

# Deep Learning

## Basics in Deep Learning

Lionel Fillatre

2024-2025

# Outline of Lecture 1

- Logistic and examination
- Introduction
- Some successful applications
- Basics in learning theory
- Basics in deep neural networks
- IT tools
- Conclusion

# Logistic and Examination

# Class website

- Class website

<https://lms.univ-cotedazur.fr/2024/user/index.php?id=1867>

- Password for the course registration: **mp89@GH34**
- **You must register yourself!**

# Goal of the course

- Overview
  - "How" Deep Learning works
  - Main advantages of DL
  - Main drawbacks of DL
  - Successful applications of DL
  - Use a neural network right away, get results and then understand it!
- Warning: Fast evolving field with a lot of ressources available online

# Timeline of this course

1. Basics in Deep Learning (16/10/2024)
2. Deep Neural Networks (23/10/2024)
3. Convolutional Neural Network (30/10/2024)
4. Power of depth + Lab report (06/11/2024)
5. Object Recognition (13/11/2024)
6. Recurrent Neural Network (20/11/2024)
7. Transformer (27/11/2024)
8. Unsupervised Deep Learning + Final exam on all the lectures (04/12/2024)

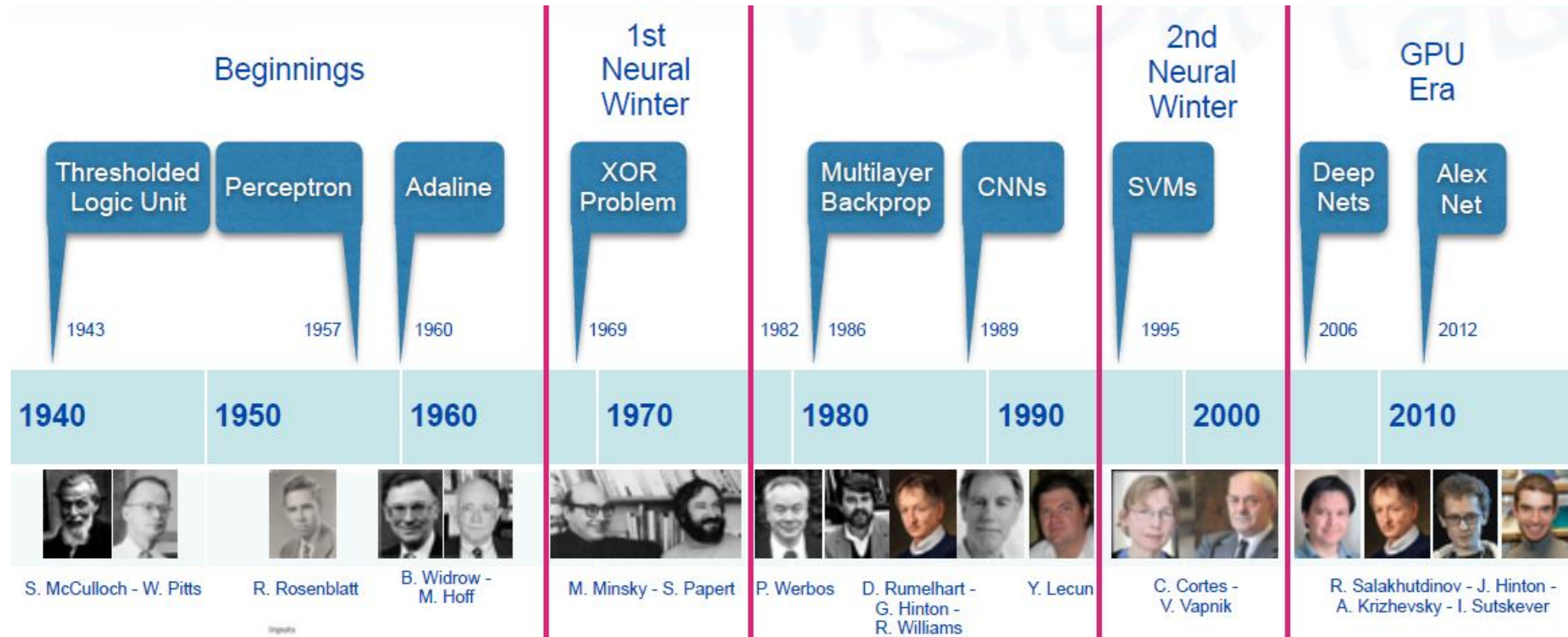
# Examination

- Lab report (**grade G1**):
  - Do the lab report (Python notebook) during the lab hours
  - Send the lab to the teacher at the end of the lab (as a pdf file)
  - Teamwork (2 students, maximum 3 students to avoid single student)
- Final examination (1 hour - **grade G2**)
  - written exam
  - Exercises focused on all the lecture topics
  - Questions on the lectures may be included
  - No questions on Python/Pytorch
- Final grade :  **$0.5 \times G1 + 0.5 \times G2$**

# Introduction



# Brief Timeline of Neural Networks



# Turing Award for Deep Learning

- The Turing Award, which was introduced in 1966, is often called the Nobel Prize of computing
- Turing Award 2018 given for: “The conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.”



# Why do we need Deep Learning?

- Computer vision
- Natural speech recognition
- Self-driving car
- And many other applications



## ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.





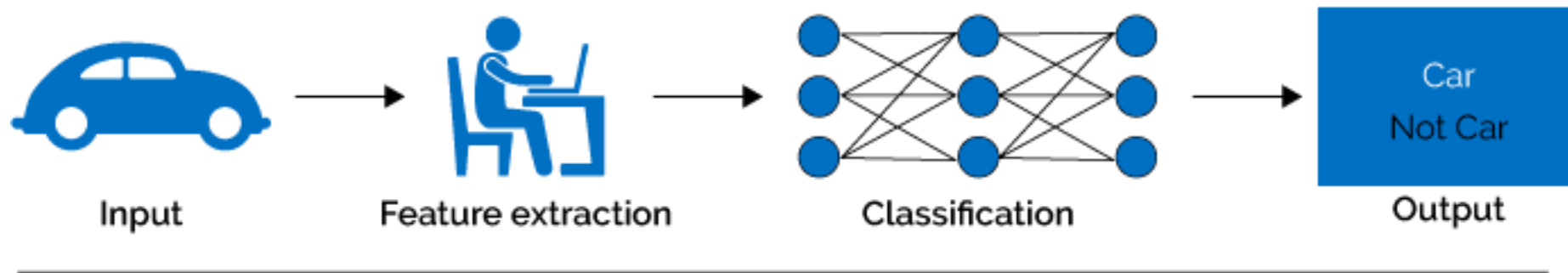
# Some Cold Water

- Problem: why ? How can we trust a blackbox?

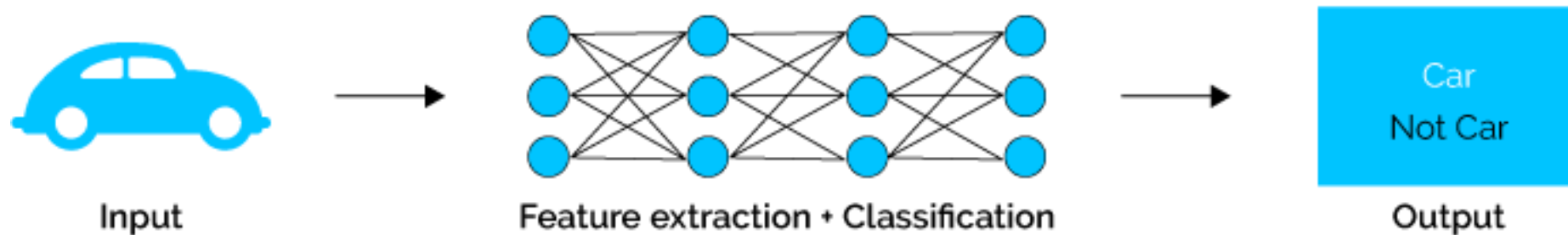


# What is Deep Learning

## Machine Learning



## Deep Learning



# Why Deep Learning Now?



- Five decades of research in machine learning
- CPUs/GPUs/storage developed for other purposes
- Lots of data from “the internet”
- Tools and culture of collaborative and reproducible science
- Resources and efforts from large corporations

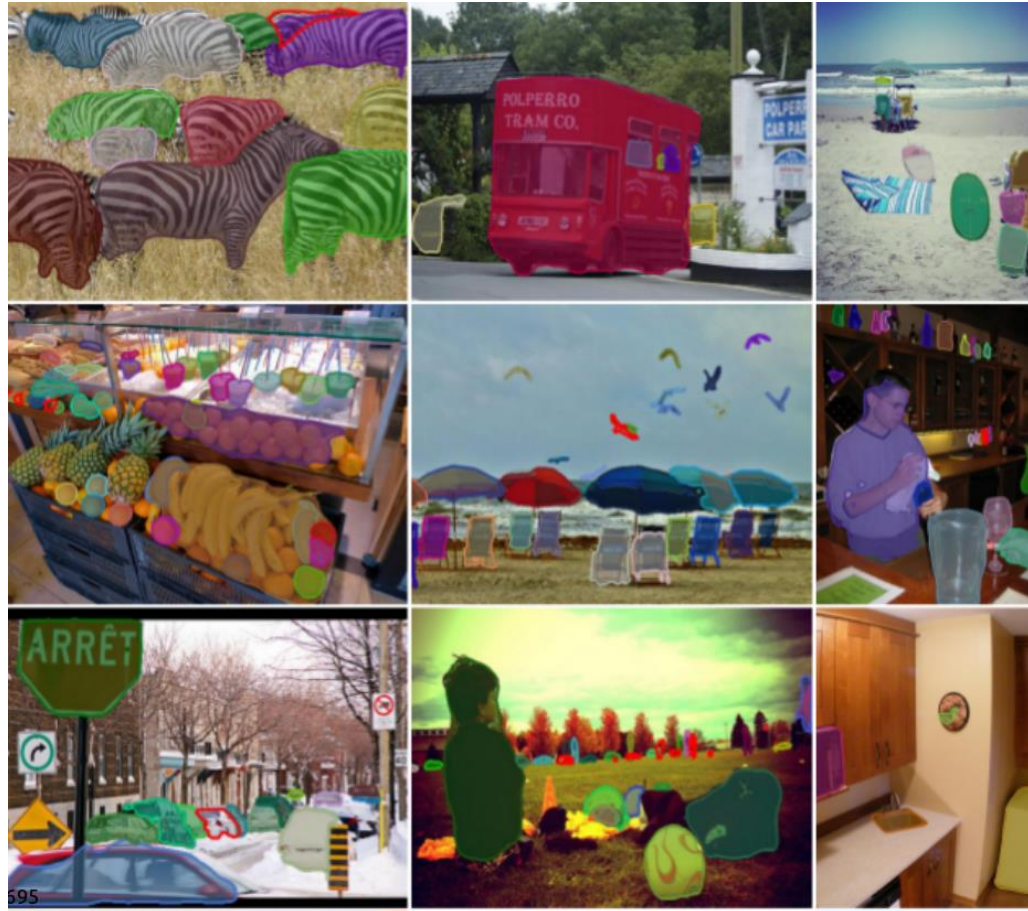


Some successful applications



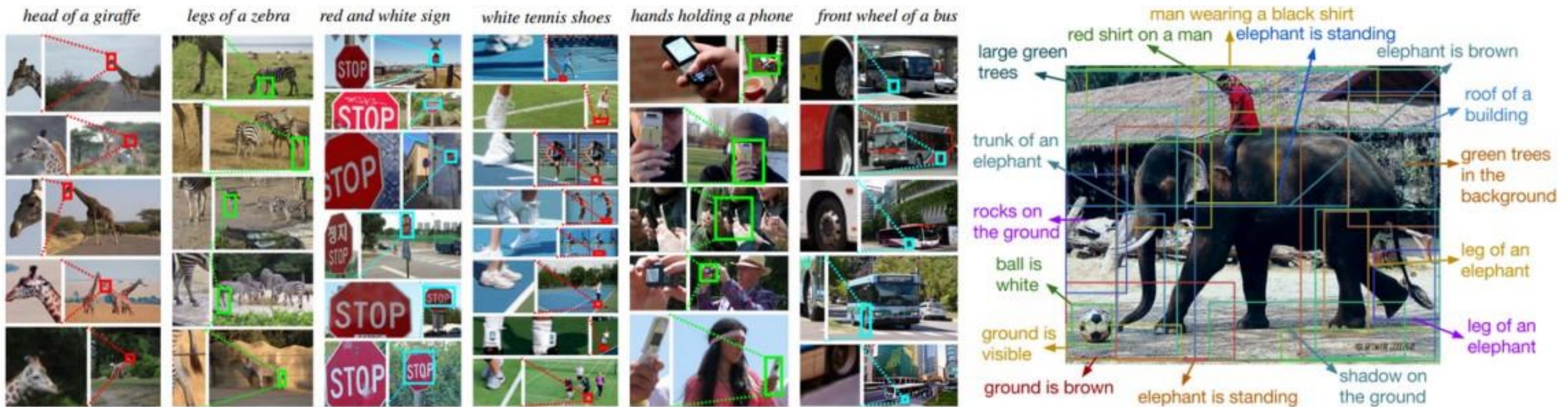
# Vision

- Object detection and segmentation

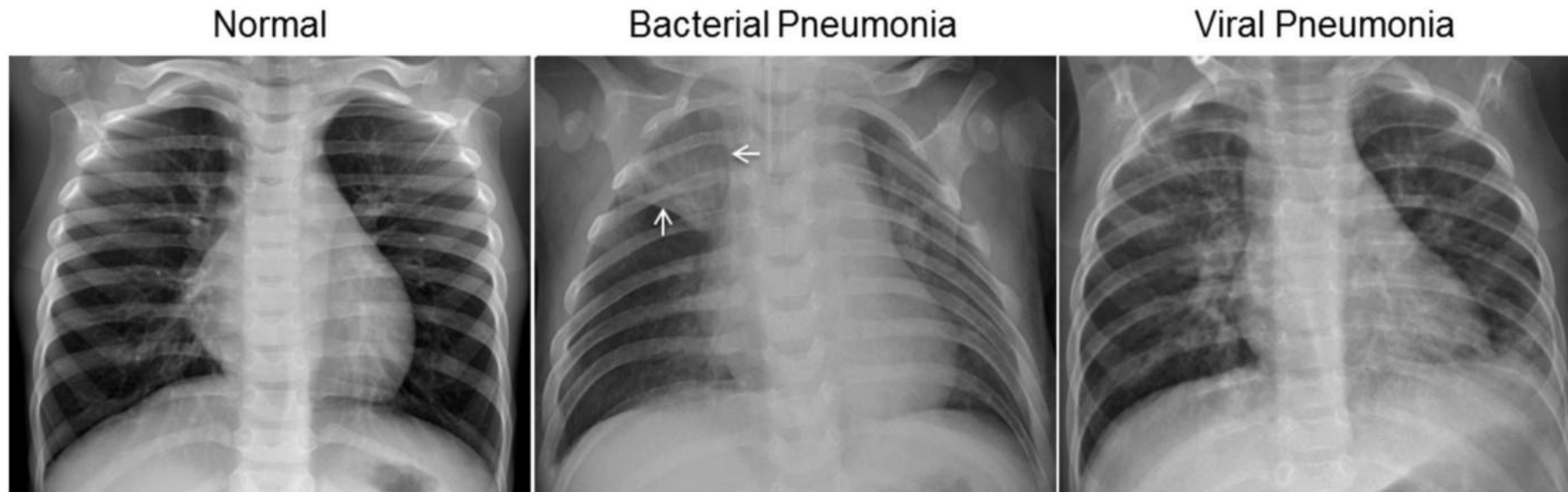




# Describing photos

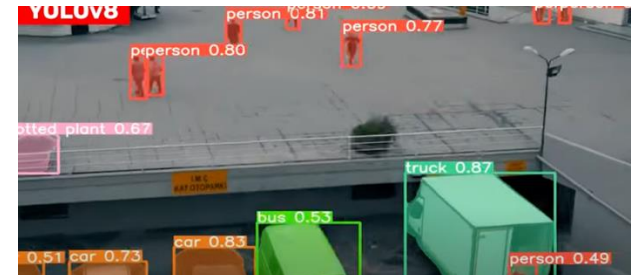
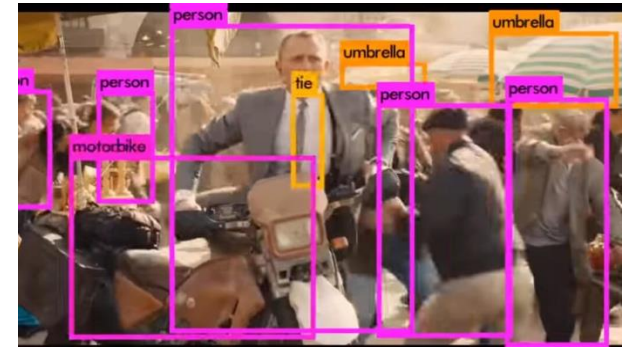


# Detecting Pneumonia with Deep Learning



# Video Analysis in Real Time

- <https://www.youtube.com/watch?v=3FvUOaxWnmg>
- <https://www.youtube.com/watch?v=VOC3huqHrss>
- <https://www.youtube.com/watch?v=QgF5PHDCwHw>



# NLP, question answering



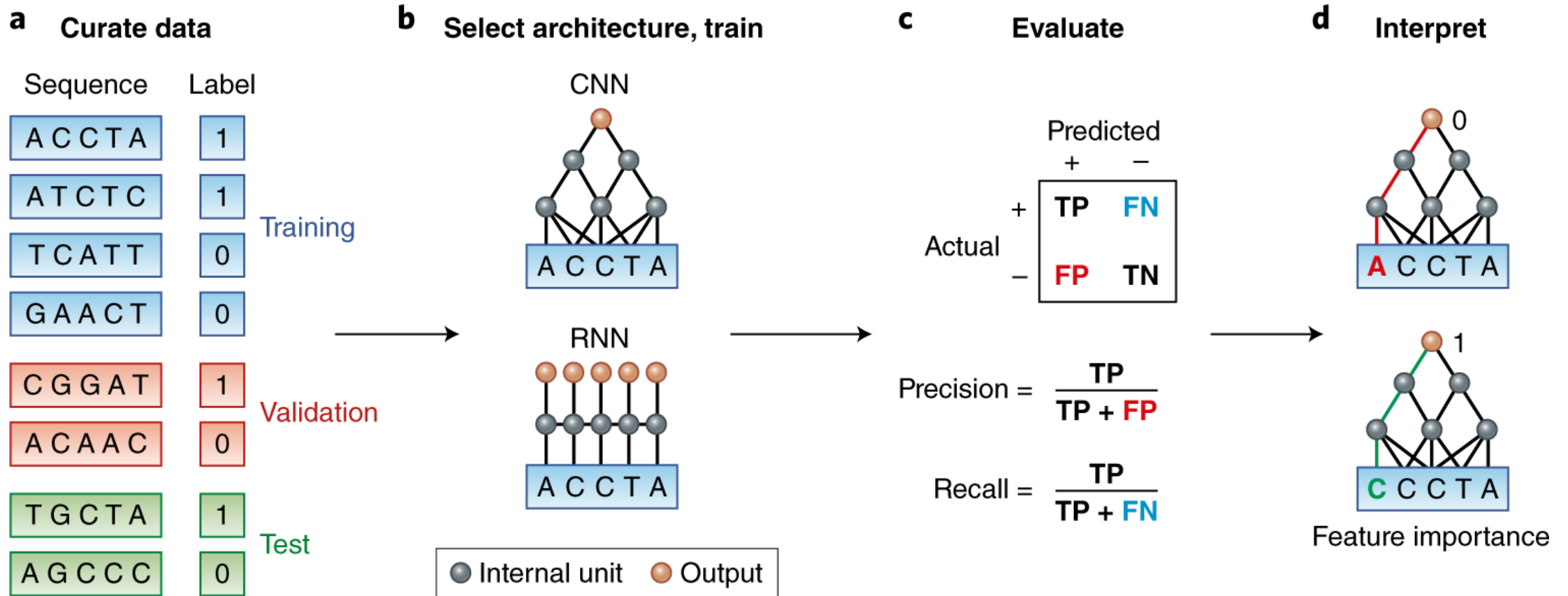
[Google Inbox Smart Reply]



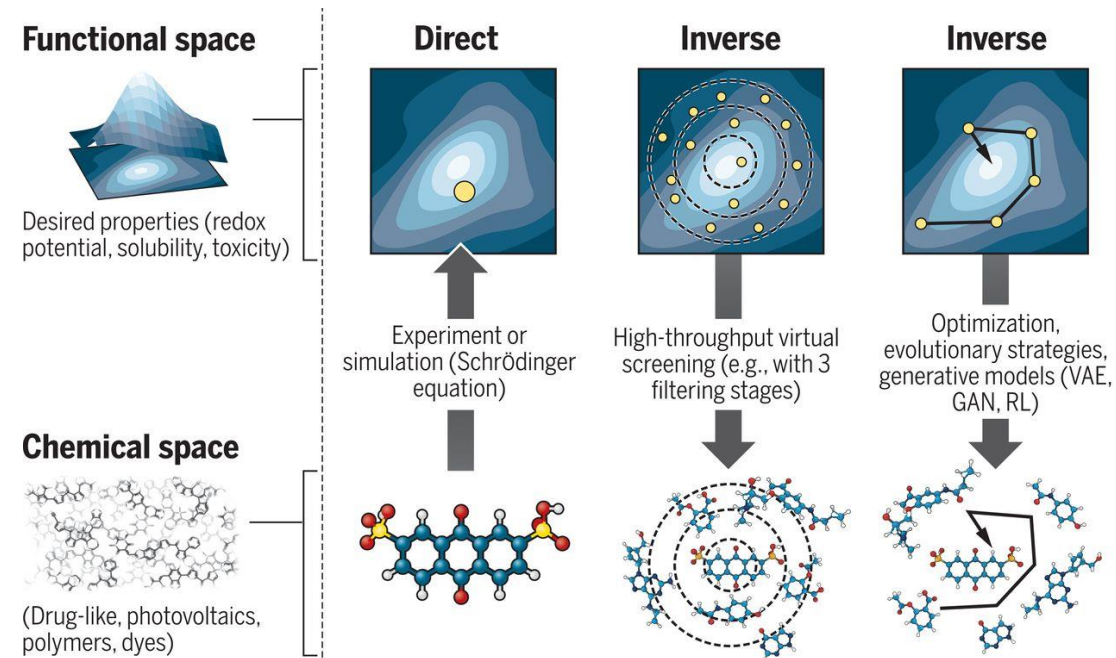
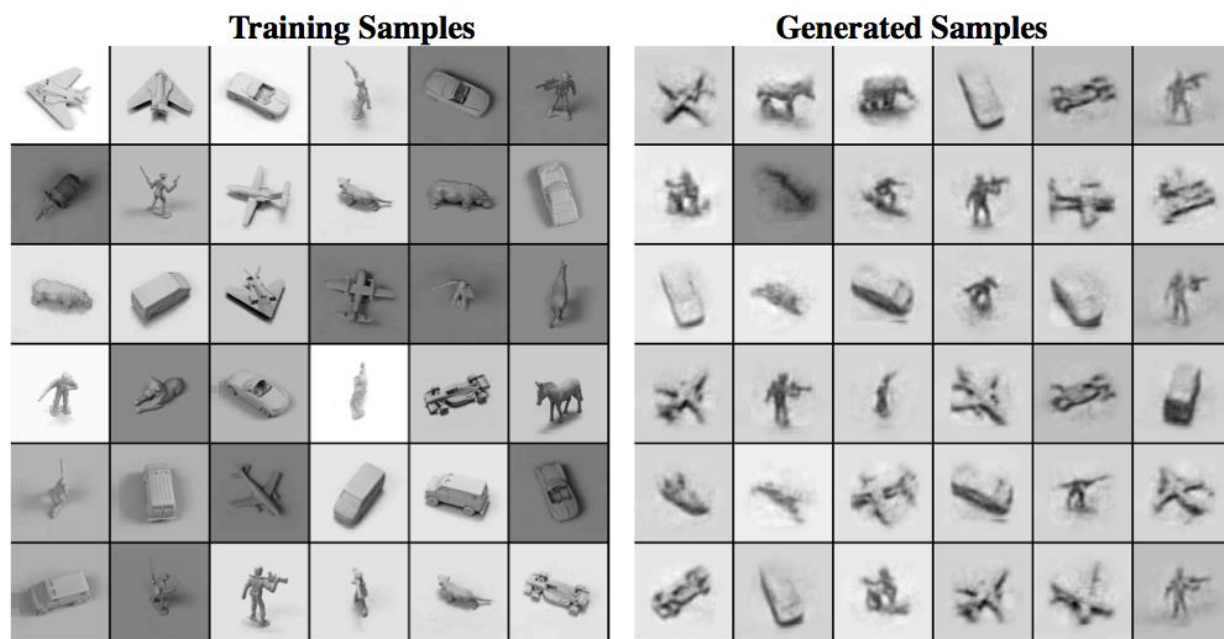
[Amazon Echo / Alexa]



# A primer on deep learning in genomics



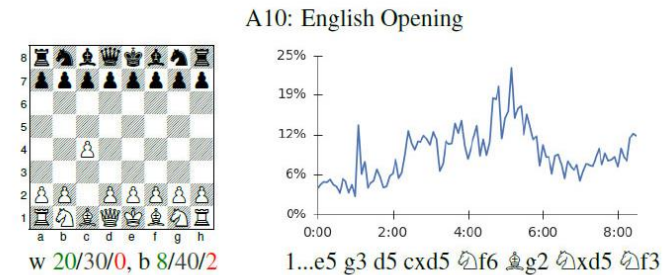
# Generative models



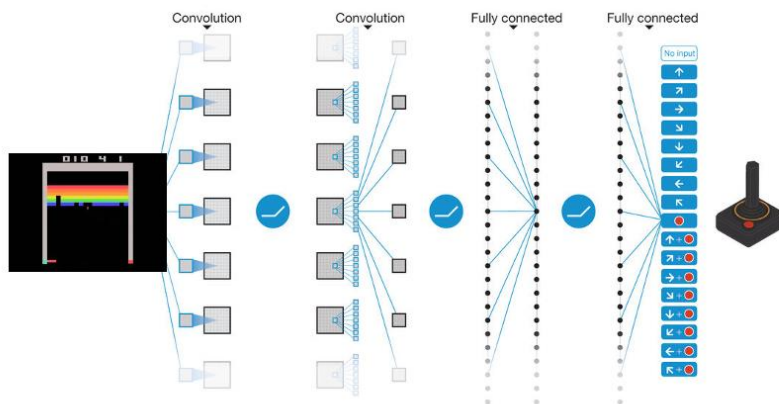
Sanchez-Lengeling, Benjamin and Aspuru-Guzik, Alán.

Inverse molecular design using machine learning: Generative models for matter engineering, Science, 2018

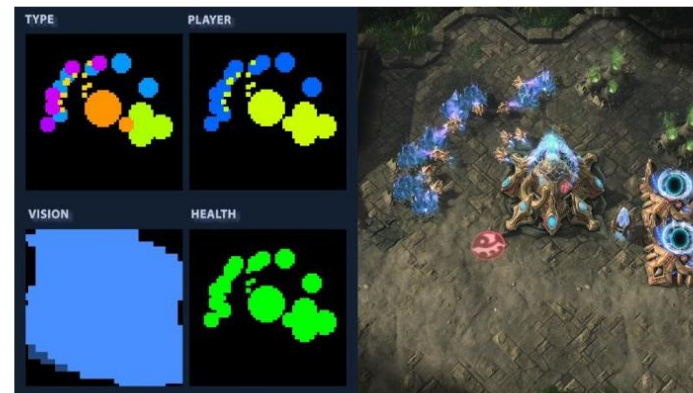
# Reinforcement Learning in Games



[Deepmind AlphaGo / Zero 2017]



[Atari Games - DeepMind 2016]

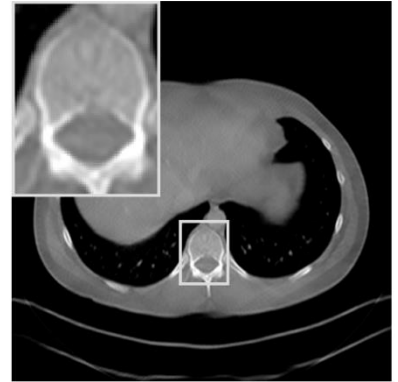


[Starcraft 2 for AI research]

# Impact of Deep Learning on Mathematics

- **Inverse Problems**

- Image denoising (Burger, Schuler, Harmeling; 2012)
- Superresolution (Klatzer, Soukup, Kobler, Hammernik, Pock; 2017)
- Limited-angle tomography (Bubba, Kutyniok, Lassas, März, Samek, Siltanen, Srinivasan; 2018)
- Edge detection (Andrade-Loarca, Kutyniok, Öktem, Petersen; 2019)



- **Numerical Analysis of Partial Differential Equations**

- Schrödinger equation (Rupp, Tkatchenko, Müller, von Lilienfeld; 2012)
- Black-Scholes PDEs (Grohs, Hornung, Jentzen, von Wurstemberger; 2018)
- Parametric PDEs (Schwab, Zech; 2018)

$$\frac{d\Psi(x)}{dx} = f(x, \Psi)$$

- **Modelling**

- Learning equations from data (Sahoo, Lampert, Martius; 2018)

$$f(x_1, x_2) = \frac{\sin(\pi x_1)}{(x_2^2 + 1)}$$



# Basics in learning theory

# Classification and Regression

- There exists a couple of random variables  $(X, Y)$ 
  - We know  $\mathcal{X}$  and  $\mathcal{Y}$  such that  $\mathbb{P}(X \in \mathcal{X}) = \mathbb{P}(Y \in \mathcal{Y}) = 1$
  - We observe a realization  $(x, y)$  of  $(X, Y)$
  - We want to construct a function  $f: \mathcal{X} \rightarrow \mathcal{Y}$  to predict  $y$  from  $x$

- We choose a loss function  $\ell: \mathcal{Y} \times \mathcal{Y} \rightarrow [0, +\infty]$  (see next slides)

- We want to find  $f^*$  such that

$$f^* \in \operatorname{argmin}_f \mathbb{E}(\ell(f(X), Y))$$

- $\mathbb{E}(\ell(f(X), Y))$  is called the risk of  $f$

- Remark:  $\mathcal{X}$  is typically  $\mathbb{R}^d$ ,  $\mathcal{Y}$  is described in the next slides

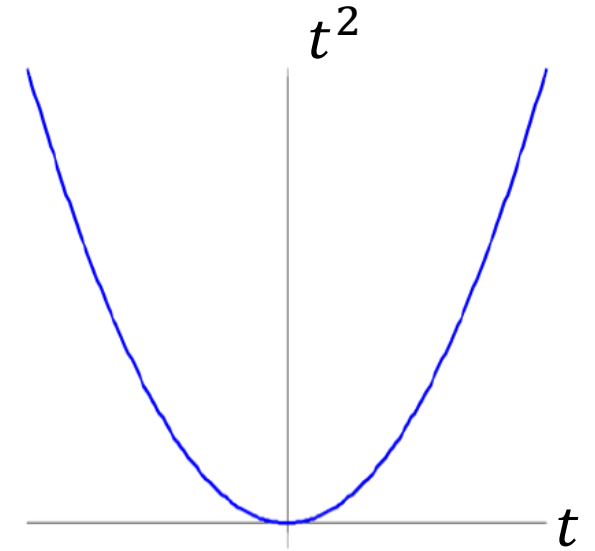
# Regression

- The set  $\mathcal{Y}$  is not-finite
  - Univariate case:  $\mathcal{Y} = \mathbb{R}$
  - Multivariate case:  $\mathcal{Y} = \mathbb{R}^K$
- In deep-learning, we construct  $K$  functions
$$f_k: \mathcal{X} \rightarrow \mathbb{R}, \quad k = 1, \dots, K$$

such that the output is the vector

$$\hat{y} = f(x) = (f_1(x), \dots, f_K(x))$$

- There are many loss functions, including the famous Euclidean loss function
$$\ell(\hat{y}, y) = \|\hat{y} - y\|^2$$



# Classification

- The set  $\mathcal{Y}$  is finite
  - Binary case:  $\mathcal{Y} = \{-1, +1\}$
  - Multiclass:  $\mathcal{Y} = \{1, \dots, K\}, K > 2$
- In deep-learning, for  $K \geq 2$ , we construct  $K$  functions

$$f_k: \mathcal{X} \rightarrow [0,1], \quad k = 1, \dots, K$$

such that  $f(x) = (f_1(x), \dots, f_K(x))$  and  $\sum_{k=1}^K f_k(x) = 1$

- Output:
  - The « raw » output is  $\hat{y} \in \operatorname{argmin}_{k=1, \dots, K} f_k(x)$
  - The « soft » output is given by  $\hat{y} = \operatorname{softmax}(f(x))$
- There are many loss functions, including the famous 0-1 loss function

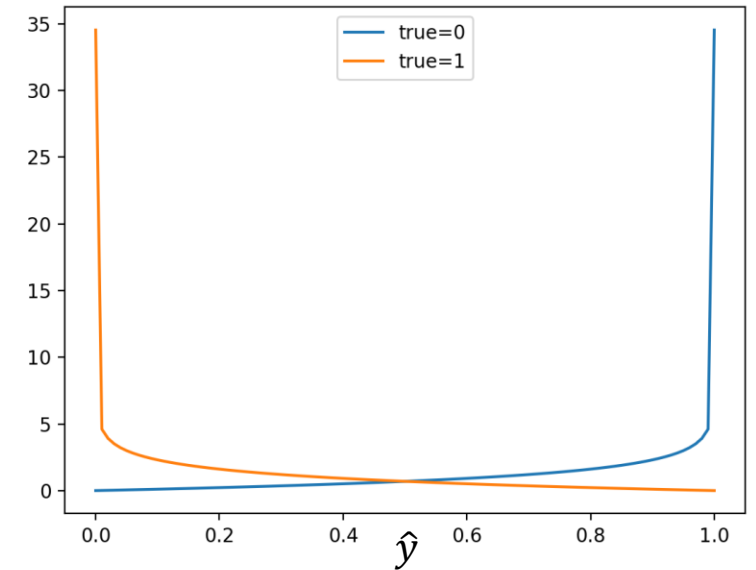
$$\ell(\hat{y}, y) = 1_{\{\hat{y} \neq y\}} = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise} \end{cases}$$

# Usual Loss Function for Classification

- Cross-entropy for binary outputs  $\hat{y}$  and  $y$ 
  - To measure a gap between two **binary** values  $\hat{y}, y \in \{0,1\}$

$$\ell(\hat{y}, y) = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$

- Interpretation:
  - If  $\hat{y} = y \in \{0,1\}$ , then  $\ell(\hat{y}, y) = 0$
  - If  $\hat{y} \neq y \in \{0,1\}$ , then  $\ell(\hat{y}, y) = +\infty$  (in practice, it is a large value)



Binary cross-entropy  
(continuous function of  $\hat{y}$ )

- Cross-entropy for discrete outputs
  - To measure a gap between two **discrete** values  $\hat{y}, y \in \{1, \dots, K\}$
  - One-hot encoding: use a binary word of size  $K$  such that all bits are '0' except one '1'
  - Example:  $\{1,2,3,4\} \Rightarrow \{1000,0100,0010,0001\}$
  - Abuse of notation: the encoded vector of  $y$  is the vector  $(y[1], y[2], \dots, y[K])$

$$\ell(\hat{y}, y) = - \sum_{j=1}^K (\hat{y}[j] \ln y[j] + (1 - \hat{y}[j]) \ln(1 - y[j]))$$

# Bayes Risk and Bayes Decision Rule

Theorem: Under certain weak assumptions on the distribution of  $(X, Y)$ , the function

$$f^b(x) = \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}(\ell(y, Y) | X = x)$$

minimizes the risk:

$$\forall f: \mathcal{X} \rightarrow \mathcal{Y}, \quad \mathbb{E}(\ell(f^b(X), Y)) \leq \mathbb{E}(\ell(f(X), Y))$$

The function  $f^b$  is called the Bayesian decision rule.

- Comments:
  - We may have  $\mathbb{E}(\ell(f^b(X), Y)) > 0$  (not all the decisions are perfect, i.e.,  $\mathbb{P}(Y = a | X = x) < 1$  for all  $a$  and some  $x$ )
  - In practice, we can not calculate the Bayesian decision rule
    - We do not know the distribution of  $(X, Y)$
    - We must represent  $f^b$  with a computer

# Example of Bayes decision rule

- Classification: binary case  $\mathcal{Y} = \{-1, +1\}$  with 0-1 loss function
- Given the observation  $x$ , for any value  $y$ , we get

$$\begin{aligned}\mathbb{E}(\ell(y, Y)|X = x) &= \ell(y, -1)\mathbb{P}(Y = -1|X = x) + \ell(y, 1)\mathbb{P}(Y = 1|X = x) \\ &= \begin{cases} \mathbb{P}(Y = -1|X = x) & \text{if } y = 1 \\ \mathbb{P}(Y = 1|X = x) & \text{if } y = -1 \end{cases}\end{aligned}$$

- Hence,  $f^b(x) = \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}(\ell(y, Y)|X = x)$  is rewritten as

$$f^b(x) = \begin{cases} -1 & \text{if } \mathbb{P}(Y = -1|X = x) \geq \mathbb{P}(Y = 1|X = x) \\ 1 & \text{if } \mathbb{P}(Y = -1|X = x) < \mathbb{P}(Y = 1|X = x) \end{cases}$$

- This is the famous MAP (Maximum A posteriori) rule

# Learning in Classification and Regression

- We observe some samples  $\mathcal{D} = (x_i, y_i)_{i=1, \dots, N}$  following the distribution of  $(X, Y)$ 
  - We are assuming that the samples are independent (i.i.d. assumption)
- We are considering a family  $\mathcal{F}$  of functions  $f_\theta$  parameterized by  $\theta$ 
  - The parameter  $\theta$  generally belongs to an Euclidean space  $\Theta$  (e.g.,  $\theta \in \mathbb{R}^n$ )
- We are expecting to solve

$$\theta^* \in \operatorname{argmin}_{\theta \in \Theta} \mathbb{E}(\ell(f_\theta(X), Y))$$

but, in practice, a naïve approach consists in minimizing the empirical risk

$$\hat{\theta} = \hat{\theta}(\mathcal{D}) \in \operatorname{argmin}_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^N \ell(f_\theta(x_i), y_i)$$



# Main issues

- Optimization
  - The optimization problem can be hard to solve (strong nonlinearity for example)
  - The number of samples is huge ( $N$  is huge)
  - The dimension of the parameters is huge ( $\theta$  has many components)
- Generalization error: is  $\hat{\theta} = \hat{\theta}(\mathcal{D})$  close to  $\theta^*$ ?
  - Possible meaning:  $\mathbb{E}(\ell(f_{\hat{\theta}}(X), Y)) \approx \mathbb{E}(\ell(f_{\theta^*}(X), Y))$
  - An other possible meaning:  $\sup_{\theta \in \Theta} \left| \mathbb{E}(\ell(f_{\theta}(X), Y)) - \frac{1}{N} \sum_{i=1}^N \ell(f_{\theta}(x_i), y_i) \right|$  is bounded?
- Approximation error: is  $f_{\theta^*}$  close to  $f^b$ ?
  - Possible meaning:  $\mathbb{E}(\ell(f_{\theta^*}(X), Y)) \approx \mathbb{E}(\ell(f^b(X), Y))$

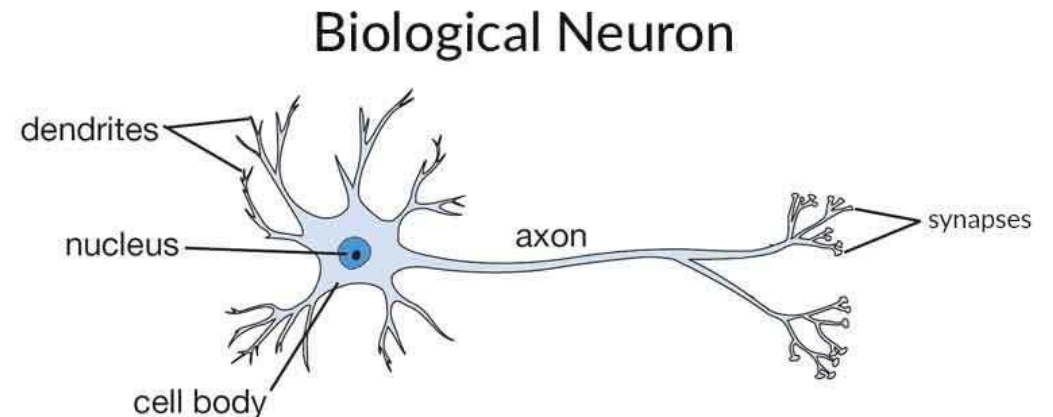
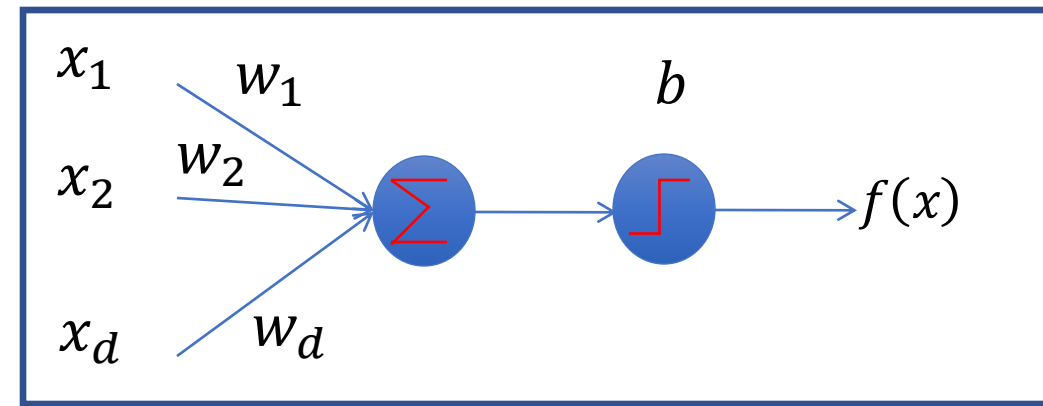
# Basics in deep neural networks

# Perceptron (Neuron Metaphor)

- The perceptron (Rosenblatt, 1957) is:

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i + b = w^T x + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- However inputs are real values and the weights can be different
- This model was originally motivated by biology, with  $w_i$  being synaptic weights,  $x_i$  the input firing rates and  $f(x)$  the output firing rate.
- This is a cartoonish biological model.



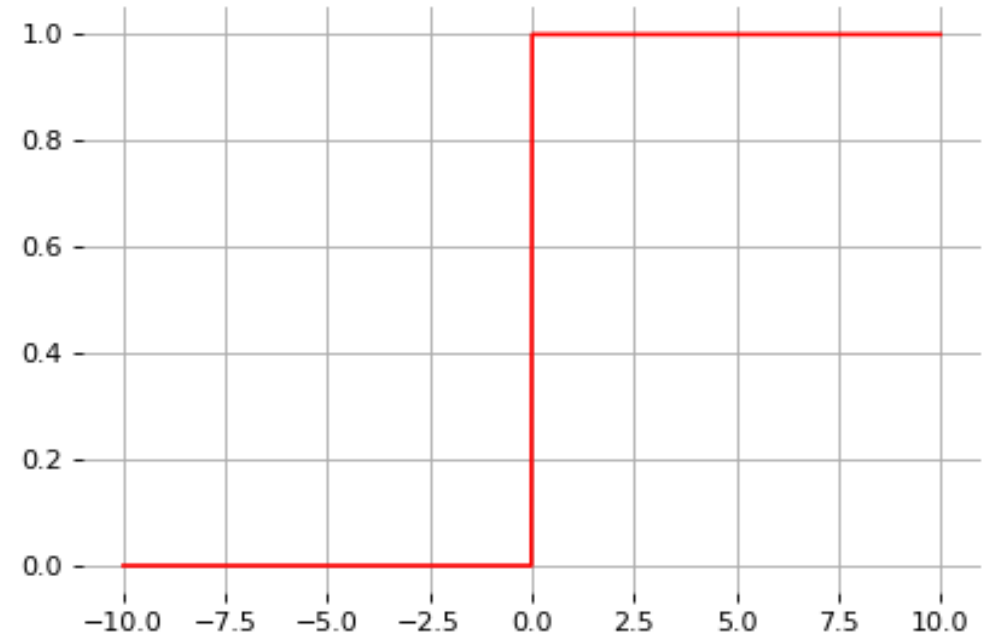
# Heavyside Activation function

- Let us define the activation function:

$$\sigma(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Then, the perceptron classification rule can be rewritten as

$$f(x) = \sigma(w^T x + b)$$

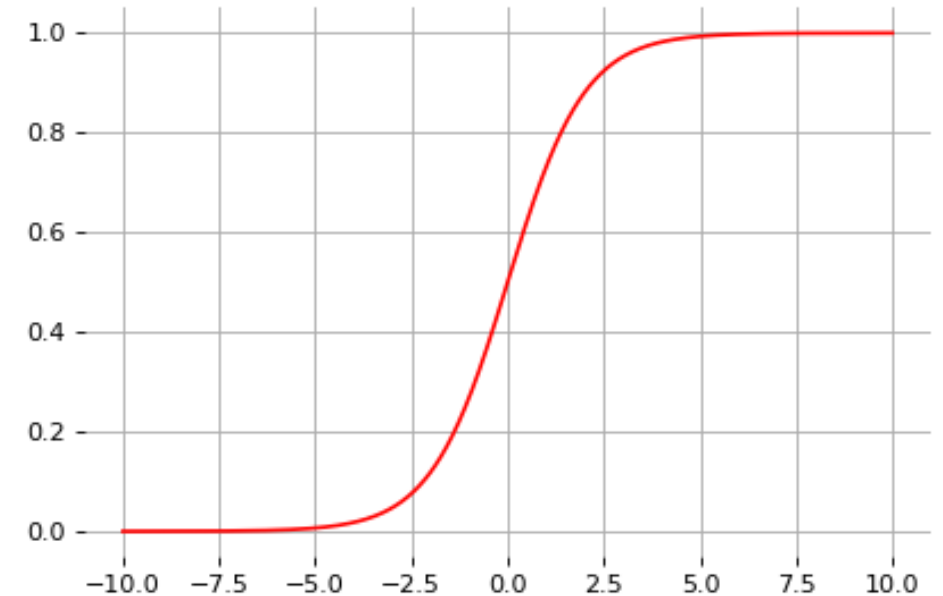


# Sigmoid Activation Function

- Note that the sigmoid function

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

looks like a soft heavyside:

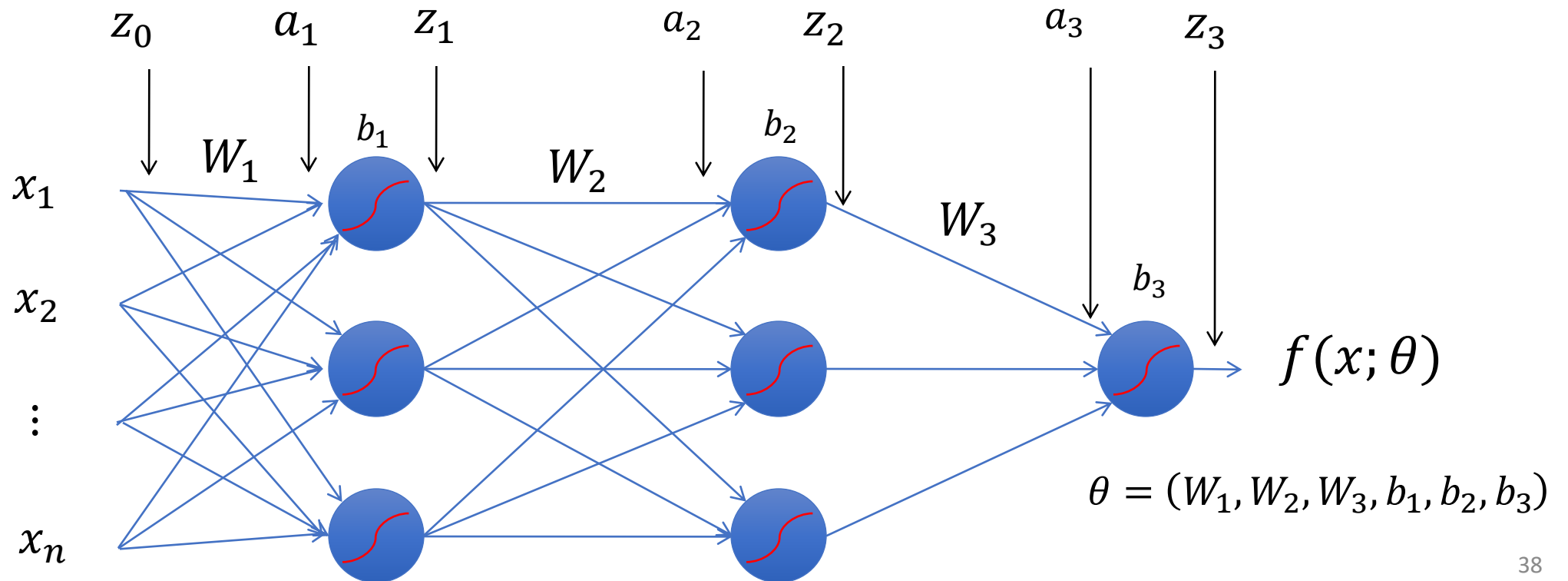


- Therefore, the overall model  $f(x) = \sigma(w^T x + b)$  is very similar to the perceptron but it is differentiable with respect to  $w$  and  $b$

# Feed-Forward Networks and All Its Variables

- Predictions are fed forward through the network to classify:

- $z_0 = x$
- $a_k = W_k z_{k-1} + b_k, k = 1, 2, 3$
- $z_k = \sigma(a_k), k = 1, 2, 3$
- $f(x) = f(x; \theta) = z_3$



# Terminology

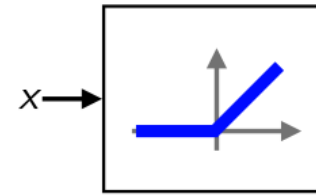
- A feedforward net is a nonlinear function composed of repeated affine transformation followed by a nonlinear action:

$$\begin{cases} f_0(x) = x \\ f_h(x) = \sigma_h(W_h f_{h-1}(x) + b_h), & 1 \leq h \leq H - 1 \\ \hat{y} = f(x) = f_H(x) = \sigma_H(W_H f_{H-1}(x) + b_H). \end{cases}$$

- We say that the networks contains  $H$  layers and  $H - 1$  hidden layers
- The size of the layers,  $n_0, n_1, \dots, n_H$ , corresponds to the number of neurons
- The weights of layer  $h$  are given by matrix  $W_h \in \mathbb{R}^{n_h \times n_{h-1}}$
- The biases of layer  $h$  are given by  $b_h \in \mathbb{R}^{n_h}$
- The nonlinear activation function of layer  $h$  is  $\sigma_h(\quad)$

# Element-wise activation functions

- Each neuron/unit is followed by a dedicated activation function
- Historically the sigmoid and tanh have been the most popular
- Many other activation functions available



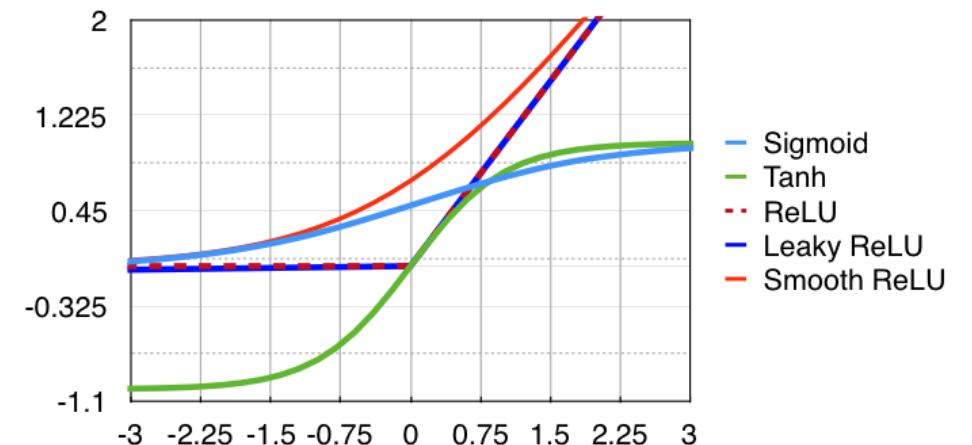
$$y = \frac{1}{1 + e^{-x}} \text{ sigmoid}$$

$$y = \tanh(x) \text{ hyperb. tan}$$

$$y = \max\{0, x\} \text{ ReLU}$$

$$y = \log(1 + e^x) \text{ Soft ReLU}$$

$$y = \epsilon x + (1 - \epsilon) \max\{0, x\} \text{ Leaky ReLU}$$





# One hidden layer network

- We can already use this to perform logistic regression.
- For model  $\hat{y}(x; w, b) = \Pr(Y = 1|x) = \sigma(w^T x + b)$ , find  $(w, b)$  that maximizes the likelihood of the dataset  $D = \{(x_i, y_i)\}$ :

$$\operatorname{argmax}_{w, b} \Pr(D|w, b) =$$

$$\operatorname{argmin}_{w, b} \sum_{(x_i, y_i) \in D} -y_i \log \sigma(w^T x_i + b) - (1 - y_i) \log(1 - \sigma(w^T x_i + b))$$

- More general notations, we minimize

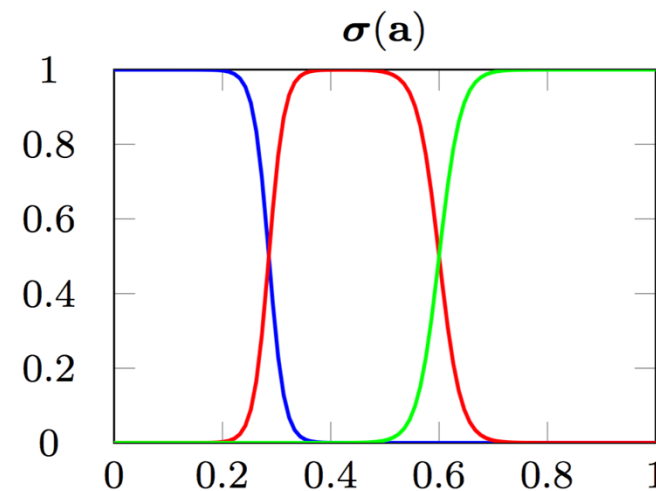
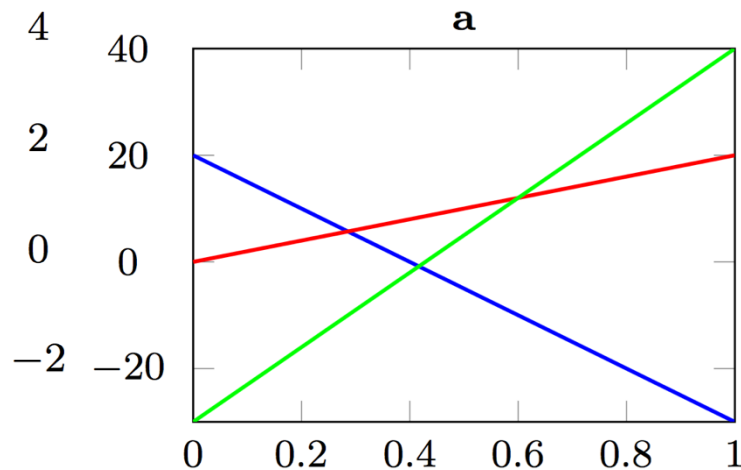
$$\mathcal{L}(W, b) = \sum_{(x_i, y_i) \in D} \ell(y_i, \hat{y}_i(x_i; w, b))$$

# Softmax for classification

- How to transform the vector output of a neural network into a probability distribution?
- The softmax function,  $\sigma: \mathbb{R}^K \rightarrow \mathbb{R}^K$ , generalizes the sigmoid function and yields a vector of  $K$  values in  $[0,1]$  by exponentiating and then normalizing

$$\sigma(a) = \text{softmax}(a) = \frac{1}{\sum_{j=1}^K e^{a_j}} (e^{a_1}, \dots, e^{a_K})$$

- As activation values increase softmax tends to focus on the maximum



# Pytorch implementation (one-hidden layer)

This implementation defines the model as a custom Module subclass.

```
class TwoLayerNet(torch.nn.Module):
```

Inheriting from `nn.Module`

```
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)
```

In the constructor, we instantiate two `nn.Linear` modules and assign them as member variables.

```
    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred
```

In the forward function we accept a Tensor of input data and we must return a Tensor of output data.

We can use Modules defined in the constructor as well as arbitrary operators on Tensors.

```
twolayer_net = TwoLayerNet(2, 20, 1)
```

Instantiates the class defined above

# IT Tools

# Useful Information Technology Tools

- Python and Jupyter notebook

<https://www.anaconda.com/download/>



- Pytorch

<https://pytorch.org/>



- Access to a GPU

# Google Colab

PYTORCH

colab



- Google Colab is a free Jupyter notebook environment that supports both CPU and GPU usage for free.
- It is extremely convenient to share notebooks and also it is good for self-study.
- Some key features
  - It is a Linux-like directory system
  - The current directory is « /content »
  - You can upload data files
  - Python is already installed; missing libraries can be installed

# Deep Learning Frameworks

- TensorFlow

- TensorFlow was created by Google and is one of the most popular deep learning frameworks.



- Keras

- Keras is another open source deep learning framework that is widely used.
- Keras is remarkably simple to use



- PyTorch (used in this course)

- Open source, primarily developed by Facebook, and is known for its simplicity, flexibility, and customizability



- Microsoft Cognitive Toolkit

- Microsoft Cognitive Toolkit is an open source deep learning framework to train deep learning models.



# Why do we need Deep Learning frameworks?

- Speed:
  - Fast GPU/CPU implementation of matrix multiplication, convolutions and backpropagation
- Automatic differentiations:
  - Pre-implementation of the most common functions and it's gradients.
- Reuse:
  - Easy to reuse other people's models
- Less error prone:
  - The more code you write yourself, the more errors



# Why Pytorch?

- Python API
- Can use CPU, GPU
- Supports common platforms:
  - Windows, Mac, Linux
- Pytorch is a thin framework which lets you work closely with programming the neural network
- Focus on the machine learn part not the framework itself
- Pythonic control flow
  - Flexible
  - Cleaner and more intuitive code
  - Easy to debug
- Python debugger
  - With Pytorch we can use the python debugger

# Pytorch API (just the main packages)

Package	Description
torch	The torch package contains data structures for multi-dimensional tensors and mathematical operations over these are defined.
torch.Tensor	A torch.tensor is a multi-dimensional matrix containing elements of a single data type.
torch.nn	A package that contains modules and extensible classes for building neural networks
torch.autograd	A package that provides classes and functions implementing automatic differentiation of arbitrary scalar valued functions
torch.nn.functional	A functional interface that contains typical operations used for building neural networks like loss functions, activation functions, and convolution operations
torch.optim	A package that implements various optimization algorithms like SGD and Adam.
torchvision	The package that provides popular datasets, model architectures, and common image transformations for computer vision.

# torch.Tensor class

- Pytorch's tensors are similar to NumPy's ndarrays

```
import torch
x = torch.tensor([5.5, 3])
print(x)
Tensor([5.5000, 3.0000])
```

```
x = torch.rand(5, 3)
print(x)
tensor([[0.6915, 0.2723, 0.4345],
        [0.2429, 0.4125, 0.5318],
        [0.1012, 0.3451, 0.3916],
        [0.8321, 0.7369, 0.2084],
        [0.3251, 0.5779, 0.3637]])
```

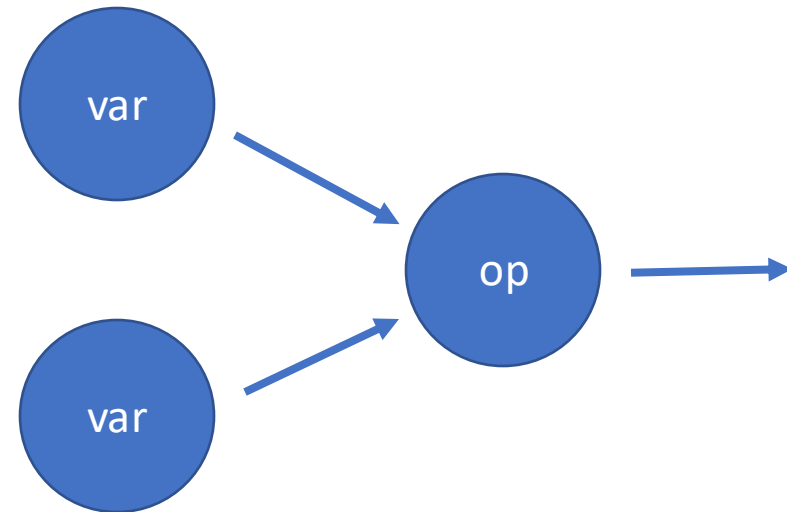
# Torch.tensor attributes

Attribute	Data type	Description
data	array_like	list, tuple, NumPy ndarray, scalar
dtype	torch.dtype	The tensor's data type
requires_grad	bool	Should autograd record operation
device	torch.device	Allocated on CPU or CUDA (GPU)

```
torch.tensor(data=[1,2,3], dtype=torch.float32, device='cpu', requires_grad=False)
```

# Computational graph

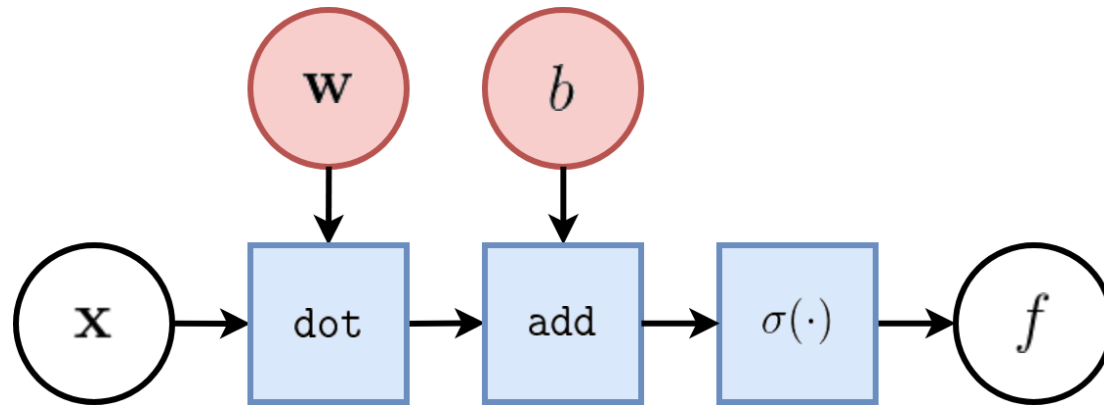
- Computational graphs
  - Help break down large computations into smaller ones by visualizing them as a graph of smaller computations,
  - Help in calculating derivatives of functions,
  - Modeled as Directed Acyclic Graph (DAG)
  - Nodes
    - Variables (tensors in Pytorch)
    - Mathematical operations
  - Edges
    - Feeding input



- Example:
  - $g(x) = wx + b$
  - $f(x) = \max(wx + b, 0) = \sigma(wx + b)$
  - $\ln(x) = 2 \operatorname{arctanh}\left(\frac{x-1}{x+1}\right) = 2\left(\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \dots\right)$

# Tensor Operations

- In terms of tensor operations, the computational graph of  $f$  can be represented as:

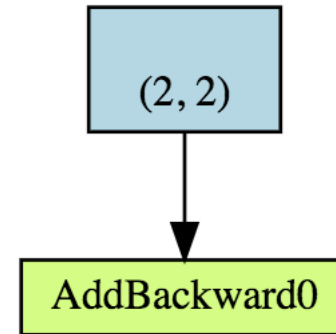


- Where
  - White nodes correspond to inputs and outputs;
  - Red nodes correspond to model parameters;
  - Blue nodes correspond to intermediate operations, which themselves produce intermediate output values (not represented).
  - This unit is the core component all neural networks!

# Example in Pytorch

```
import torch
import torchviz

x = torch.ones(2, 2)
x.requires_grad_(True)
y = x + 2
torchviz.make_dot(y)
```



# Conclusion



# Conclusion

- A vast number of applications
- A vast number of topics and architectures
- Deep Learning is based on some main concepts
- What are these concepts?
- It is crucial to be able to implement these concepts, for example in Python with a dedicated framework
- Deep learning is essentially based on deep neural networks
- Gradient descent is a key step for training deep neural networks
- Do not underestimate the study of the generalization error