

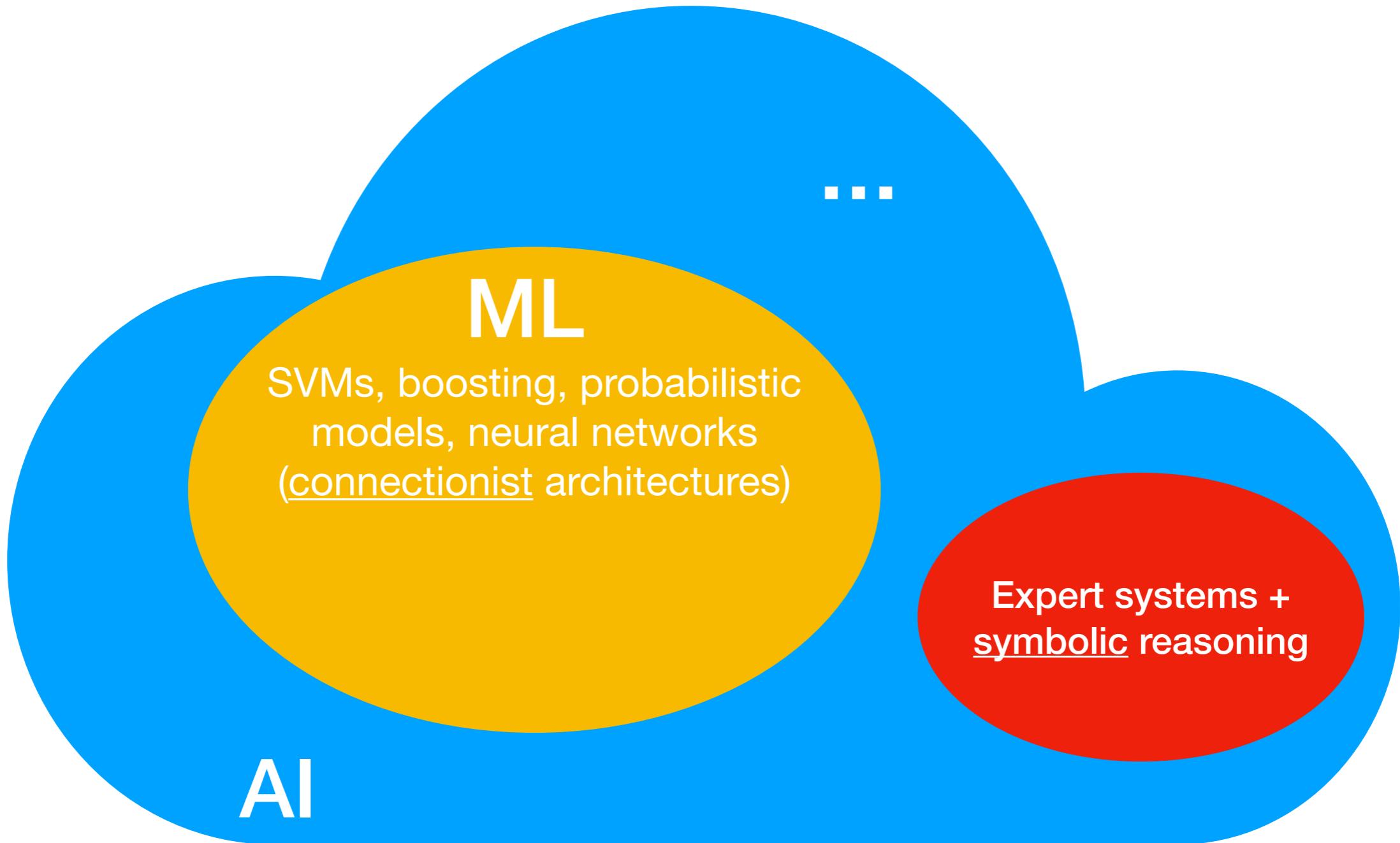
# **CS/DS 541: Class 1**

Jacob Whitehill

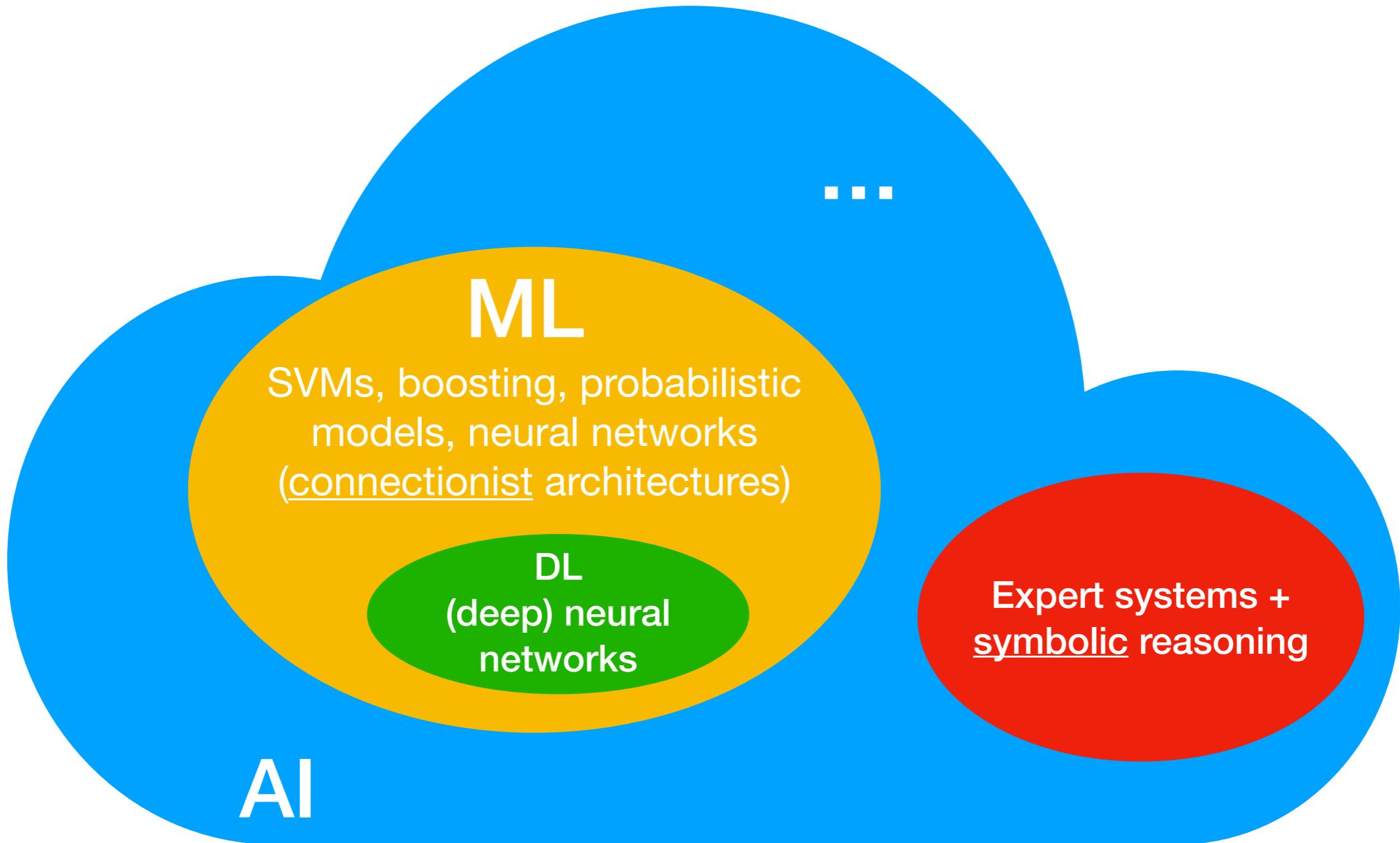
**DL** ∈ **AI**



# **DL** ∈ **AI**

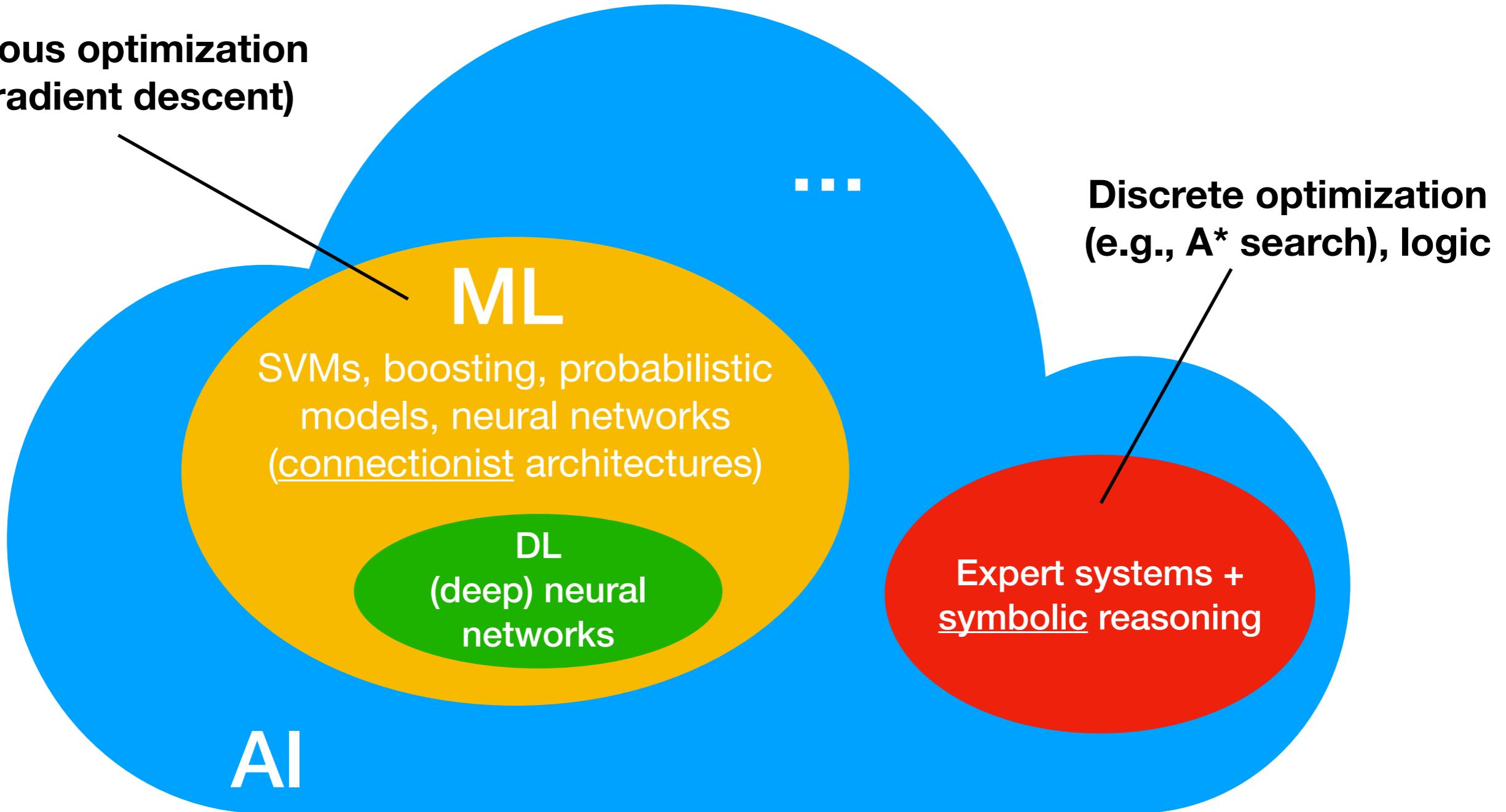


# **DL** ∈ **AI**



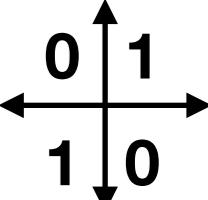
# **DL** ∈ **AI**

**Continuous optimization**  
(e.g., gradient descent)



# ML and DL

- Big developments:
  - 1940-50s: early algorithms for artificial neural networks (NNs), including the Perceptron algorithm (2-layer NN with discontinuous activation function).
  - 1960s-70s: disillusionment due to failure to solve simple problems (e.g., XOR).
  - 1980s: back-propagation to optimize multi-layer NNs with continuous activations.
  - 1990s: disillusionment due to complexity in optimizing non-convex functions.
  - 2000s: prominence of SVMs, probabilistic & Bayesian methods
  - 2010s-today: renaissance of neural networks.



# Demos

- NVIDIA's BB8 self-driving car.
- Google Duplex personal assistant.
- DeepMind's Atari game player.
- OpenAI's GPT-3 for English -> shell-script translation.  
See 2:43
- Realistic face generation non-existent people.

# Why the renaissance?

# Why the renaissance?

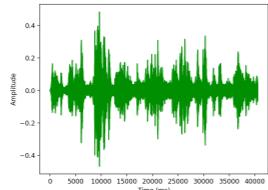
1. More data
  - Cheap sensors (e.g., cameras)
  - Social media
2. Faster hardware
  - GPUs
  - Cloud computing
3. Improved algorithms
  - Capture symmetries in the input
  - Faster optimization

# Domains

- Images



- Audios



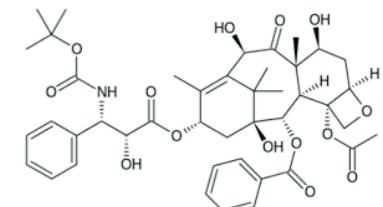
- Videos



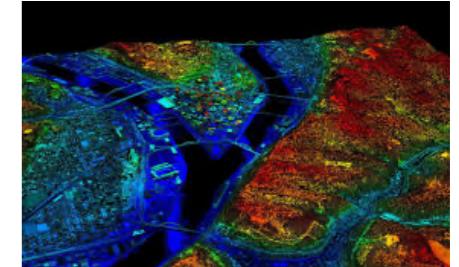
- Sequences (e.g., text, market data)

There once was a person from Nantucket...

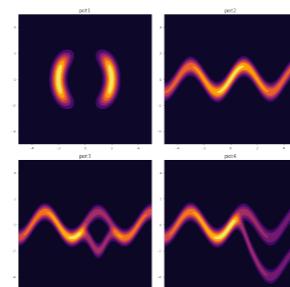
- Graphs (e.g., social networks, molecules)



- Sets (e.g., point-cloud of LiDAR measurements)

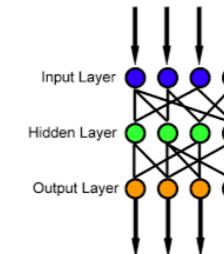


- Probability distributions

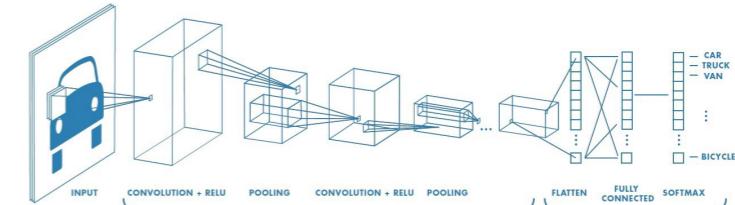


# Models and algorithms

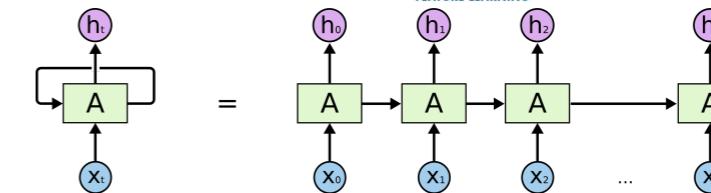
- Feed-forward neural networks (FFNN/DNN)



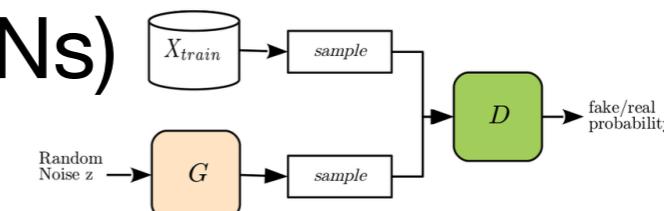
- Convolutional neural networks (CNN)



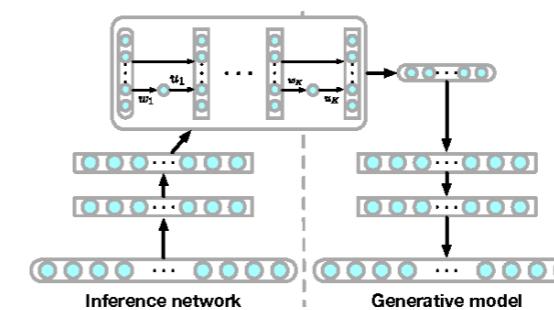
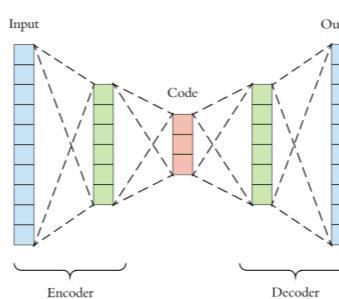
- Recurrent neural networks (RNN)



- Generative-adversarial networks (GANs)



- Auto-encoders (AEs), variational AEs (VAEs), and normalizing flows (NF)



# Training regimes

- Supervised learning
- Unsupervised learning
- Reinforcement learning
- Embeddings, 1-shot &  $k$ -shot learning
- Adversarial training
- Density estimation
- Generative models

# Techniques

- Weight initialization and symmetry-breaking [Orthogonal weight initialization, Dropout](#)
  - How to initialize the NN weights and conduct training to reach a good local minimum?
- Optimization methods [Momentum, RMSProp, AdaGrad, Adam](#)
  - Exploit higher-order structure of optimization landscape
- Feature normalization [BatchNorm, LayerNorm, GroupNorm](#)
  - Make optimization landscape smoother and more uniform
- Weight sharing [Convolution, recurrence](#)
  - Exploit symmetries to reduce the number of parameters
- Variational approximation [VAEs, NFs](#)
  - Estimate an intractable probability distribution using a function approximator with tractable gradient updates.

# Techniques

- Bottlenecks [AE bottlenecks, Inception layer, 1-D convolution](#)
  - Compress data to retain most important information
- Gradient propagation [Activation functions \(ReLU, ELU\), residual connections](#)
  - Ensure gradient neither vanishes nor explodes
- Transfer learning and domain adaptation [Pre-training & fine-tuning, adversarial domain adaptation](#)
  - Exploit knowledge from a different dataset/problem
- Policy gradient & log-likelihood methods [REINFORCE](#)
  - Handle non-differentiable intermediate values.
- Attention [Soft, hard, multi-head](#)
  - Which inputs or activations are most important?

# **Evolution of ideas**

- Good ideas quickly spread from one area to another, e.g.:

**Images**

**Graphs**

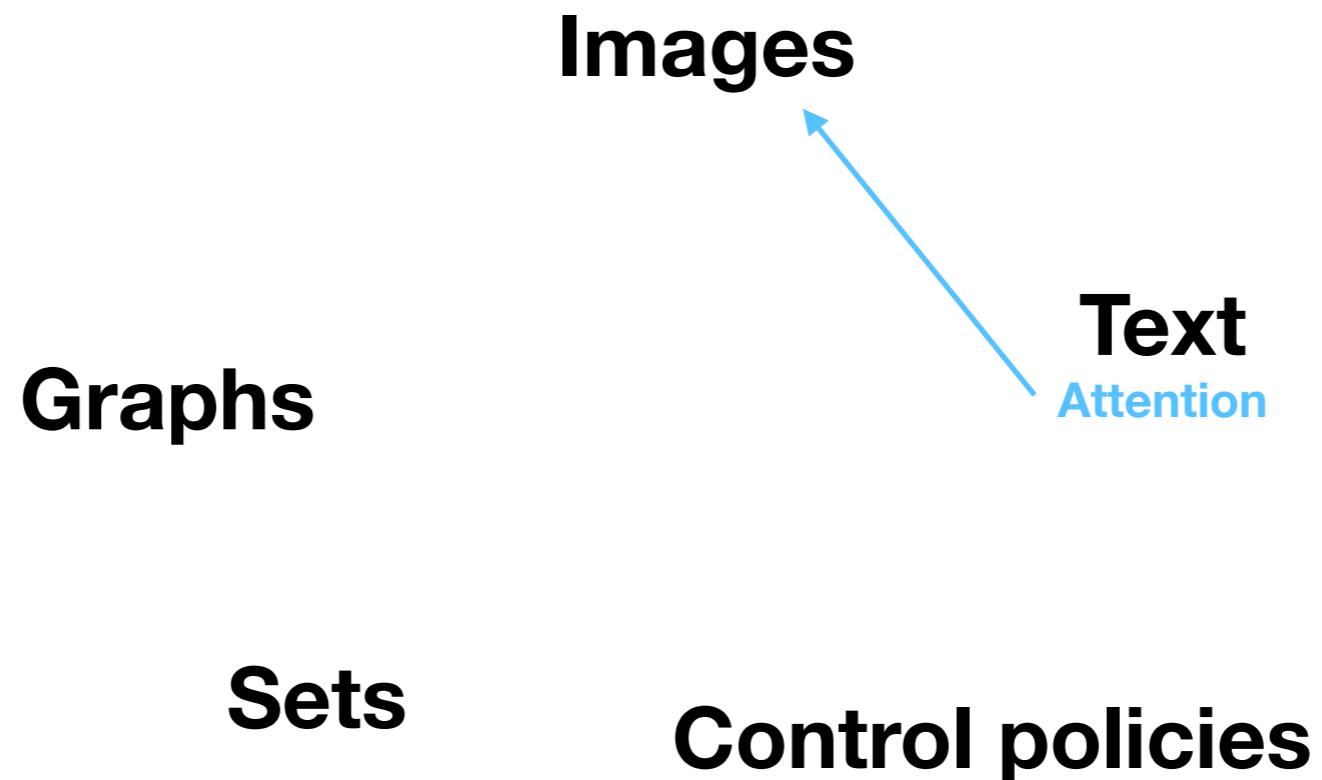
**Text**

**Sets**

**Control policies**

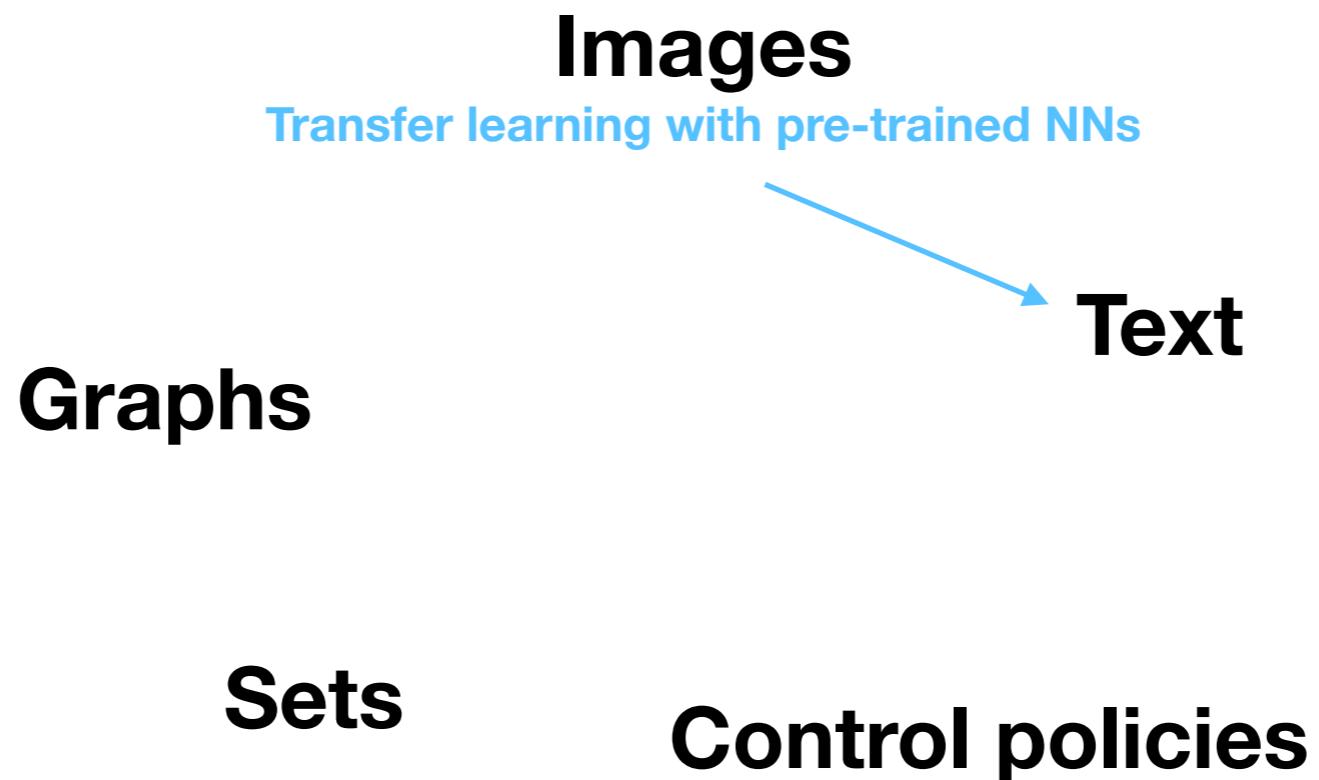
# Evolution of ideas

- Good ideas quickly spread from one area to another, e.g.:
  - Neural attention models



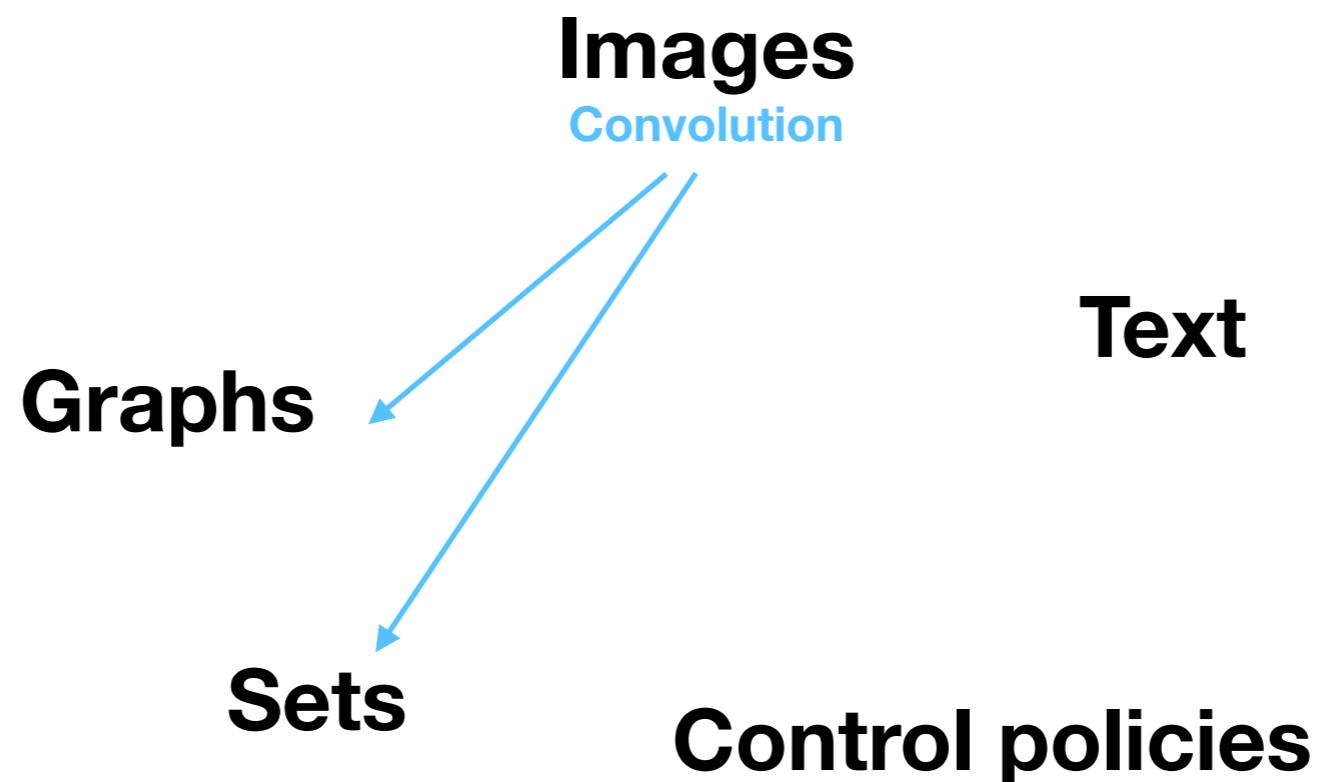
# Evolution of ideas

- Good ideas quickly spread from one area to another, e.g.:
  - Transfer learning techniques



# Evolution of ideas

- Good ideas quickly spread from one area to another, e.g.:
  - Convolution



# Evolution of ideas

- Good ideas quickly spread from one area to another, e.g.:
  - Policy gradient techniques to handle non-differential intermediate values (e.g., samples).

Images

Graphs

Sets

Text

REINFORCE

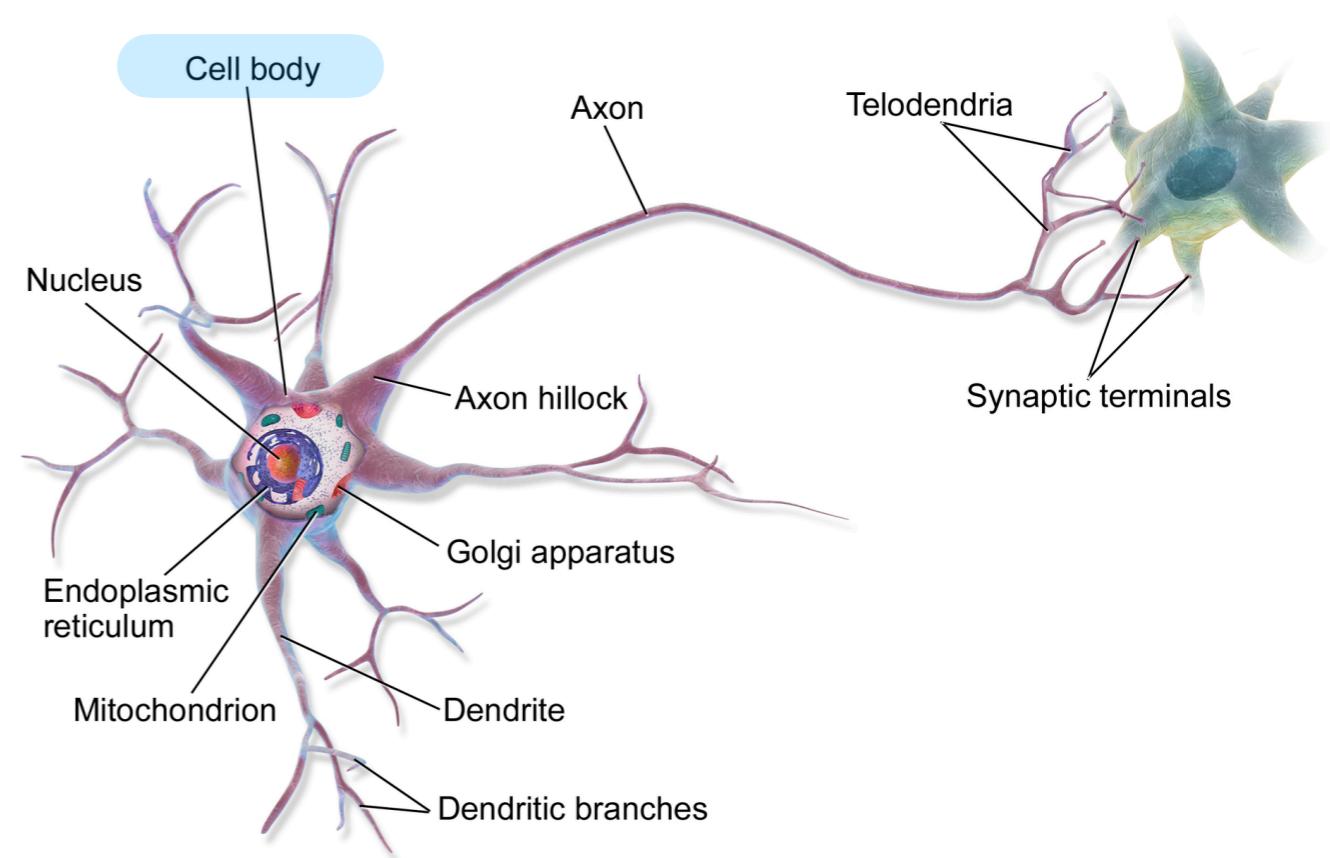
Control policies

# Artificial neural networks and Neuroscience

- Since the early 20th century, scientists have explored how the human/animal intelligence can be distilled into a machine.
- Humans+animals are especially good at seeing and moving in the world.
- What does neuroscience tell us about how to build AI?

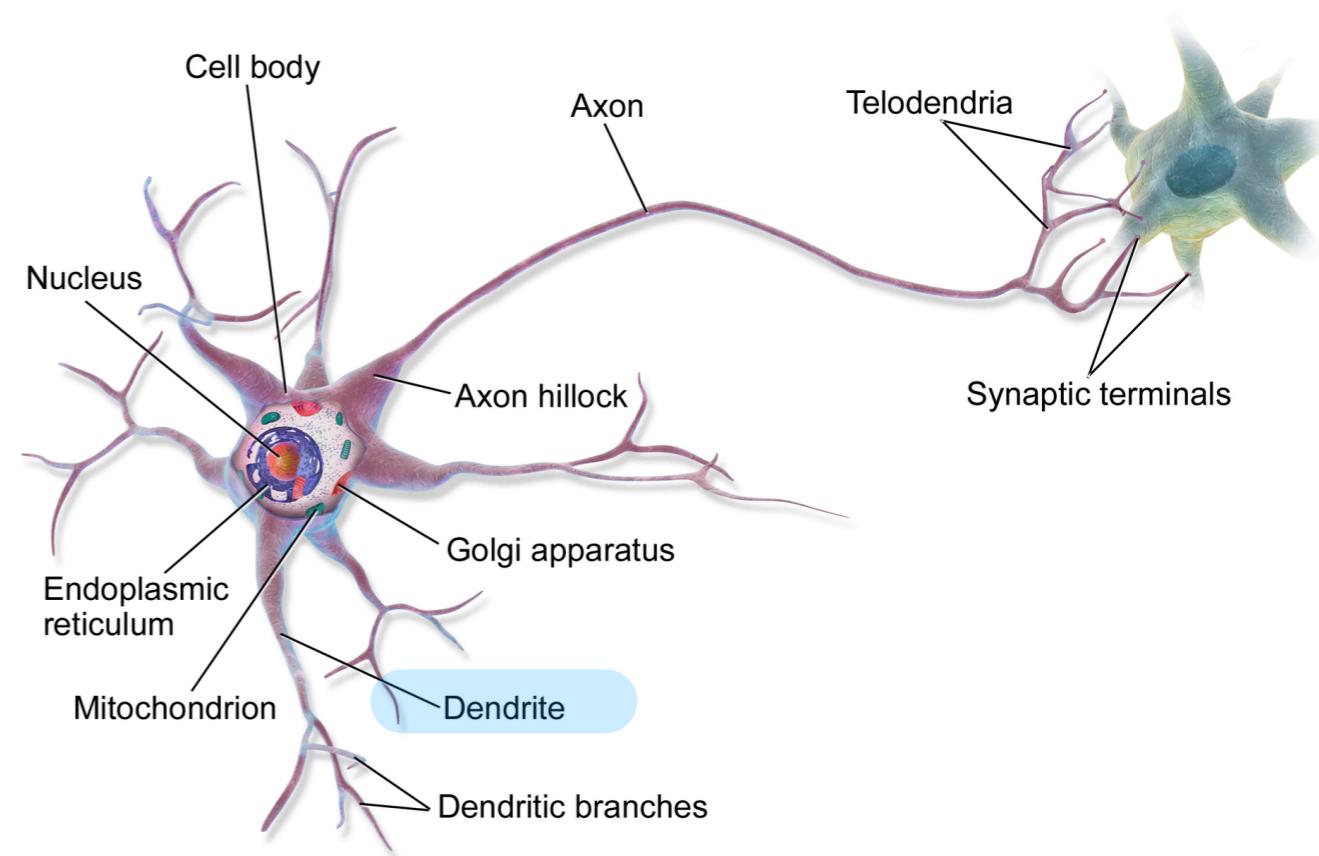
# Biological neural networks

- In biological systems, neural networks consist of a network of connected **neurons**, i.e., brain cells.
- Neurons consist of several components:
  - Soma (cell body)



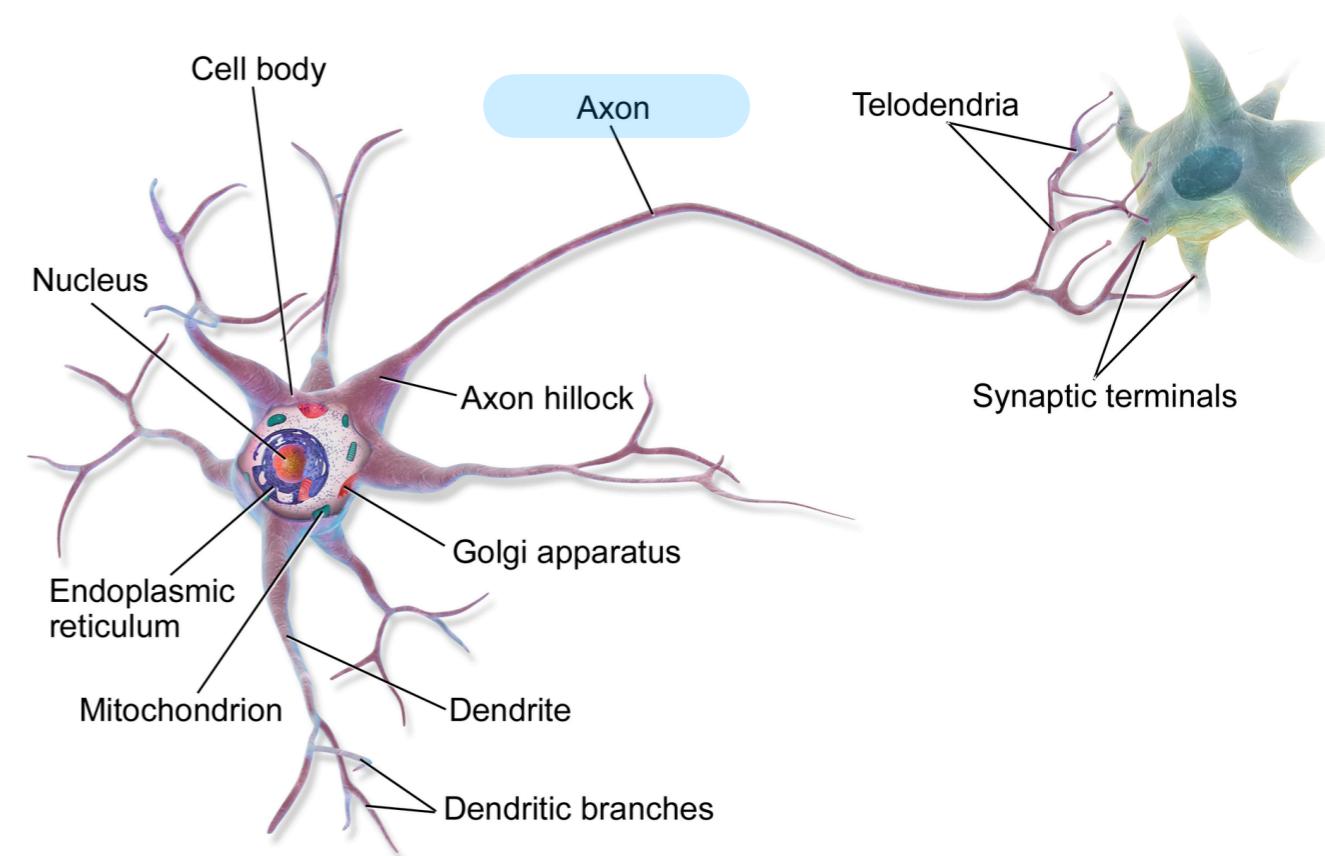
# Biological neural networks

- In biological systems, neural networks consist of a network of connected **neurons**, i.e., brain cells.
- Neurons consist of several components:
  - Soma (cell body)
  - Dendrites (receptors from other neurons)



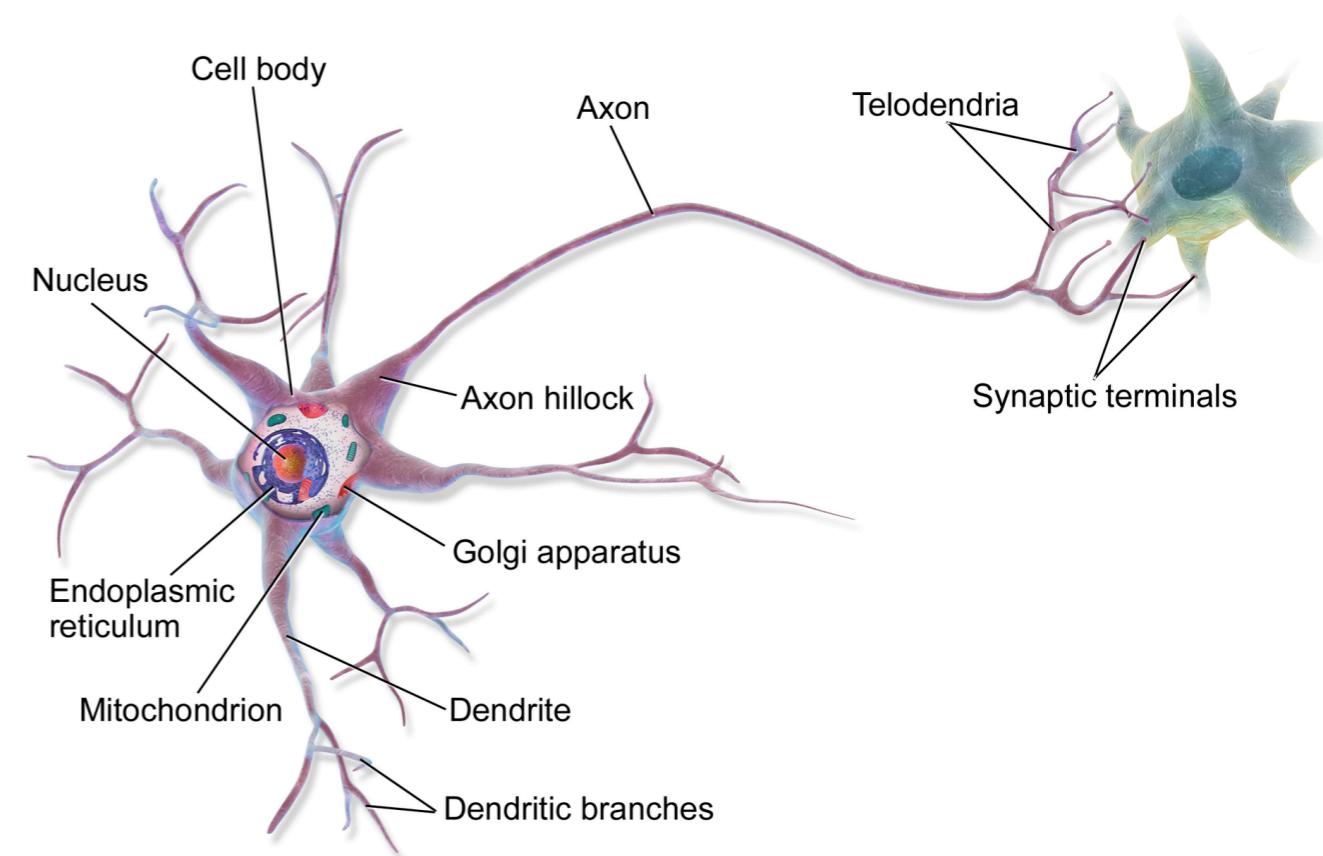
# Biological neural networks

- In biological systems, neural networks consist of a network of connected **neurons**, i.e., brain cells.
- Neurons consist of several components:
  - Soma (cell body)
  - Dendrites (receptors from other neurons)
  - Axons (transmitters to other neurons)



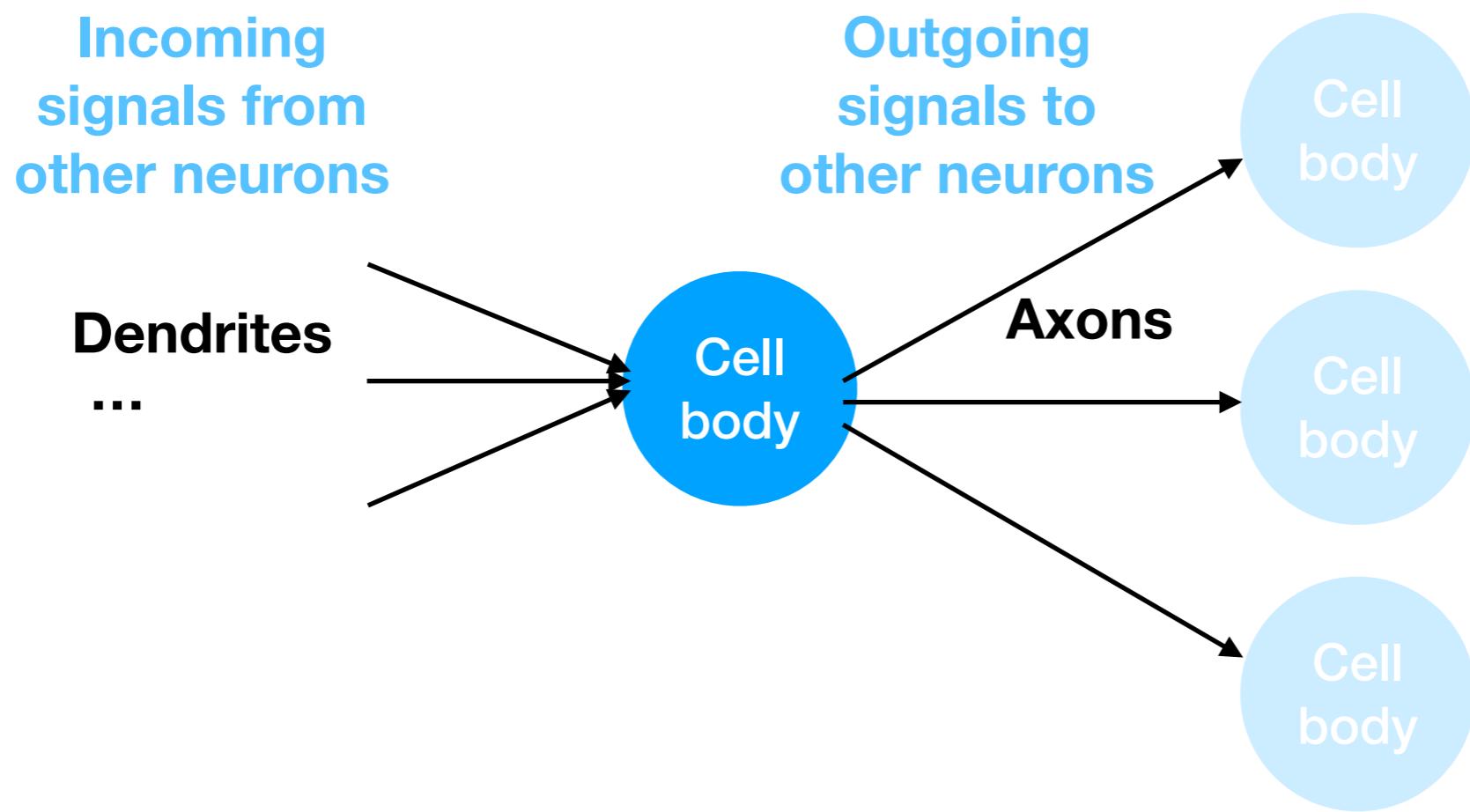
# Biological neural networks

- Neurons can transmit electrical potentials to other neurons via chemical neurotransmitters.
- When the voltage change within one neuron exceeds some threshold, an **all-or-none** electrical potential is transmitted along the axon to neighboring neurons.



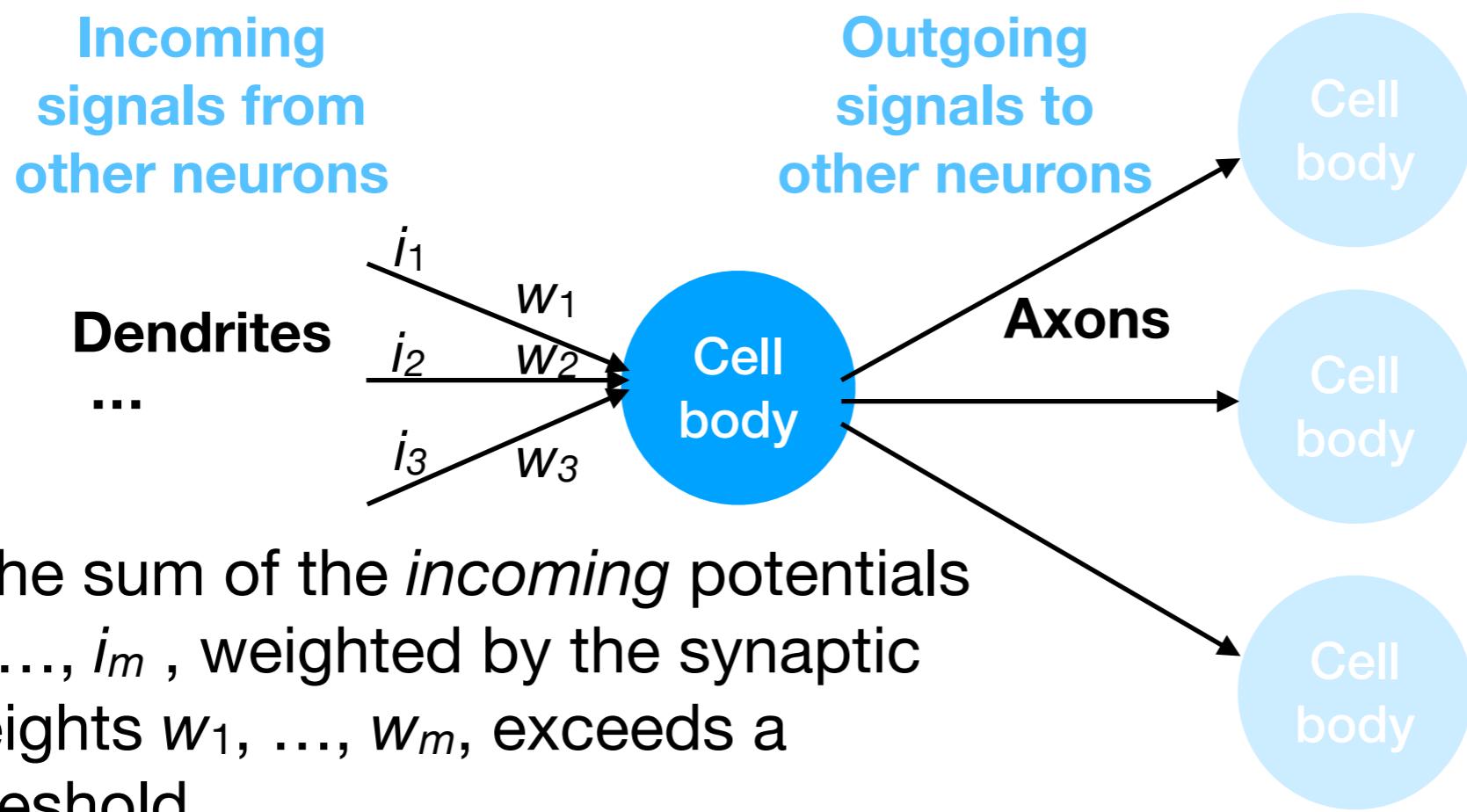
# Artificial neural networks

- We can model this process using an artificial neural network:



# Artificial neural networks

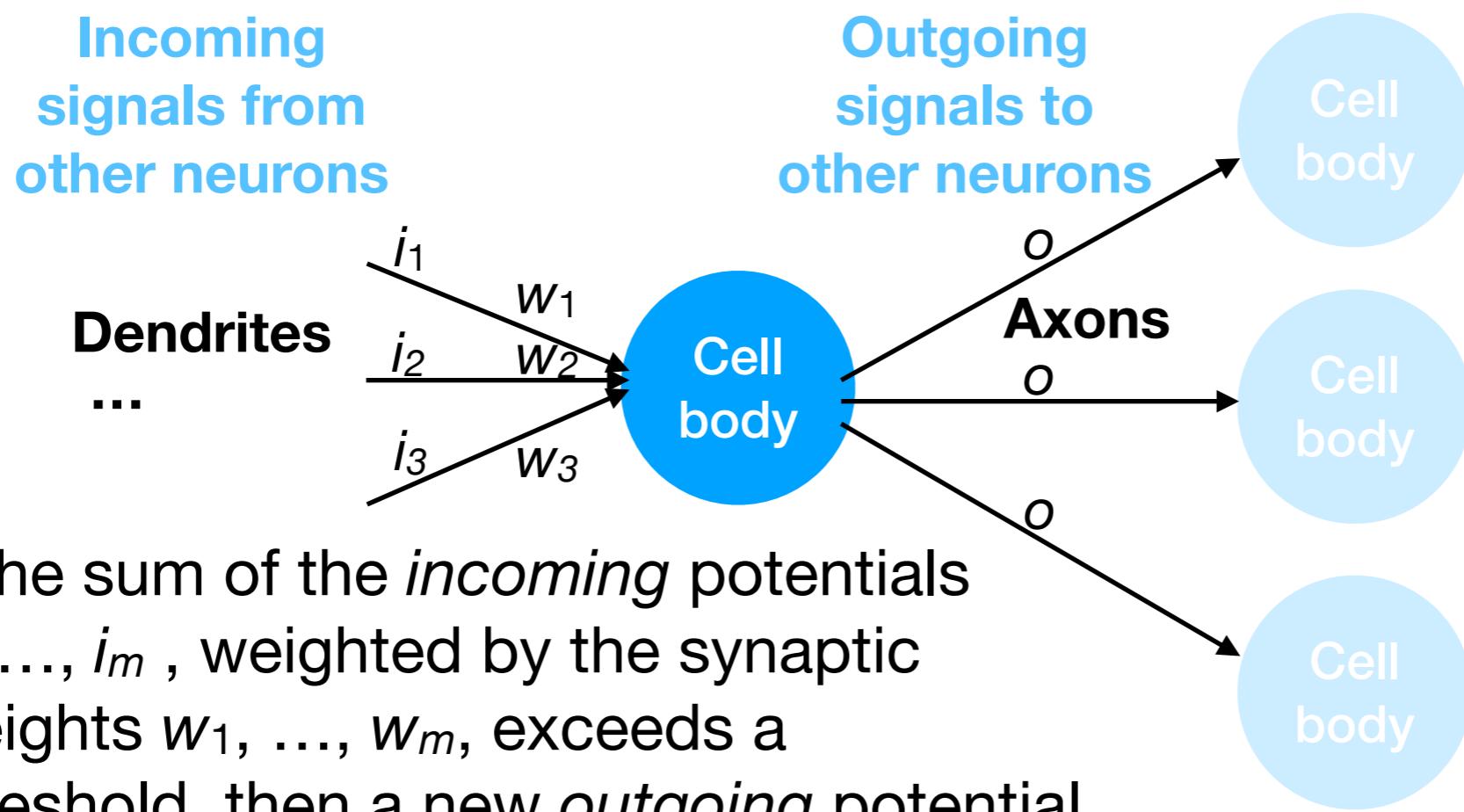
- We can model this process using an artificial neural network:



- If the sum of the *incoming* potentials  $i_1, \dots, i_m$ , weighted by the synaptic weights  $w_1, \dots, w_m$ , exceeds a threshold

# Artificial neural networks

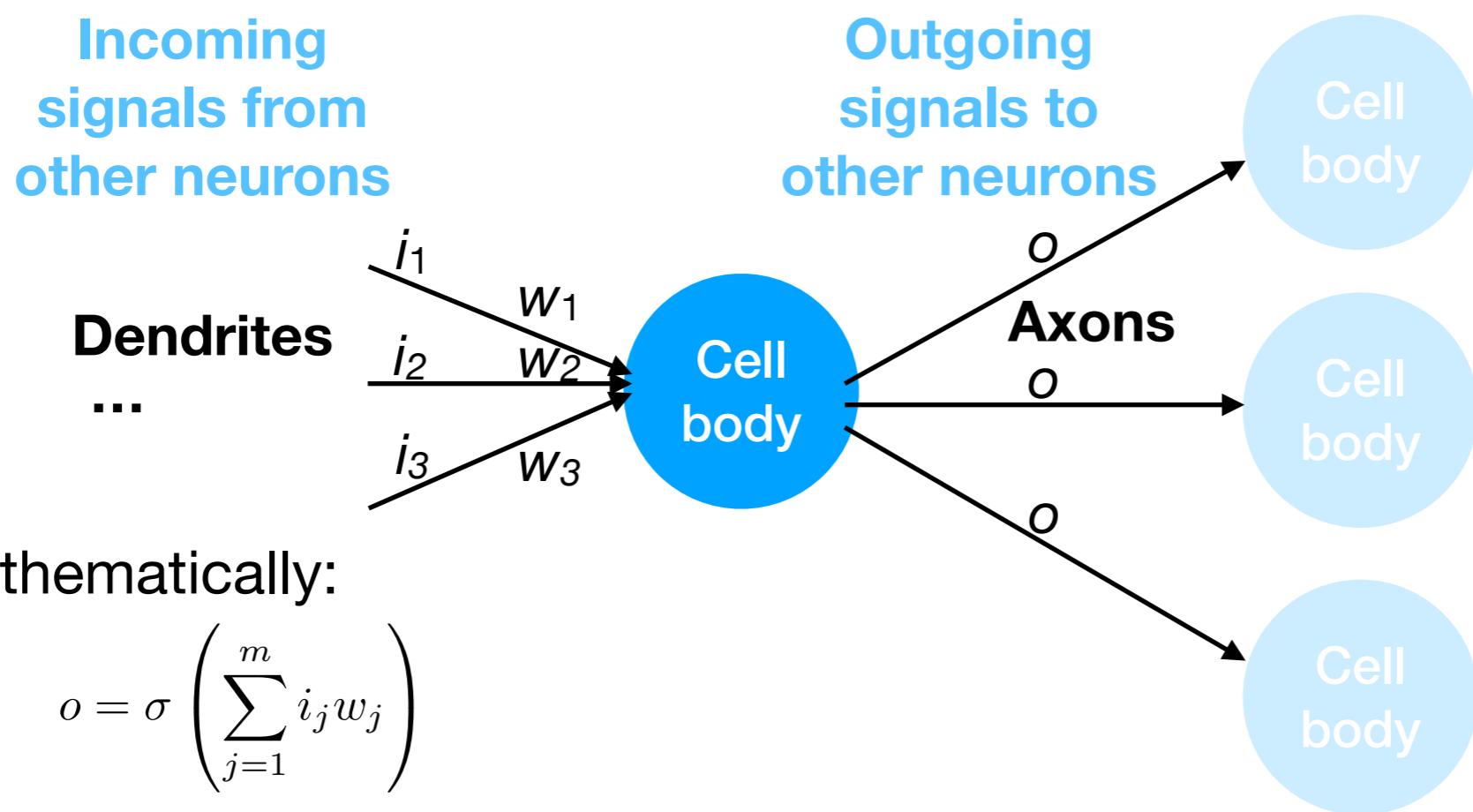
- We can model this process using an artificial neural network:



- If the sum of the *incoming* potentials  $i_1, \dots, i_m$ , weighted by the synaptic weights  $w_1, \dots, w_m$ , exceeds a threshold, then a new *outgoing* potential  $o$  is transmitted along the axons.

# Artificial neural networks

- We can model this process using an artificial neural network:



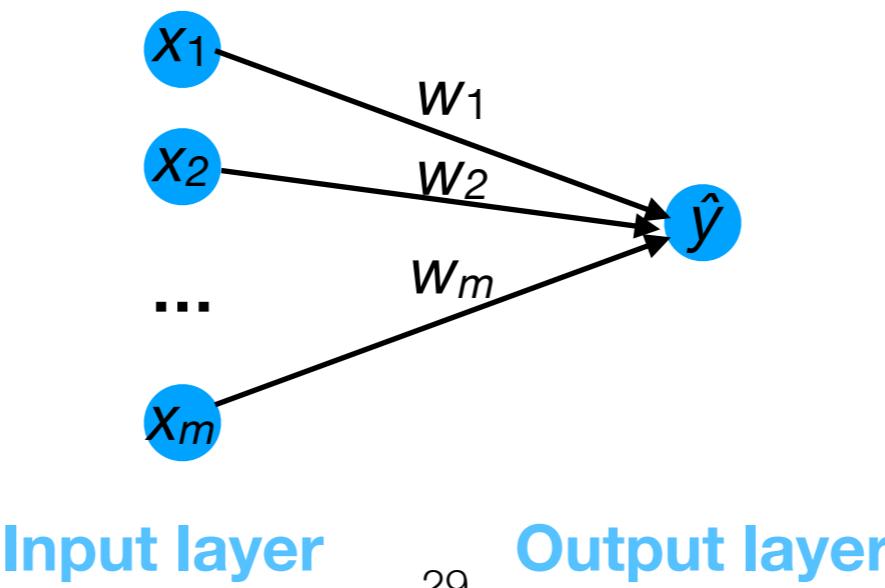
- Mathematically:

$$o = \sigma \left( \sum_{j=1}^m i_j w_j \right)$$

where  $\sigma$  is (usually) some non-linear  
**activation function**.

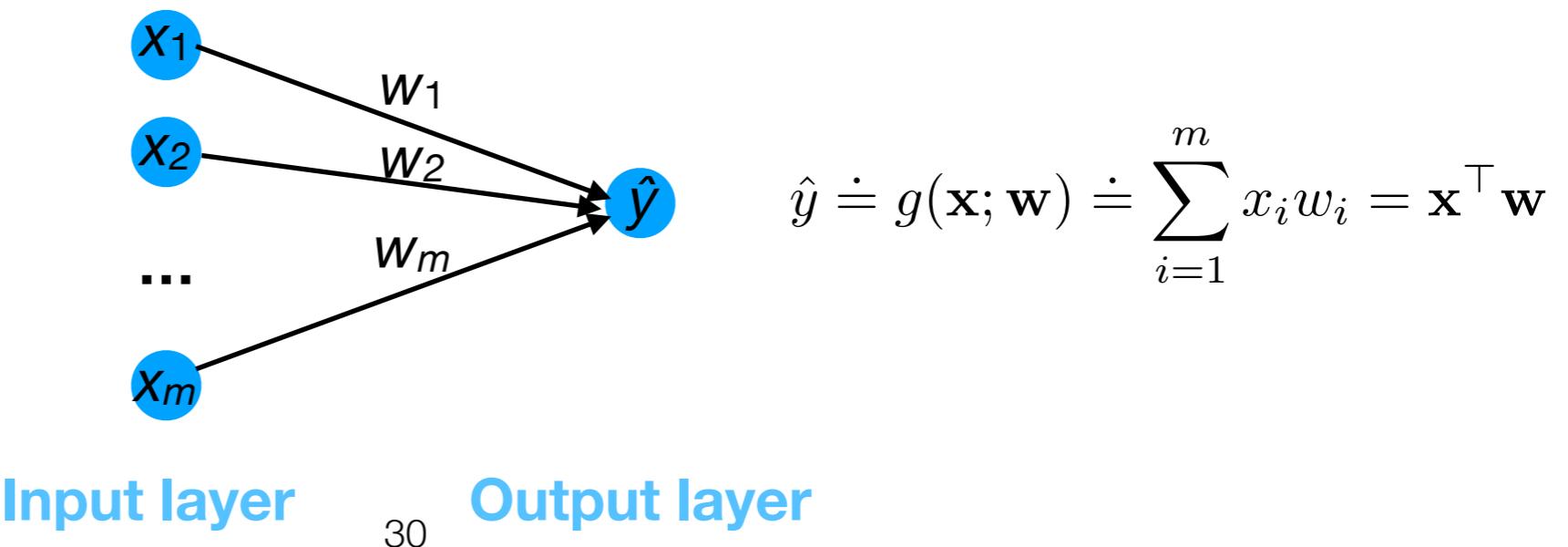
# NNs as a mathematical function

- If we designate a subset of neurons as the input and a subset of neurons as the output, then neural networks can define mathematical functions, i.e.,  $\hat{y} = f(\mathbf{x})$ .
- One common network design is a **feed-forward** (FF) network, consisting of multiple layers of neurons, each of which feeds to the next layer.
- Here is a simple example, which is equivalent to linear regression:



# NNs as a mathematical function

- The input layer  $\mathbf{x}$  directly transmits the values  $x_1, \dots, x_m$  to the next layer.
- The output layer  $\hat{y}$  computes the sum of the inputs multiplied by the synaptic weights  $\mathbf{w}$ .
- In this network, no activation function was used to transform the weighted sum of incoming potentials to  $\hat{y}$ .

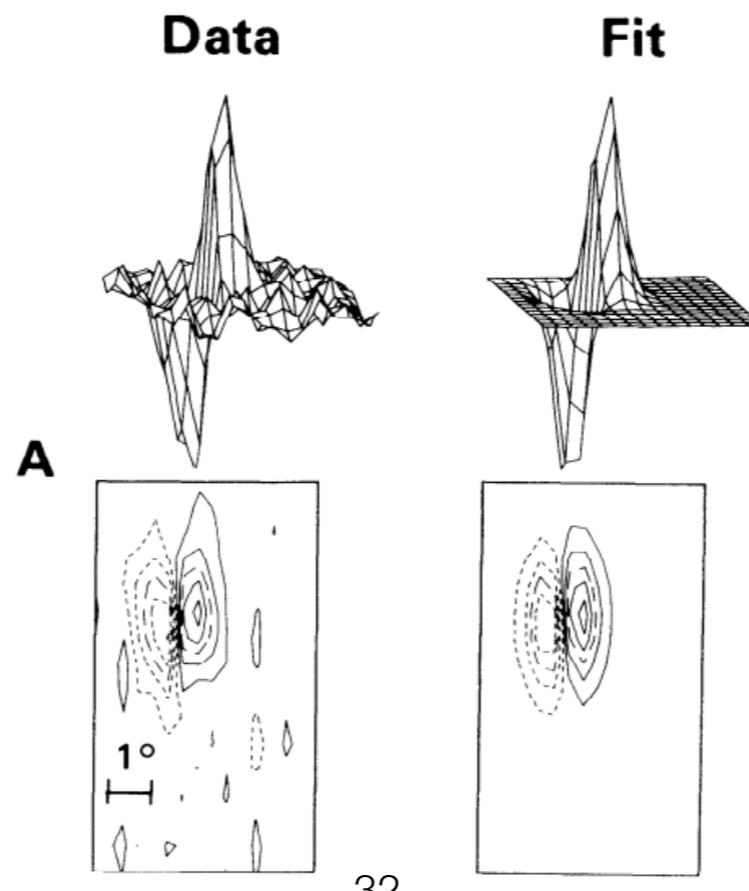


# Artificial neural networks and Neuroscience

- NN and DL have benefited from synergy between:
  - **Neuroscience**: develop computational models of neurons to understand the brain.
  - **Computer science**: use intuition of how human brain works to develop better artificial neural networks.

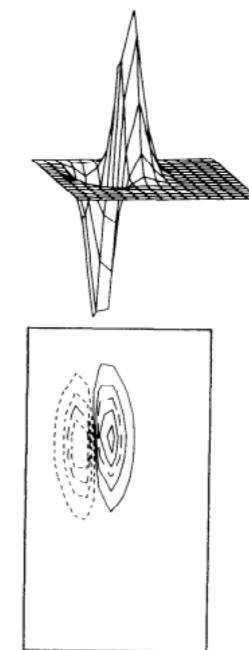
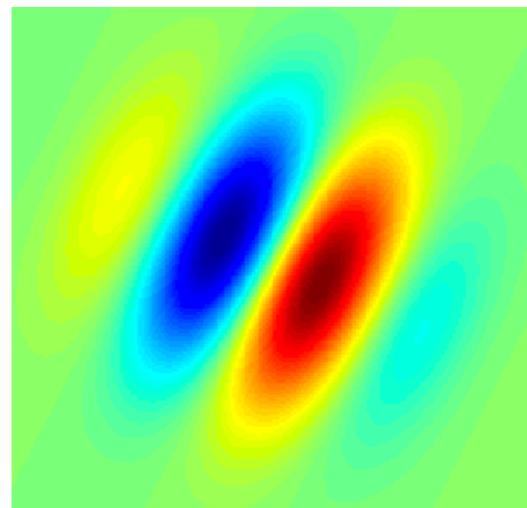
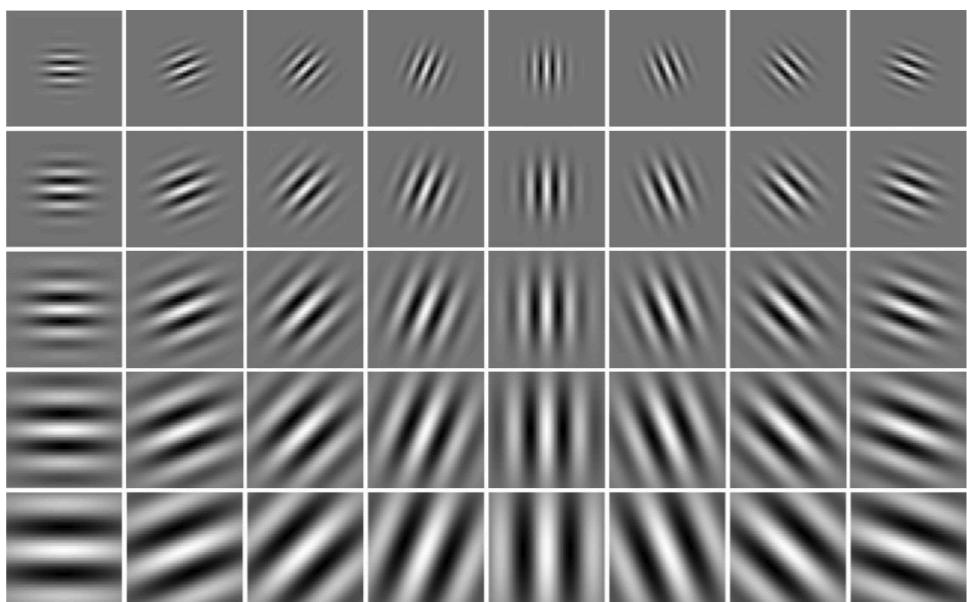
# Artificial neural networks and Neuroscience

- Example (Jones & Palmer 1987): identifying the impulse-response (IR) of cats' visual cortex (V1 layer).
- Stimulate V1 neurons with different light patterns, and measure the electrical response.



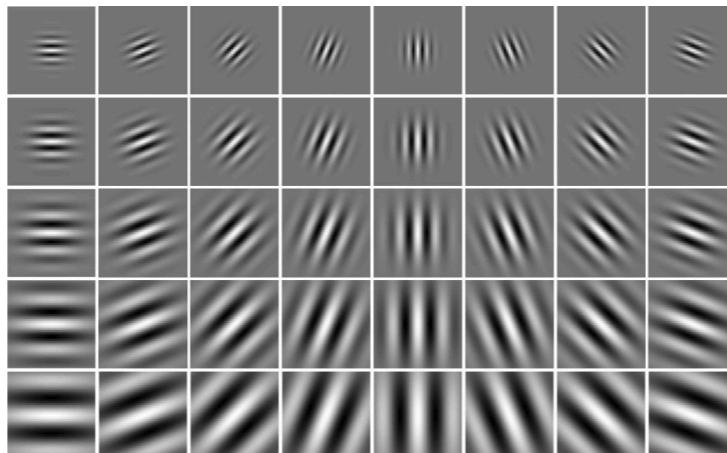
# Artificial neural networks and Neuroscience

- It turns out that the IR can be modeled using 2-D Gabor filters.
- Gabor filters are **convolution kernels** that respond preferentially to 2-D line patterns of different **orientations** and **frequencies**.



# Artificial neural networks and Neuroscience

- These convolution kernels are often found (and trained implicitly) in modern convolutional neural networks (CNNs):



**Gabor kernels**



**CNN kernels**

# What is “deep”?

- Much of DL (and ML in general) can be formulated as computing a function  $f$  — which defines the behavior of a **machine** — that transforms input  $\mathbf{x}$  into desired output  $\hat{y}$ , e.g.:
  - $\mathbf{x}$  is the chemical structure of a drug;  $\hat{y}$  is its efficacy in treating cancer.
  - $\mathbf{x}$  is an image of a natural scene;  $\hat{y}$  is the set of objects (e.g., mountain, tree) shown in the scene.
  - $\mathbf{x}$  is a LiDAR point cloud;  $\hat{y}$  is the vector of control inputs to the car.
  - $\mathbf{x}$  is a random number from some fixed probability distribution;  $\hat{y}$  is an image generated from  $\mathbf{x}$ .

# What is “deep”?

- In “classical” ML (e.g., SVMs, boosting, decision trees),  $f$  is often a “shallow” function of  $\mathbf{x}$ , e.g.:

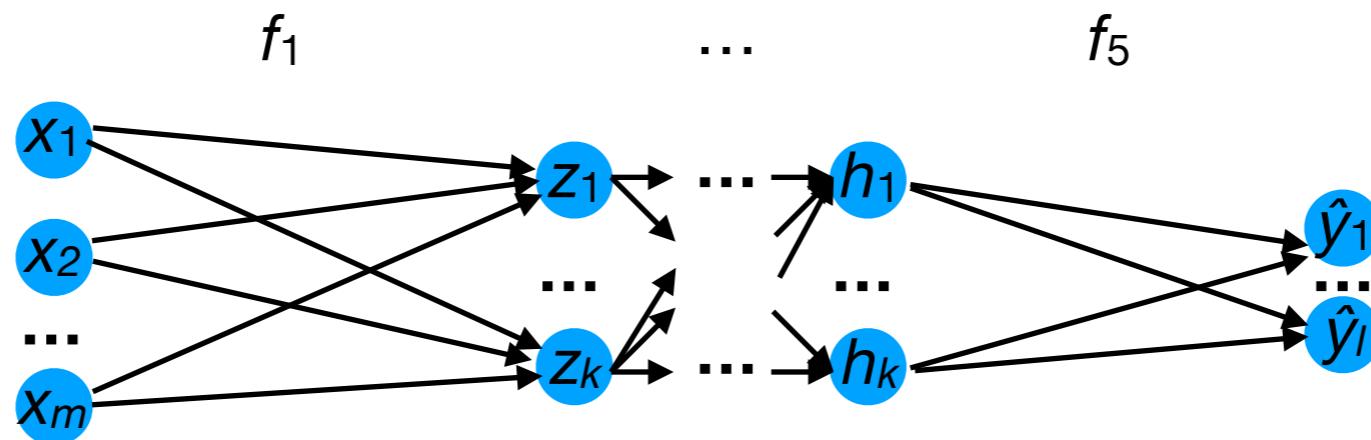
$$\hat{y} = f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$$

- In contrast, with DL,  $f$  is the **composition** (possibly 1000s of “layers” deep!) of many functions, e.g.:

$$f(\mathbf{x}) = f_5(f_4(f_3(f_2(f_1(\mathbf{x})))))$$

# What is “deep”?

- Architecturally, this corresponds to an artificial neural network with many layers:



$$f(\mathbf{x}) = f_5(f_4(f_3(f_2(f_1(\mathbf{x})))))$$

# Applications

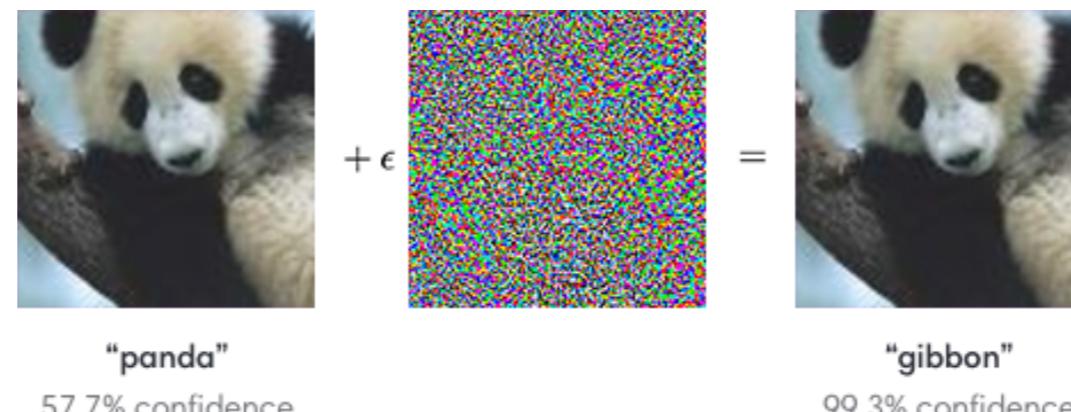
- Object recognition (Krizhevsky, et al. 2012)
- Deep reinforcement learning for computer games (Mnih, et al. 2013)
- Stock price prediction
- Image/video recognition and transcription (Donahue, et al. 2015)
- Gmail smart reply (Kannan, et al. 2016)
- Pathology and medical diagnosis (Esteva, et al. 2017)

# Final projects from previous years' course offerings

- “Shallow Depth-of-field Effect using Semantic Segmentation” (Dsouza, Havannavar, and Inamdar 2018)
- “High Fidelity Handbag Generator” (Xiong, Ni, Dai, and Hu 2019)
- “Visual Question Answering” (Ramakrishnan, Agrawal, Shrotri, Phadnis 2020).

# Mistakes

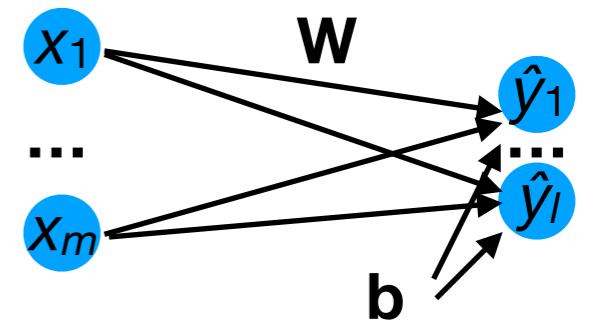
- While DL has achieved wonders, it has also achieved blunders, e.g.:



- One recurrent theme in ML is that machines often believe with high confidence that its erroneous output is correct.
- They can also make huge errors on subgroups that were not well-represented in the training data.
- To avoid and solve these problems, it is essential to understand the mathematics of how ML algorithms work.

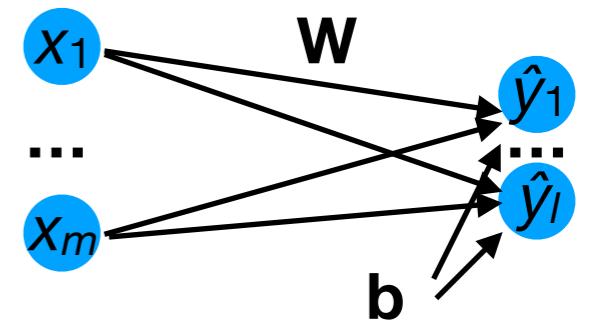
# Course outline

- Part I: Shallow neural networks & optimization methods
  - Mathematical fundamentals:
    - Linear algebra
    - Probability theory
  - Shallow architectures:
    - Linear regression
    - Logistic & softmax regression
  - Latent variable models:
    - Principal component analysis (PCA)



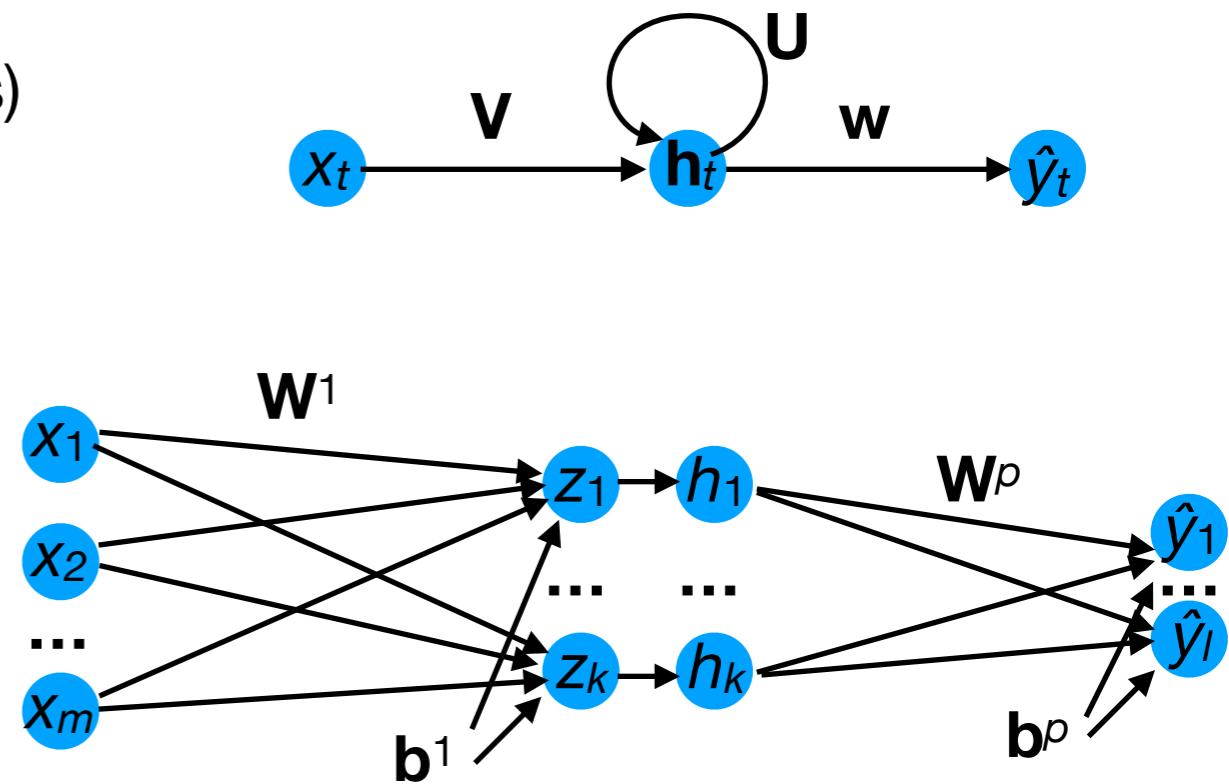
# Course outline

- Part I: Shallow neural networks & optimization methods
  - First-order optimization methods:
    - Gradient descent
    - Stochastic gradient descent (SGD)
  - Second-order optimization methods:
    - Newton's method
  - Data pre-processing:
    - Whitening transformations



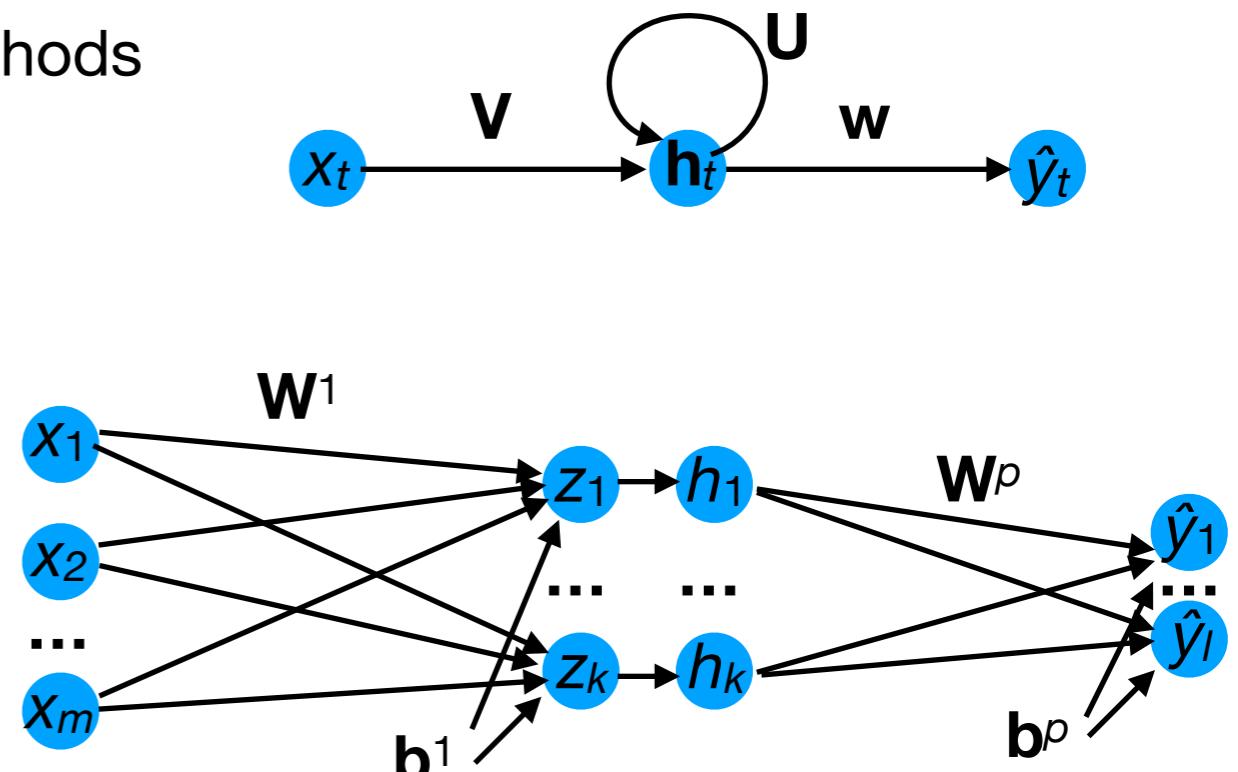
# Course outline

- Part II: Deep neural networks & optimization methods
  - Feed-forward architectures:
    - Multi-layer perceptrons (MLPs)
    - Auto-encoders
    - Convolutional neural networks (CNNs)
  - Recurrent architectures:
    - Recurrent neural networks (RNNs)
  - Generative architectures:
    - Variational auto-encoders (VAEs)
    - Generative-adversarial networks (GANs)
  - Geometric deep learning:
    - Graph convolutional NNs



# Course outline

- Part II: Deep neural networks & optimization methods
  - Optimization techniques:
    - Activation functions
    - Batch/layer/group normalization
    - Weight initialization
    - Second-order approximation methods
  - Regularization techniques:
    - Weight decay
    - Dropout
    - Data augmentation
    - Pre-training
    - Weight sharing



# Course outline

- Part III: State-of-the-art deep learning & applications
  - Recent papers from:
    - NeurIPS
    - ICML
    - ICLR
    - AAAI
    - CVPR
    - Arxiv

# Course structure

- High-level course goals:
  - Learn the mathematical basis of important deep learning models and algorithms.
  - Acquire practical proficiency in designing and training deep neural networks.
- Grading:
  - Homework: 40%
  - Final project: 40%
  - Exam: 15%
  - Class participation (paper presentation, speed dating): 5%

# **Notation and definitions**

# Notation

- Scalars are non-bold and *italic*:  $y$ ,  $\hat{y}$
- Vectors are lower-case and **bold**:  $\mathbf{z}$ ,  $\mathbf{w}$
- Matrices are UPPER-CASE and **bold**:  $\mathbf{X}$
- ${}^T$  indicates matrix transpose:  $\mathbf{X}^T$
- (Parenthesized) superscripts specify the example:  $\mathbf{x}^{(3)}$

# Notation

- Vectors are column vectors unless noted otherwise.
- $n$  refers to number of examples;  $i$  is the index variable.
- $m$  refers to number of features;  $j$  is the index variable.

# Multiplication

- Multiplication of matrices **A** and **B**:
  - Only possible when: **A** is  $(n \times k)$  and **B** is  $(k \times m)$
  - Result:  $(n \times m)$
- The **inner product** between two column vectors (same length) **x**, **y** can be written as:  $\mathbf{x}^T\mathbf{y}$ . The result is a scalar.
- The **outer product** between two column vectors (same length) **x**, **y** can be written as:  $\mathbf{x}\mathbf{y}^T$ . The result is a matrix.
- The **Hadamard** (element-wise) **product** between two matrices **A** and **B** is written as  $\mathbf{A} \odot \mathbf{B}$ .

# Derivatives

- The **gradient** of a function  $f : \mathbb{R}^m \rightarrow \mathbb{R}$  w.r.t. input  $\mathbf{x}$  is the column vector of first partial derivatives:

$$\nabla_{\mathbf{x}} f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix}$$

# Derivatives

- The **Jacobian** of a function  $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$  w.r.t. input  $\mathbf{x}$  is the matrix of first partial derivatives:

$$\text{Jac}[f] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \cdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

- Note that, for  $n=1$ ,

$$\nabla_{\mathbf{x}} f = \text{Jac}[f]^{\top}$$

# Derivatives

- The **Hessian** of a function  $f : \mathbb{R}^m \rightarrow \mathbb{R}$  w.r.t. input  $\mathbf{x}$  is the matrix of second partial derivatives:

$$H[f] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \cdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_m \partial x_m} \end{bmatrix}$$

- Note that

$$H[f] = \text{Jac}[\nabla_{\mathbf{x}} f]$$

# Exercises

- Compute the gradient and Hessian of  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ , where

$$f(\mathbf{x}) = 2x_1x_2 - x_3 \log x_1 - \frac{1}{x_3}$$

$$\mathbf{H}[f] = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \cdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_m \partial x_m} \end{bmatrix}$$

$$\nabla_{\mathbf{x}} f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_m} \end{bmatrix}$$

# Exercises

- Compute the gradient and Hessian of  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ , where

$$f(\mathbf{x}) = 2x_1x_2 - x_3 \log x_1 - \frac{1}{x_3}$$

- Solution:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} 2x_2 - \frac{x_3}{x_1} \\ 2x_1 \\ -\log x_1 + \frac{1}{x_3^2} \end{bmatrix}$$

$$H[f](\mathbf{x}) = \begin{bmatrix} \frac{x_3}{x_1^2} & 2 & -\frac{1}{x_1} \\ 2 & 0 & 0 \\ -\frac{1}{x_1} & 0 & -\frac{2}{x_3^3} \end{bmatrix}$$

# **Linear regression (aka 2-layer NN)**

# Linear regression

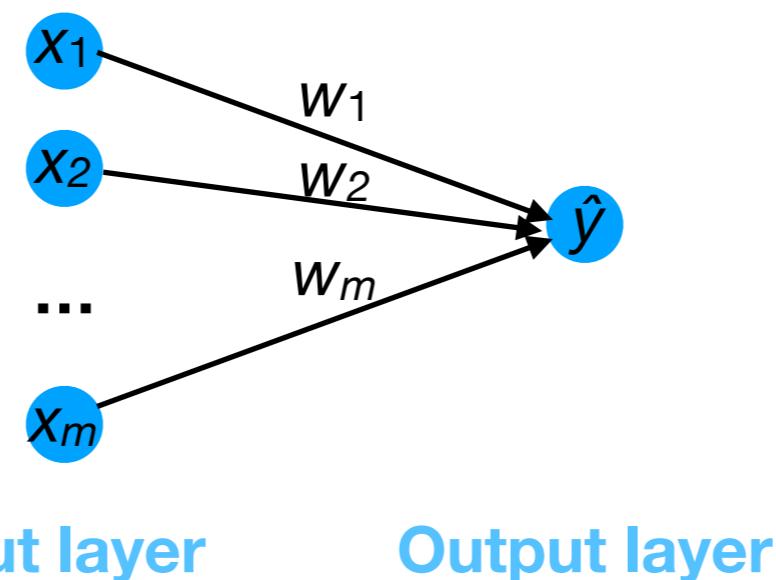
- Let dataset  $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$
- Want to create a neural network — which we can also treat as a “machine” — to estimate each  $y^{(i)}$  with high accuracy.
- Let us define the machine by a function  $g$  (with parameters  $\mathbf{w}$ ) whose output  $\hat{y}$  is linear in its inputs:

$$\hat{y} \doteq g(\mathbf{x}; \mathbf{w}) \doteq \sum_{j=1}^m x_j w_j = \mathbf{x}^\top \mathbf{w}$$

# Linear regression

- Note that this function is equivalent to a 2-layer neural network (with no activation function):

$$\hat{y} \doteq g(\mathbf{x}; \mathbf{w}) \doteq \sum_{j=1}^m x_j w_j = \mathbf{x}^\top \mathbf{w}$$



# Linear regression

- Example application: automated face analysis to estimate a person's age from their face image.
- Here, we treat the image  $\mathbf{x}$  as a *vector* (even though it represents a 2-d image).

