COMP307/AIML420 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

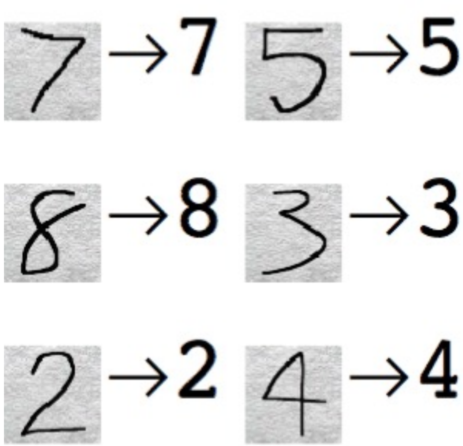
Neural Networks 1: Perceptron and MLP

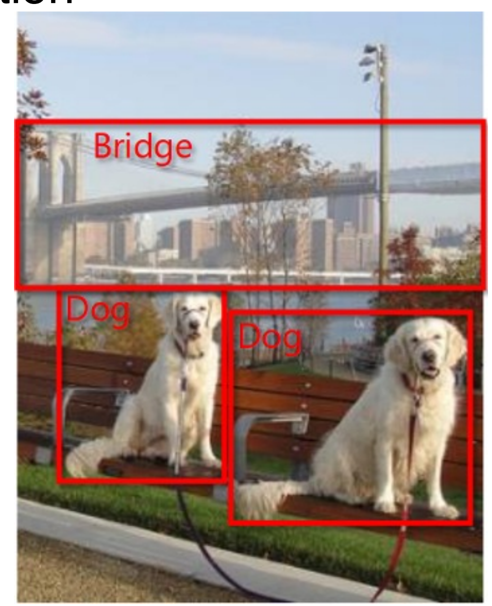
Outline

- Why neural networks / current status
- Origin
- Perceptron
- Perceptron learning
- What can (not) perceptron learn
- Extending the perceptron to an MLP

Why Neural Networks?

- Many applications, such as
  - Generative models:
    - Large language models (LLMs) → ChatGPT, Gemini
    - Image and video generation → stable diffusion, Sora
  - Computer Vision/Image processing
    - Autonomous vehicles
    - Image classification
    - Anomaly detection





Where are we going?

- SoA rapidly advancing

- Is Artificial General Intelligence (AGI) close?
  - Large language models (LLMs) already know more than humans do
    - LLMs can pass university exams
  - Computers don't require 25 years of learning: just copy
  - Recursive self improvement
- AI alignment problem
  - AI may find unexpected solutions that are not good for us

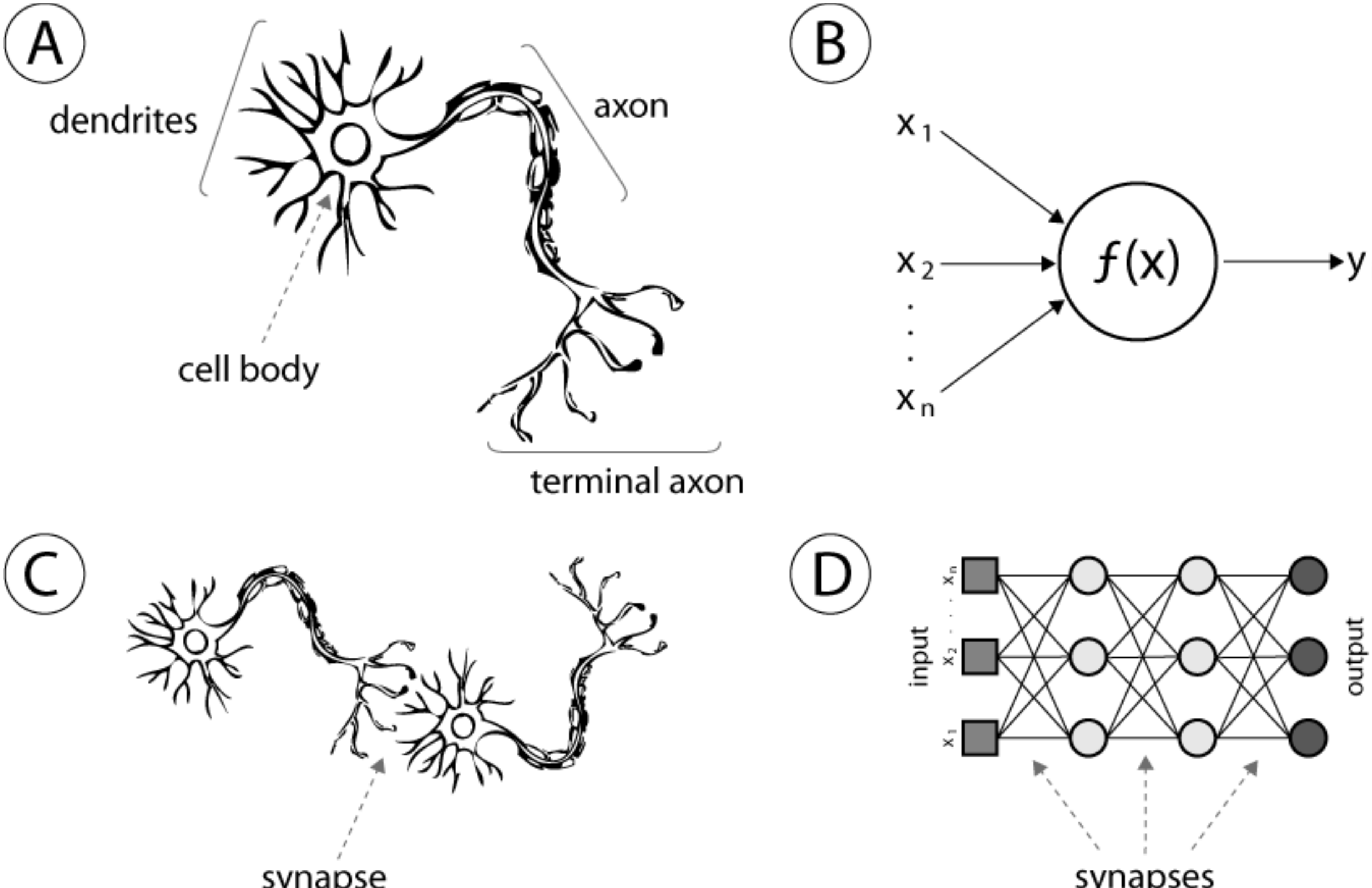
Origin

- Human brain shows amazing capability in
  - Learning
  - Perception
  - Adaptability
  - Parallel processing
  - …

- A bit slow, though
- 86 billion neurons
  - vs 200 billion stars in our galaxy and 200 billion galaxies
  - About 0.7 quadrillion neuronal connections = parameters

- Simulate human brain to achieve the above functionalities

Origin


A
dendrites
axon
cell body
terminal axon
B
x_1
x_2
x_n
$f(x)$
y_i
C
synapse
D
input
x_1
x_2
x_n
output
synapses

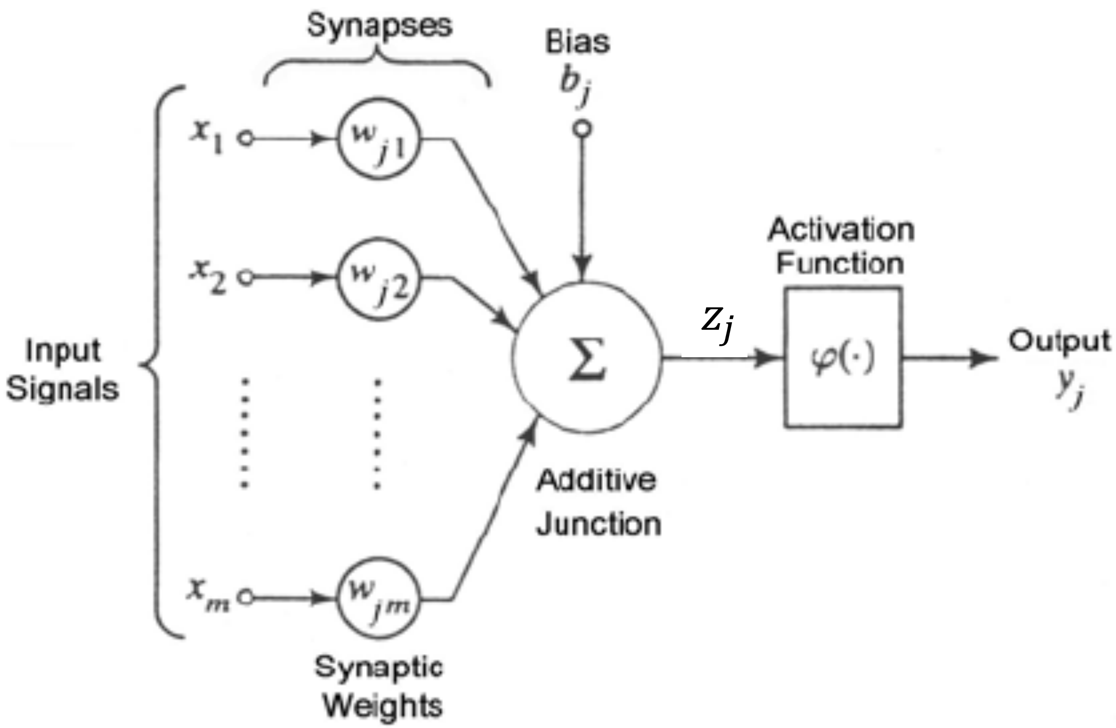
Origin

- Facts about human brain
  - About 10^{11} (100 billion) neurons, massively connected
  - Each neuron is connected to just under 10^4 other neurons
  - About 10^{15} (a quadrillion) connections (parameters) in total
  - Brain message passing million times slower than electronic circuits
    - 200 Hz ”clock rate”
    - But can observe relative delay between ears down to 10 microseconds
  - Slow but very efficient for complex decision making
  - Usually less than 100 serial stages
    - 100 step rule (half second)

- In contrast:
  - Honeybee: 1 million neurons, 1 billion synapses
  - Mouse: 70 million
  - Crocodile: 80 million
  - Grey parrot: 1.5 billion
  - Dog: 4 billion

MRI tractography

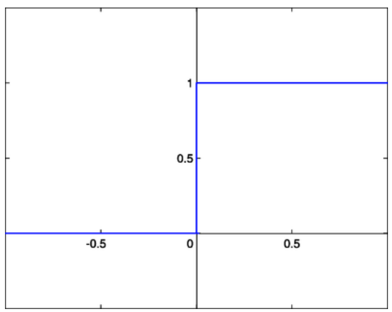
Artificial Neuron



$$z_j = \sum_{i=1}^{m} w_{ji} x_i + b_j$$

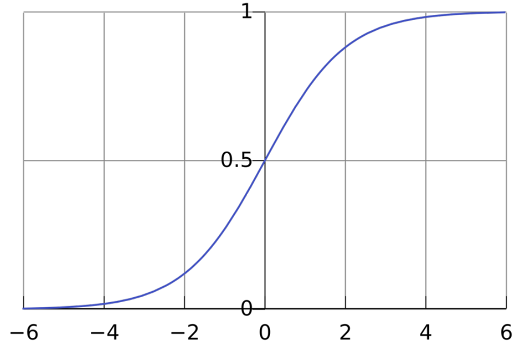
$$y_j = \varphi(z_j)$$

Activation Functions



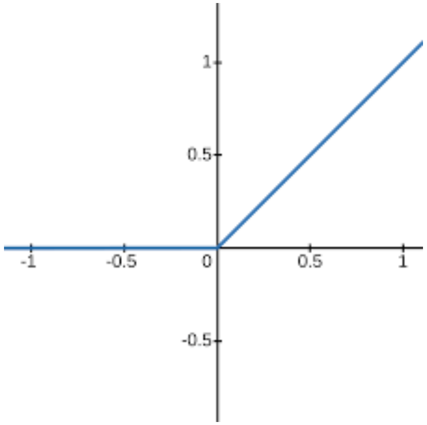
$$y = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{elsewhere} \end{cases}$$

threshold



$$y = \frac{1}{1 + e^{-\beta x}}$$

sigmoid

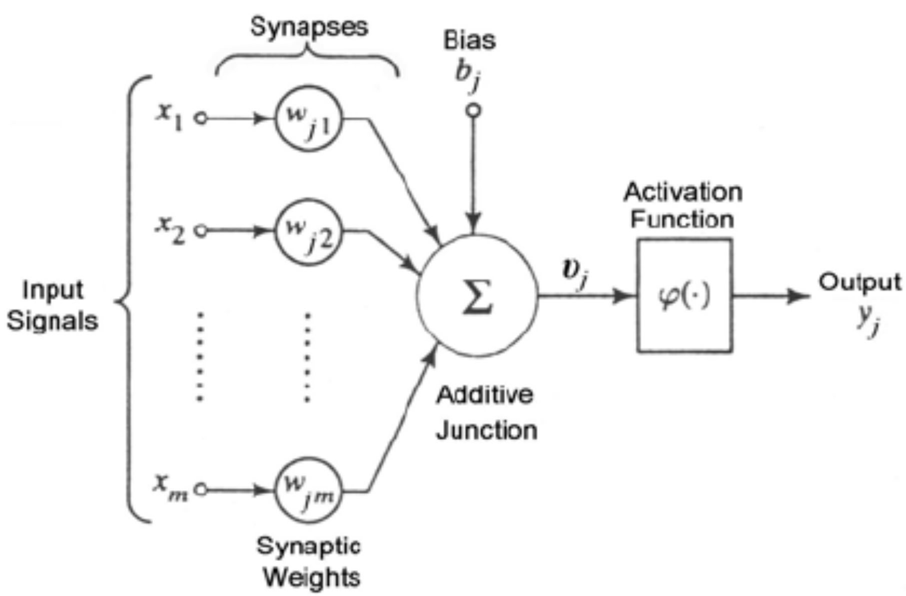


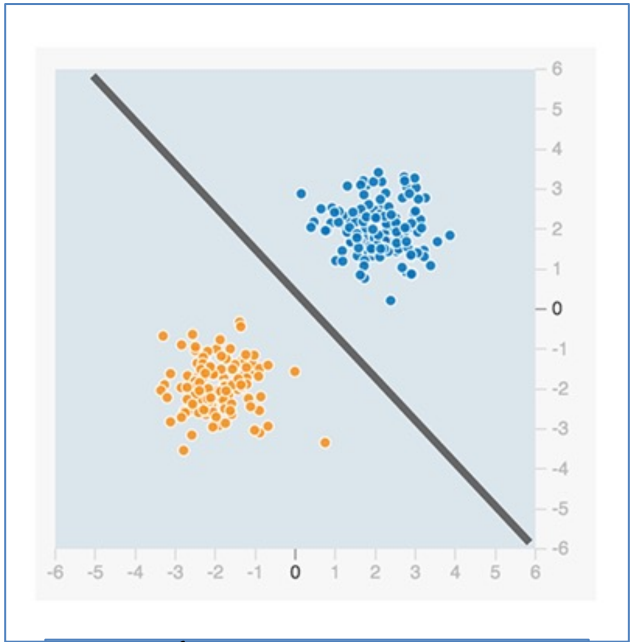
$$y = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{elsewhere} \end{cases}$$

ReLu
(rectified linear unit)

Perceptron

- Perceptron is single artificial neuron for binary classification
  - Invented 1943 (McCulloch and Pitts)
  - Real-valued inputs binary output
  - Threshold activation function



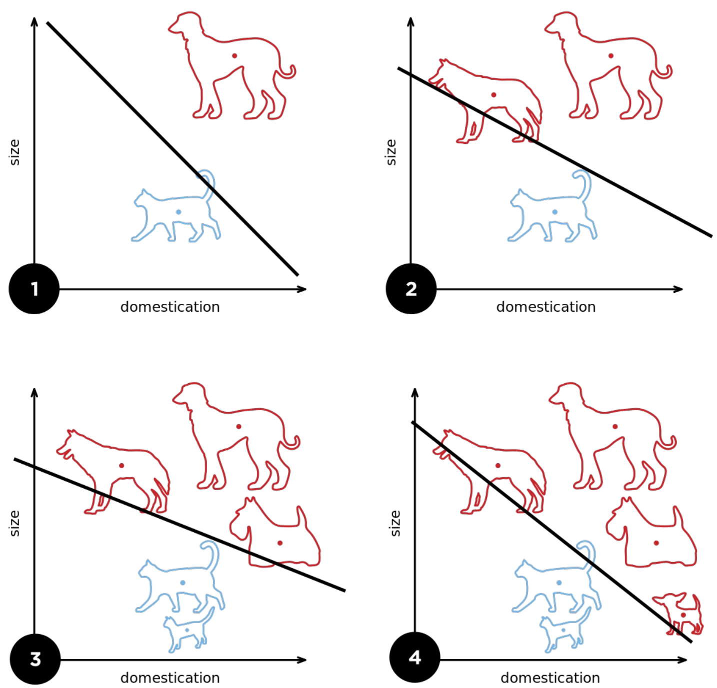


$$y = \begin{cases} 1, \text{if } x_1 + x_2 > 0, \\ 0, \text{otherwise.} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_{ji} x_i + b_j > 0, \\ 0, & \text{otherwise} \end{cases}$$

Perceptron

- To perform linear classification
  - Two inputs: a line
  - Three inputs: a plane
  - Etc.
- Can do on-line learning
  - Update w_{ji} and b_j along with new examples



Learning Perceptron

- How to get the optimal weights and bias?
- Only consider accuracy
  - Optimal if 100% accuracy on training set
  - Can have many optimal solutions
- To simplify notation, transform bias to a weight:
  - $w_{j0} = b_j$ with constant $x_0 = 1$

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=1}^{m} w_{ji} x_i + b_j > 0, \\ 0, & \text{otherwise} \end{cases}$$

$$\boxed{b_j = w_{j0} \cdot 1 = w_{j0} x_0}$$

$$y_j = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_{ji} x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$

Learning Perceptron

- Context:
  - Initialise weights and threshold randomly (or all zeros)
  - Given a new example $(x_1, x_2, \ldots, x_m, d)$
    - Input feature vector: $(x_1, x_2, \ldots, x_m)$
    - Output (class label): d
    - Predicted (by perceptron) output y

$$\boxed{y = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_i x_i > 0, \\ 0, & \text{otherwise} \end{cases}}$$

- Basic learning algorithm:
  - If $y = 0$ and $d = 1$:
    - increase $b = w_0$, increase w_i for positive x_i, decrease w_i for negative x_i
  - If $y = 1$ and $d = 0$:
    - decrease $b = w_0$, decrease w_i for positive x_i, increase w_i for negative x_i
  - Repeat for each new example until the desired behaviour is achieved
  - Can also repeat all data and start again (multiple epochs)

Learning Perceptron

- Implementation:
  - Initialise weights and threshold randomly
  - Given a new example $(x_1, x_2, \dots, x_m, d)$
    - Input feature vector: $(x_1, x_2, \dots, x_m)$
    - Output (class label): d
    - Predicted output y

$$y = \begin{cases} 1, & \text{if } \sum_{i=0}^{m} w_i x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$

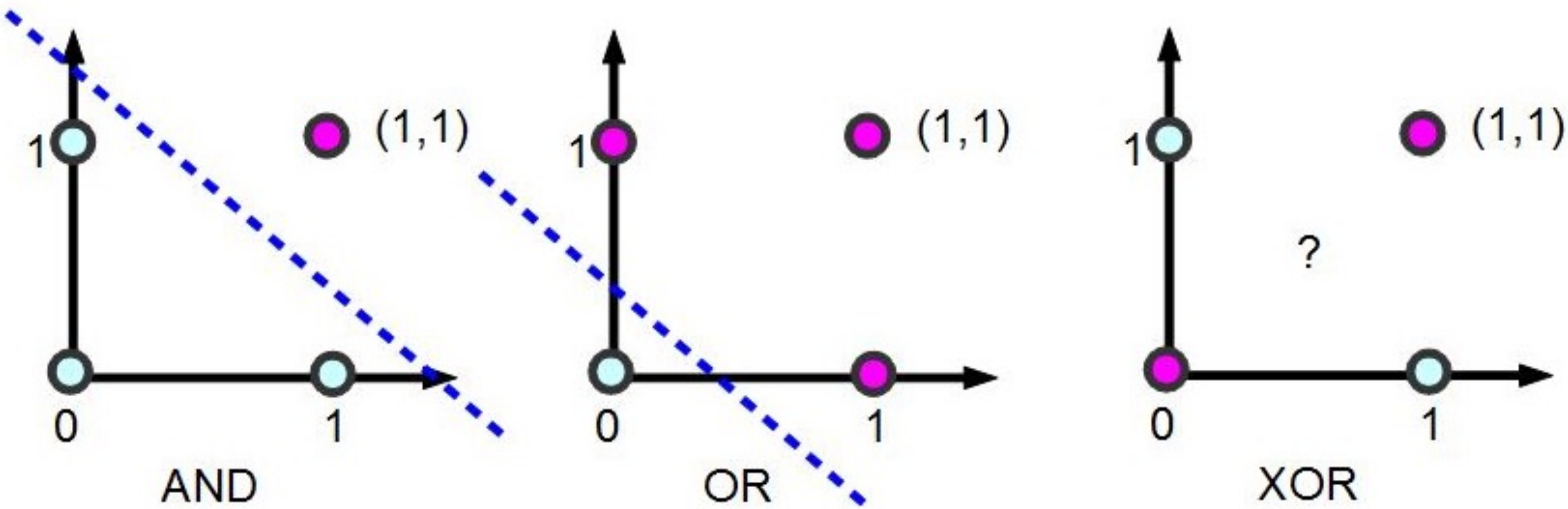
- Learning algorithm:

$$w_i \leftarrow w_i + \eta(d - y)x_i, \qquad i = 0, 1, 2, \cdots, m$$

  - Where $\eta \in [0,1]$ is called the learning rate
  - Repeat the process, possibly over multiple epochs, until convergence or pre-set maximum steps

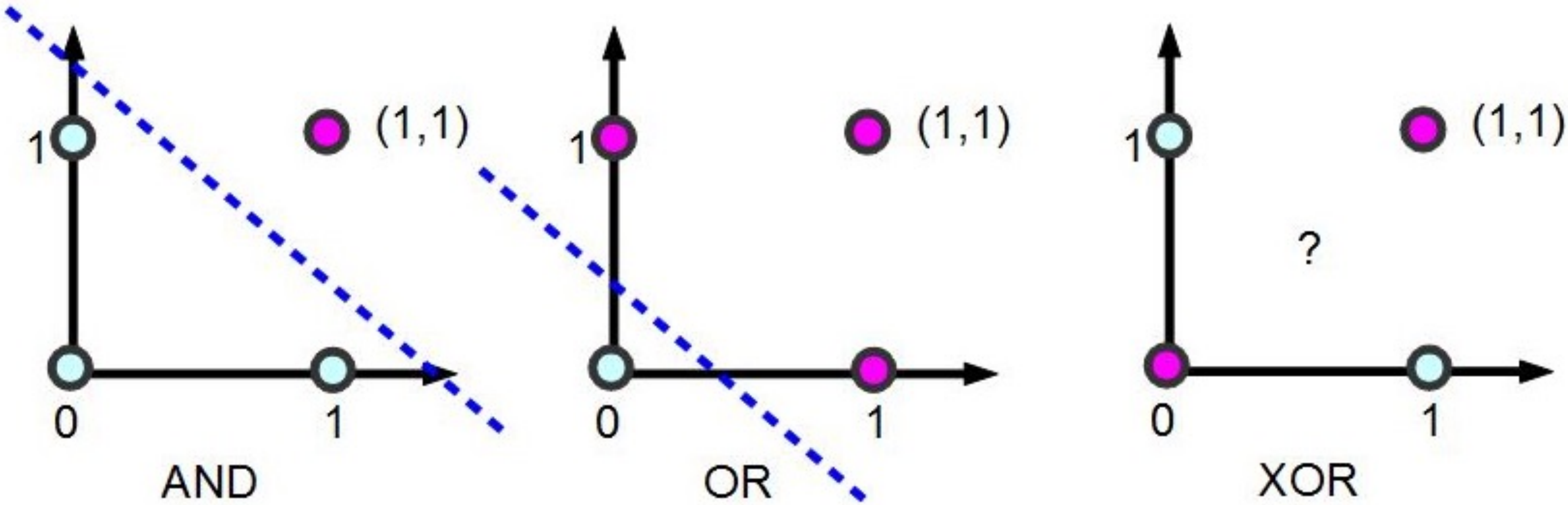
Problem with Perceptron

- What can the perceptron learn?



Problem with Perceptron

- What can the perceptron learn?



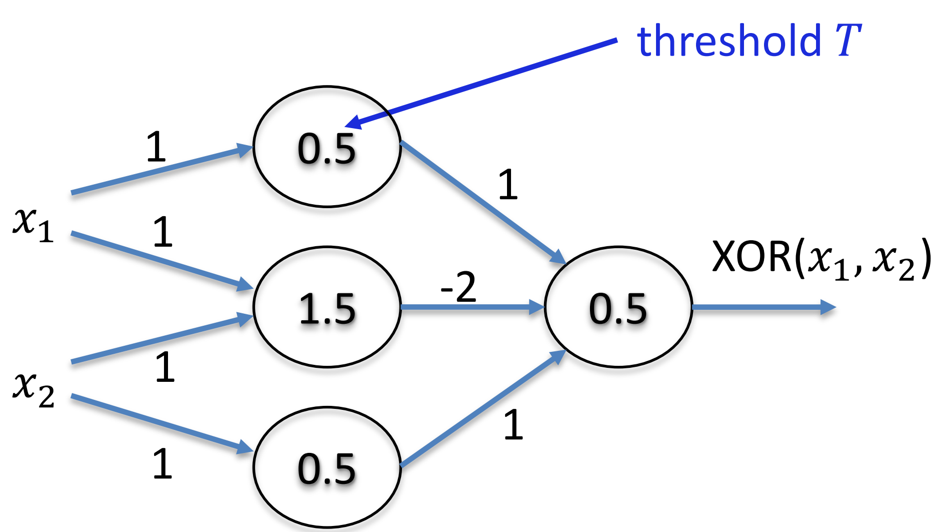
- *Perceptron convergence theorem*: The perceptron learning algorithm will converge **if and only if** the training set is linearly separable.
- Cannot learn XOR (Minsky and Papert, 1969)

Making it work for XOR

- A solution with two layers (three neurons and one neuron, respectively)

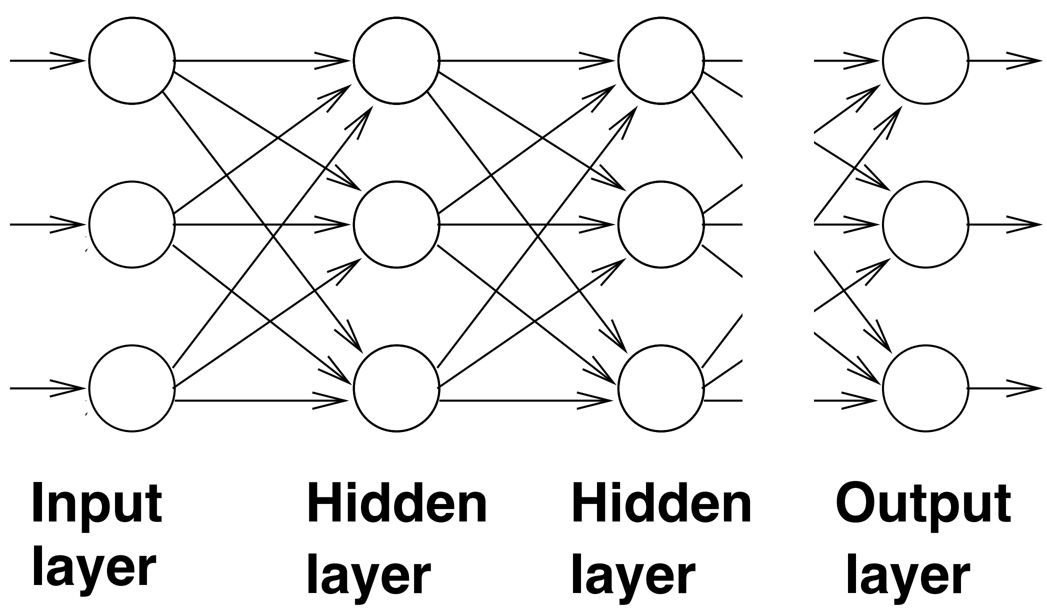
x_1	x_2	y =class
0	0	0
1	0	1
0	1	1
1	1	0

$$y = \begin{cases} 1, & \text{if } \sum_i w_i x_i - T > 0 \\ 0, & \text{otherwise} \end{cases}$$



Multilayer Perceptron (MLP)

- We saw that more neurons/layers can do more
- MLP: any number of “hidden” layers with any number of nodes
- All outputs of previous layer connected to all neurons of a layer
- All connections associated with a weight, nonlinearity operates on sum of weighted previous-layer outputs
  - Each layer requires a weight matrix (matrix of parameters), W
  - Layer operation is now $y = \phi(Wx + b)$



Why MLP?

- We want to approximate some desired function
  - Consider a D-dimensional vector of binary inputs (zeros and ones)
    - 2^D possible input vectors
    - *A detector for each of the* 2^D *possible inputs*:
      - Consider a detector neuron for input vector v with $\mathrm{v_j} \in \{0,1\}$
      - Set the weights for neuron i to $\mathrm{w_{ji}} = 4v_j - 2$ and threshold to $T = D + \sum_j w_{ji} - 0.5$
      - Neuron i will output a $y_i = 1$ only if the input is $x = v$; else $y_i = 0$
    - This is a universal approximator in this binary space
  - More generally, wide one-layer networks can be universal function approximators

- Despite this, deep rather than wide networks are used in practice as deep networks are easy to train
  1. Define a mathematical objective function (what do we want)
     - Objective function has *parameters* as argument (for given database)
  2. Make network differentiable, so we can search for the ~~best~~ good parameters by sliding down the objective function with gradient descent

- MLP is conceptually simplest deep network

Summary

- Neural networks now ubiquitous
  - Text / image /video generation, image analysis, control, ···
- Alignment will be a serious issue
- Perceptron – the simplest neural network
- Learning for a perceptron and its limitation
- Multi-layer perceptron (MLP), which is a standard component of modern networks

- Another view of similar content