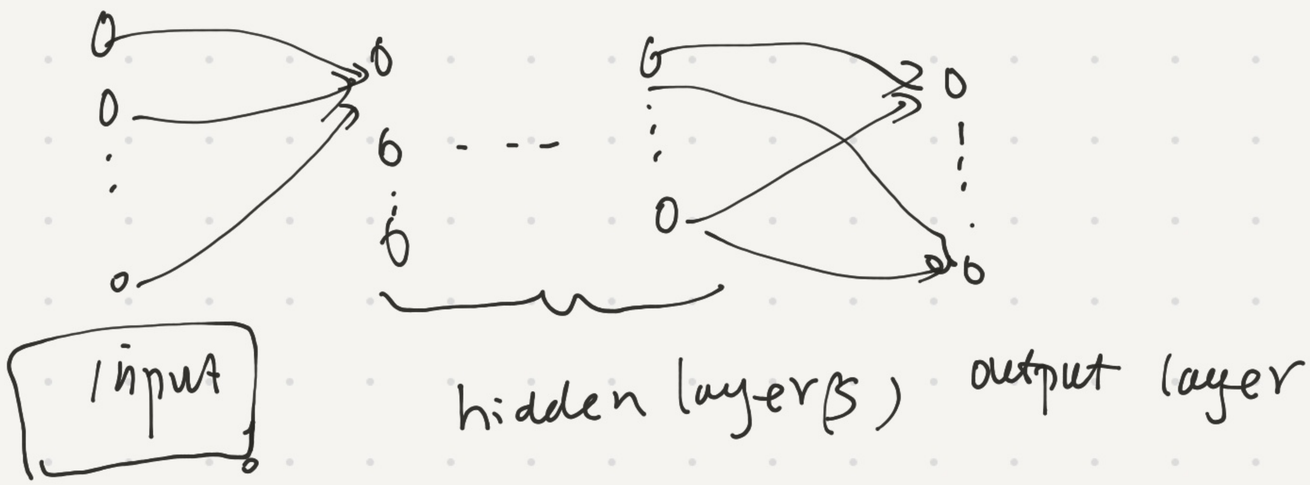
Lectures 16 & 17 | Tips for NN design



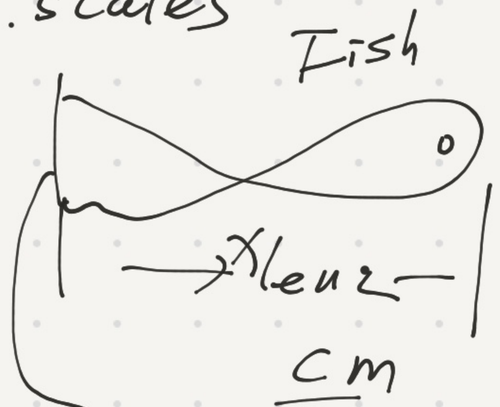
① Scaling input vectors.

$x = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}$ → features may have different scales

feature vectors

$x_{len} \in [0, 200]$ cm.

$x_{mass} \in [0, 40]$ kg



We can use standardization tech.

$$x'_{len} = \frac{x_{len} - mean(x_{len})}{std(x_{len})} \sim \underbrace{N(0, 1)}_{\text{normal distribution}} \quad \begin{matrix} \text{mean} \\ 0 \end{matrix} \quad \begin{matrix} \text{std} \\ 1 \end{matrix}$$

$$x'_{mass} = \frac{x_{mass} - mean(x_{mass})}{std(x_{len})} \sim N(0, 1)$$

Normalization → covert values to $[0, 1]$

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \in [0, 1].$$

x: input feature.

x': output feature.

② Determin the # hidden nodes.

hidden nodes determins # of parameters.

---- the complexity of the model.

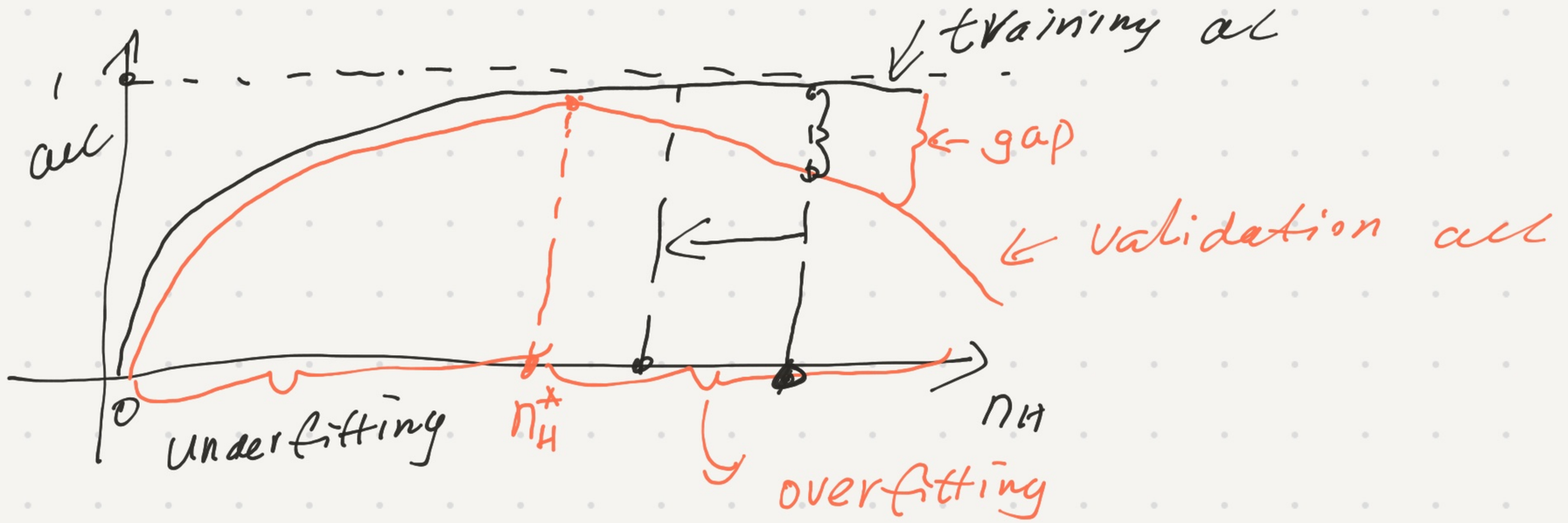
3-layer NN: $\underbrace{(n_{in}}_{\text{input nodes}} \times n_H) + (n_H \times \underbrace{n_{out}}_{\text{output nodes}})$: Parameters.

$$= n_H \times (n_{in} + n_{out})$$

How to choose n_H:

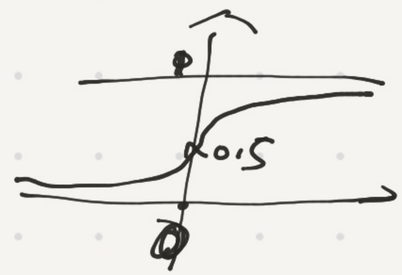
- small-scall NN:
  - $n_H <$ total # of training samples
  - choose an n_H to make total # of parameters $\approx n/10$ (↳ training samples)
- large-scale NN.

start from large n_H, then 'decay' n_H



③ Activation functions

- sigmoid / logistic function: $sigm(x) = \frac{1}{1+e^{-x}}$

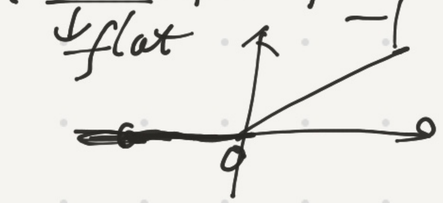


most popular act before 2011.

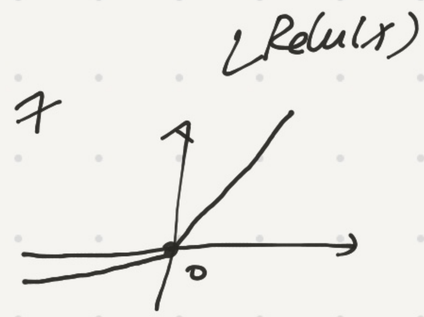
- hyperbolic tan. $tan(x) = \frac{e^{2x}-1}{e^{2x}+1}$

Deep neural Networks.

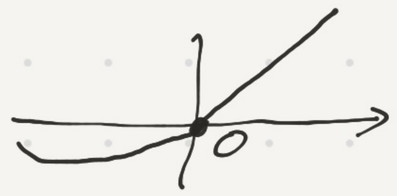
- Relu (2011) $Relu(x) = \max\{0, x\}$



- Leaky Relu (2014) $LRelu(x) = \begin{cases} x & x>0 \\ 0.01x & otherwise \end{cases}$



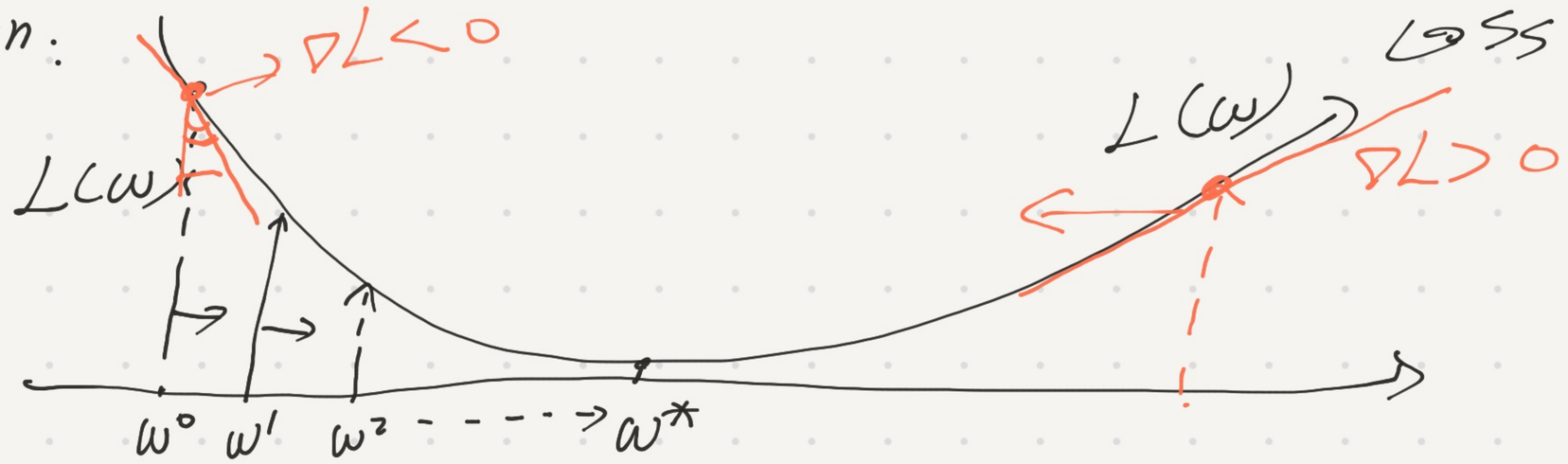
- $swish(x) = x \cdot sigm(x)$
- $Elu(x) = \begin{cases} x & x>0 \\ a(e^x-1) \end{cases}$



④ Batch and epochs

Optimization:

$w^* = \arg\min_w L(w)$



Gradient descent approach: $w^0 \to w^1 \to w^2 \dots \to w^k = w^*$

w^0: initial parameter(s): generate randomly.

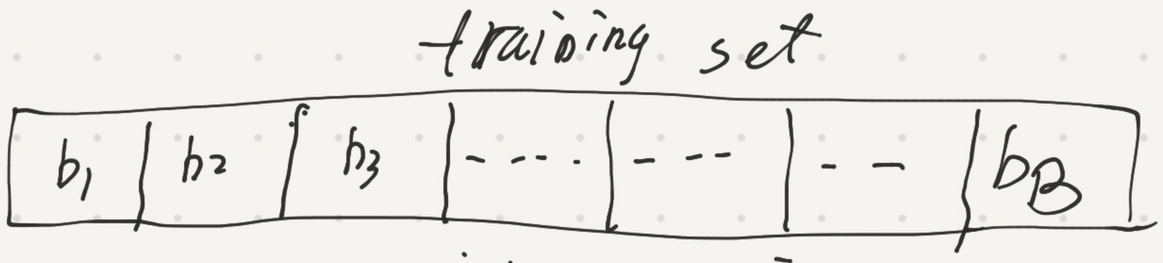
$w^{i+1} = w^i + \varepsilon \cdot (-\nabla_{w^i} L)$

w^i: previous parameter; ε: learning rate: 0.0001; $\nabla_{w^i} L$: gradient of L over w^i

$w^1 = w^0 + \varepsilon \cdot (-\nabla_{w^0} L)$ → can make sure that we are approaching w^*

$\nabla_{w^i} L$ → will involve all training data → expensive and infeasible.

mini-batch optimization approach.



split the training set into B mini-batches; each iteration, only one Batch is involved in $(\nabla_w L)$

For epoch in # of max epochs:
    shuffle the training set; prepare the mini batches
    For i to B:
        $w^{i+1} = w^i + \nabla_{w^i}(L)$