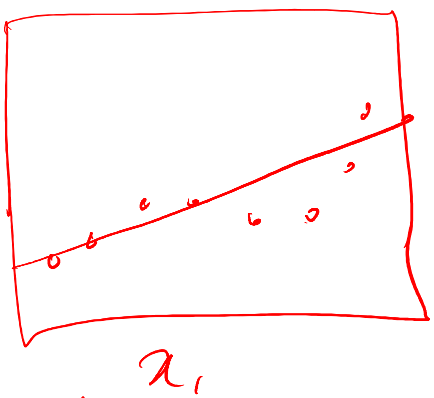
Introduction to Neural Networks

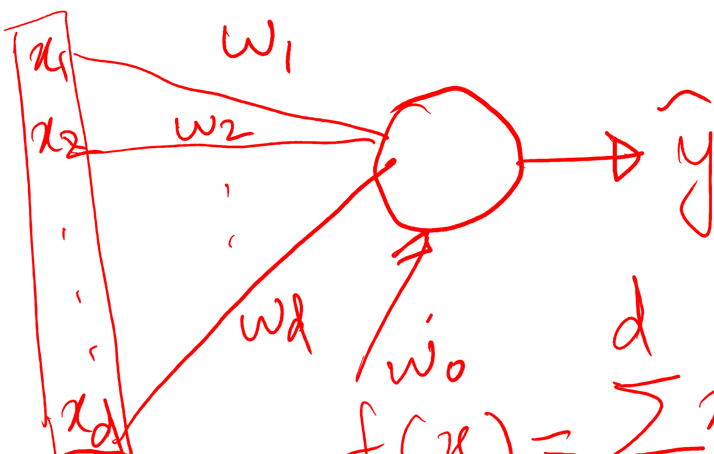
Original author: Preethi Jyothi
Modified by: Sunita Sarawagi

Linear models

Regression



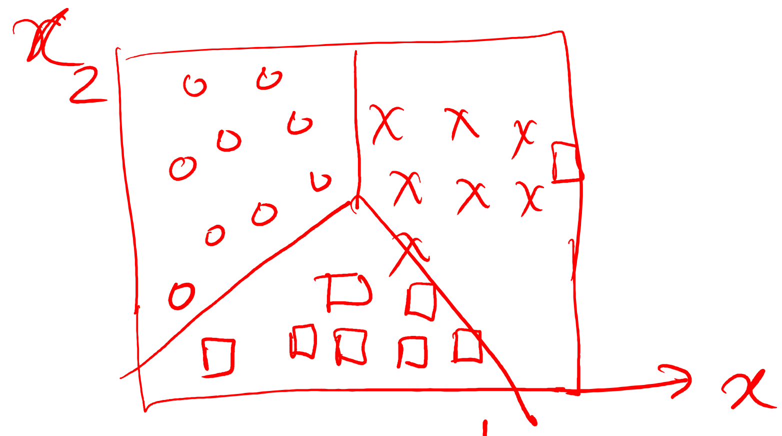
$d = 1$



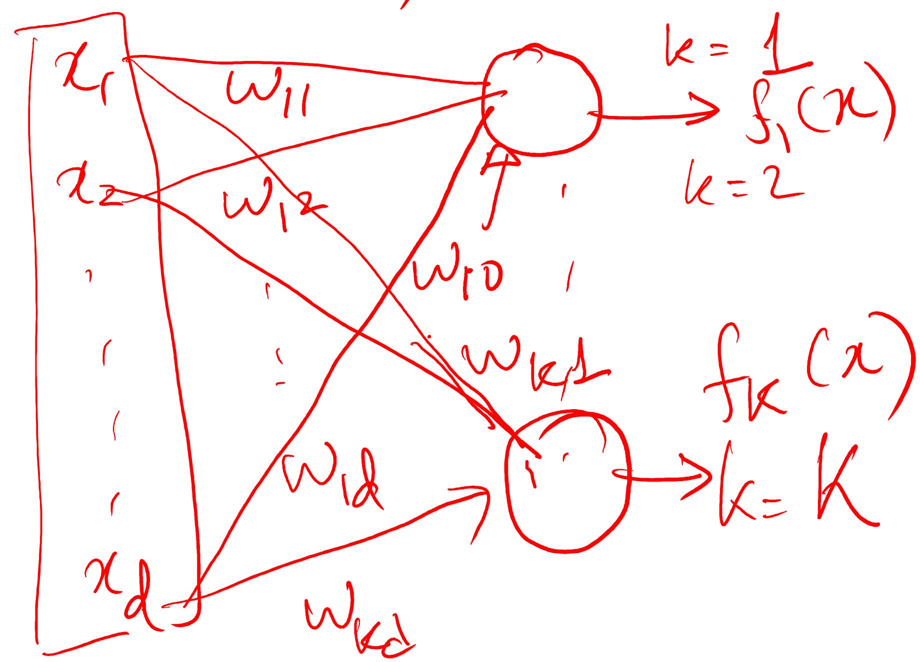
$$f(x) = \sum_{j=1}^{d} x_j w_j + w_0$$

x

Classification



$K = 3, \quad d = 2$



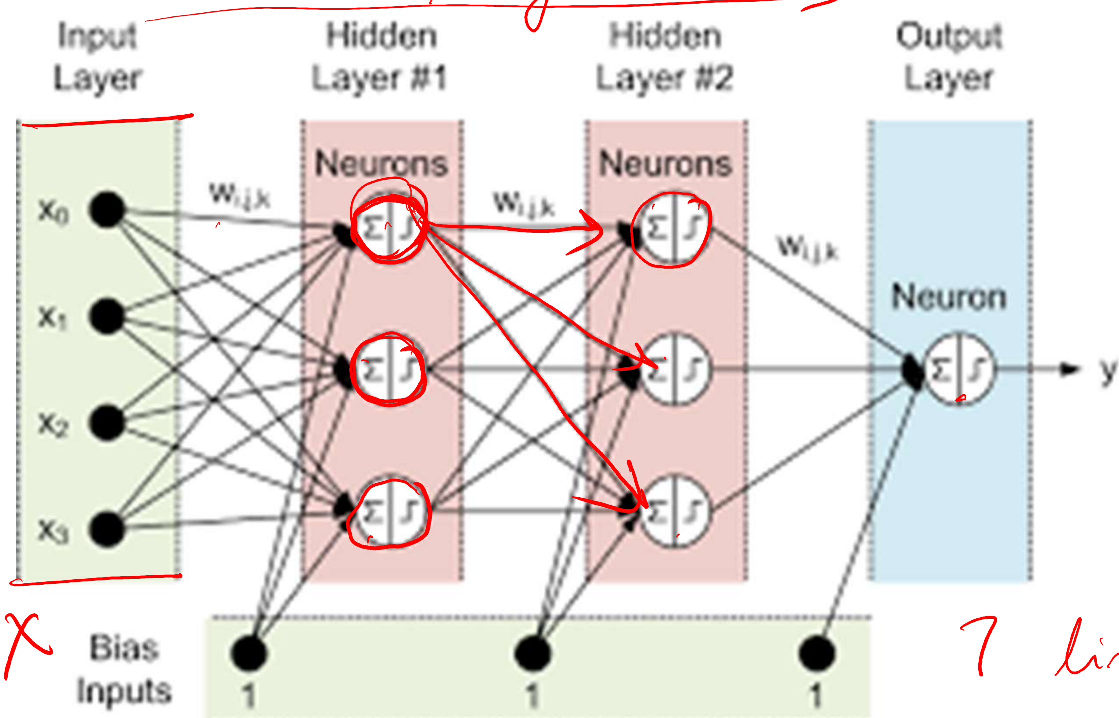
Non-linear functions?

- Embed input into a higher dimensional space
  - Example to create a quadratic decision surface design $\phi(x) = [x^2, x, 1]$
  - Burden on user to design the right embedding.
    - Difficult for complicated data types, e.g. image, speech, time-series, text, etc.
- Use universal kernel, e.g. RBF kernel (To be discussed)
  - Expensive to train, at least quadratic in the size of the input.
  - Cannot scale to millions of examples.

Traditional Neural Networks

Many linear functions connected layer-wise after simple non-linear activation

layer



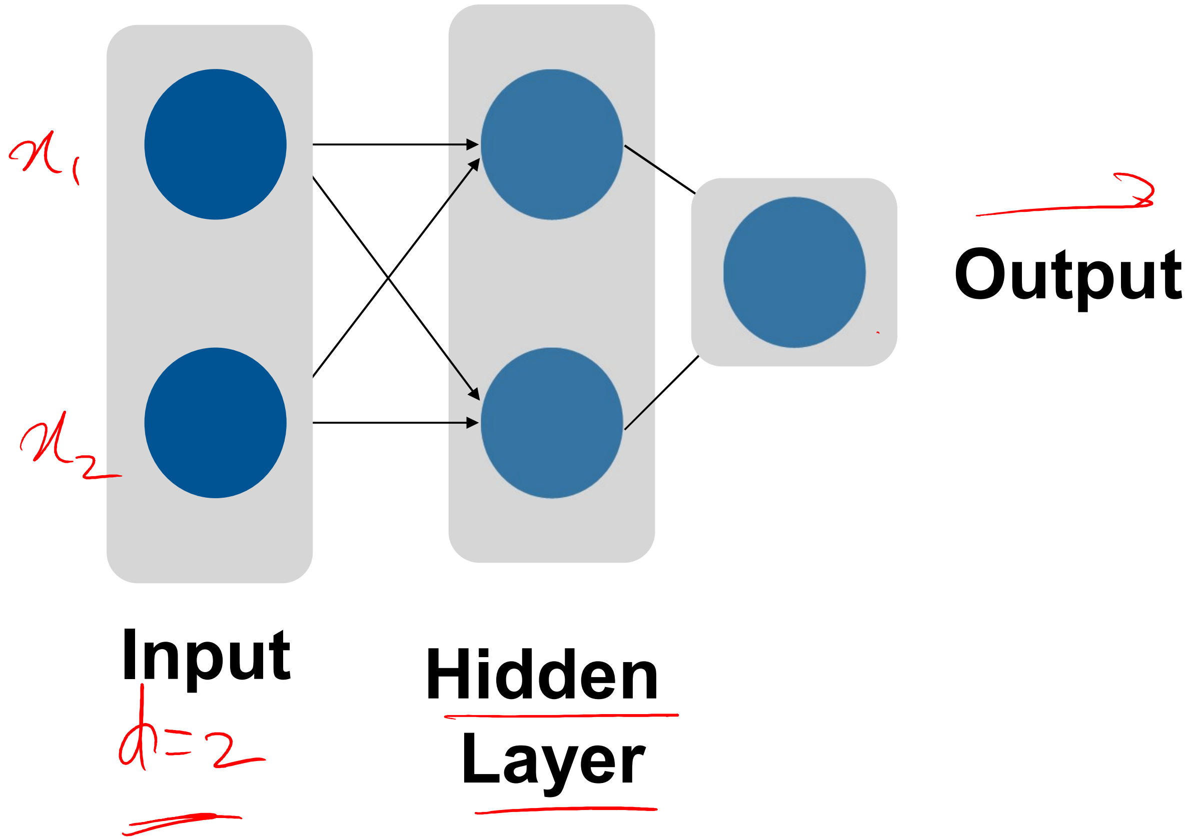
x

- ⋆ Universal function approximator
- ⋆ High-capacity

width

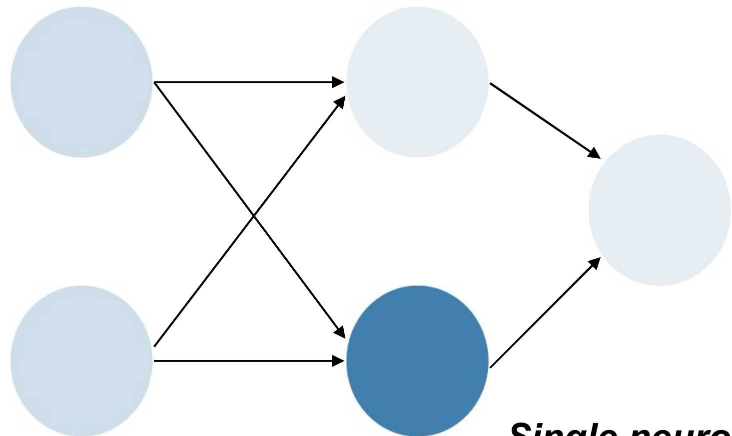
? linear function

Feed-forward Neural Network

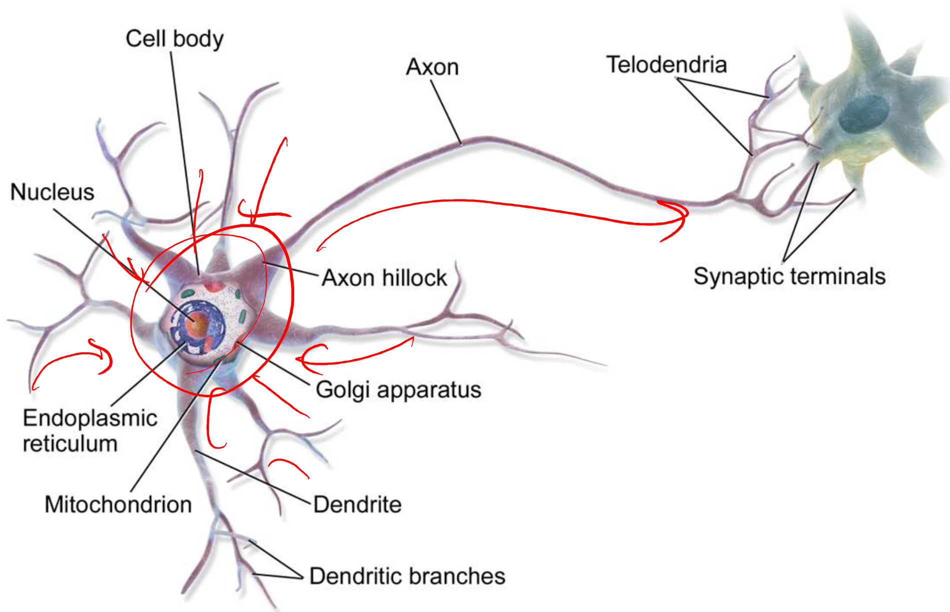


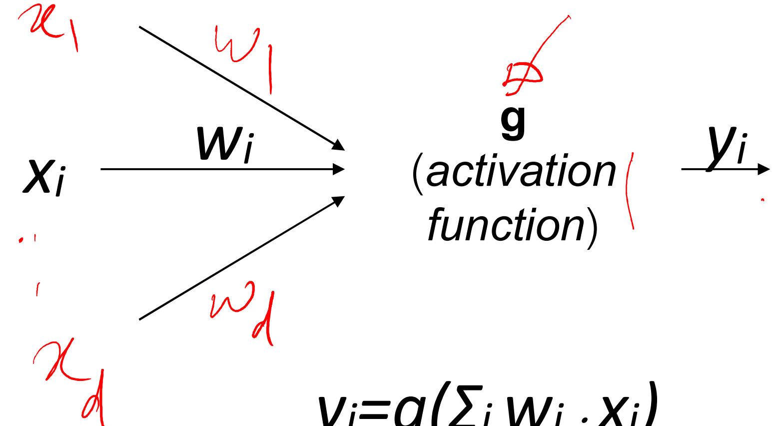
Feed-forward Neural Network

Brain Metaphor



Single neuron



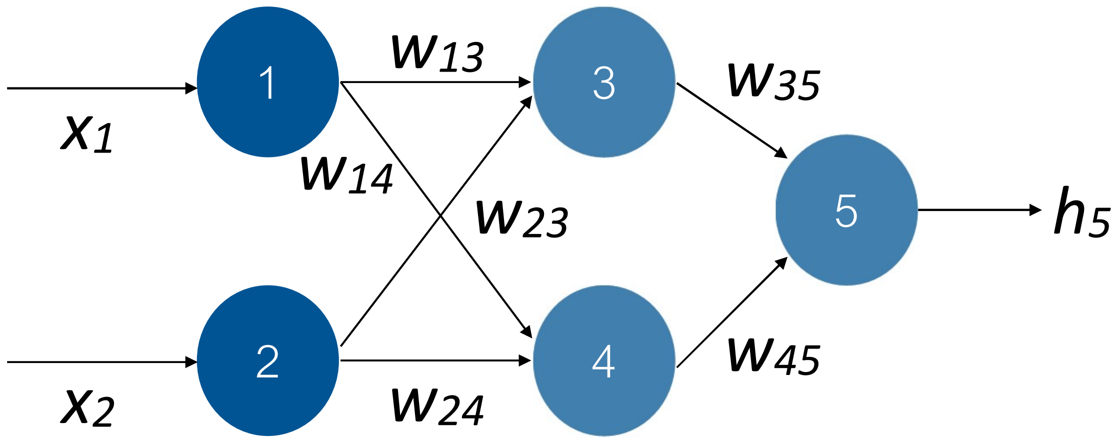


$$y_i = g(\Sigma_i\, w_i \cdot x_i)$$

Image from: https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png

Feed-forward Neural Network

Parameterized Model



$$
\begin{aligned}
h_5 &= g(w_{35} \cdot h_3 + w_{45} \cdot h_4) \\
&= g(w_{35} \cdot (g(w_{13} \cdot x_1 + w_{23} \cdot x_2)) + \\
&\qquad w_{45} \cdot (g(w_{14} \cdot x_1 + w_{24} \cdot x_2)))
\end{aligned}
$$

Parameters of the network: all w_{ij} (and biases not shown here)

If **x** is a 2-dimensional vector and the layer above it is a 2-dimensional vector **h**, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{g}(\mathbf{Wx} + \mathbf{b})$$

where w_{ij} in **W** is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and **b** is the bias vector

Activation Functions (g)

- Want a function that is efficient to compute, easy to optimize (informative gradient), almost linear

- Cannot be linear: Will get back a linear classifier otherwise

$$h5 = g(w_{35} \cdot h_3 + w_{45} \cdot h_4)$$

$$h_5 = g(w_{35} \cdot (g(w_{13} \cdot x_1 + w_{23} \cdot x_2)) + w_{45} \cdot (g(w_{14} \cdot x_1 + w_{24} \cdot x_2)))$$

If g(z)=z is a linear function we get:

$$h_5 = \underbrace{(w_{35}w_{13} + w_{45}w_{14})}\,x_1 + \underbrace{(w_{35}w_{23} + w_{45}w_{24})}\,x_2$$

$$w_1 x_1 \quad + \quad w_2 x_2$$

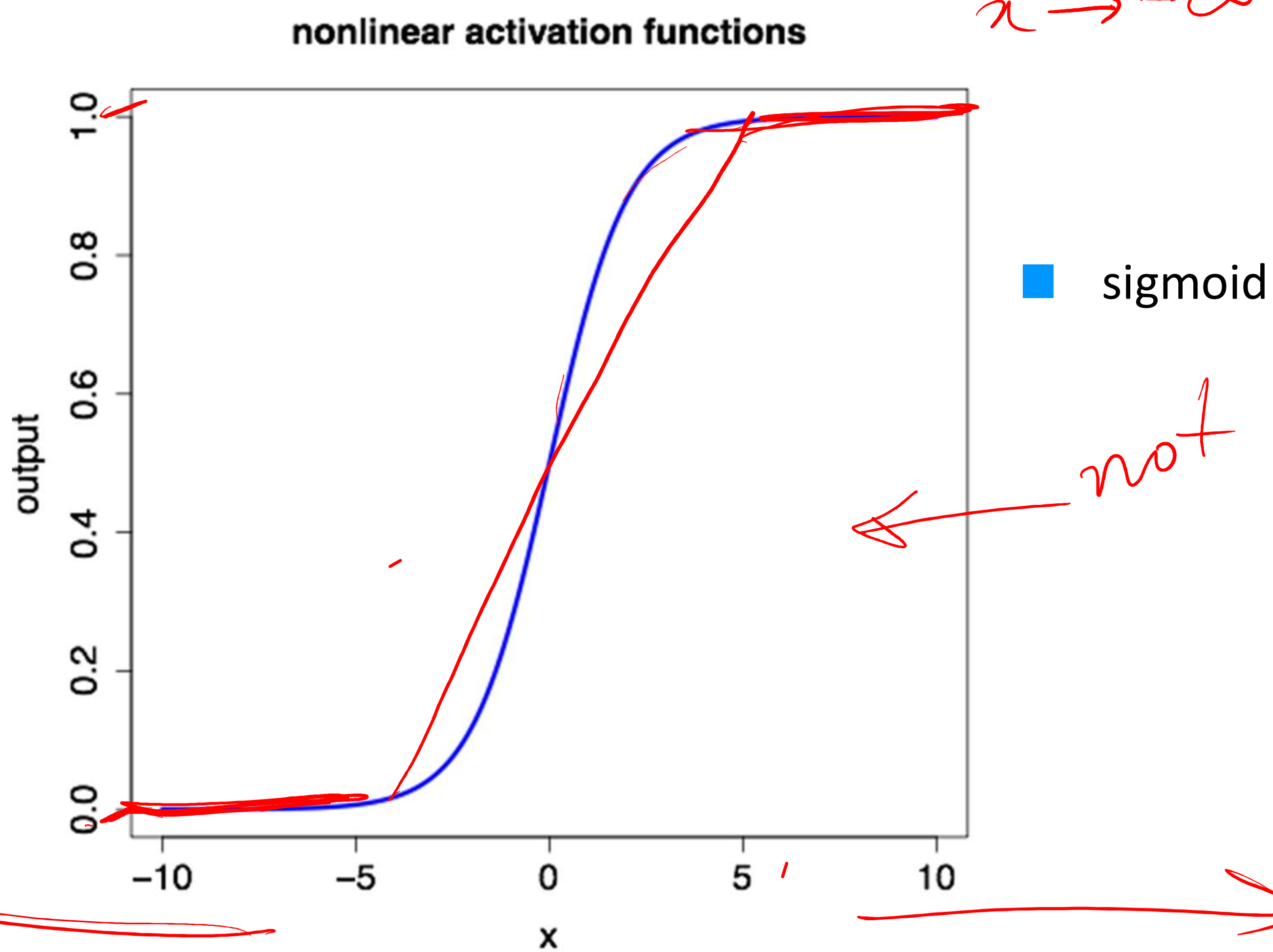
Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

$$\frac{1}{1+e^{-x}}$$

$$\underset{x \to \infty}{\sigma(x)} = \frac{1}{1+0} \approx 1$$

$$\underset{x \to -\infty}{\sigma(x)} = \frac{1}{1+e^{\infty}} \approx 0$$



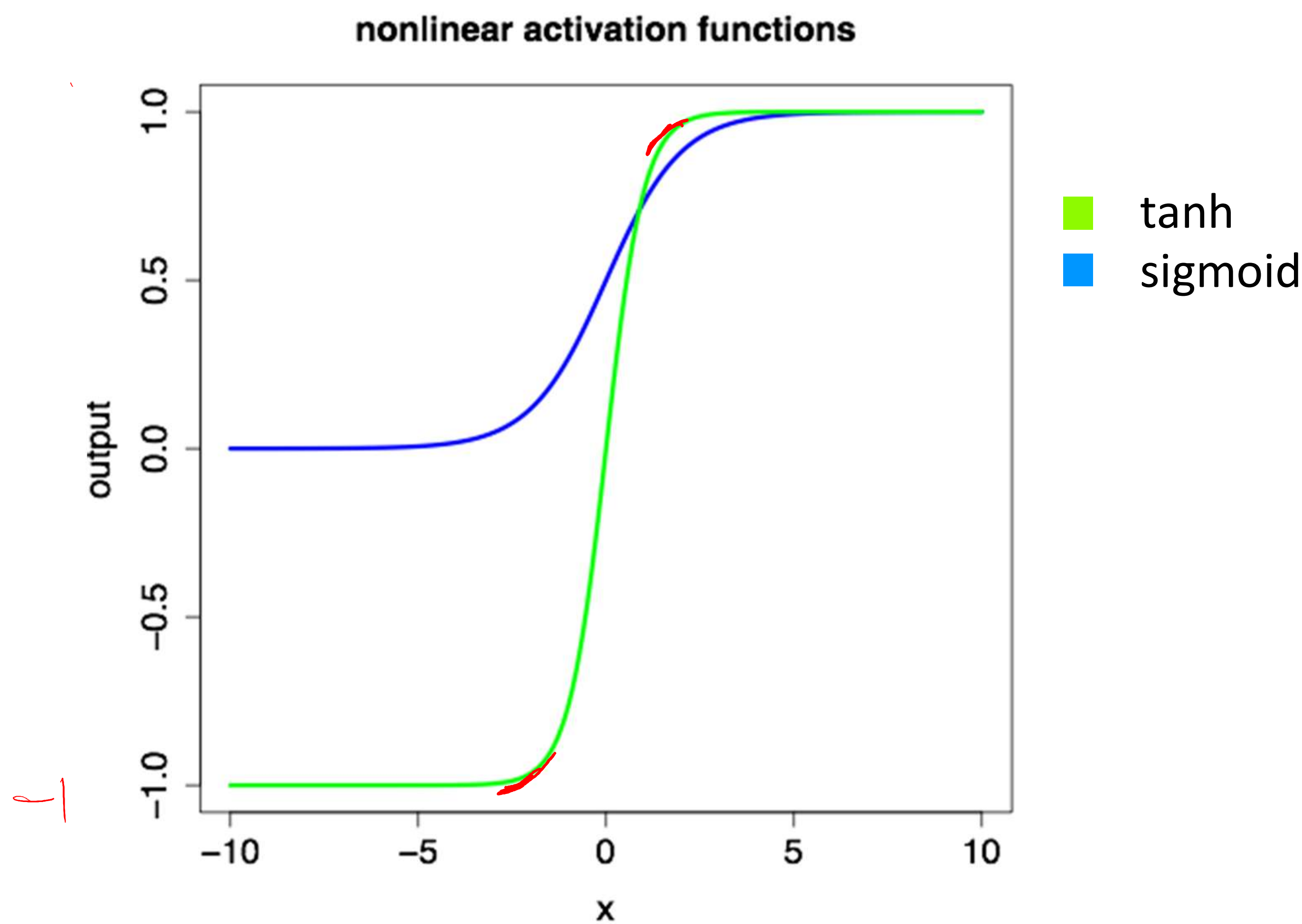
not convex

$-\infty \longleftarrow$ $\longrightarrow \infty$

Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

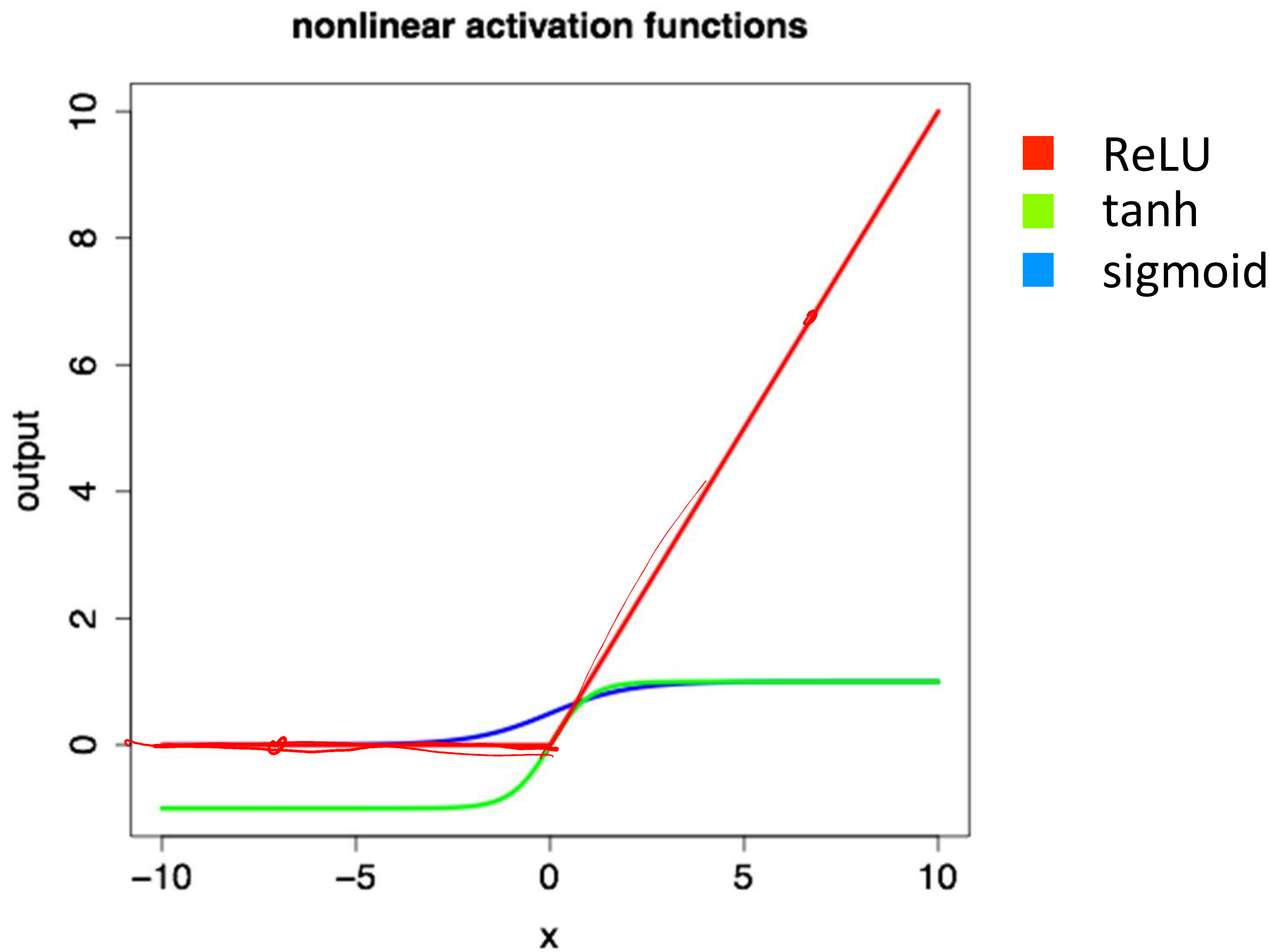


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\mathrm{RELU}(x) = \max(0, x)$ convex

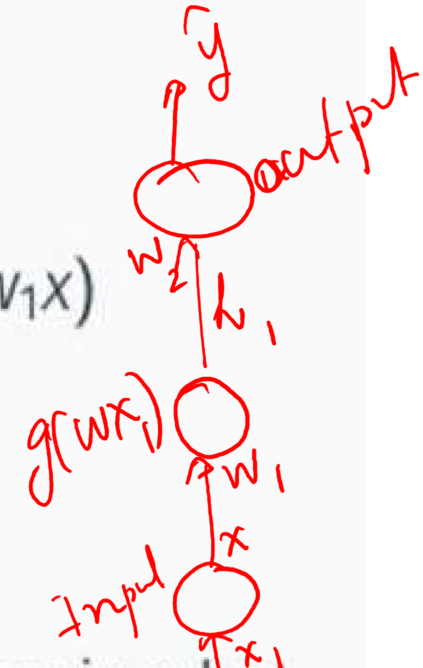


Choosing g()

Considerations: want some non-linearity, informative gradient (e.g. when convex), fast computation, close to linear

Role of the gradient of g during training

Training objective of DNN with one hidden unit $h = g(w_1 x)$



$J(w_1, w_2, x, y) = L(hw_2 y) = L(\underbrace{g(w_1 x)}_{h} w_2 y)$

Gradient of above w.r.t w_1 is $L' w_2 y g' x$

If $g' = 0$, the gradient becomes zero and we do not know in what direction to move w_1.

Choosing g()

- RELU: not differential but okay since gradient is informative. second-derivative zero in most places (useful for optimization)
    - Caution: watch out for inactive RelU: initialize affine input bias parameter to small positives. Gradient zero ==> information flow to lower layers is blocked.
- Sigmoid/Tanh: $\tanh(z) = 2\,\mathrm{sigmoid}(2z)$. Non-convex. Well-behaved (linear) only for small values of z, gradients very small for small or large z, problem for multi-layer network.

Neural Network demo

Playground.tensorflow.org

Example XOR

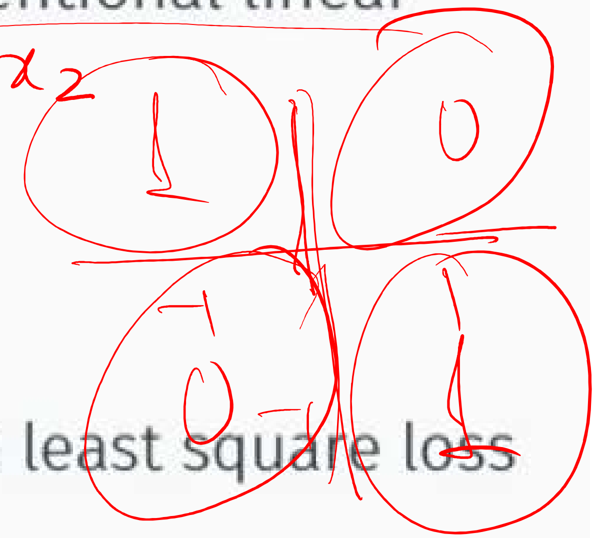
Neural networks can model decisions that conventional linear classifiers cannot.

$y = f^*(x) = x_1 \oplus x_2$

Training data = all four combinations.

$x_1 \quad x_2 \quad f^*(x) = x_1 \oplus x_2$

x_1	x_2	$f^*(x)$
$\emptyset$ (−1)	$\emptyset$ (−1)	0
1	1	0
$\emptyset$ (−1)	1	1
1	0 (−1)	1



Linear classifier $\hat{y} = w_1x_1 + w_2x_2 + b$ trained with least square loss yields $w_1 = w_2 = 0, b = 1/2$

Cannot discriminate

Non-linear classifier such as one with $\boxed{x_1x_2}$ as feature $(\hat{y} = w_1x_1 + w_2x_2 + w_3x_1x_2 + b)$ can discriminate but the burden is on us to create the useful non-linear features.

$w_1 = 0, \quad w_2 = 0 \quad w_3 = -1, \quad b = +1$

$\hat{y} = 0 + 0 + (-1) + 1 = 0 \qquad x_1 = x_2 = -1$

$\hat{y} = 0 + 0 + (+1) + 1 = 2 \qquad x_1 = -1, \; x_2 = 1$

Example: XOR

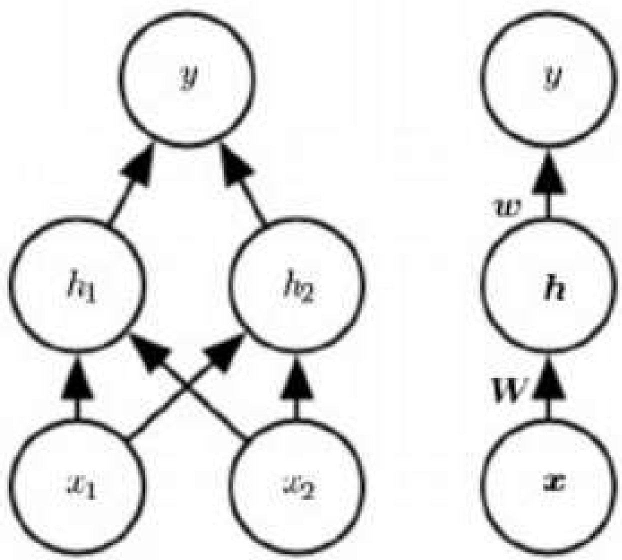
A generic two layer neural network with ReLU:

$$y = f(x) = W^2 \max(0, W^1 x + b^1) + b^2$$

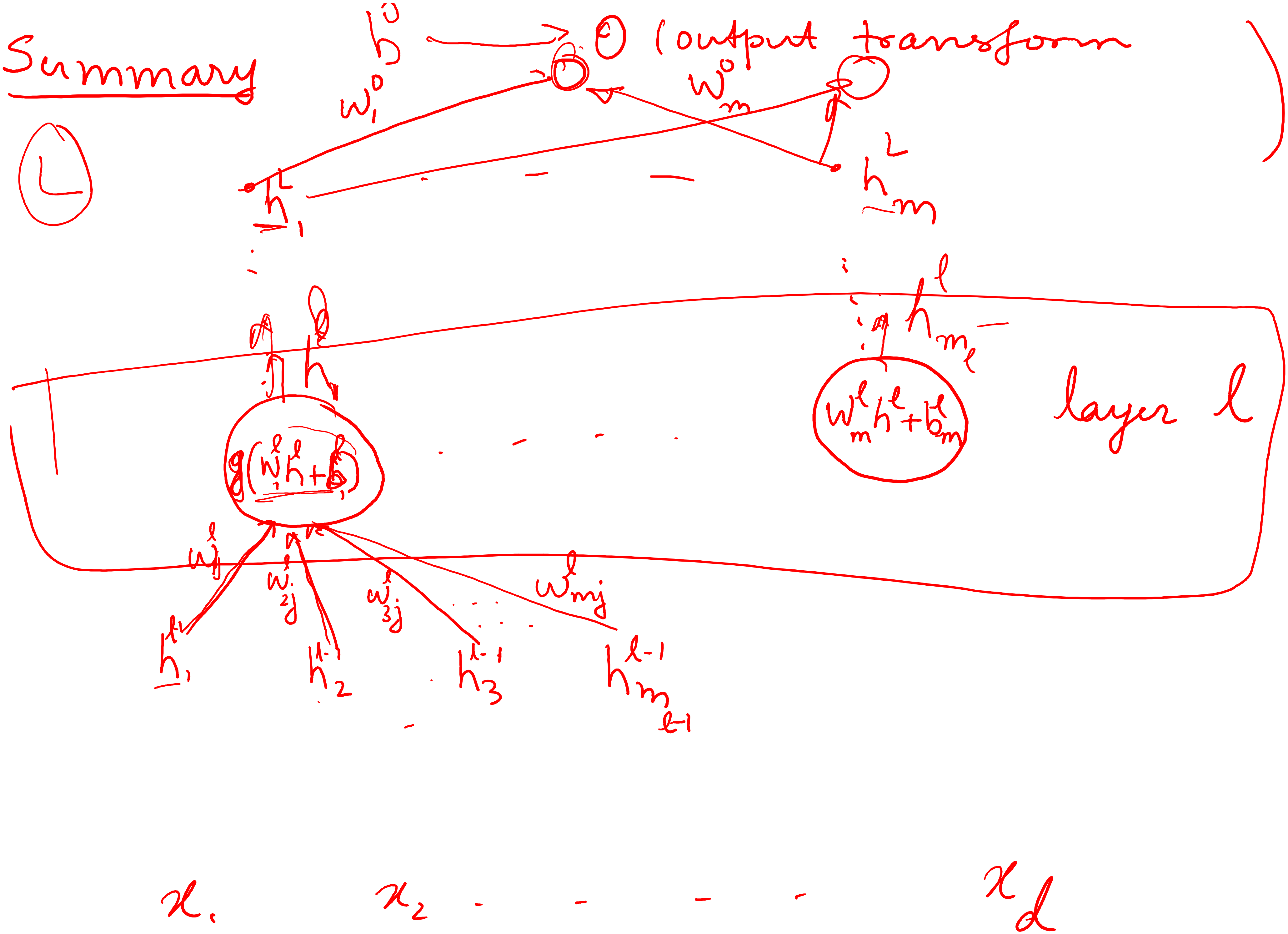
Role of non-linear transform.

$$W^1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, W^2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b^2 = 0$$



Summary


b^0
O (output transform)
w_1^0
w_m^0
L
h_1^L
h_m^L
h_1^l
$h_{m_l}^l$
$g(w_1^l h^l + b_1^l)$
$w_m^l h^l + b_m^l$
layer l
w_{1j}^l
w_{2j}^l
w_{3j}^l
w_{mj}^l
h_1^{l-1}
h_2^{l-1}
h_3^{l-1}
$h_{m_{l-1}}^{l-1}$
x_1
x_2
x_d

[But what is a neural network? | Chapter 1, Deep learning - YouTube]

Network Architecture

Choosing the number of layers and width of the network and connection between layer

- Universal approximation theorem: A network with one hidden layer (sigmoid type activation) can approximate any continuous function from a closed and bounded set given enough hidden units.
- Proof also extended to work for RELU activations.
- Not useful in practice:
    - number of hidden units required may be exponentially large,
    - the parameters of the network may not be easily learnable: might overfit on a wrong function.

Effect of depth

- Many functions can be efficiently represented with multiple hidden layers but require exponential width with single hidden layer
- The number of linear regions carved out via d inputs, l+1 depth, c units per hidden layer is $O\left(C(c,d)^{dl}c^d\right)$
- Empirically too, larger depth leads to better generalization and lower error.