CS 540 Introduction to Artificial Intelligence
Neural Networks I: Perceptron

University of Wisconsin–Madison
Fall 2025, Section 3
October 6, 2025

Announcements

- HW4 due Friday 10/6 at 11:59 pm
- HW5 out on Friday (linear regression & gradient descent)
- Midterm exam: Thursday 10/23 from 7:30 pm to 9:00 pm
- Professor Brown's Wednesday office hours start at 11 am

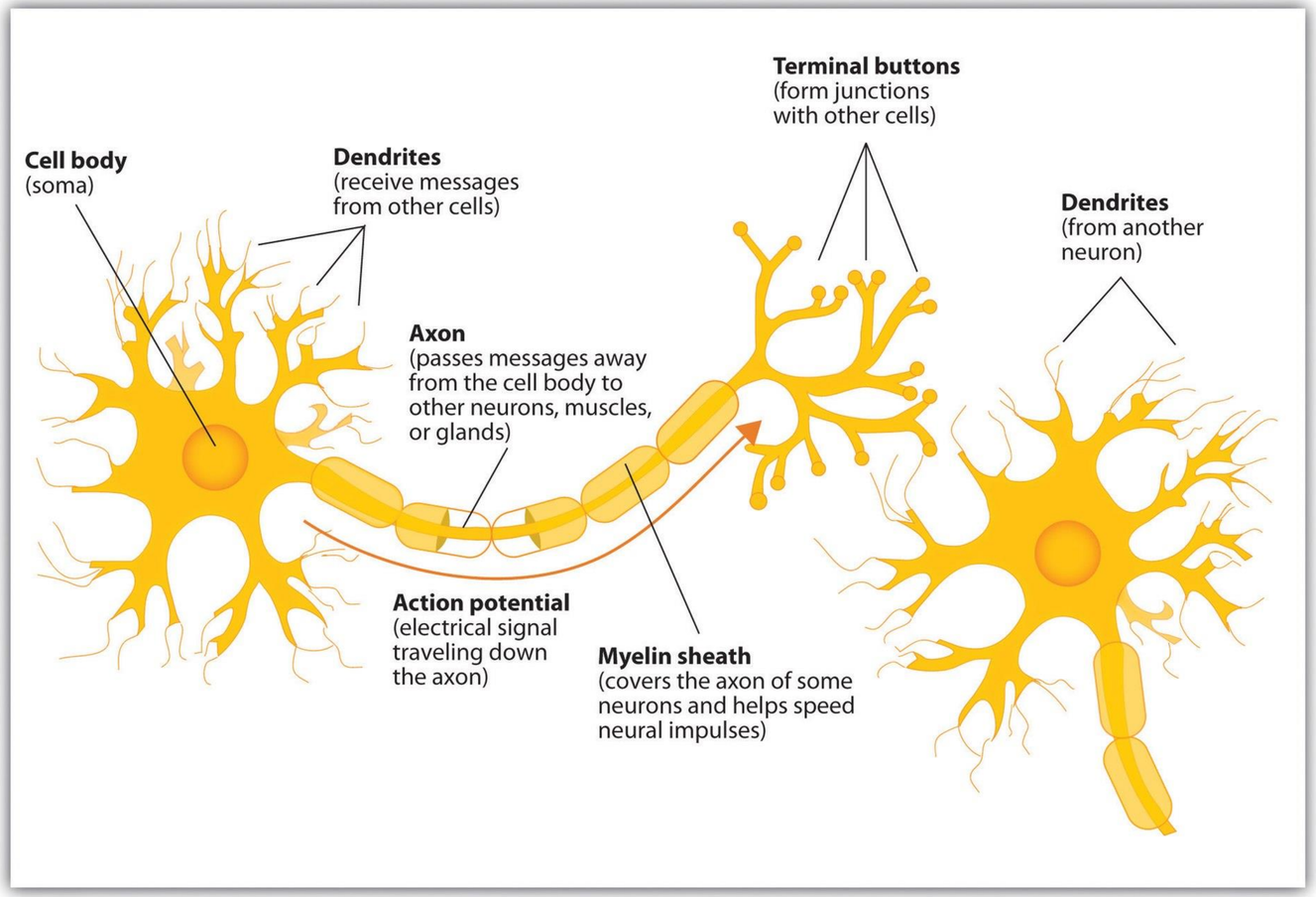
ML: Unsupervised Learning
ML Linear Regression
Machine Learning: K - Nearest Neighbors & Naive Bayes
Machine Learning: Neural Networks I (Perceptron)
Machine Learning: Neural Networks II

Part I: Single-layer Neural Network

Inspiration from neuroscience

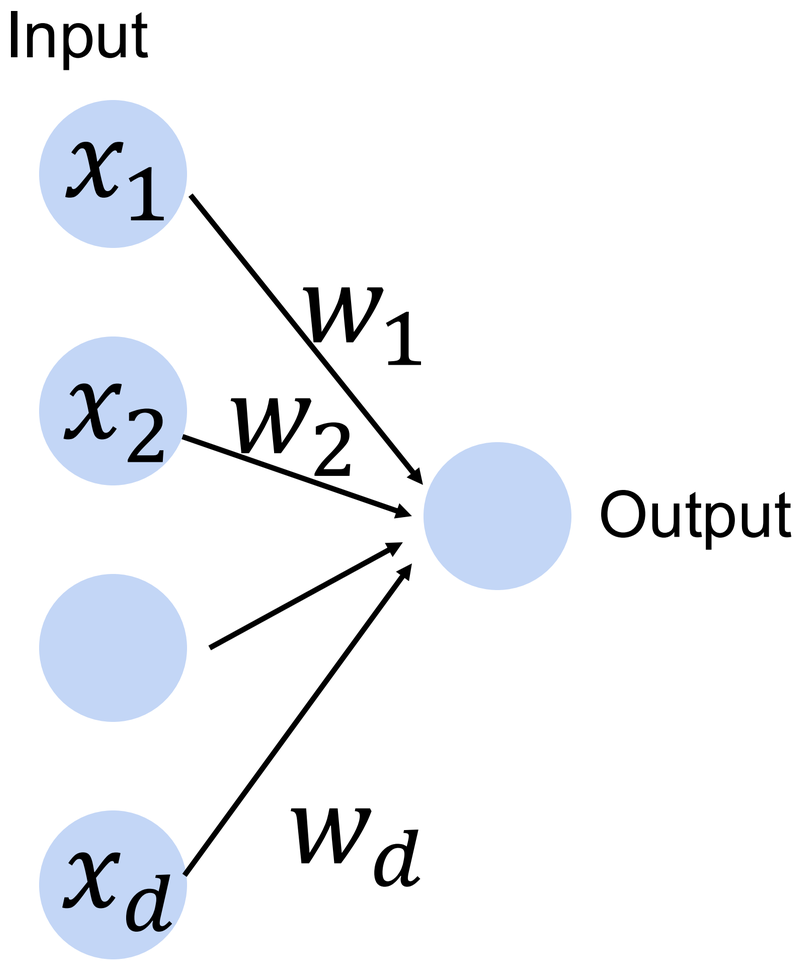
- Inspirations from human brains
- Networks of simple and homogenous units

(wikipedia)



Perceptron

Cats vs. dogs?

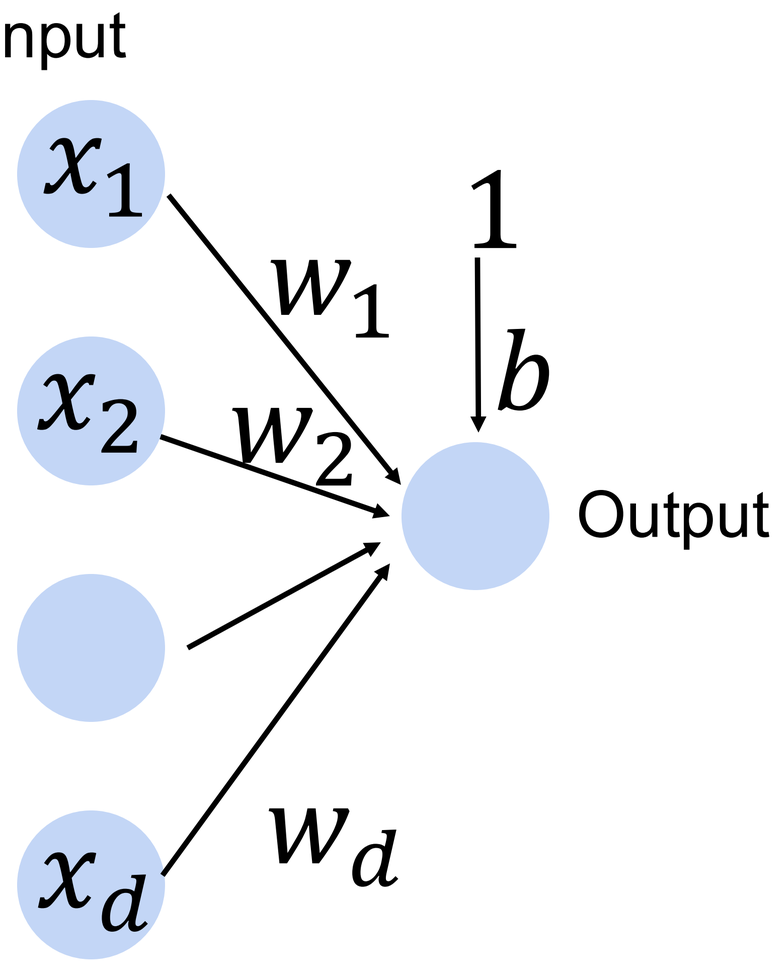


Linear Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$f = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Cats vs. dogs?



Perceptron

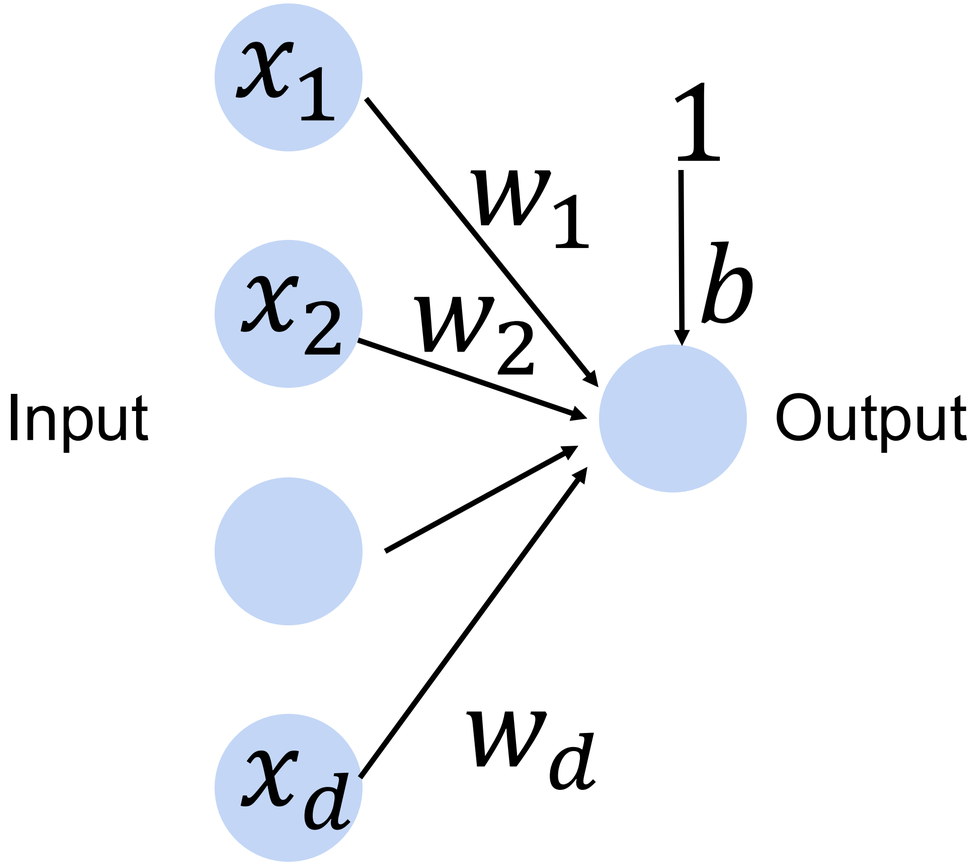
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

Hard threshold activation function

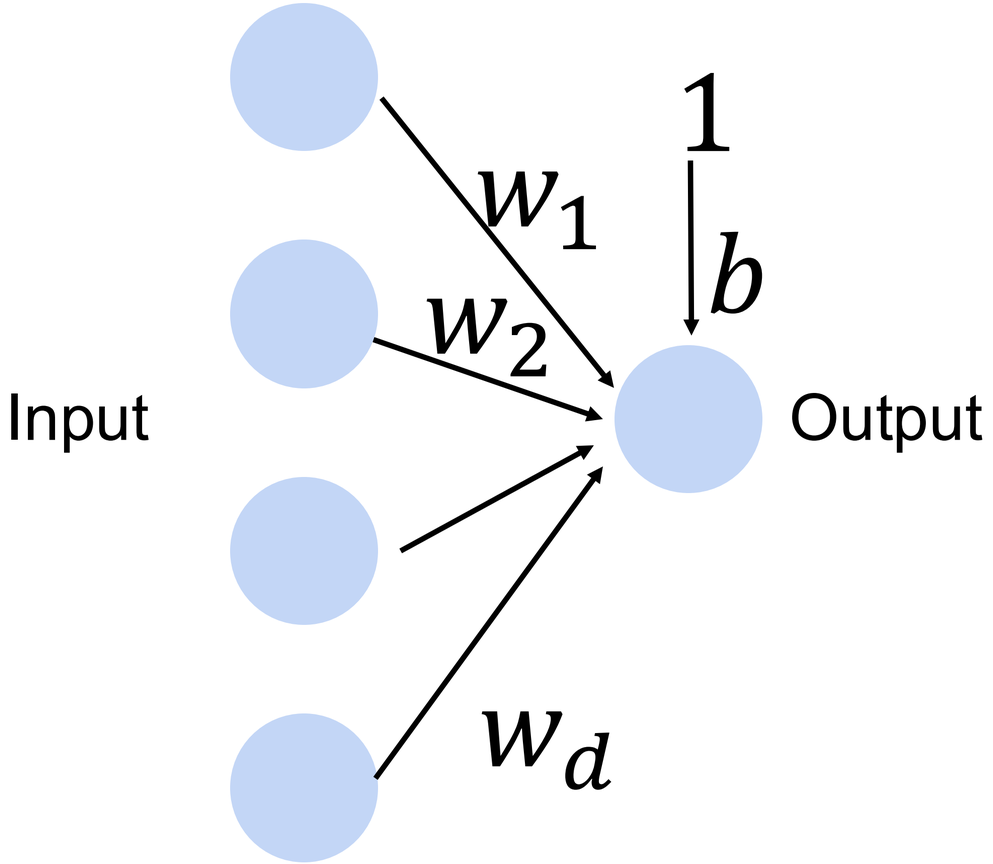
Cats vs. dogs?



Perceptron

- Goal: learn parameters $\mathbf{w} = \{w_1, w_2, \ldots, w_d\}$ and b to minimize the classification error

Cats vs. dogs?



Basic Recipe for Supervised Learning

1. Select model class
2. Select loss
3. Optimize parameters
4. Evaluate output

Usually: apply a variant of **gradient descent**

Usually: compute loss on **test set**

Basic Recipe: Perceptron

1. Select model class
2. Select loss
3. Optimize parameters
4. Evaluate output

1. Linear decision rules

   All functions like: $f_{w,b}(x) = \sigma(\langle w, x \rangle + b)$
2. Misclassification error
3. Perceptron Learning Rule

   Similar to gradient descent
4. Test/train split

Training the Perceptron

- Iterative algorithm to decrease loss on training data
  - Start with initial weights w_0, b_0
  - When perceptron misclassifies a point, update the weights

- Randomly pick next training example
  - It is a **stochastic** training algorithm
  - Similar to stochastic gradient descent (SGD)

The Perceptron Learning Rule

Perceptron Learning Algorithm

Input: dataset (X, y)
number of steps T, step size η

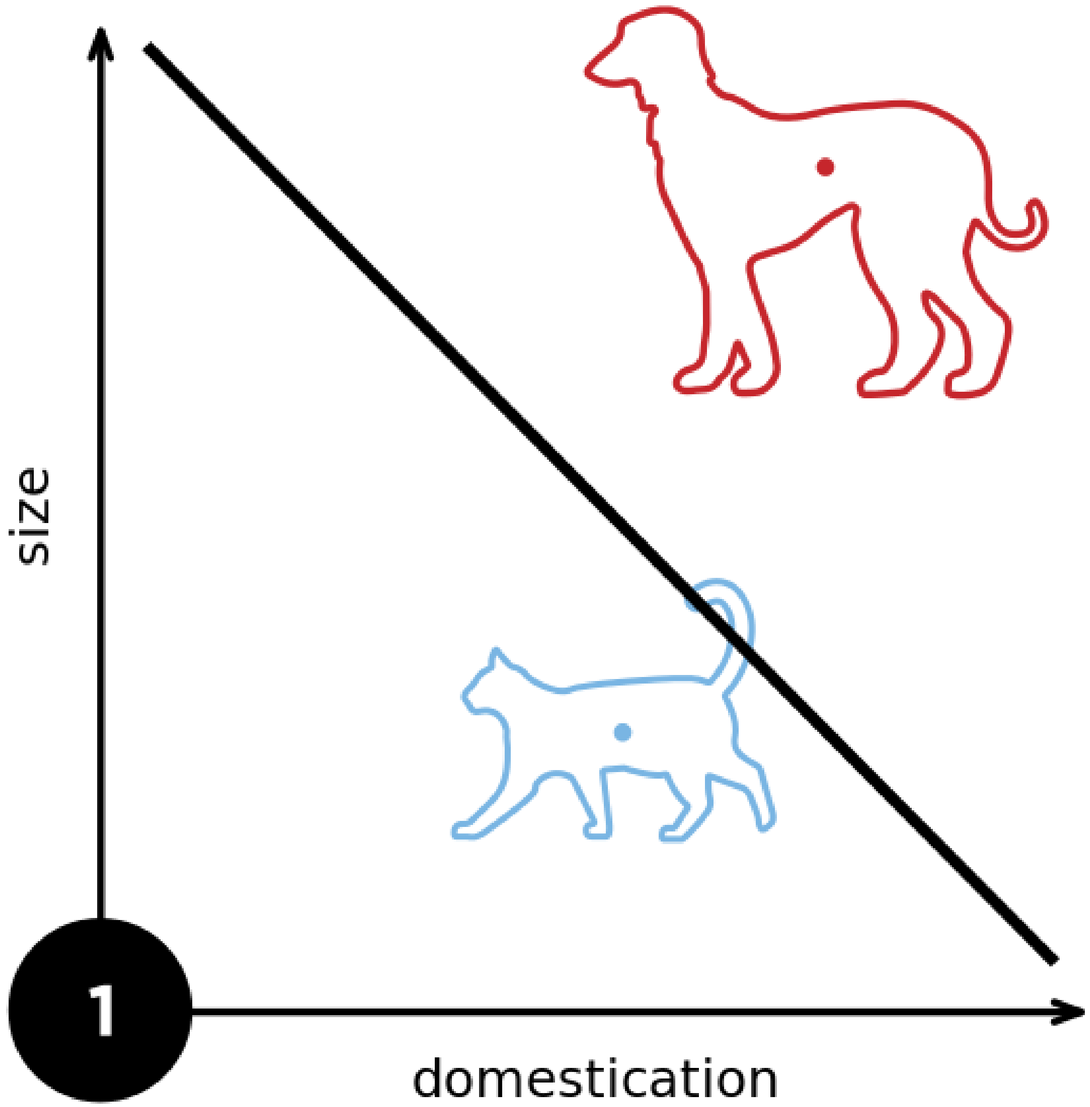
1. Initialize w_0, b_0
2. For $t = 1, 2, \ldots, T$
3. Pick random (x_i, y_i)
4. Predict $\hat{y}_i \leftarrow \sigma(\langle w_{t-1}, x_i \rangle + b_{t-1})$
5. If $\hat{y}_i \neq y_{\mathrm{i}}$:
6. $w_t \leftarrow w_{t-1} + \eta(y_i - \hat{y}_i)x_i$
7. $b_t \leftarrow b_{t-1} + \eta(y_i - \hat{y}_i)$
8. Return w_T

Gradient Descent

Input: dataset (X, y), loss function L,
number of steps T, step size η

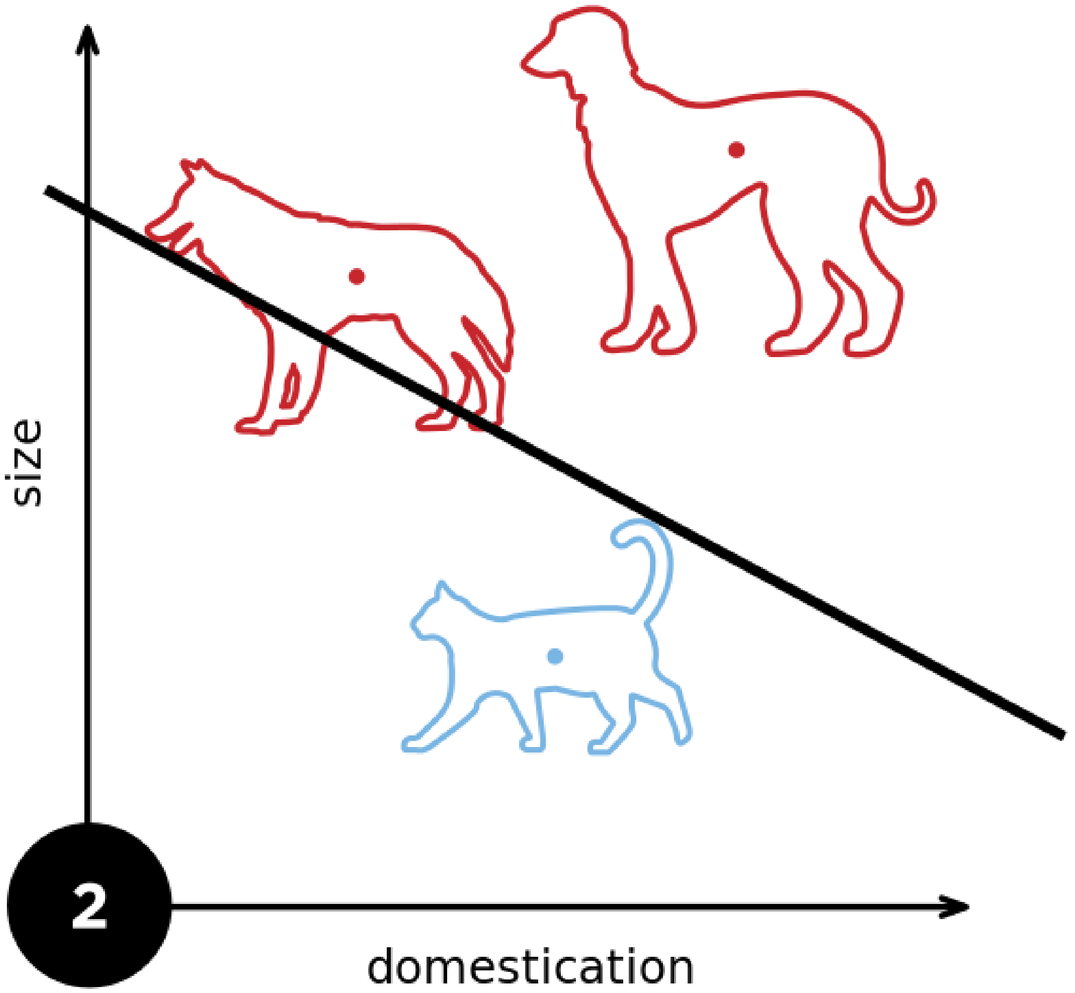
1. Initialize w_0
2. For $t = 1, 2, \ldots, T$
3. Calculate $g_t = \nabla L(w_{t-1}; X, y)$
4. Update $w_t \leftarrow w_{t-1} - \eta g_t$
5. Return w_T

Perceptron



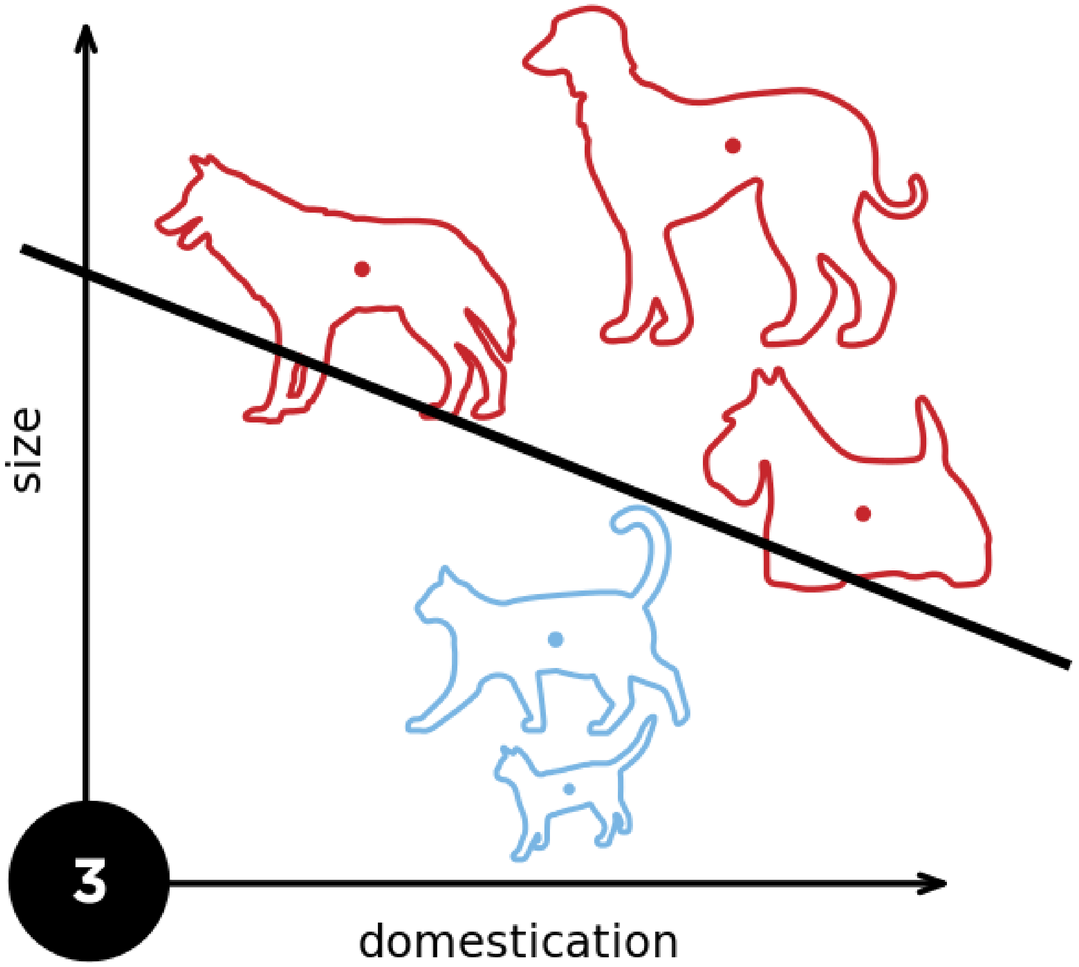
From wikipedia

Perceptron



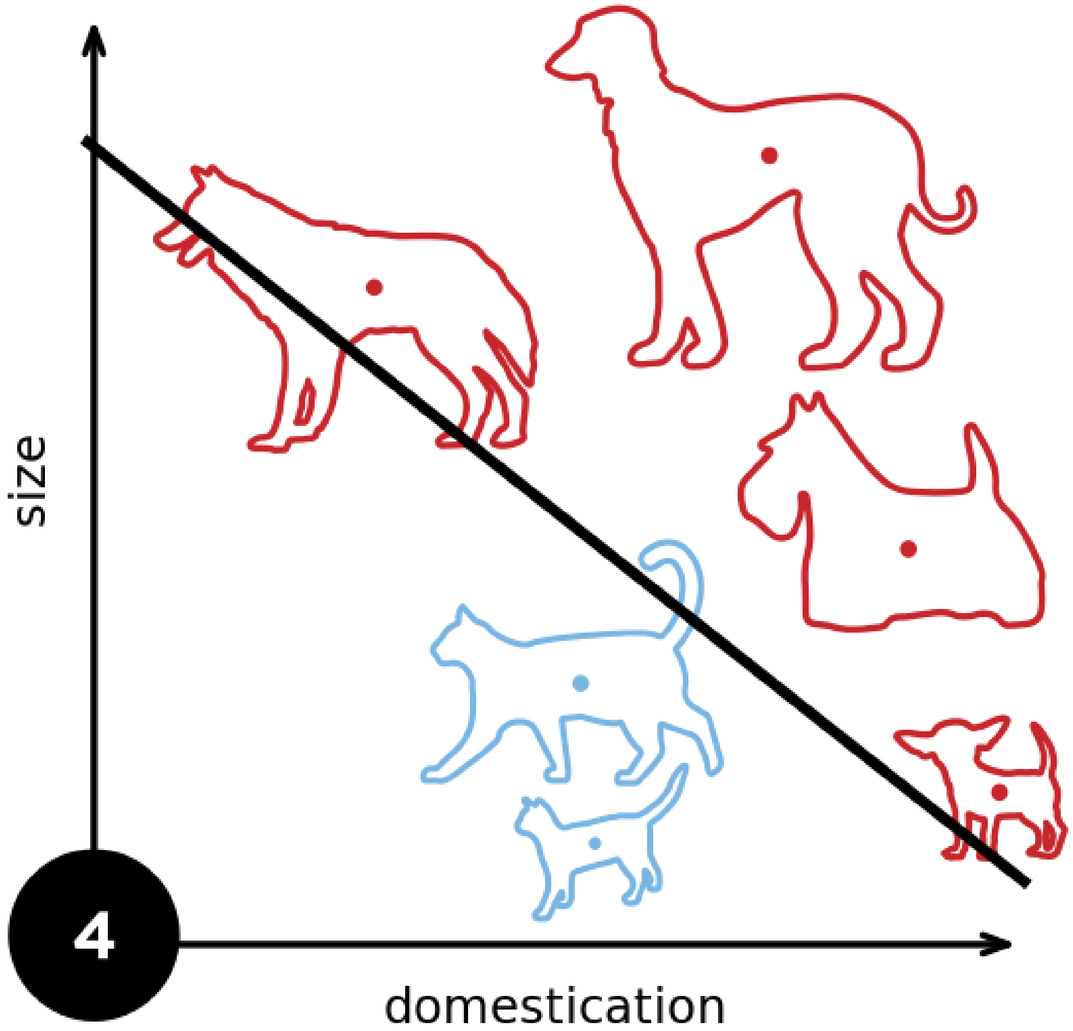
From wikipedia

Perceptron



From wikipedia

Perceptron



From wikipedia

Learning AND function using perceptron

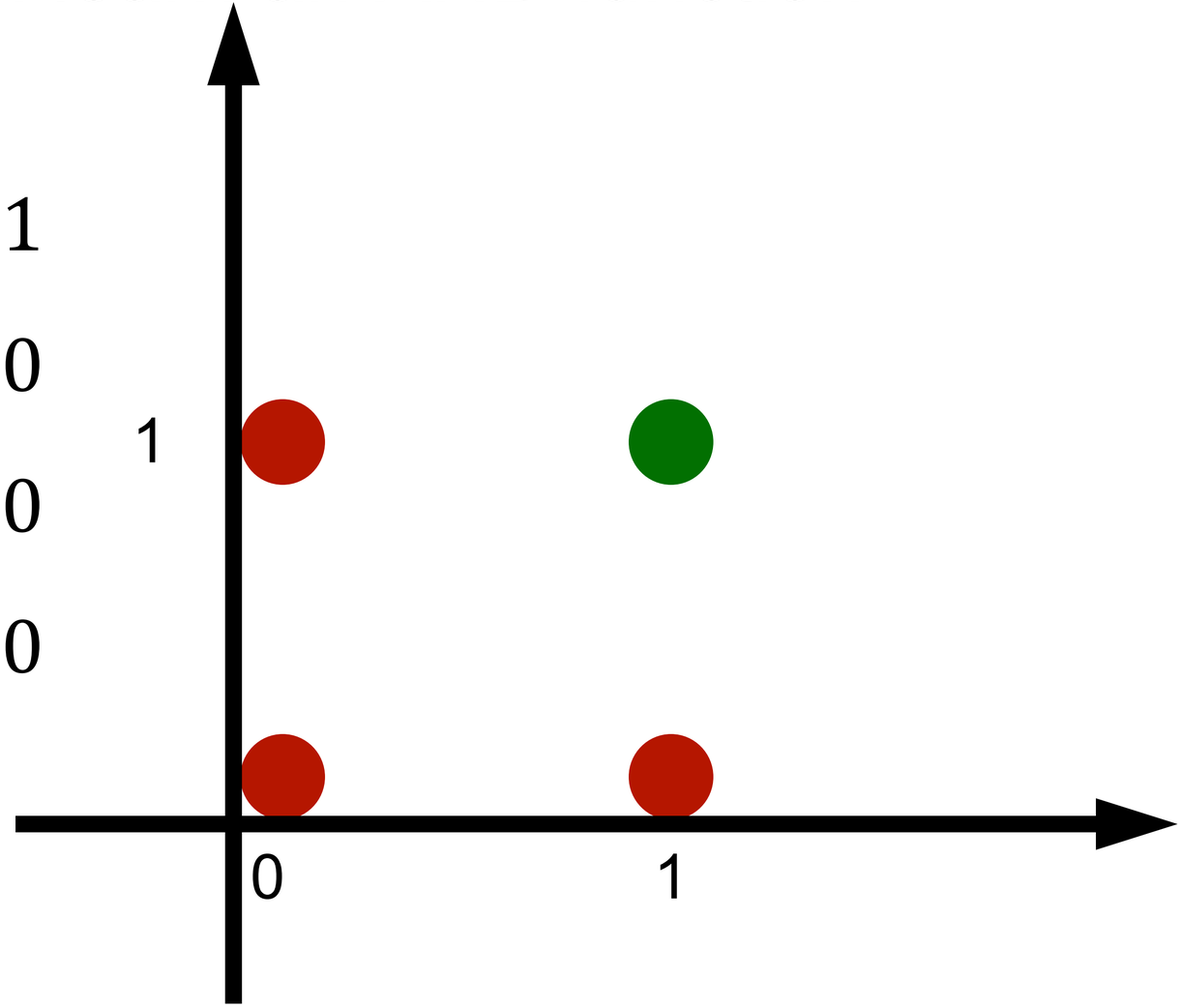
The perceptron can learn an AND function

$x_1 = 1, x_2 = 1, y = 1$

$x_1 = 1, x_2 = 0, y = 0$

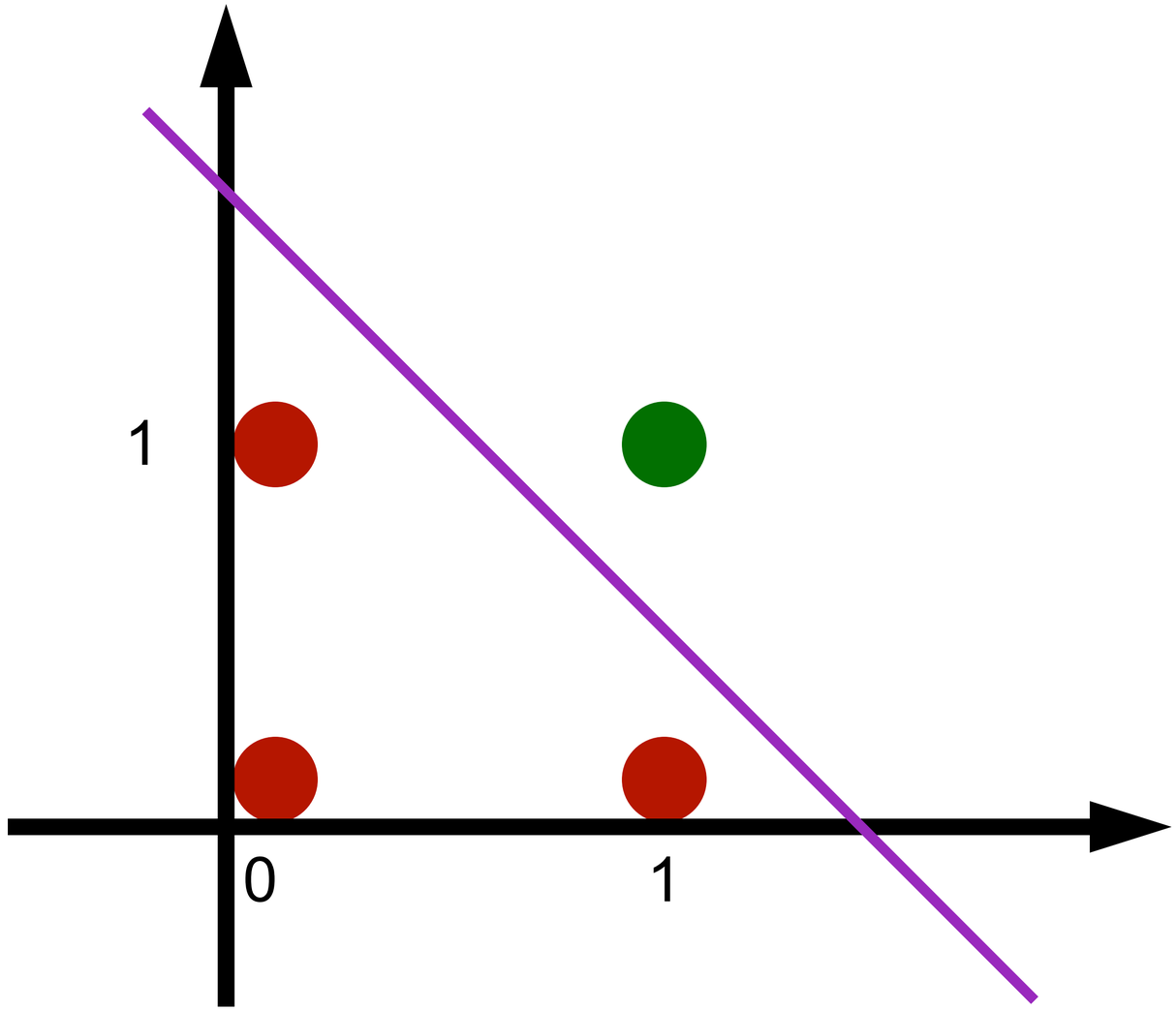
$x_1 = 0, x_2 = 1, y = 0$

$x_1 = 0, x_2 = 0, y = 0$



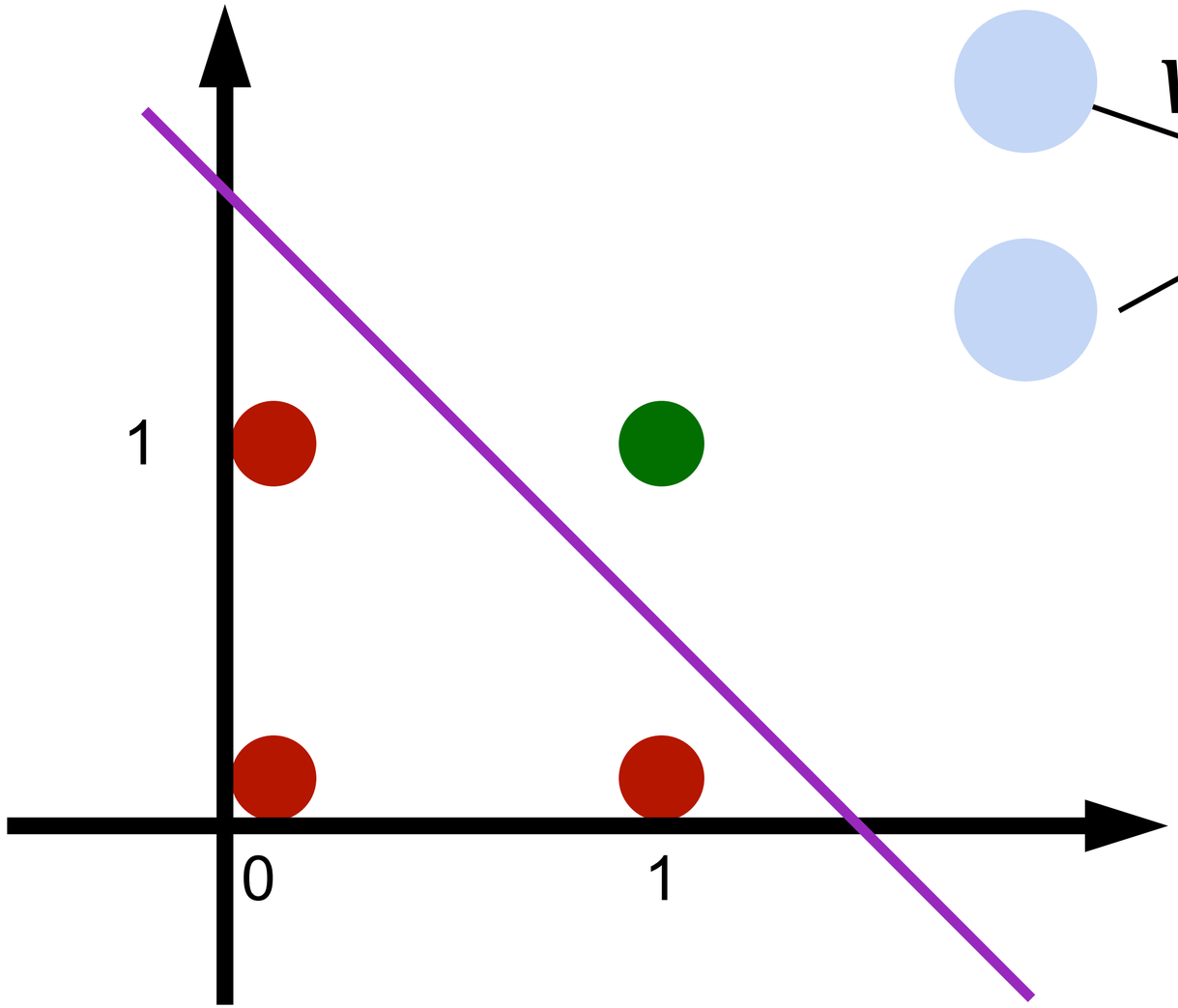
Learning AND function using perceptron

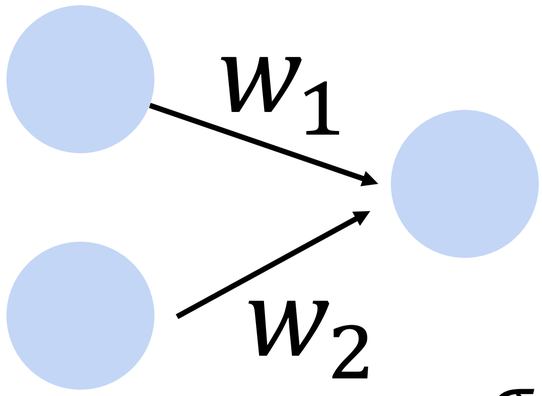
The perceptron can learn an AND function



Learning AND function using perceptron

The perceptron can learn an AND function





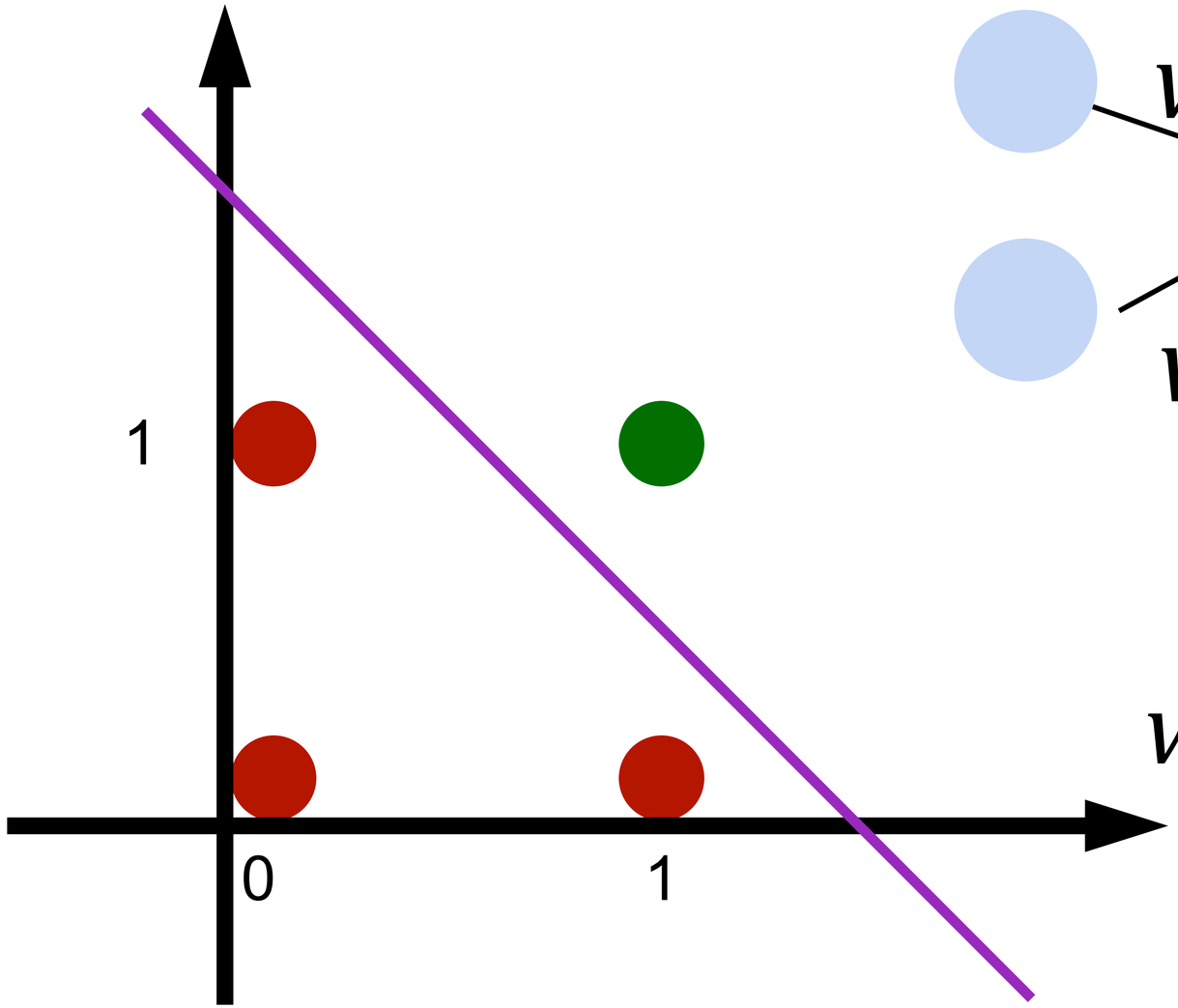
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

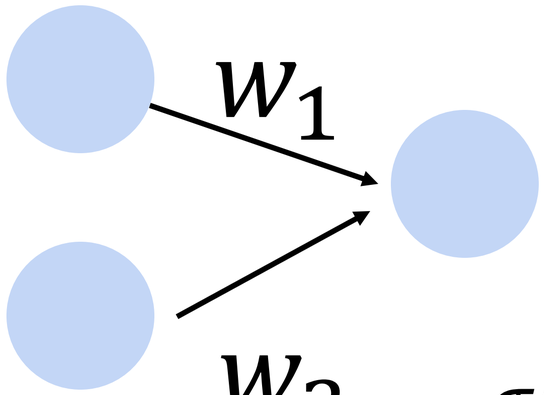
$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

What's w and b?

Learning AND function using perceptron

The perceptron can learn an AND function





$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -1.5$$

Learning OR function using perceptron

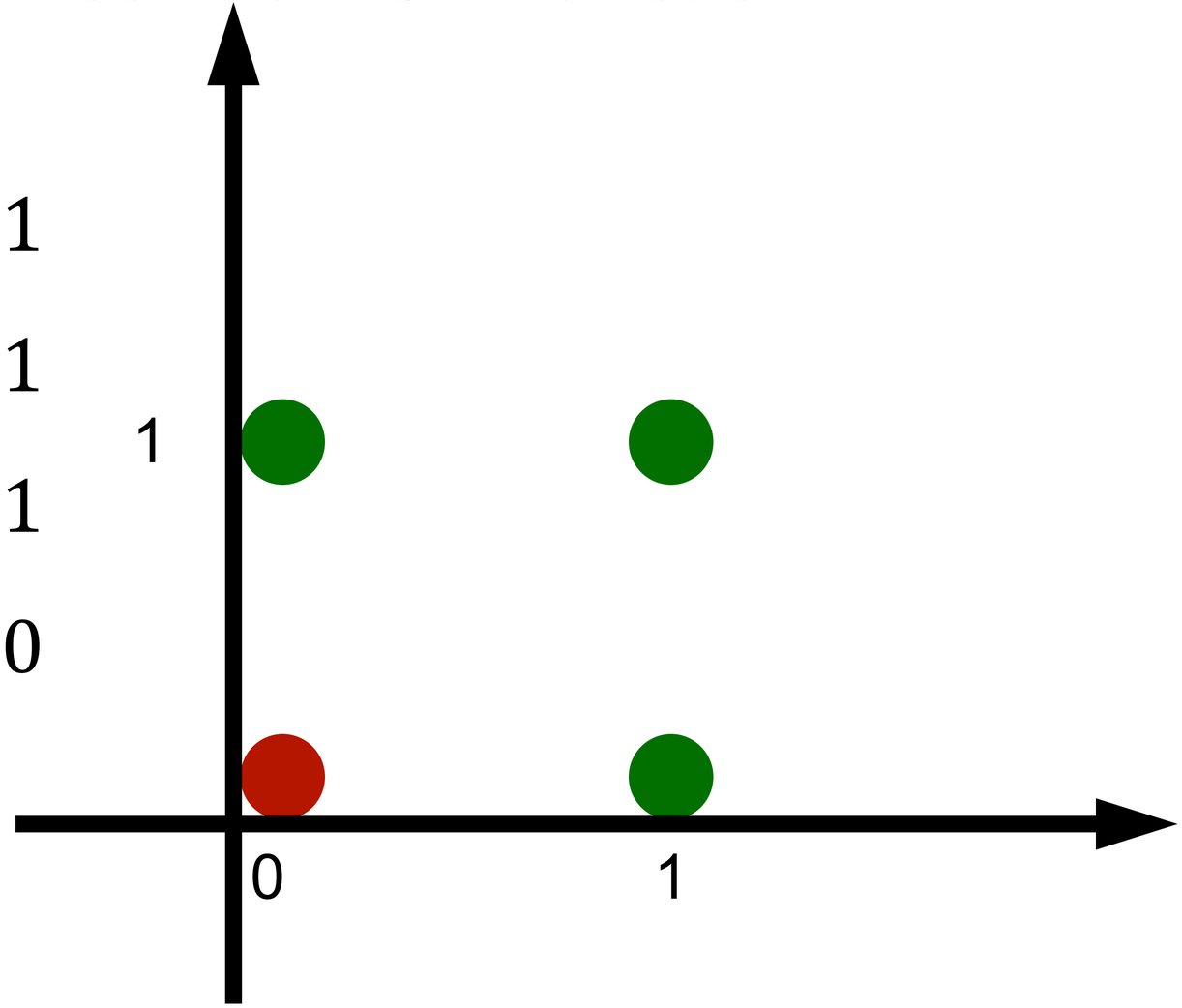
The perceptron can learn an OR function

$x_1 = 1, x_2 = 1, y = 1$

$x_1 = 1, x_2 = 0, y = 1$

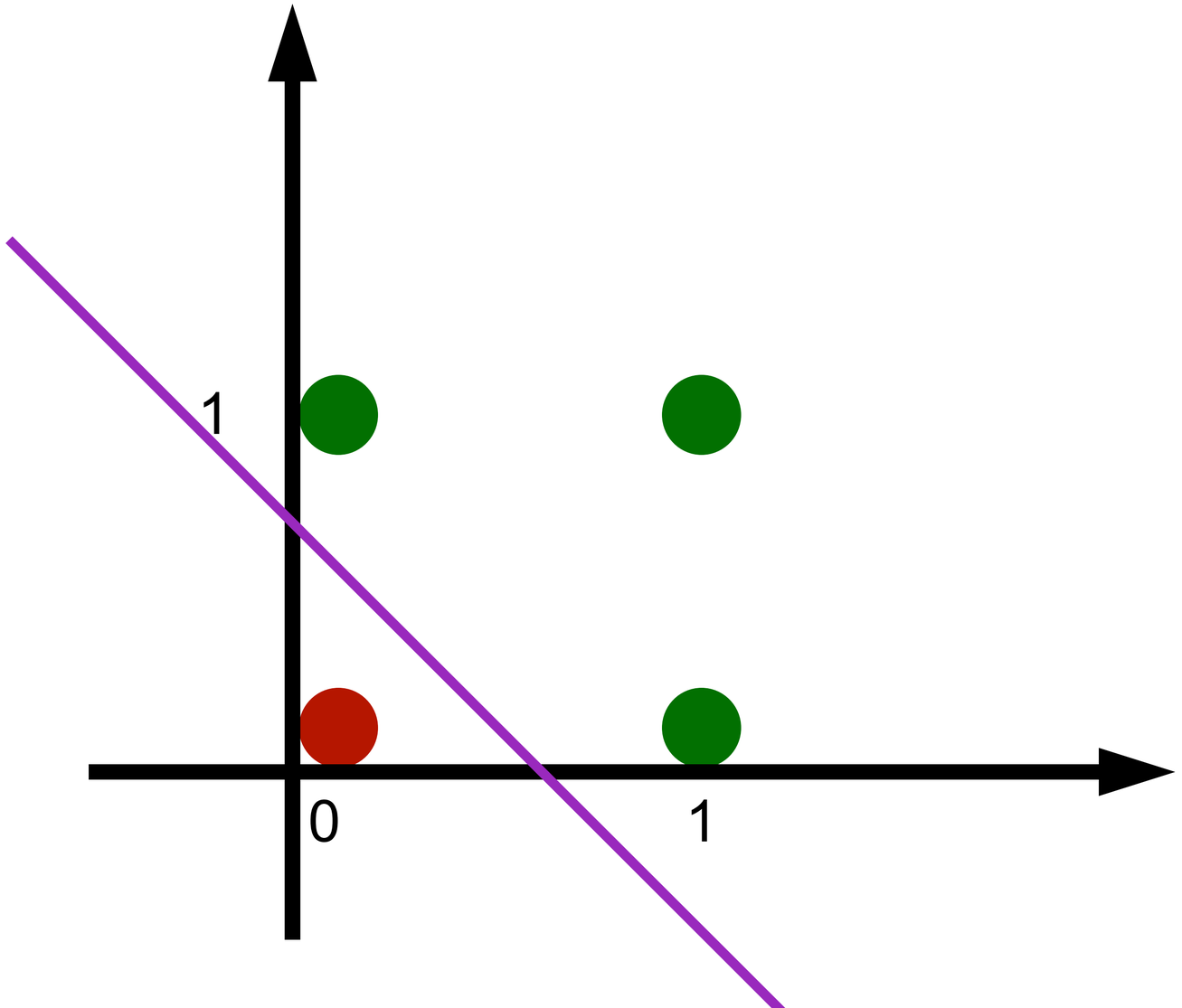
$x_1 = 0, x_2 = 1, y = 1$

$x_1 = 0, x_2 = 0, y = 0$



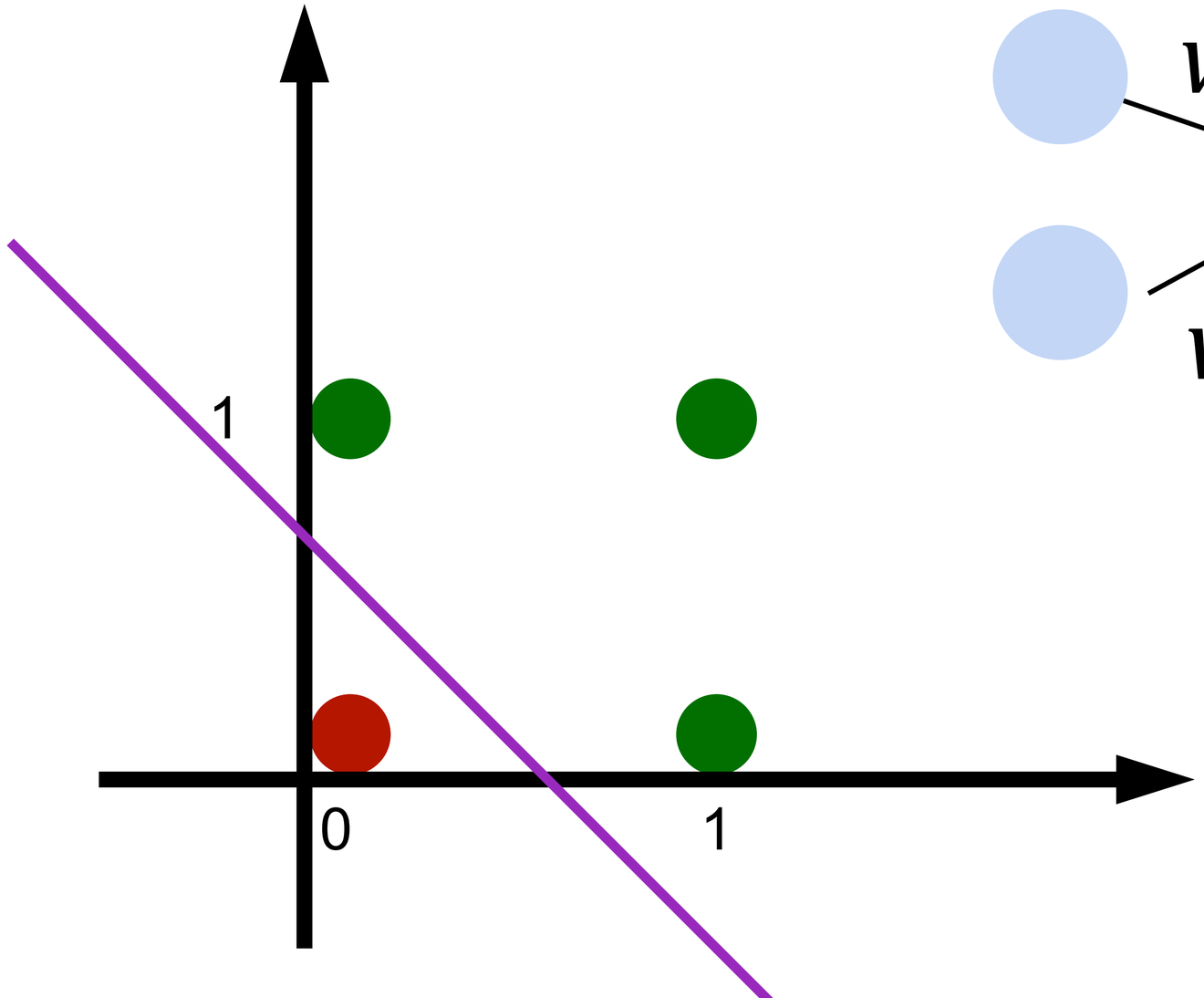
Learning OR function using perceptron

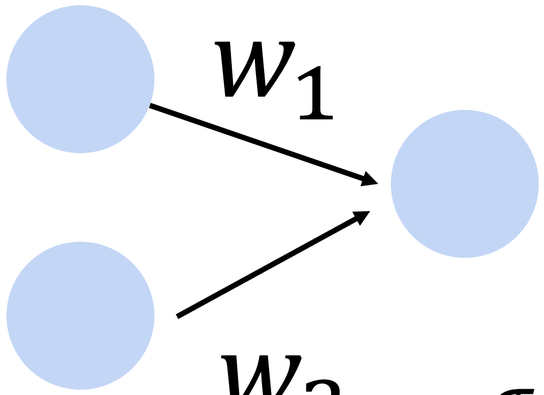
The perceptron can learn an OR function



Learning OR function using perceptron

The perceptron can learn an OR function





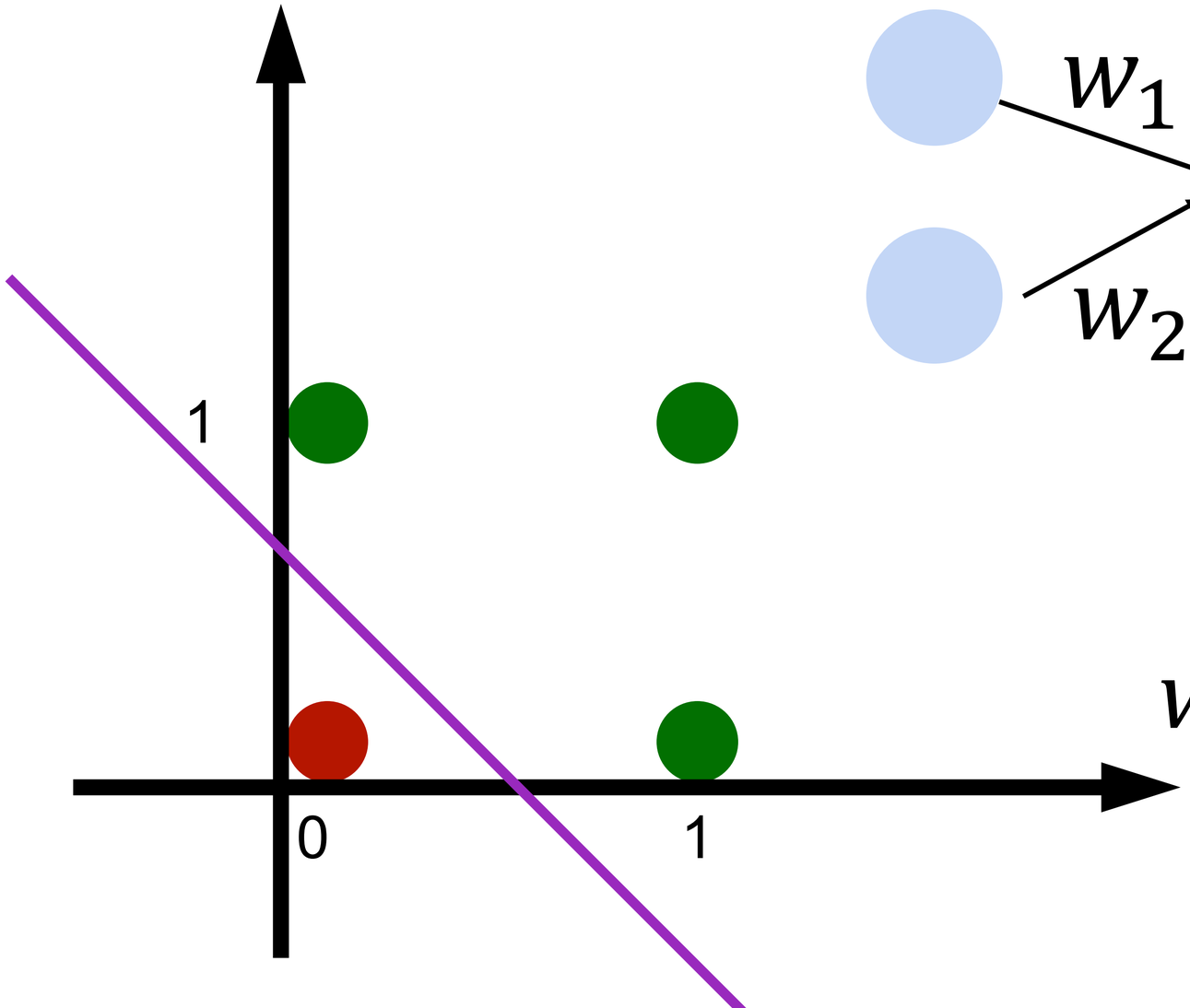
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

What's w and b?

Learning OR function using perceptron

The perceptron can learn an OR function



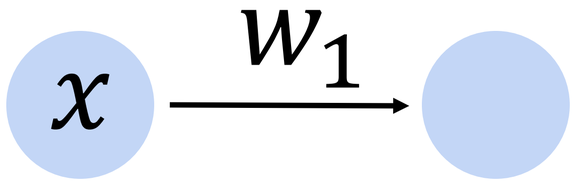
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(x) = \begin{cases} 1 & if\, x > 0 \\ 0 & otherwise \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -0.5$$

Learning NOT function using perceptron

The perceptron can learn NOT function (single input)



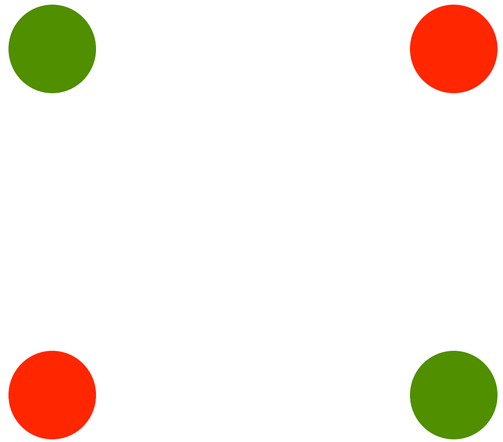
Output $\sigma(xw_1 + b)$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

$$w_1 = -1, b = 0.5$$

XOR Problem (Minsky & Papert, 1969)

The perceptron cannot learn an XOR function
(it can only generate linear separators)

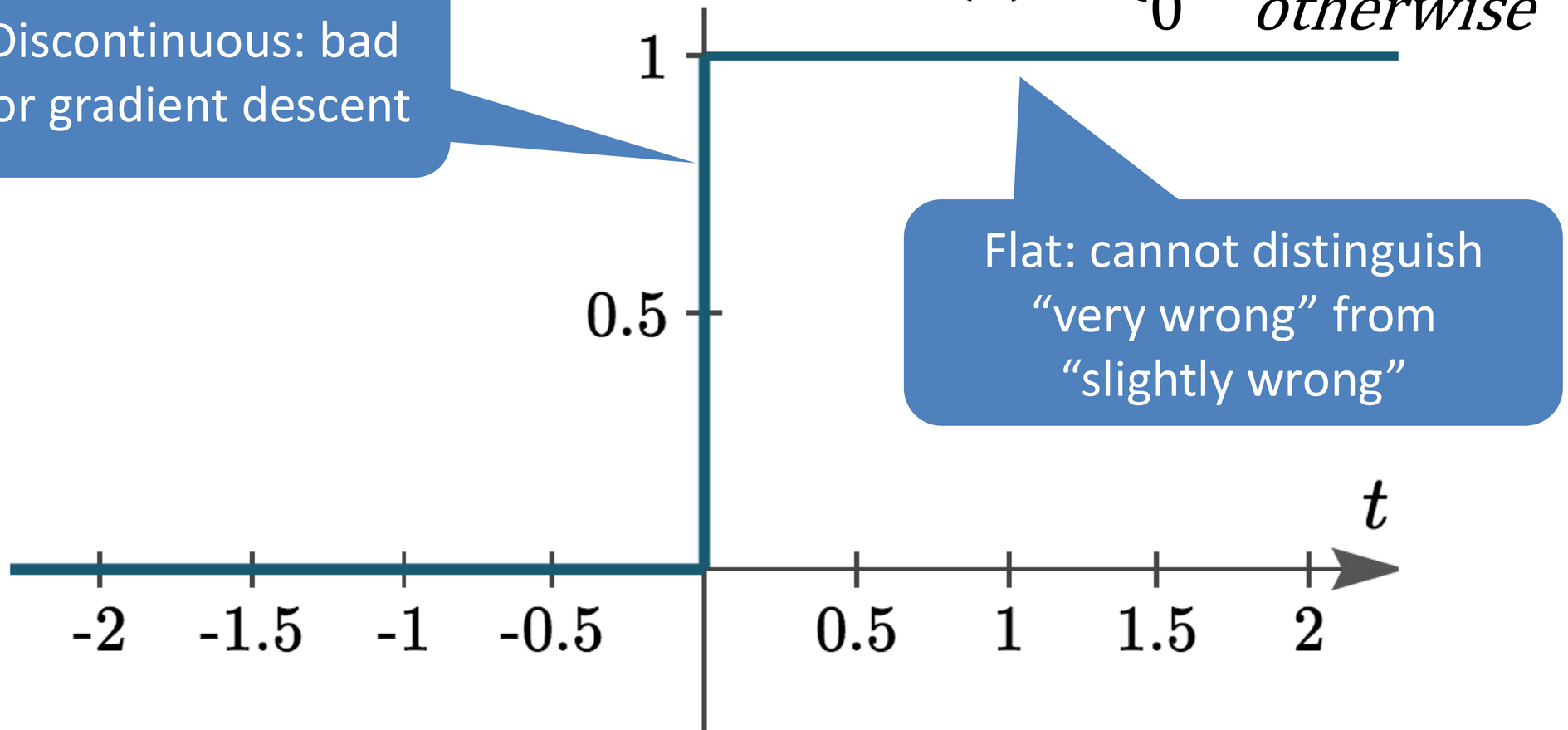


This contributed to the first AI winter

Activation Functions: Beyond Hard Thresholds

Perceptron uses hard threshold/step function

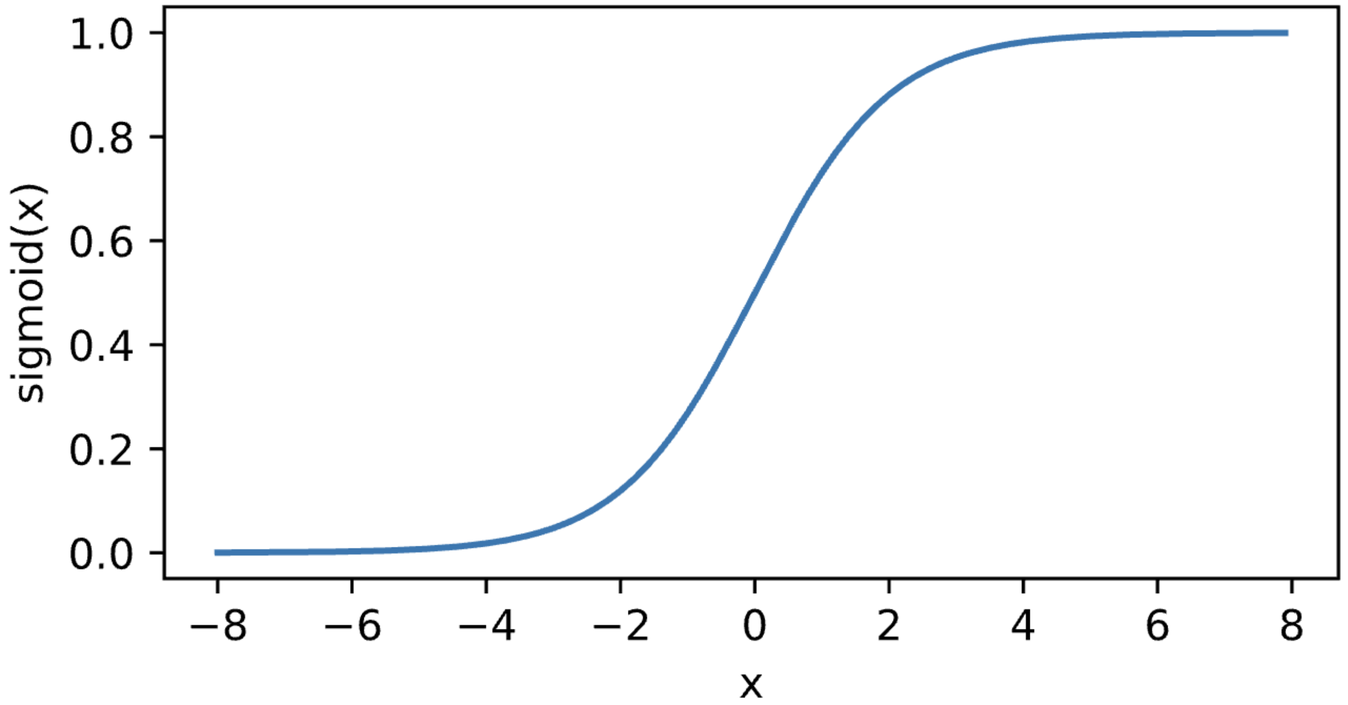
$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$



Sigmoid/Logistic Activation

Map input into [0, 1], a **soft** version of $\sigma(z) = \begin{cases} 1 & if\, z > 0 \\ 0 & otherwise \end{cases}$

$$\sigma(z) = \text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$$

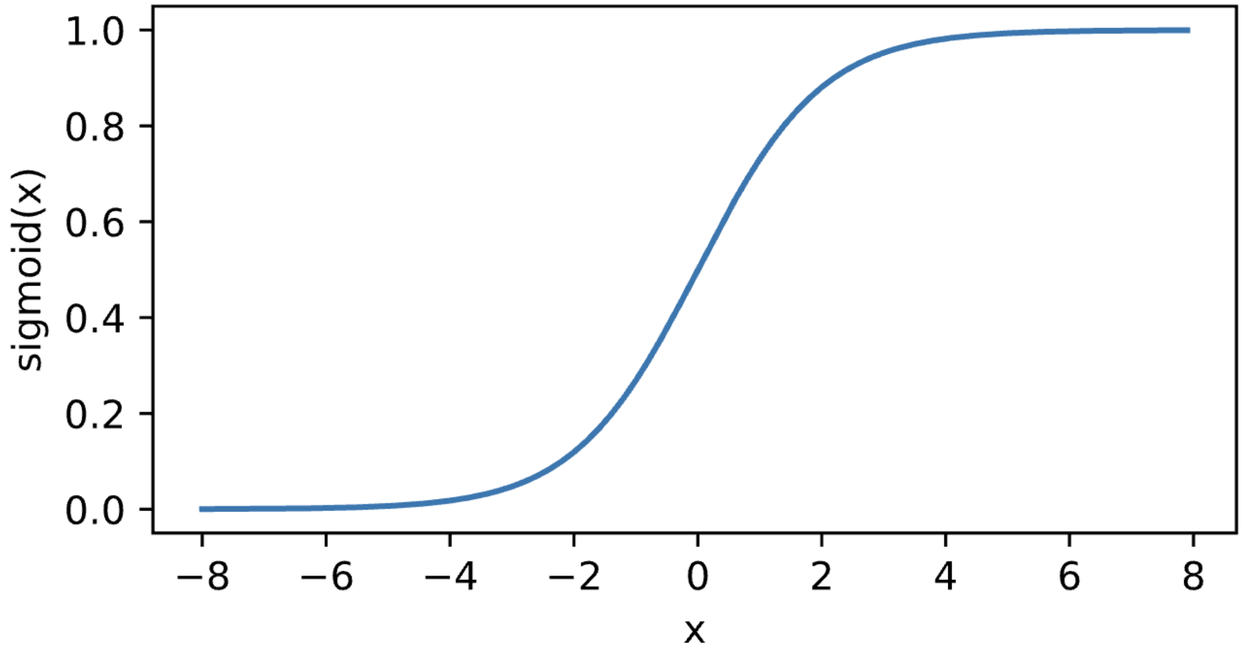


Aside: Logistic Regression

$$\mathbf{x} \in \mathbb{R}^d, y = \{-1, +1\}$$

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$$

$$p(y = -1|\mathbf{x}) = 1 - \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T\mathbf{x})}$$



Aside: Logistic Regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ $\mathbf{x} \in \mathbb{R}^d, y = \{-1, +1\}$

Training: maximize likelihood estimate (on the conditional probability)

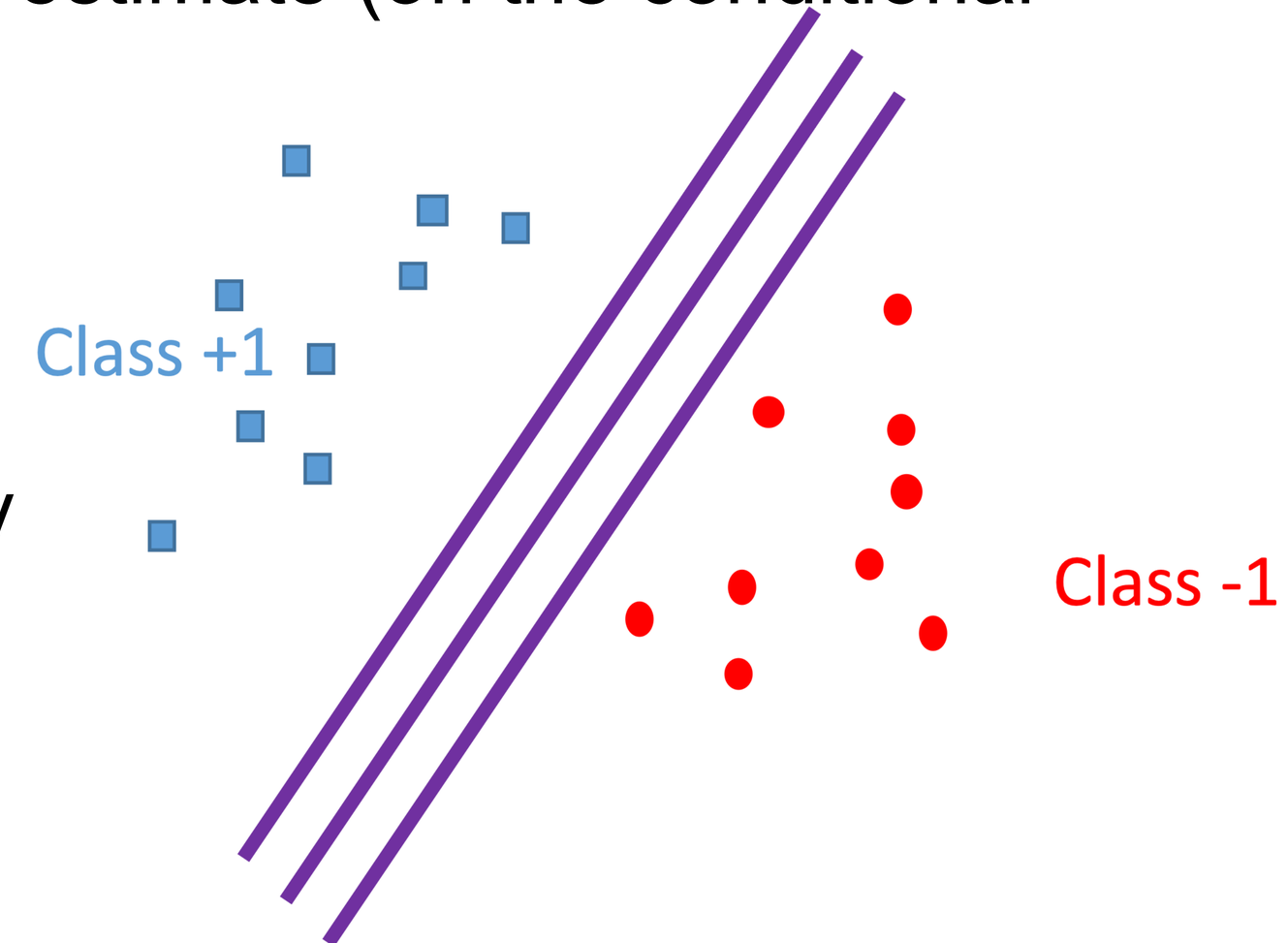
$$\max_{\mathbf{w}} \sum_i \log \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

Aside: Logistic Regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ $\mathbf{x} \in \mathbb{R}^d, y = \{-1, +1\}$

Training: maximize likelihood estimate (on the conditional probability)

When training data is linearly separable, many solutions

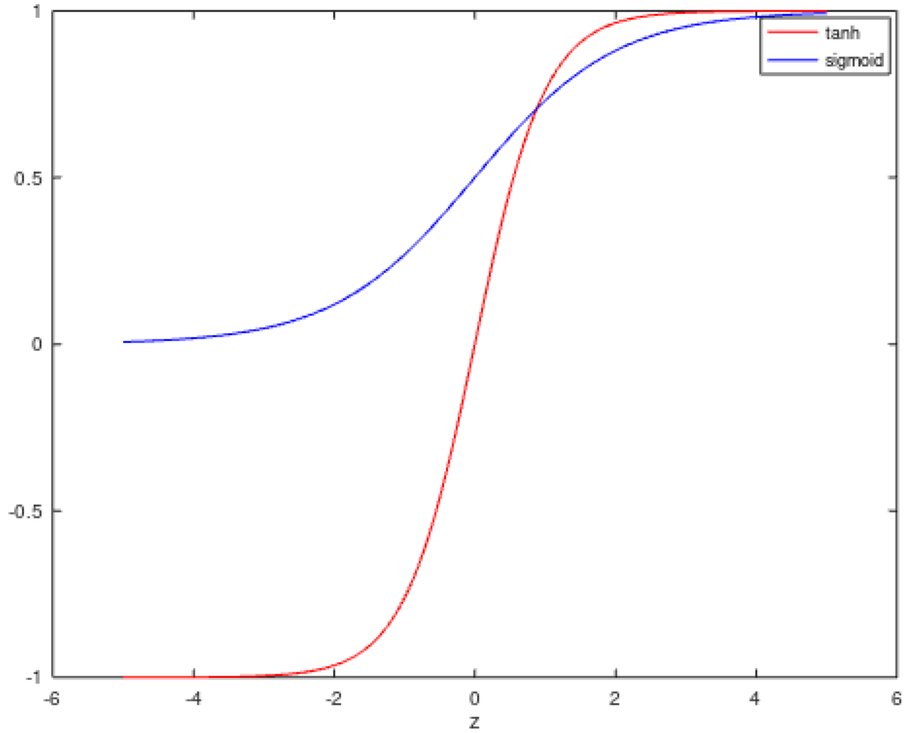


Tanh Activation

Map inputs into (-1, 1)

$$\sigma(z) = tanh(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}$$

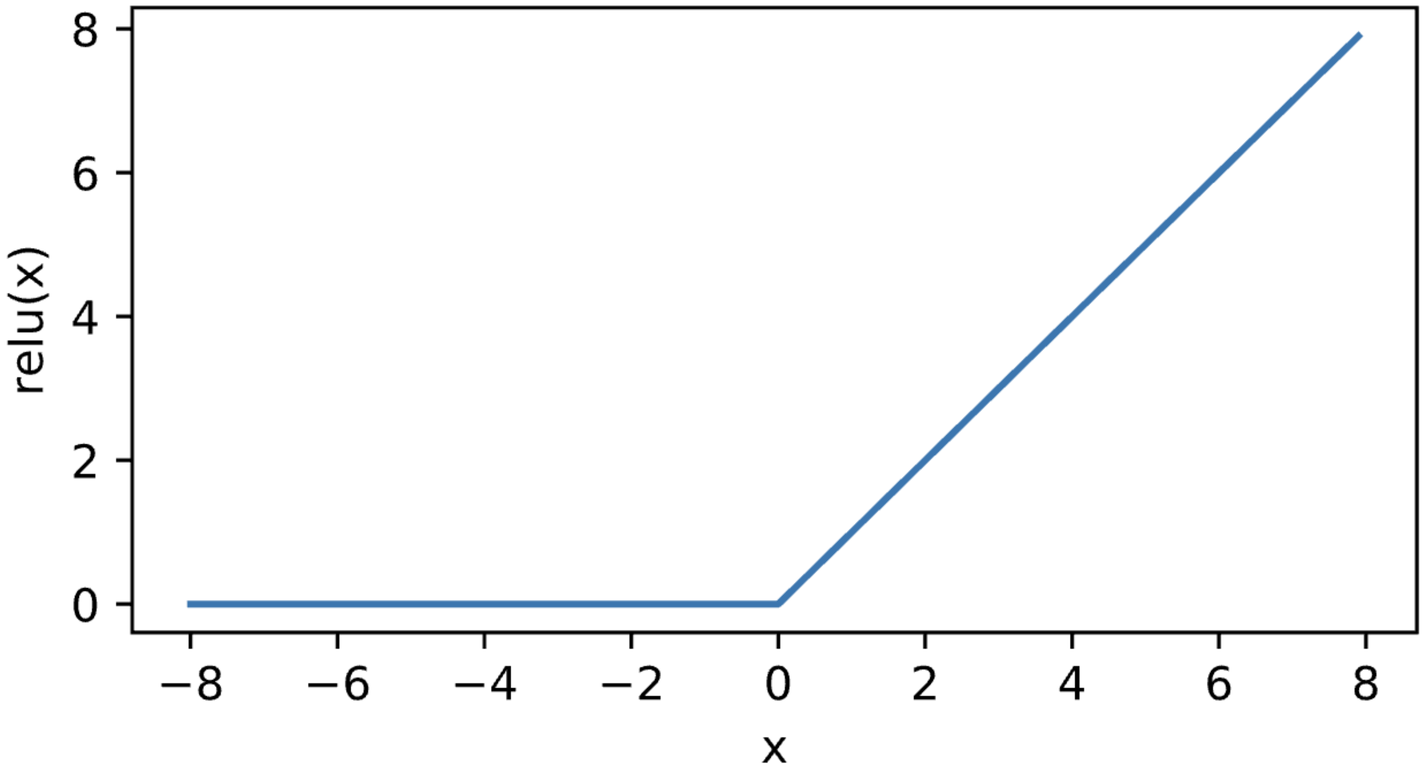
$$tanh(z) = 2sigmoid(2z) - 1$$



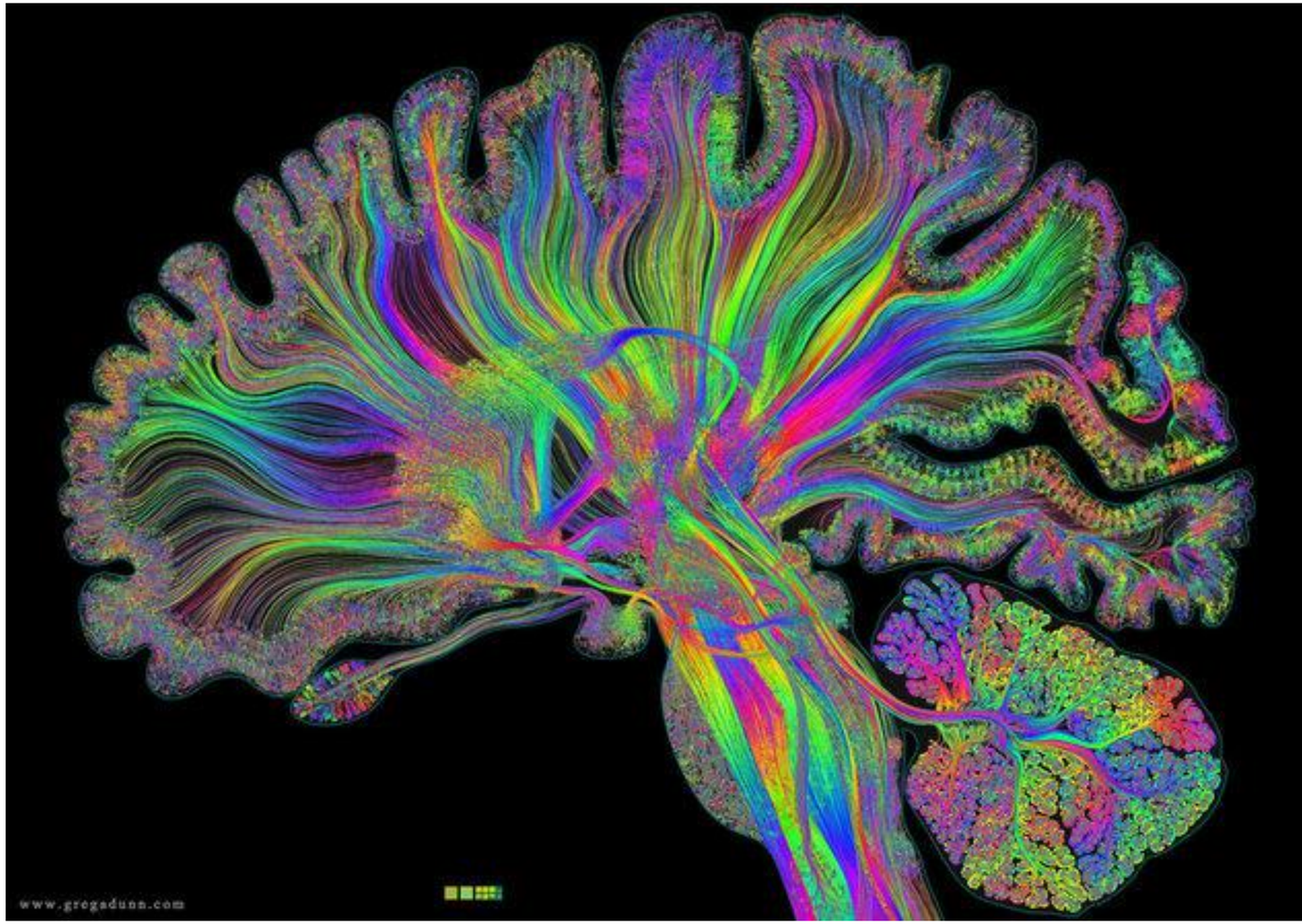
ReLU Activation

ReLU: rectified linear unit (commonly used in modern neural networks)

$$\mathrm{ReLU}(x) = max(x, 0)$$



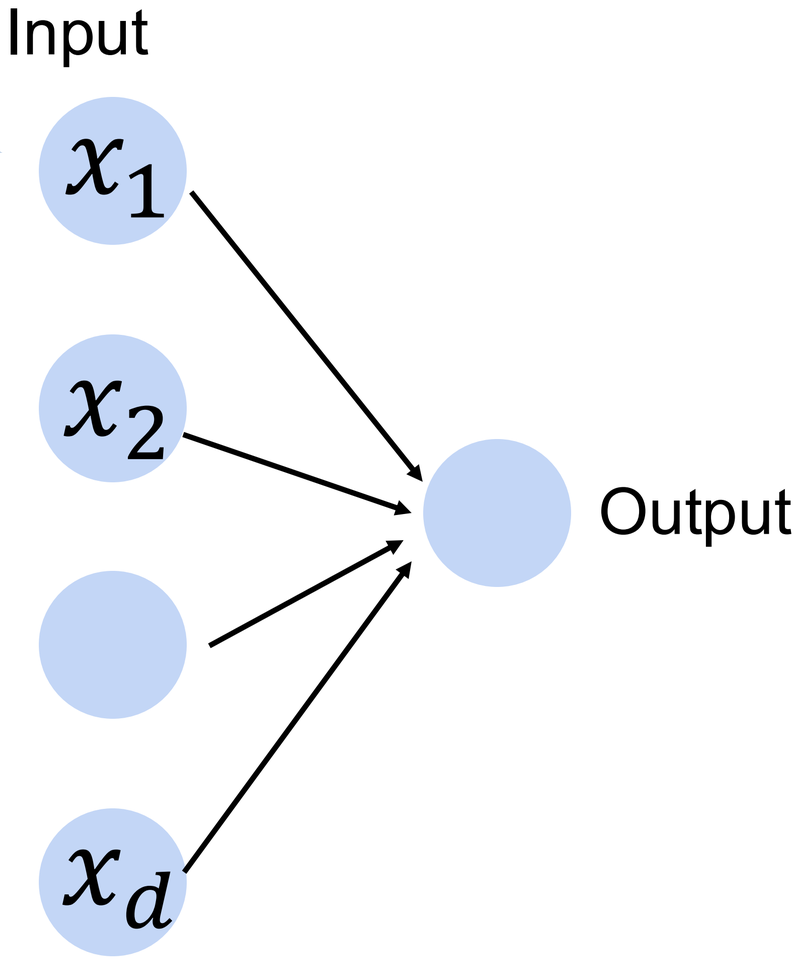
Multilayer Perceptron



The perceptron has one layer of weights

Each input node receives one scalar feature (e.g., one pixel)

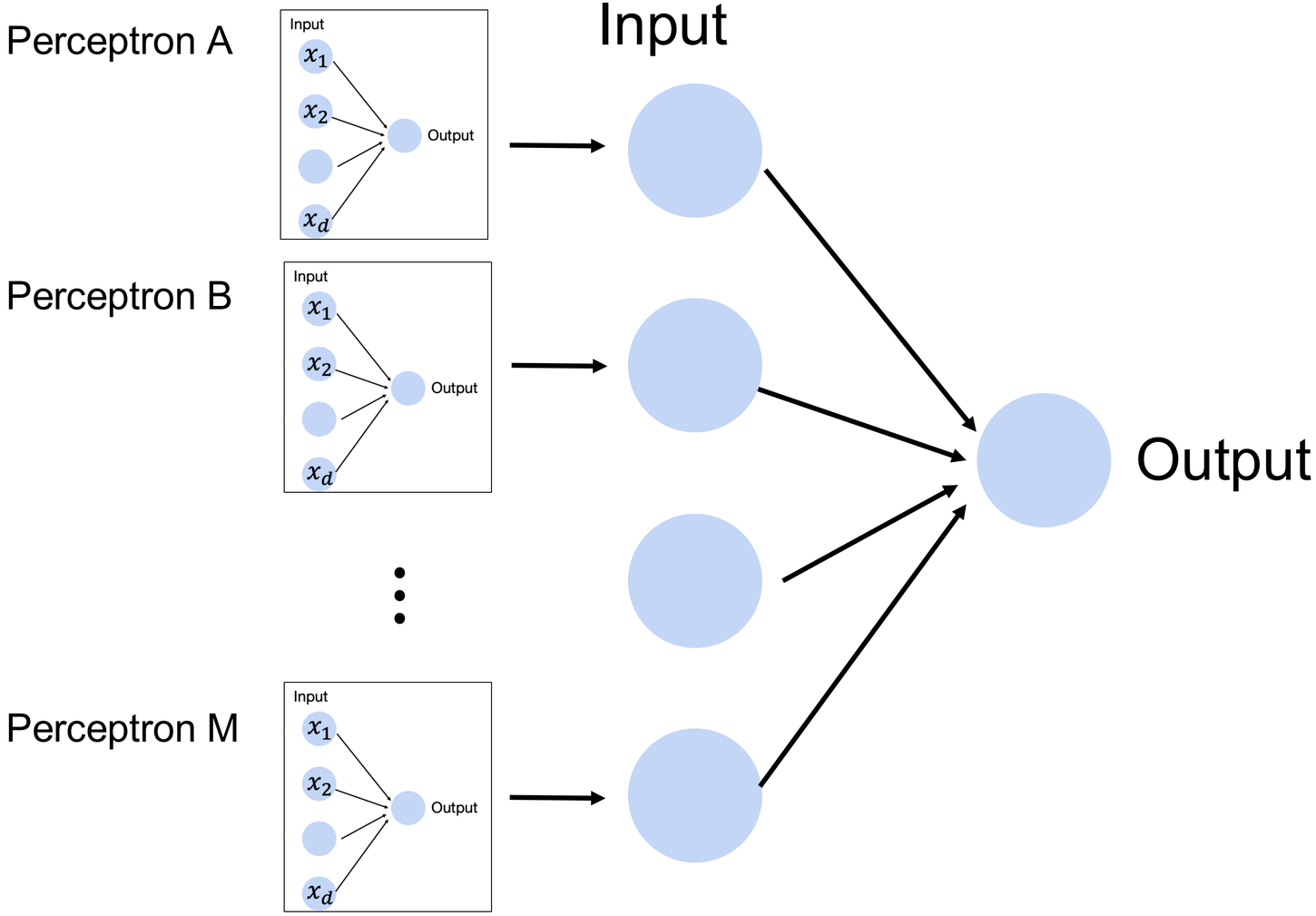
Cats vs. dogs?



Beyond one layer

Big Idea: take our inputs from other perceptrons!

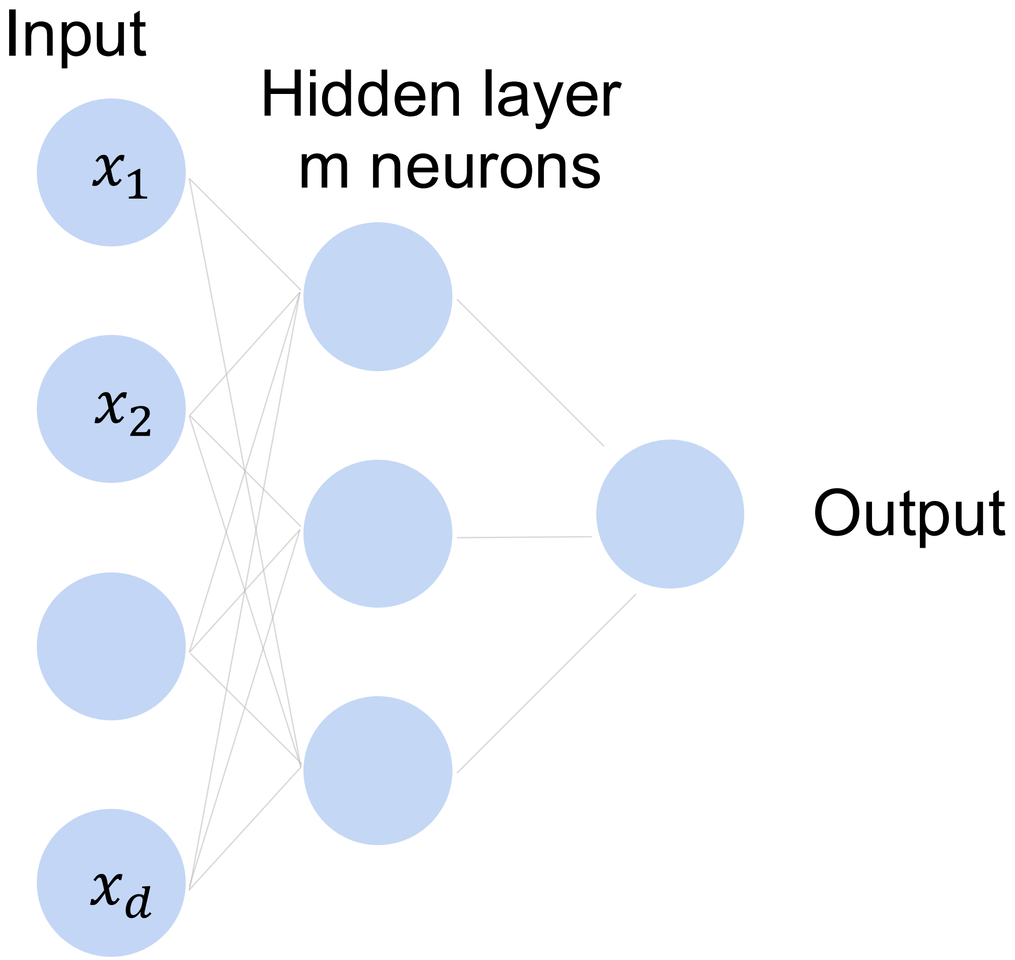
Cats vs. dogs?



Multilayer Perceptron with One "Hidden" Layer

How to classify

Cats vs. dogs?



Single Hidden Layer

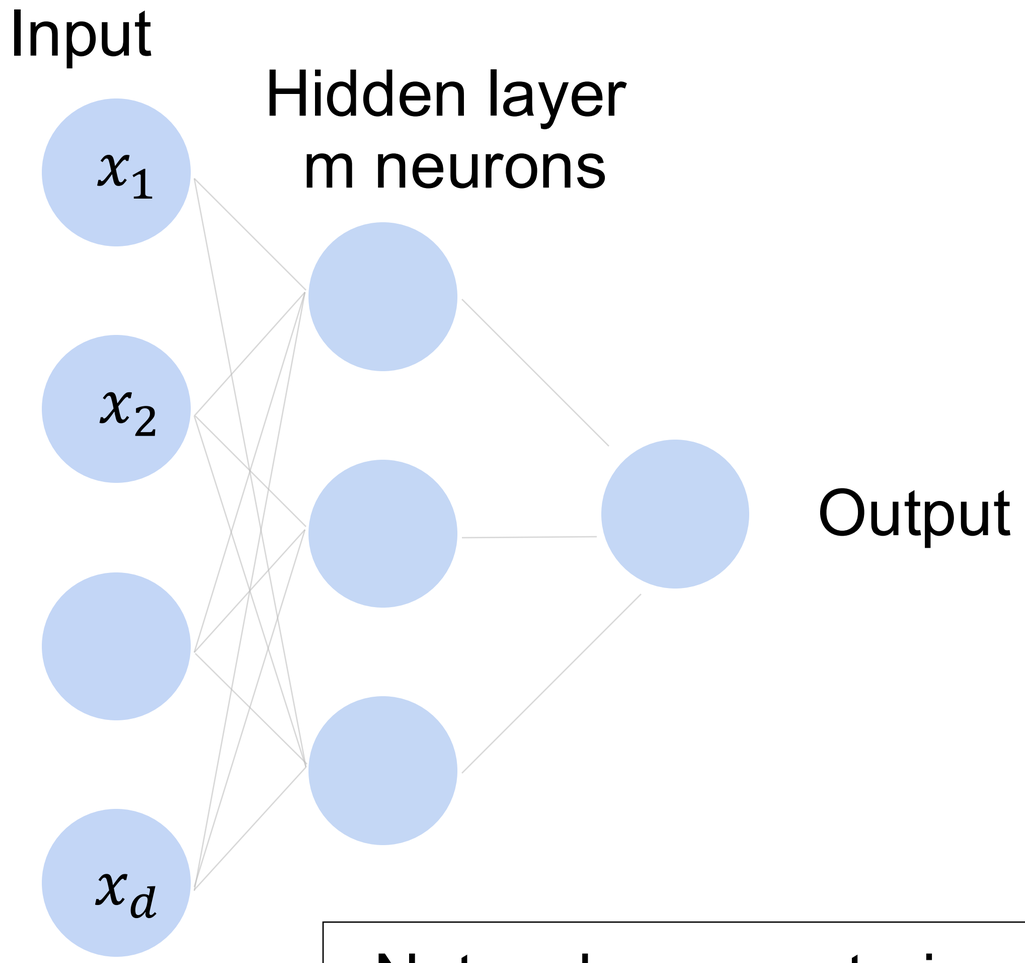
- Input $x \in \mathbb{R}^d$
- Hidden $W_h \in \mathbb{R}^{m \times d}, b_h \in \mathbb{R}^m$
- Intermediate output

$$h = \sigma(W_h x + b_h)$$

σ is an element-wise activation function

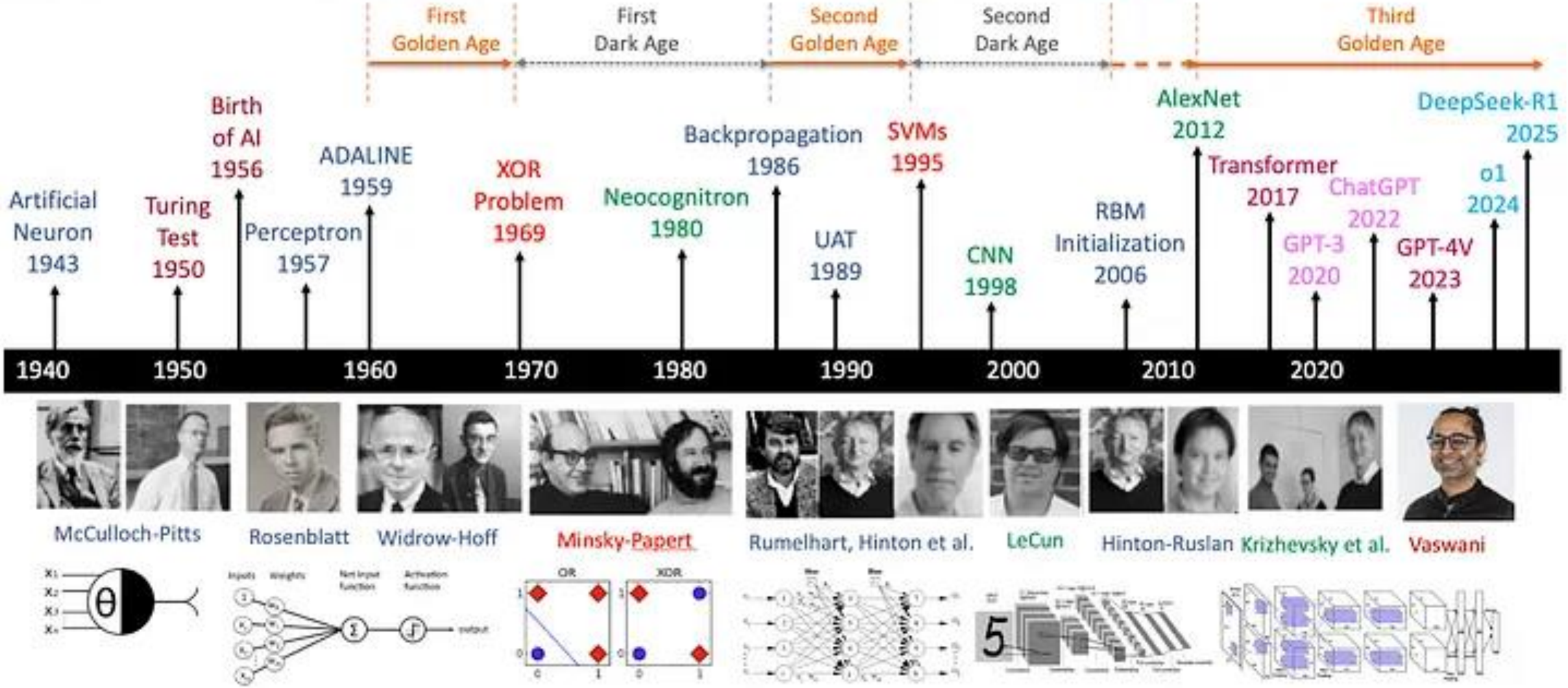
- Final output/final perceptron

$$\sigma(\langle w_f, h \rangle + b_f)$$



Network parameterized by W_h, b_h, w_f, b_f

A Brief History of AI with Deep Learning



https://medium.com/@lmpo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b

What we've learned today…

- (Single-layer) Perceptron
  - Motivation
  - Training
  - Representing AND, OR, NOT
  - Different activation functions
- Multilayer Perceptron
- Brief history of neural networks