

McGill **Artificial Intelligence** Society



# Lecture 3: Neural Networks

Slides based off of Machine Learning at Berkeley  
<https://github.com/mlberkeley/Machine-Learning-Decal-Fall-2018>



# Today's Lesson Plan

Linear and Logistic Regression Review

Motivation

The Perceptron

Feed-forward Neural Networks

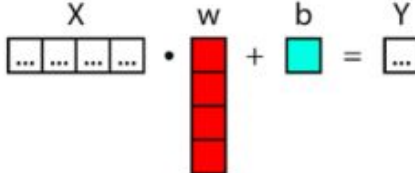
Learning In NNs

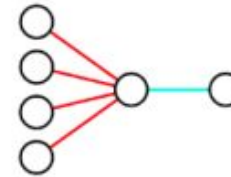
Coding Demo



# Linear Regression

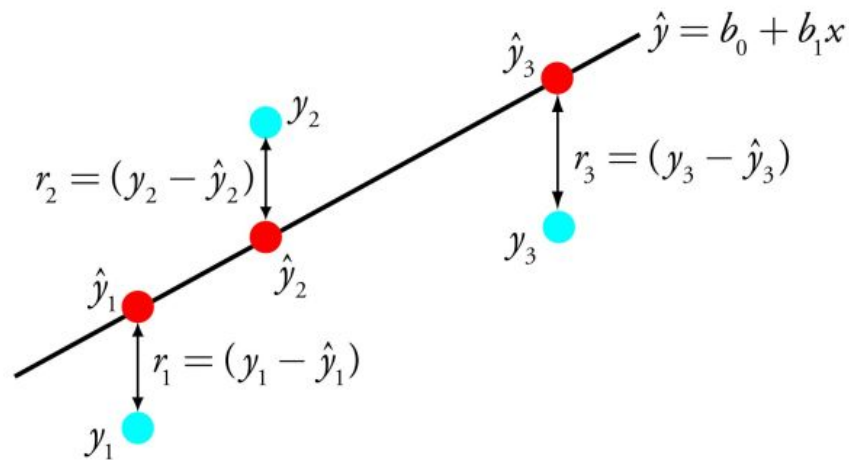
$$Y = f(X, w_1, b_1)$$

$$f(x, w, b) = x \cdot w + b$$




Let  $\hat{y}_i = h(x) = b_0 + b_1 x$

$$\min J(b_0, b_1)$$



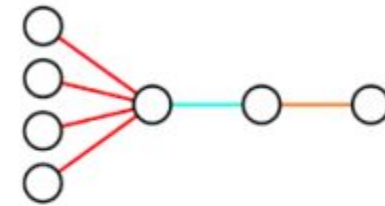
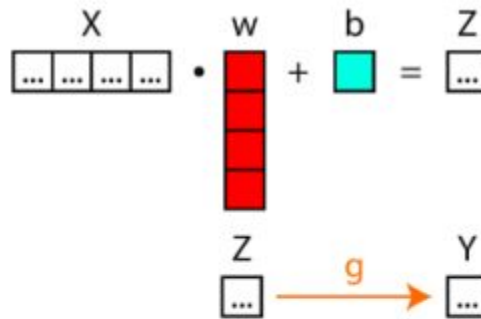
$$RSS(w) = \sum_{i=1}^N (y_i - w^T x_i)^2$$

$$J(b_0, b_1) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

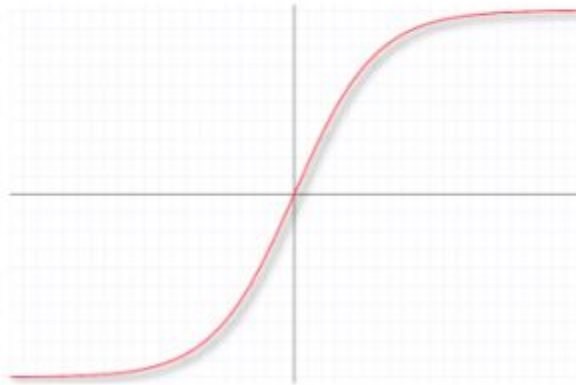
# Logistic Regression

$$Z = f(X, w_1, b_1)$$

$$Y = g(Z)$$



$$g(x) = \frac{1}{1 + e^{-x}}$$



$$J(b) = - \sum_{i=1}^m \left( y^{(i)} \cdot \ln z^{(i)} + (1 - y^{(i)}) \cdot \ln (1 - z^{(i)}) \right)$$

# Motivation

What I see

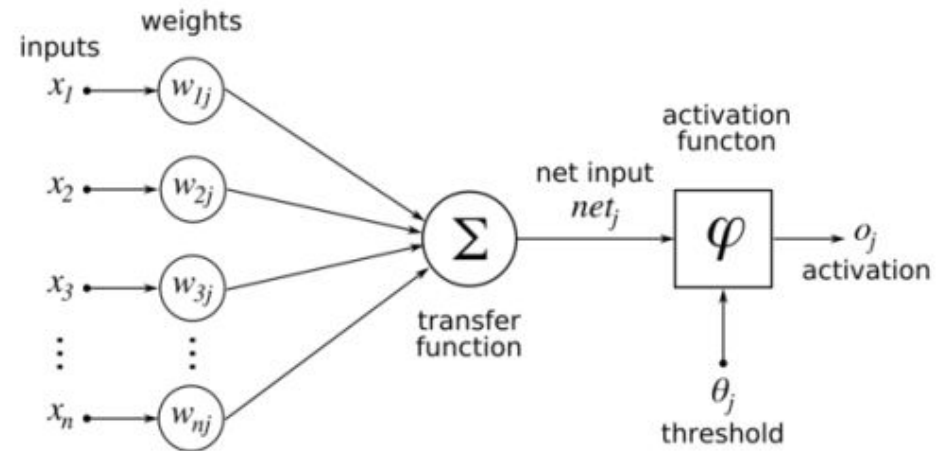
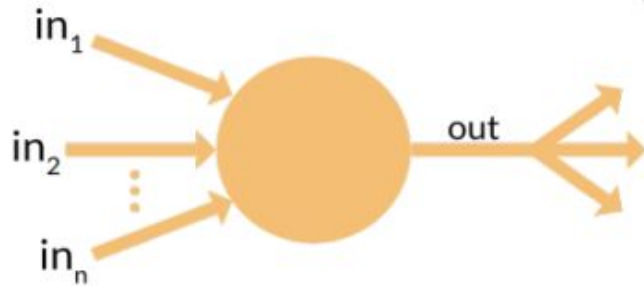
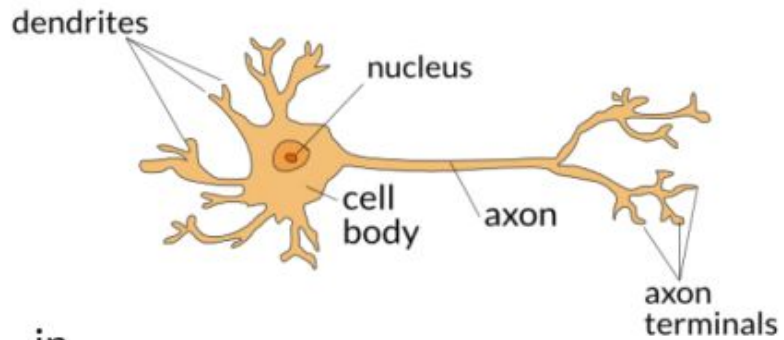


What a computer sees

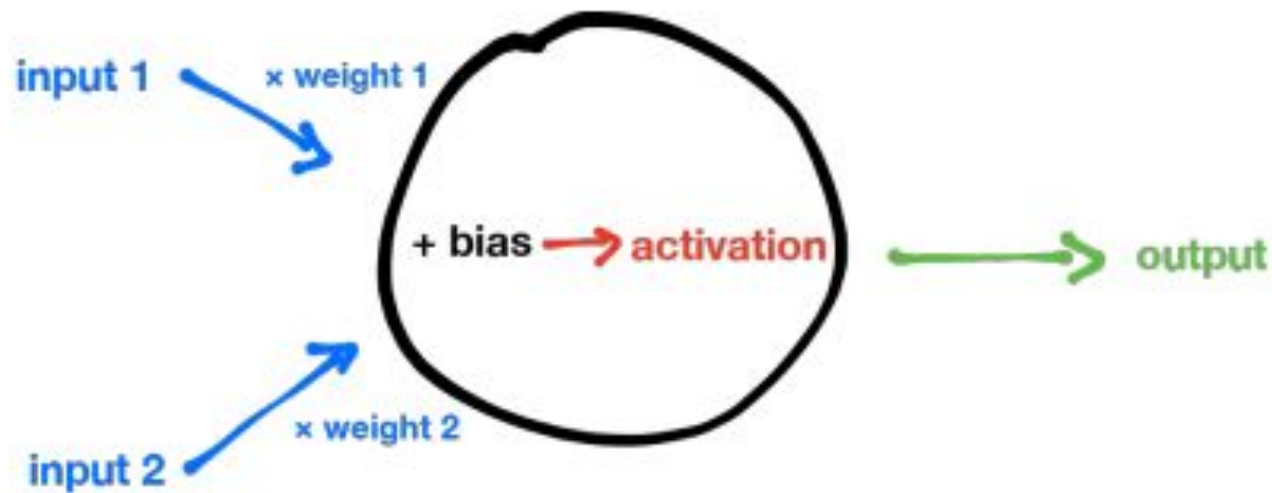
06	02	22	97	38	18	00	40	00	78	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	40	87	17	40	98	43	69	48	04	56	42	00
81	49	31	73	55	79	14	29	93	71	40	47	53	88	30	03	49	13	36	45
52	70	95	23	04	40	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	47	43	89	41	92	36	54	22	40	40	28	46	33	13	80
24	47	32	40	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	44	23	47	10	26	38	40	47	59	34	70	66	18	38	44	70
47	24	20	48	02	42	12	20	95	43	94	39	43	08	40	91	46	49	94	21
24	55	38	05	46	73	99	26	97	17	78	78	96	83	14	88	34	89	43	72
21	36	23	09	75	00	74	44	20	45	35	14	00	41	33	97	34	31	33	95
78	17	53	28	22	75	31	47	15	94	03	80	04	42	16	14	09	53	54	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	34	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	40	21	58	51	54	17	58
19	40	81	48	05	94	47	49	28	73	92	13	86	32	17	77	04	89	55	40
04	32	08	83	97	35	99	14	07	97	57	32	14	26	26	79	33	27	90	44
88	36	48	87	57	42	20	72	03	46	33	47	46	55	12	32	43	93	53	49
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	42	74	36
20	49	34	41	72	30	23	88	34	42	99	49	42	47	59	85	74	04	34	14
20	73	38	29	78	31	90	01	74	31	49	71	48	84	81	16	23	57	05	54
01	70	54	71	83	51	54	49	16	92	33	48	41	43	52	01	89	19	47	48

- Ability to learn complex, non-linear functions
- Does not impose fixed relationships in data
- Does not assume the inherent data distribution
  - Better at modeling non-constant variance

# Biological Inspiration



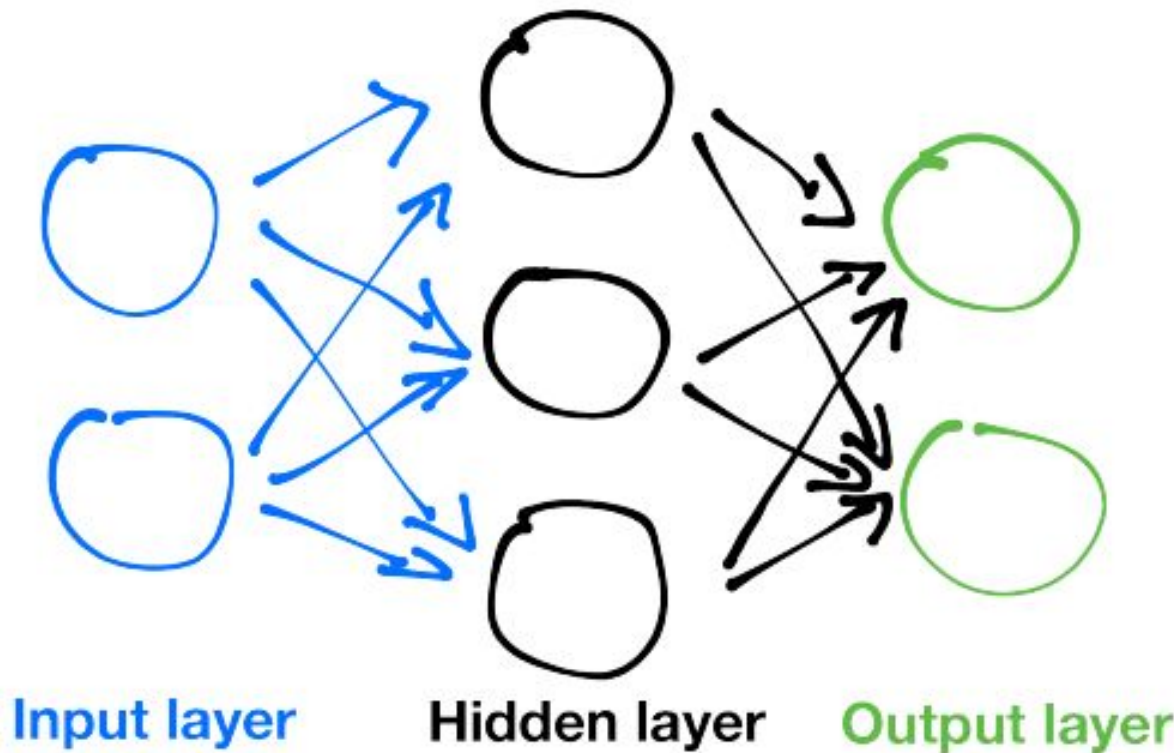
# Biological Inspiration



- Inputs to the neuron are multiplied by weights
- Bias then summed with weighted inputs
- Non-linear activation function applied on  $wTx + b$



## Biological Inspiration - A Layered Architecture



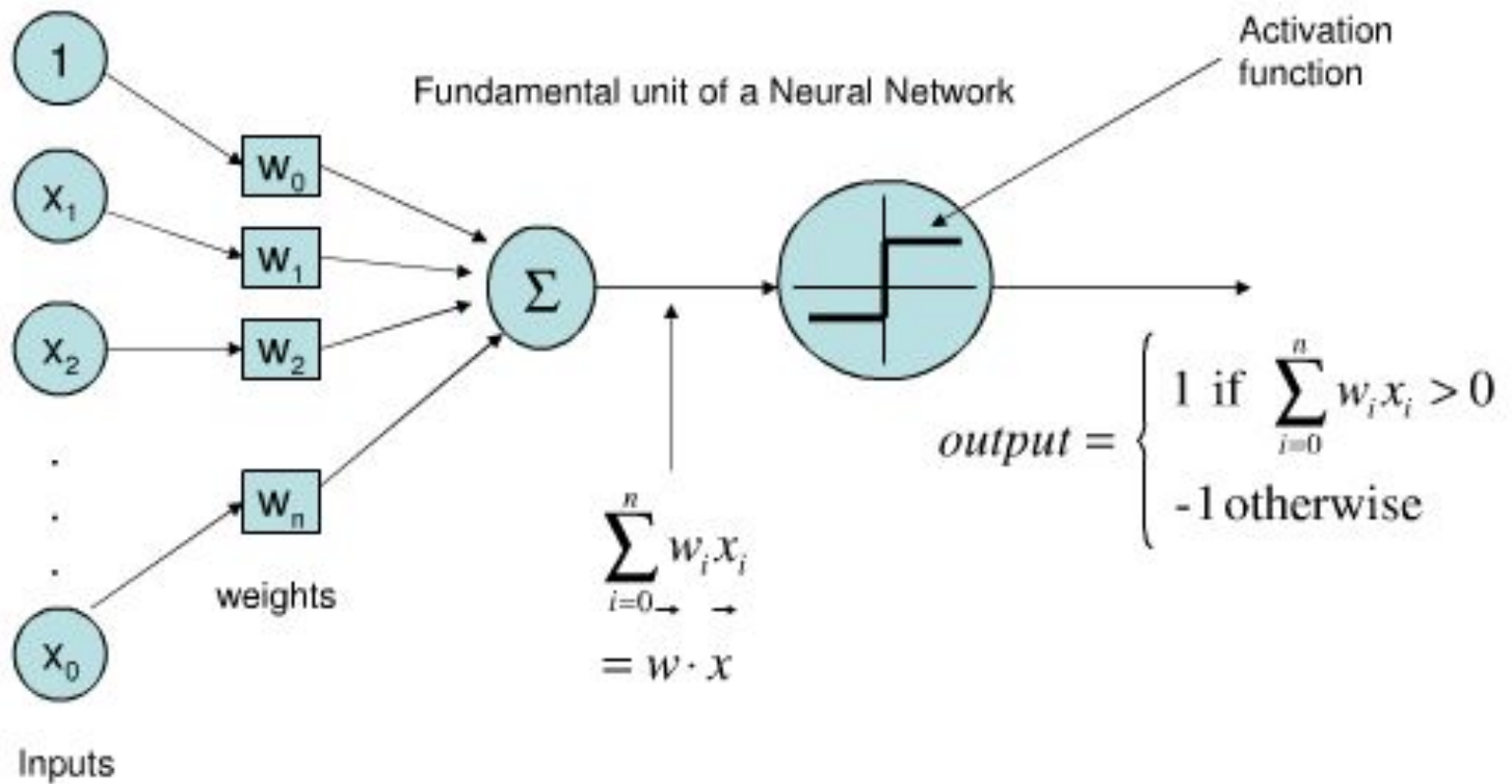
**Input Layer:** takes in input data (size corresponds to input space)

**Hidden Layer:** neurons hidden from view (this is where the magic happens)

**Output Layer:** neurons in this layer provide the output of the network

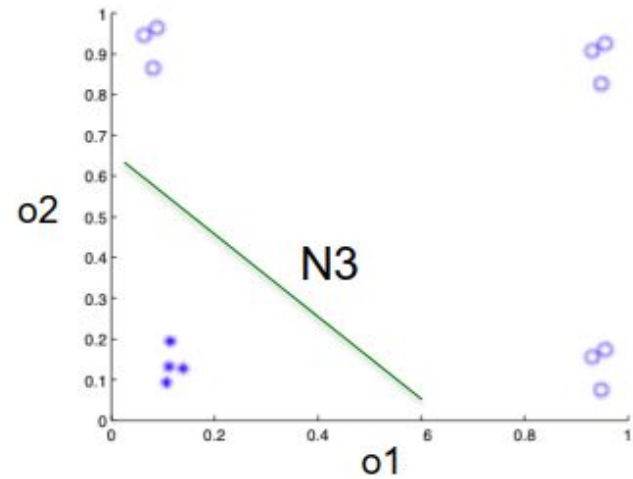
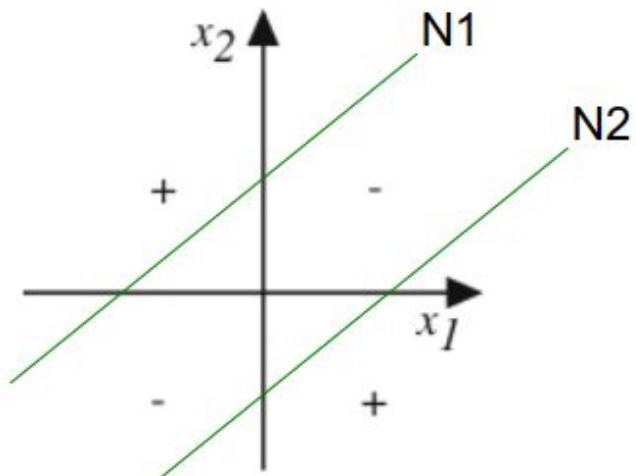


# The Perceptron



[Image Credits From StackExchange](#)

# Perceptron Decision Boundary

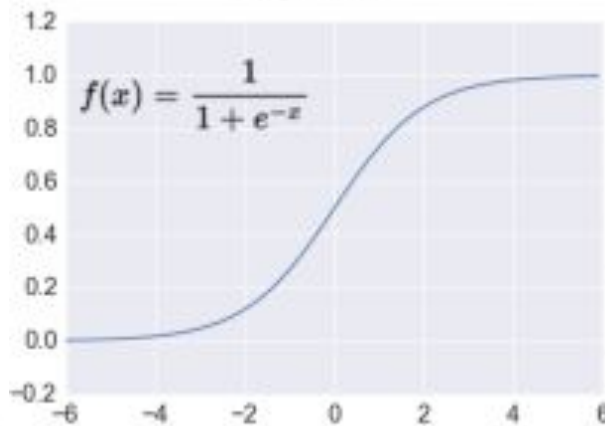


# Activation Functions

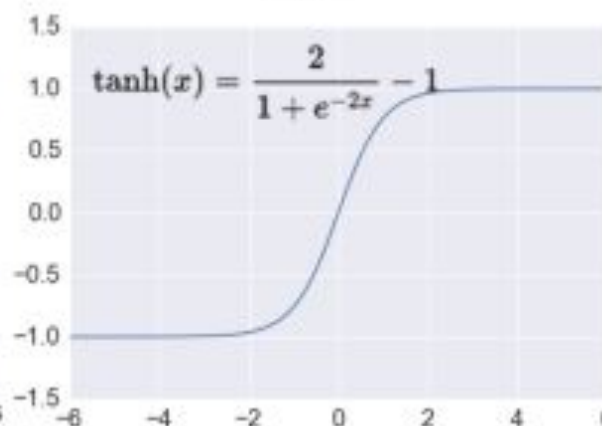
For gradient descent to work, we need activation functions that are:

- Continuous
- Differentiable
- Monotonically Increasing

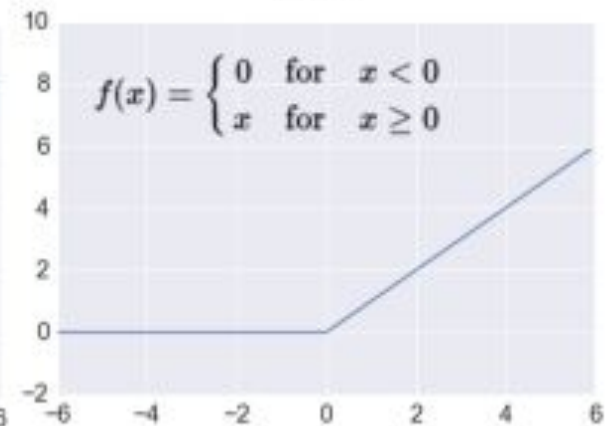
Sigmoid



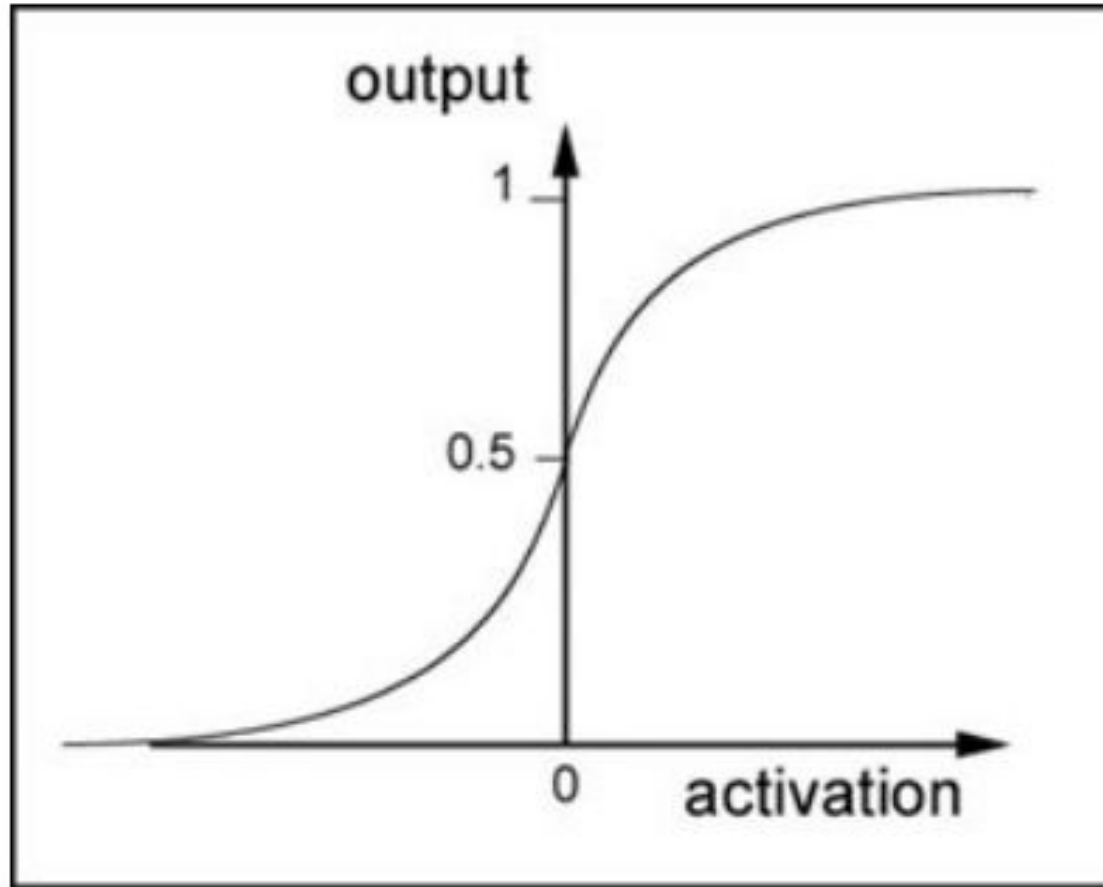
TanH



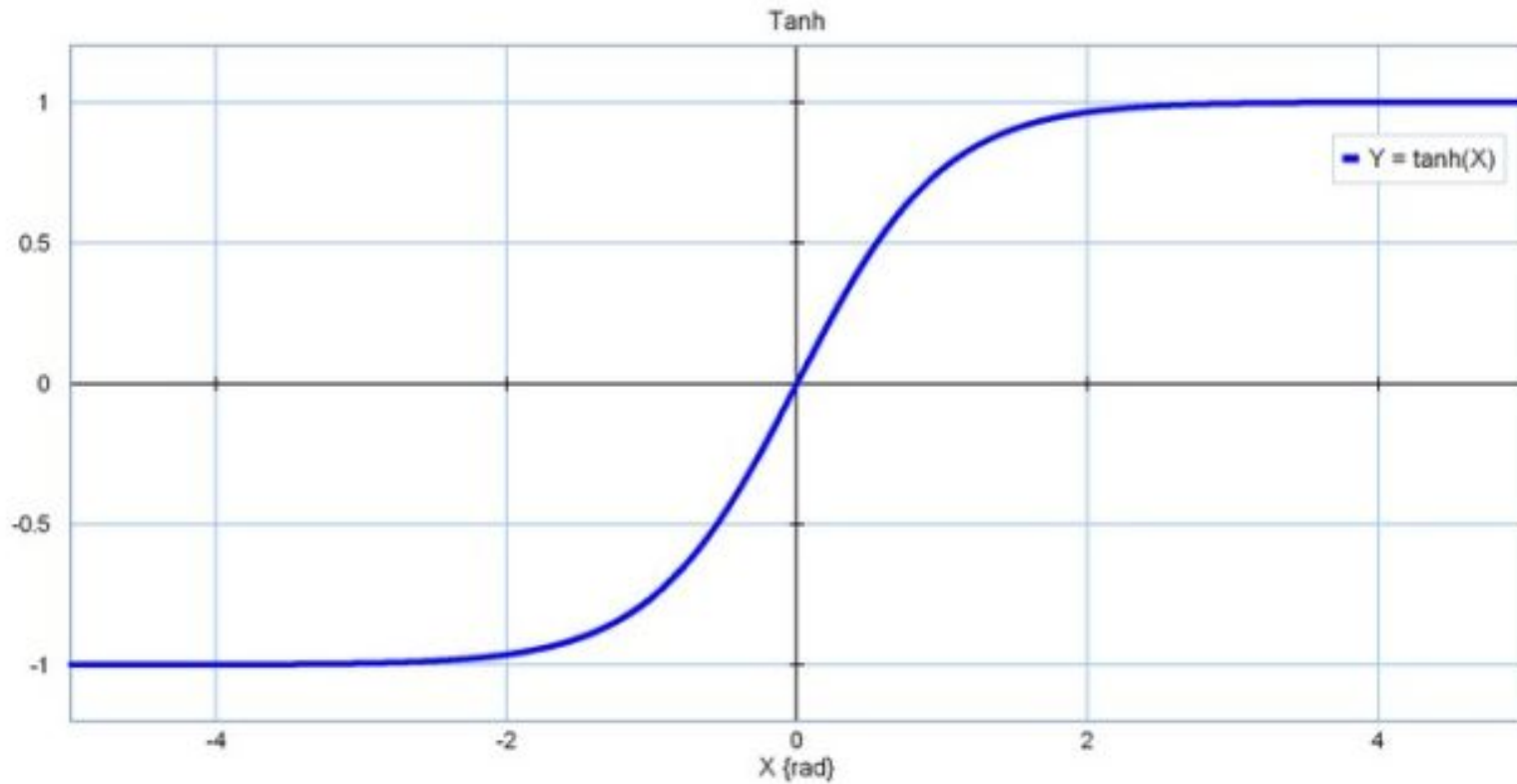
ReLU



## Sigmoid/Logistic

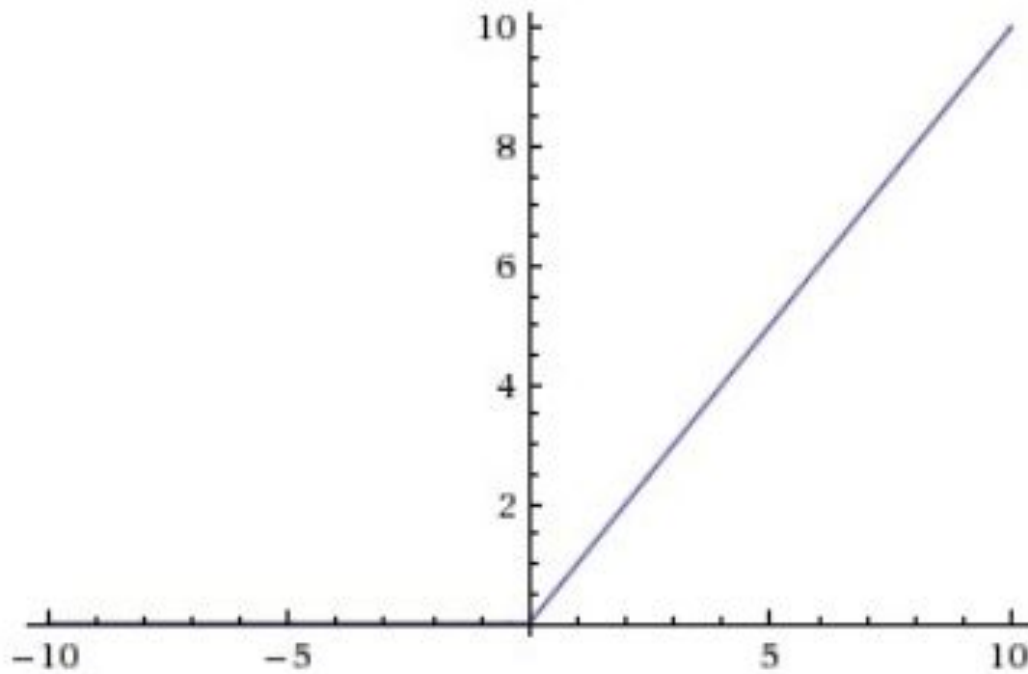


## Hyperbolic tangent (tanh)

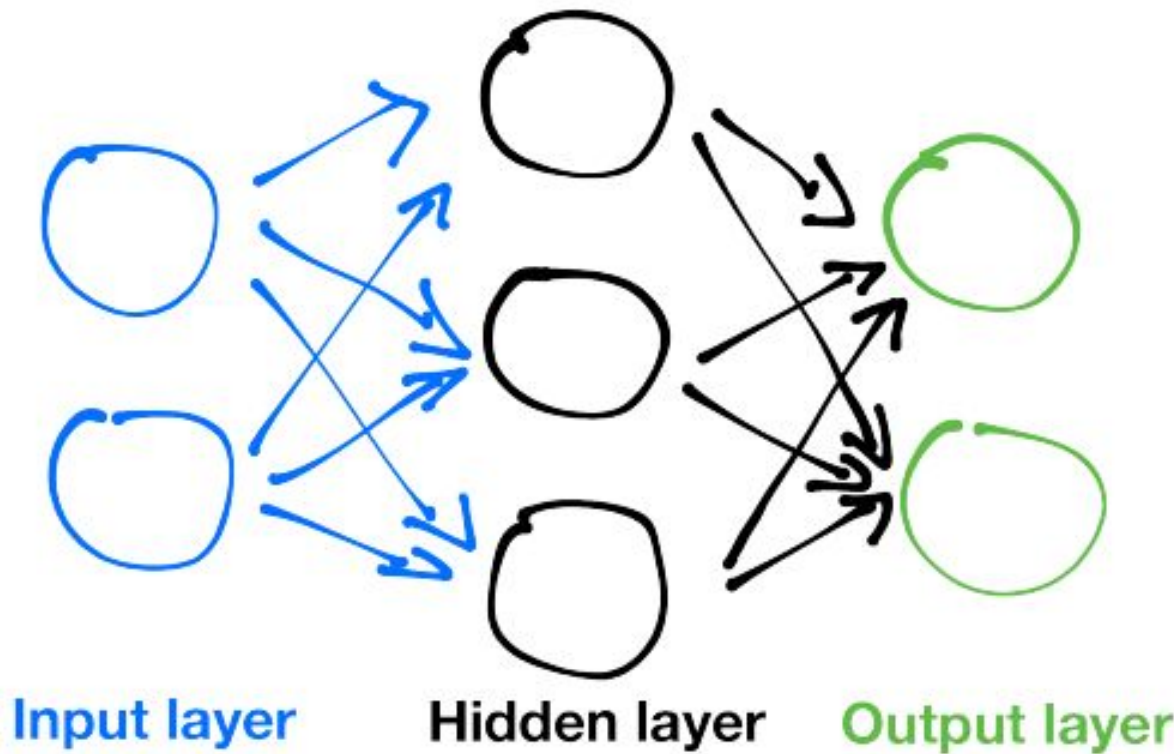


# Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$



# Neural Networks



**Input Layer:** takes in input data (size corresponds to input space)

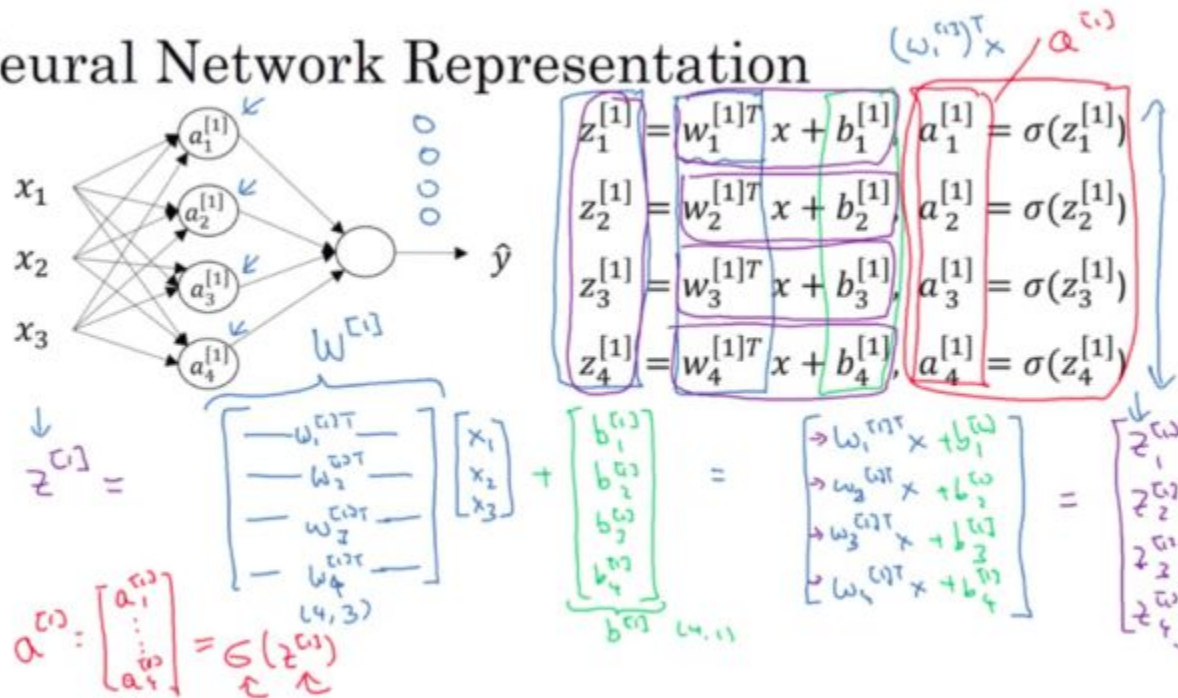
**Hidden Layer:** neurons hidden from view (this is where the magic happens)

**Output Layer:** neurons in this layer provide the output of the network

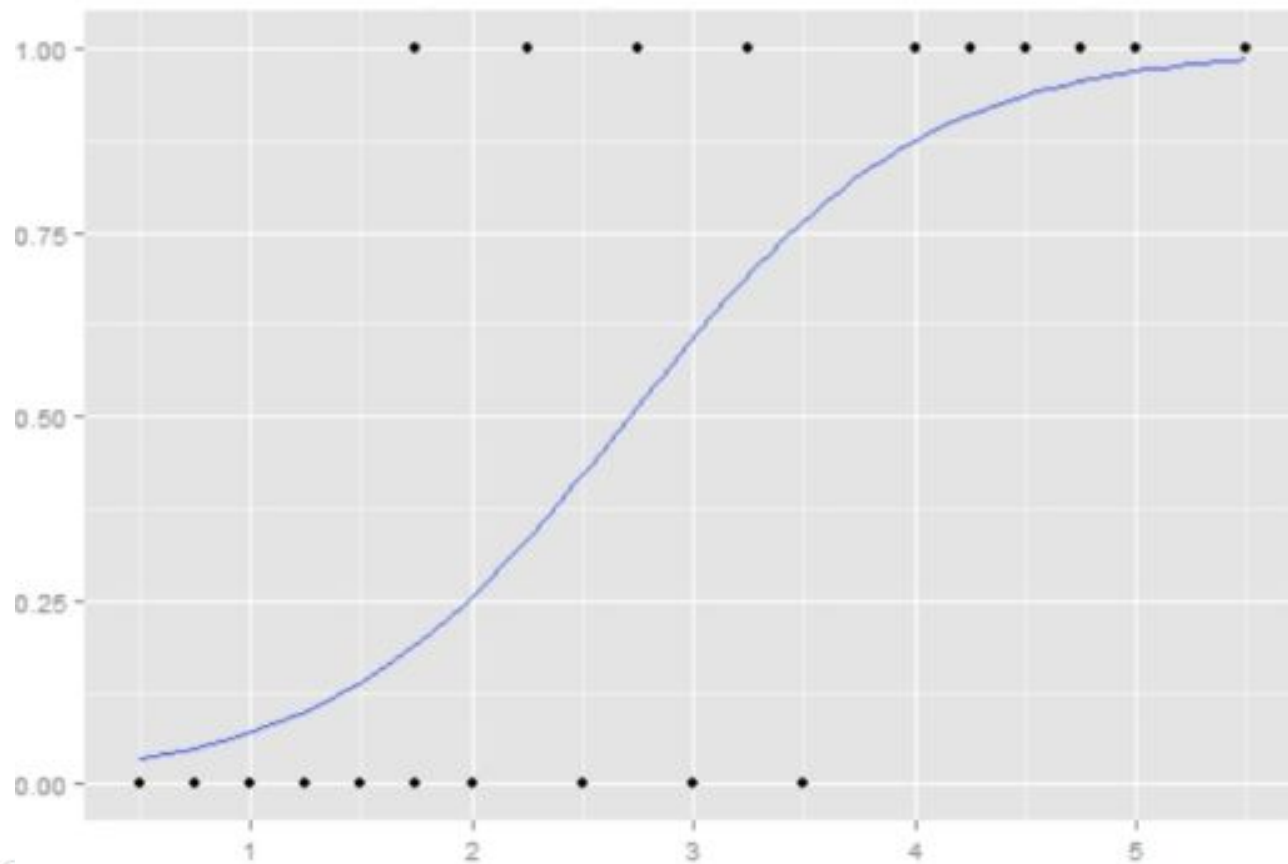


# Neural Networks in Matrix Form

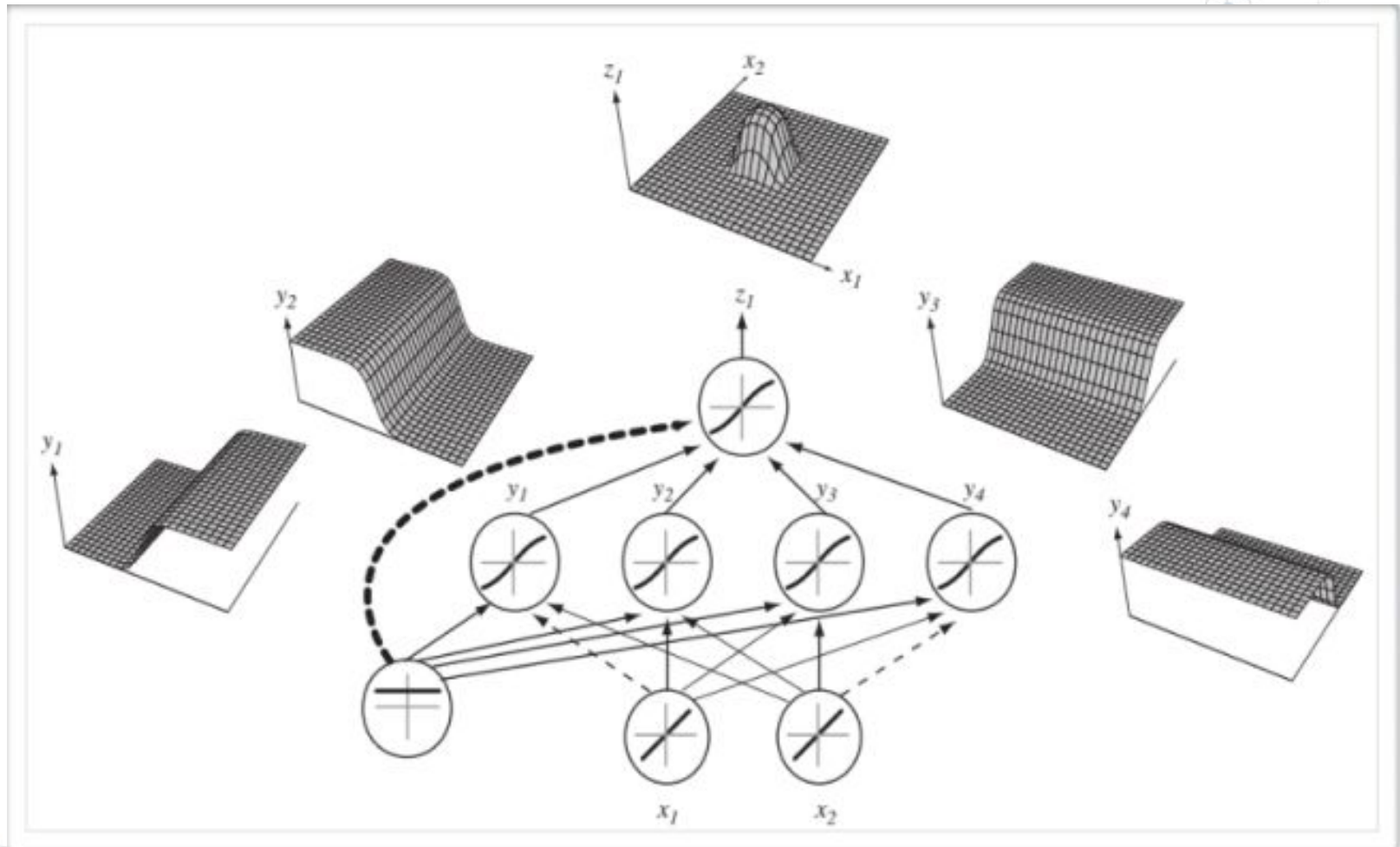
## Neural Network Representation



# Neural Network Expressivity



# Neural Network Expressivity



# Neural Network Expressiveness

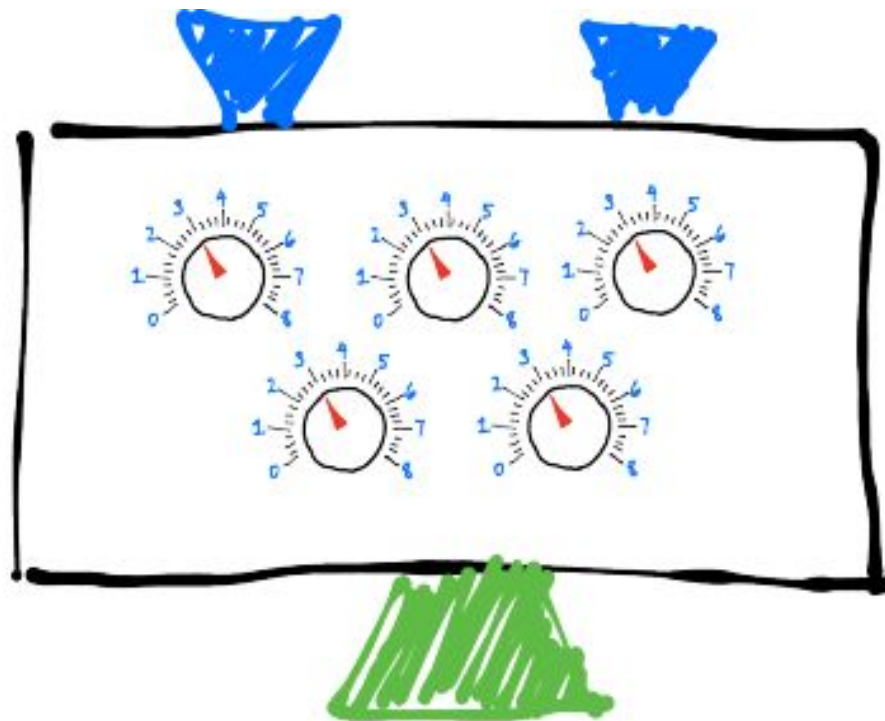
- Neural networks approximate functions without assuming an initial data distribution
- NNs learn function approximations (anything that maps an input, to a single output)

$$f : X \rightarrow Y$$

## Universal Approximation Theorem

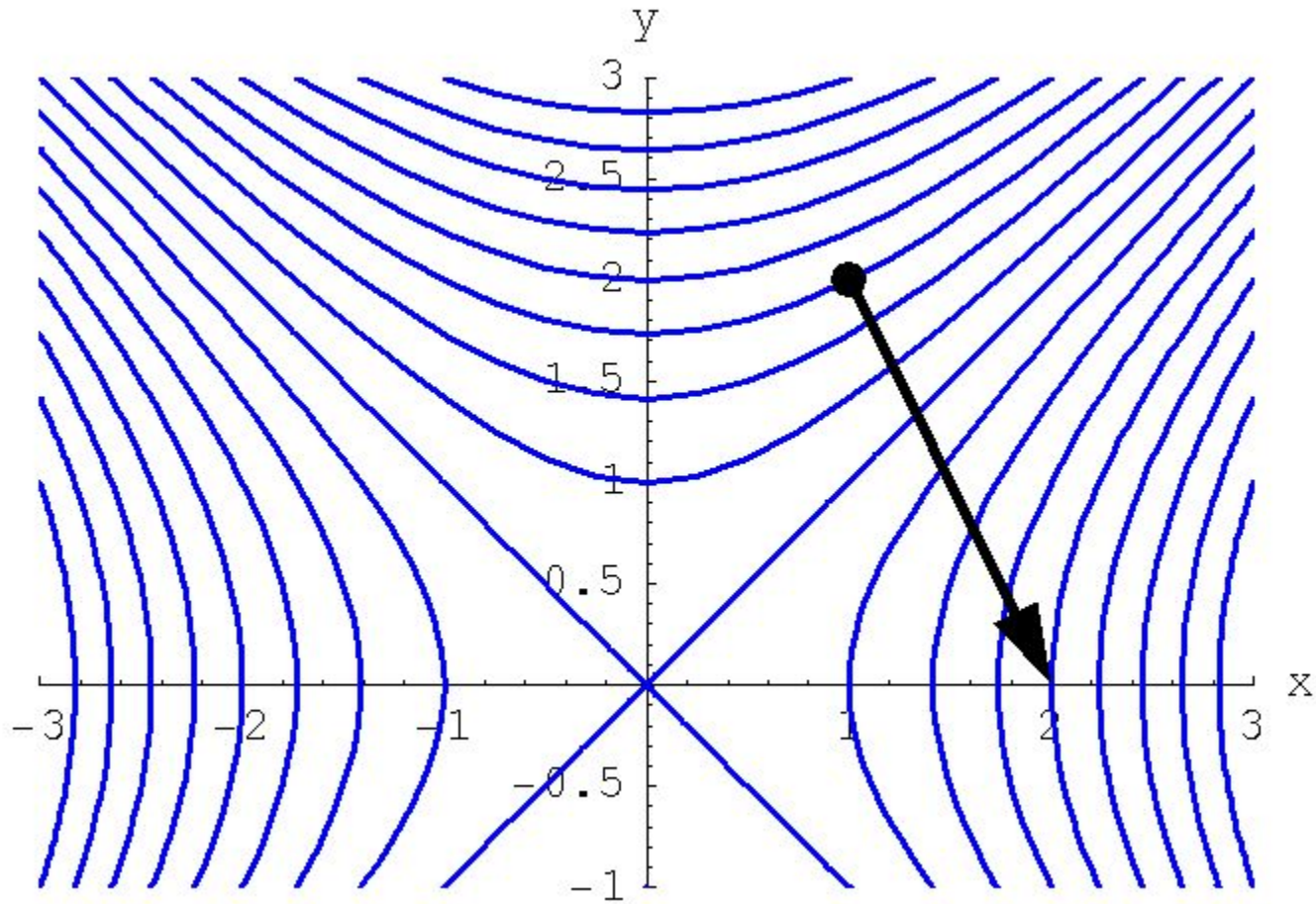
“Every bounded continuous function can be approximated with arbitrary precision by a single-layer neural network”  
(Hornik, 1991)

# Learning in Neural Networks





# Optimization Via Gradient Descent



# Gradient Descent

Define a differentiable, convex objective to minimize:

$$C(\mathbf{x}, \text{parameters}) = \frac{1}{2}(y - \hat{f}(\mathbf{x}))^2$$

Where:

- $y$  is the actual output
- $\hat{f}$  is our hypothesis function (output of the network) dependent on our learned parameters and inputs

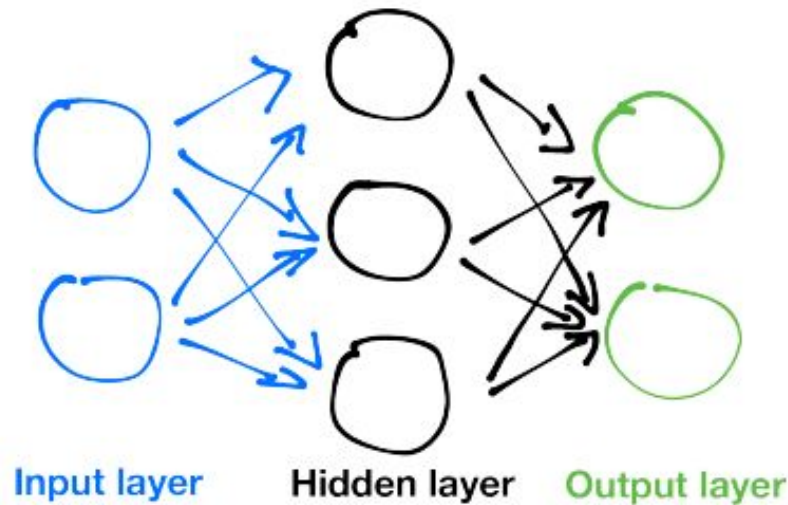
$$J(\mathbf{w}) = \frac{1}{2}(y - h_{\mathbf{w}}(\mathbf{x}))^2 = \frac{1}{2}(y - o_{N+H+1})^2$$

---

We use gradient descent to allow us to find a local minima of  $C$  given the derivative of  $C$  with respect to its parameters



# Backpropagation



Neural network:

$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

# Stochastic Gradient Descent

- Initialize all weights to small random numbers.
- Repeat until convergence:

- Pick a training example.

- Feed example through network to compute output  $o = o_{N+H+1}$ .

- For the output unit, compute the correction:

$$\delta_{N+H+1} \leftarrow o(1 - o)(y - o)$$

- For each hidden unit  $h$ , compute its share of the correction:

$$\delta_h \leftarrow o_h(1 - o_h)w_{N+H+1,h}\delta_{N+H+1}$$

- Update each network weight:

$$w_{h,i} \leftarrow w_{h,i} + \alpha_{h,i}\delta_h x_{h,i}$$

Initialization

Forward  
pass

Backpro-  
pagation

Gradient  
descent

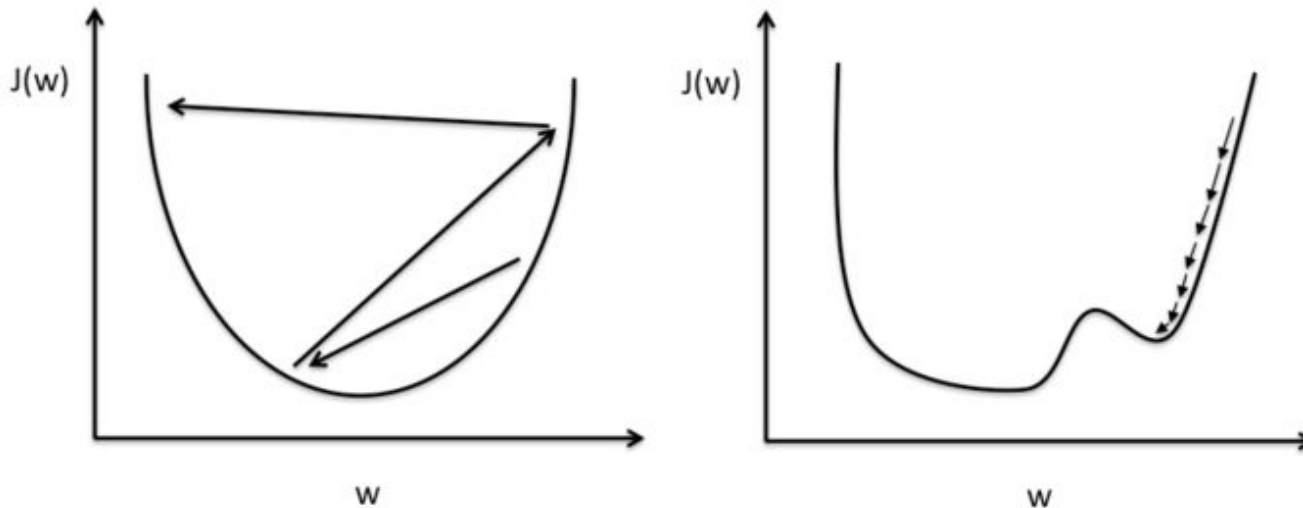
# Forms of Gradient Descent

**Stochastic Gradient Descent:** compute error using a single sample at a time, update weights, repeat

**Batch Gradient Descent:** compute error on all examples, update weights based on error, repeat

**Mini-batch Gradient Descent:** randomly select a subset from the training data, calculate error on subset, update weights, repeat

## Picking $\alpha$ - Adaptive Learning Rates



ADAM, RMSProp, Adagrad, etc.

[More Information on Stochastic Gradient Descent and Different Optimization Methods](#)



# Coding Demo

# Thanks!

## Any questions?

Reminders:

Homework 2 due before next lecture.

