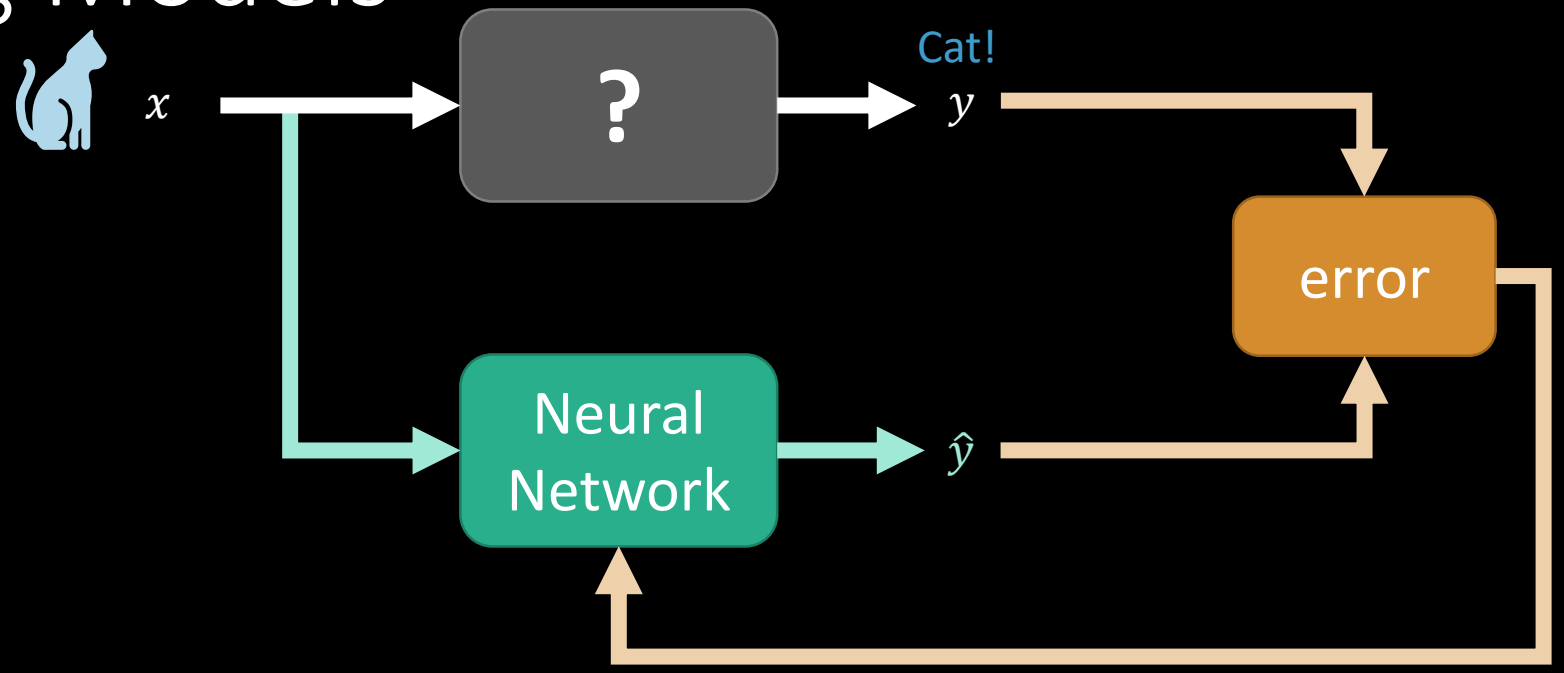
Neural Networks I

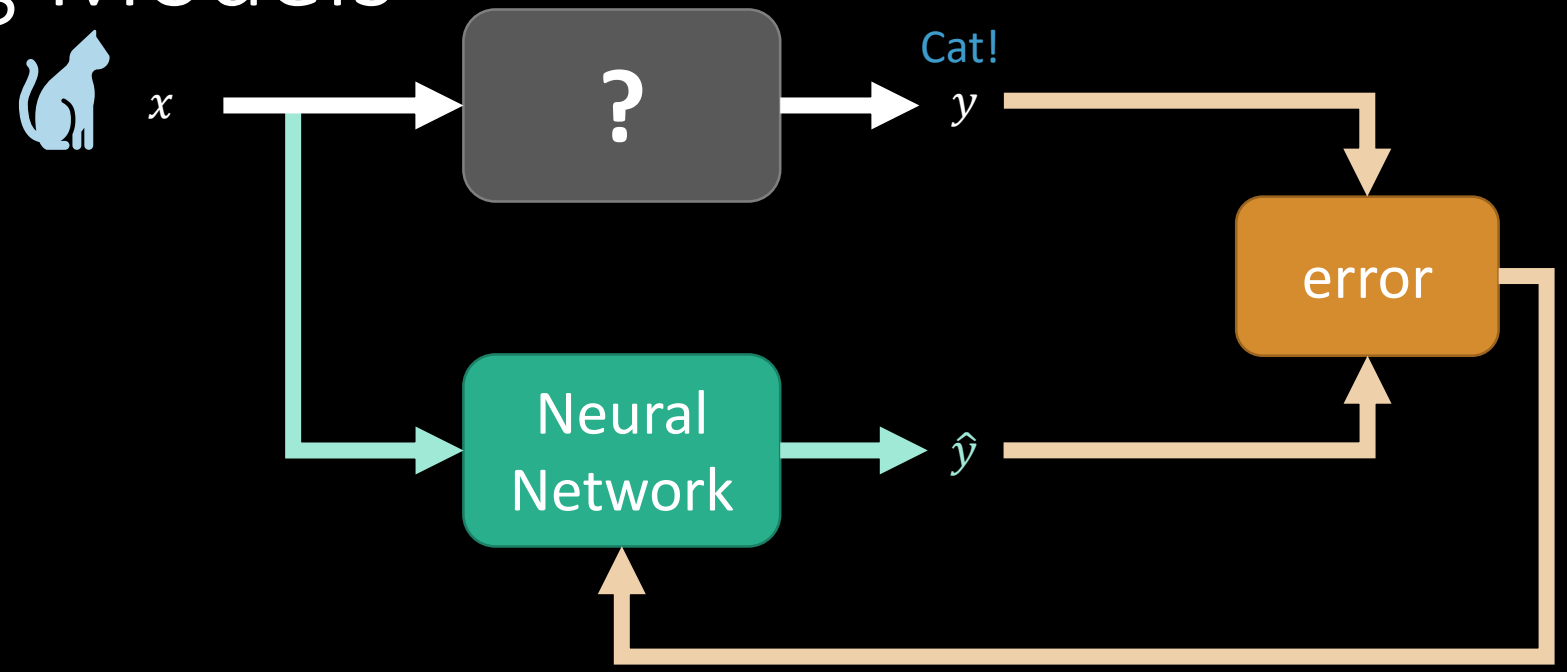
ECE 498: Safe Autonomy

Katie DC

Learning Models



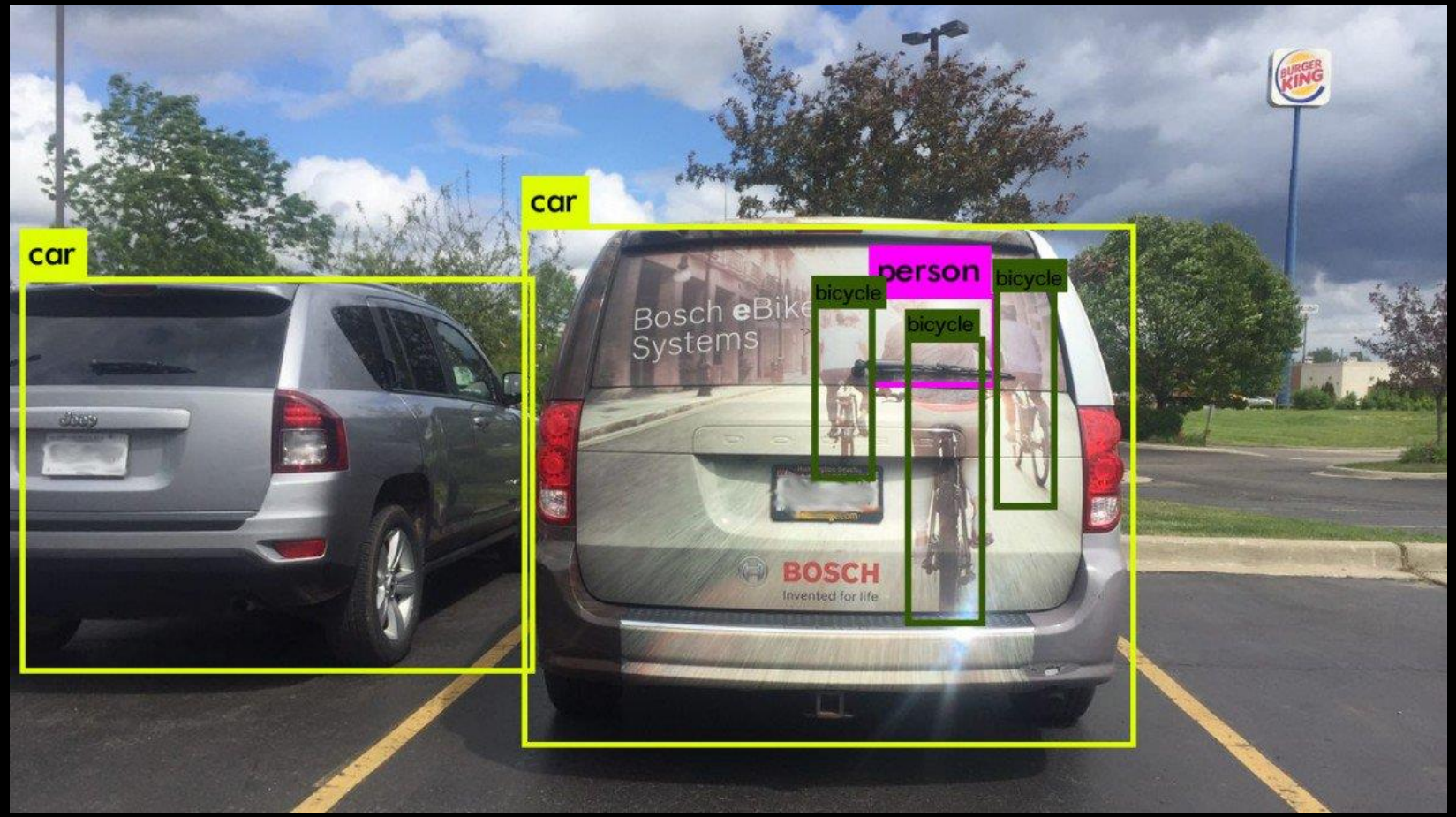
Learning Models



Neural Networks (or learned models) are good when:

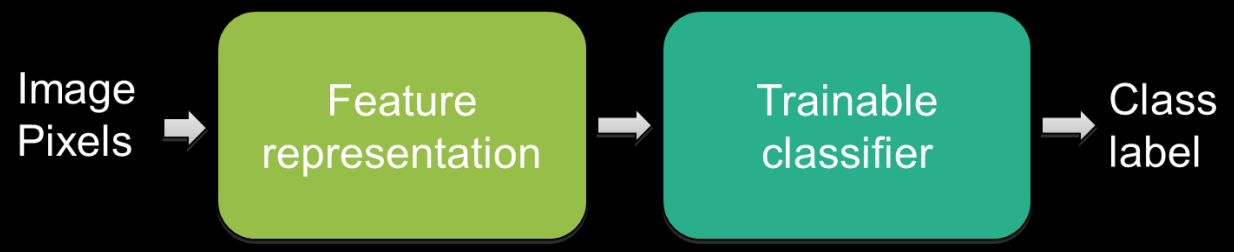
- Your system needs to learn and adapt
- Original is highly nonlinear / multi-variable
- Physics / model based approaches are not available or are too computationally expensive

What is image recognition?

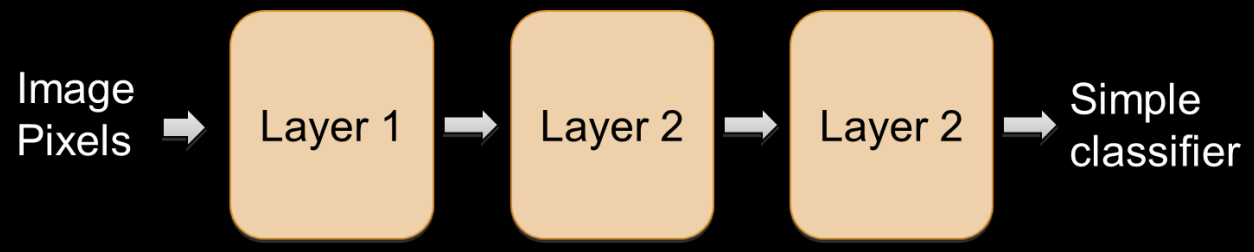


From Shallow to Deep Learning

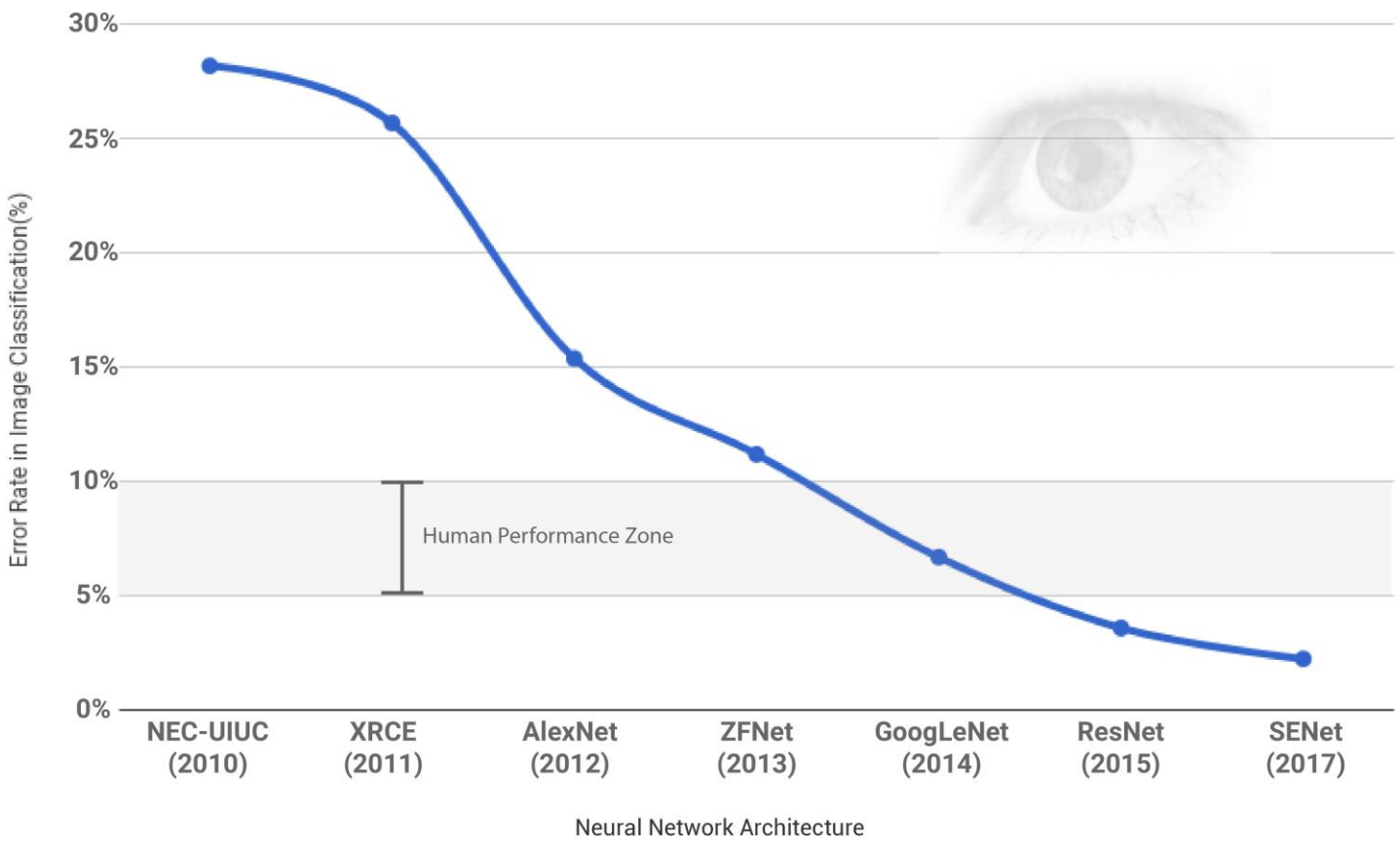
Traditional "Shallow" Pipeline



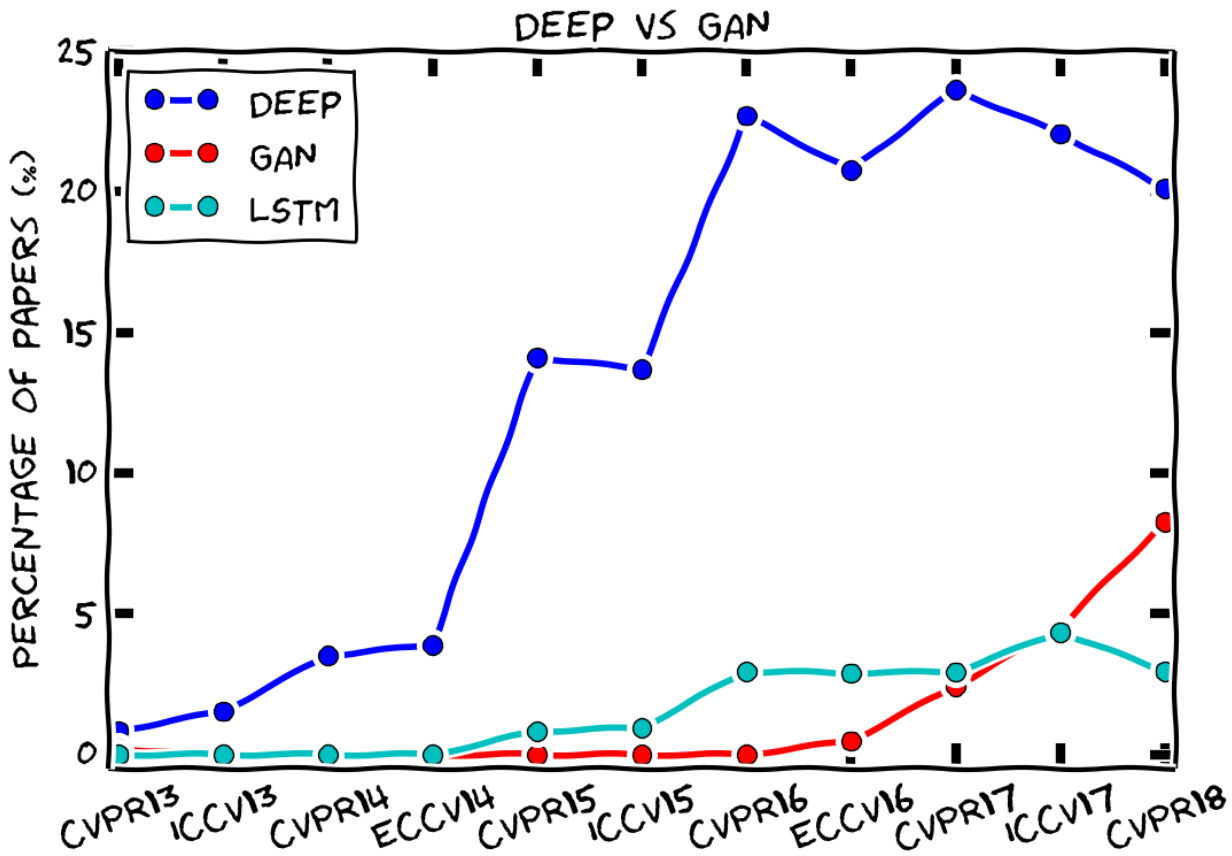
"Deep" Recognition Pipeline



Classification Improvements



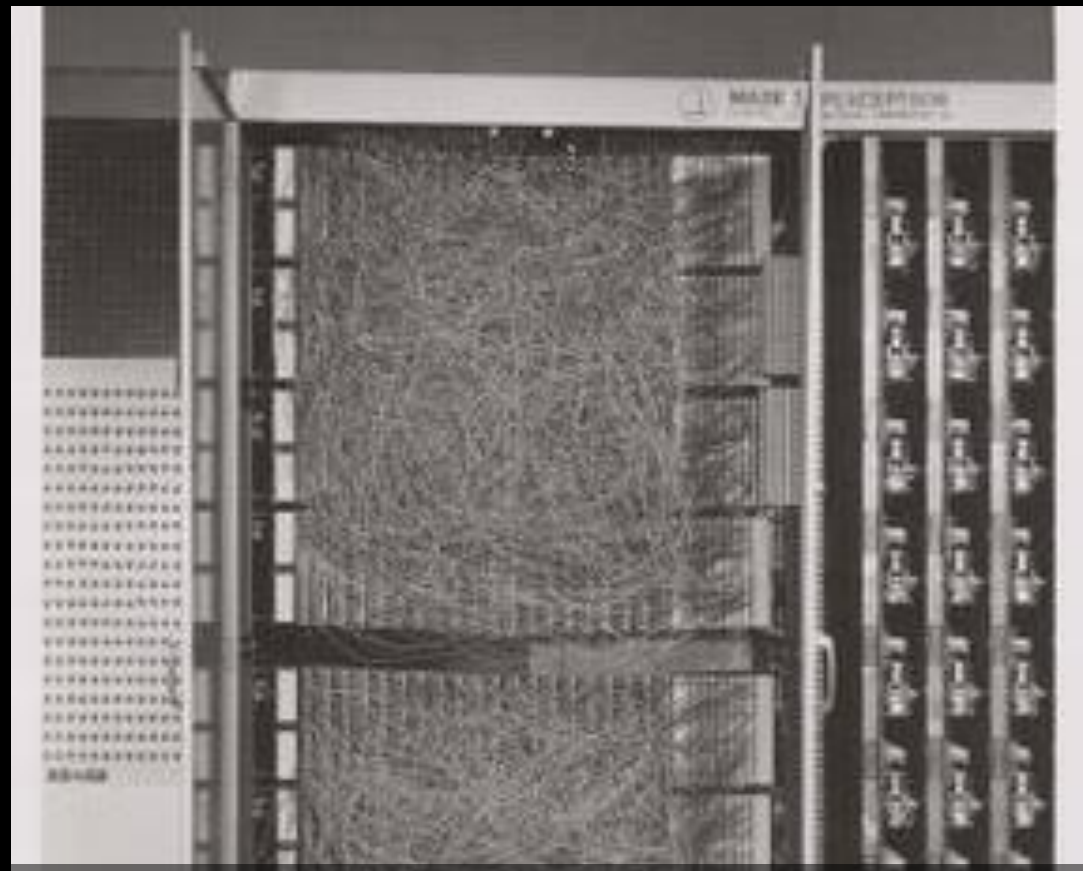
Deep Learning in Computer Vision Conferences



http://jponttuset.cat/are-gans-the-new-deep/

History of Neural Networks

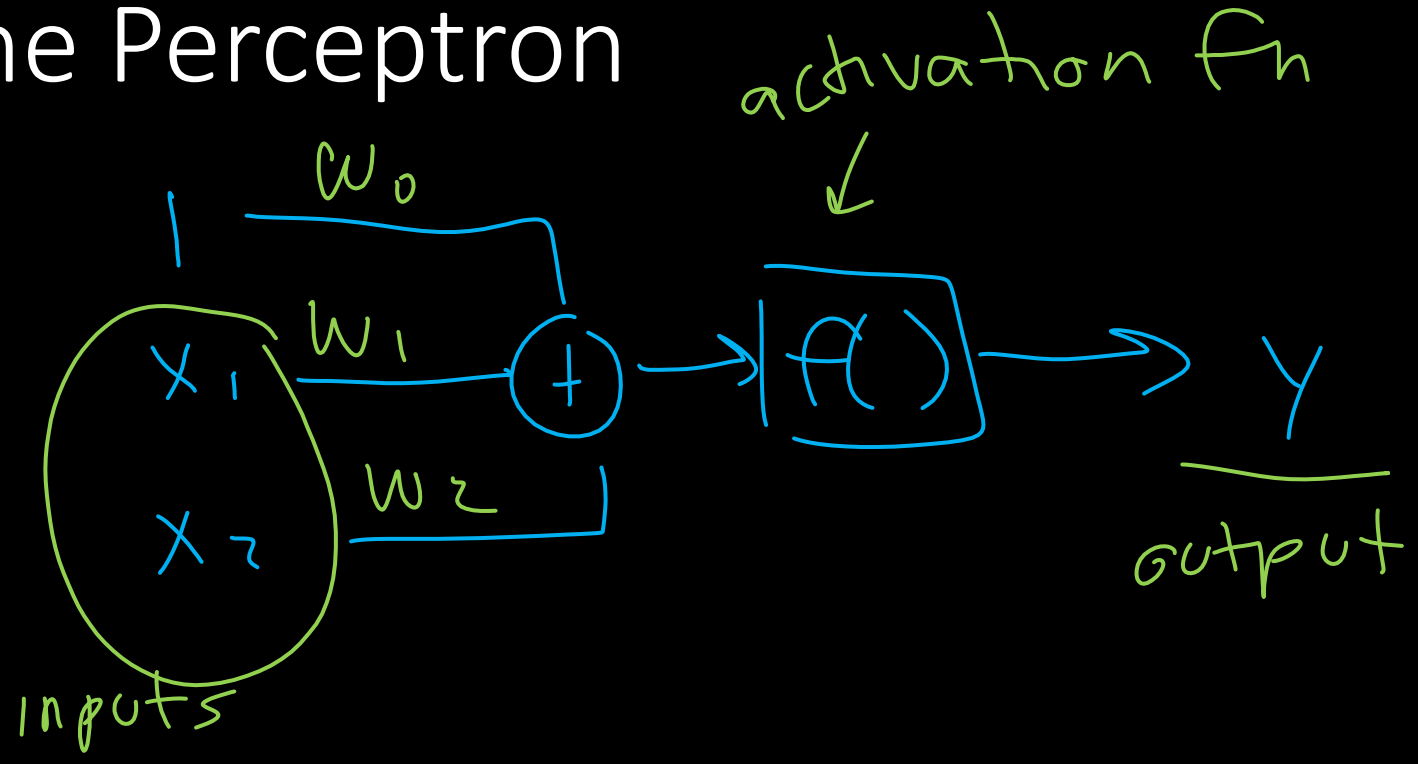
- Created in 1957 and presented as the first learning machine
- In 1969, Marvin Minsky and Seymour Papert showed that a perceptron could not learn an XOR function
  - Three years later, a series of papers introduced networks capable of modeling nonlinear fns and XOR
- Original work is often-miscited, and caused a significant decline in interest and funding of neural network research
  - It took ten more years for research to recover – resurgence in the 1980s and again in 2010s
- **Key take away:** neural networks may be powerful, but so are words.

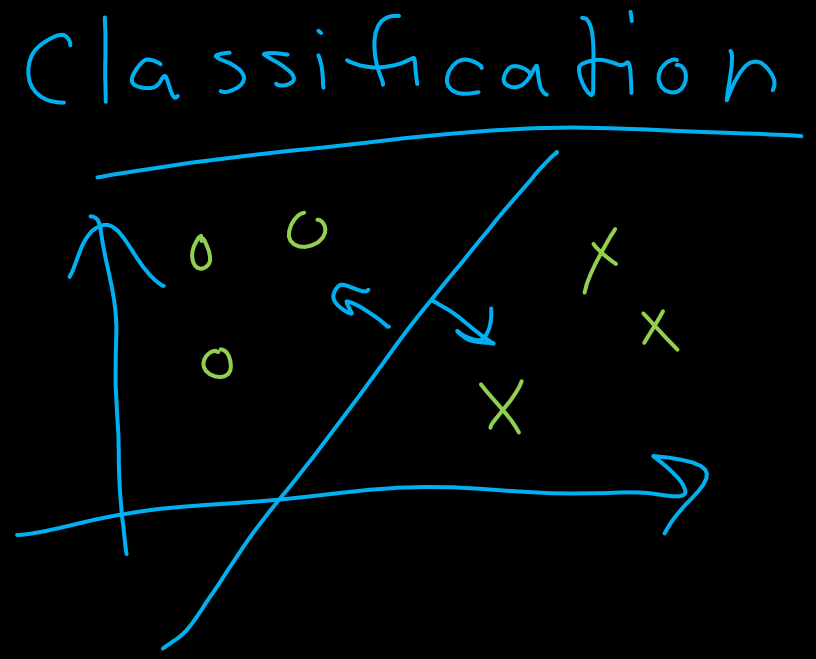


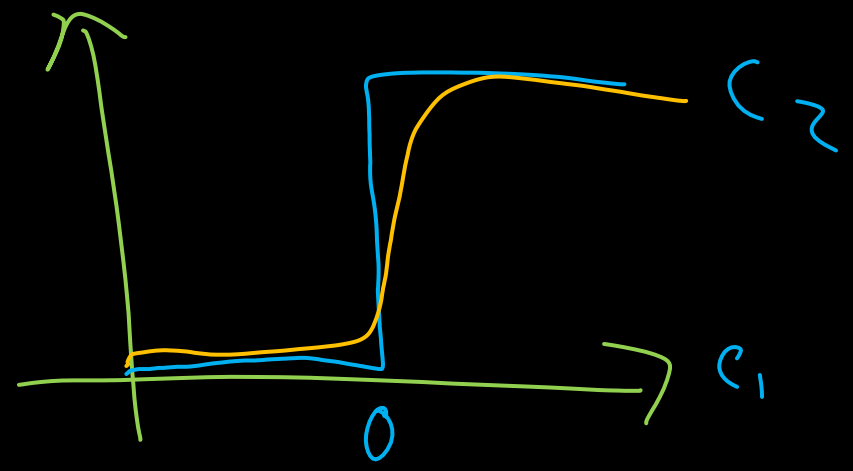
"…the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Mark I Perceptron by Frank Rosenblatt from Wikipedia

The Perceptron







Perceptron Learning Algorithm (1)

If the classes are linearly separable, then $\exists$ a weight vector that

$$w^T x \geq 0 \quad \forall\, x \text{ assoc w/ } c_1$$

$$w^T x < 0 \quad \forall\, x \text{ assoc w/ } c_2$$

Dataset w/ K examples

Perceptron Learning Algorithm (2)

while epoch < max or converged

for $k = 1 \ldots K$

$\hat{y}_k = f(w_k^T x_k)$

check if $\hat{y}_k$ is correct

if yes

$w_{k+1} = w_k$

if not

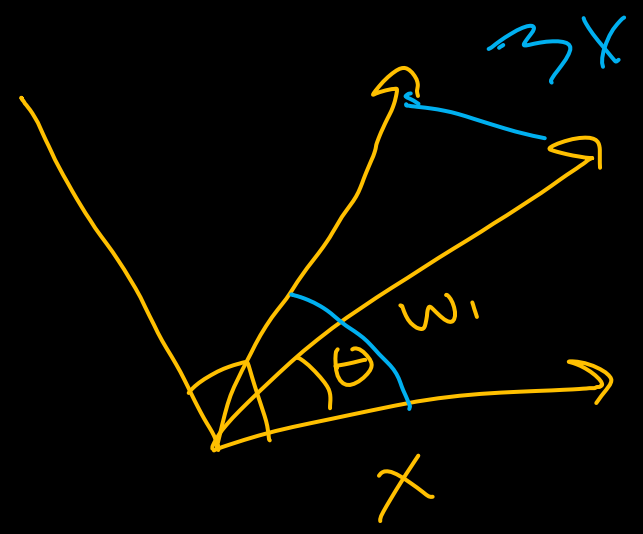
$w_{k+1} = w_k + \eta x_k$, if $\hat{y}_k \geq 0$, y_k is C_2

learning rate

Perceptron Learning Algorithm (3)

Recall the dot product

$$w^T x = |w||x| \cos\theta$$



Training via Gradient Descent

$$w_{k+1} = w_k + \Delta w_k$$

$$= w_k + \underbrace{\eta_k}_{\text{learning rate}} \overbrace{(y_k - \hat{y}_k)}^{\text{error}} x_k$$

General Optimization Problems

One a cost function is decided, an optimization problem is formulated

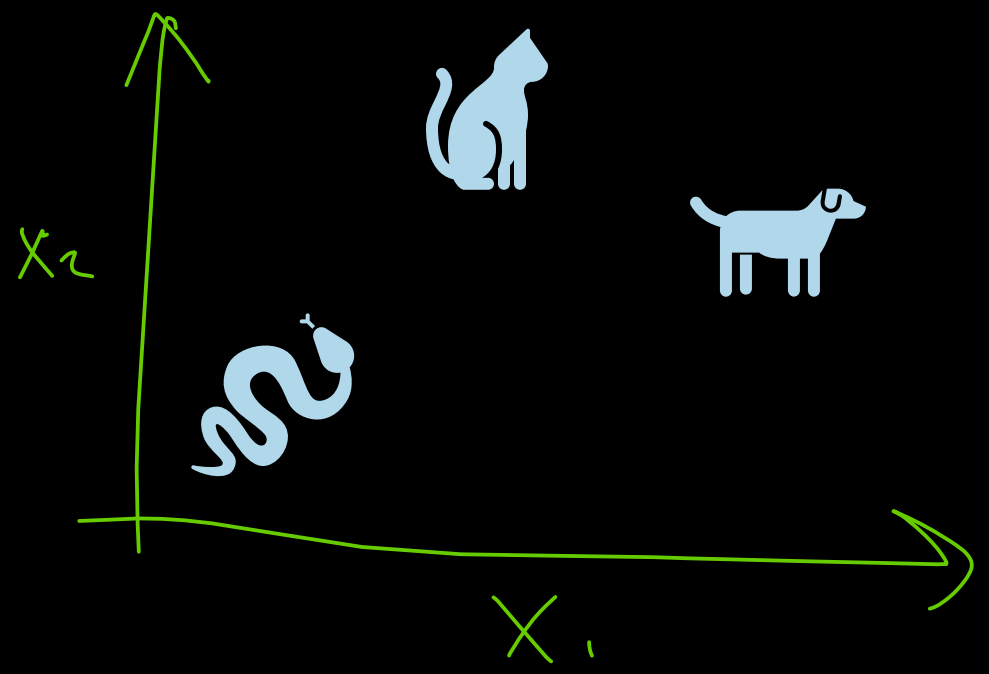
$$\underset{x,u}{\text{minimize}} \quad J(x,u)$$

$$\cancel{\text{subject to} \quad g(x,u) \geq 0}$$

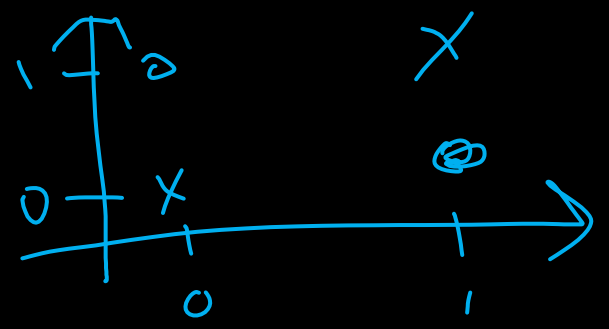
- Depending on the cost function, this problem can be nice and easy to solve or quite difficult
- Convex optimization problems have a global minimum, and are readily solved
- What if we have a nonlinear or nonconvex problem?

Extensions to Multi-Class Classification

- one vs one
  - Train $\frac{K(K-1)}{2}$ binary classifiers for a K-way multiclass problem
  - Voting scheme is then applied
- one vs all
  - Train K classifiers where each class is the positive and all others are negatives
- Neural networks with multiple outputs



The Famous XOR Counterexample



Suppose we have a perceptron with a linear activation:

$$y = f(w^\top x) = w^\top x$$

$$y = [w_0\ w_1\ w_2][1\ \ x_1\ x_2]^\top = w_0 + w_1 x_1 + w_2 x_2$$

XOR gives the following inequalities:

$$w_0 + w_1 0 + w_2 0 \leq 0 \ \rightarrow w_0 \leq 0$$

$$w_0 + w_1 0 + w_2 1 > 0 \ \rightarrow w_0 > -w_2$$

$$w_0 + w_1 1 + w_2 0 > 0 \ \rightarrow w_0 > -w_1$$

$$w_0 + w_1 1 + w_2 1 \leq 0 \ \rightarrow w_0 \leq -w_1 - w_2$$

$$w_2 > 0$$

$$w_1 > 0$$

$$w_1 + w_2 \leq 0$$

Universal Function Approximators

Let $\varphi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the *m*-dimensional unit hypercube $[0,1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f; that is,

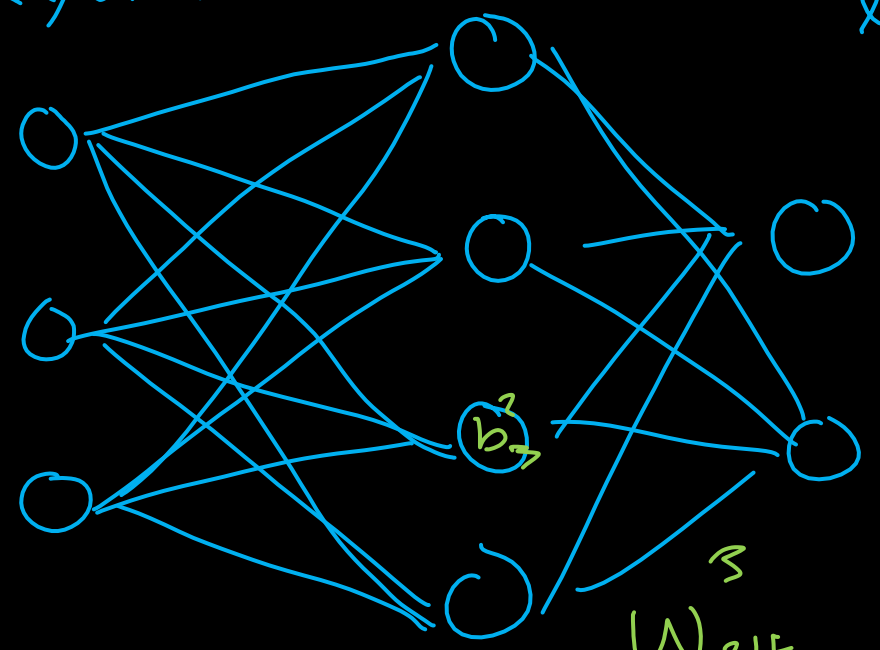
$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

A feedforward network with a single hidden layer containing a finite number of units can approximate continuous functions on compact subsets of R^n, under mild assumptions on the activation function.

Multi-Layer Perceptron (MLP)

layer 1 layer 2 layer 3 = L



a_1^3

W_{24}^3

$a_j^L \rightarrow y_j$

z_j^l weighted input

weights $w_{jk}^l \rightarrow$ assoc $l-1$

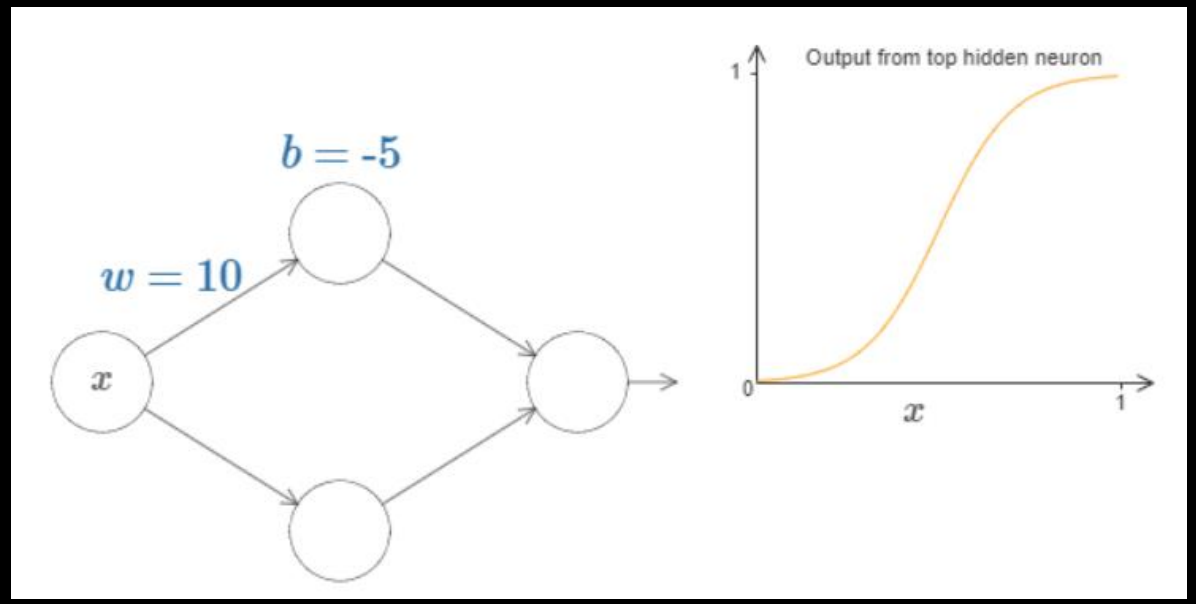
$\hookrightarrow$ assoc l

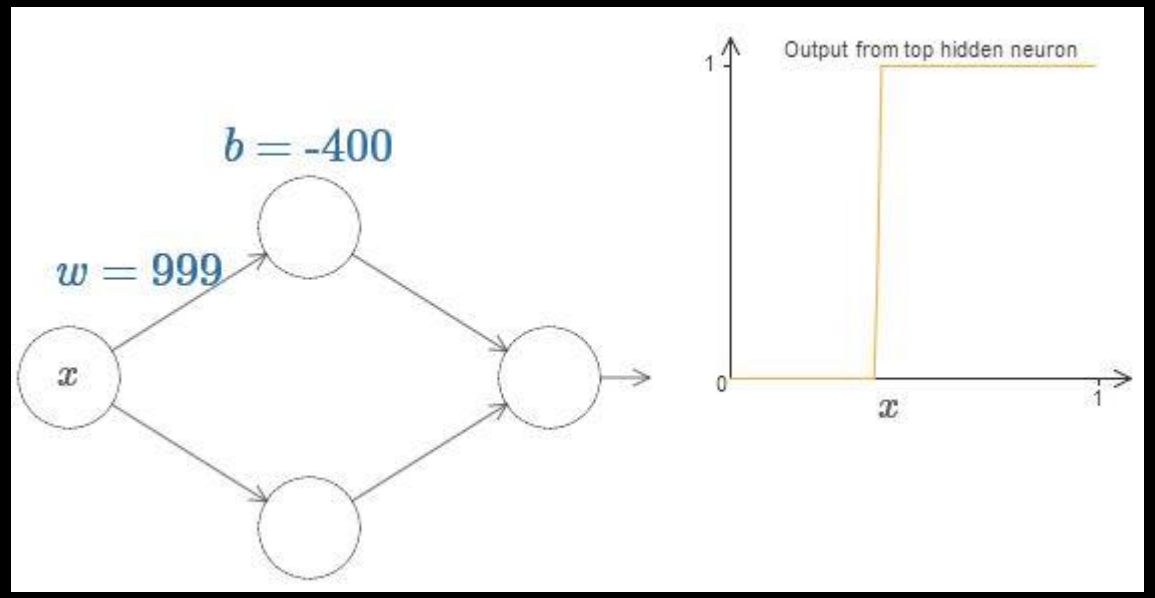
bias b_j^l

activation. $a_j^l = f\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$

$= f\left(w^l a^{l-1} + b^l\right)$

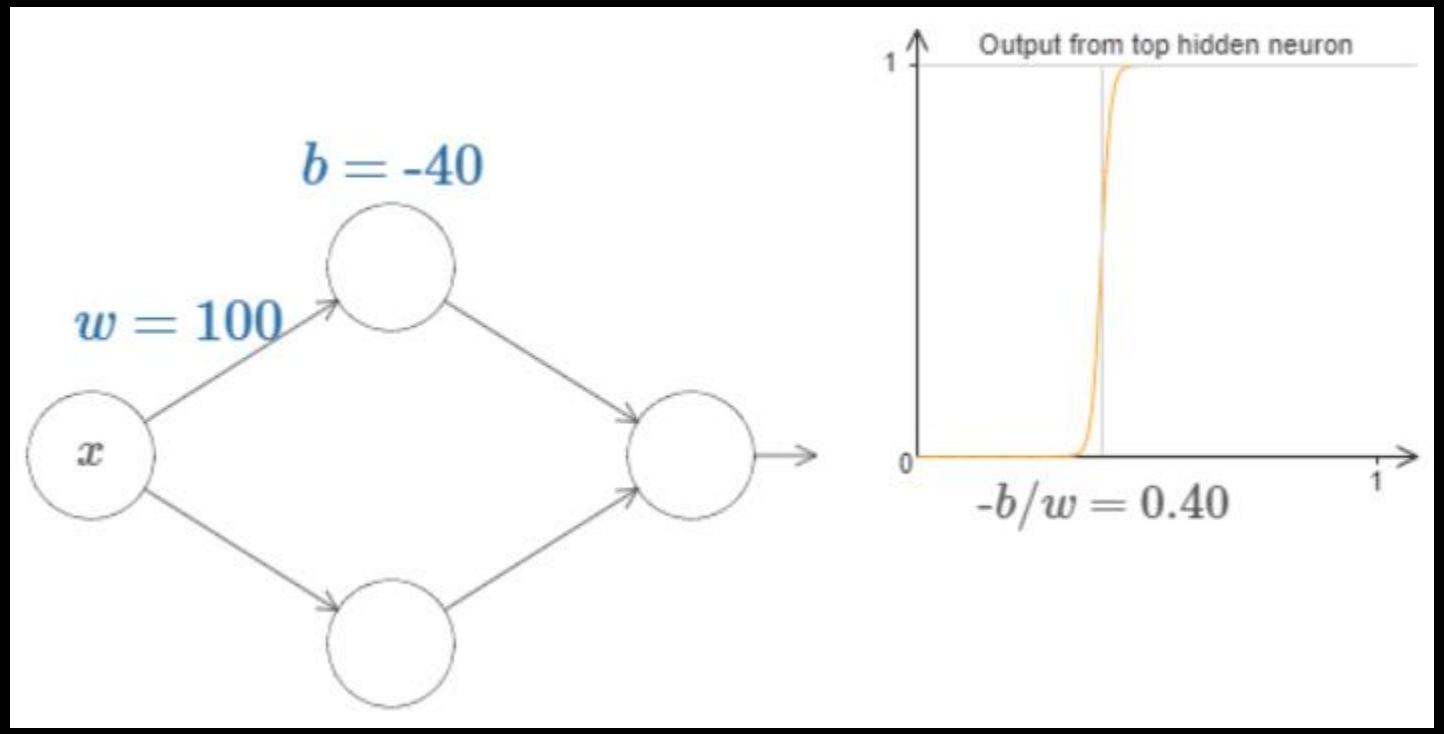
Universal Approximators





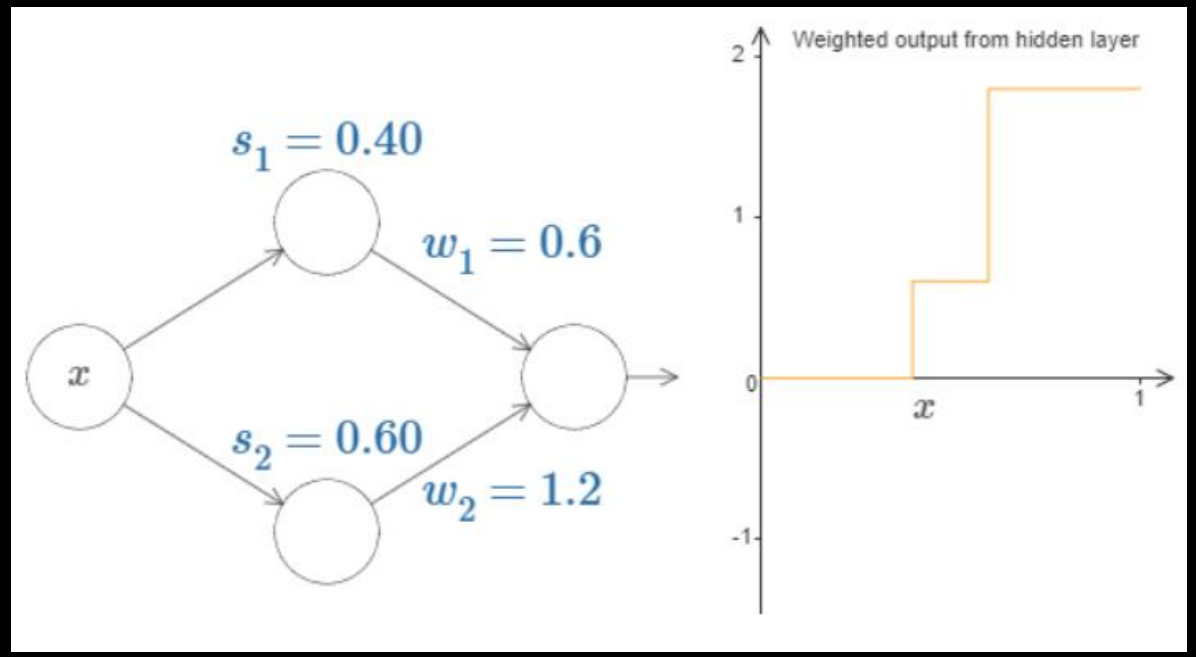
Visual proof by Michael Nielson.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

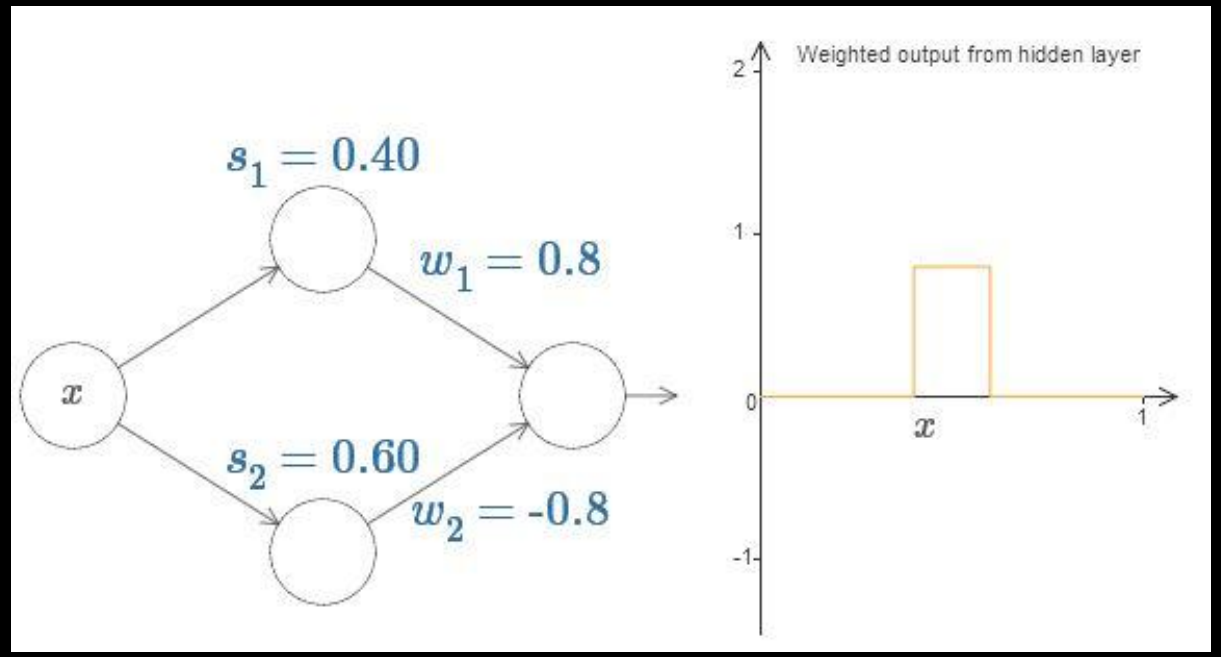
Universal Approximator



Visual proof by Michael Nielson.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

Universal Approximators

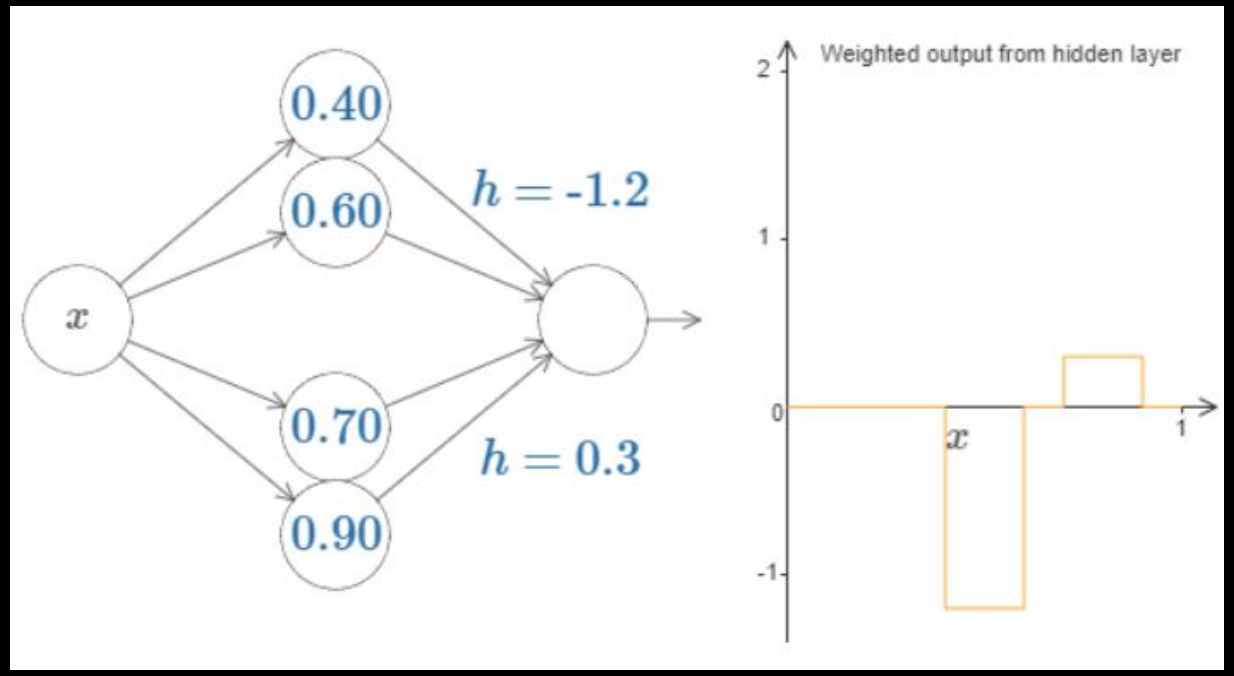


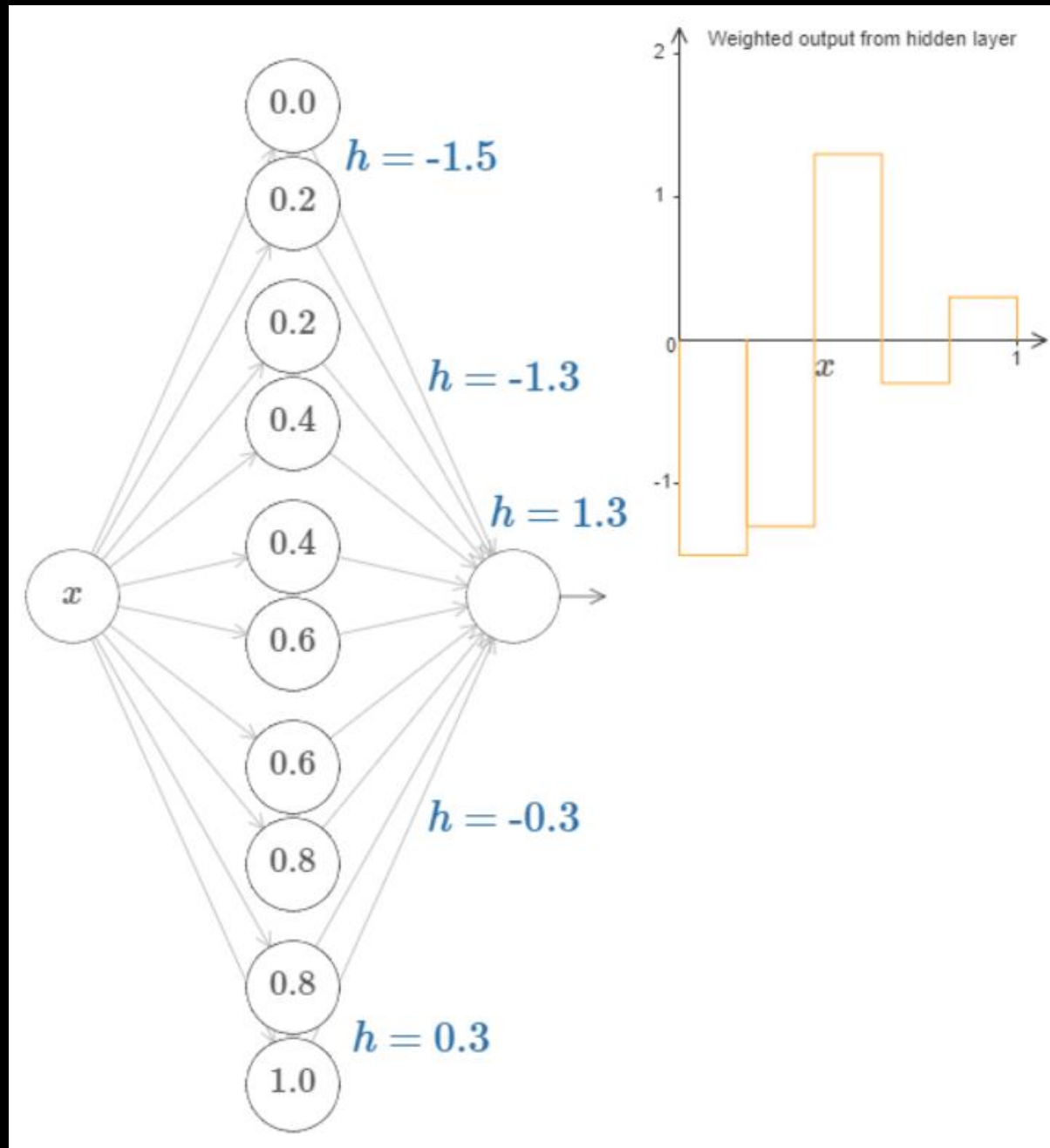


Final output is a weighted sum of hidden layer outputs.

Visual proof by Michael Nielson.
Neural Networks and Deep Learning / Chap. 4: A visual proof that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

Universal Approximators

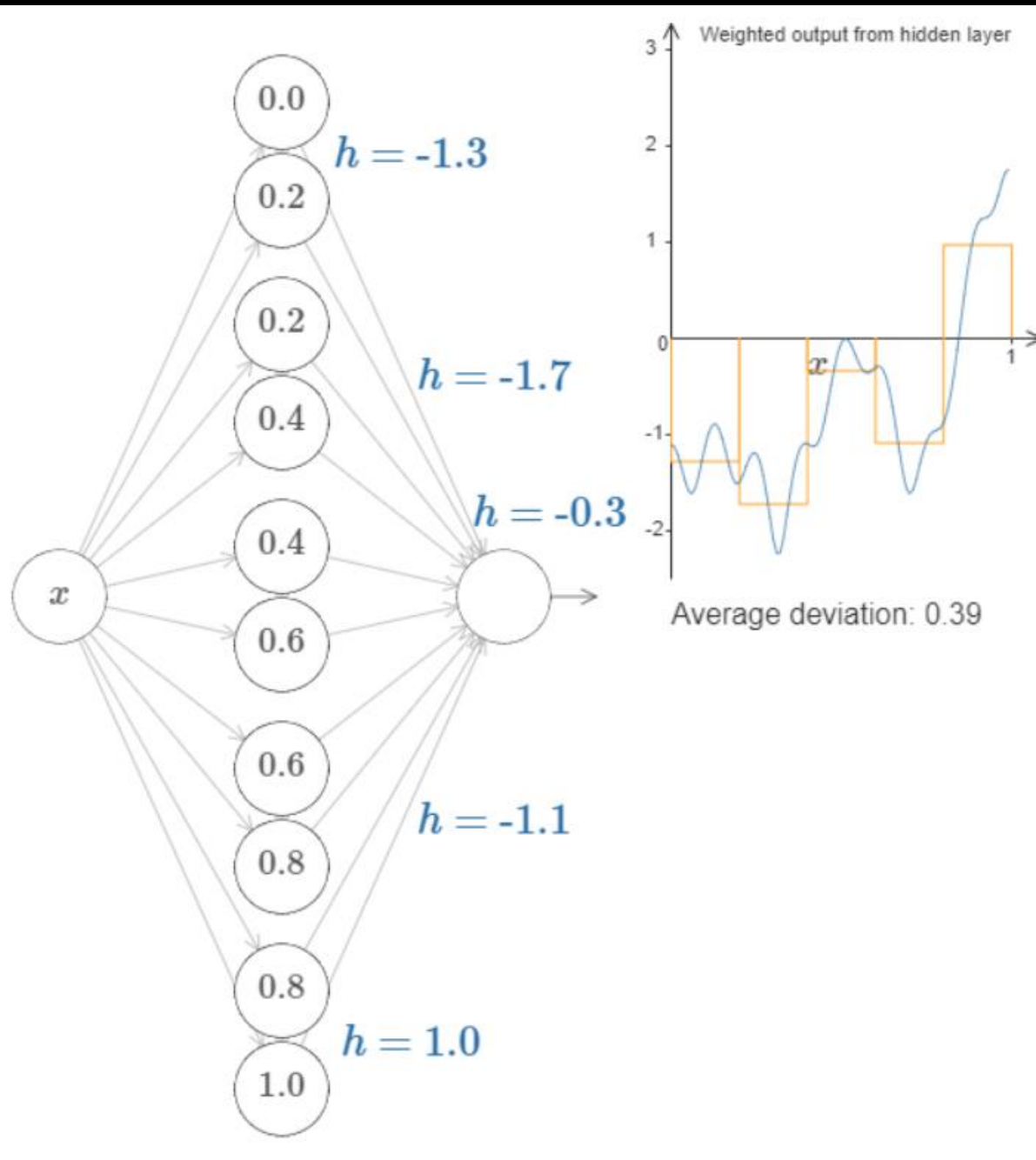




Universal Approximators

Given a smooth function, network parameters can be adapted to find a reasonable approximation

- single-layer neural networks essentially build a lookup table



Backpropagation (1)

Backpropagation (2)

optimize weights wrt errors

$$E(w) = \frac{1}{2} \sum_{r} e_r^2 \longrightarrow e = y_k - a_k^L$$

$r \longrightarrow$ output unit

we propose

$$w_{j'} \leftarrow w_{j'} + \Delta w_{j'}$$

$$\leftarrow w_{j'} + \eta \, \partial E / \partial w_{j'}$$

sensitivity of err wrt w

Backpropagation (3)

$$E = \frac{1}{2}\sum_r e_r^2 \,, \quad e = y - a^L$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial e_j} \cdot \frac{\partial e_j}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ji}}$$

e_j: error on unit j

a_j: output from unit j

z_j: weighted input $- \; a_j = f(z_j)$

$$\frac{\partial E}{\partial w_{ji}} = e_j \cdot (-1) \cdot f_j' \cdot a_i$$

output from unit i

Backpropagation (4)

$$w_{ji} \leftarrow w_{ji} - \eta \, \partial E / \partial w_{ji}$$

$$\leftarrow w_{ji} + \eta \, \ell_j f_j' \, a_i$$

↳ for layer L, we know ℓ_j

→ what about hidden layers?

Backpropagation (5)

for any unit p connected to q

$$\Delta w_{qp} = \eta \delta_q \alpha_p$$ local gradient

if q is output

$$\delta_q = e_q f_q'(\ )$$

if q is hidden

$$\delta_q = \sum_r \delta_r w_{rq} f_q'$$

Backpropagation Review

1. **Input:** x
2. **Feedforward:** For each layer, compute z^l and a^l
3. **Compute error:** E or δ^l
4. **Backpropagate** the error: For each inner layer, compute the delta rule
5. **Output:** The gradient of the cost function

Next time…

- Deep architectures
- Tips and tricks
- Common applications and uses
- Risks for autonomy