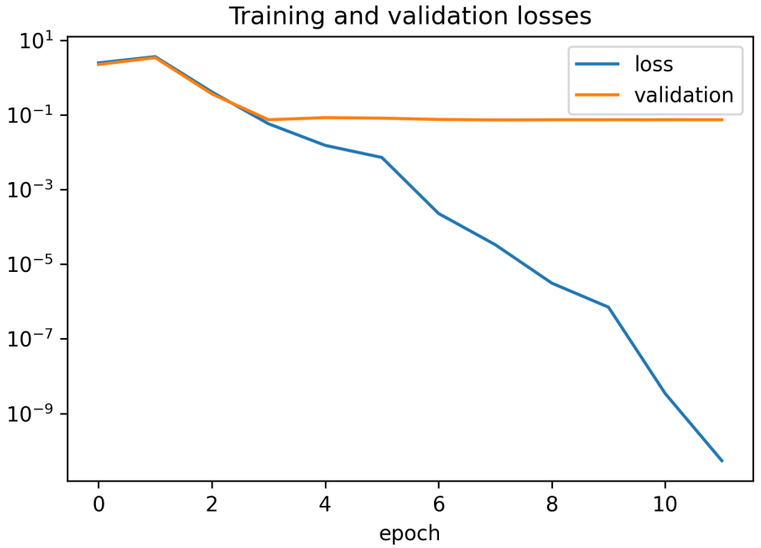

Training and validation losses
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
loss
validation
0
10
20
30
40
epoch

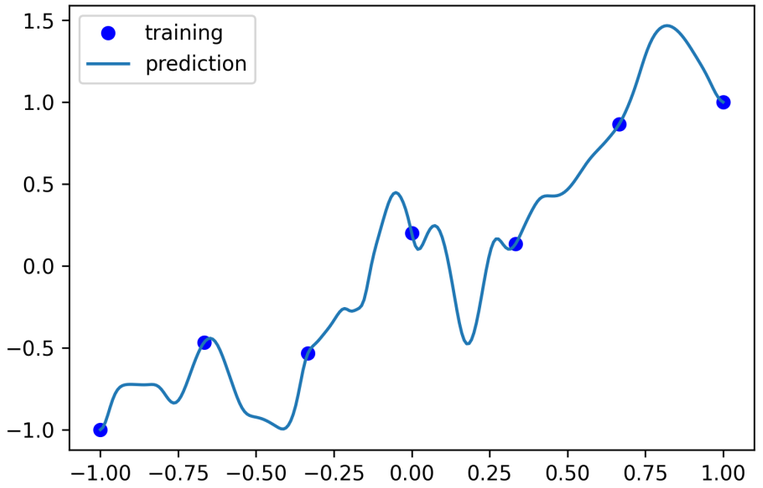

2
1
0
−1
−2
−3
training
prediction
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

This is because we start from a very oscillatory initial guess (large weights).

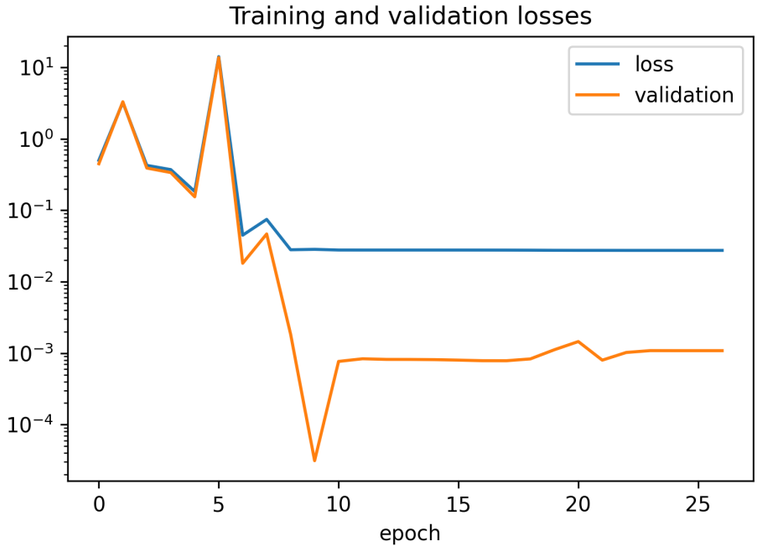
The training brings it closer to the data but because our model is under-constrained (DNN is too complex for this simple task) we fail to converge to anything reasonable.

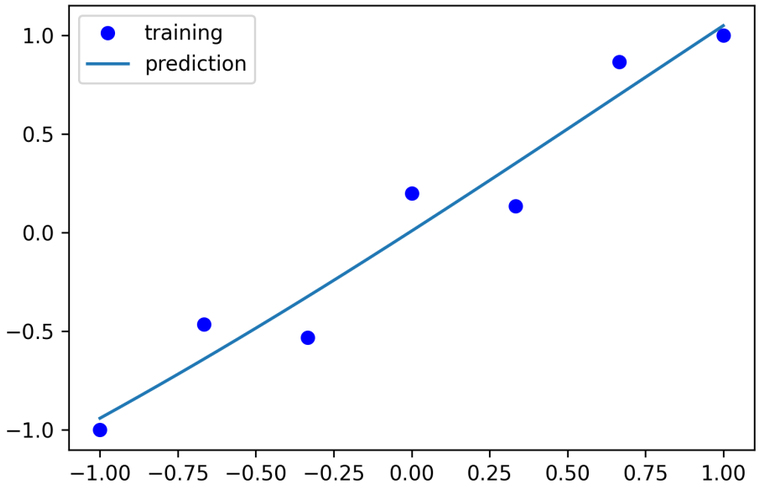
Let's reduce the interval to $[-1, 1]$.





And then to $[-0.1, 0.1]$.





The scheme to initialize the DNN can have a strong regularizing effect.

This is because for overparameterized networks, we may have many local minima.

The initialization scheme will lead the network to different local minima.

If we start with small weights, we will tend to end up with a solution with small weights, which means it will be smoother.

If we start with large weights, then all bets are off. We may get a very oscillatory solution.