# CS60050
# Machine Learning

## Neural Networks - Introduction

Somak Aditya
Assistant Professor

Sudeshna Sarkar

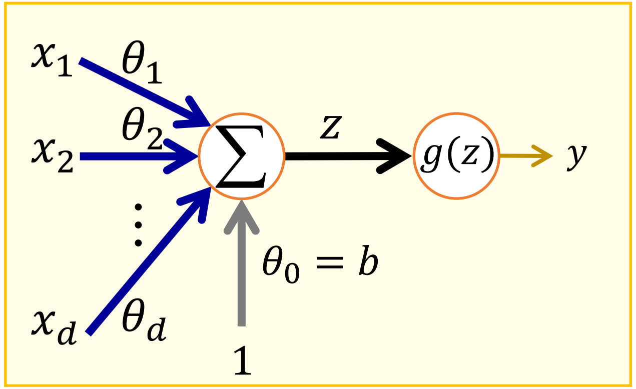
Department of CSE, IIT Kharagpur

Oct 12, 2023

# Neural Networks

- Origins: Algorithms inspired by the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s. (*the famed AI Winter*)
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

# Linear Models



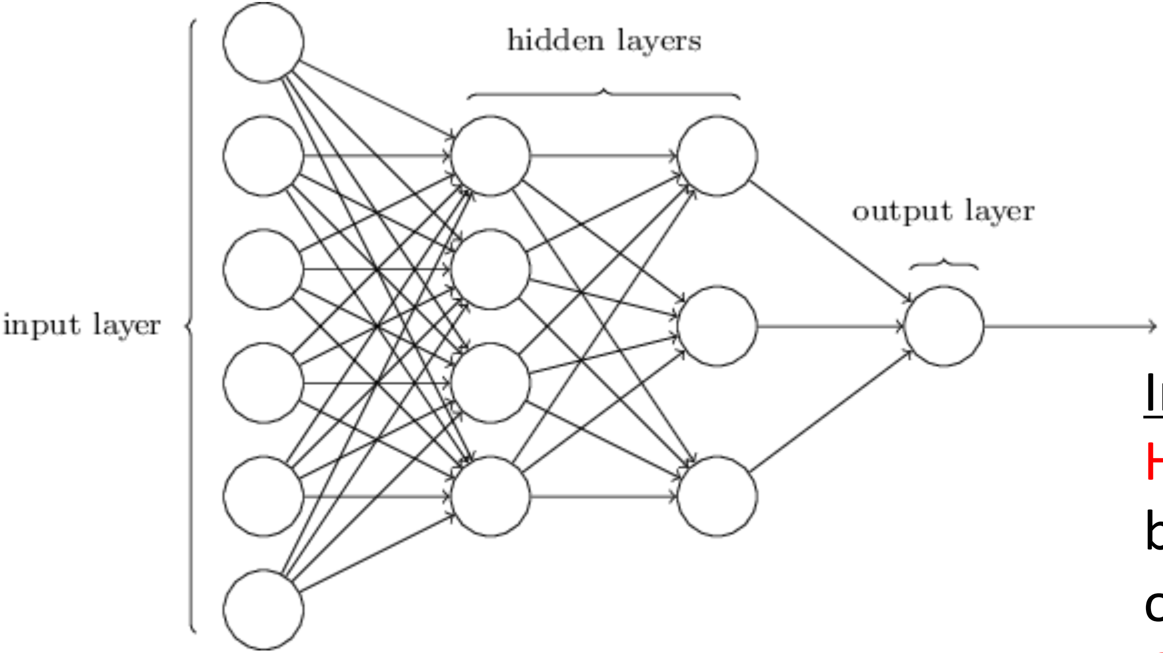

$\boldsymbol{w} = [\theta_1\ \theta_2 \dots \theta_d]^T$ and $\boldsymbol{x} = [x_1\ x_2 \dots x_d]^T$

$$\boldsymbol{z} = b + \sum_{i=1}^{d} \theta_i x_i = [\boldsymbol{\theta}^T b] \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$\boldsymbol{y} = g(z)$$

- Regression: $g(z) = z$
- Classification:
  - Binary: $g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$
  - Multi-class

# Multilayer Perceptrons (MLP)

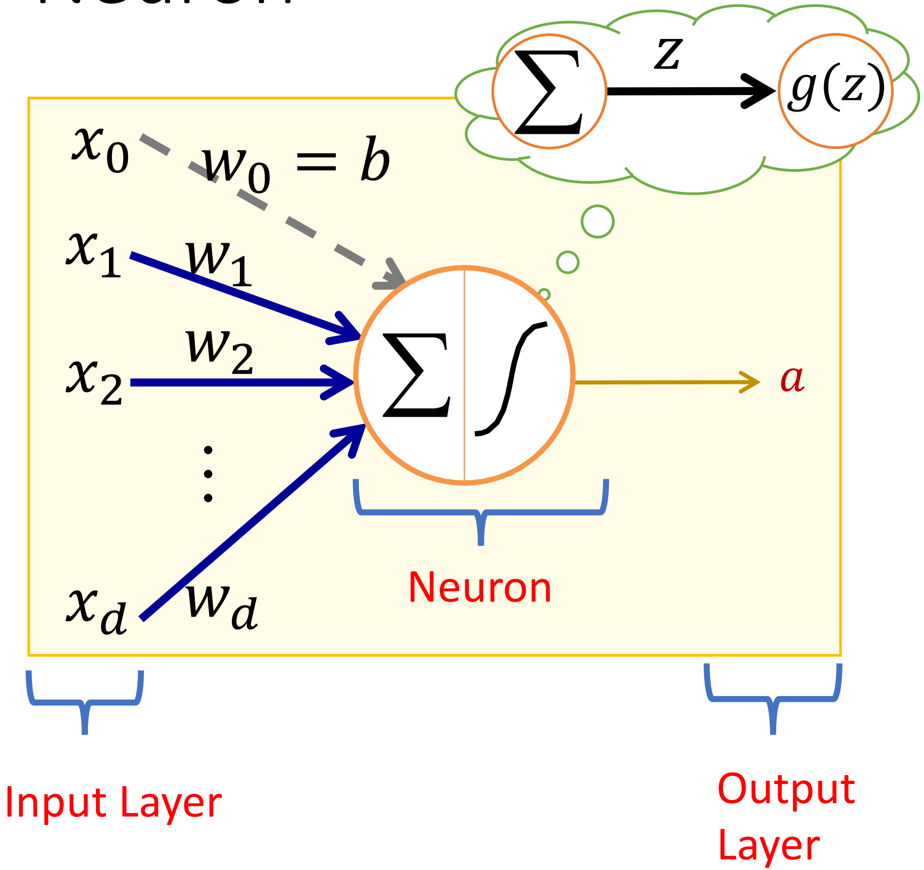



<u>Intuition</u>:-

Hidden Layer: Extracts better representation of the input data

Output layer: Does the classification

# Neuron




$\boldsymbol{w} = [w_1\ w_2 \dots w_d]^T$ and $\boldsymbol{x} = [x_1\ x_2 \dots x_d]^T$

$$\boldsymbol{z} = b + \sum_{i=1}^{d} w_i x_i = [\mathbf{w}^T b]\begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}$$

$$\boldsymbol{a} = g(z)$$

Terminologies:-

$\boldsymbol{x}$**:** input, $\boldsymbol{w}$**:** weights, $\boldsymbol{b}$**:** bias

$a$: pre-activation (input activation)

$g$: activation function

$y$: activation (output activation)

# Output Units: Linear

$$\hat{y} = w^T a + b$$

Used to produce the mean of a conditional Gaussian distribution:

$$p(\mathbf{y} \mid \mathbf{x}) = N(\mathbf{y}; \hat{\mathbf{y}}, \sigma)$$

Maximizing log-likelihood $\Longrightarrow$ Minimizing squared error

# Output Units: Sigmoid

$$\hat{y} = \sigma(w^T a + b)$$

$$J(\theta) = -\log p(y|x)$$
$$= -\log \sigma\big((2y-1)(\boldsymbol{w}^T \boldsymbol{a} + b)\big)$$

# Output Softmax Units

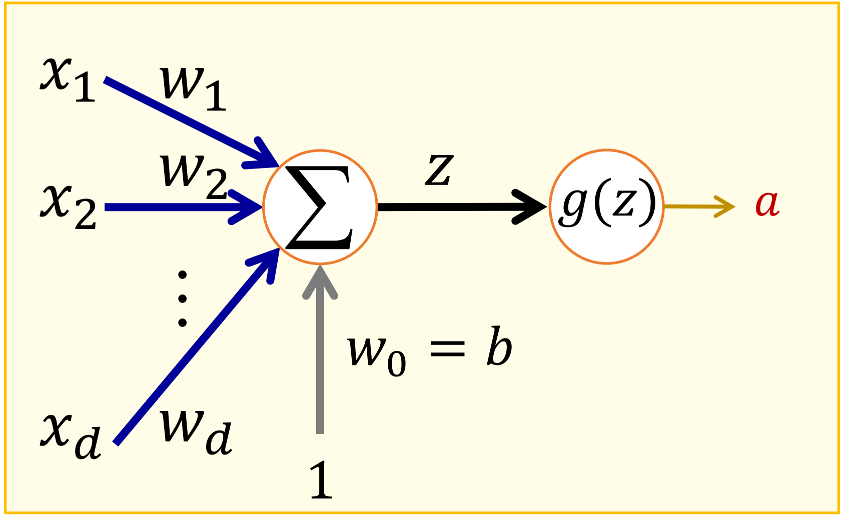
Need to produce a vector $\hat{y}$ with $\hat{y}_i = p(y = i|x)$

$$\operatorname{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$\log \operatorname{softmax}(z)_i = z_i - \log \sum_j \exp(z_j)$$

# Artificial Neuron – hidden unit

$\boldsymbol{w} = [w_1\ w_2 \ldots w_d]^T$ and $\boldsymbol{x} = [x_1\ x_2 \ldots x_d]^T$




$$\boldsymbol{z} = b + \sum_{i=1}^{d} w_i x_i = [\boldsymbol{w}^T b] \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$$

$$\boldsymbol{a} = g(z)$$

Terminologies

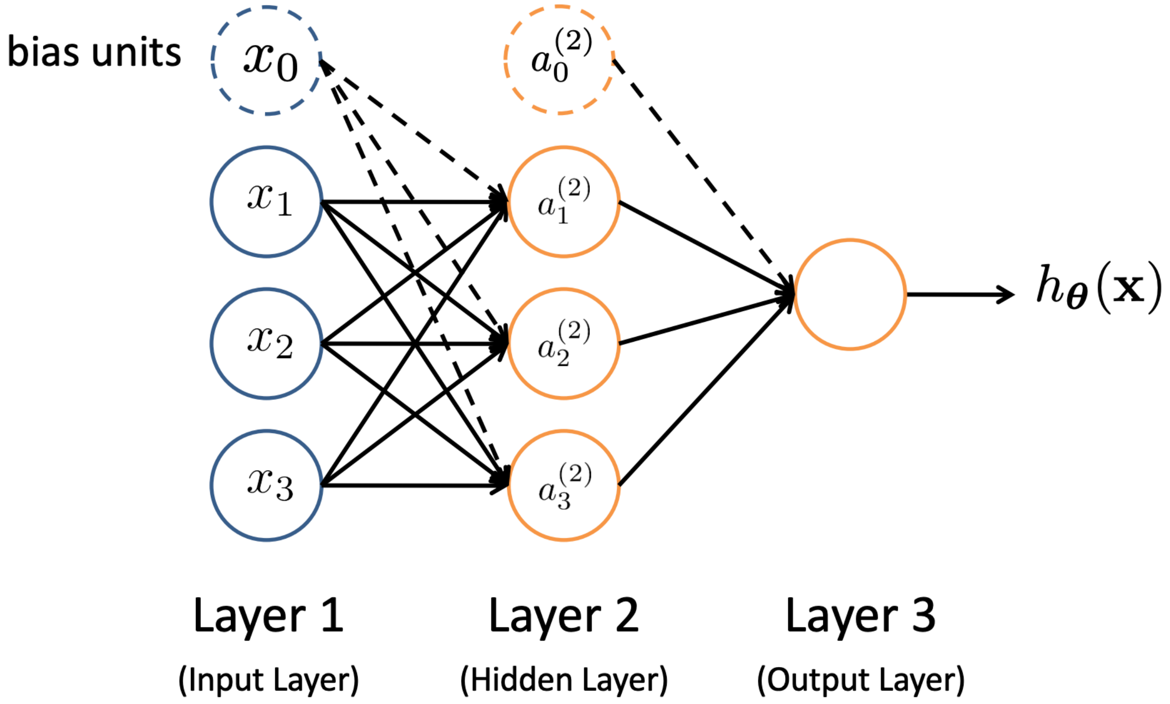
$\boldsymbol{x}$**:** input, $\boldsymbol{w}$**:** weights, $\boldsymbol{b}$**:** bias

$z$: pre-activation (input activation)

$g$: activation function

a: activation at hidden units

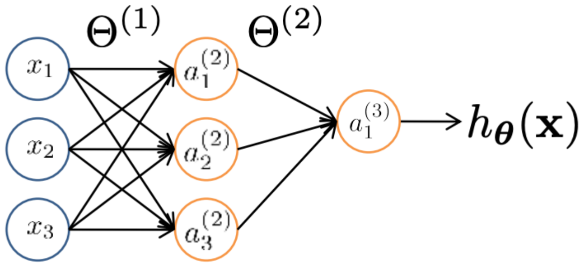
# Neural Network – Forward Pass



# Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value – Typically this is a nonlinear function, often **a sigmoid function corresponding to the "threshold" of that node**

# Neural Network




$a_i^{(j)}$ = "activation" of unit *i* in layer *j*

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer *j* to layer *j* + 1

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}
$$

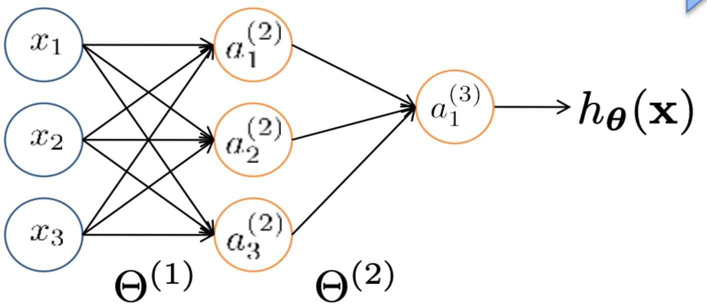
# Vectorized Representation

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$




**Feed-Forward Steps:**

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Based on slide by Andrew Ng

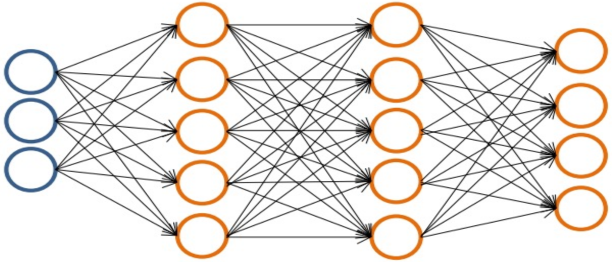
# Other Network Architectures

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

# Loss Functions

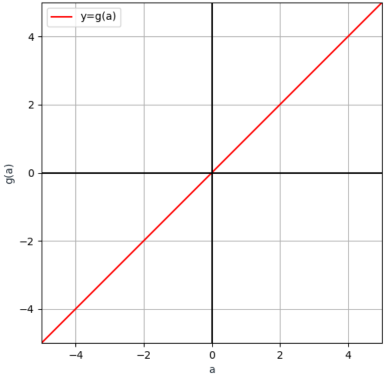
## Regression

- Squared error: $\mathrm{L}(y, \hat{y}) = (y - \hat{y})^2$

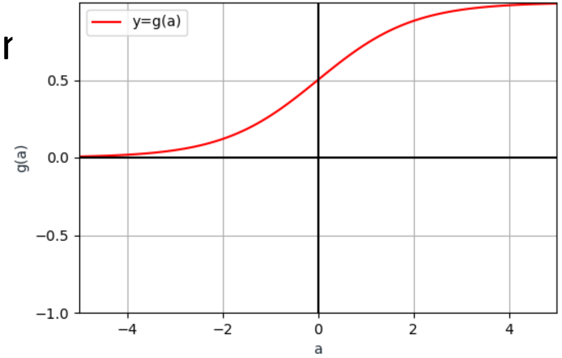
## Classification

- Cross entropy: : $L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_k y_k \log \hat{y}_k$
- Easy to scale to k classes.

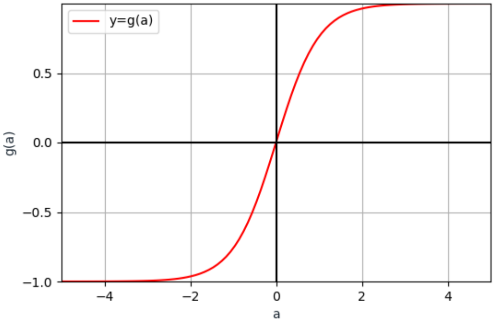
# Common Activation Functions



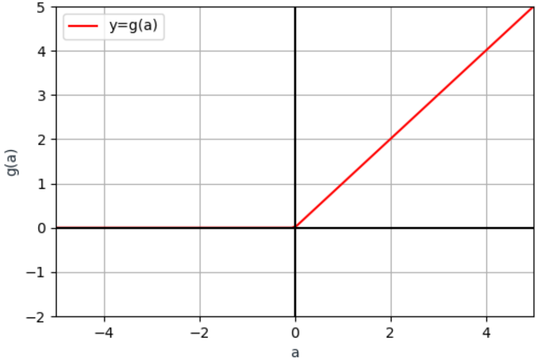

Linear activation functior
- $g(a) = a$
- Unbounded
- $g'(a) = 1$




Sigmoid activation function
- $g(a) = \sigma(a) = \frac{1}{1+\exp(-a)}$
- Bounded (0, 1)
- Always positive
- $g'(a) = g(a)(1 - g(a))$

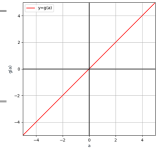
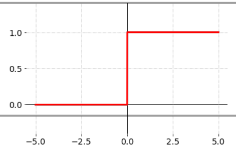
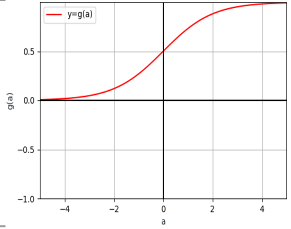
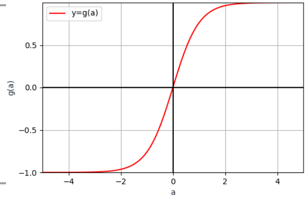
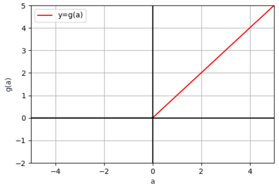



tanh activation function
- $g(a) = tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$
- Bounded (-1, 1)
- Can be positive or negative
- $g'(a) = 1 - g^2(a)$




ReLU activation function
- $g(a) = \max(0, a)$
- Bounded below by 0
- But not upper-bounded
- $g'(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$

# Common Activation Functions

| Name | Function | Gradient | Graph |
|---|---|---|---|
| Linear | $a$ | $1$ |  |
| Binary step | $\mathrm{sign}(a)$ | $g'(a) = \begin{cases} 0, & a \neq 0 \\ NA, & a = 0 \end{cases}$ |  |
| Sigmoid | $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$ | $g'(a)$ $= g(a)(1 - g(a))$ |   |
| Tanh | $\tanh(a)$ $= \dfrac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$ | $g'(a) = 1 - g^2(a)$ |   |
| ReLU | $g(a) = \max(0, a)$ | $g'(a)$ $= \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases}$ |   |

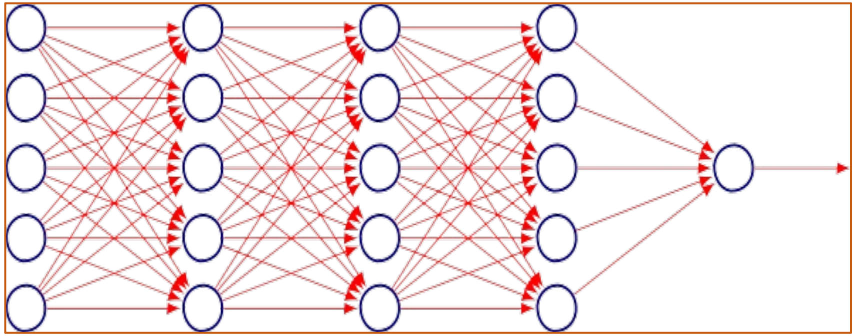
# Feedforward Networks and Backpropagation

# Introduction

- Goal: Approximate some unknown ideal function $f^*: X \rightarrow Y$
- Ideal classifier: $y = f^*(x)$ for $(x, y)$
- Feedforward Network: Define parametric mapping $y = f(x;\theta)$
- Learn parameters $\theta$ to get a good approximation to $f^*$ from training data
- Function $f$ is a composition of many different functions e.g.

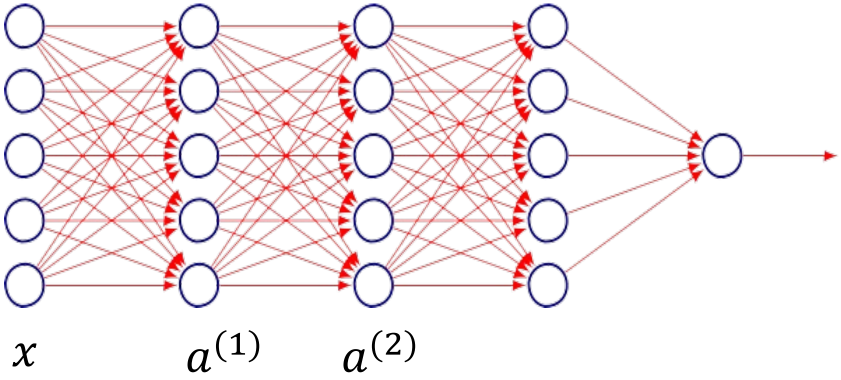
$$f(x) = f^3\left(f^2\left(f^1(x)\right)\right)$$

- Training: Optimize $\theta$ to drive $f(x;\theta)$ closer to $f^*(x)$
  - Only specifies the output of the *output layers*
  - Output of intermediate layers is not specified by D, hence the nomenclature *hidden layers*



- Neural: Choices of $f^{(i)}$'s and layered organization, loosely inspired by neuroscience

# Universality and Depth




- First layer:

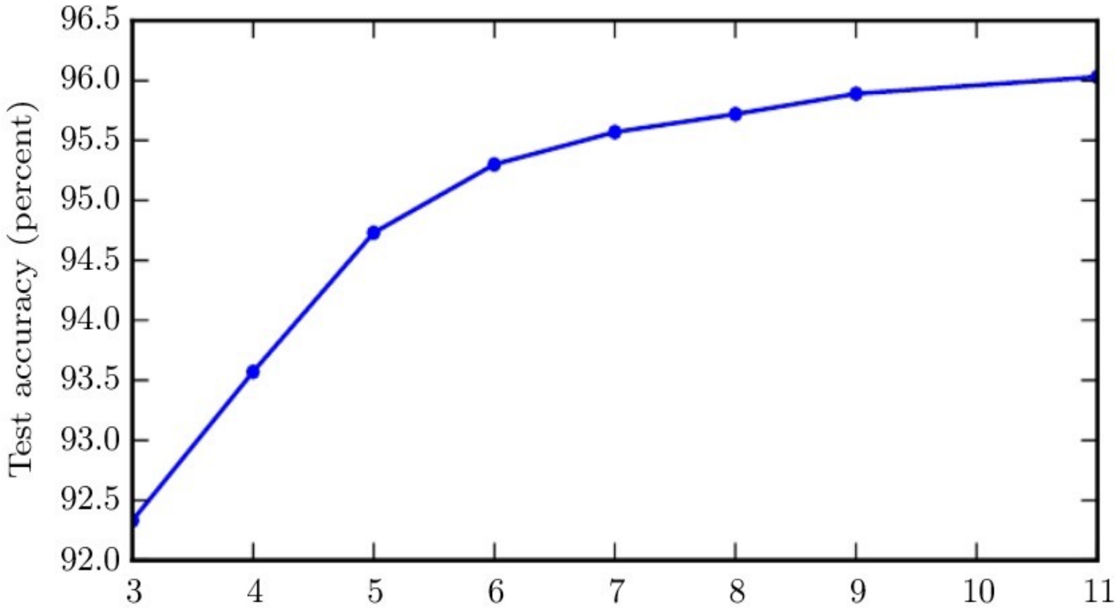
$$a^{(1)} = g^1\left(W^{(1)^T}x + b^{(1)}\right)$$
$$a^{(2)} = g^2\left(W^{(2)^T}a^{(1)} + b^{(2)}\right)$$

- How do we decide depth, width?
- In theory how many layers suffice?

# Universality

- Theoretical result [Cybenko, 1989]: 2-layer net with linear output with some squashing non-linearity in hidden units can approximate any continuous function over compact domain to arbitrary accuracy (*given enough hidden units!*)
- Implication: Regardless of function we are trying to learn, we know a large MLP can represent this function
- But not guaranteed that our training algorithm will be able to learn that function
- Gives no guidance on how large the network will be (exponential size in worst case)

# Advantages of Depth



# A visual proof that neural nets can compute any function

- See Chapter 4 of **Neural Networks and Deep Learning**

by Michael Nielsen

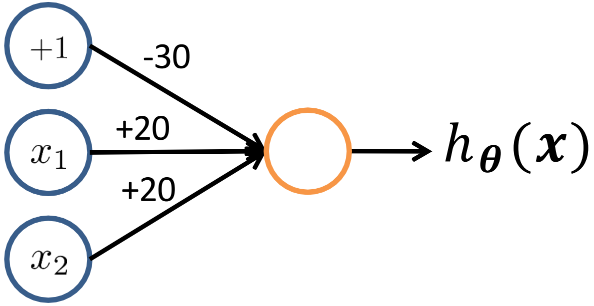
- http://neuralnetworksanddeeplearning.com/chap4.html

# Representing Boolean Functions

**Simple example: AND**

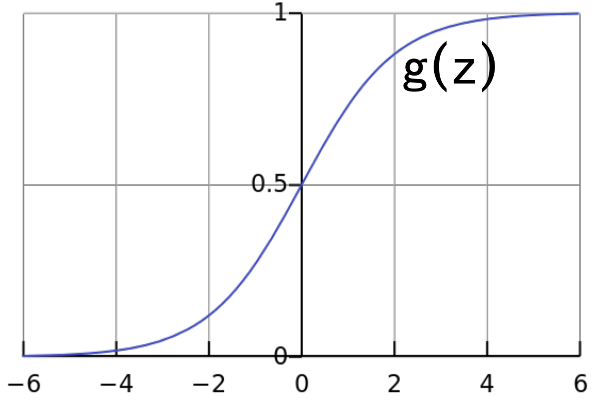
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



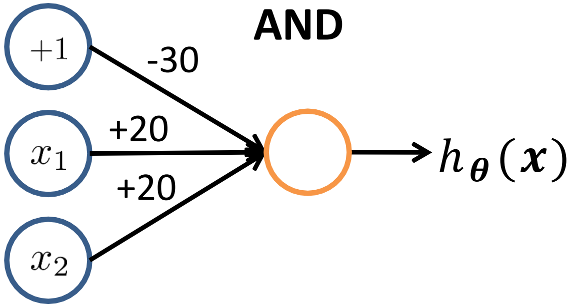

$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$

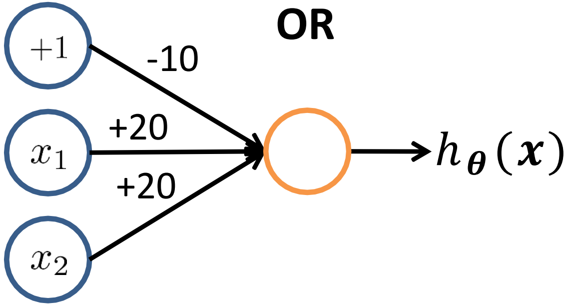
Logistic / Sigmoid Function

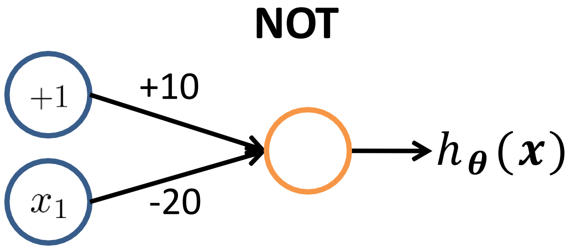


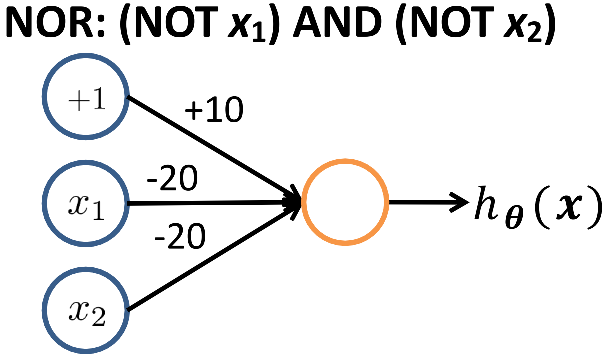

| $x_1$ | $x_2$ | $h_\Theta(\mathbf{x})$ |
|---|---|---|
| 0 | 0 | $g(-30) \approx 0$ |
| 0 | 1 | $g(-10) \approx 0$ |
| 1 | 0 | $g(-10) \approx 0$ |
| 1 | 1 | $g(10) \approx 1$ |

# Representing Boolean Functions



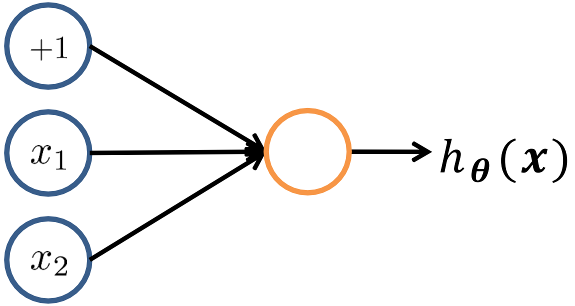



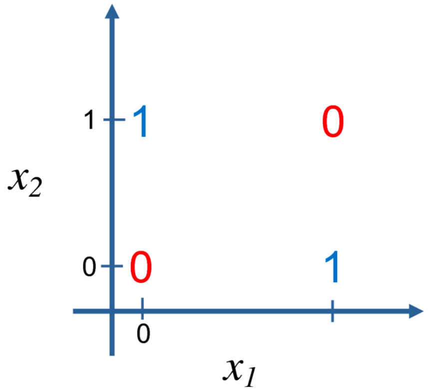



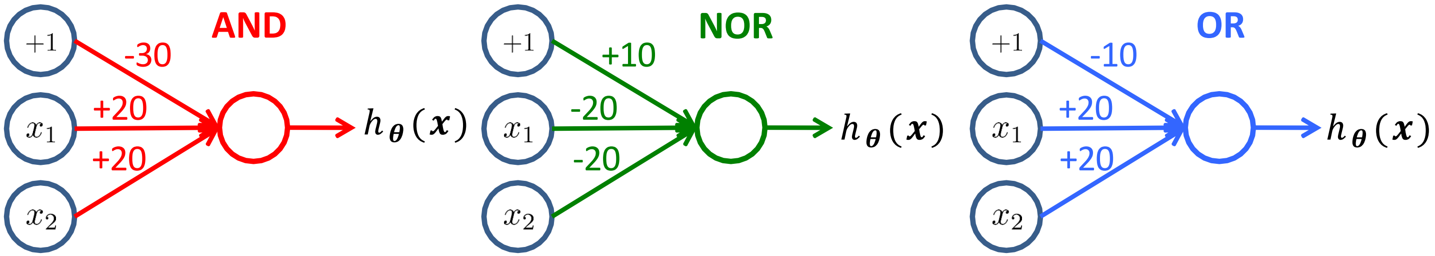
# Representing Boolean Functions

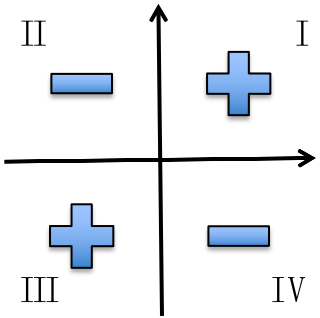
**XOR: ($x_1$ AND (NOT $x_2$)) OR ((NOT $x_1$) AND $x_2$)**

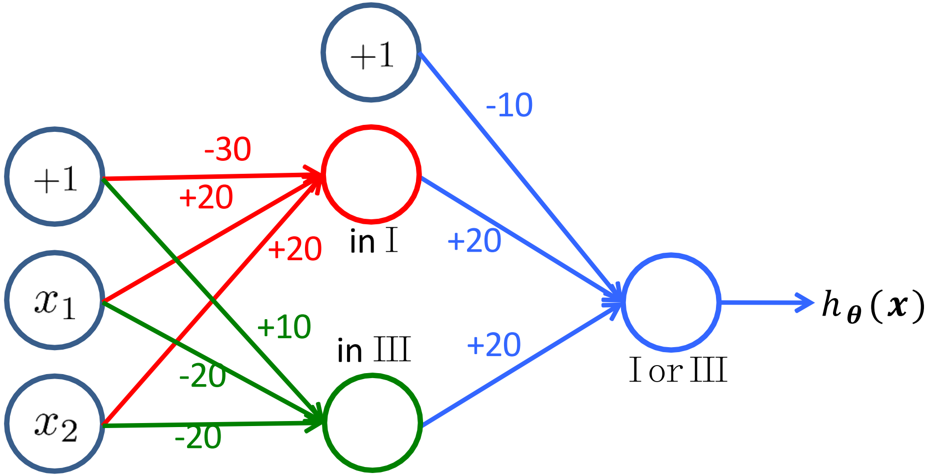


# Combining Representations to Create Non-Linear Functions




Based on example by Andrew Ng