# CS60010: Deep Learning

Spring 2023

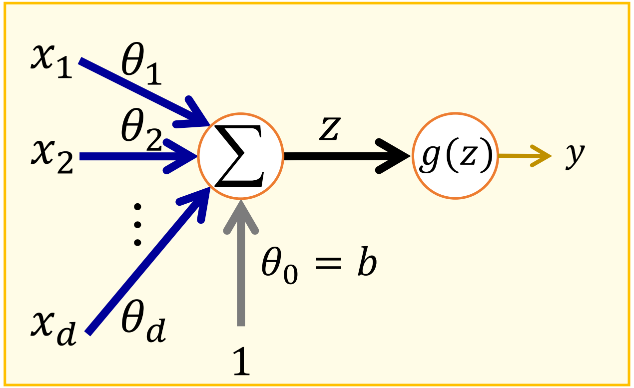
Sudeshna Sarkar

**Multilayer Perceptron - Introduction**

**Sudeshna Sarkar**

20 Jan 2023

# Linear Models



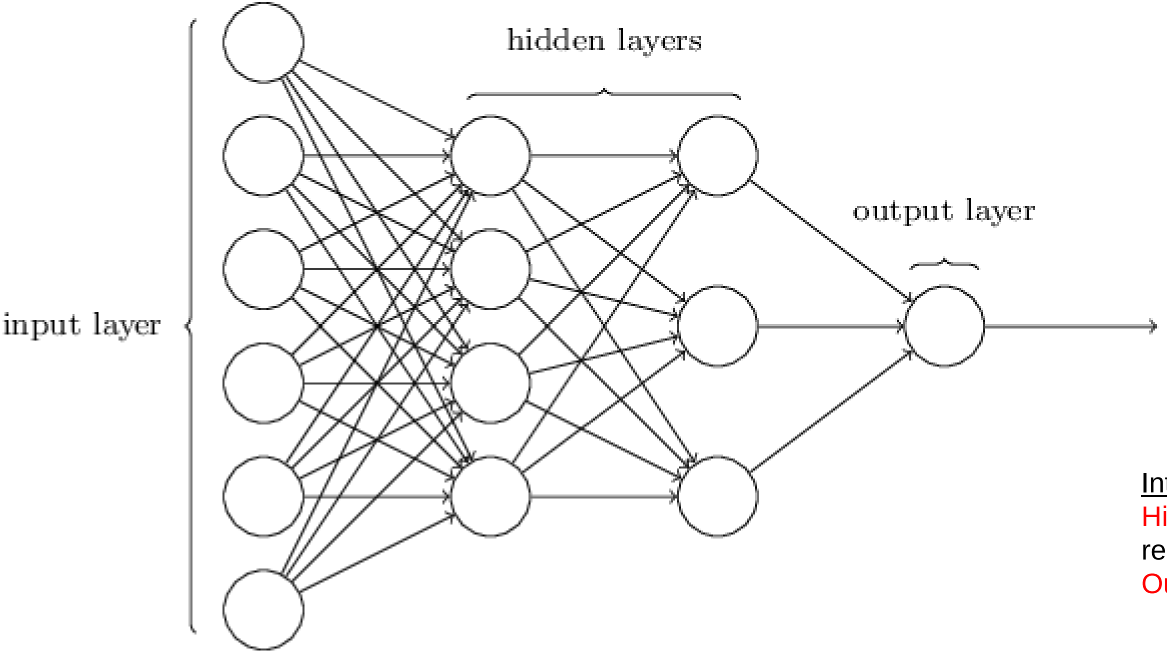

$\boldsymbol{w} = [\theta_1\ \theta_2 \dots \theta_d]^T$ and $\boldsymbol{x} = [x_1\ x_2 \dots x_d]^T$

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

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

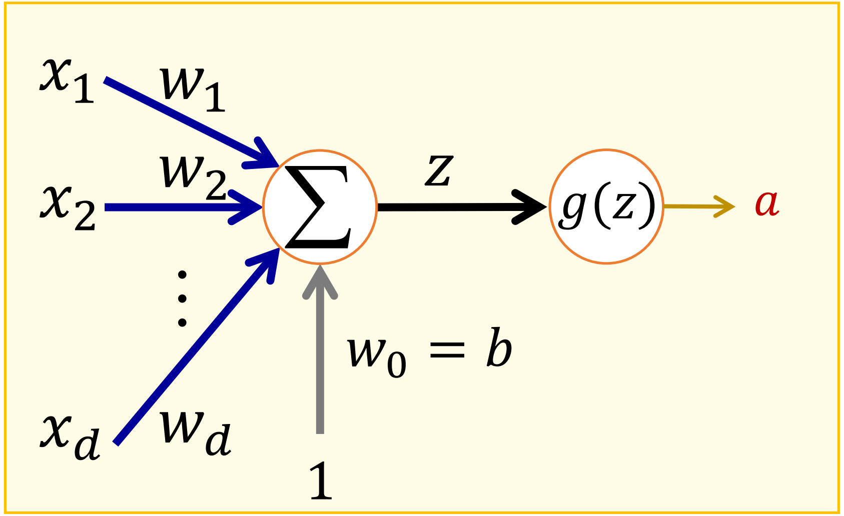
- Regression: $g(z) = z$
- Classification:
  - Binary: $g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$
  - Multi-class

# multilayer perceptrons




<u>Intuition</u>:-
Hidden Layer: Extracts better representation of the input data
Output layer: Does the classification

# Neuron




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

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

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

<u>Terminologies</u>:-

$\boldsymbol{x}$**:** 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 $\hat{y}_i = p(y = i|x)$

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

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

# Loss Functions

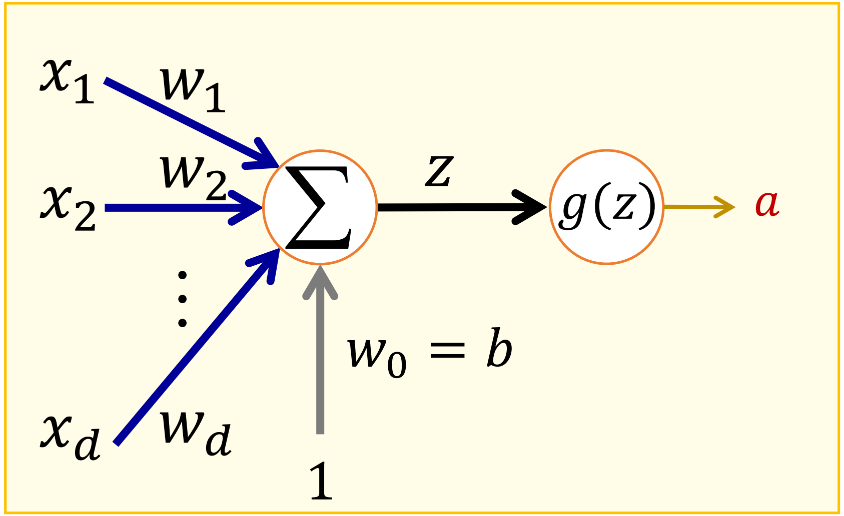
## Regression

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

## Classification

- Cross entropy: : $L(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = -\sum_k y_k \log \hat{y}_k$

# Artificial Neuron – hidden unit




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

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

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

Terminologies

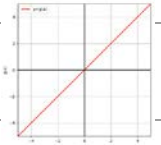
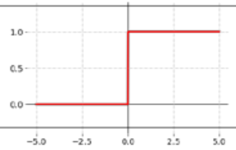
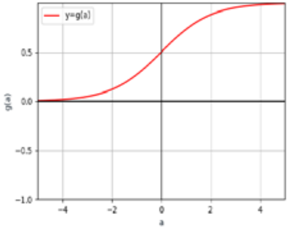
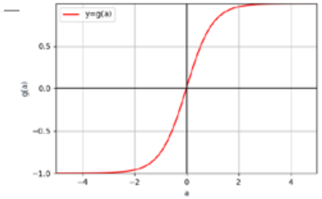
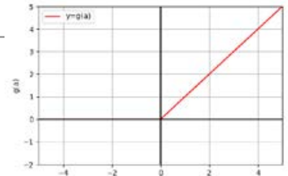
$\boldsymbol{x}$: input, $\boldsymbol{w}$: weights, $\boldsymbol{b}$: bias

$z$: pre-activation (input activation)

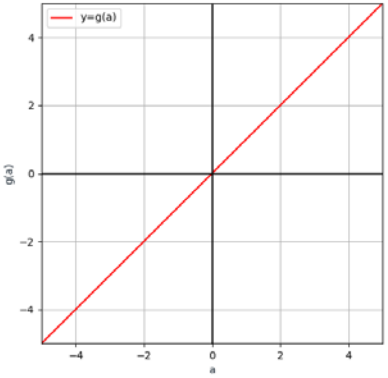
$g$: activation function

a: activation at hidden units

# Common Activation Functions

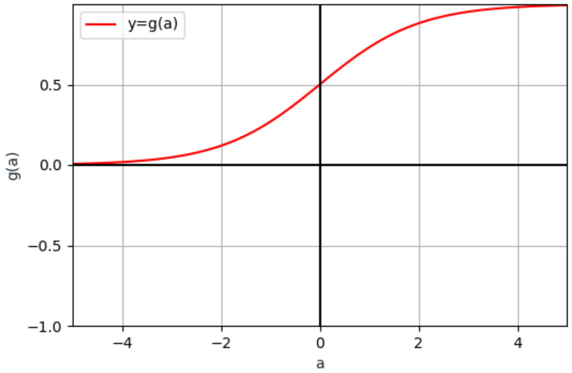
| Name | Function | Gradient | Graph |
|---|---|---|---|
| Linear | $a$ | $1$ |  |
| Binary step | $\mathrm{sign}(a)$ | $g'(a) = \begin{cases} 0, & a \neq 0 \\ NA, & a = 0 \end{cases}$ |  |
| Sigmoid | $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$ | $g'(a)$ $= g(a)(1 - g(a))$ |  |
| Tanh | $\tanh(a)$ $= \dfrac{\exp(a)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

# Common Activation Functions



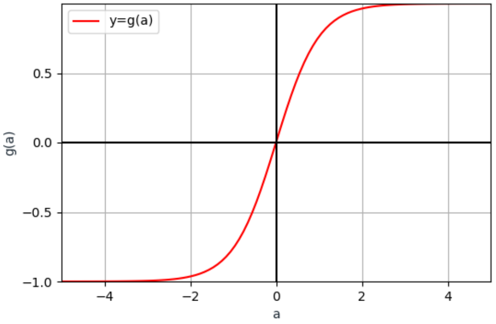

Linear activation function

- $g(a) = a$
- Unbounded
- $g'(a) = 1$



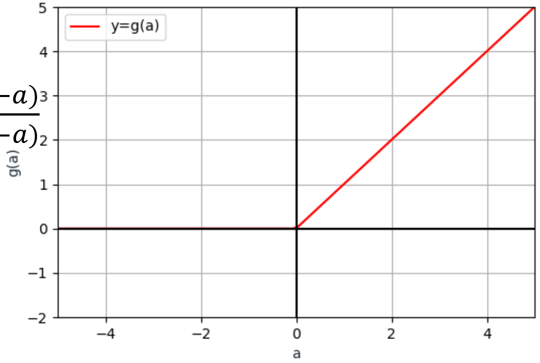

Sigmoid activation function

- $g(a) = \sigma(a) = \frac{1}{1+\exp(-a)}$
- 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
- $g'(a) = 1 - g^2(a)$



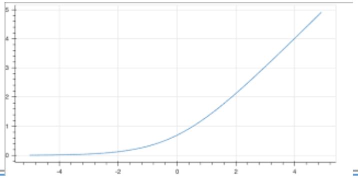
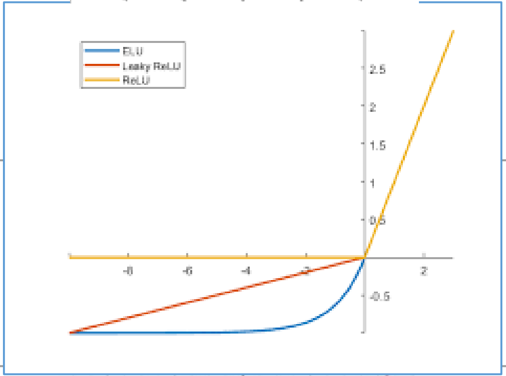
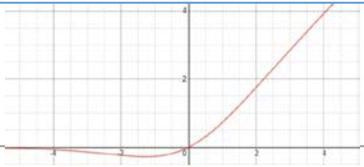

ReLU activation function

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

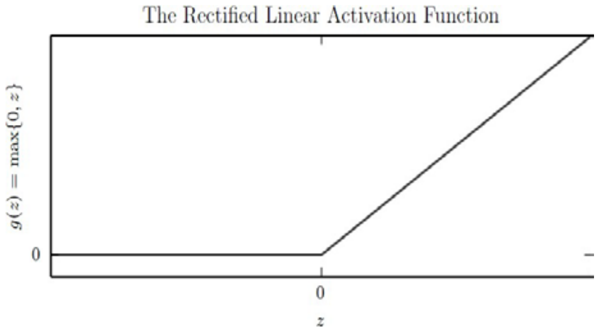
# Activation Functions for Hidden Nodes

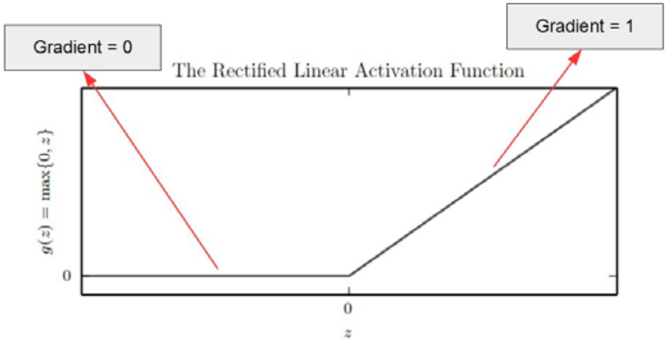
| Name | Function | Gradient | Graph |
|---|---|---|---|
| Sigmoid | $\sigma(z) = \frac{1}{1+\exp(-z)}$ | $g'(z)$ $= g(z)(1 - g(z))$ | |
| Tanh | $\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$ | $g'^{(z)} = 1 - g^2(z)$ | |
| ReLU | $g(z) = \max(0, z)$ | $g'(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$ | |
| Softplus | $g(z) = \ln(1 + e^z)$ | | |

# More activation functions

| Name | Function | Gradient | Graph |
|---|---|---|---|
| Softplus | $g(z) = \ln(1 + e^z)$ | $g'(z) = \dfrac{1}{1 + e^{-z}}$ |  |
| Leaky Relu | $g(z) = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$ | $g'(z) = \begin{cases} \alpha, & z < 0 \\ 1, & z \geq 0 \end{cases}$ |  |
| ELU | $g(z) = \begin{cases} z, & z > 0 \\ \alpha(e^z - 1), & z \leq 0 \end{cases}$ | $g'(z) = \begin{cases} 1, & z > 0 \\ \alpha(e^z), & z \leq 0 \end{cases}$ | |
| swish | $g(z) = z \cdot \sigma(\beta z)$ | $g'(z) = \beta g(\beta z) + \sigma(\beta z)(1 - \beta g(\beta z))$ |  |

# Rectified Linear Units




- Activation function: $g(z) = \max\{0, z\}$ with $z \in \mathbb{R}$
- Give large and *consistent* gradients when active
- **Good practice:** Initialize **b** to a small positive value (e.g. 0.1) Ensures units are initially active for most inputs and derivatives can pass through

Positives:

- Gives large and *consistent* gradients (does not saturate) when active
- Efficient to optimize, converges much faster than sigmoid or tanh

Negatives:

- Non zero centered output
- Units ”die” i.e. when inactive they will never update

# Swish

$$f(x) = x * \sigma(\beta x)$$

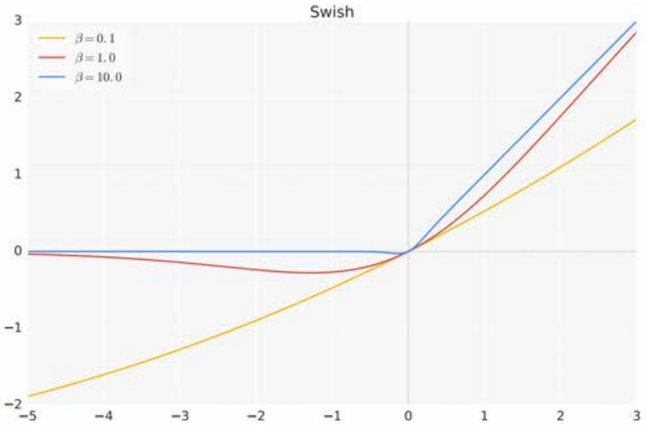



Figure 4: The Swish activation function.

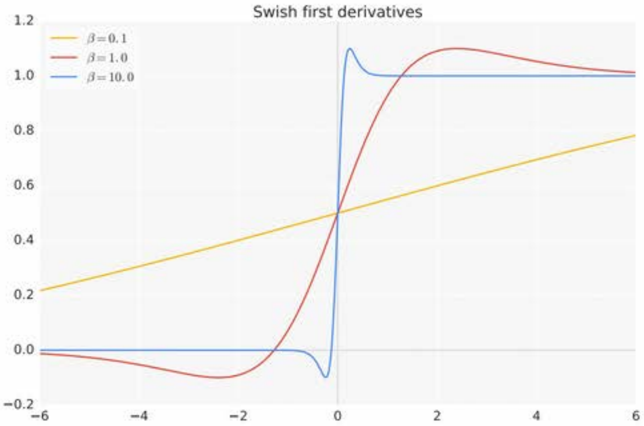


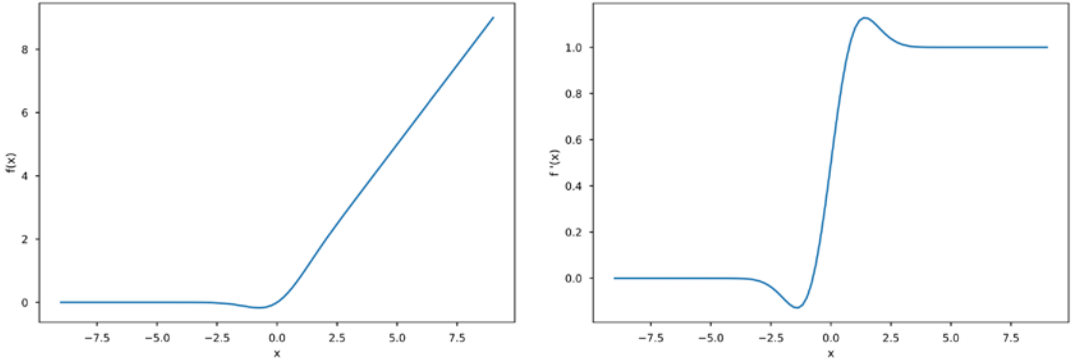

Figure 5: First derivatives of Swish.

# GELU

$$\mathrm{gelu}(x) = x\,\Pr(X \le x) \qquad X \sim \mathcal{N}(0,1)$$

$$\mathrm{gelu}(x) = \frac{1}{2}x\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

GELU function and it's Derivative

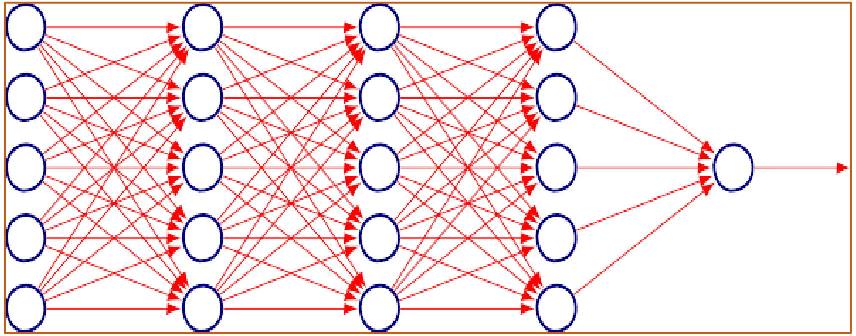


# Feedforward Networks and Backpropagation

# Introduction

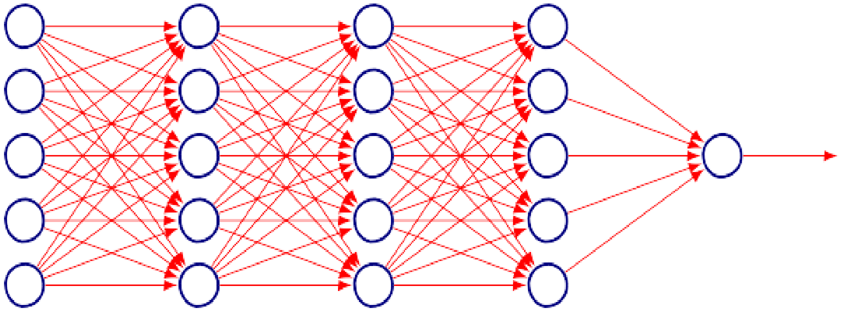
- Goal: Approximate some unknown ideal function $f^*: X \to Y$
- Ideal classifier: $y = f^*(x)$ for $(x, y)$
- Feedforward Network: Define parametric mapping $y = f(x; \theta)$
- Learn parameters $\theta$ to get a good approximation to $f^*$ from training data
- Function $f$ is a composition of many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

# 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

# Advantages of Depth