neural networks 1: inverted classroom slides

Patrick van der Smagt

how do we find the best weights in an NN?

Q: use an optimiser, e.g., CG, Adam, ...

Those optimisers work best if they have access to $\partial E/\partial w_{ij}$, the gradient of the loss w.r.t. w_{ij}. Back-propagation is the name of the method to compute this gradient.

why do we use nonlinear activation functions?

"They serve as an activation function to imitate the biological activation function of a neuron."
"They adapt to the data during the training"
"to determine the weight w of the data"
"...transform data..."
"**to introduce nonlinearity**"

why do we use nonlinear activation functions?

$$y = W_n \, \phi(W_{n-1} \, \phi(\dots \, \phi(W_0 X) \dots))$$

why do we use nonlinear activation functions?

$$y = W_n \quad (W_{n-1} \quad (\dots \quad (W_0 X) \dots))$$

why do we use nonlinear activation functions?

$$
\begin{aligned}
y &= W_n \quad (W_{n-1} \quad (\dots \quad (W_0 X) \dots)) \\
&= (W_n W_{n-1} \dots W_1 W_0) X
\end{aligned}
$$

why do we use nonlinear activation functions?

$$
\begin{aligned}
y &= W_n \quad (W_{n-1} \quad (\dots \quad (W_0 X) \dots)) \\
&= (W_n W_{n-1} \dots W_1 W_0) X \\
&= W' X
\end{aligned}
$$

why do we use nonlinear activation functions?

"They can be nonlinear while our optimisation problem stays linear in w."

$$y = W_n \, \phi(W_{n-1} \, \phi(\dots \, \phi(W_0 X) \dots))$$

why neural networks over linear regression?

It's all about basis functions. How do you choose them?

polynomial:

$$\sum_i x^i$$

sigmoid:

$$\frac{1}{1+\exp(-cx)}$$

$$\tanh(cx)$$

Gaussian:

$$\exp(-cx^2)$$

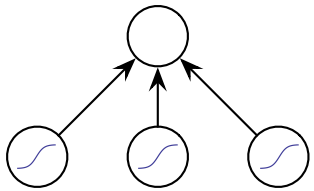
RELU, softmax:

$$\max(0, x)$$

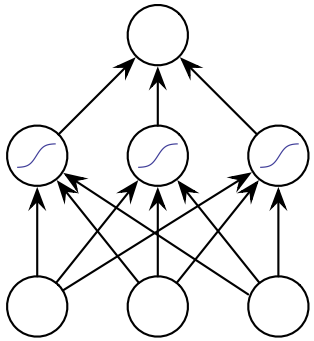
$$\ln(1+e^x)$$

what is "deep" about deep neural networks?

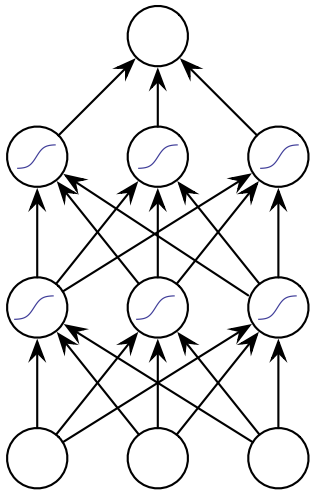
linear regression



neural network with 1 hidden layer
don't say: NN with 2 layers



neural network with 2 hidden layers
don't say: NN with 3 layers



what is "deep" about deep neural networks?

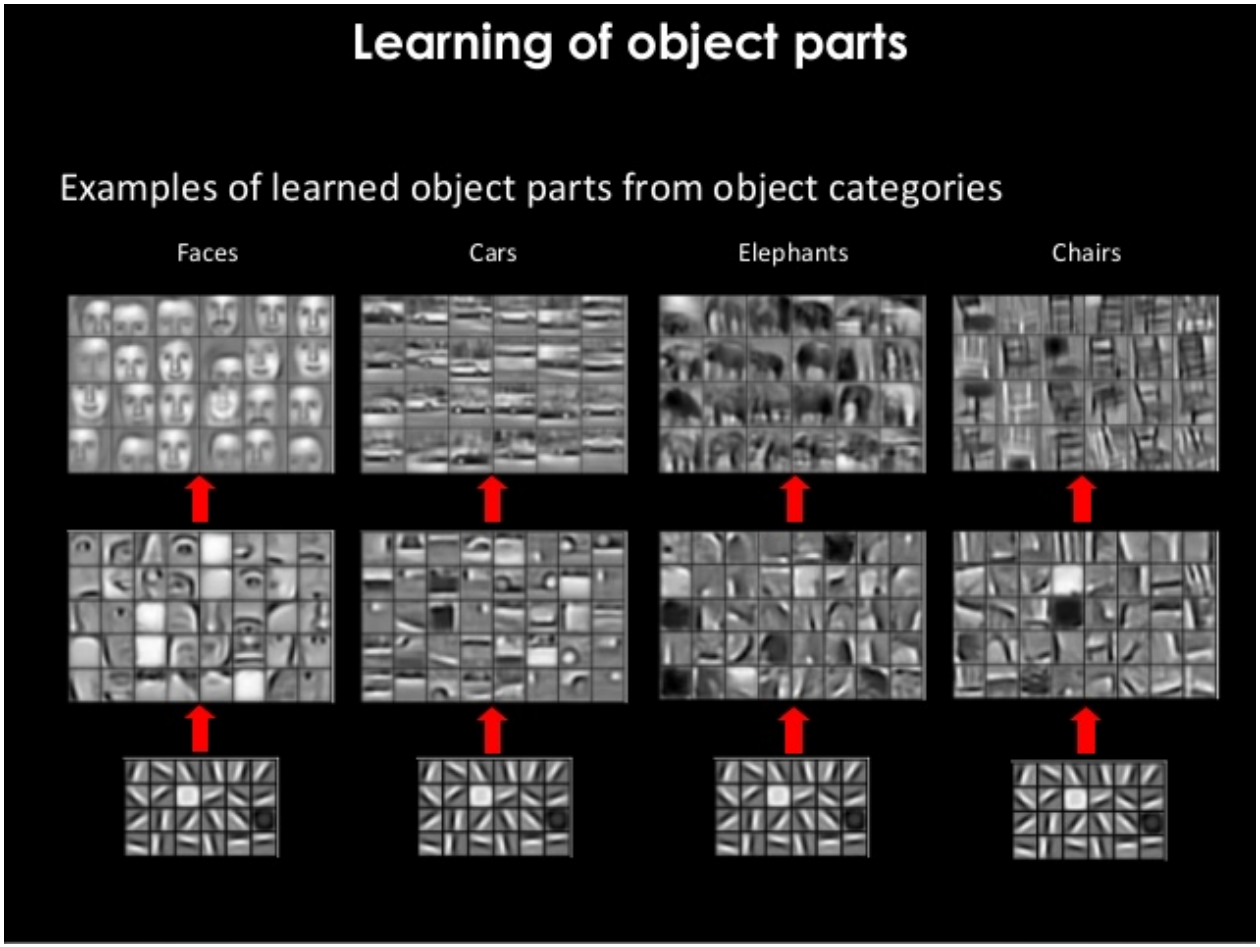
"sounds fancy": a typical Machine Learning disease

Some papers use very deep NN, with hundreds of hidden layers (example: Microsoft winning ImageNet in Dec. 2015 with 152 hidden layers: `https://arxiv.org/abs/1512.03385`).

why do deep neural networks work better?

one learns "features" of "features". This allows for *better generalisation*.

A "wide" network tends to memorise data.



but remember the vanishing gradient

It is usually true that

$$\partial E(\boldsymbol{w})/\partial w_{ij}^{(H)} \gg \partial E(\boldsymbol{w})/\partial w_{ij}^{(H-1)}$$

i.e., the lower you get in the network, the more the gradient vanishes.

After all,

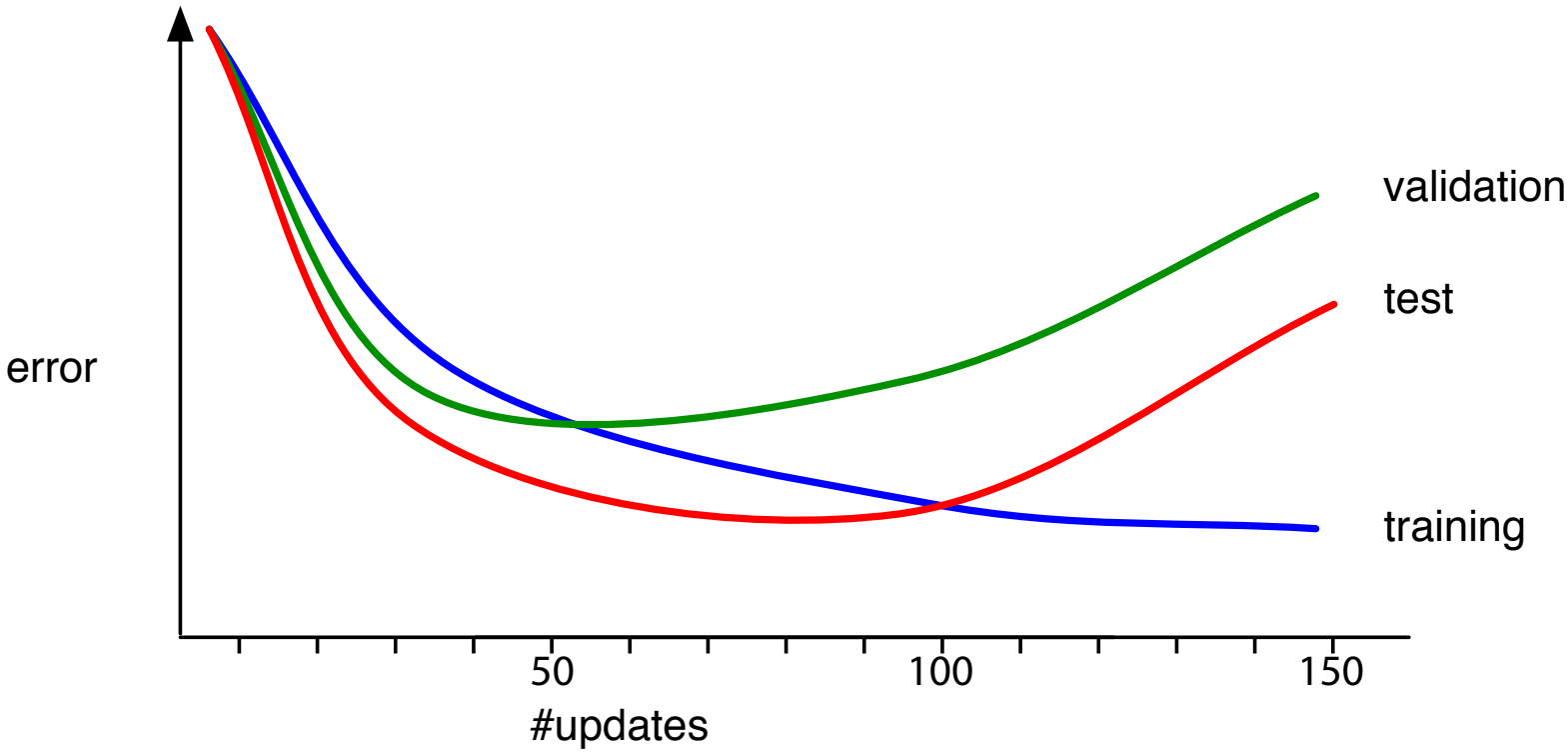
$$\frac{\partial E(\boldsymbol{w})}{\partial w_{H-1,i,j}} = \delta_{H-1,j} x_i = \sum_l \underbrace{\delta_{H,l}}_{\text{small}} \times \underbrace{w_{Hlk} x_i}_{\text{small}} = \text{smaller!}$$

Difference between Shifting and Scaling Inputs

It's all about weight initialisation to keep $w^T x$ at reasonable values.

See `scaling.ipynb`

early stopping: where do we stop training?



goal: find the best set of *hyperparameters* for your model

early stopping: where do we stop training?

training set: $\approx$ 60–70% of the data, *randomly selected*
validation set: $\approx$ 30–20% of the data, *randomly selected*
test set: $\approx$ 20–10% of the data, *randomly selected*

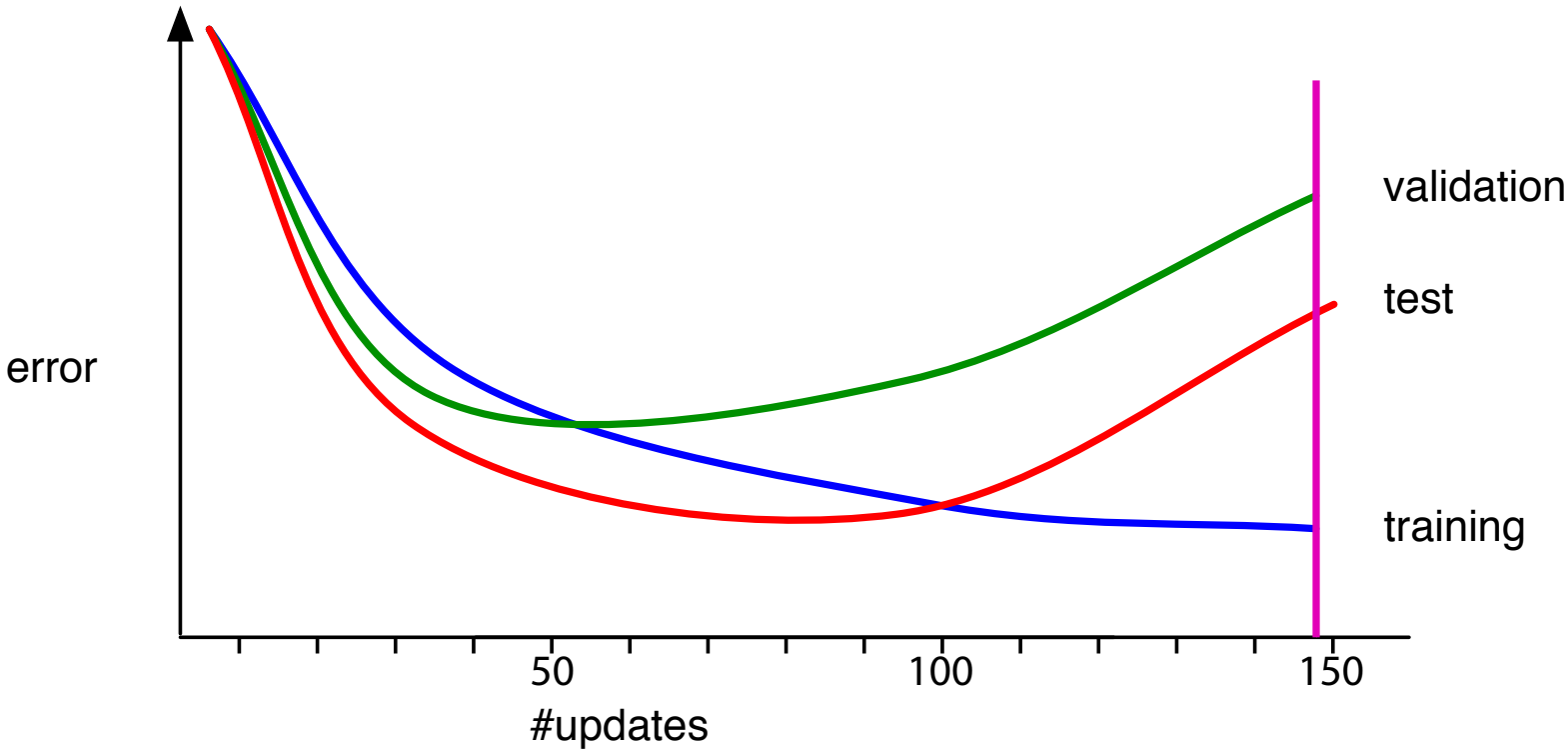
randomly selecting data is hard: often subsequent data points are not iid

too few data $\rightarrow$ leave-one-out-cross-validation compensates by repeated computation of the result for different data set permutations

but **never ever** report your training error as your result accuracy

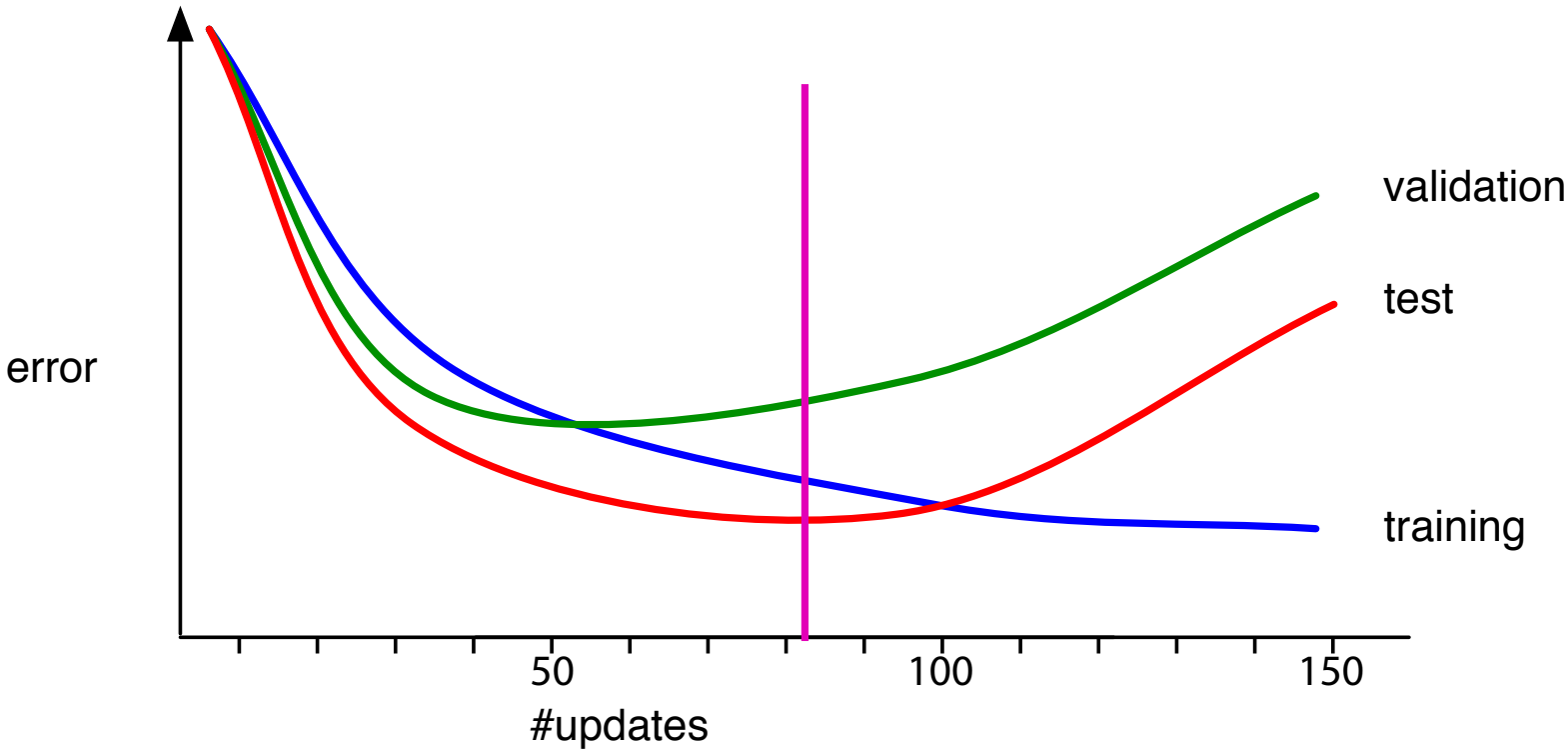
and please don't report your validation error as your result accuracy

early stopping: where do we stop training?



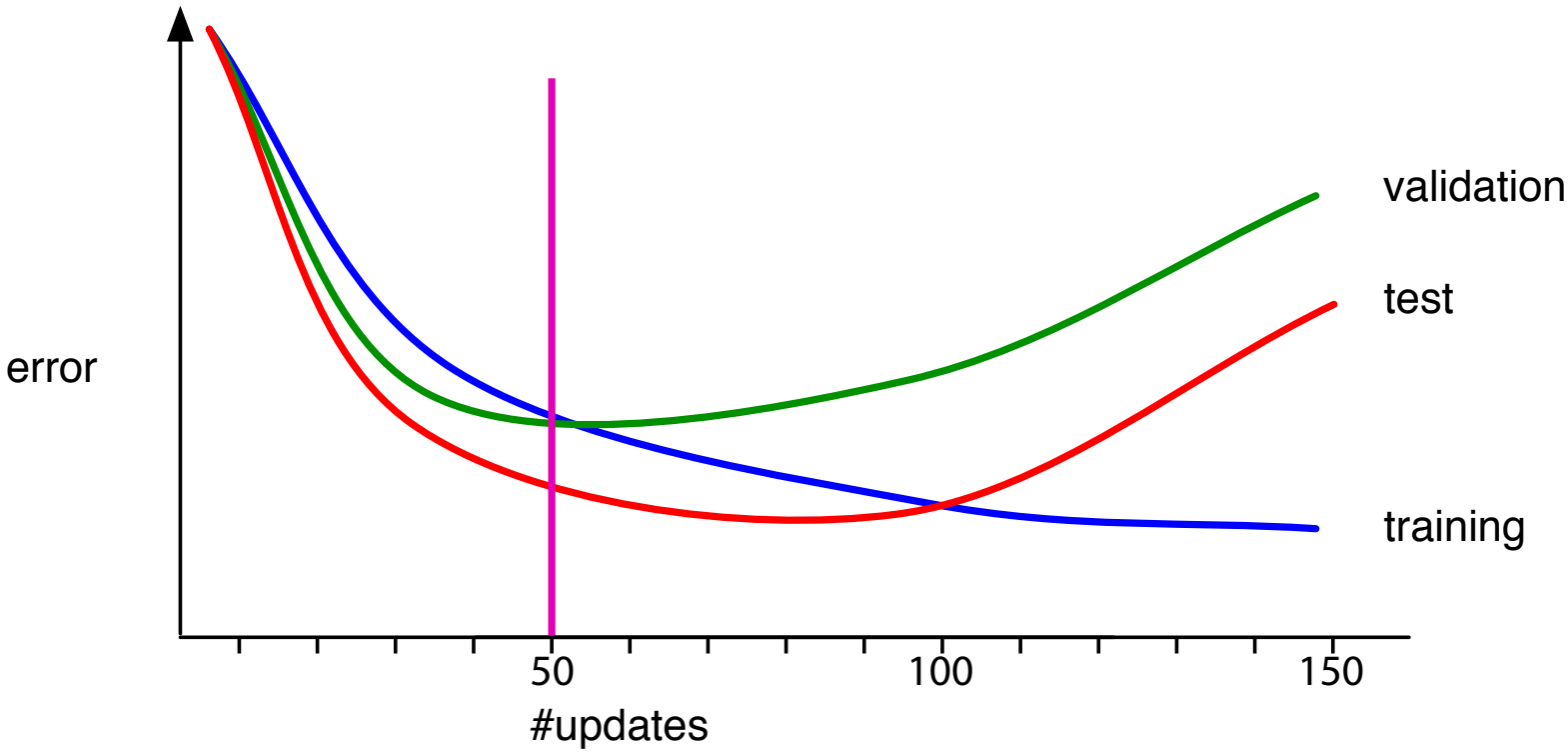
at the lowest training set loss $\Longrightarrow$ bad generalisation

early stopping: where do we stop training?



at the lowest test set loss $\Longrightarrow$ bad practice

early stopping: where do we stop training?



at the lowest validation set loss $\Longrightarrow$ then report your error on the test set

stochastic vs. batch training

We'd prefer to update our parameters after each data point (on-line learning). But:

- inefficient computation (think GPU)
- no tractable way of computing a stable gradient

Full batch learning:

- often does not fit on your GPU
- leaves out desired stochasticity

We therefore usually use mini-batches of size $\approx$ 100–1000