Neural networks for classification and regression

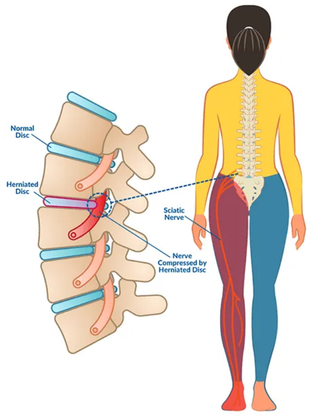
Outline

- Hour 1
  - Review, examples of ML in IGM
  - Neural networks introduction
- Hour 2: Neural network training

Last week: k-NN in python

Note: the only difference between two exercises are the datasets

- **Dataset:** Biomechanic features of orthopaedic patients, and the type of injury
- **Goal:** Classify the condition of patients based on biomechanic features



	pelvic_tilt	degree_spondylolisthesis	class
0	22.552586	-0.254400	Hernia
1	10.060991	4.564259	Hernia
2	22.218482	-3.530317	Hernia
3	24.652878	11.211523	Hernia
4	9.652075	7.918501	Hernia
...	...	...	...
85	9.906072	20.315315	Spondylolisthesis
86	17.879323	22.123869	Spondylolisthesis
87	10.218996	37.364540	Spondylolisthesis
88	16.800200	24.018575	Spondylolisthesis
89	23.896201	27.283985	Spondylolisthesis

- **Dataset:** Pinguin body features and their specie type
- **Goal:** Classify the specie of a new observed pinguin

	species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
0	Chinstrap	49.0	19.5	210.0	3950.0
1	Chinstrap	50.9	19.1	196.0	3550.0
2	Gentoo	42.7	13.7	208.0	3950.0
3	Chinstrap	43.5	18.1	202.0	3400.0
4	Chinstrap	49.8	17.3	198.0	3675.0

Review problem

$N = 200$ dataset

- **Dataset:** Biomechanic features of 200 orthopaedic patients, and the type of injury $x^i \in \mathbb{R}^2$
  - Pelvic tilt (degree), Spondylolisthesis SP (degree), label: normal (1), Hernia (2), Spondylosithesis (3) disk $y^i \in \{1, 2, 3\}$
- **Goal:** given a new patient measurement, classify the disk, $x^{test} \in \mathbb{R}^2$

- **Approach 1: k-NN**

After tuning the distance and k using cross-fold validation, you found 4-NN with Manhattan distance to have an accuracy of 72% on your test set. Write the pseudo-code to classify the disk condition of this new patient

for $i = 1, \dots, N$

$d^i := dist(x^{test}, x^i) = |x^i_1 - x^{test}_1| + |x^i_2 - x^{test}_2|$

find 4 smallest d^i's : i_1, i_2, i_3, i_4 indices

$y^{test} = mode(y^{i_1}, y^{i_2}, y^{i_3}, y^{i_4})$

- **Approach 2: Naive Bayes**

Let's first simplify the problem

  - pelvic tilt: **low/high**, SP: **low/high**

Write the pseudo-code to classify the disk condition for a patient who has **low** pelvic tilt and **low** SP

$P(c \mid x^{test} = [\text{"low"}, \text{"low"}])$ is proportional to

$P(x^{test}_1 = \text{"low"} \mid c) P(x^{test}_2 = \text{"low"} \mid c) P(c)$

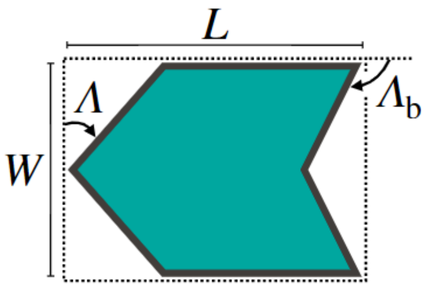
$P(x^{test}_1 = \text{"low"} \mid c) = \frac{|\{i \text{ s.t. } x^i_1 = \text{"low"} \wedge y^i = c\}|}{|\{i \text{ s.t. } y^i = c\}|}$

Example application of ML used in Génie mécanique

Prof. Josie Hughes: Computational robot design and fabrication lab

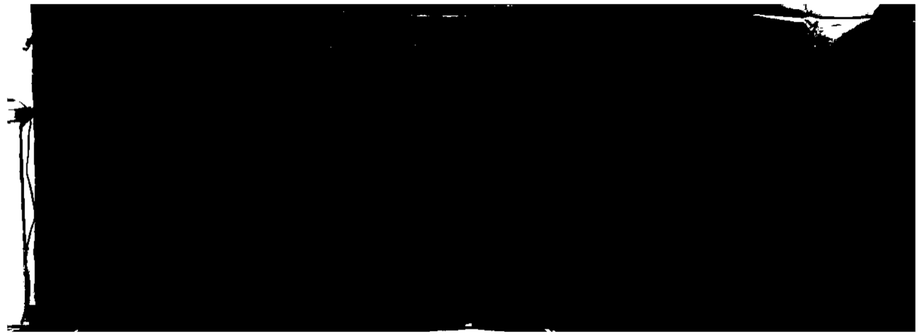
Automated fabrication & optimization of paper planes

Optimization & Exploration

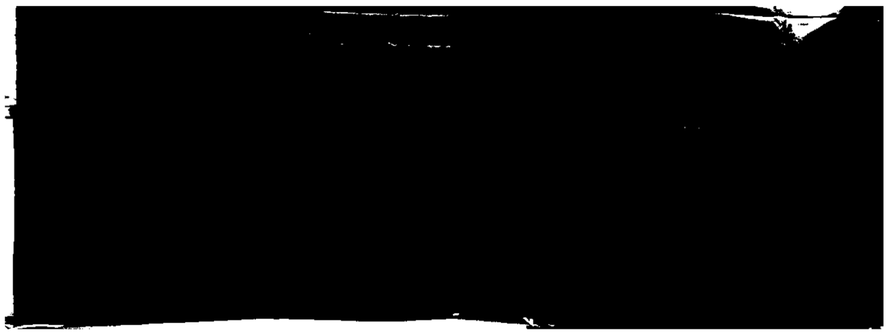


Nose Dive

Mid glide



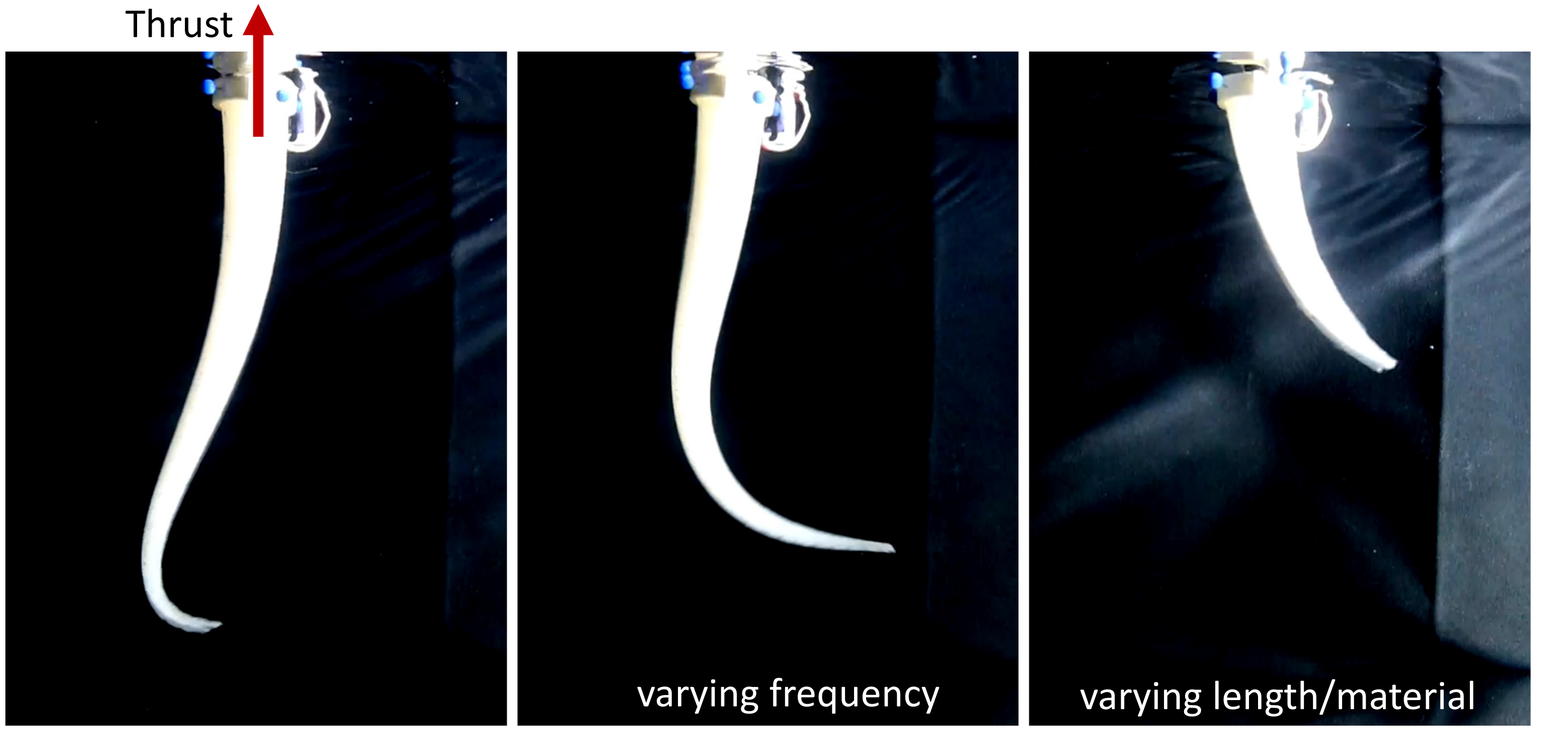
Recovery Glide



- Predict the flight behaviour of a paper plane (and its variance) given the design parameters
- Optimally sample from the design space to build up the model to minimize physical experimentation
- Optimize the design of light weight MAV robotic systems.

Soft Tentacles

Simulation & Modelling for design optimization

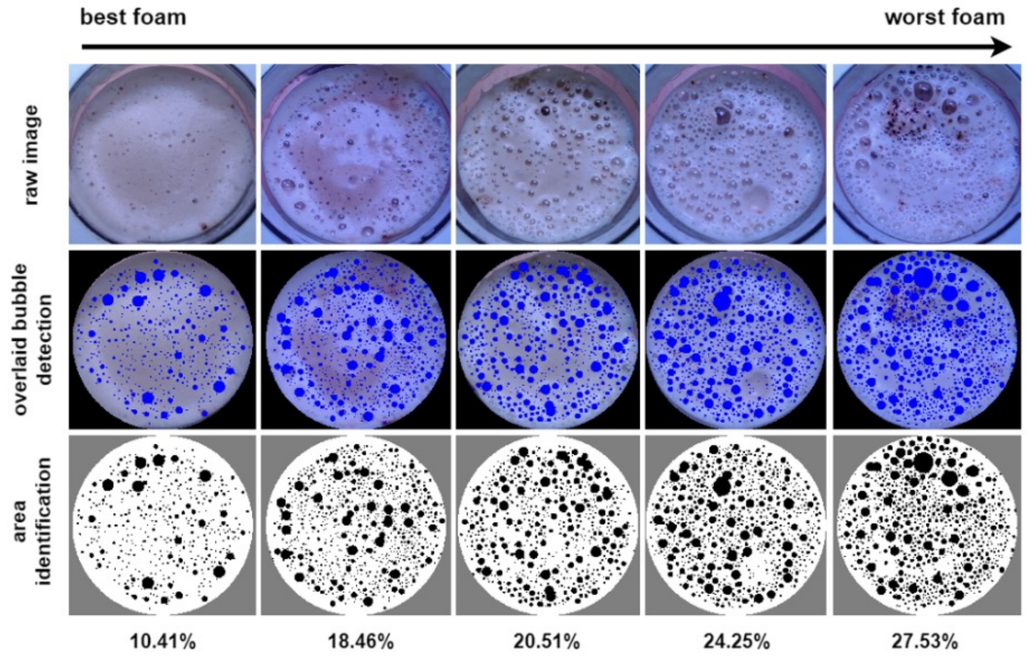


- Predict the thrust generated by soft structures for different controllers and morphologies given a training data set
- Can be used to optimize the design of soft swimming robots

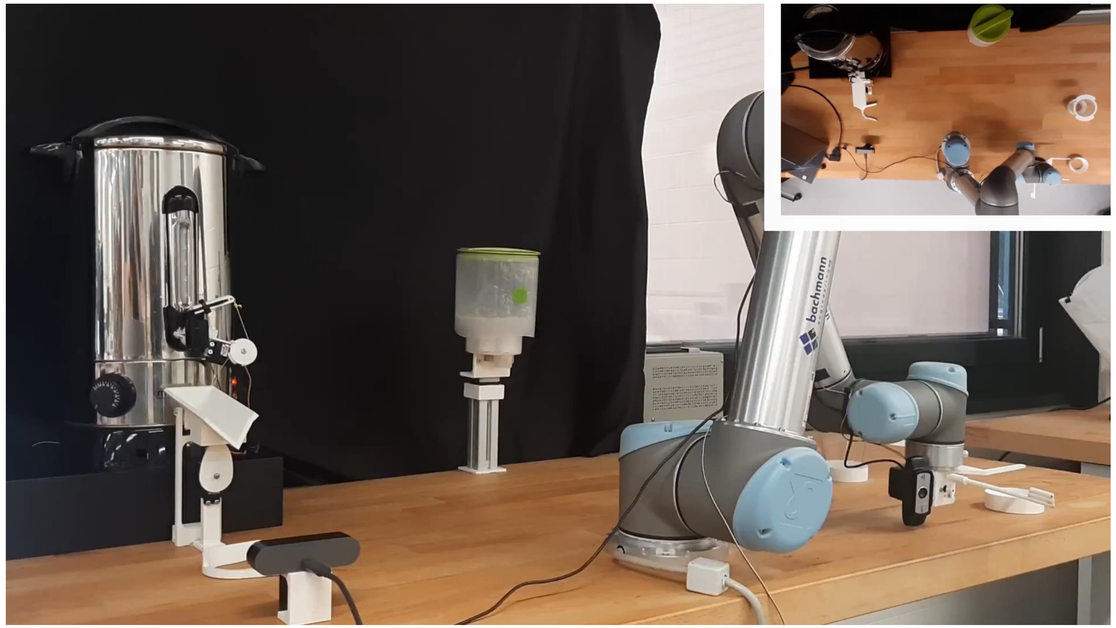
Robots ‘food scientists’

Optimizing coffee foam

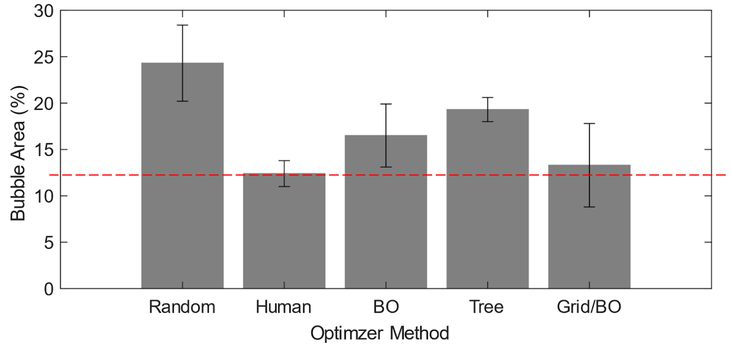
Metric for coffee bubbles



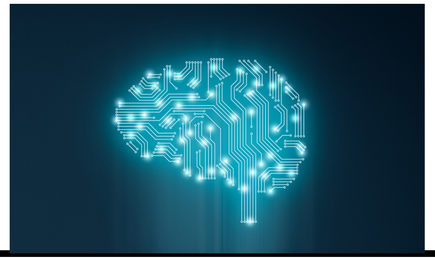
Objective: minimize bubbles, maximize height



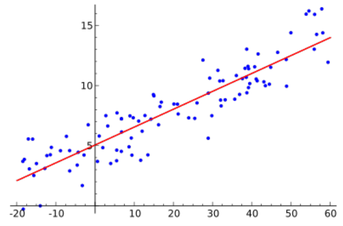
100+ coffees later…



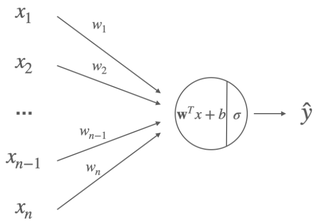
Introduction



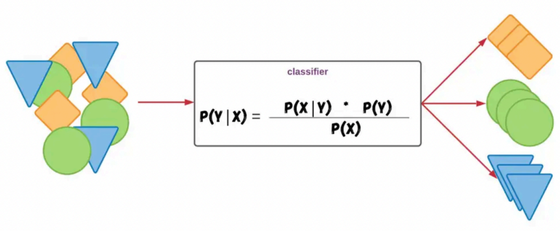
Linear regression



Logistic regression

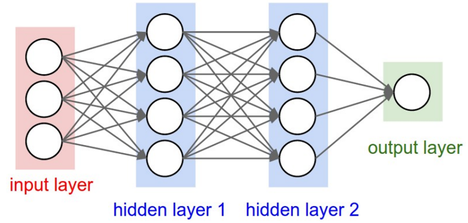


Naive Bayes

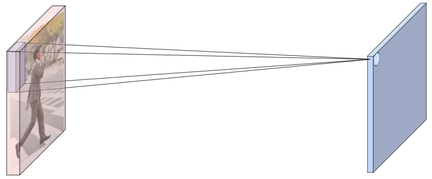


KNN

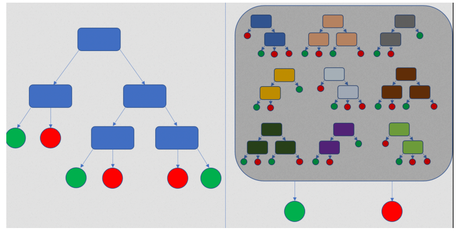
Neural networks



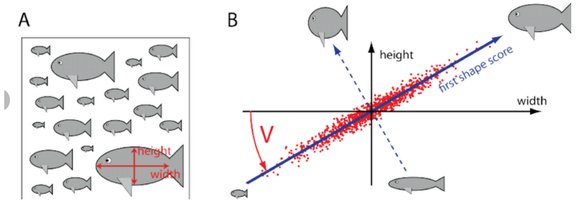
Convolutional neural networks



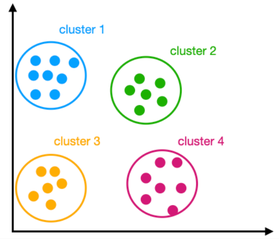
Decision-trees



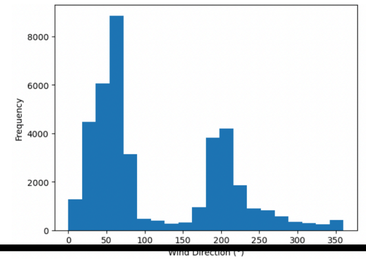
Dimensionality reduction



Clustering



Reinforcement learning



AI ethics

Neural networks

- Why deep learning?
- Neural networks
  - Architecture
  - Activation functions
- Training
  - Back propogation
  - Stochastic gradient descent

Review of supervised learning

< classification
regression

feature vectors, independent variables $x^i \in \mathbb{R}^d$

Labels, dependent variables, target, outcome y^i $\quad \{x^i, y^i\}_{i=1}^N$

Training data/set/example $\{x^i, y^i\}_{i=1}^N$

Sample, sample point

f is determined based on our ML approach:
- linear / logistic regression f_θ parametrized by θ
- Naive Bayes, kNN

Why deep learning?
Logistic regression review

Logistic regression for d-dimensional data:

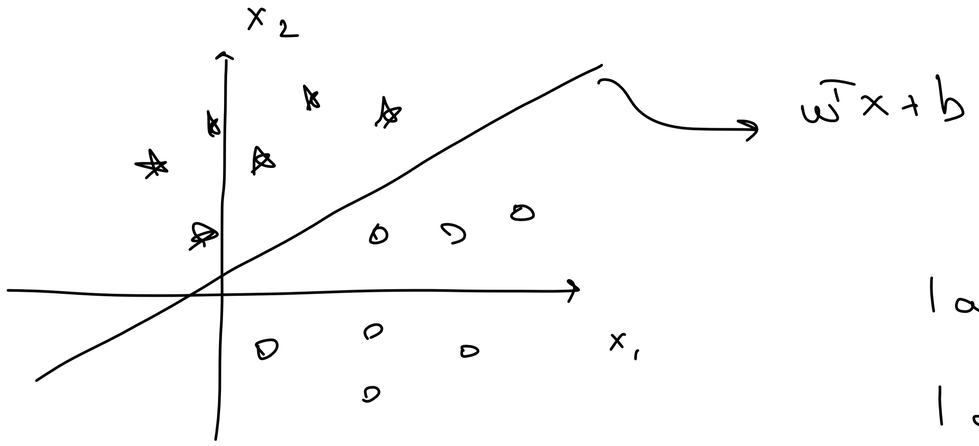
Input: $x \in \mathbb{R}^d$

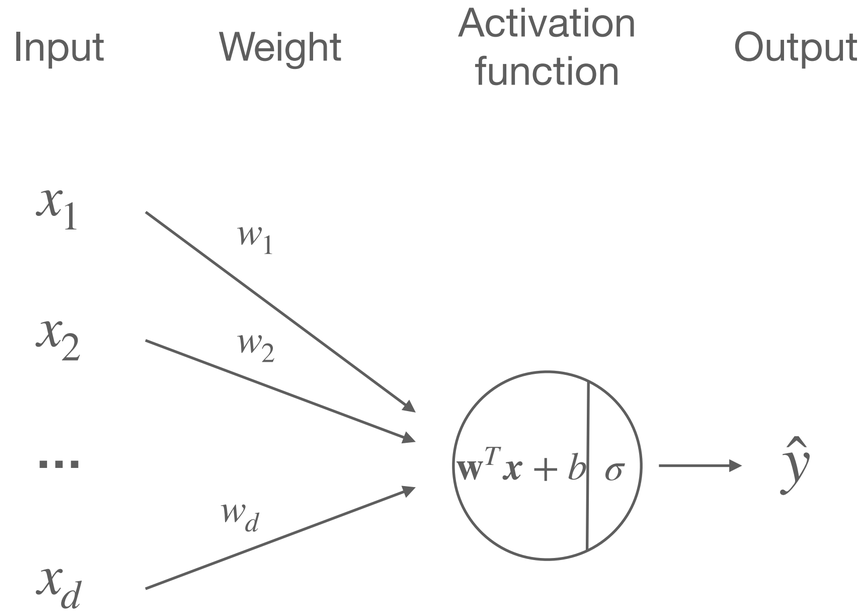
Weights: $w \in \mathbb{R}^d$

Bias: $b \in \mathbb{R}$

$\rightarrow \; w^T x + b =: z$

Logistic: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$





label ★ if $w^T x + b > 0$

label ○ otherwise.

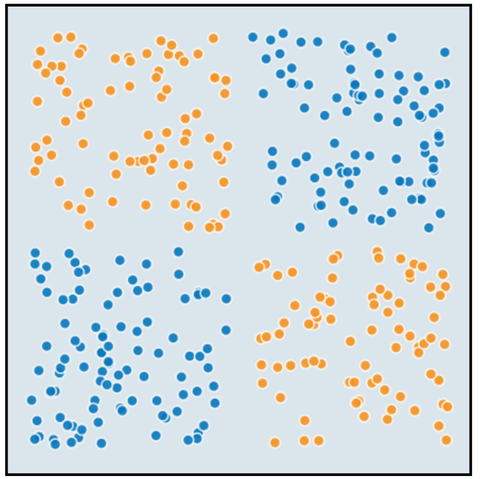
Why deep learning?
Limitations of logistic regression

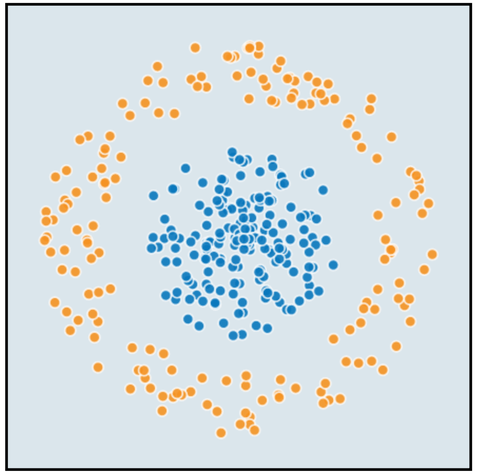
Issue: Logistic regression performs badly on non-linearly separable data

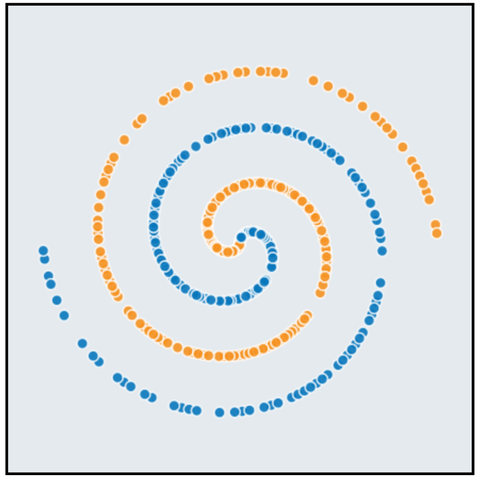
Potential fix: Use feature engineering to make data linearly separable, then use logistic regression

However:

- Features that linearly separate the data can be hard to find manually, specially in high dimension

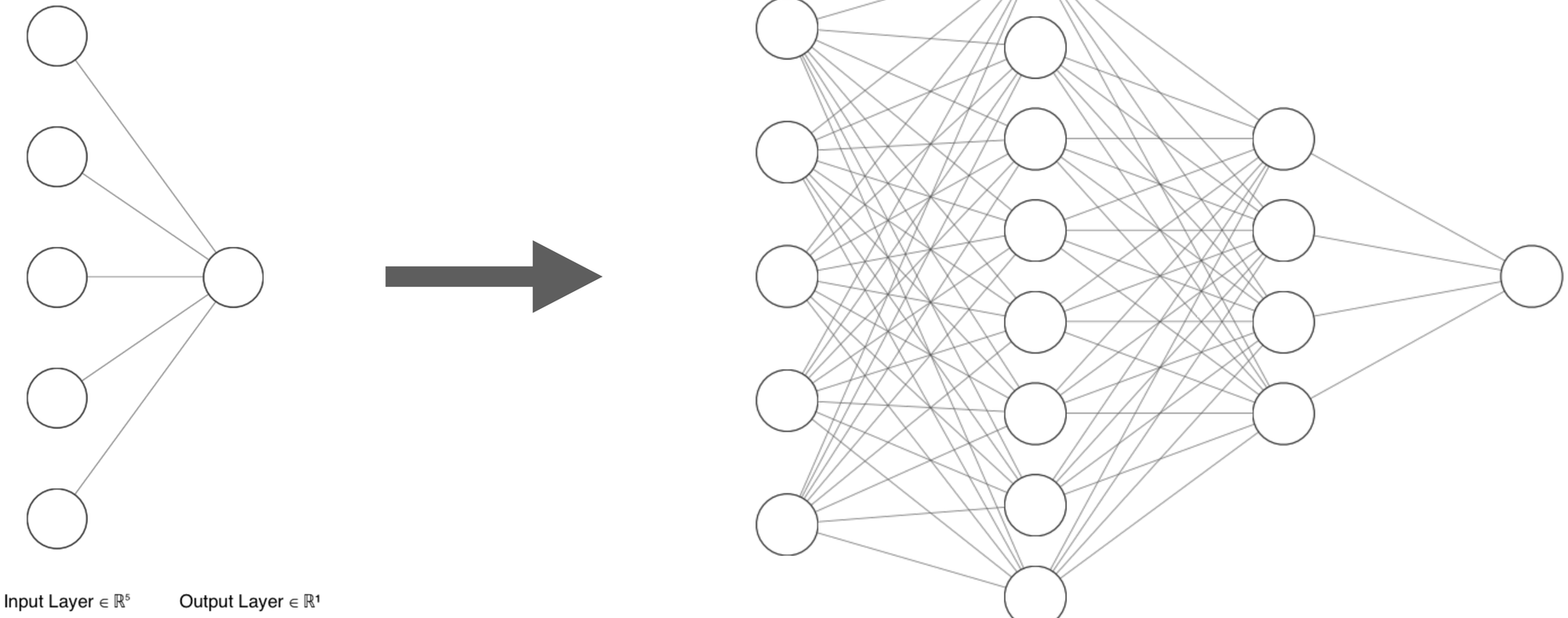






Why deep learning?
From logistic regression to neural networks

Neural networks have been successful in learning complex, non-linear functions



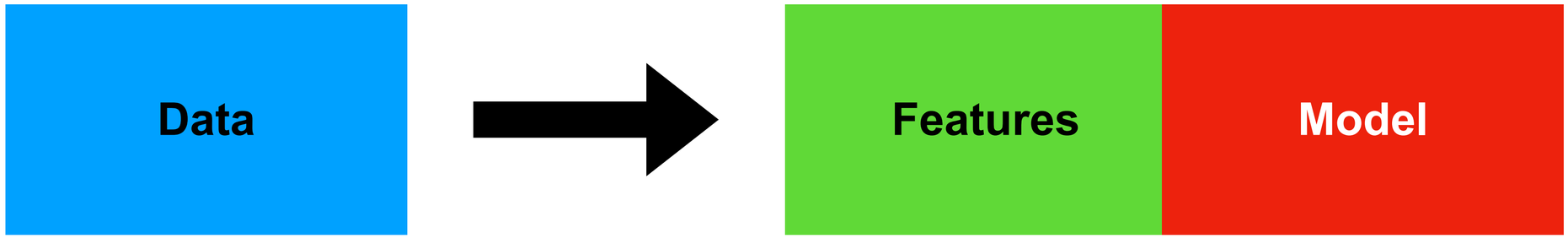
Why deep learning?

New way to approach ML

Before deep learning:

Hand-design the features

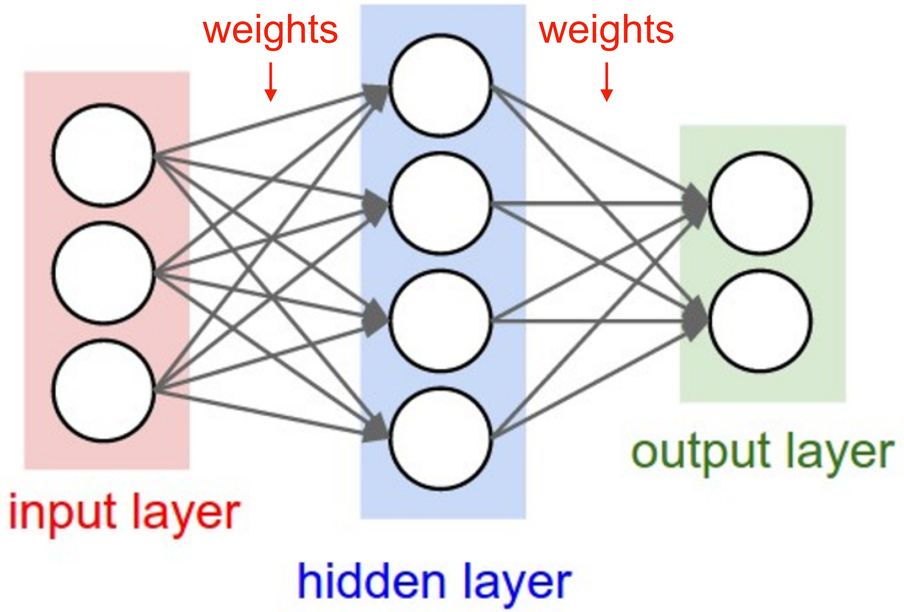
Deep learning:



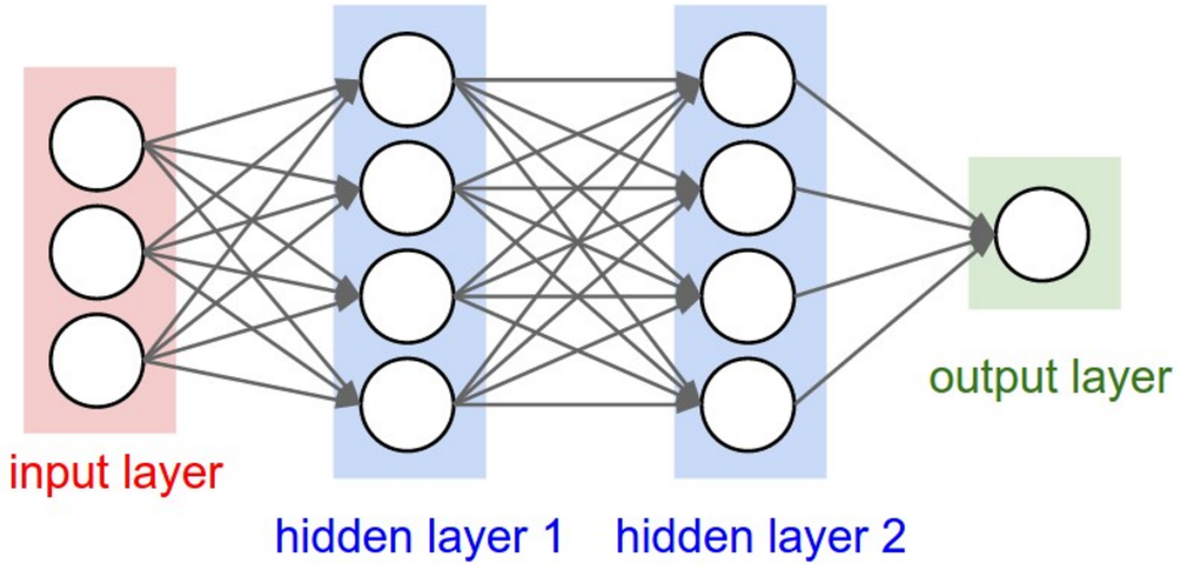
Deep neural networks derive
useful features from the data!

Neural networks

Neural networks
Representation



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”



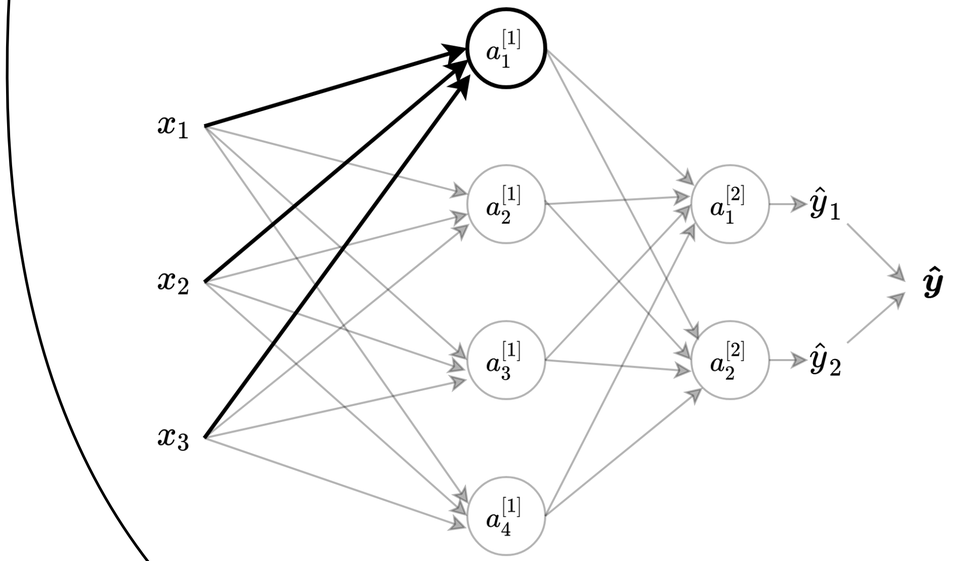
“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

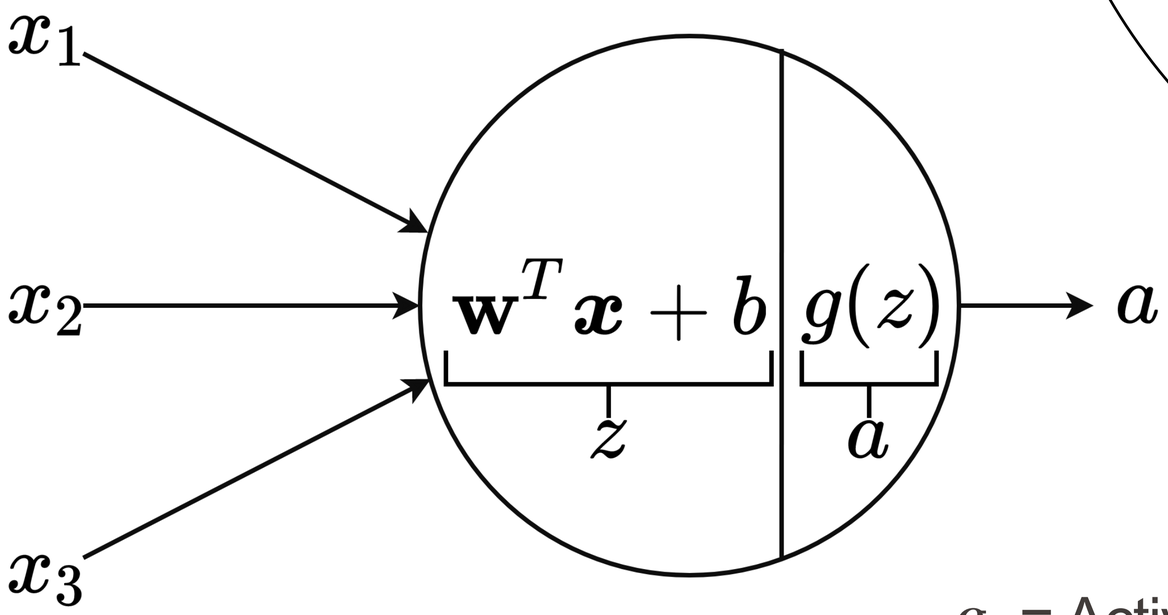
“Fully-connected” layers

Each neuron of a layer is connected
to all neurons of the following layer

Neural networks

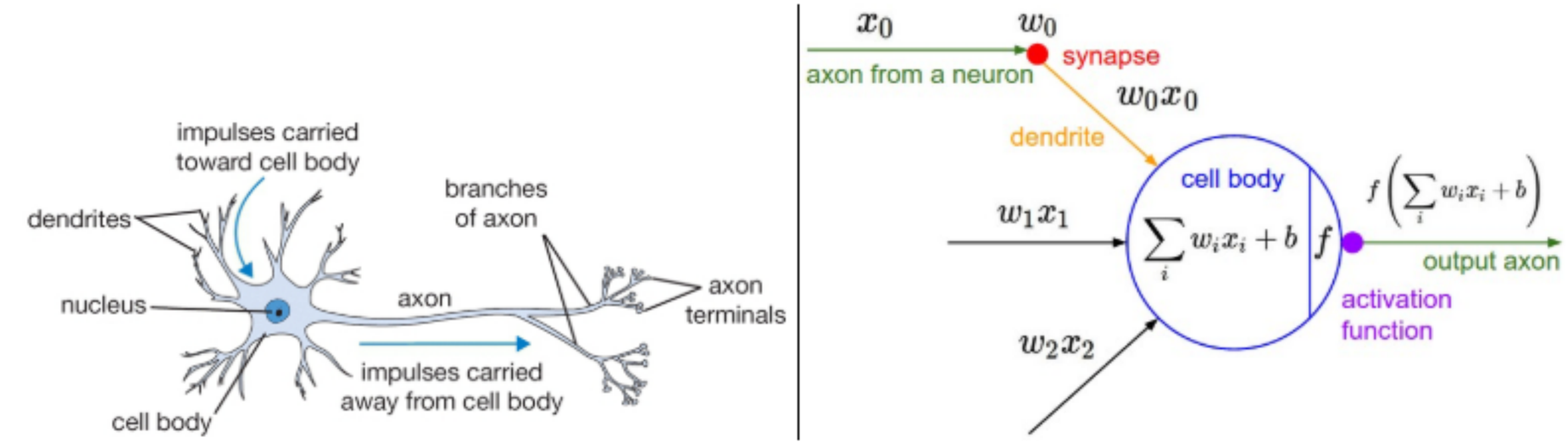
Inside a neuron





g = Activation function

Connections to biological neurons



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Source: towardsdatascience.com

Applications - nowadays everywhere!

- chatGPT / large language models
- midjourney ..
- object recognition (ex: classify obstacle / non-obstacle, dog, cat, ...)
- digit / hand-writing recognition

⋮

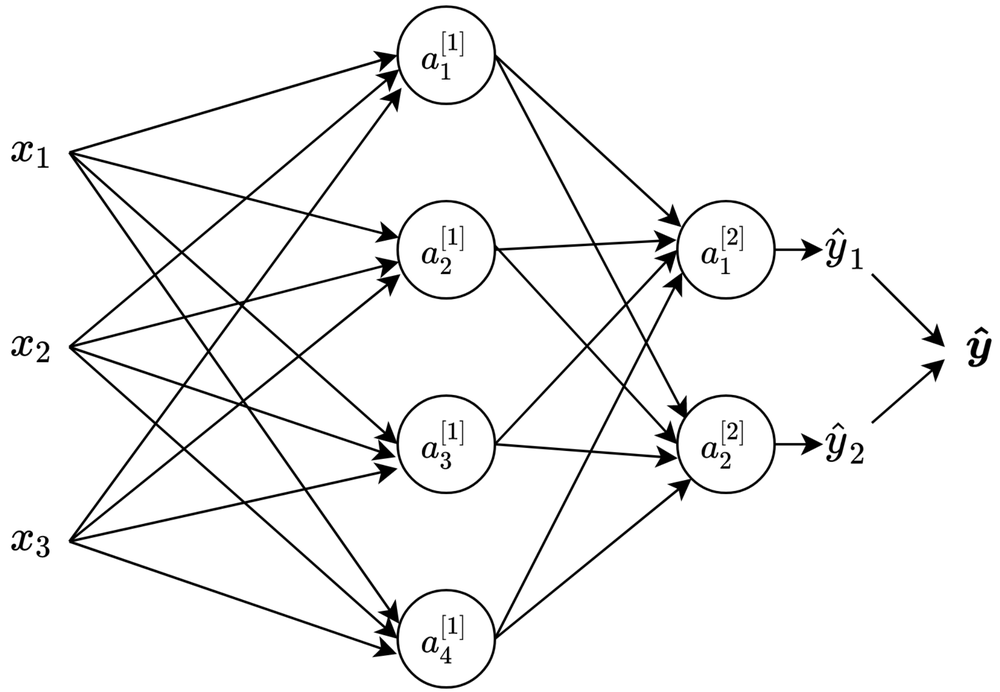
Engineering

control : dynamical system modelling
controller design..

Neural networks
Representation

$a_i^{[l]}$ ← Layer
$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leftarrow \text{Shape (3, 1)}$$



Neural networks
Representation

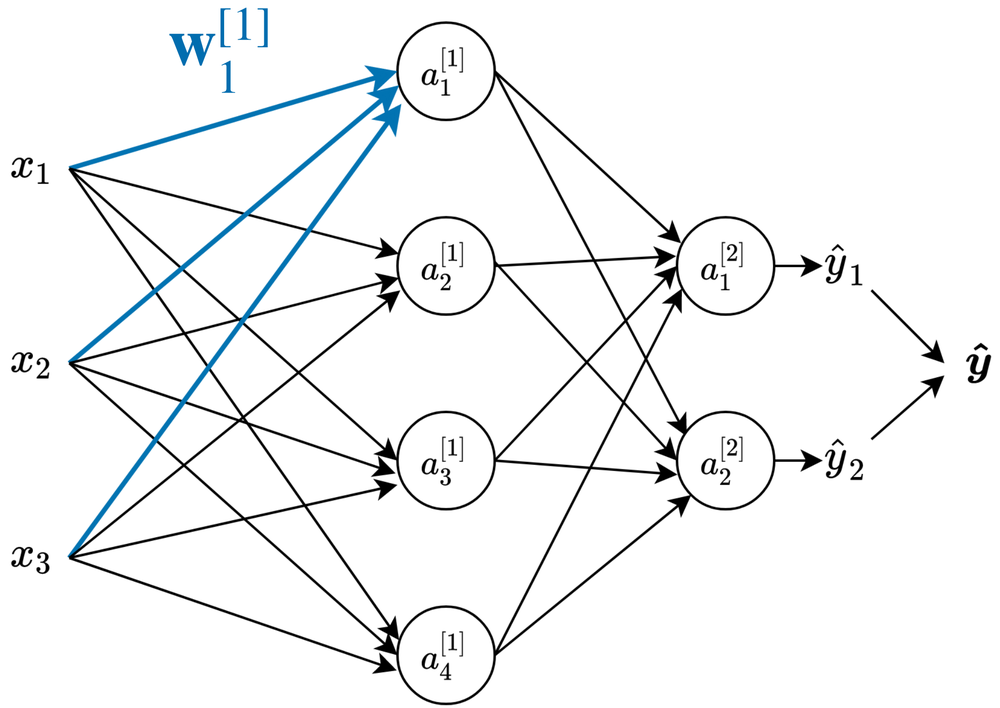
$a_i^{[l]}$ ← Layer

$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix}$$



Neural networks
Representation

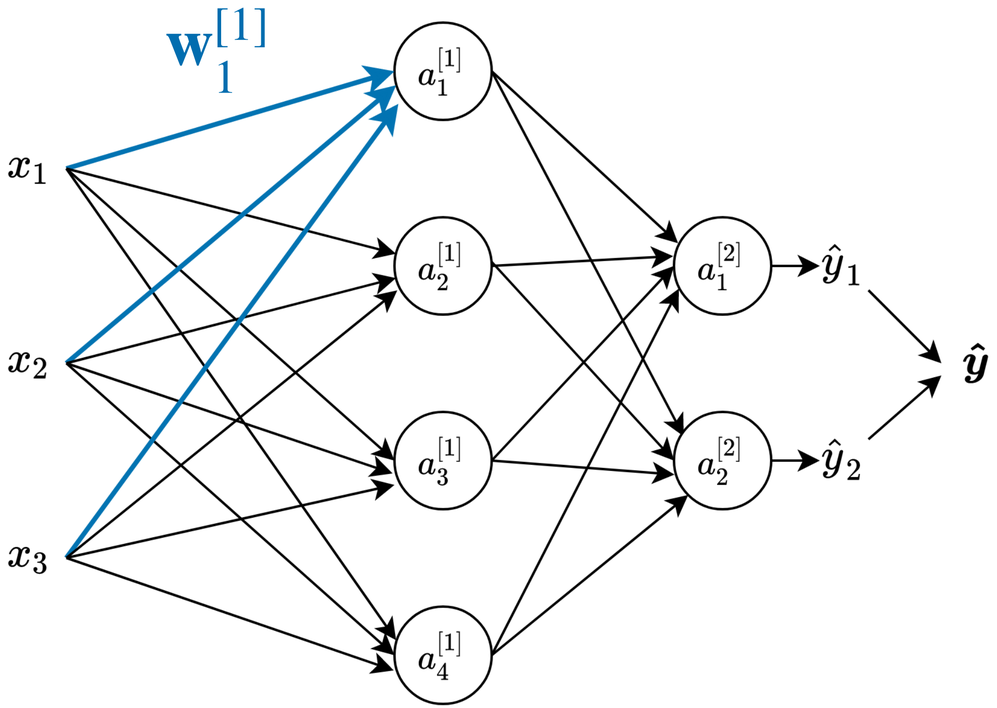
$a_i^{[l]}$ ← Layer

$a_i^{[l]}$ ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix}$$



$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]}$$
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]}$$
$$z_3^{[1]} = \mathbf{w}_3^{[1]T}\boldsymbol{x} + b_3^{[1]}$$
$$z_4^{[1]} = \mathbf{w}_4^{[1]T}\boldsymbol{x} + b_4^{[1]}$$

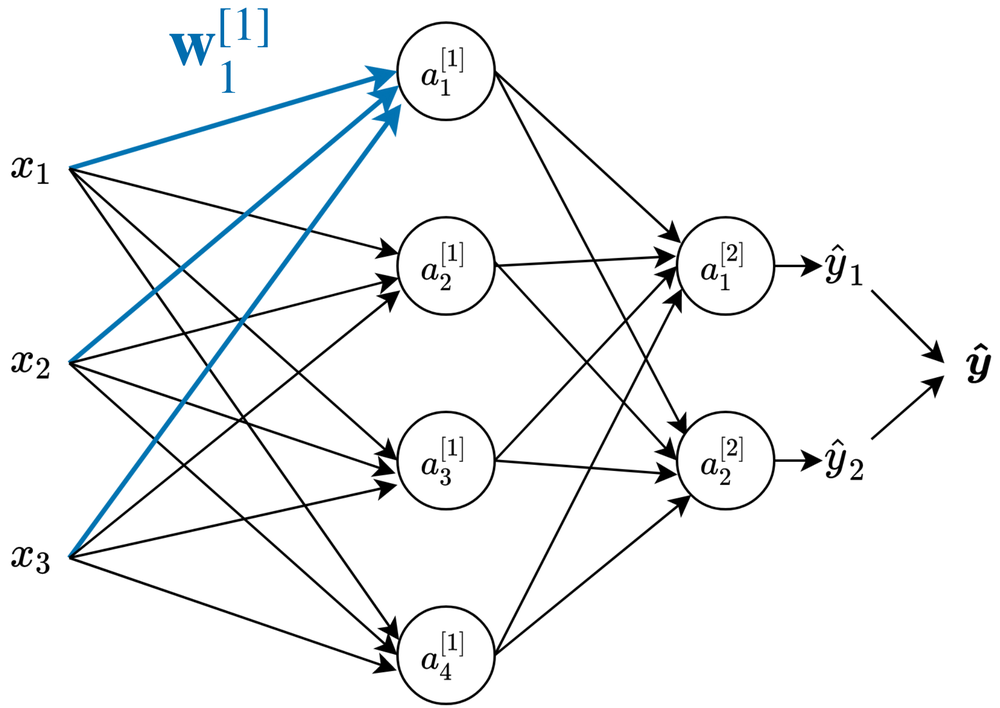
Neural networks
Representation

$a_i^{[l]}$ ← Layer; ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Weight vector for first node of first layer:

$$\mathbf{w}_1^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} \\ w_{1,2}^{[1]} \\ w_{1,3}^{1]} \end{bmatrix} \leftarrow \text{Shape (3, 1)}$$



$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]}$$
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]}$$
$$z_3^{[1]} = \mathbf{w}_3^{[1]T}\boldsymbol{x} + b_3^{[1]}$$
$$z_4^{[1]} = \mathbf{w}_4^{[1]T}\boldsymbol{x} + b_4^{[1]}$$

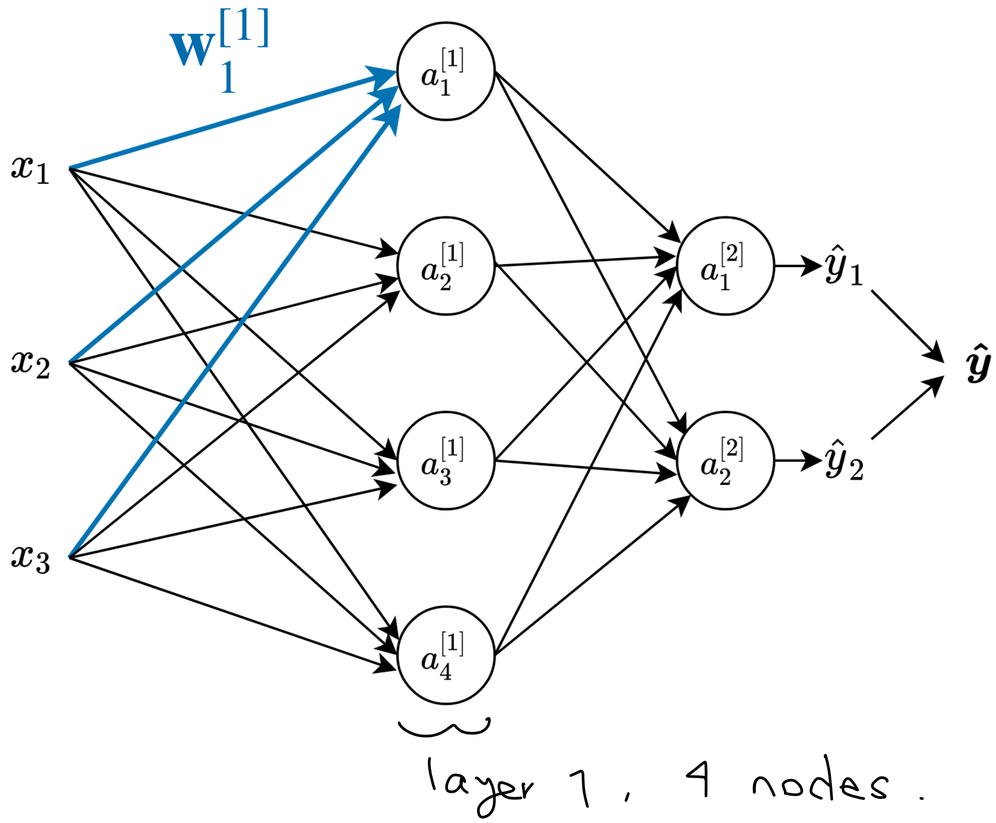
Apply activation →

$$a_1^{[1]} = g^{[1]}(z_1^{[1]})$$
$$a_2^{[1]} = g^{[1]}(z_2^{[1]})$$
$$a_3^{[1]} = g^{[1]}(z_3^{[1]})$$
$$a_4^{[1]} = g^{[1]}(z_4^{[1]})$$

Neural networks
Representation

$a_i^{[l]}$ ← Layer; ← Node in layer

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in \mathbb{R}^3$$



Vector notation:

weight vector

bias vector

$$\mathbf{W}^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \mathbf{w}_1^{[1]} & \mathbf{w}_2^{[1]} & \mathbf{w}_3^{[1]} & \mathbf{w}_4^{[1]} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \qquad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} \in \mathbb{R}^4$$

Shape (3, 4)

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]} \xrightarrow{\text{Apply activation}} \mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$z^{[1]} \in \mathbb{R}^4$, $W^{[1]} \in \mathbb{R}^{3\times 4}$

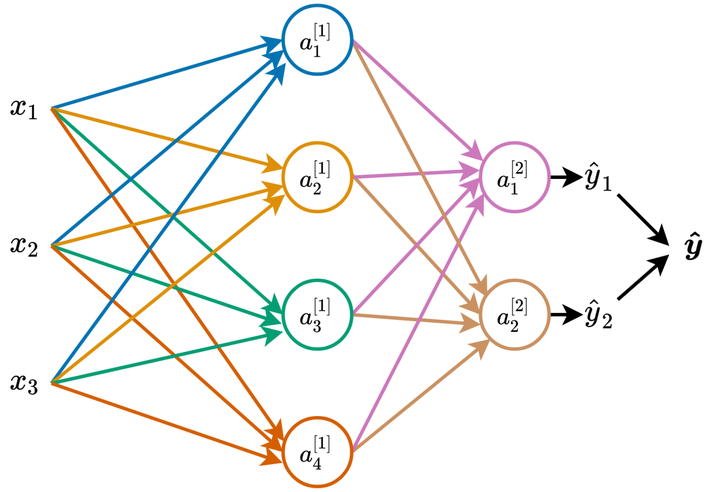
Activation functions

NN - Activation Function

Introduction

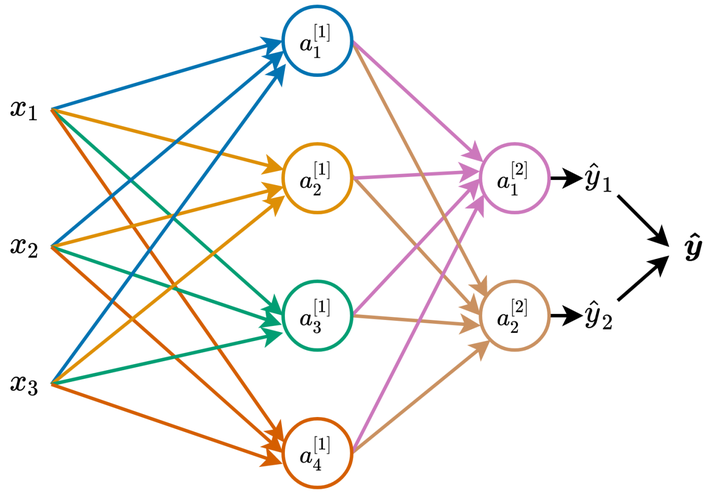
$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

Q: What happens if we remove the activations?



NN - Activation Function
Introduction

$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$



Q: What happens if we remove the activations?

$$\hat{\mathbf{y}} = \mathbf{W}^{[2]T}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$$

$$\hat{\mathbf{y}} = \mathbf{W}^{[2]T}\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{W}^{[2]T}\mathbf{b}^{[1]} + \mathbf{b}^{[2]}$$

Define $\mathbf{W}'^{T} = \mathbf{W}^{[2]T}\mathbf{W}^{[1]T}$ Define $\mathbf{b}' = \mathbf{W}^{[2]T}\mathbf{b}^{[1]} + \mathbf{b}^{[2]}$

$$\hat{\mathbf{y}} = \mathbf{W}'^{T}\boldsymbol{x} + \mathbf{b}'$$

A: We end up with a linear classifier!

NN - Activation functions
Introduction

To model a nonlinear problem:

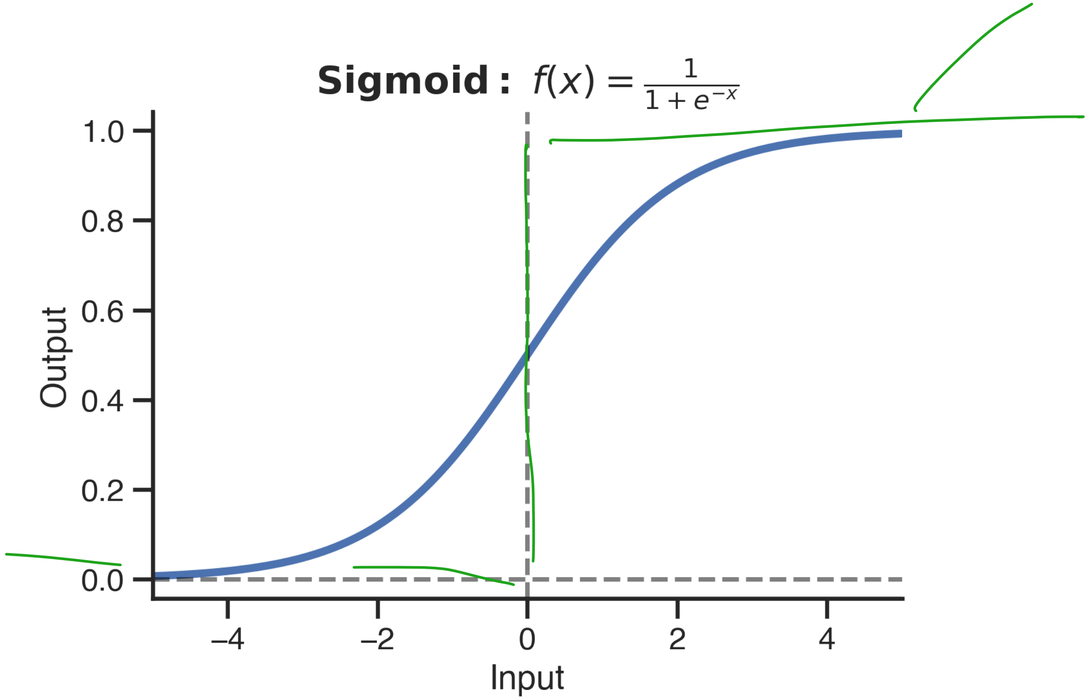
- Pass the output of each neuron through a nonlinear function, called **activation function**
- **Connection to neuron firing in brain**

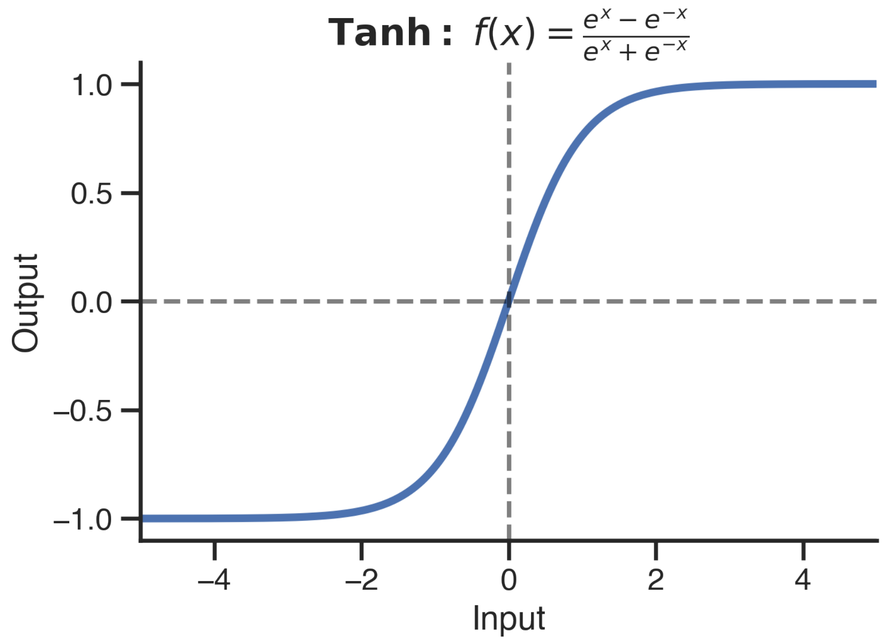
Some well-known activation functions:

- Sigmoid
- Tanh
- ReLU

NN - Activation functions

Overview





Sigmoid (σ):

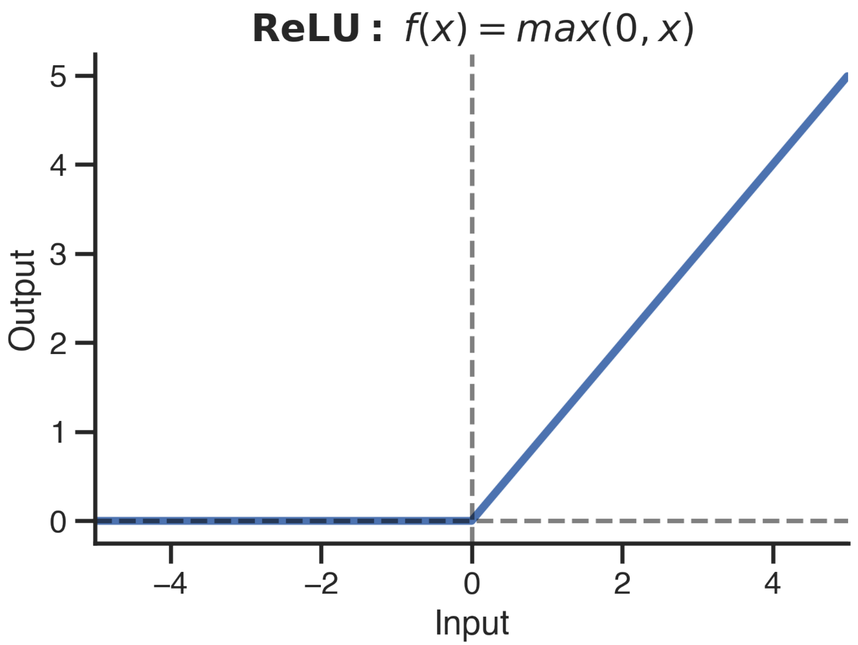
- Squashes input in a [0, 1] range
- Approximately nullifies gradient (for “large” positive or negative inputs) -> **vanishing gradient problem**
- rarely used except for final layer of binary classification network

Tanh:

- Squashes input in a [-1, 1] range
- Like sigmoid, nullifies gradient (for “large” positive or negative inputs)
- Zero-centered, preferable over sigmoid as an activation
- Rarely used in practice (ReLU is more popular)

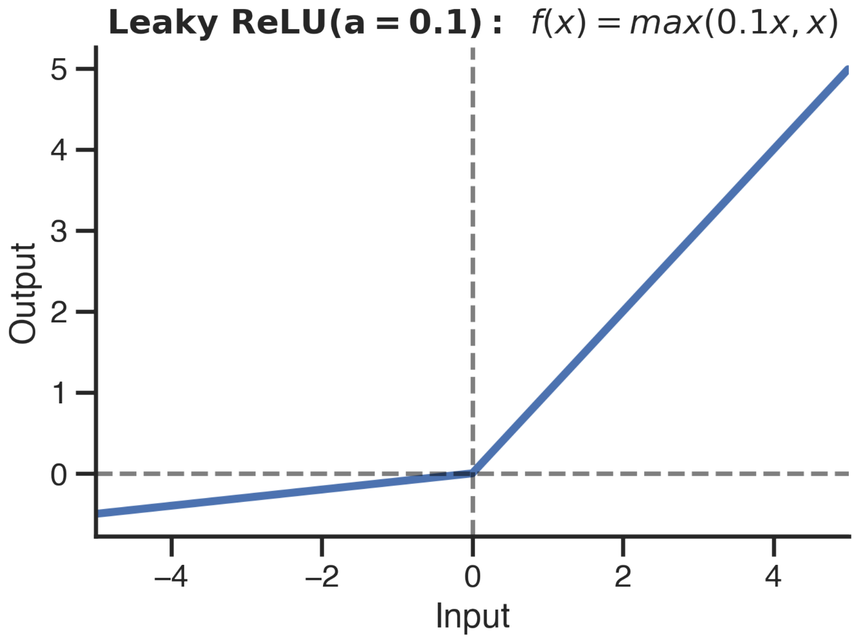
NN - Activation functions

Overview



Rectified Linear Unit (ReLU):

- Easily computed, simple gradient
- Greatly accelerates convergence of gradient descent
- Saturates in only one direction, suffers less from vanishing gradient problem
- commonly used in practice



Leaky ReLU:

- Attempts to fix “dying ReLU” problem by having a small negative slope for x < 0.
- Leaky ReLU and other ReLU variants (ELU, SELU, GELU, Swish, etc…) are sometimes used over ReLU

NN - Activation functions

Derivatives

Sigmoid:

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

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

Tanh:

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

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^{2}(x)$$

Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} = \max(0,\, x)$$

$$\frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Note: Derivative of ReLU is undefined for $x = 0$. By convention, it is set to 0.

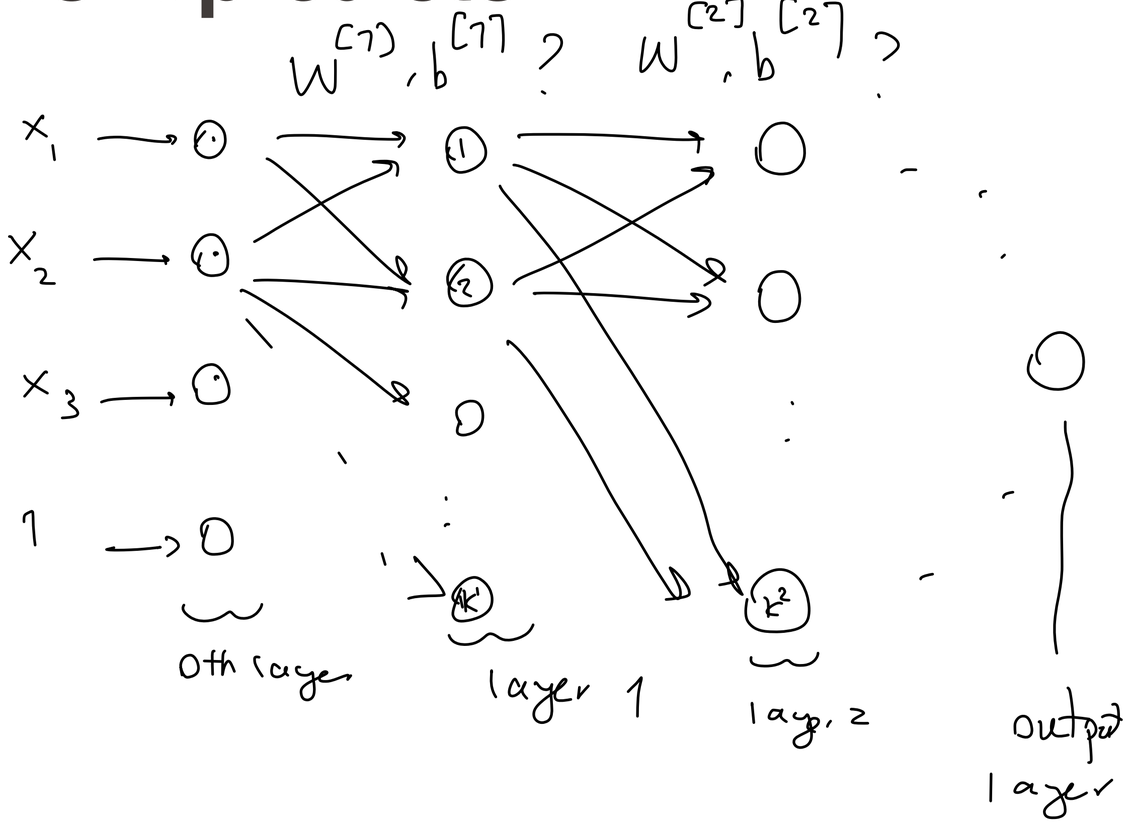
Training neural nets

Determining the neural network predictor

Training

1) Choose the neural network architecture

- number of layers
- type of activation function
- number of nodes in each layer



2) Choose a loss function

regression : $\frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2$

classification : cross-entropy loss

$\hat{y}^i$: predicted based on NN.

3) Minimize the loss

$$\min_{W^{[k]}, b^{[k]}_{k=1,\dots,T}} J_{\text{loss}}\left(\{y^i\}_{i=1}^{N}, \{W^{[k]}, b^{[k]}\}_{k=1}^{T} \right)$$

where T is the number of layers.

Neural networks
Training

Forward pass of 2 layer NN (for a single example):

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]T}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\hat{\mathbf{y}} = \mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$

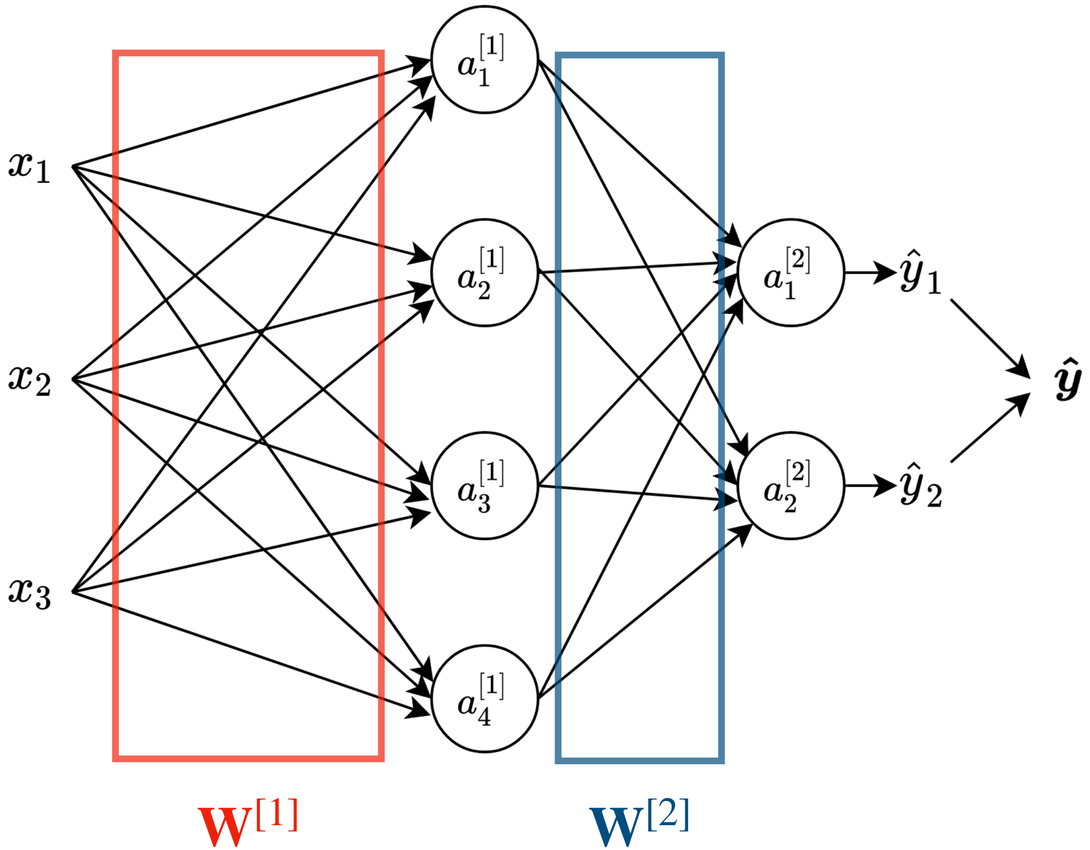
$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

depth 2 network ...

you can see how to

generalize to depth T

network



Neural networks
Training

Forward pass of 2 layer NN (for a single sample):

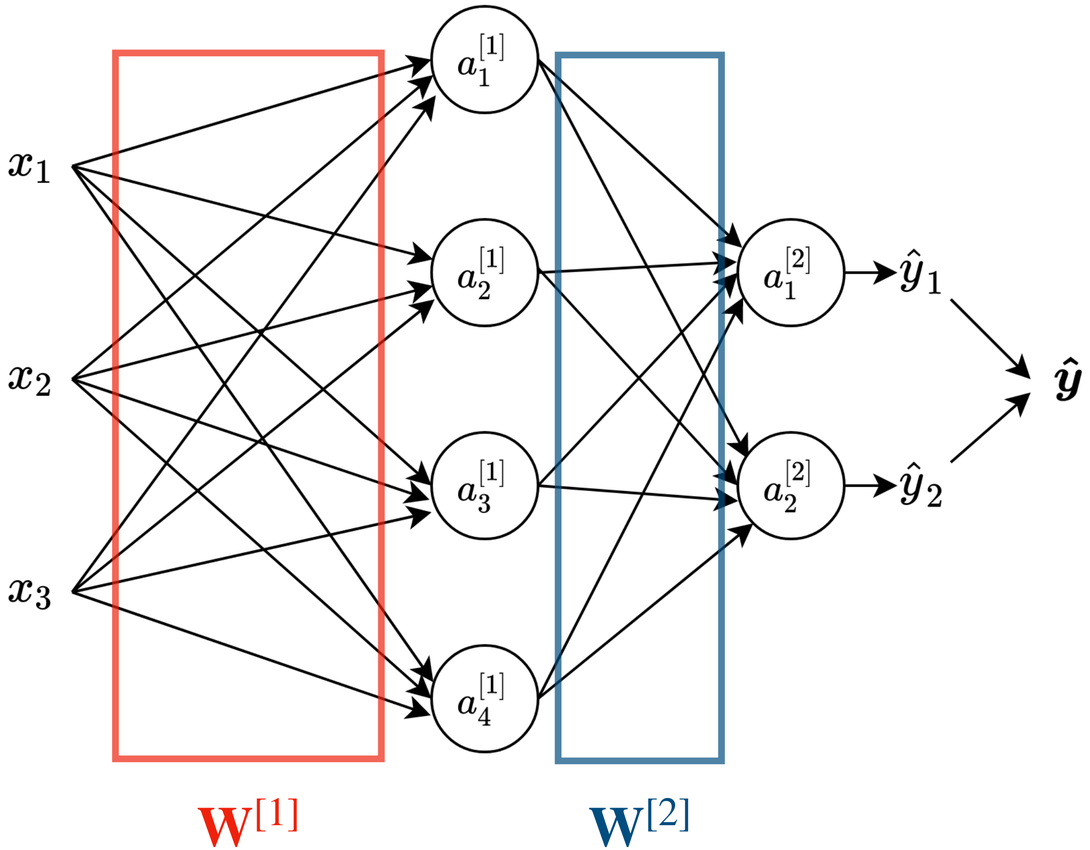
$$\hat{\mathbf{y}} = g^{[2]}(\mathbf{W}^{[2]T} g^{[1]}(\mathbf{W}^{[1]T}\boldsymbol{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]})$$

To train, we need a **loss function**: $L(\hat{\mathbf{y}}, \mathbf{y})$

Using that loss function, we want to update
$\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}$
using **gradient descent**.

$W^{[1]} \in \mathbb{R}^{3\times 4}$, $W^{[2]} \in \mathbb{R}^{4\times 2}$

$b^{[1]} \in \mathbb{R}^{4}$, $b^{[2]} \in \mathbb{R}^{2}$



Loss function

Regression $\{(x^i, y^i)\}_{i=1}^N$, $y^i \in \mathbb{R}$

mean square error (MSE)

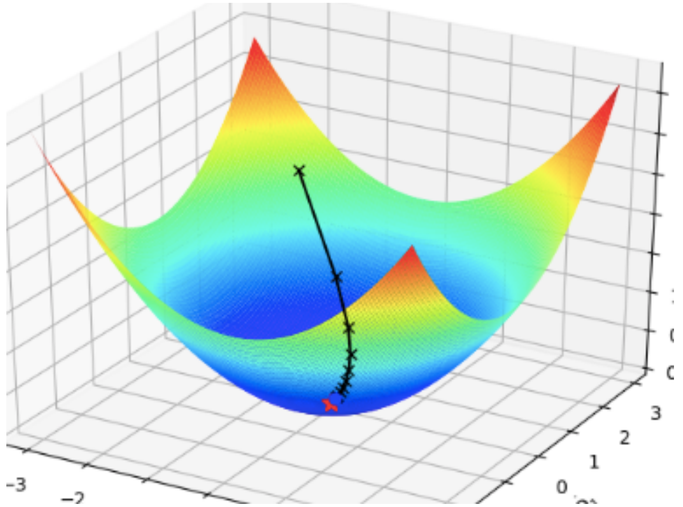
$$L(W,b) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2$$ - $\hat{y}^i$ prediction of neural net

- output layer has often no non-linearity.

Classification $\{(x^i, y^i)\}_{i=1}^N$ $y^i \in \{1, 2, \dots, K\}$

- output layer has K neurons, followed by cross-entropy loss.

Gradient descent



Need to compute: $\frac{\partial L}{\partial \mathbf{W}^{[i]}}, \frac{\partial L}{\partial \mathbf{b}^{[i]}}$

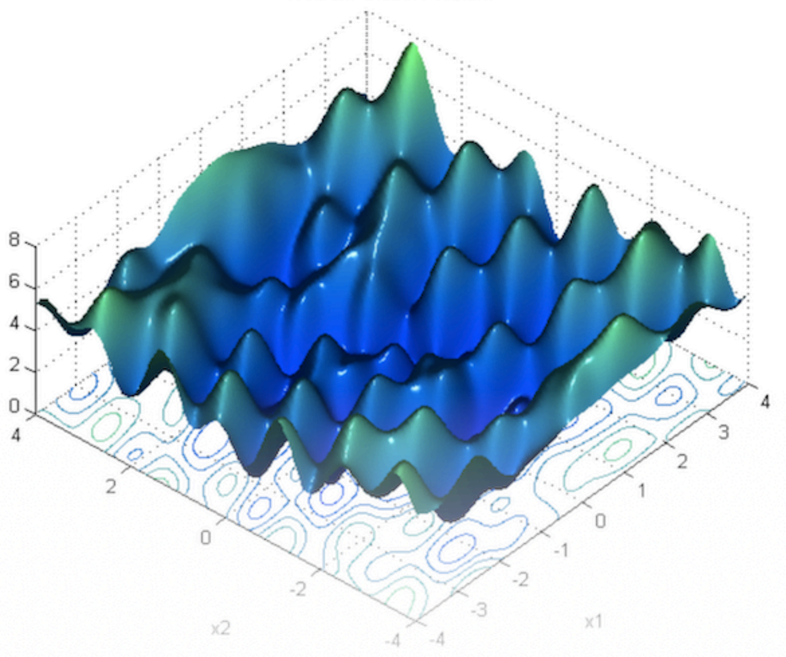
=> Gradient of loss with respect to weights and biases of each layer

Once gradients are computed,

update weights with:

- $\mathbf{W}_{t+1}^{[i]} := \mathbf{W}_{t}^{[i]} - \alpha_t \frac{\partial L}{\partial \mathbf{W}^{[i]}}(\mathbf{W}_t, \mathbf{b_t})$
- $\mathbf{b}_{t+1}^{[i]} := \mathbf{b}_{t}^{[i]} - \alpha_t \frac{\partial L}{\partial \mathbf{b}^{[i]}}(\mathbf{W}_t, \mathbf{b_t})$

where α_t is the learning rate



Neural networks
Forward / Backward pass

Forward pass: Compute the output of a neural network for a given input

Backward pass: Compute derivatives of the network parameters given the output

During **training**, you need both the **forward** pass and the **backward** pass.

During **prediction** (inference**)**, you only need the **forward** pass.

Inference: the process of using a trained machine learning model for prediction

EPFL

Computing gradients
Back propagation

example $x \in \mathbb{R}^d$, $y \in \mathbb{R}$

$L(\hat{y}^i, y^i) = (\hat{y}^i - y^i)^2$, (Recall $L(W,b) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2$)

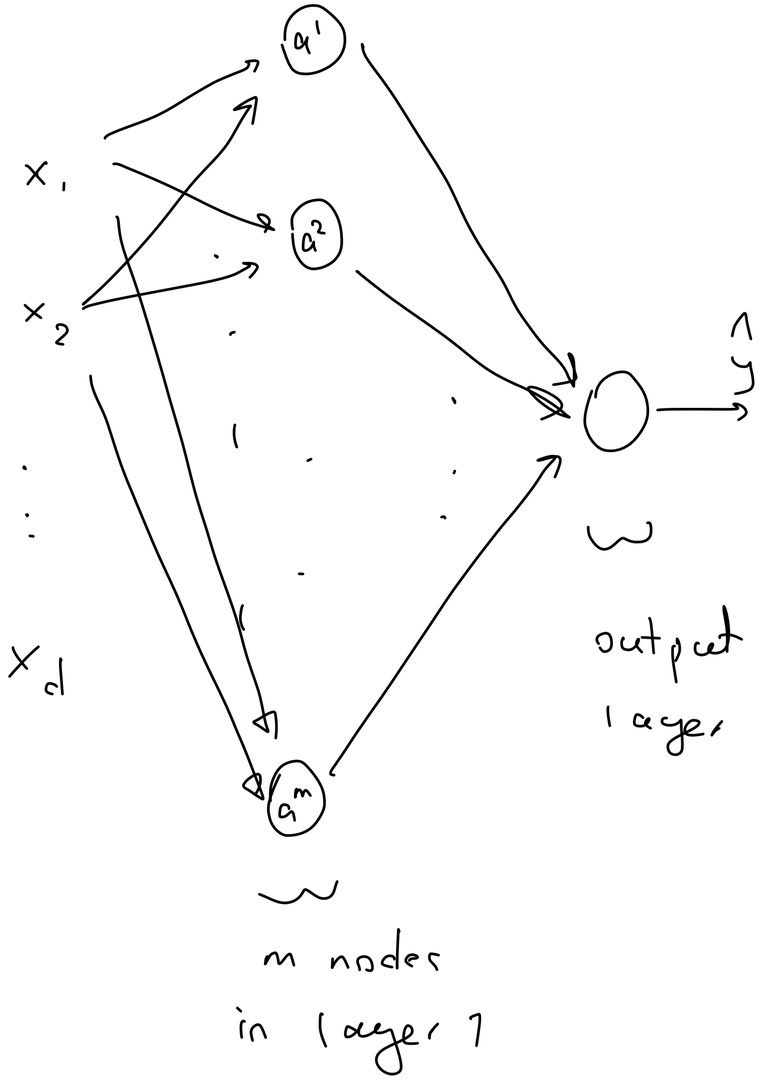
We need to compute

$$\frac{\partial L(W,b)}{\partial w_{j,l}}, \quad \frac{\partial L(W,b)}{\partial b_j}, \quad \frac{\partial L(W,b)}{\partial w_{0,k}}, \quad \frac{\partial L(W,b)}{\partial b_0}$$

where $\{w_{j,l}\}_{j=1,\dots,m,\ l=1,\dots,d}$ are weights of layer 1, $\{b_j\}_{j=1}^{m}$ are biases of layer 1

$\{w_{0,k}\}_{k=1,\dots,m}$ are weights of the output layer.

$b_0 \in \mathbb{R}$ bias of output layer



EPFL

Computing gradients
Back propagation

Now, we focus on $\frac{\partial}{\partial w_{0,k}} (y - \hat{y})^2$, where we drop i superscript for simplicity

$$\hat{y} = g_0(b_0 + w_{0,1} a_1 + w_{0,2} a_2 + \cdots + w_{0,m} a_m) = g_0(z_0), \quad g_0 : \mathbb{R} \to \mathbb{R}$$

$$a_j = g(b_j + w_{j,1} x_1 + w_{j,2} x_2 + \cdots + w_{j,d} x_d) = g(z_j), \quad j = 1, \ldots, m.$$

$$\frac{\partial (y-\hat{y})^2}{\partial w_{0,k}} = 2(y-\hat{y}) \frac{\partial \hat{y}}{\partial w_{0,k}} = 2(y-\hat{y}) \frac{dg_0}{dz_0} \frac{\partial z_0}{\partial w_{0,k}} = \underbrace{2(y-\hat{y}) \frac{dg_0}{dz_0}}_{\gamma_0} a_k = \gamma_0 a_k$$

$$\frac{\partial (y-\hat{y})^2}{\partial w_{j,l}} = 2(y-\hat{y}) \frac{\partial \hat{y}}{\partial w_{j,l}} = 2(y-\hat{y}) \frac{dg}{dz_0} \frac{\partial z_0}{\partial a_j} \frac{\partial a_j}{\partial w_{j,l}}$$

$$= \gamma_0 \underbrace{w_{0,j} \frac{dg}{dz_j}}_{\gamma_j} \frac{\partial z_j}{\partial w_{j,l}} = \gamma_0 \gamma_j x_l$$