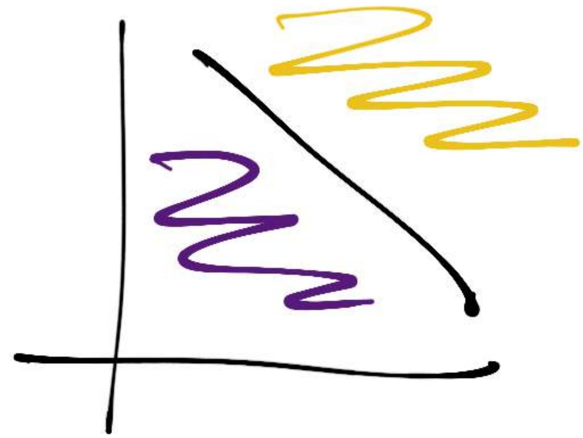
Advanced Machine Learning, 20 Aug 2019

Feedforward Neural Networks

Perceptron model

Linear separability



$$\bar{x} \longrightarrow y$$

Introduce nonlinearity

$$\bar{x} \longrightarrow \phi(\bar{x}) \longrightarrow y$$

$x \longrightarrow \phi(x)$ Geometric transformation

↳ Use "kernel trick"

Find K s.t. $K(x,y) = \phi(x) \cdot \phi(y)$

How to discover ϕ? (Or, equivalently, K)?

Brute Force

Very high dimension (infinite)

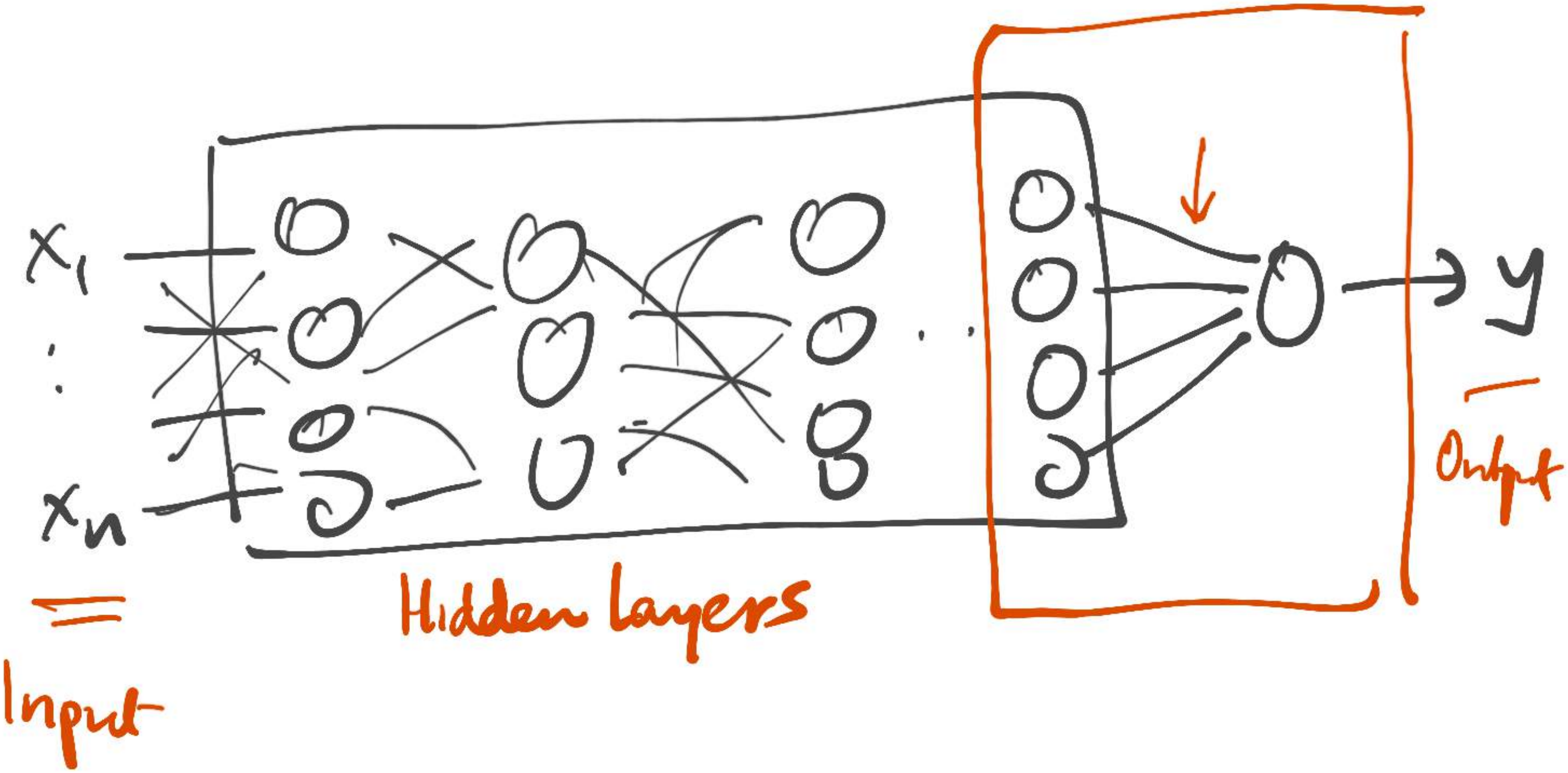
RBF (radial basis)

Hand craft

by experience

Feedforward NN - network of perceptrons

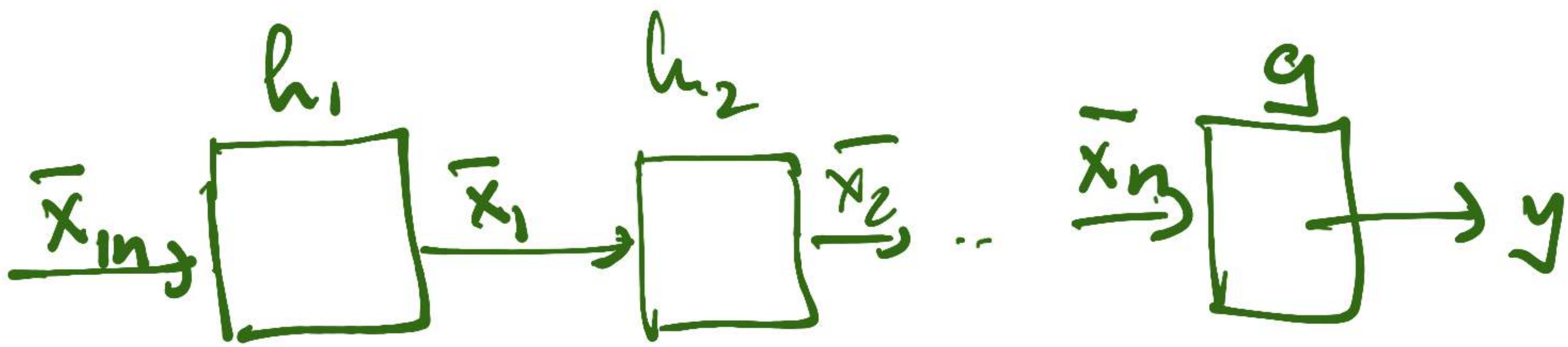
Directed acyclic graph



Hidden layers compute $\Phi(x)$

NN → learn ϕ from training data

Weights of edges in the hidden layers



$y = w^T x + b$

Cascaded computation

$g(x$

$$g(h_n(\ \dots\ h_2(h_1(\bar{x}_{in}))))$$

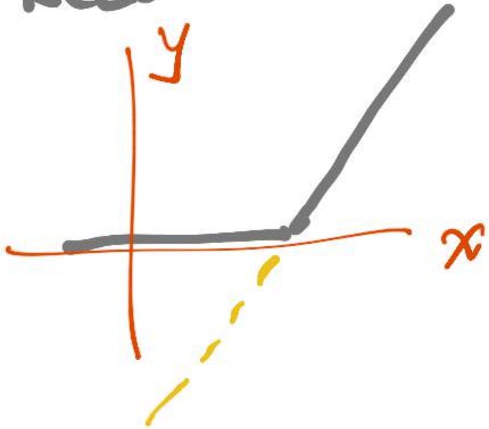
Composition of linear fns is a linear fn

Need to introduce non-linearity (in the hidden layers)

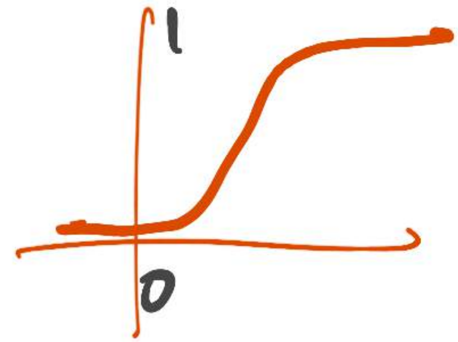
Rectified Linear Unit (RELU)

$$\max(0, \underbrace{w^T x + c}_{g(x)}) + b$$

RELU



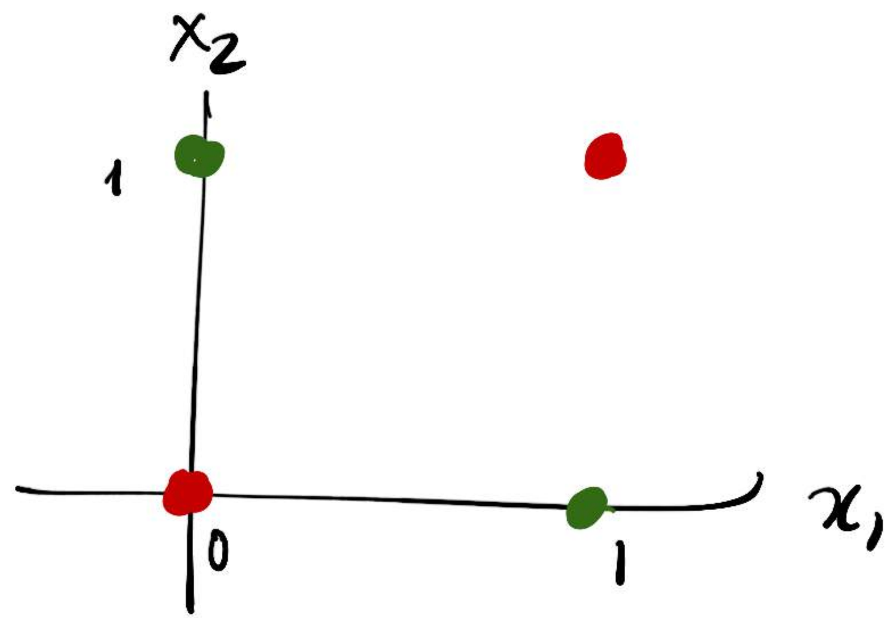
Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$



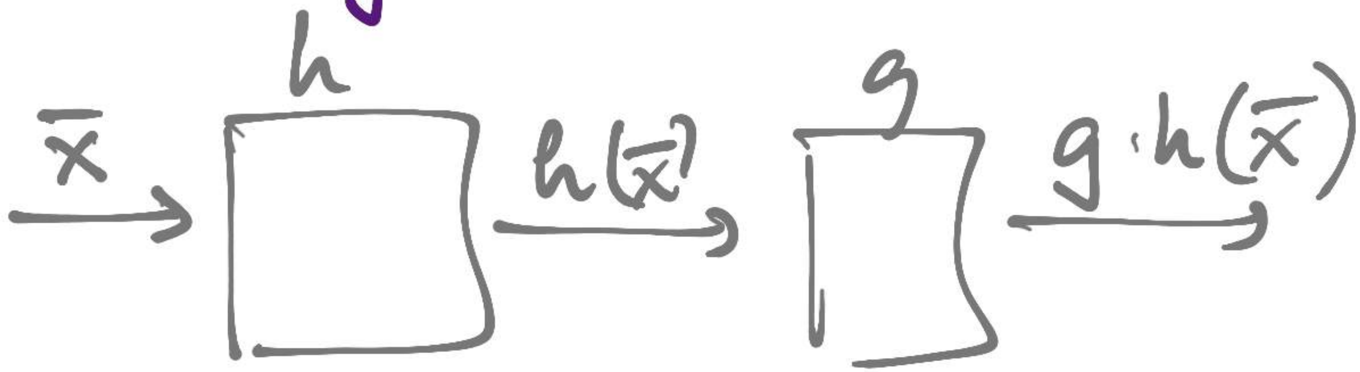
$z = w^T x + c$

Classical example $\text{XOR}(x_1, x_2)$

$x_1, x_2 \in \{0,1\}$



One hidden layer



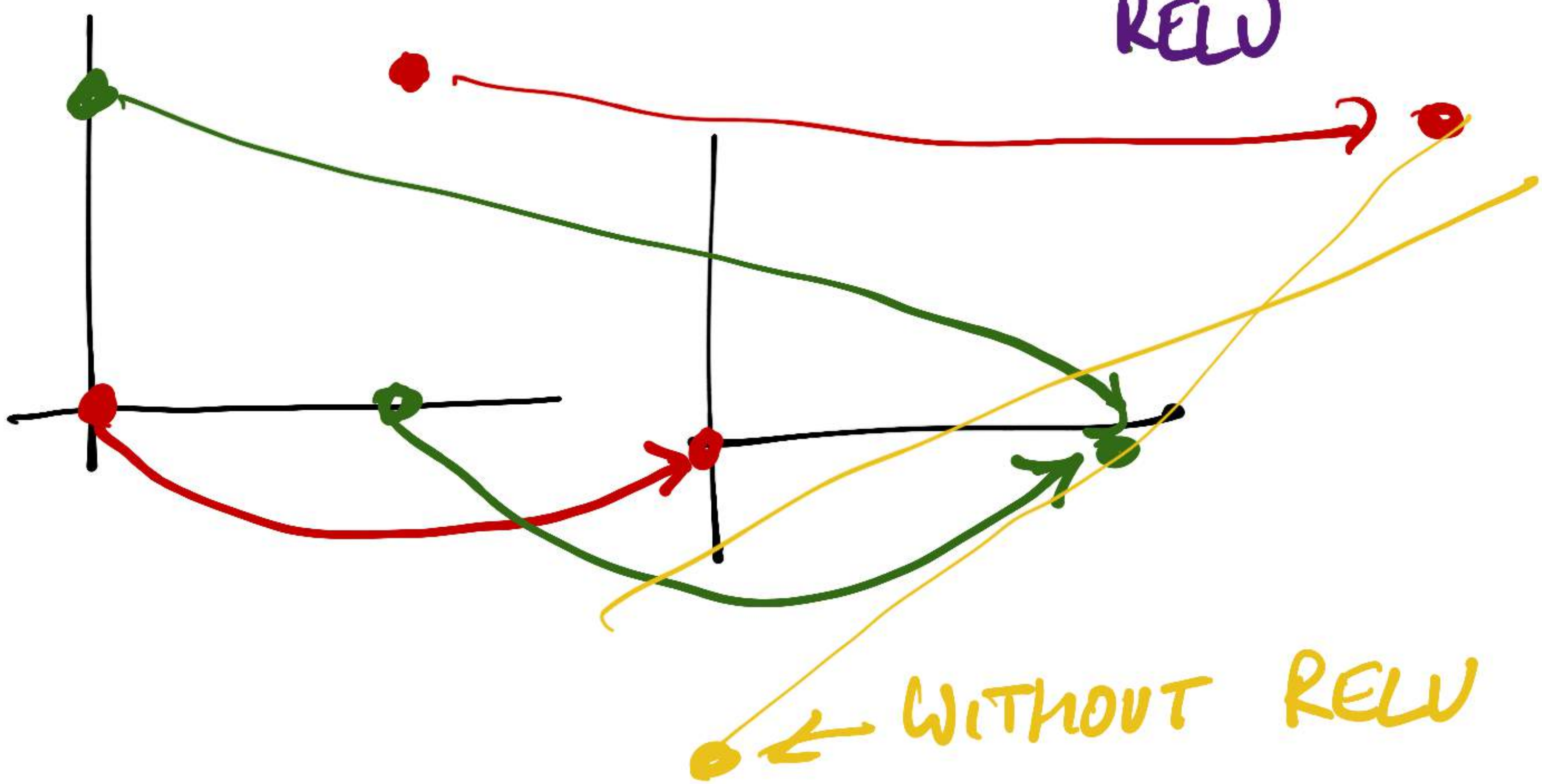
$Wx + c$ $\qquad$ $wx + b$

Solution for XOR

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$c = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \qquad b = 0$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \xrightarrow{\text{RELU}} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



Goal - find W through learning

1. Gradient descent

Loss function

Back propagation → Calculating gradients for hidden weights

Stochastic gradient descent

2. Choice of "activation functions"

3. Architecture - layers, connectivity etc

Gradient descent

Loss function / cost

Shift weights according to gradient wrt loss function

Mean Squared Error (MSE)

Cross entropy

Why these functions ?

Maximum likelihood estimators (MLE)

Assuming we are computing probabilistic output. $p(y|\bar{x})$

Sequence of Heads & Tail $\Rightarrow$ Observation O

$\Rightarrow p^* = \frac{\# \text{Heads}}{\text{Total}}$

n heads t tails

Hypothesis h for the coin probability

$$h^n (1-h)^t$$

$$\arg\max_{h} p(O|h) = p^*$$

NN computes MLE for $p(y|\bar{x})$

$p^*(y|\bar{x})$ — what NN reports

"Real" $p(y|\bar{x})$

Want p^* to be as close to p as possible

Information theory

Entropy $H = -\sum p \log(p)$

$= \mathbb{E}_{x \sim p} \log(x)$

Minimizes encoding of given distribution

Optimal encoding for Q, apply it to P

KL distance

$$D_{\mathrm{KL}}(P \| Q) = \mathbb{E}_{\mathrm{x} \sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x} \sim P}[\log P(x) - \log Q(x)] .$$

Cross Entropy

$$H(P,Q) = H(P) + D_{\mathrm{KL}}(P\|Q)$$

$$-\sum p \log(P) + \sum p \log(P) - \sum p \log(Q)$$

$$-\sum p \log(Q)$$

NN learns $p^*(y|x)$

Distance from $p(y|x)$

$$H(p, p^*) = -\mathbb{E}_{x\sim p} \log(p^*)$$

Minimize $-E_{x\sim p}$ .. $\Rightarrow$ Maximize $E_{x\sim p} \log(p^*)$

Back to MLE

Observations $o_1, o_2, \ldots, o_n$

hypothesis h

$$\underset{h}{\operatorname{argmax}} \prod_{i=1}^{n} P(o_i | h)$$

$$\underset{h}{\operatorname{argmax}} \prod_{i=1}^{n} P(O_i \mid h)$$

$$= \underset{h}{\operatorname{argmax}} \log\left(\prod_{i=1}^{n} P(O_i \mid h)\right)$$

$$= \underset{h}{\operatorname{argmax}} \frac{1}{n} \sum_{i=1}^{n} \log\left(P(O_i \mid h)\right)$$

$$= \mathbb{E}_{x \sim p} \log(p^*)$$

Cross Entropy is equivalent to MLE

Specific cases

Linear regression

$P(y|x)$

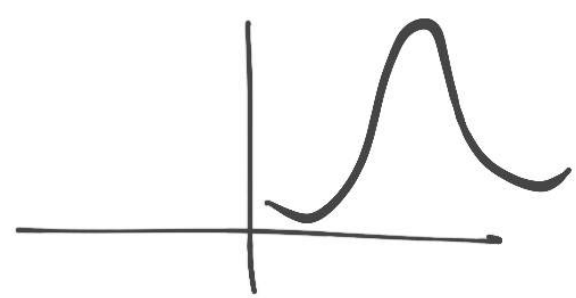
$\downarrow$

$x \rightarrow y^*$

$N(--, \sigma)$

Normal distribution centered at computed y^*

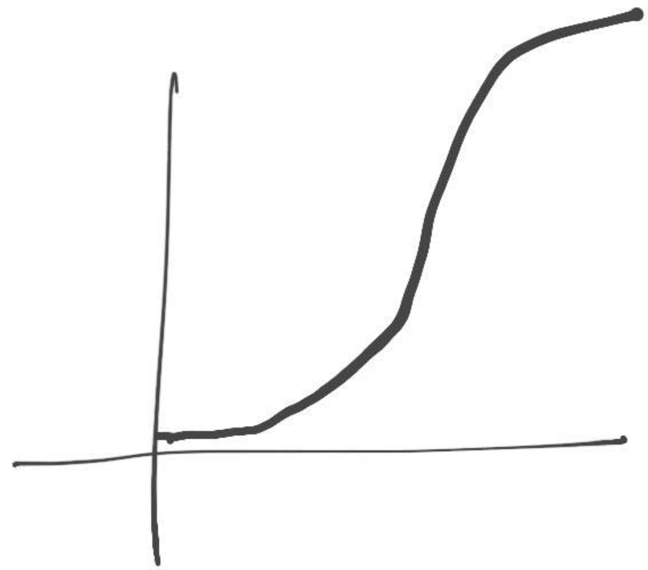
$\downarrow$



MSE

2 way sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$P(y=1) = y$$

$$P(y=0) = 1-y$$

$$p^*(x) = \sigma(z) = a$$

$$\sum p(x)\,\underset{a}{\log}\, p^*(x) = y\log(a) + (1-y)\log(1-a)$$