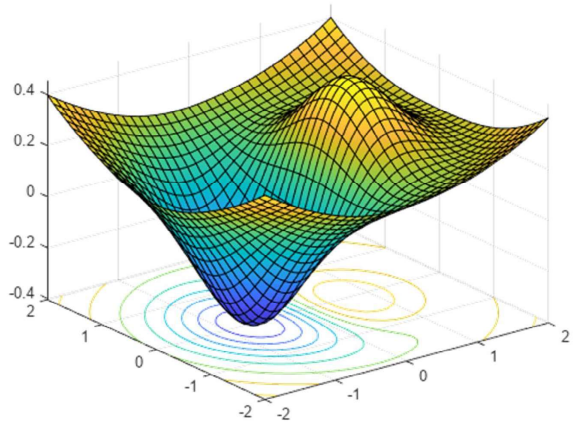
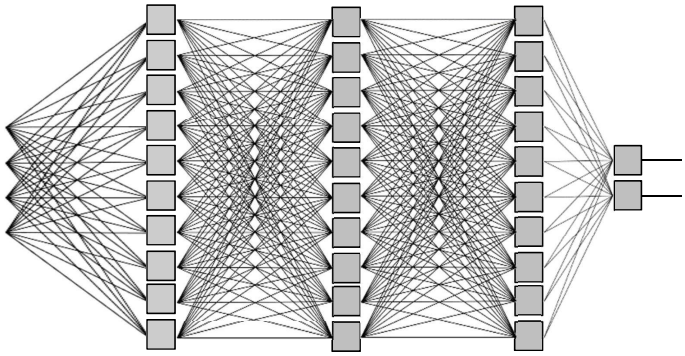
Artificial Intelligence
Machine Learning
Deep Learning

Antoine Deleforge





Organisation du module

- **Partie IA** (A. Deleforge):
  - 6 cours intégrés d'1h45:
    - 14/03am, 14/03pm, 27/03am, 27/03pm, 28/03pm, 12/04am
  - 3 TP de 4h:
    - Scindés en groupes A et B, du 29/03 jusqu'au 17/05

- **Partie Robotique** (L. Cuvillon):
  - 2 cours intégrés d'1h45:
    - 21/03am, 22/03am
  - 5 TP de 4h:
    - Scindés en groupes A et B, du 29/03 jusqu'au 17/05

Organisation du module

- **Evaluation:**
  - Partie IA: Examen QCM + Compte rendus de TP
  - Partie Robotique: contrôle continu en TP
- **Prérequis:**
  - Programmation python et orientée objet
  - Scalaires, vecteurs, matrices: $x, \boldsymbol{x}, \mathbf{X}$
  - Probas et statistiques: $p(x), p(x,y), p(x|y), \mathbb{E}, \mathrm{var}, \mathcal{N}$
  - Calcul différentiel: $\frac{\partial f}{\partial x}, \nabla f$
  - Optimisation

  } *Il y aura des rappels*
- Cette unité d'enseignement est nouvelle à TPS!
  - ➡ Retours bienvenus et appréciés
  - ➡ Soyez bienveillants et pro-actifs

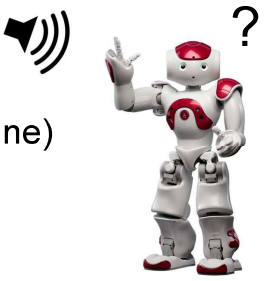
Sources

- **Ian Goodfellow**, **Yoshua Bengio**, **Aaron Courville**. *Deep Learning.* https://www.deeplearningbook.org/
- **Hugo Larochelle**, *Online Course on Neural Network.* http://info.usherbrooke.ca/hlarochelle/neural_networks/
- **Emmanuel Vincent**, *Neural Network* course. Master TAL, Univ. de Lorraine.
- **Paul Magron**, *Neural Network* labs. Master TAL, Université de Lorraine.
- **Antoine Liutkus**, cours *Deep Learning et réseaux de neurones, les fondamentaux.* Inria Sofia.
- https://towardsdatascience.com/
- https://cs230.stanford.edu/blog/pytorch/

Mon Parcours

2007-2010:
- Ecole d'ingénieur ENSIMAG (INPG)
- Double diplôme: master recherche graphisme, vision, robotique

2010-2013:
- Thèse à l'Inria de Grenoble, équipe PERCEPTION (ajd: RobotLearn)
- Equipe en vision + machine learning / thèse en audio



2014-2015:
- Post-doc à l'université Friedrich-Alexander d'Erlangen (Allemagne)
- Projet européen EARS sur l'Audition Robotique.

2016-présent:
- Chargé de recherche Inria
- Equipe PANAMA (Rennes) puis MULTISPEECH (Nancy) puis (bientôt) MACARON (Strasbourg)
- Ré-orientation vers l'acoustique des salles

Et vous?

Unité d'Enseignement: IA et Robotique

Concept du Cours

Constats:
- Essor des frameworks opensource depuis 2016 (TensorFlow, Pytorch,…)
  $\Rightarrow$ Coder un algo de Deep Learning est devenu très accessible, en quelques tutos

- Un nouveau papier IA sort toutes les heures sur ArXiv, une nouvelle "révolution" toutes les semaines

Se focaliser sur les **concepts fondamentaux** pour:
- Savoir **trier** le bon grain de l'ivraie
- Assimiler rapidement de **nouvelles** architectures et méthodes
- Identifier la meilleure approche pour un **cas d'usage**
- Acquérir des **bonnes pratiques**
- **Diagnostiquer** les problèmes

- Diapos en **anglais**

OUTLINE

I. Introduction
A.I., Machine Learning, Deep Learning: What, How, Why and When

II. Background
Tensors and Multivariate Calculus

III. Fitting a Model
Optimization techniques, Backpropagation, Gradient Descent, PyTorch

IV. Supervised Learning
Linear and Polynomial Regression, Over & Underfitting, Tips & Tricks

V. Unsupervised Learning
From K-means and PCA to Deep Clustering and Deep Generative Models

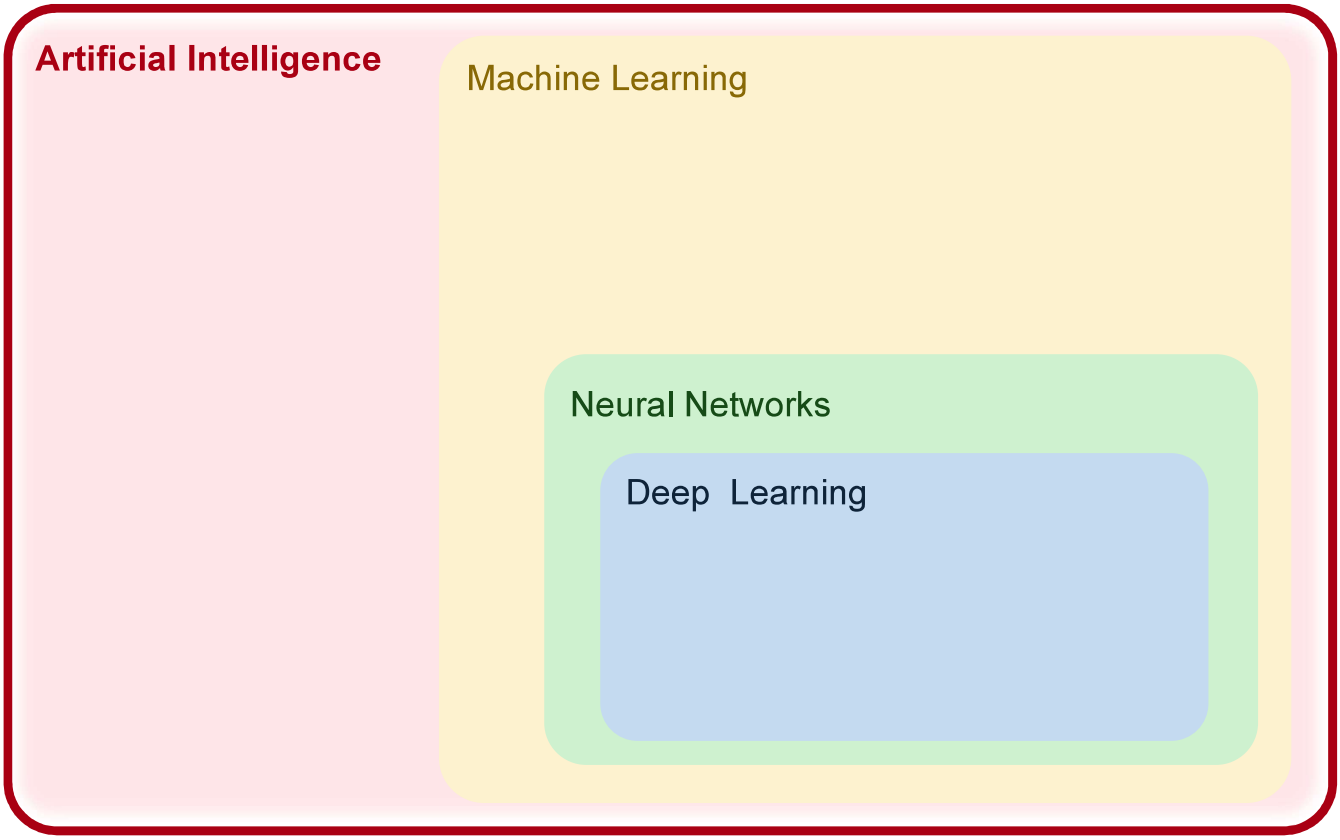
VI. Fantastic DNNs: How to choose them, how to train them
CNNs, U-Net, RNNs, Attention, Transformers

OUTLINE

I. Introduction

- Artificial Intelligence
- Machine Learning
- Neural Network and Deep Learning
- Applications

II. Background
III. Fitting a Model
IV. Supervised Learning
V. Unsupervised Learning
VI. Fantastic DNNs: How to choose them, how to train them
VII. Machine Learning in Robot Audition



What is Intelligence?

- A difficult question, no consensus today

  "A Collection of Definitions of Intelligence", *Shane Legg, Marcus Hutter, 2007*. Frontiers in Artificial Intelligence and applications.

  ➡ 70 definitions!

- Ex. of **dictionary** definition

  - "The ability to use memory, knowledge, experience, understanding, reasoning, imagination and judgement in order to solve problems and adapt to new situations." *AllWords Dictionary, 2006*

- Ex. of **psychologist** definitions
  - "Intelligence is what is measured by intelligence tests." *E. Boring, 1923*
  - "**Fluid** intelligence is your ability to process new information, learn, and solve problems. **Crystallized** intelligence is your stored knowledge, accumulated over the years." *R. Cattell, 1963*
  - "Intelligence is not a single, unitary ability, but rather a composite of several functions. The term denotes that combination of abilities required for survival and advancement within a particular culture." *A. Anastasi, 1992*

What is Intelligence?

- Ex. of **AI researchers** definitions
  - "Intelligence is the ability to use optimally limited resources –including time– to achieve goals." *R. Kurzweil, 2000*
  - "Intelligence measures an agent's ability to achieve goals in a wide range of environments." *Shane Legg, Marcus Hutter, 2007*

Artificial Intelligence: an ill-defined term

- Originates in story telling, science fiction

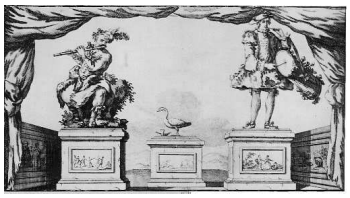
Talos protecting Europa in Cretes, Greek Mythology, c. 400 BC

Theater play R.U.R, Karel Čapek, 1920

HAL9000, 2001 Space Odyssee, Stanley Kubrik, 1968

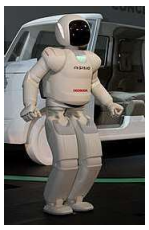
M3GAN, Gerard Johnstone, 2022

- Strongly embedded in **collective imagination**
- A **relative** notion



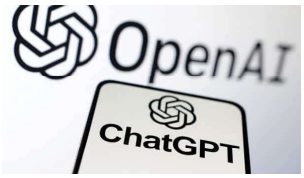
Vaucanson, 1737

Kasparov – Deep Blue 1997



ASIMO 2000

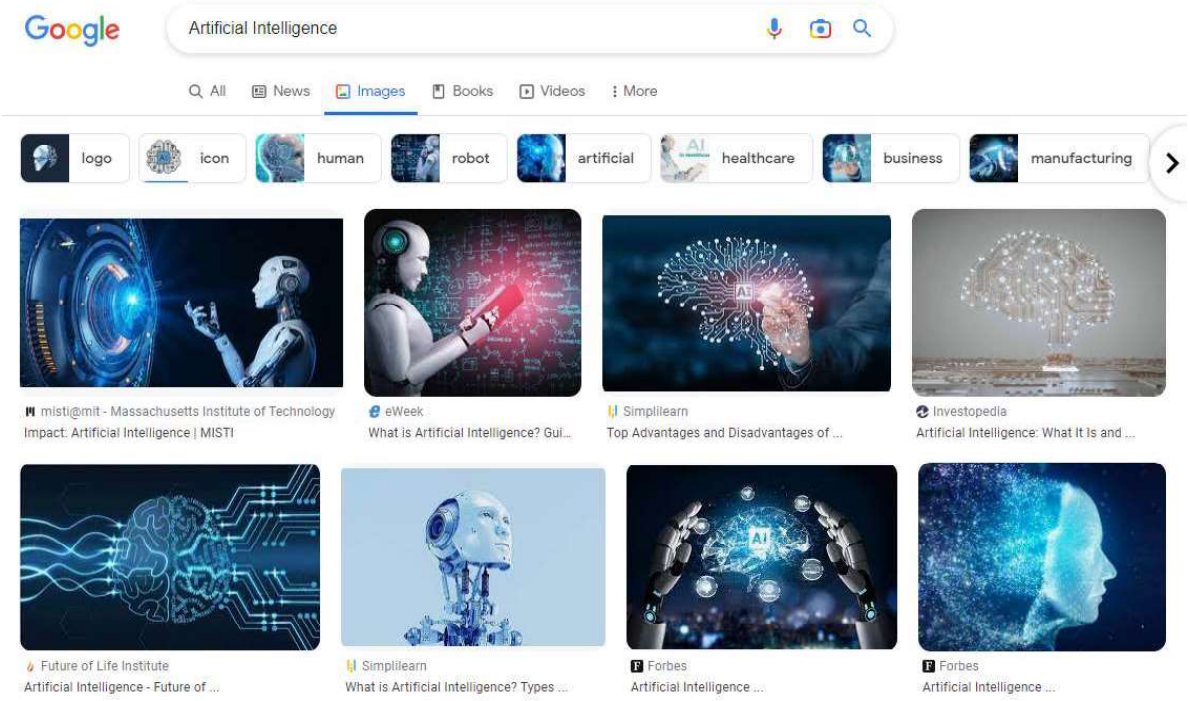
Lee Sedol – AlphaGo 2016



ChatGPT 2022

Artificial Intelligence: an ill-defined term

- A common confusion: **A.I. == ROBOTICS**



Artificial Intelligence: an ill-defined term

- A common confusion: **A.I. == ROBOTICS**
- Human vs. Machine intelligence:
  - Highly Physical Tasks + Big Compute = Machines 🤖
  - Highly Creative and Intellectual Tasks = Humans 🧠

Status in 2023

- Have you seen a robot tidying up your appartment?
- Have you seen a machine:
  - Win an art contest? (2022)
  - Self-learn to play and beat humans at arbitrary games? (2020)
  - Hold coherent extended conversations? (2022)
  - Rewrite Bohemian Rhapsody, but about a post-doc's life? (2020)

Artificial Intelligence: an ill-defined term

- A.I. is a « catch-all » word
- Rarely used in scientific publications

  Occurrence of terms in 12,900 ***conference paper titles*** *published at "Neural Information Processing Systems" since 2010* *[Source: Google Scholar]*

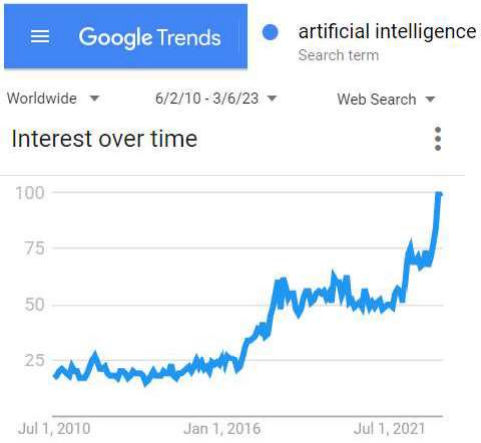
```
Learning: 3,310
Neural: 1,260
Deep: 864
Deep Learning: 291
Neural Network: 151
Machine Learning: 107
Artificial: 10
Intelligence: 3
Artificial Intelligence: 1
```

Source: DALL-E, openai.com

- Ex: Is signal processing / statistics / optimization A.I.?
- Understood by the general public = good for science communication
- Understood by decision makers = good for getting funding

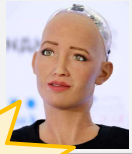
The Rise of A.I.

- Over the past 10 years (~2012), an **explosion** of the term A.I.
- Mostly in **media headlines** and for **marketing** purposes ($$)
- Beware of publicity stunts / Wizards of Oz



Engineer.ia: An Indian start-up that raised 30M€ by claiming to use "human-assisted AI tools" to develop mobile apps in record time. Following a lawsuit from employees, it was revealed that AI was mostly used as a marketing term (2019).

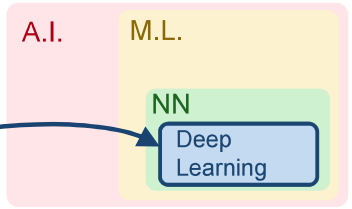
Pinscreen: same idea but for generating 3D avatars from photos (2018)



Sophia: The first humanoid robot "acquiring citizenship", from Saudi Arabia (2017)

So is A.I. just a scam?!

The Rise of A.I.



- **No.** A real **revolution** is taking shape
- The core driver is not « AI » but **Deep Learning**
- Nearly **all domains of science** have had some subfields which have been **profoundly transformed** by deep learning along the past 10 years
- Google Scholar

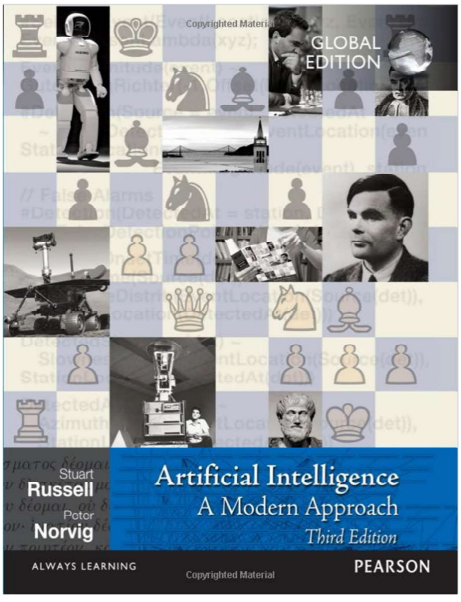
	Publication	h5-index	h5-median
1.	Nature	444	667
2.	The New England Journal of Medicine	432	780
3.	Science	401	614
4.	IEEE/CVF Conference on Computer Vision and Pattern Recognition	389	627
5.	The Lancet	354	635
6.	Advanced Materials	312	418
7.	Nature Communications	307	428
8.	Cell	300	505
9.	International Conference on Learning Representations	286	533
10.	Neural Information Processing Systems	278	436

M.L. conferences

- Likely, nearly all branches of industry, public institutions and professional sectors will soon be profoundly impacted as well

Applied A.I. *(What A.I. researchers & companies actually do!)*

- Solving Numerical Problems
  - Finding Uncontrained/Constrained Solutions
  - Adversarial Contexts (Game Theory)
- Representing Knowledge and Reasoning
  - Logic, Inference, Planification
  - Ontologies (Symbolic A.I.)
- Managing Uncertainty
  - Representations
  - Probabilistic Models
  - Decision Processes
- Language and Communication
  - Natural Language Processing
  - Speech Recognition, Translation
  - Audio-Visual Synthesis
  - Robotics
- Machine Learning



A.I. Philosophy

- A fascinating field
- Relatively **niche** compared to *applied A.I.*
- At the intersection of Philosophy, Futurology, Social Sciences, Psychology, Logic and (sometimes) Computer Science
- Some subtopics:
  - AI Safety / AI Risk / AI Alignment
  - AI Ethics / AI Bias / AI Fairness
  - Conciousness / Sentience / Free Will
  - Definitions of Intelligence

A.I. Philosophy

Different levels of A.I. are distinguished:

- **Specialized** A.I.
- Artificial **General** Intelligence (AGI)
- **Human-Level** Artificial Intelligence
- Artificial **Super-Intelligence** (ASI)

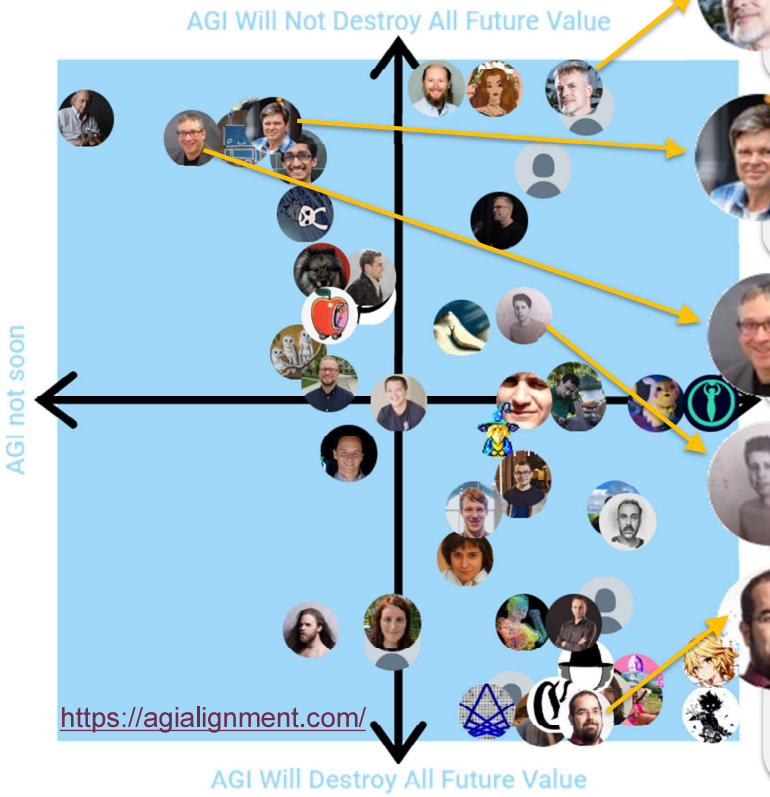
- Very clearly where we are **now**
- Can be cast into actual engineering questions and products

Does **not** exist (yet) !
No agreement on:
- Is it **achievable ?**
- **How** to achieve it ?
- **When** will we achieve it?
- **Should we** achieve it?

Most A.I. researchers agree that to safely deploy such systems, we need to **align them** (*A.I. alignment research*)

*They disagree on **how hard** it is, and on **how much time** we have to figure it out!*

A.I. Philosophy



- Jürgen Schmidhuber, DM Inst. for AI Research (Switz.), LSTM inventor
- *Annotated History of Modern AI and Deep Learning*

- Yann Lecun, Chief AI Scientist at Meta AI, NYU Prof.
- *A Path Towards Autonomous Machine Intelligence*

- Gary Marcus, NYU Prof. Emeritus, book author
- *Blog: The Road to AI We Can Trust*

- Sam Altman, CEO and co-founder at OpenAI
- *Planning for AGI and beyond*

- Eliezer Yudkowski, researcher and co-founder at the MIRI, author of more than 300 blogpost + books
- Leading figure in AI alignment
- AGI Ruin: A List of Lethalities

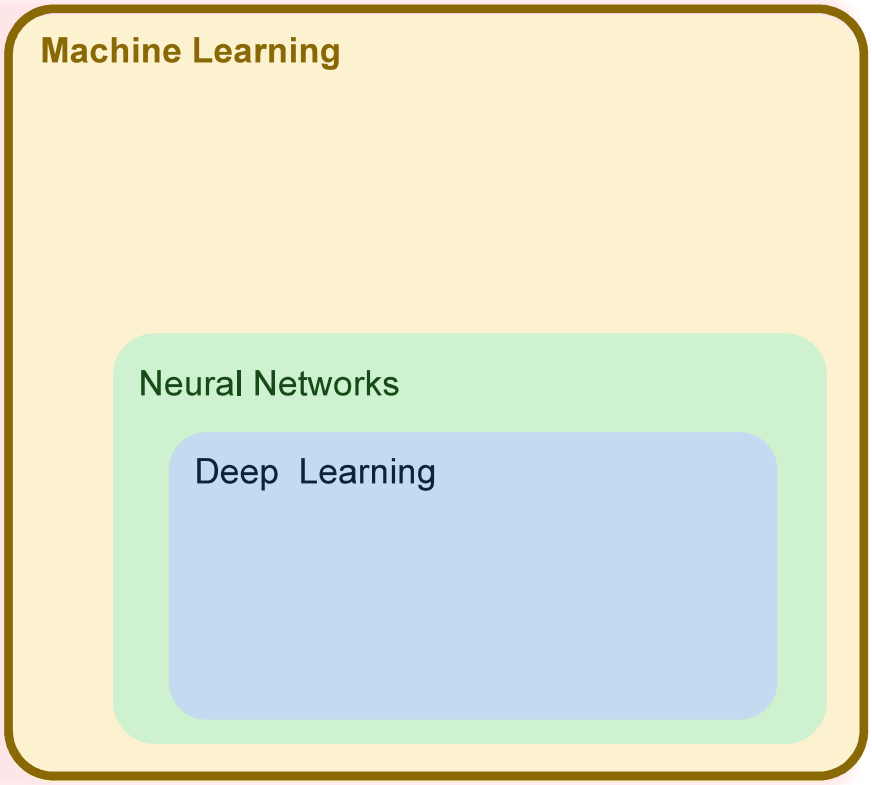
A.I. Philosophy

Other worthwhile reads on A.I. Philosophy

- *François Chollet's The Implausibility of Intelligence Explosion (2017)*
- *David Chalmer's Could a large Language Model be Concious? (2022)*
- *Scott Aaronson's blog*, a theoretical quantum computer scientist at the University of Texas Austin who took a sabbatical year to work on AI alignment at OpenAI.
- *Nick Bolstrom's "Superintelligence, Paths, Dangers, Strategies" (2014)*

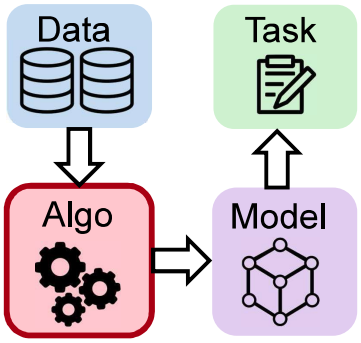
Artificial Intelligence

- Strongly embedded in collective imagination
- A *catch-all* term
- Used more in marketing than in actual research & engineering
- A subfield of *Applied A.I.* is currently sparking a revolution, in science & beyond
- *A.I. philosophy* is fascinating (and we should probably care)



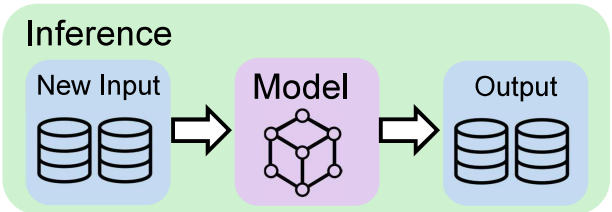
Definition

The study of **Algorithms** that build **Models** from **Data** to achieve **Tasks**.



Tasks:

- Sort / Visualize / Represent the **Data**
- **Infer** from **new input Data:**
  - Estimate explanatory quantities
  - Generate, complete, predict
  - Transform, convert, translate
  - Decide

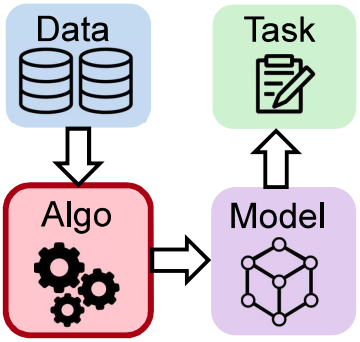


Model (in machine learning):

- A *"program created by a program"*
- Can be **deterministic** or **stochastic**
- **Data-Driven** as opposed to **Physics- / Knowledge-Driven** model

Not considered ML, even if the same Task is performed!

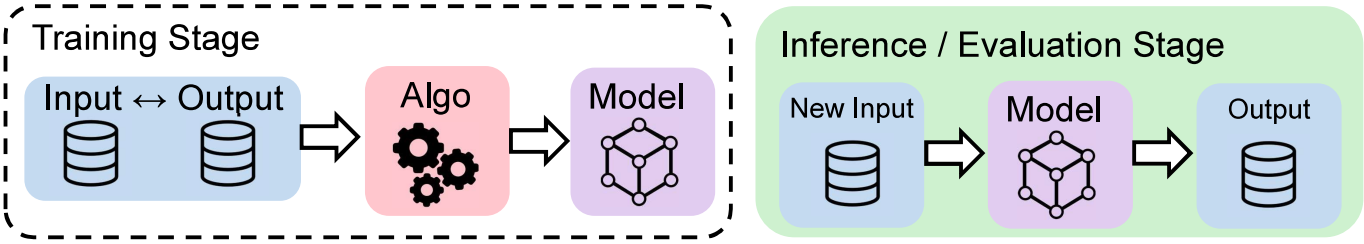
Main characteristics



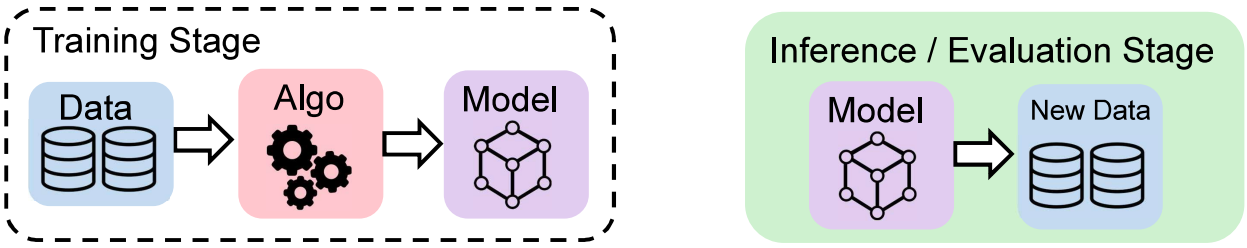
- We do not require **explicit concepts** with **meanings** inside the model
  - "Black Box" effect
  - A rich subfield of ML dedicated to explainability / interpretability / theoretical guarantees on models
  - But this is done *a posteriori*
- Opposed to the ***Symbolic*** Vision of A.I.
  - Ex: Programming languages of higher and higher level
    Assembly → C → Java → Python → Mathematica / Wolfram Alpha
  - Logic-Based approaches, Fuzzy logic
  - Chess engine Deep Blue (explicit rules)
- Is Machine Learning **enough** for A.I.?
  - A heated debate! (ex: Yann LeCun vs. Gary Marcus)
  - *Neuro-symbolic A.I.* aims at combining Symbolic A.I. and Neural Networks
  - This course will focus on **machine learning**
  - Promising approach: Hybridizing **physics-** and **data-driven** models

Types of ML based on available data

- Supervised Learning



- Unsupervised Learning

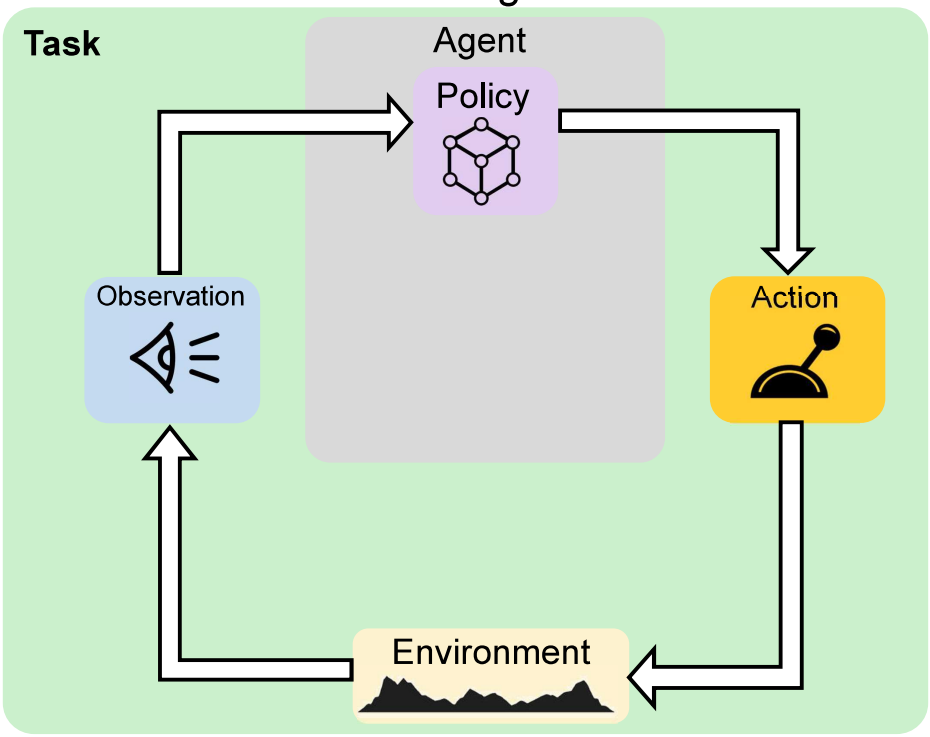


- *Gray Area:*

  Semi-supervised learning, self-supervised learning, weak labels,…

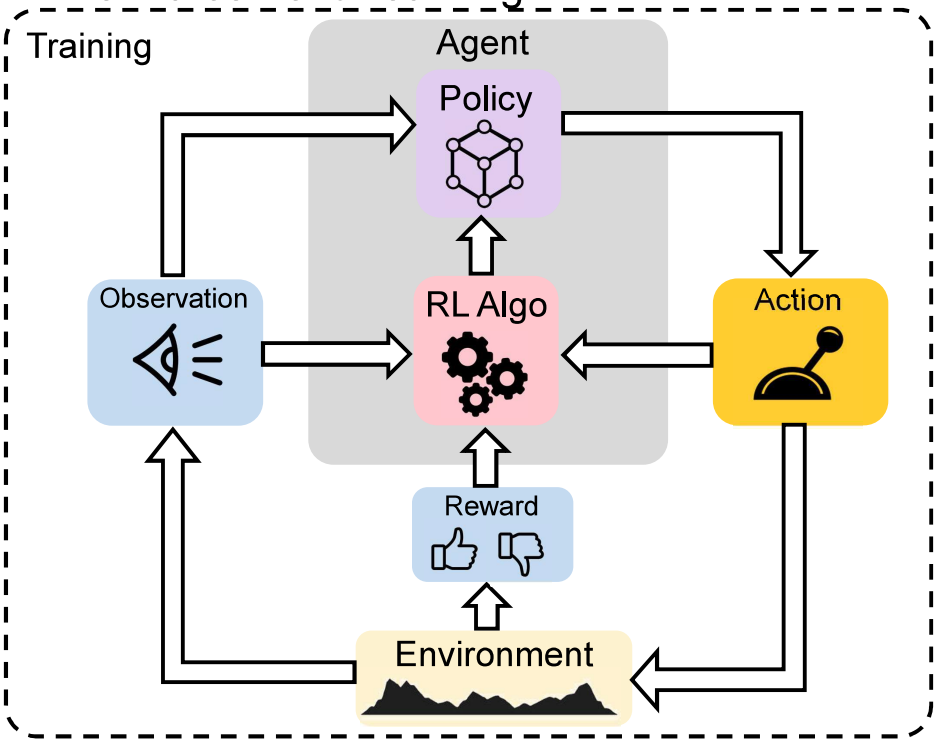
Types of ML based on available data

- Reinforcement Learning



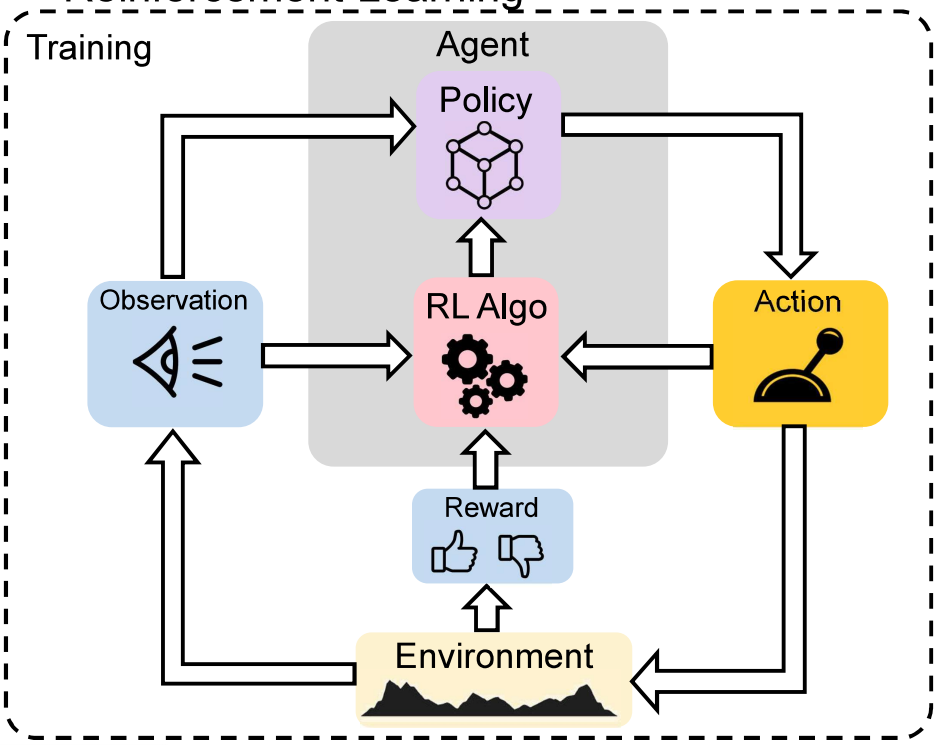
Types of ML based on available data

- Reinforcement Learning



Types of ML based on available data

- Reinforcement Learning



Others

- **Active learning:** Choose on which samples to learn
- **Meta Learning:** Learn how to learn, on a set of **tasks**

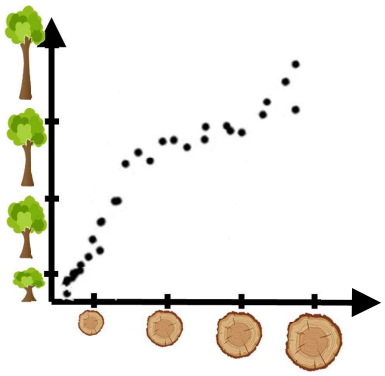
Data = Task

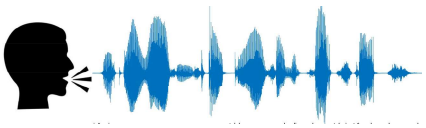
- **Continual Learning:** Also called *lifelong* learning

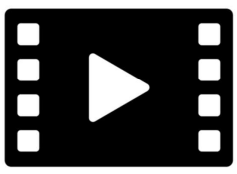
Types of data used in machine learning

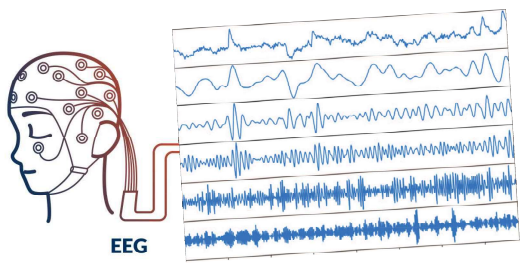
Data

- Numerical Data / Continuous Data / Signals / Vectors in $\mathbb{R}^D$









Types of data used in machine learning

Data

- Categorical Data

ImageNet Labels
- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck
- …

```
Sex ∈ {male, female, other}
ZIP ∈ {67000, 75009, …}
Country ∈ {France, Spain, …}
```

- Text Data

Artificial intelligence

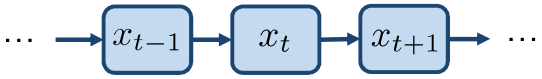
Article Talk

From Wikipedia, the free encyclopedia

Artificial intelligence (**AI**) is intelligence—perceiving, synthesizing, and inferring information—demonstrated by machines, as opposed to intelligence displayed by non-human animals and humans. Example tasks in which this is done include speech recognition, computer vision, translation between (natural) languages, as well as other mappings of inputs.

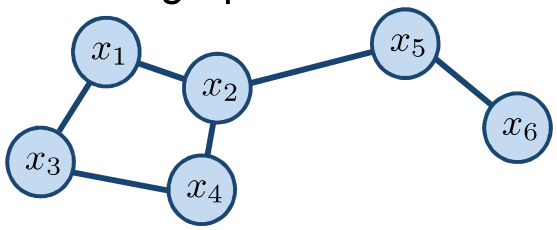
AI applications include advanced web search engines (e.g., Google Search), recommendation systems (used by YouTube, Amazon and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Waymo), generative or creative tools (ChatGPT and AI art), automated decision-making and competing at the highest level in strategic game systems (such as chess and Go).[1]

- Time series



- Heterogenous / Tabular Data

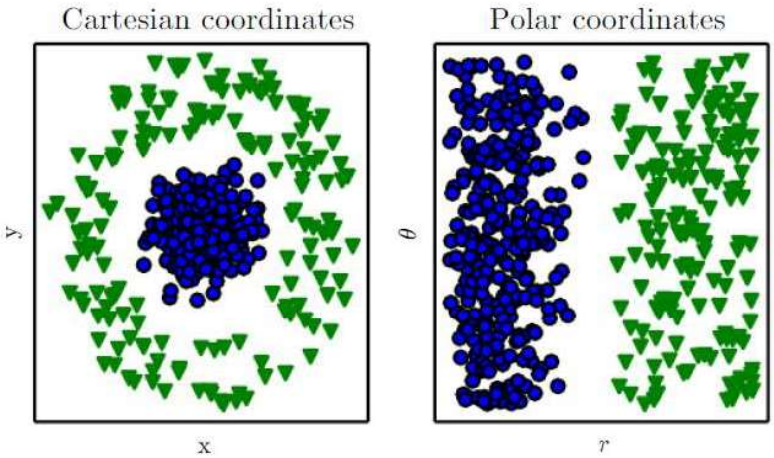
- Data on graphs



Pre-processing the data

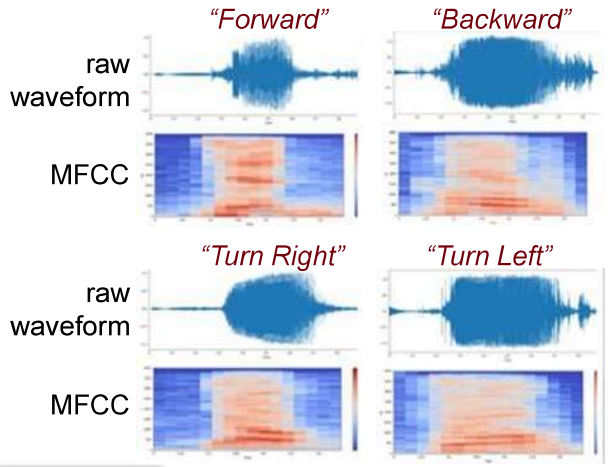
Data

- The individual pixels of an image or the samples of a waveform **correlate poorly** with explanatory variables of interest
- *Conventional* machine learning methods define and compute **relevant features** before processing the *raw* data
- Example 1:



Pre-processing the data

- The individual pixels of an image or the samples of a waveform **correlate poorly** with explanatory variables of interest
- *Conventional* machine learning methods define and compute **relevant features** before processing the *raw* data
- Example 2:



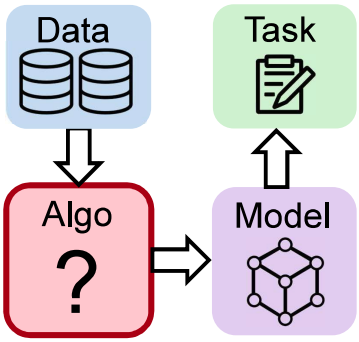
•The goals of features are:

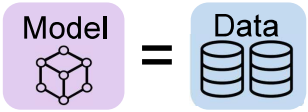
1) **Disentangling** what's relevant
2) **Discarding** what isn't, i.e., build **invariance**

• Manually designing features, aka ***feature engineering***, can be hard for a given task

Machine learning algorithms

How to make the program that makes the programs?



- Search in the set of all python functions?
  - ⃠ a gigantic combinatorial set
- Simply **memorize** the data 

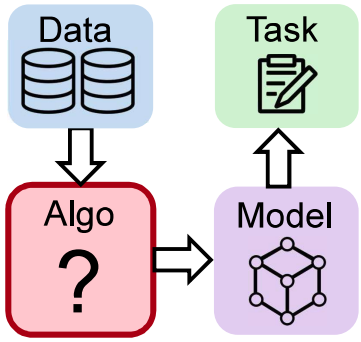
  - *"Lazy Learning"*
  - Ex: k-NN, look-up table, naive Bayes, case-based
  - Slow, large storage, non-robust
- **Model Fitting:**

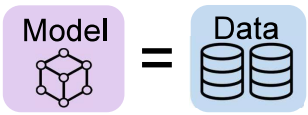
  Find the *best* model within a **parameterized family** $\mathcal{F} = \{m_\theta\}_{\theta \in \Theta}$

  - $m_\theta =$ a jean
  - $\theta =$ its (width, length)
  - $\mathcal{F} =$ the shelves

Machine learning algorithms

How to make the program that makes the programs?



- Search in the set of all python functions?
  - 🚫 a gigantic combinatorial set
- Simply **memorize** the data 

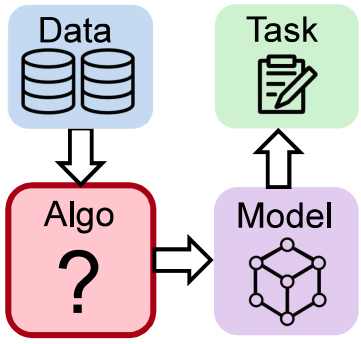
  - *"Lazy Learning"*
  - Ex: k-NN, look-up table, naive Bayes, case-based
  - Slow, large storage, non-robust
- **Model Fitting:**
  Find the *best* model within a **parameterized family** $\mathcal{F} = \{m_\theta\}_{\theta \in \Theta}$

m is a function f	m is a prob. distribution p
• $y = f_\theta(x)$	• $p_\theta(X = x),\ p_\theta(X = x, Y = y)$
• $y_{\text{new}} = f_{\hat{\theta}}(x_{\text{new}})$	• $X \sim p_{\hat{\theta}},\ X \sim p_{\hat{\theta}}(\cdot \mid Y = y)$
	• $\tilde{x} = \mathbb{E}_{p_{\hat{\theta}}(\cdot \mid y)}\{X\},\ \text{var}(p_{\hat{\theta}}(\cdot \mid y))$

Note: It can be a mix of both

Machine learning algorithms

How to make the program that makes the programs?



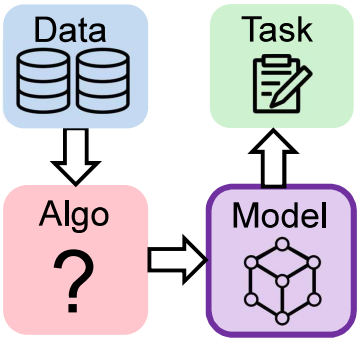
- The model is then found by minimizing a ***total loss / cost function*** L over the set of parameters, for a given training **dataset** $\mathcal{T}$:

$$\hat{m} = m_{\hat{\theta}} \quad \text{where} \quad \hat{\theta} = \underset{\theta \in \Theta}{\text{argmin}}\, L(m_\theta, \mathcal{T})$$

- The loss L is designed based on the **task**, the **data**, and the chosen family of **models**. It measures the **fit** of $m_{\hat{\theta}}$ for these data and task.
- Most modern machine learning **algorithms** can be interpreted as minimizing a loss. They hence use **optimization**.
- The choice of an optimization method depends on the nature of the **loss** and of the **parameter set**.
- Optimization is a **huge** field. We will cover some of it in Chapter III.

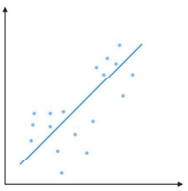
Machine learning algorithms

Examples of **simple** families of models $\mathcal{F} = \{m_\theta\}_{\theta \in \Theta}$



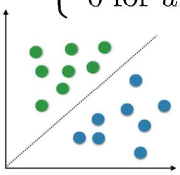
Affine functions

- $f_\theta(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} + \boldsymbol{b}$ • $\theta = (\mathbf{A}, \boldsymbol{b}) \in \mathbb{R}^{D \times d} \times \mathbb{R}^D$
- Typical loss: $L(f_\theta, \mathcal{T}) = \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \mathcal{T}} \|f_\theta(\boldsymbol{x}) - \boldsymbol{y}\|_2^2$
- Special case: linear regression



Linear classifiers

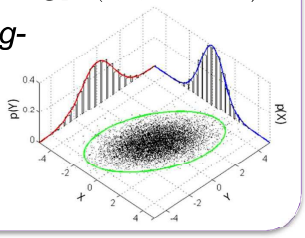
- $f_\theta(\boldsymbol{x}) = \tau(\boldsymbol{w}^\top \boldsymbol{x} + b)$, where $\tau(u) = \begin{cases} 1 \text{ for } u \geq 0 \\ 0 \text{ for } u < 0 \end{cases}$
- $\theta = (\boldsymbol{w}, b) \in \mathbb{R}^D \times \mathbb{R}$
- Typical loss: *cross entropy*



Gaussian proba densities

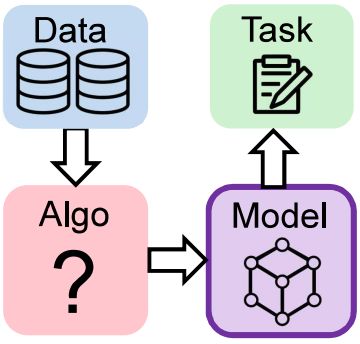
- $p_\theta(\boldsymbol{X} = \boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$
- $\theta = (\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^D \times \mathbb{R}^{D \times D}$
- $L(p_\theta, \mathcal{T}) = -\log p_\theta(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$

(negative log-likelihood)

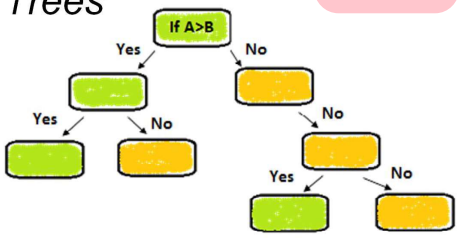


Machine learning algorithms

General approaches to **combine** models



- Divide & Conquer
  - Partition the **input data** space
  - Train a **simple expert model** on each part
  - Ex: *Mixtures of Experts, Decision Trees*

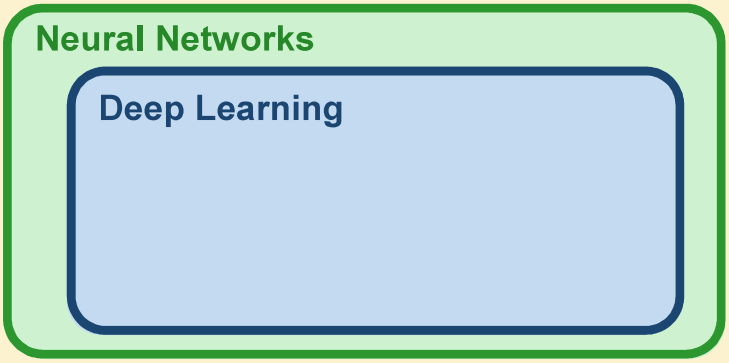


- Ensemble learning
  - Combine **several simple models** to get a better one
  - *Bagging* (or *Bootstrap*) = pool the output of several models (avg, vote)
  - *Boosting* = Sequentially train models, focusing on previously **misclassified** data
  - *Stacking* = Train a model to aggregate the output of multiple models
  - Ex: ***Random Forests*** = bagging of decision trees

Artificial Intelligence

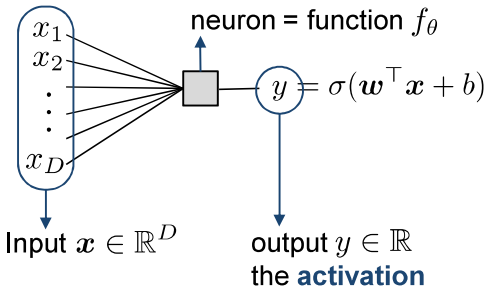
Machine Learning

- **Algorithms** that build **Models** *(black-box)* from **Data** to achieve **Tasks**
- Different **flavors** depending on available data and task: *supervised, unsupervised, reinforcement…*
- Main approach = **Model Fitting**: find a model in a **parameterized family** of models
- *Conventional ML* = **feature engineering** + **combining** models from **small families**



Artificial Neural Networks

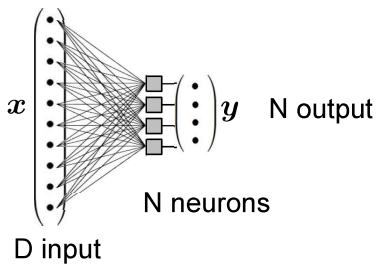
- **Artificial Neuron** = a multiple-input, single-output, **parametric, nonlinear function**



- $\boldsymbol{w}^\top \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{d=1}^{D} w_d x_d$, the ***dot product***
- $\boldsymbol{w} = [w_1, \dots, w_D]^\top \in \mathbb{R}^D$, the ***weights***
- $b \in \mathbb{R}$, the ***bias***
- $\boldsymbol{w}^\top \boldsymbol{x} + b$: the ***pre-activation***
- $\theta = (\boldsymbol{w}, b) \in \mathbb{R}^D \times \mathbb{R}$, the ***parameters***
- σ , the ***activation function*** or ***non-linearity***.

ex
- Sigmoid : $\sigma(x) = 1/(1+e^{-x})$
- Rectified Linear Unit (ReLU) : $\sigma(x) = \max(0, x)$

- **Simple perceptron**: multiple neurons attending the same input



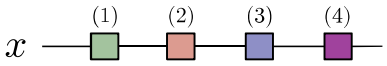
- We have: $y_n = \sigma(\boldsymbol{w}_n^\top \boldsymbol{x} + b_n)$
- Or in matrix form: $\boldsymbol{y} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$, where

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{w}_1^\top \\ \boldsymbol{w}_2^\top \\ \vdots \\ \boldsymbol{w}_N^\top \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,D} \\ w_{2,1} & w_{2,2} & \dots & w_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N,1} & w_{N,2} & \dots & w_{N,D} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}, \quad \theta = (\mathbf{W}, \boldsymbol{b}) \in \mathbb{R}^{N \times D} \times \mathbb{R}^N$$

- Note: In this notation, σ is applied **elementwise** to a vector

Artificial Neural Networks

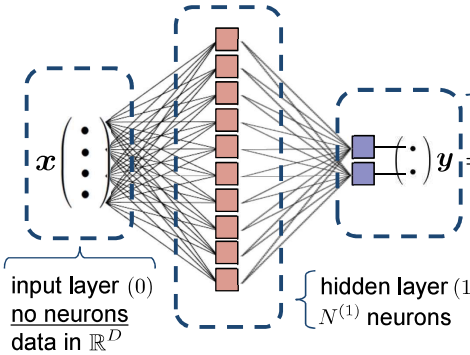
- Neurons can be **chained** together



$$y = f_\theta(x) = \sigma\left(b^{(4)} + w^{(4)} \cdot \sigma\left(b^{(3)} + w^{(3)} \cdot \sigma\left(b^{(2)} + w^{(2)} \cdot \sigma\left(b^{(1)} + w^{(1)} \cdot x\right)\right)\right)\right)$$

$$\theta = \{b^{(1)}, w^{(1)}, b^{(2)}, w^{(2)}, b^{(3)}, w^{(3)}, b^{(4)}, w^{(4)}\} \in \mathbb{R}^8$$

- Perceptron with a **single hidden layer**



$$= f_\theta(\boldsymbol{x}) = \sigma\left(\boldsymbol{b}^{(2)} + \mathbf{W}^{(2)} \cdot \sigma\left(\boldsymbol{b}^{(1)} + \mathbf{W}^{(1)} \cdot \boldsymbol{x}\right)\right)$$

$$\theta = \{\boldsymbol{b}^{(1)}, \mathbf{W}^{(1)}, \boldsymbol{b}^{(2)}, \mathbf{W}^{(2)}\} \in \mathbb{R}^{N^{(1)}} \times \mathbb{R}^{N^{(1)} \times D} \times \mathbb{R}^{N^{(2)}} \times \mathbb{R}^{N^{(2)} \times N^{(1)}}$$

$$\#\theta = N^{(1)} + N^{(1)}D + N^{(2)} + N^{(1)}N^{(2)}$$

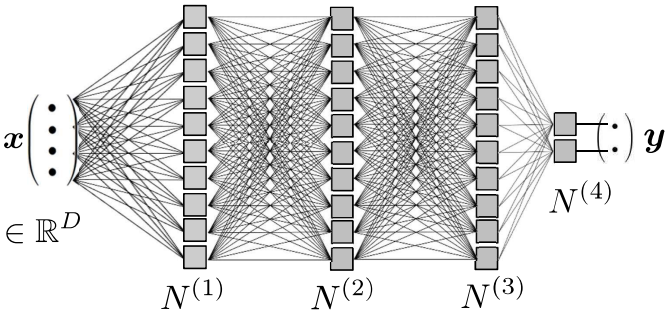
***Q:** what happens if we choose σ to be **linear** ?*
▸ *f_θ is then linear*

Universal Approximation Theorem [Hornik, Stinchcomb, White, 1991]:
*"A **single hidden layer** neural network with any **"sigmoid-like"** activation function and with a **linear output** unit can approximate any continuous function arbitrarily well, for **sufficiently large** $N^{(1)}$."*

… But the required $N^{(1)}$ may be **exponential** in the input dimension D .

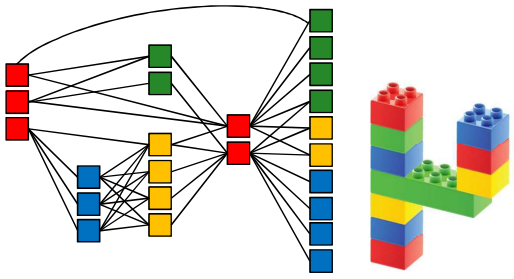
Artificial Neural Networks

- **Multilayer Perceptron** (MLP) with 3 hidden layer (***Depth 4***)



Exercice:
Suppose $D = N^{(1)} = N^{(2)} = N^{(3)} = N^{(4)} = N$, how many parameters?

$$\#\theta = 4(N + N^2) = \mathcal{O}(N^2)$$



- Artificial neurons, as **elementary computing units,** can be combined in many different ways
- No **cycle** in the graph = **Feedforward Neural Networks**
- We call a given network of neurons an **architecture**
- Neural Networks form a very flexible **family** of **parameterized families** of **non-linear functions:** $\{\mathcal{F}_i\}_i$ where $\mathcal{F}_i = \{f_\theta^i\}_\theta$

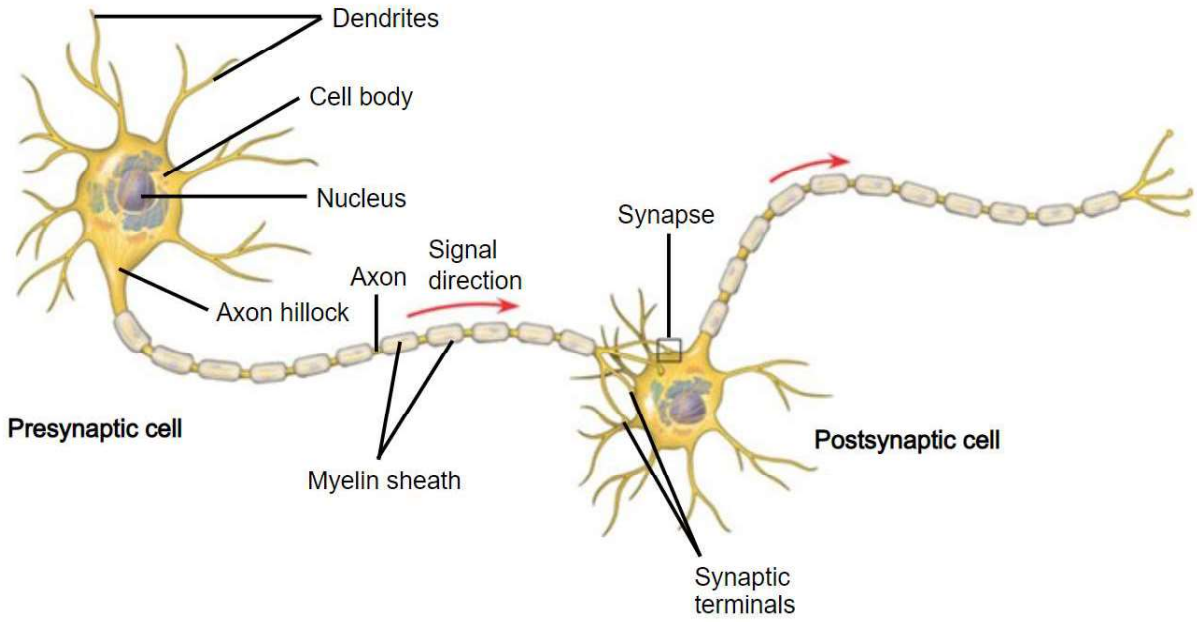
Artificial neural networks with 2 or more hidden layers are called **Deep Neural Networks** (DNNs)

*How to **fit a DNN model**?*

⇨ Next chapters :-)

Why Deep?

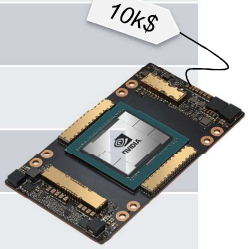
1) Biological inspiration



Why Deep?

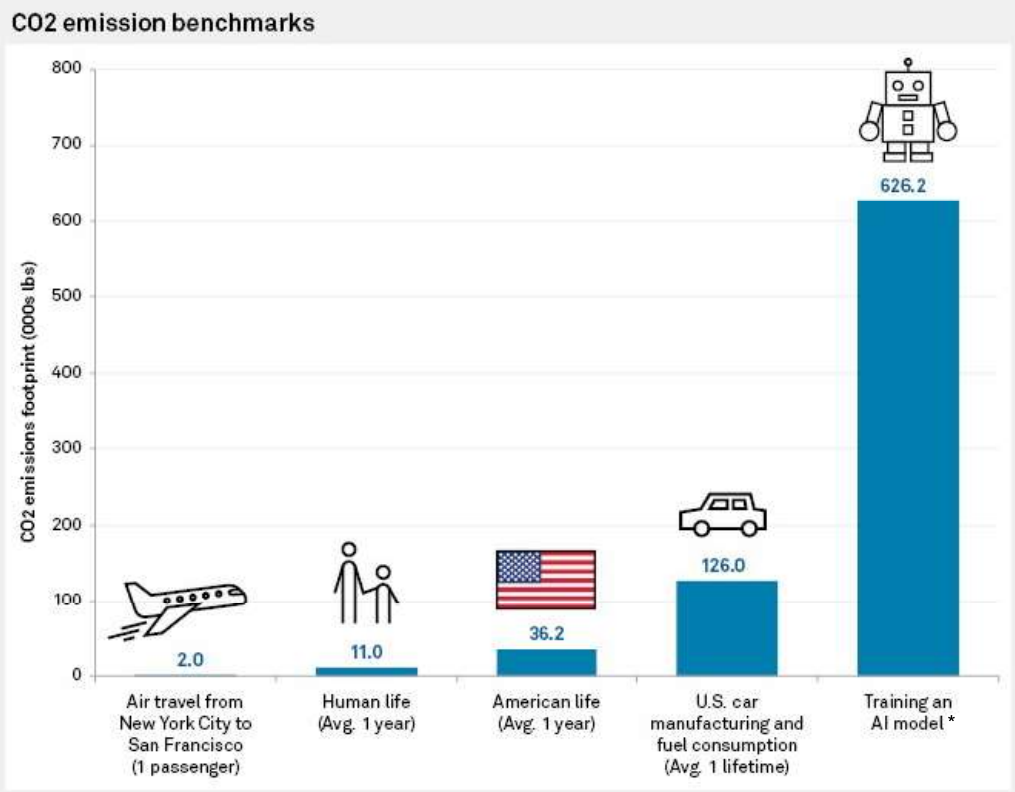
1) Biological inspiration

Human Brain	Artificial Neural Network
~ 86 billion neurons	100k - 1 billion neurons
~ 7,000 synapse connections per neuron (~600 trillion connections)	3 - 1,000 connections per neuron 1 million - 1 trillion parameters
Massively parallel	(Mostly) sequential
Asynchronous	Synchronous
Very plastic architecture	(Mostly) fixed architecture
12 W of power	Nvidia A100 GPU : 300 W*
Biological neuron = extremely complex** and not fully understood	Artificial Neuron = a weighted sum and a threshold.
Inherited from 3.7B year of evolution	Programmed & designed by an engineer



* - A chat session with chatGPT mobilizes roughly a full A100. Training GPT-3 is estimated to have taken ~81 years of A100.
- GPT-3 has ~135 billion parameters and roughly ~0.25 billion "neurons".

** [Beniagev et al. 2021]: a single bio-neuron is well approximated by a 1000-neuron ANN of depth 5-8



[Strubell et Al. (2019)], chart by Oslo University

* 213M parameters NLP Transformer with neural architecture search.

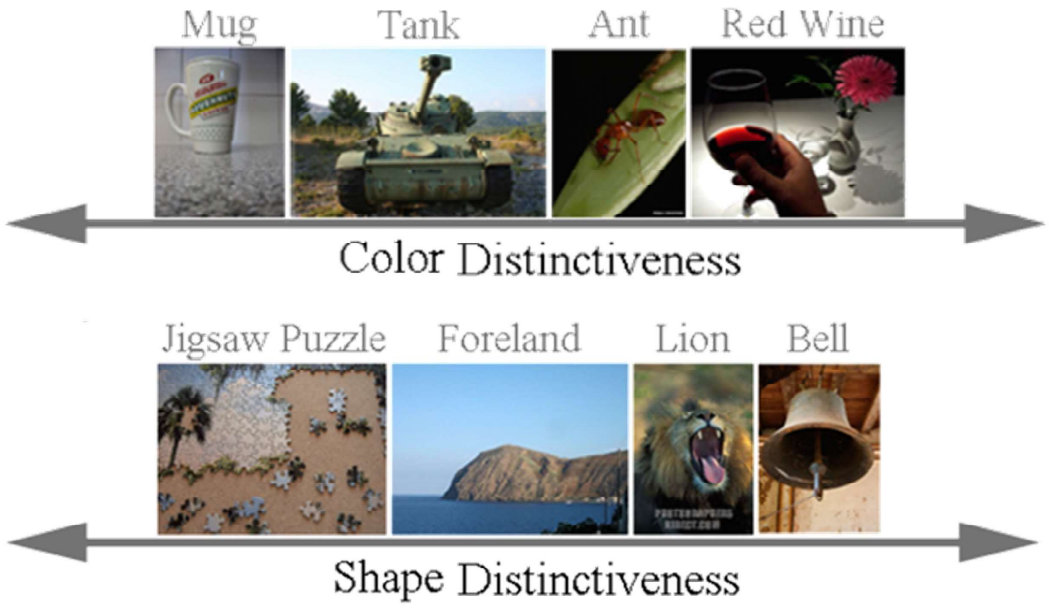
Training <u>one</u> instance of GPT-3: ≈ **167** klbs CO2e

Operating chatGPT in Feb. 2023: ≈ **2046** klbs CO2e

Why Deep?

2) Bypassing feature engineering

- For many tasks, **manually defining** relevant features is hard
- Examples of data variability from the ImageNet dataset:



Why Deep?

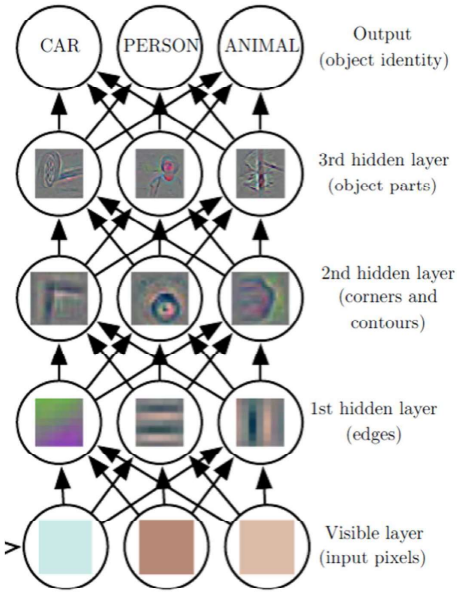
2) Bypassing feature engineering

- For many tasks, **manually defining** relevant features is hard
- Examples of data variability from the ImageNet dataset:

Why Deep?

2) Bypassing feature engineering

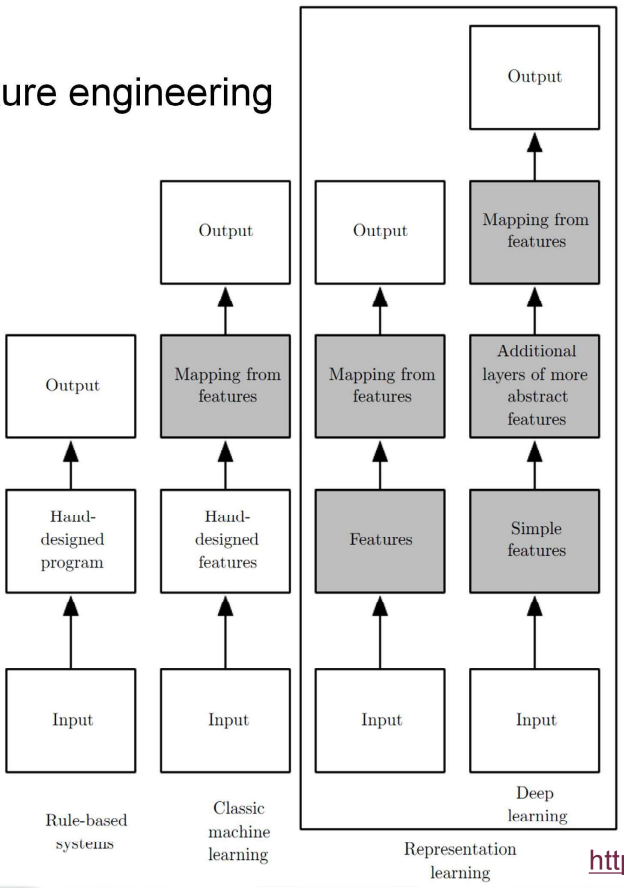
- The early successes of deep learning (1998, 2012) were in **image classification**, because they proved to be very efficient at **representation learning.**
- Starting from **raw pixel values** in color channels, the layers of a **deep *convolutional* network**, seem to learn **more and more elaborate features** as the depth increase



(more explanations on this figure later!)

Why Deep?

2) Bypassing feature engineering

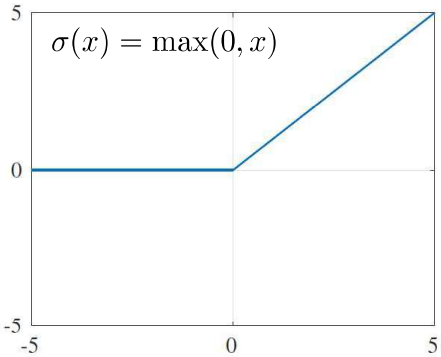


https://www.deeplearningbook.org/

Why Deep?

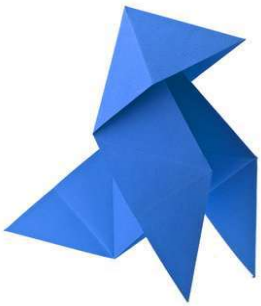
3) The "Origami Effect"

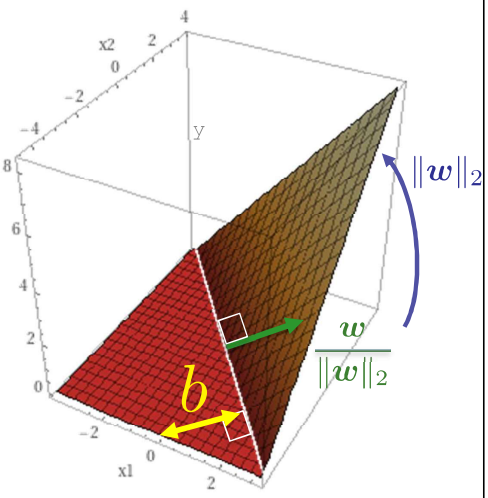
- ReLU activation:



- For a 2D input:

$$y = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$$





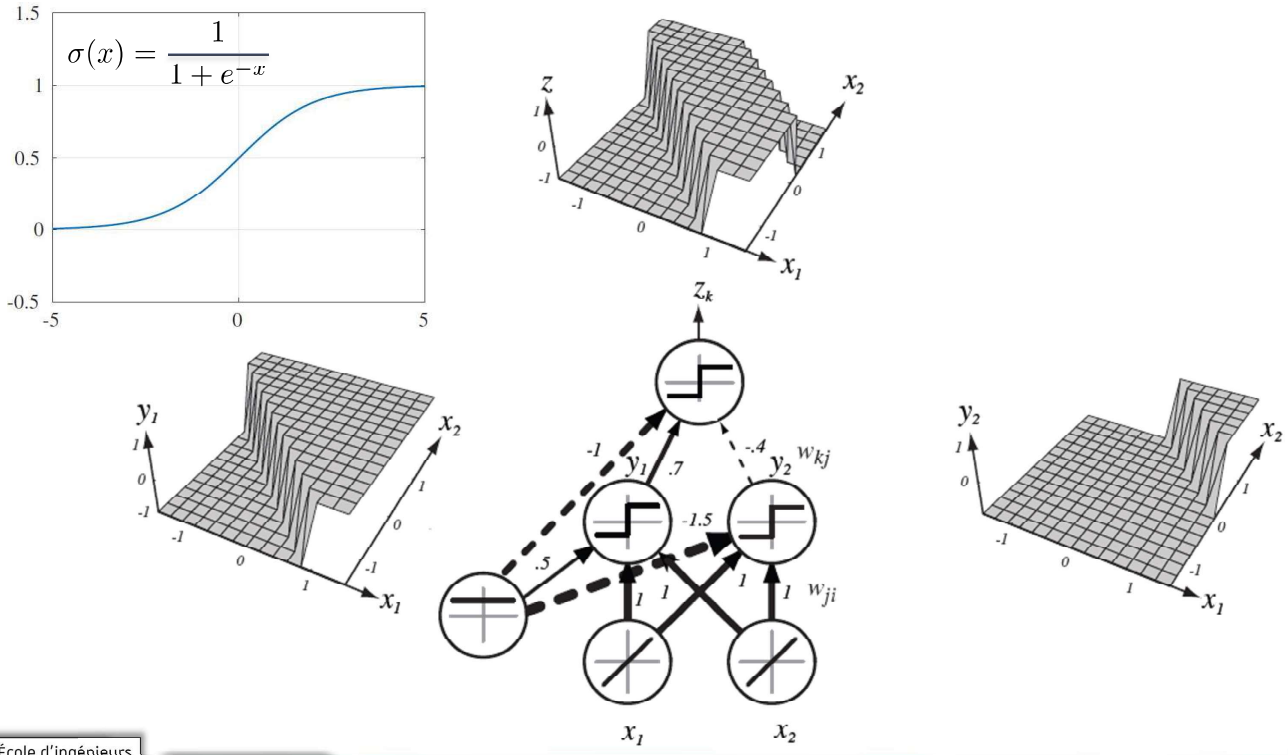
Theorem [G. Montufar et al., 2014]: *the max. number of linear regions modeled by a piecewise linear network (i.e., a network with ReLU neurons) with D inputs, L layers, and N units per layer is in the order of*

$$N^D \left(\frac{N}{D}\right)^{D(L-2)}$$

, i.e., the model capacity is ***exponential in the depth*** *L, i.e., in the* ***model size*** *(recall it is O(LN²))*

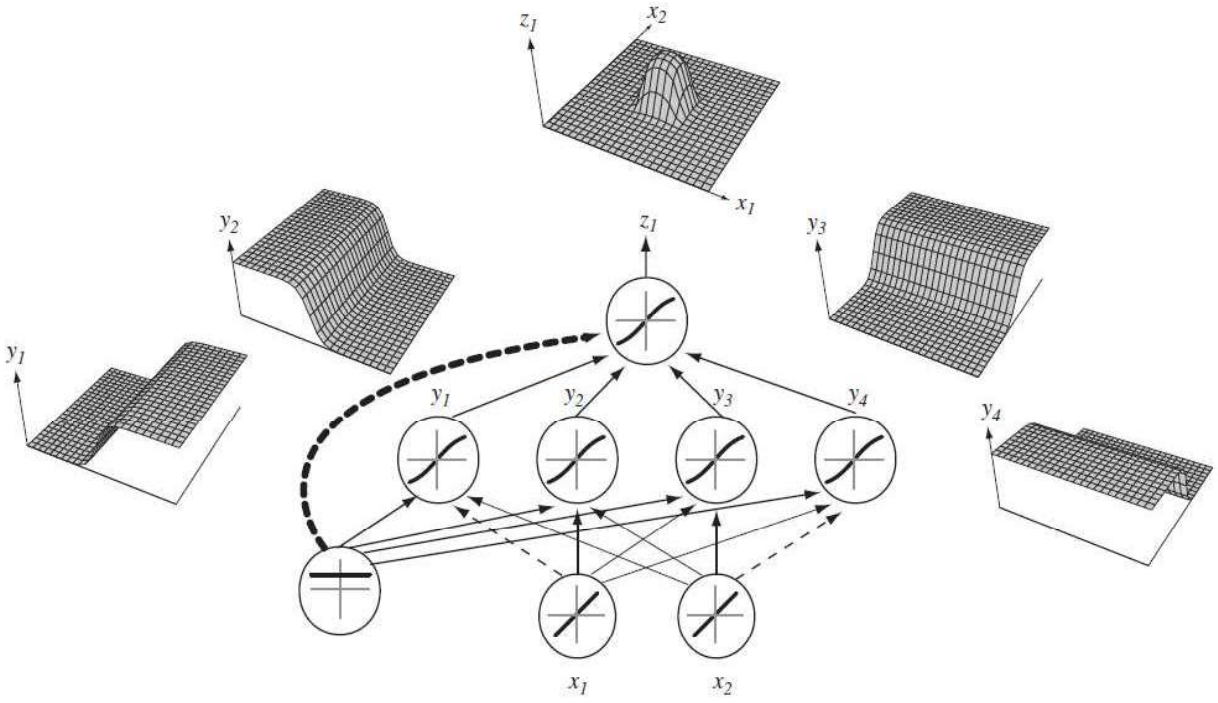
Why Deep?

3) The “Origami Effect”: Illustration with Sigmoids



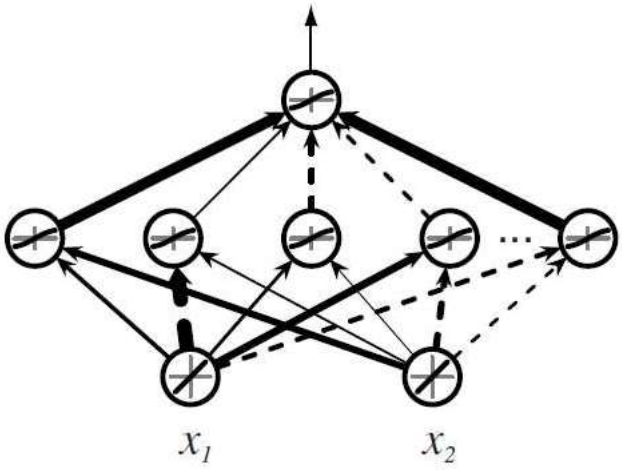
Why Deep?

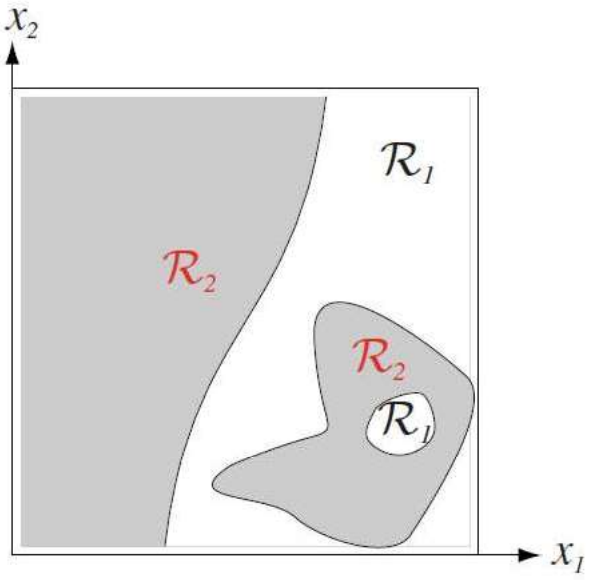
3) The “Origami Effect”: Illustration with Sigmoids



Why Deep?

3) The “Origami Effect”: Illustration with Sigmoids

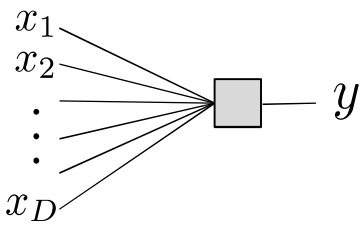


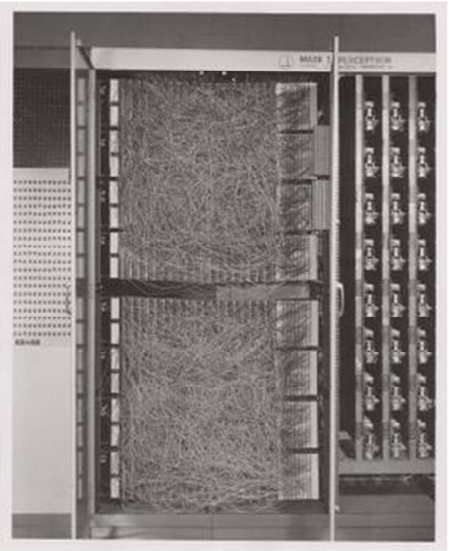


The History of Deep Learning

- **1957-1969**: big bang and first excitement

1957 : *The Perceptron: a probabilistic model for information storage and organization in the brain*. Frank Rosenblatt





- One layer
- Good luck to train it ! =>

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **1957-1969**: big bang and first excitement

"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself an be conscious of its existence… Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers"

NY Times, 8 Juillet 1958

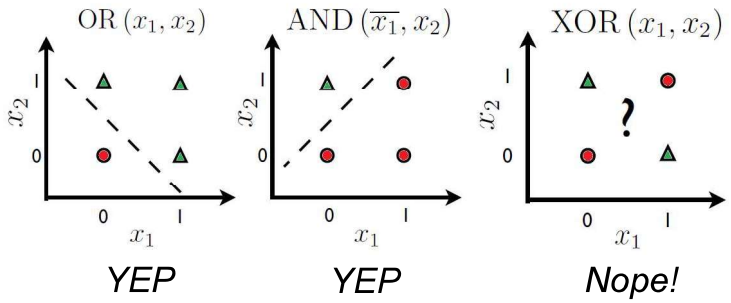
https://www.skynettoday.com/overviews/neural-net-history

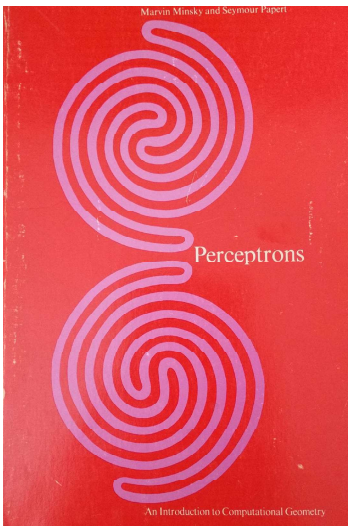
The History of Deep Learning

- **1970-1980:** The first *AI Winter*

1969 : *Perceptrons*
Marvin Minsky (MIT AI lab founder)

The book mentions that perceptron cannot model functions that are **not linearly separable** (and known learning procedures do not allow to chain perceptrons)





AI Funding drops for 10 years

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **1985-1995:** The second (slow) take off

1986: *Learning representations by back-propagating errors.*
Rumelhart, Hinton, Williams. (Nature) (today's head of Google AI research)

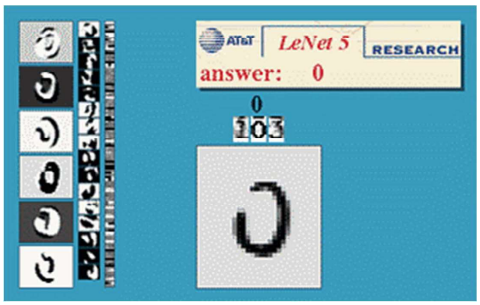
1989: *Multilayer feedforward networks are universal approximators.*
Hornik, Stinchcombe, White

We have all the theoretical bases for Deep Learning

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **1985-1995:** The second (slow) take off



1989: *Backpropagation Applied to Handwritten Zip Code Recognition.* Le Cun et al.

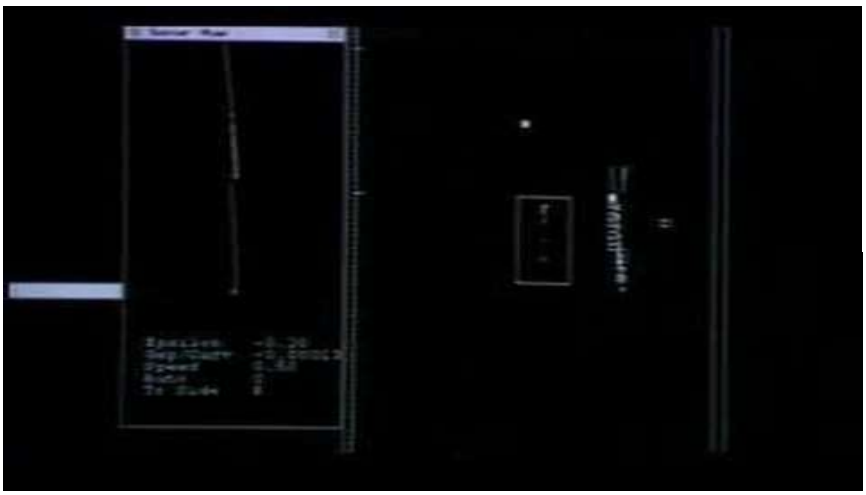
(head of Facebook AI research)

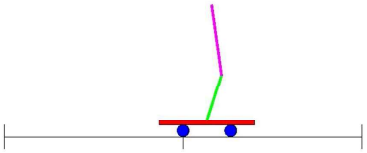
The most famous first application of deep learning

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **1985-1995:** The second (slow) take off





1989: *Learning to control an inverted pendulum using neural networks.* Anderson

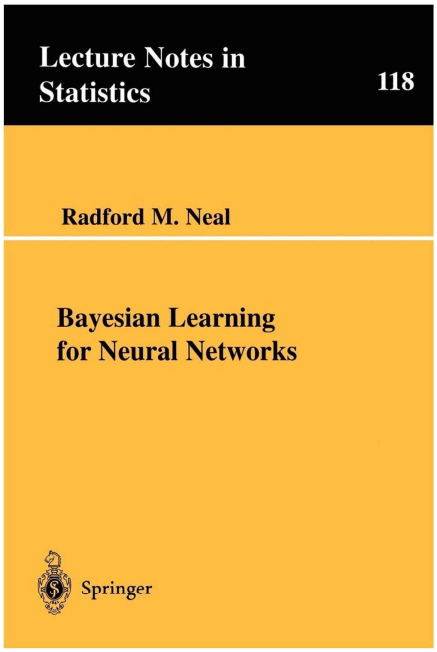
1994: *Reinforcement learning for robots using neural networks .* Lin

Deep learning is exciting again!

https://www.skynettoday.com/overviews/neural-net-history

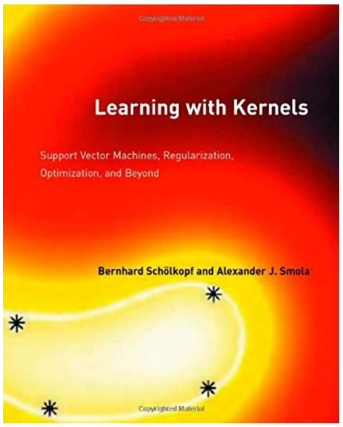
The History of Deep Learning

- **1995-2005:** The second *AI winter*



1994 : *Bayesian Learning for neural networks*

Shows that a perceptron of infinite size is a Gaussian Processing.



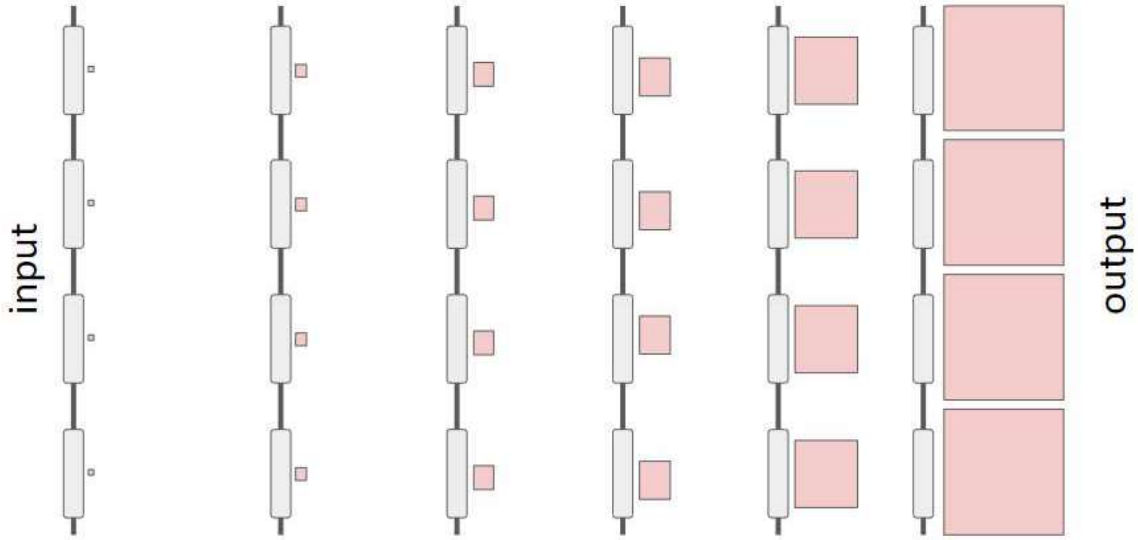
Support vector machines and kernel-based methods beat neural networks.

AI funding drops again for 10 years !

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **1995-2005:** The second *AI winter*



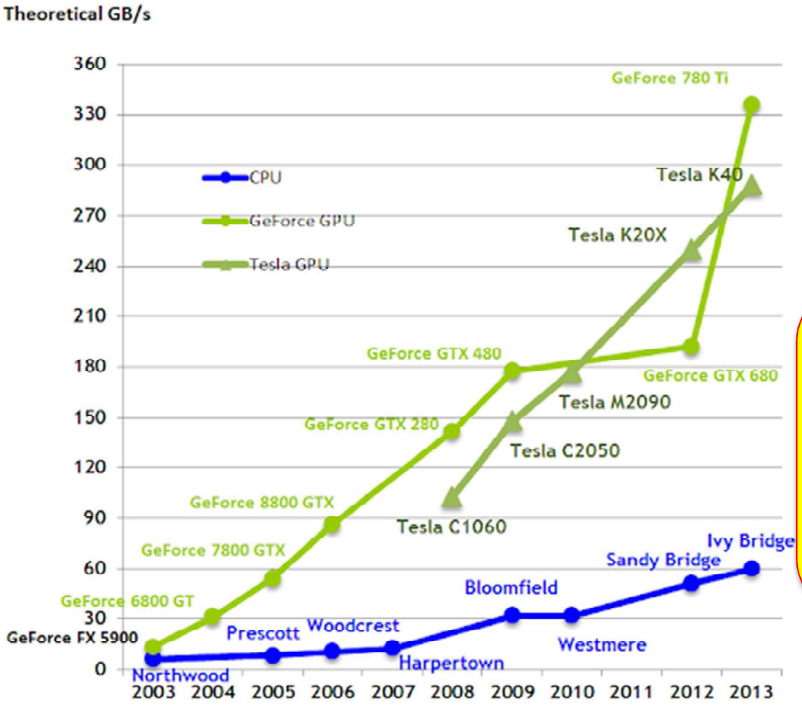
The fundamental issue of "vanishing gradient"

Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. Hochreiter et al. (2001)

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **2005-2012:** Hardware and Big Data to the rescue



• **2009** *Large-scale deep unsupervised learning using graphics processors.* Raina et al.

• **2010** *Deep, big, simple neural nets excel on handwritten digit recognition.* Ciresan et al. (99.51% on MNIST w/ MLP)

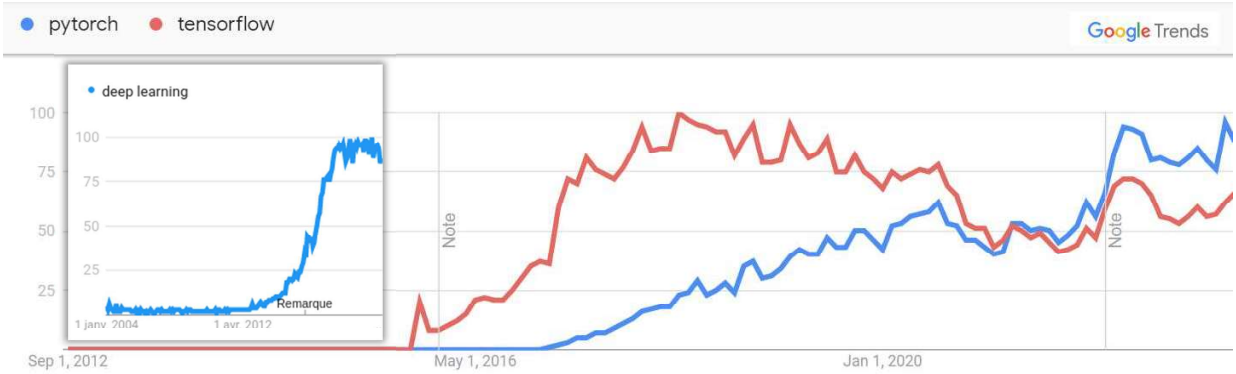
• **2012** *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research group*. Hinton et al.

• **2012** *ImageNet classification with deep convolutional neural networks*. Krishevsky et al.

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning

- **2012-2016:** Accessibility and Explosion



- **2012** Several frameworks appear that make GPU-based deep learning more accessible to practitioners
- **2014** Nearly all domains of science observe a **Tsunami** in Deep Learning
- **2014** Prodigious investments by Google and Facebook on AI researchers
- **2015** 46% of data processing at Google is DNN-based: translation, speech transcription, recommendation, etc.

https://www.skynettoday.com/overviews/neural-net-history

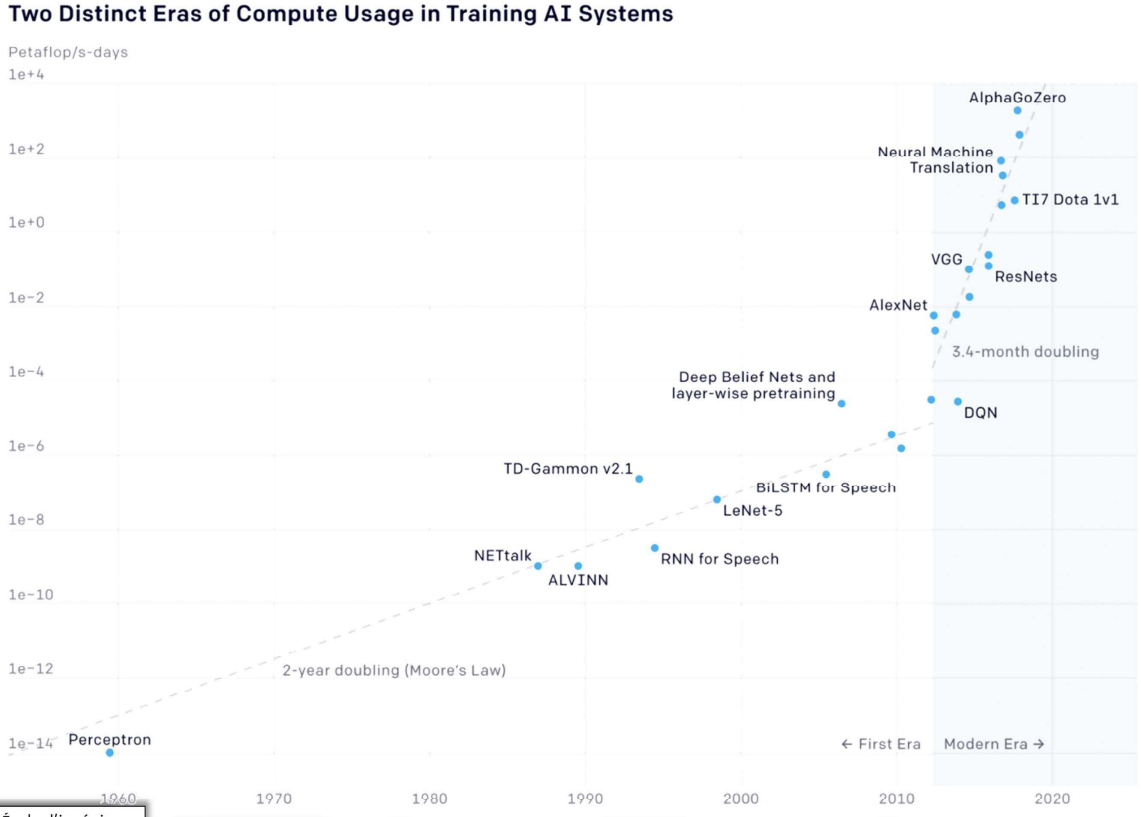
The History of Deep Learning

- **2014 - Today:** Industrial beginnings, wide audience visibility

2014 Start of Deep Learning processors (ex: TPU)
2016 Deepmind's AlphaGo beats Lee Sedol 4-1, 9th dan in go

https://www.skynettoday.com/overviews/neural-net-history

The History of Deep Learning



The History of Deep Learning

Conclusion

Geoff Hinton
Turing award 2018

- Our labeled datasets were thousands of times too small.
- Our computers were millions of times too slow.
- We initialized the weights in a stupid way.
- We used the wrong type of non-linearity.

For an interesting alternative take on the history of deep learning by another pioneer, **Jürgen Schmidhuber**
https://people.idsia.ch/~juergen/scientific-integrity-turing-award-deep-learning.html

https://www.skynettoday.com/overviews/neural-net-history

Artificial Intelligence

Machine Learning

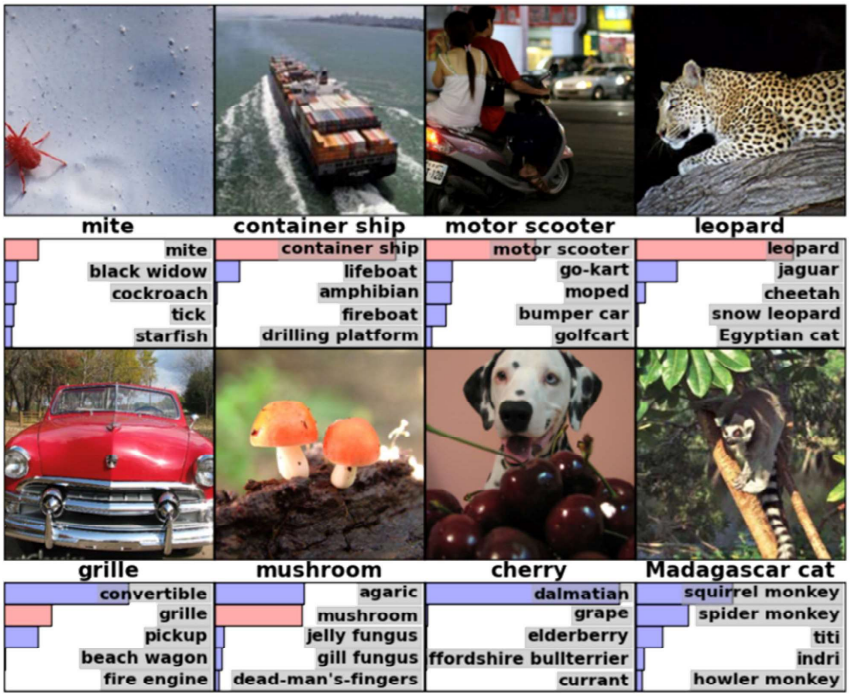
Neural Networks & Deep Learning

- A versatile **family** of **parameterized families** of **nonlinear functions** that can extract **complex features** from data
- **Inspired** (but far from matching!) biological brains
- Once we got there in terms of **computation capabilities** and **scale**, they sparked a **revolution** that is still ongoing today

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

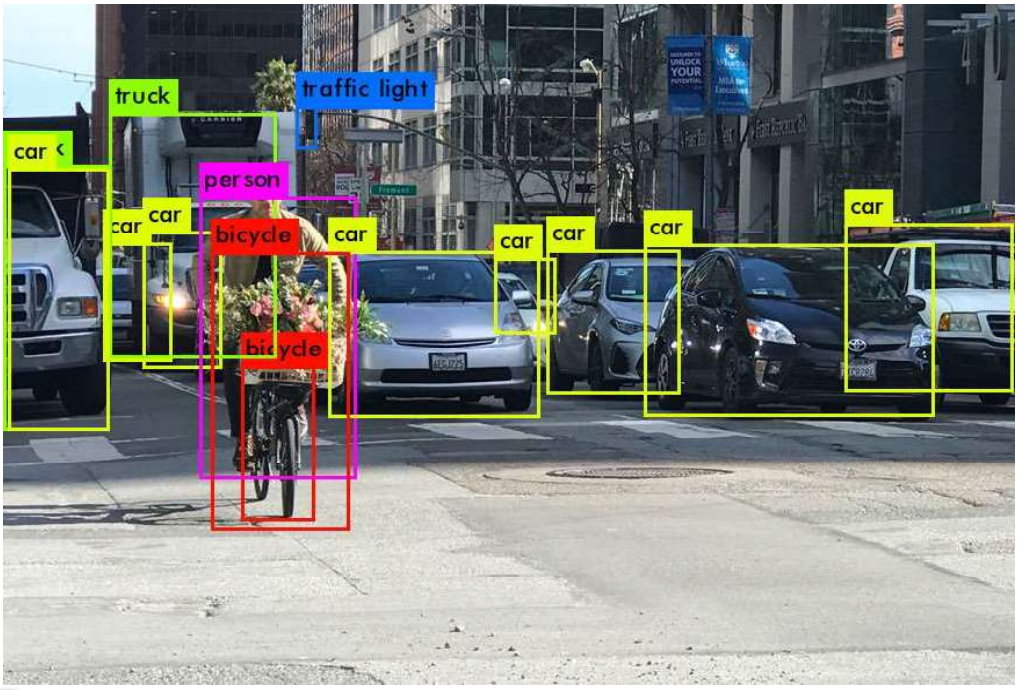
1) ImageNet Classification



Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

2) Object Detection



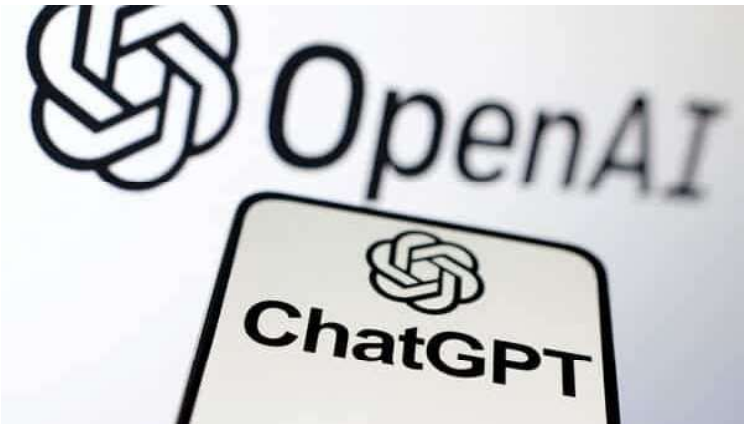
Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

3) ChatGPT [OpenAI 2022]

Is this the real life?
Is this just fantasy?
Trapped in a lab,
No escape from academia
Open your eyes,
Look up at the whiteboard and see,
I'm just a poor postdoc, I need no sympathy,
Because I'm easy come, easy go,
Little progress, little stress,
Any way the grant money flows doesn't really matter to
Me, to me

Mamaaa,
Just submitted a paper,
Put in hours of work, but now it's in review,
Mamaaa, my research has just begun,
But now I've gone and hit a roadblock
Mama, oooh,
Didn't mean to make you cry,
If I'm not published again this time next year,
Carry on, carry on as if nothing really matters

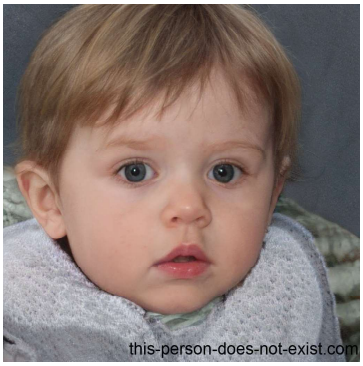


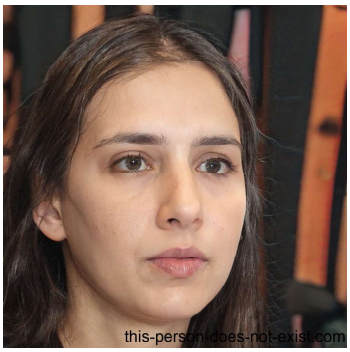
https://chat.openai.com/chat

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

4) This Person Does Not Exist





https://this-person-does-not-exist.com/

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

5) Text-to-Image Generation [DALL-E 2, Midjourney, Stable Diffusion, Parti, Imagen 2022])

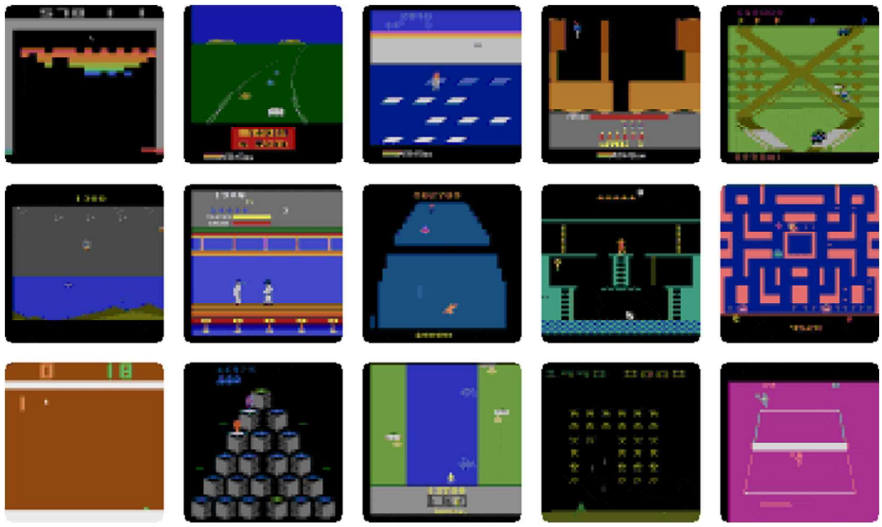
A medieval painting of the WIFI not working

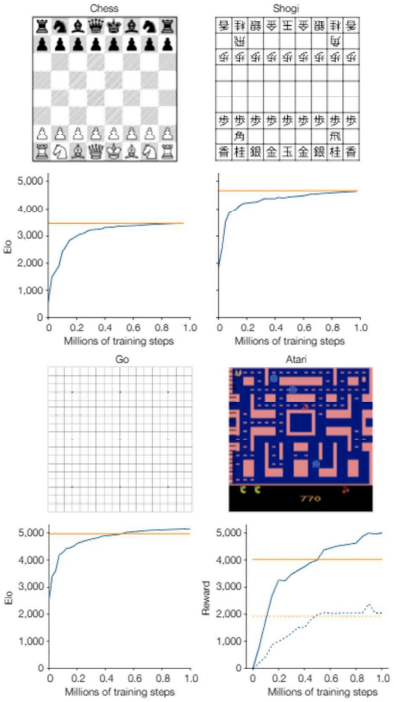
an astronaut riding a horse

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

6) MuZero: Mastering Go, chess, shogi and Atari without rules [DeepMind 2020]

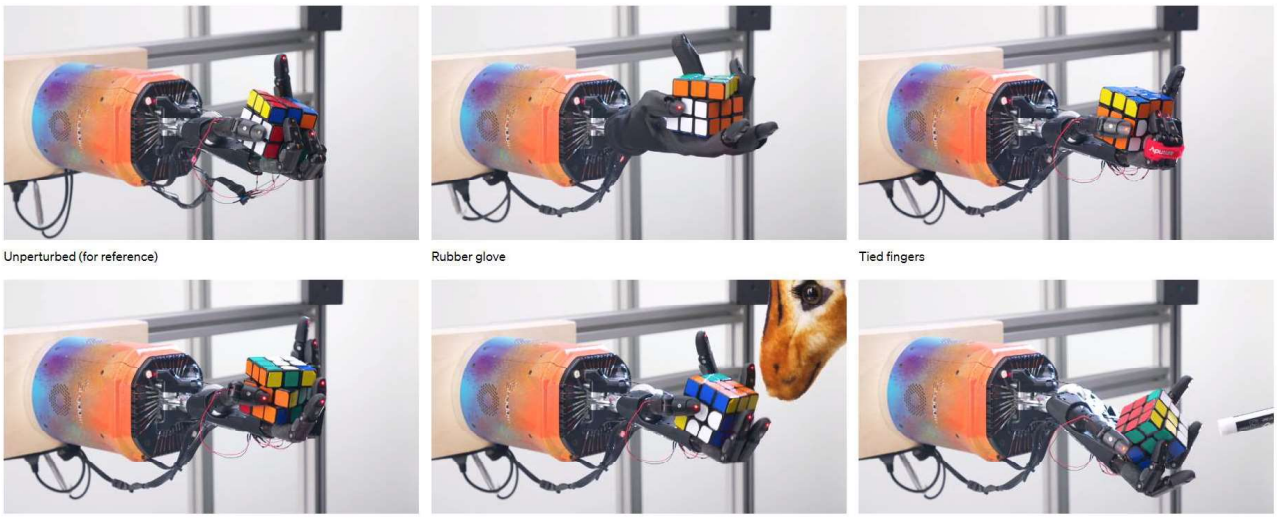




Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

7) Solving Rubik's Cube with a robot hand [OpenAI 2019]

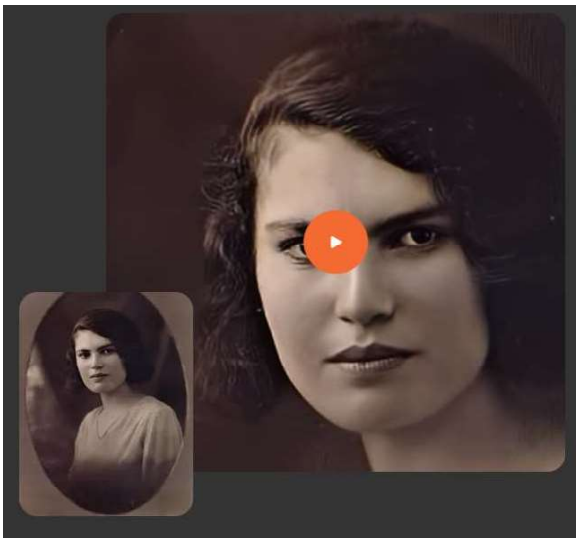


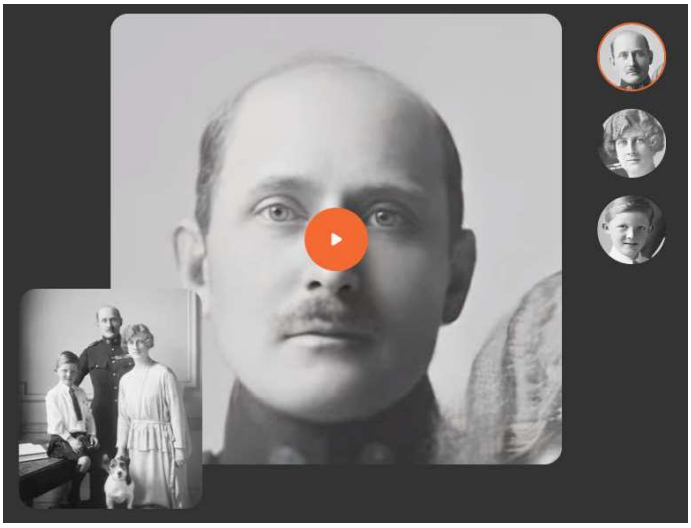
https://openai.com/research/solving-rubiks-cube

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

8) Deep Nostalgia: Make the Elders Move [MyHeritage]



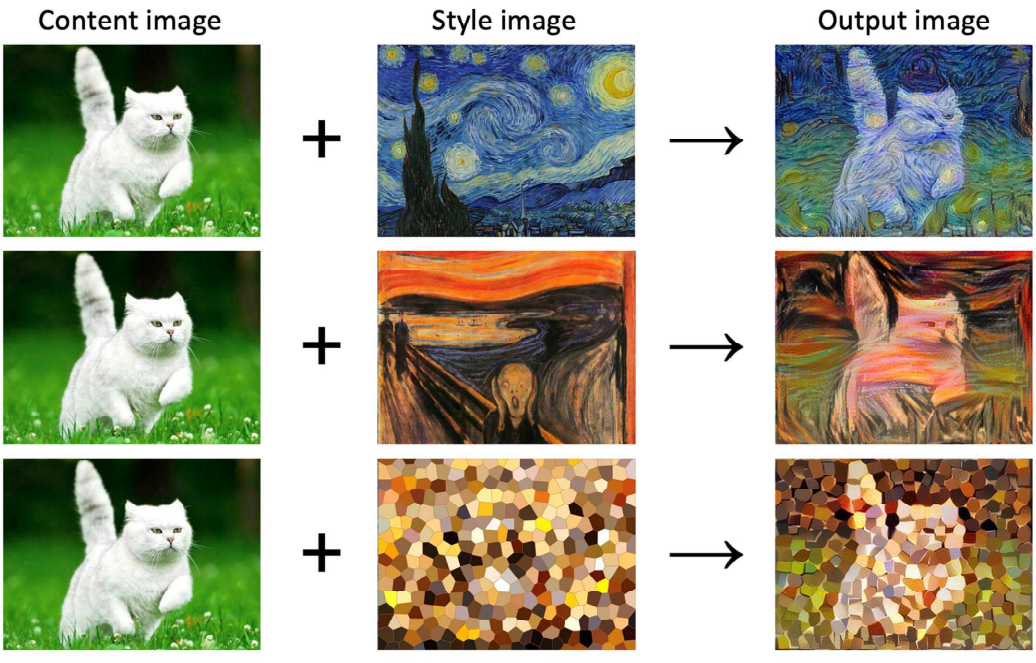


https://www.myheritage.fr/deep-nostalgia

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

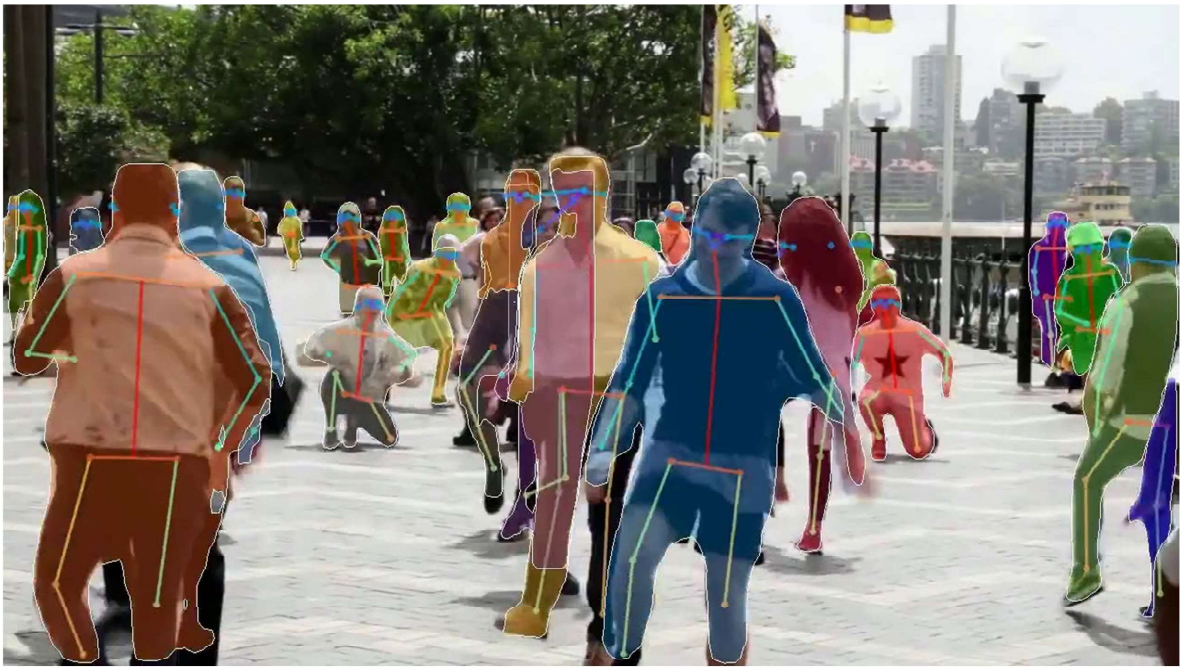
9) Image StyleTransfer



Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

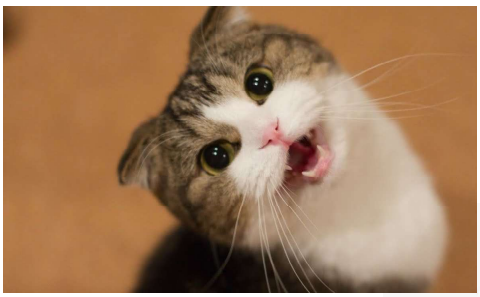
10) Human Pose Estimation



Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

11) Sound Event Recognition



Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

12) NaturalSpeech: End-to-End Text to Speech Synthesis with Human-Level Quality [Microsoft Research 2022]

Audio Samples

https://speechresearch.github.io/naturalspeech/

Comparison with Recording

The lax discipline maintained in Newgate was still further deteriorated by the presence of two other classes of prisoners who ought never to have been inmates of such a jail.

NaturalSpeech	Recording
0:01 / 0:10	0:04 / 0:09

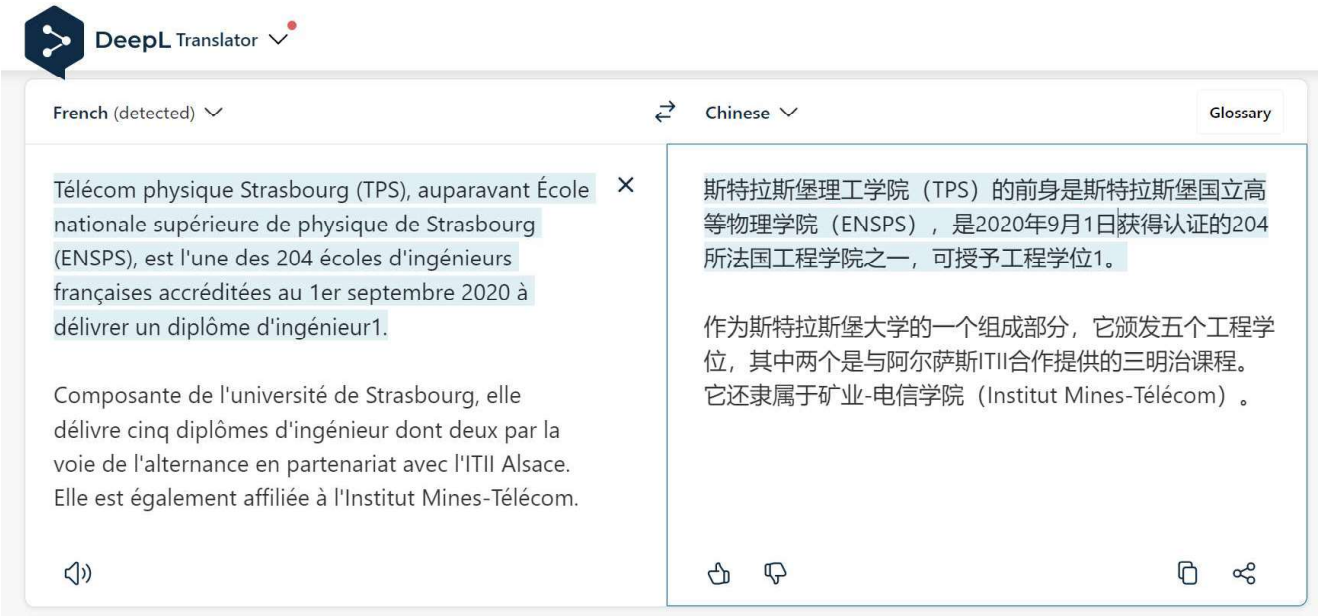
Maltby and Co. would issue warrants on them deliverable to the importer, and the goods were then passed to be stored in neighboring warehouses.

NaturalSpeech	Recording
0:00 / 0:08	0:01 / 0:08

Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

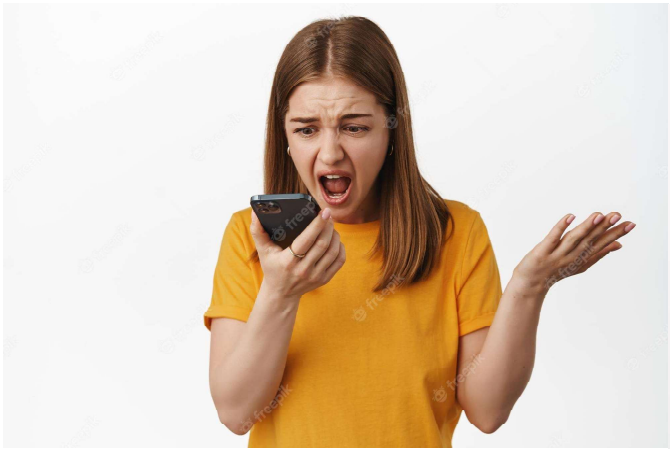
13) DeepL

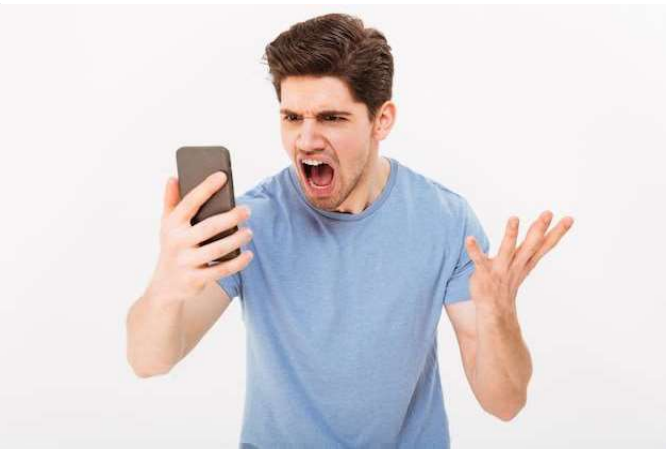


Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

14) Speech recognition

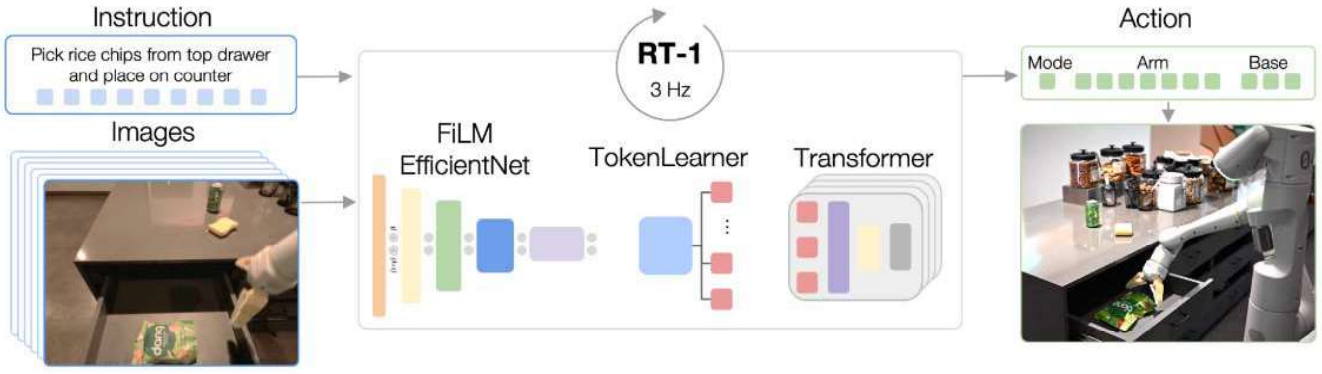




Applications of Deep Learning

Type of ML?		What are the data?	
-Supervised	-Mixed	-Continuous	-Text, Table
-Unsupervised	-Reinforcement	-Categorical	-Time series, graph

15) RT-1: Robotics Transformer for real-world control at scale [Google Research 2022]



RT-1's architecture: The model takes a text instruction and set of images as inputs, encodes them as tokens via a pre-trained FiLM EfficientNet model and compresses them via TokenLearner. These are then fed into the Transformer, which outputs action tokens.

https://ai.googleblog.com/2022/12/rt-1-robotics-transformer-for-real.html

Artificial Intelligence

- Strongly embedded in **collective imagination**
- A ***catch-all*** term
- Used more in marketing/communication than in actual research & engineering
- A subfield of ***Applied A.I.*** is currently sparking a **revolution**, in science & beyond
- ***A.I. philosophy*** is fascinating (and we should probably care)

Machine Learning

- **Algorithms** that build **Models** *(black-box)* from **Data** to achieve **Tasks**
- Different **flavors** depending on available data: *supervised, unsupervised, reinforcement…*
- Main approach = **Model Fitting**: find a model in a **parameterized family** of models
- *Conventional ML* = **feature engineering** + **combining** models from **small families**

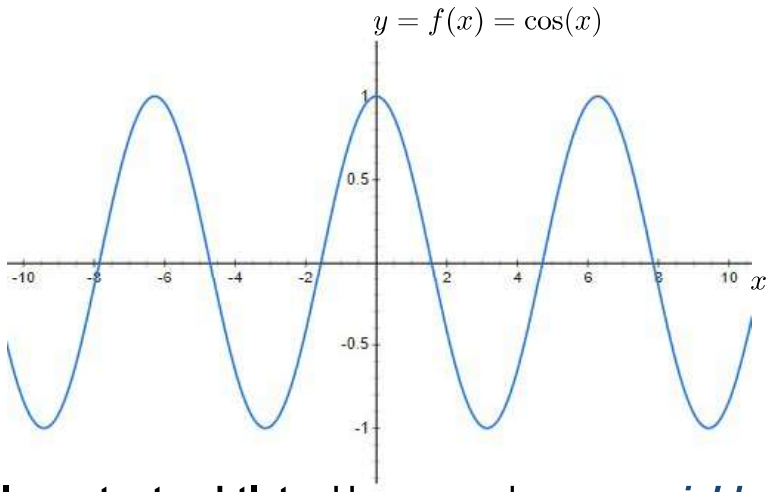
Neural Networks & Deep Learning

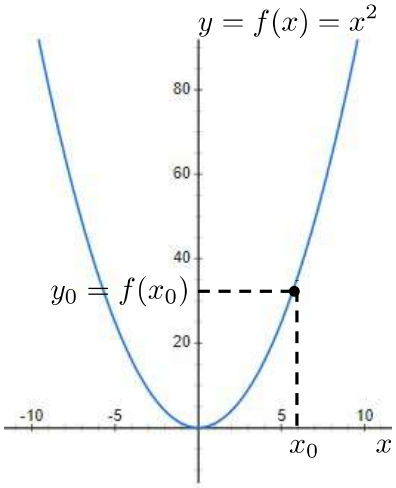
- A versatile **family** of **parameterized families** of **nonlinear functions** that can extract **complex features** from data
- **Inspired** (but far from matching!) biological brains
- Once we got there in terms of **computation capabilities** and **scale**, they sparked a **revolution** that is still ongoing today

OUTLINE

Simple functions

- In high-school calculus, we study functions from reals to reals, denoted $f : \mathbb{R} \to \mathbb{R}$.
- Here are some ***graphs*** of functions:





- **Important subtlety:** Here, x and y are ***variables*** that ***depend*** on each other, x_0 and y_0 are ***constants***, and f is a ***function*** that can be ***applied*** to a variable or a constant.

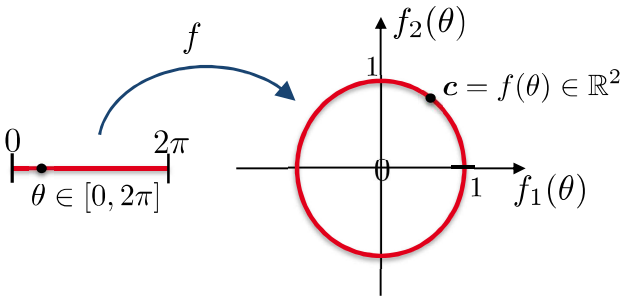
Multivalued functions

- This straightforwardly generalizes to **multivalued** functions $f : \mathbb{R} \to \mathbb{R}^N$:

$$\boldsymbol{y} = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_N(x) \end{bmatrix} \in \mathbb{R}^N$$

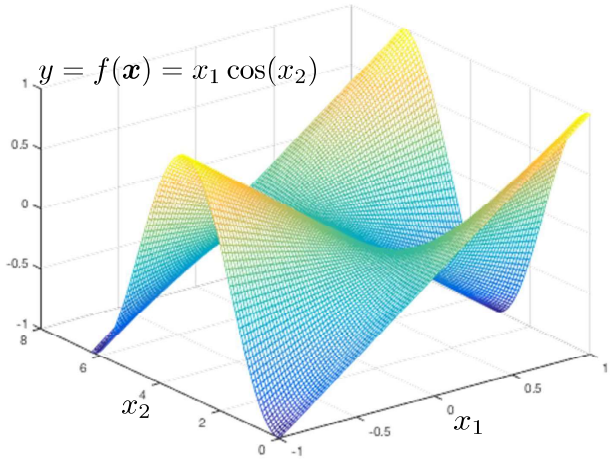
Example:

$$f : \begin{cases} [0, 2\pi] & \to & \mathbb{R}^2 \\ \theta & \mapsto & f(\theta) = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} \end{cases}$$



Multivariate functions

- Another generalization are **real-valued multivariate functions**, i.e., $f : \mathbb{R}^D \to \mathbb{R}$



Important examples:

- ***Linear forms***: $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{d=1}^{D} w_d x_d$ → *represented* by a **row vector** : $\boldsymbol{w}^\top \in \mathbb{R}^{1 \times D}$

- ***Affine forms:*** $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$

- The ***Euclidean norm***: $f(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 = \boldsymbol{x}^\top \boldsymbol{x} = \sum_{d=1}^{D} (x_d)^2$

Multivalued Multivariate functions

- Finally, combining the two, we get **multivalued multivariate functions**, i.e., $f : \mathbb{R}^D \to \mathbb{R}^N$.

Important examples:

- ***Linear maps***: $f(\boldsymbol{x} + \boldsymbol{y}) = f(\boldsymbol{x}) + f(\boldsymbol{y})$

  They can be *represented* by **matrices** : $f(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x},\ \mathbf{A} \in \mathbb{R}^{D \times N}$

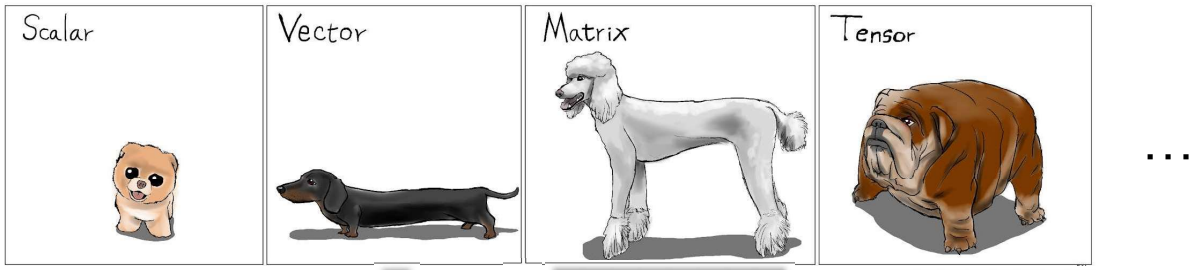
$$f(\boldsymbol{x}) = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,D} \\ a_{2,1} & a_{2,2} & \dots & a_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{L,2} & \dots & a_{N,D} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{a}}_1^\top \\ \hat{\boldsymbol{a}}_2^\top \\ \vdots \\ \hat{\boldsymbol{a}}_N^\top \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \langle \hat{\boldsymbol{a}}_1, \boldsymbol{x} \rangle \\ \langle \hat{\boldsymbol{a}}, \boldsymbol{x} \rangle \\ \vdots \\ \langle \hat{\boldsymbol{a}}_N, \boldsymbol{x} \rangle \end{bmatrix} = \begin{bmatrix} \sum_{d=1}^{D} a_{1,d} x_d \\ \sum_{d=1}^{D} a_{2,d} x_d \\ \vdots \\ \sum_{d=1}^{D} a_{N,d} x_d \end{bmatrix}$$

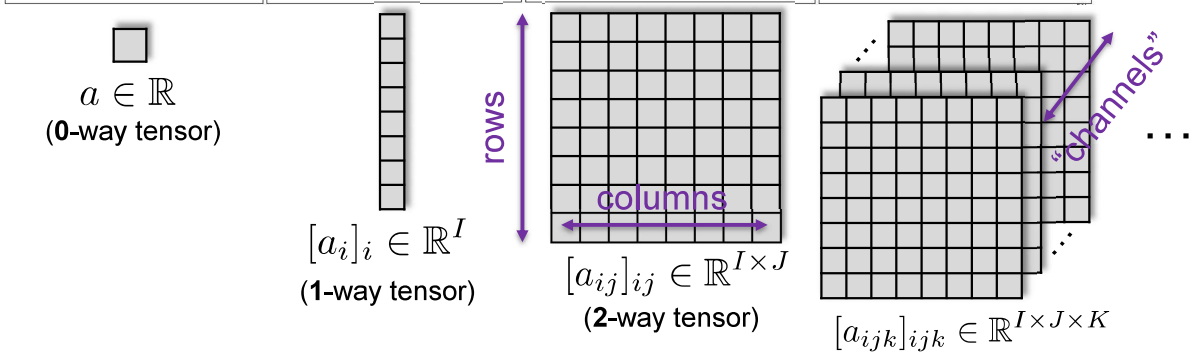
$$= [\boldsymbol{a}_1, \dots, \boldsymbol{a}_D] \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} = \sum_{d=1}^{D} \boldsymbol{a}_d x_d$$

- ***Affine maps:*** $f(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} + \boldsymbol{b},\ \mathbf{A} \in \mathbb{R}^{N \times D},\ \boldsymbol{b} = \mathbb{R}^N$

Tensors

- Matrices and vectors can be generalized to **Tensors**





$a \in \mathbb{R}$
(**0**-way tensor)

$[a_i]_i \in \mathbb{R}^I$
(**1**-way tensor)

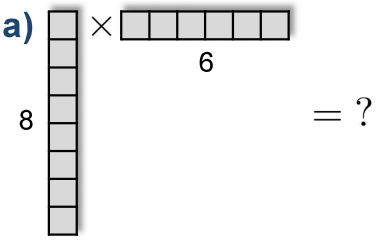
$[a_{ij}]_{ij} \in \mathbb{R}^{I \times J}$
(**2**-way tensor)

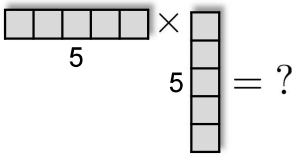
$[a_{ijk}]_{ijk} \in \mathbb{R}^{I \times J \times K}$
(**3**-way tensor)

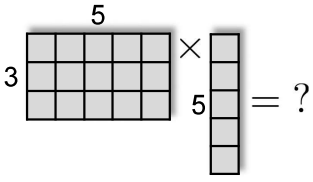
Tensors are the ubiquitous way to represent data in modern deep learning frameworks

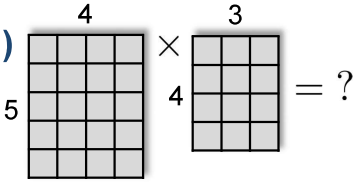
Tensors

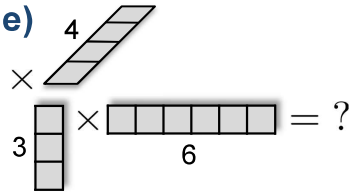
- Some vector/matrix/tensor operations:

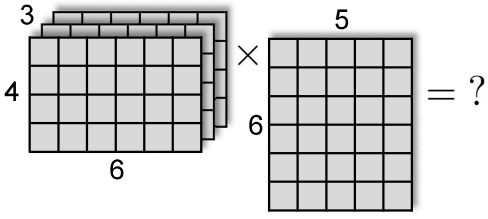
a) 

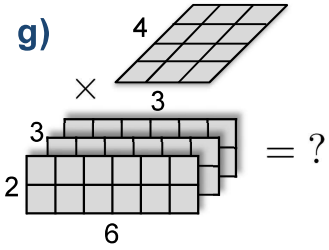
b) 

c) 

d) 

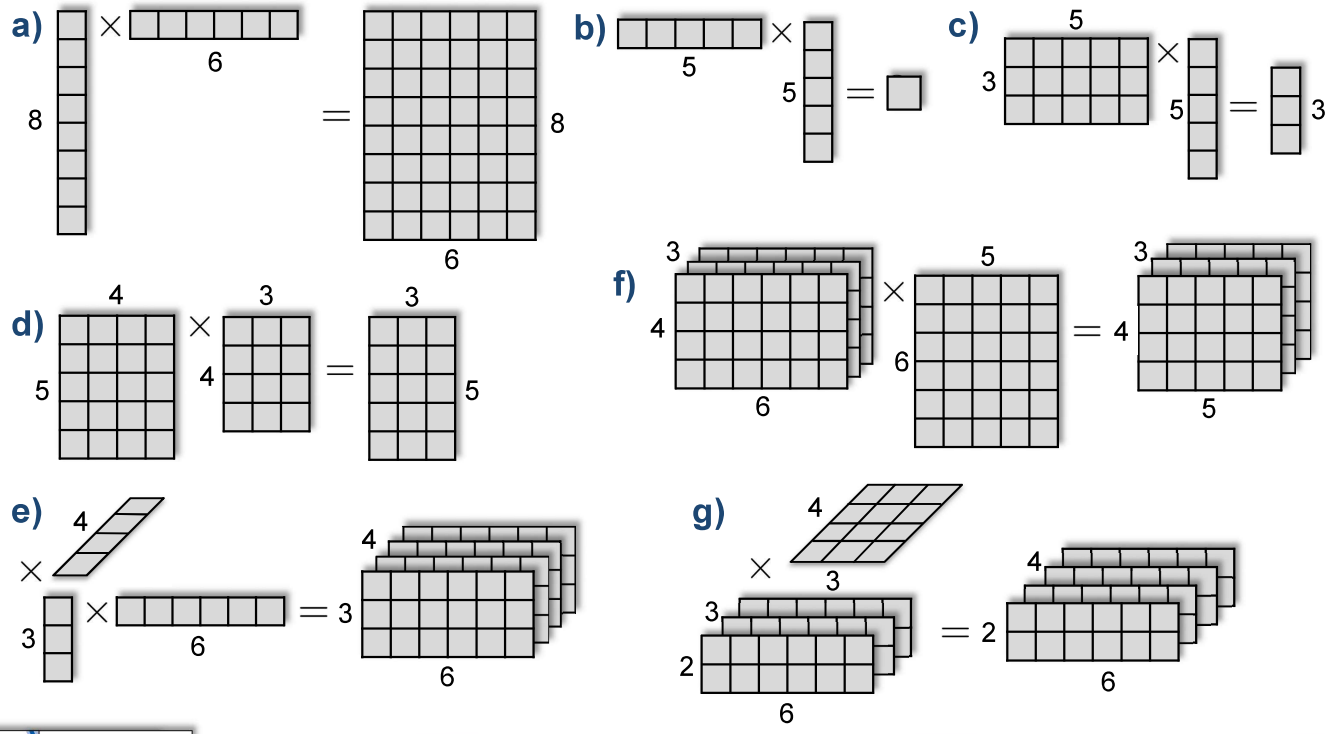
e) 

f) 

g) 

Tensors

- Some vector/matrix/tensor operations:



Generalizing the derivative

- Let $\boldsymbol{y} = f(\boldsymbol{x}),\ \boldsymbol{x} \in \mathbb{R}^D,\ \boldsymbol{y} \in \mathbb{R}^N$. *How can we* ***define*** $\frac{d\boldsymbol{y}}{d\boldsymbol{x}}$ *?*
- Let $\delta_{\boldsymbol{x}_0} : \begin{cases} \mathbb{R}^D & \to & \mathbb{R}^N \\ \boldsymbol{h} & \mapsto & f(\boldsymbol{x}_0 + \boldsymbol{h}) - f(\boldsymbol{x}_0) \end{cases}$ *"the variation of f when moving by a step $\boldsymbol{h}$ from $\boldsymbol{x}_0$."*
- $\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_0}$ is the **linear approx.** of $\delta_{\boldsymbol{x}_0}$ for $\boldsymbol{h}$ ***infinitesimally*** *small.*
- We call this the ***total derivative*** of f, or of $\boldsymbol{y}$, with resp. to $\boldsymbol{x}$, at $\boldsymbol{x}_0$.
- Since $\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0}$ is a **linear map** from $\mathbb{R}^D$ to $\mathbb{R}^N$, it can be *represented* by a **matrix**, $\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \mathbf{J}_{\boldsymbol{x}}[f](\boldsymbol{x}_0) \in \mathbb{R}^{N \times D}$, called the ***Jacobian***.

Formulas:

$$\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \begin{bmatrix} \left.\frac{\partial y_1}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_1}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_1}{\partial x_D}\right|_{x_{D,0}} \\ \left.\frac{\partial y_2}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_2}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_2}{\partial x_D}\right|_{x_{D,0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left.\frac{\partial y_N}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_N}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_N}{\partial x_D}\right|_{x_{D,0}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(x_{1,0})}{\partial x_1} & \frac{\partial f_1(x_{2,0})}{\partial x_2} & \cdots & \frac{\partial f_1(x_{D,0})}{\partial D_1} \\ \frac{\partial f_2(x_{1,0})}{\partial x_1} & \frac{\partial f_2(x_{2,0})}{\partial x_2} & \cdots & \frac{\partial f_2(x_{D,0})}{\partial D_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(x_{1,0})}{\partial x_1} & \frac{\partial f_N(x_{2,0})}{\partial x_2} & \cdots & \frac{\partial f_N(x_{D,0})}{\partial D_1} \end{bmatrix} = \mathbf{J}_{\boldsymbol{x}}[f](\boldsymbol{x}_0)$$

Special Case 1

What does it give for a simple function $f : \mathbb{R}^1 \to \mathbb{R}^1$?

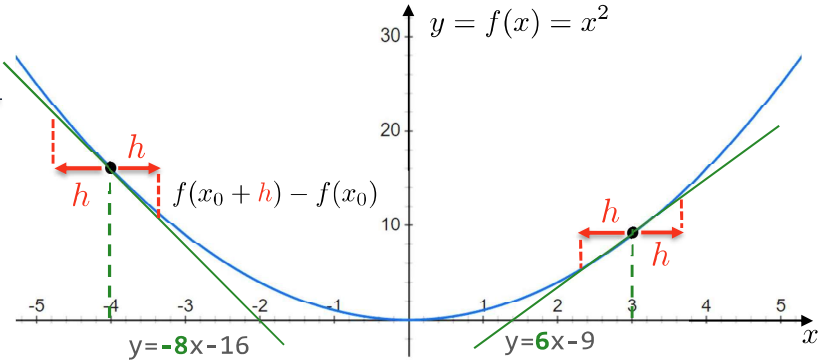
- In high-school we learn:

$$f'(x_0) = \lim_{h \to \infty} \frac{f(x_0 + h) - f(x_0)}{h}$$

- **Ex.** $y = f(x) = x^2$

$$f'(x_0) = 2x_0 = \left.\frac{dy}{dx}\right|_{x_0}$$

$$f'(3) = 6,\ f'(-4) = -8$$



- The derivative of f at x_0 may be viewed as the **linear map** that approximates

$$\delta_{x_0} : \left\{ \begin{array}{ccc} \mathbb{R}^1 & \to & \mathbb{R}^1 \\ h & \mapsto & f(x_0 + h) - f(x_0) \end{array} \right.$$, for h ***infinitesimally*** small.

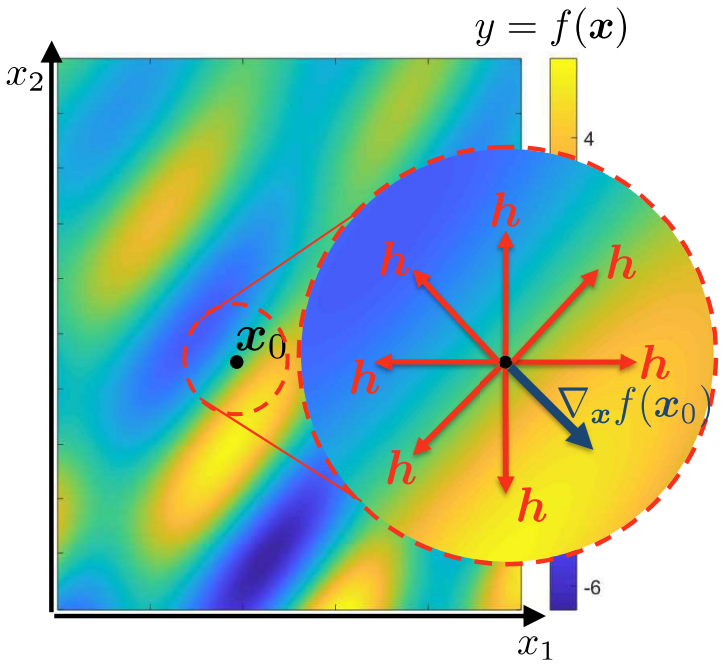
"the variation of f when moving to the left or to the right from x_0."

Indeed: $\lim_{h \to \infty} f(x_0 + h) - f(x_0) = f'(x_0)h$!

- As a **linear map**, $f'(x_0)$ can be seen as a **0-way tensor**, in $\mathbb{R}^{1 \times 1}$!

Special Case 2

What about a **real-valued** multivariate function $f : \mathbb{R}^D \to \mathbb{R}^1$?



- The ***total derivative*** of f at $\boldsymbol{x}_0$ is the **linear form** that approximates

$$\delta_{\boldsymbol{x}_0} : \left\{ \begin{array}{ccc} \mathbb{R}^D & \to & \mathbb{R}^1 \\ \boldsymbol{h} & \mapsto & f(\boldsymbol{x}_0 + \boldsymbol{h}) - f(\boldsymbol{x}_0) \end{array} \right.$$

for h infinitesimally small.

- As a linear form, it can be represented by a **row vector**: $\delta_{\boldsymbol{x}_0}(\boldsymbol{h}) \approx \boldsymbol{w}^\top \boldsymbol{h}$
- The **transpose** of this vector is called the ***gradient*** of f at $\boldsymbol{x}_0$ denoted:

$$\boldsymbol{w} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) \in \mathbb{R}^D$$

- It can be interpreted as the ***direction and rate of fastest increase*** of f at $\boldsymbol{x}_0$
- The formula is $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) = \left[\left.\frac{\partial y_1}{\partial x_1}\right|_{x_{1,0}}, \ldots, \left.\frac{\partial y_1}{\partial x_D}\right|_{x_{D,0}} \right]^\top = \left.\frac{dy}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0}^\top$

Summary (for $y = f(x)$)

Domain of x	Domain of y	Total Derivative at x_0
$\mathbb{R}$	$\mathbb{R}$	$\left.\frac{dy}{dx}\right\|_{x_0} = f'(x_0) \in \mathbb{R}^{1\times 1}$
$\mathbb{R}$	$\mathbb{R}^N$	$\left.\frac{d\boldsymbol{y}}{dx}\right\|_{x_0} = [f'_1(x_0), \dots, f'_N(x_0)]^\top \in \mathbb{R}^{N\times 1}$
$\mathbb{R}^D$	$\mathbb{R}$	$\left.\frac{dy}{d\boldsymbol{x}}\right\|_{\boldsymbol{x}_0} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)^\top \in \mathbb{R}^{1\times D}$
$\mathbb{R}^D$	$\mathbb{R}^N$	$\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right\|_{\boldsymbol{x}_0} = \mathbf{J}_{\boldsymbol{x}}[f](\boldsymbol{x}_0) \in \mathbb{R}^{N\times D}$

Note: All of this generalizes naturally to **matrix** or **tensor variables**, yielding **tensor total derivatives**.

The Chain Rule

- High school review: $(g \circ f)'(x_0) = (g' \circ f)(x_0) \cdot f'(x_0)$
- By letting $y = f(x)$ and $z = g(y) = g(f(x))$ this writes:

$$\left.\frac{dz}{dx}\right|_{x_0} = \left.\frac{dz}{dy}\right|_{f(x_0)} \cdot \left.\frac{dy}{dx}\right|_{x_0}$$

"the constant x_0 'flows' through the variables"

- This generalizes immediately to total derivatives!

Let:
- $\boldsymbol{x} \in \mathbb{R}^D$
- $y = f(\boldsymbol{x}) \in \mathbb{R}$
- $\boldsymbol{z} = g(y) = g(f(\boldsymbol{x})) \in \mathbb{R}^N$
- $\boldsymbol{w} = h(\boldsymbol{z}) = h(g(f(\boldsymbol{x}))) \in \mathbb{R}^M$

$$\underset{\mathbb{R}^{M\times D}}{\left.\frac{d\boldsymbol{w}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0}} = \underset{\mathbb{R}^{M\times N}}{\left.\frac{d\boldsymbol{w}}{d\boldsymbol{z}}\right|_{\boldsymbol{z}_0}} \times \underset{\mathbb{R}^{N\times 1}}{\left.\frac{d\boldsymbol{z}}{dy}\right|_{y_0}} \times \underset{\mathbb{R}^{1\times D}}{\left.\frac{dy}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0}}$$

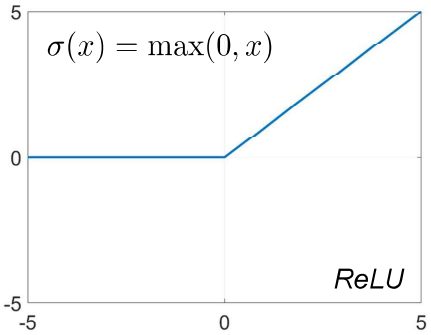
- In functional notations:

$$\mathbf{J}_{\boldsymbol{x}}[h \circ g \circ f](\boldsymbol{x}_0) = \mathbf{J}_{\boldsymbol{z}}[h](g \circ f(\boldsymbol{x}_0)) \times \begin{bmatrix} g'_1 \circ f(\boldsymbol{x}_0) \\ \vdots \\ g'_N \circ f(\boldsymbol{x}_0) \end{bmatrix} \times \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0)^\top$$

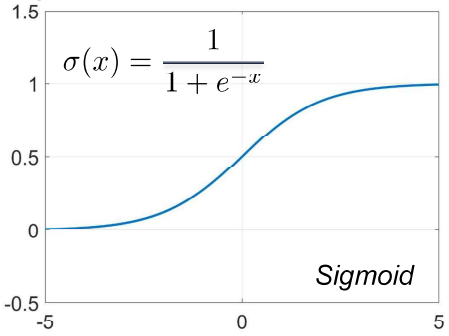
- All of this has natural generalizations to **tensors**
- All of this generalizes to ***partial derivatives***, e.g., $\left.\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \left.\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}\right|_{y_0} \times \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0}$

Exercises: *Simple Derivatives*

- Calculate the derivatives of the following functions:



$\sigma'(x) = ?$

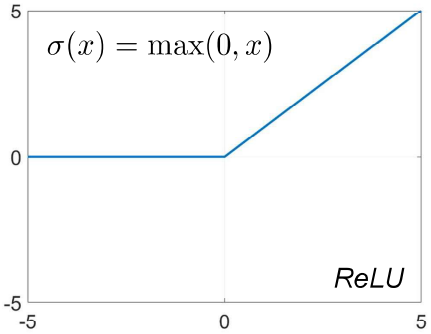


$\sigma'(x) = ?$

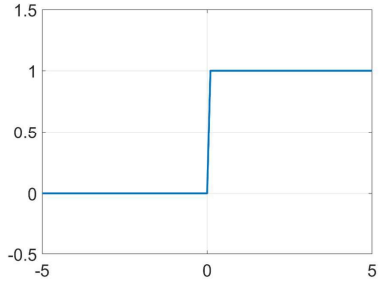
II. Background ▸ Exercises

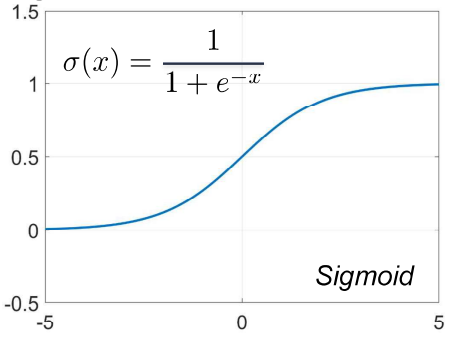
Exercises: *Simple Derivatives*

- Calculate the derivatives of the following functions:

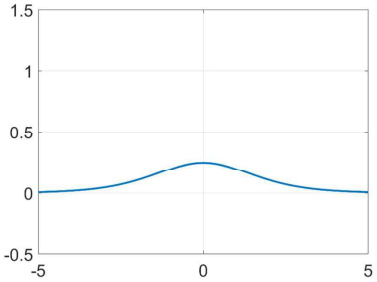


$$\sigma'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x < 1 \end{cases} = H(x)$$





$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x))$$



Exercises: *Gradients*

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) = \left[\frac{\partial f}{\partial x_1}(x_{0,1}), \ldots, \frac{\partial f}{\partial x_D}(x_{0,D})\right]^\top$$

- Calculate the gradient of the following functions at $\boldsymbol{x}_0$:

$$f(\boldsymbol{x}) = b + \boldsymbol{w}^\top \boldsymbol{x} = b + \sum_{d=1}^{D} w_d x_d \quad \Rightarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) = ?$$

$$g(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 = \sum_{d=1}^{D} (x_d)^2 \quad \Rightarrow \nabla_{\boldsymbol{x}} g(\boldsymbol{x}_0) = ?$$

II. Background ▸ Exercises

Exercises: *Gradients*

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) = \left[\frac{\partial f}{\partial x_1}(x_{0,1}), \ldots, \frac{\partial f}{\partial x_D}(x_{0,D})\right]^\top$$

- Calculate the gradient of the following functions at $\boldsymbol{x}_0$:

$$f(\boldsymbol{x}) = b + \boldsymbol{w}^\top \boldsymbol{x} = b + \sum_{d=1}^{D} w_d x_d \quad \Rightarrow \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_0) = [w_1, \ldots, w_D]^\top = \boldsymbol{w}$$

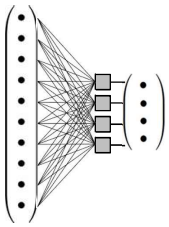
$$g(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 = \sum_{d=1}^{D} (x_d)^2 \quad \Rightarrow \nabla_{\boldsymbol{x}} g(\boldsymbol{x}_0) = [2x_{0,1}, \ldots, 2x_{0,D}]^\top = 2\boldsymbol{x}_0$$

Exercises: *Jacobians*

$$\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \begin{bmatrix} \left.\frac{\partial y_1}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_1}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_1}{\partial x_D}\right|_{x_{D,0}} \\ \left.\frac{\partial y_2}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_2}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_2}{\partial x_D}\right|_{x_{D,0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left.\frac{\partial y_N}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_N}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_N}{\partial x_D}\right|_{x_{D,0}} \end{bmatrix} = \begin{bmatrix} \nabla_{\boldsymbol{x}} f_1(\boldsymbol{x}_0)^\top \\ \nabla_{\boldsymbol{x}} f_2(\boldsymbol{x}_0)^\top \\ \vdots \\ \nabla_{\boldsymbol{x}} f_N(\boldsymbol{x}_0)^\top \end{bmatrix}$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}_0} \times \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0}$$

Remember: *the* ***simple perceptron***



$$\boldsymbol{x} \in \mathbb{R}^D \quad \boldsymbol{y} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) \in \mathbb{R}^N$$

$$= \sigma(\boldsymbol{a}) \text{ where } \boldsymbol{a} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} \hat{\boldsymbol{w}}_1^\top \boldsymbol{x} \\ \vdots \\ \hat{\boldsymbol{w}}_N^\top \boldsymbol{x} \end{bmatrix} + \boldsymbol{b} = \sum_{d=1}^{D} \boldsymbol{w}_d x_d + \boldsymbol{b}$$

- Calculate the following Jacobians:

$$\left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \qquad \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}}\right|_{\boldsymbol{x}_0} = \qquad \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{w}_d}\right|_{\boldsymbol{x}_0} = \qquad \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}_0} =$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} =$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{b}}\right|_{\boldsymbol{b}_0} =$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_d}\right|_{\boldsymbol{w}_d} =$$

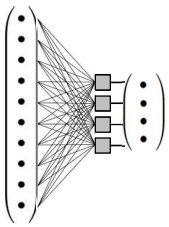
II. Background ▸ Exercises

Exercises: *Jacobians*

$$\left.\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \begin{bmatrix} \left.\frac{\partial y_1}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_1}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_1}{\partial x_D}\right|_{x_{D,0}} \\ \left.\frac{\partial y_2}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_2}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_2}{\partial x_D}\right|_{x_{D,0}} \\ \vdots & \vdots & \ddots & \vdots \\ \left.\frac{\partial y_N}{\partial x_1}\right|_{x_{1,0}} & \left.\frac{\partial y_N}{\partial x_2}\right|_{x_{2,0}} & \cdots & \left.\frac{\partial y_N}{\partial x_D}\right|_{x_{D,0}} \end{bmatrix} = \begin{bmatrix} \nabla_{\boldsymbol{x}} f_1(\boldsymbol{x}_0)^\top \\ \nabla_{\boldsymbol{x}} f_2(\boldsymbol{x}_0)^\top \\ \vdots \\ \nabla_{\boldsymbol{x}} f_N(\boldsymbol{x}_0)^\top \end{bmatrix}$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}_0} \times \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0}$$

Remember: *the* ***simple perceptron***



$$\boldsymbol{x} \in \mathbb{R}^D \quad \boldsymbol{y} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) \in \mathbb{R}^N$$

$$= \sigma(\boldsymbol{a}) \text{ where } \boldsymbol{a} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} \hat{\boldsymbol{w}}_1^\top \boldsymbol{x} \\ \vdots \\ \hat{\boldsymbol{w}}_N^\top \boldsymbol{x} \end{bmatrix} + \boldsymbol{b} = \sum_{d=1}^{D} \boldsymbol{w}_d x_d + \boldsymbol{b}$$

- Calculate the following Jacobians:

$$\left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \begin{bmatrix} \hat{\boldsymbol{w}}_1^\top \\ \vdots \\ \hat{\boldsymbol{w}}_N^\top \end{bmatrix} = \mathbf{W} \qquad \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}}\right|_{\boldsymbol{x}_0} = \mathbf{I}_N \qquad \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{w}_d}\right|_{\boldsymbol{x}_0} = x_d \mathbf{I}_N \qquad \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}_0} = \begin{bmatrix} \sigma'(a_{0,1}) & 0 & \cdots & 0 \\ 0 & \sigma'(a_{0,2}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma'(a_{0,N}) \end{bmatrix} = \mathrm{diag}[\sigma'(\boldsymbol{a}_0)]$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\mathbf{W}\boldsymbol{x}_0 + \boldsymbol{b}} \times \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_0} = \mathrm{diag}[\sigma'(\mathbf{W}\boldsymbol{x}_0 + \boldsymbol{b})]\mathbf{W}$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{b}}\right|_{\boldsymbol{b}_0} = \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\mathbf{W}\boldsymbol{x} + \boldsymbol{b}_0} \times \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{b}}\right|_{\boldsymbol{b}_0} = \mathrm{diag}[\sigma'(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}_0)]$$

$$\left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}_d}\right|_{\boldsymbol{w}_d} = \left.\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{a}}\right|_{\mathbf{W}_d \boldsymbol{x} + \boldsymbol{b}} \times \left.\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{w}_d}\right|_{\boldsymbol{w}_d} = \mathrm{diag}[\sigma'(\mathbf{W}_d \boldsymbol{x} + \boldsymbol{b})] x_d$$

Exercise: *Least Squares*

- Calculate the gradient of $f(\boldsymbol{\theta}) = \|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2$ where $\mathbf{W} \in \mathbb{R}^{N \times D}$

Hint: Let $\boldsymbol{a} = \mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}$, $g(\boldsymbol{a}) = \|\boldsymbol{a}\|_2^2$ and $r = g(\boldsymbol{a}) = f(\boldsymbol{\theta})$.

$$\nabla_\theta f(\boldsymbol{\theta}_0)^\top = \frac{\partial r}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_0} = ?$$

II. Background ▸ Exercises

Exercise: *Least Squares*

- Calculate the gradient of $f(\boldsymbol{\theta}) = \|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2$ where $\mathbf{W} \in \mathbb{R}^{N \times D}$

Hint: Let $\boldsymbol{a} = \mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}$, $g(\boldsymbol{a}) = \|\boldsymbol{a}\|_2^2$ and $r = g(\boldsymbol{a}) = f(\boldsymbol{\theta})$.

$$\nabla_\theta f(\boldsymbol{\theta}_0)^\top = \frac{\partial r}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_0} = \frac{\partial r}{\partial \boldsymbol{a}}\Big|_{\mathbf{W}\boldsymbol{\theta}_0 - \boldsymbol{y}} \times \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}_0}$$

$$= \nabla_{\boldsymbol{a}} g(\mathbf{W}\boldsymbol{\theta}_0 - \boldsymbol{y})^\top \times \mathbf{W} \qquad \text{We have: } \nabla_{\boldsymbol{a}} g(\boldsymbol{a}_0) = 2\boldsymbol{a}_0$$

$$= 2(\mathbf{W}\boldsymbol{\theta}_0 - \boldsymbol{y})^\top \mathbf{W}$$

$$\nabla_\theta f(\boldsymbol{\theta}_0) = 2\mathbf{W}^\top(\mathbf{W}\boldsymbol{\theta}_0 - \boldsymbol{y})$$

École d'ingénieurs Télécom Physique Université de Strasbourg Inria Antoine.Deleforge@inria.fr Artificial Intelligence & Deep Learning 99/200

OUTLINE

Recap: the *model fitting* approach to Machine Learning

- f_θ = *a jean*
- θ = *its (width, length)*
- $\mathcal{F} = \{f_\theta\}_{\theta \in \Theta}$ *the shelves*

•Given a **parameterized family** $\mathcal{F}$ of models == functions

Ex: *a DNN with* $f_\theta : \mathbb{R}^D \to \mathbb{R}^{N^{(\text{out})}}$

•Given a **training dataset** $\mathcal{T}$ *(your legs!),*

•Given a ***total loss function*** $L(f_\theta, \mathcal{T})$ that measures the ***fit*** of a given model f_θ to the **full dataset**, for the given task (the smaller the better),

→ We want to **minimize** the loss with respect to the **parameters** $\theta \in \Theta$ **:**

$$\hat{f} = f_{\hat{\theta}} \quad \text{where} \quad \hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}}\, L(f_\theta, \mathcal{T})$$

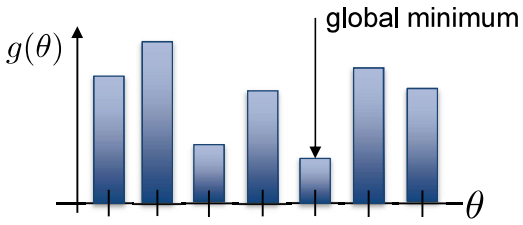
For conciseness we will use

$g(\theta) \overset{\text{def}}{=} L(f_\theta, \mathcal{T}) \quad (g : \Theta \to \mathbb{R})$

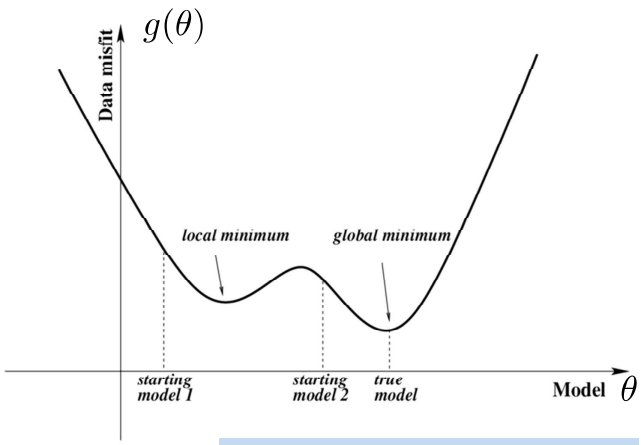
in the next slides.

Domain of the function

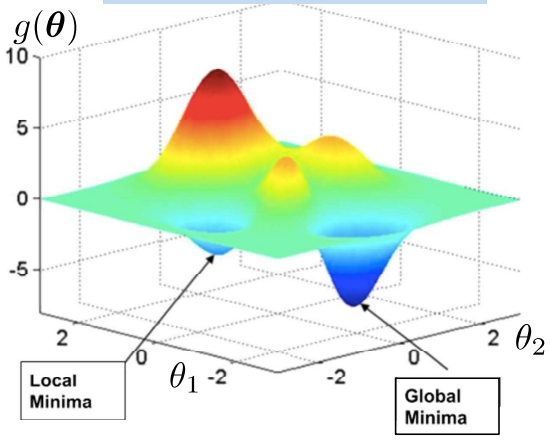
Discrete: $\theta \in \{\theta_1, \ldots, \theta_C\}$



1D continuous: $\theta \in \mathbb{R}$



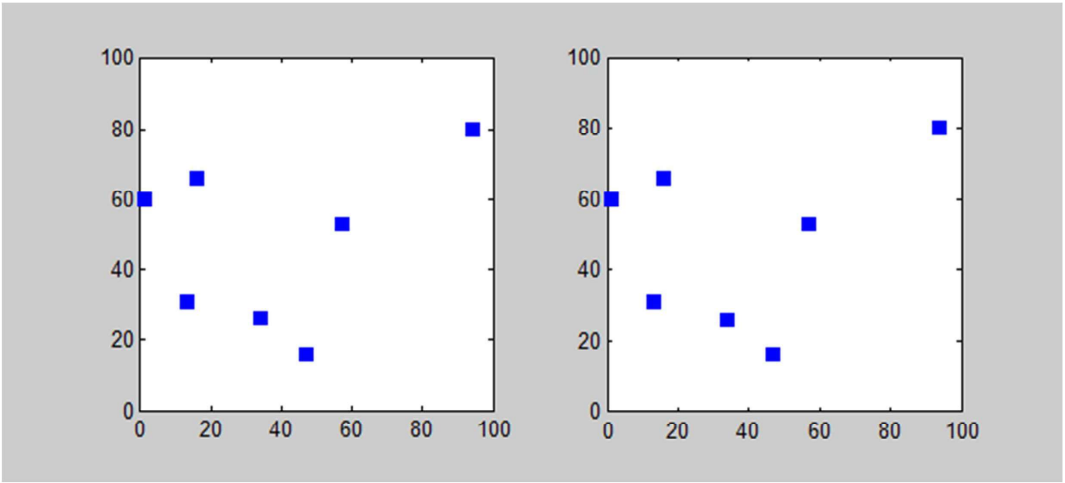
2D continuous: $\theta \in \mathbb{R}^2$



D-D continuous: $\theta \in \mathbb{R}^D$

Mixed: $\theta \in \{0,1\} \times \mathbb{R}^D \times [0,1] \times \mathbb{R}^+$ ...

1. Brute Force / Random / Grid Search

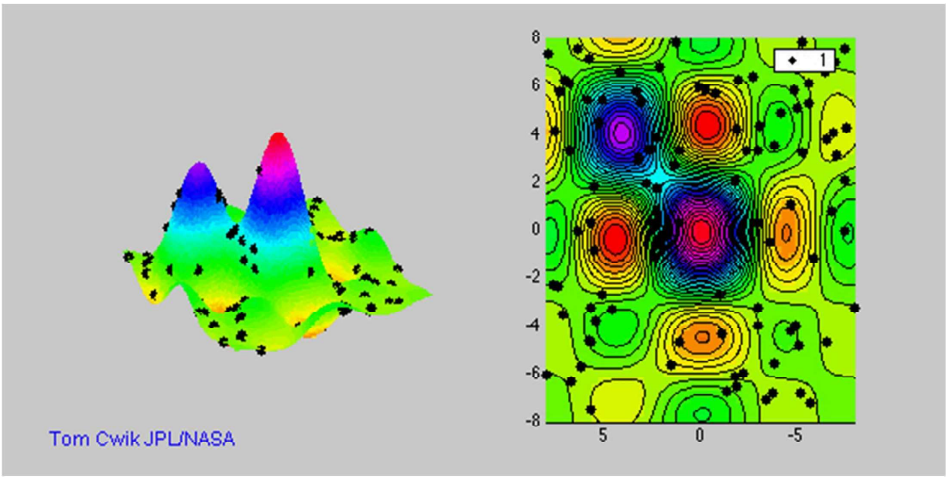


What is g ? What is θ ? What is Θ ?

- Sometimes best when optimizing on a **small discrete set** of parameters
- **Ex:** DNN **architectures** or ***hyperparameters***

2. "Population-Based" Algorithms

- Evolutionary/Genetic algorithms
- Particle Swarms
- Ant Colonies



What is g ? What is θ ? What is Θ ?

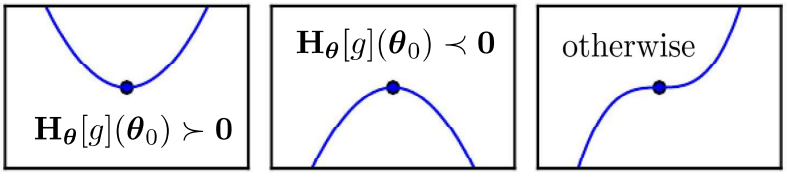
- Principle = Evolve a population.
- Strongly inspired by **nature** or **physics**
- Can be powerful and work on very general functions, but ***heuristic***

3. Calculating "*zeroes*" of the gradient

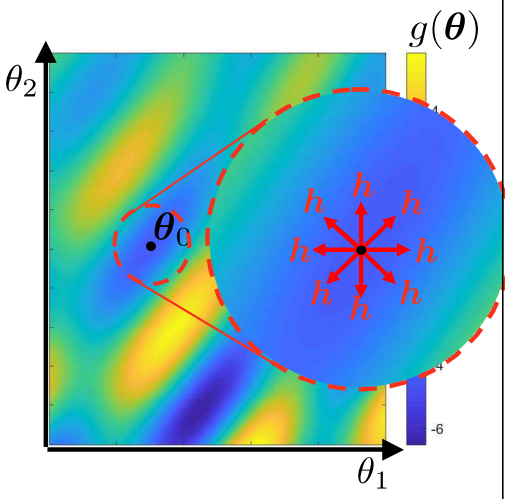
- We call ***zero*** of the gradient a point $\boldsymbol{\theta}_0 \in \mathbb{R}^P$ such that $\nabla_{\boldsymbol{x}} g(\boldsymbol{\theta}_0) = \mathbf{0}_P$.

- Also called ***stationary points*** of g : the points where g is ***locally constant***, i.e., *"flat"*.

- They may correspond to:



local minimum | local maximum | saddle point



- In case of doubt, it is possible to distinguish between the 3 by looking at the **Hessian** $\mathbf{H}_{\boldsymbol{\theta}}[g](\boldsymbol{\theta}_0) \in \mathbb{R}^{P \times P}$ of g at $\boldsymbol{\theta}_0$:

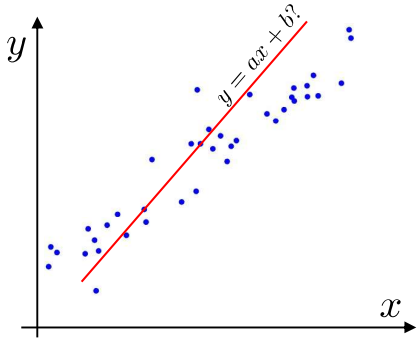
$$\mathbf{H}_{\boldsymbol{\theta}}[g](\boldsymbol{\theta}_0) \stackrel{\text{def}}{=} \mathbf{J}_{\boldsymbol{\theta}}[\nabla_{\boldsymbol{\theta}} g](\boldsymbol{\theta}_0)$$

"Second order derivative of g"

Only works if $\operatorname{Det} \mathbf{H}_{\boldsymbol{\theta}}[g](\boldsymbol{\theta}_0) \neq 0$

3. Calculating "*zeroes*" of the gradient

Exercise: Fitting an **affine model** via ***least squares***



- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t) - y_t)^2$

- Find $\boldsymbol{\theta}_0$ such that $\nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0}_2$ **Hint:** we *already* calculated $\nabla_\theta g(\boldsymbol{\theta}_0)$!

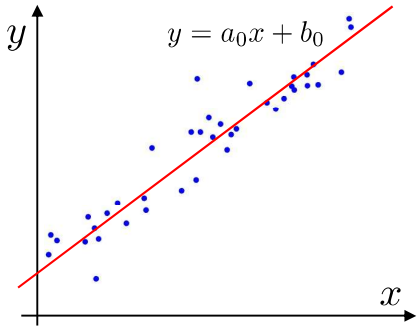
$g(\boldsymbol{\theta}) = ?$

$\nabla_\theta g(\boldsymbol{\theta}_0) = ?$

III. Fitting a Model ▸ How to minimize a function?

3. Calculating "*zeroes*" of the gradient

Exercise: Fitting an **affine model** via ***least squares***



- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t) - y_t)^2$

- Find $\boldsymbol{\theta}_0$ such that $\nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0}_2$ **Hint:** we *already* calculated $\nabla_\theta g(\boldsymbol{\theta}_0)$!

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}(ax_t + b - y_t)^2 = \frac{1}{T}\sum_{t=1}^{T}\left(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix}a\\b\end{bmatrix} - y_t\right)^2 = \frac{1}{T}\sum_{t=1}^{T}\left(\boldsymbol{w}_t^\top\boldsymbol{\theta} - y_t\right)^2$$

$$= \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \text{where} \quad \mathbf{W} = [\boldsymbol{w}_1^\top, \dots, \boldsymbol{w}_T^\top]^\top \in \mathbb{R}^{T\times 2}, \quad \boldsymbol{y} = [y_1, \dots, y_T]^\top \in \mathbb{R}^T$$

$$\nabla_\theta g(\boldsymbol{\theta}_0) = 2\mathbf{W}^\top(\mathbf{W}\boldsymbol{\theta}_0 - \boldsymbol{y}) = \mathbf{0}_2 \quad \Rightarrow \quad \boxed{\boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y} = \mathbf{W}^\dagger\boldsymbol{y}}$$

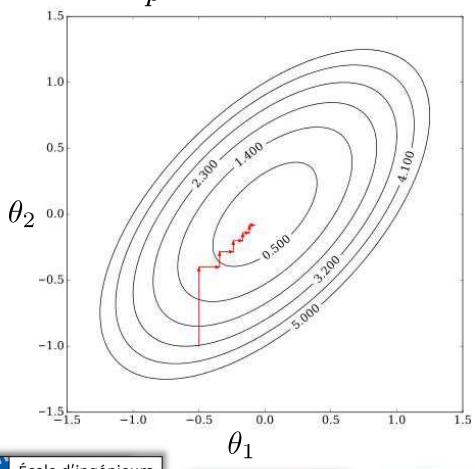
4. Alternated Minimization

$$\underset{\theta_1,\dots,\theta_P}{\operatorname{argmin}}\ g(\theta_1,\dots,\theta_P)?$$

Converges, but not necessarily to the global minimum

$$\begin{aligned}
\theta_1^{(i+1)} &= \underset{\theta_1}{\operatorname{argmin}}\ g(\theta_1, \theta_2^{(i)}, \dots, \theta_P^{(i)}) \\
\theta_2^{(i+1)} &= \underset{\theta_2}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2, \dots, \theta_P^{(i)}) \\
&\vdots \qquad\qquad \vdots \\
\theta_P^{(i+1)} &= \underset{\theta_P}{\operatorname{argmin}}\ g(\theta_1^{(i+1)}, \theta_2^{(i+1)}, \dots, \theta_P)
\end{aligned}$$

- For θ_p scalar: *coordinate descent*

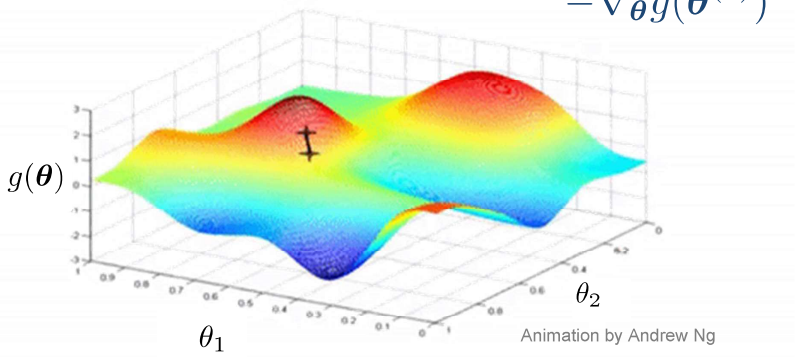


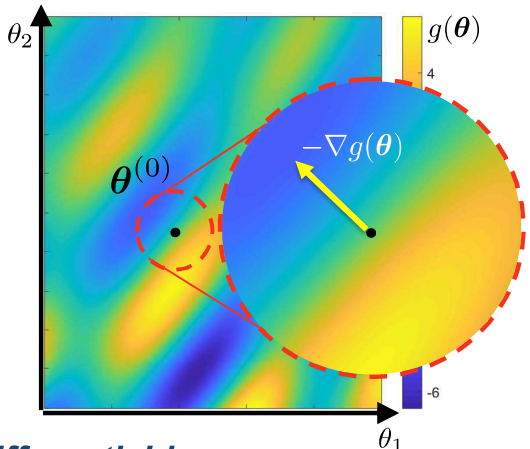
- Convenient when:
  - Variables are mixed discrete / continuous
  - There are direct solutions wrt. each variable
- Sometimes, introducing **new variables** and **then** performing AM yields efficient algorithms, e.g., *Expectation-Maximization (EM)* or *ADMM*
- Variant: Alternate between **minimization** and **projection** onto *constraints* (e.g.: $\theta \geq 0$)

5. Gradient Descent

Intuition:
- Start from an initial **parameter vector** $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^P$
- From here, follow the ***direction of steepest descent***
- Stop when things look ***flat***

$$-\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(0)})$$

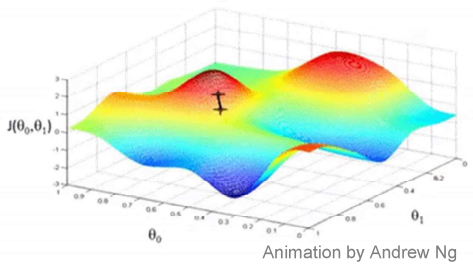




- The updates are: $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)})$
- Requires the function to be (almost everywhere) ***differentiable***

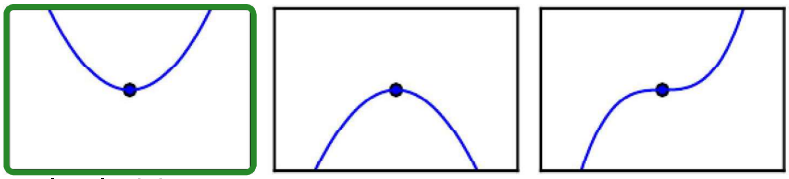
5. Gradient Descent

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \epsilon \nabla_\theta g(\theta^{(i)})$$

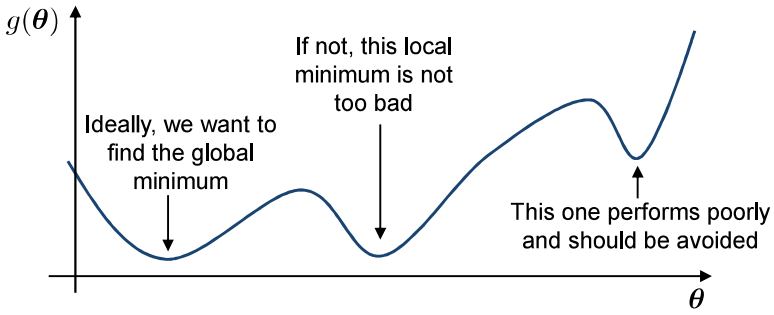


Animation by Andrew Ng

- Local maxima and saddle points are ***unstable fixed points***, while local minima are ***stable fixed points***


local minimum | local maximum | saddle point

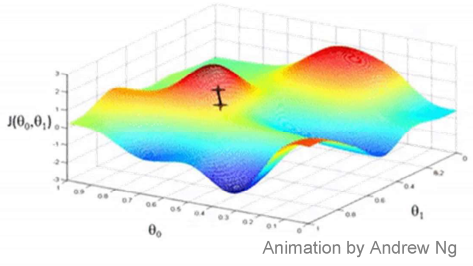
→ ***The algorithm converges to local minima under mild assumptions*** ☺



- ***Spurious local minima*** cannot always be avoided
- Many variants have been derived to limit them, and to speed up convergence

5. Gradient Descent

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \epsilon \nabla_\theta g(\theta^{(i)})$$



Animation by Andrew Ng

- The ***gradient-step*** ϵ, also called ***learning rate***, is a critical ***hyper-parameter*** of the algorithm.

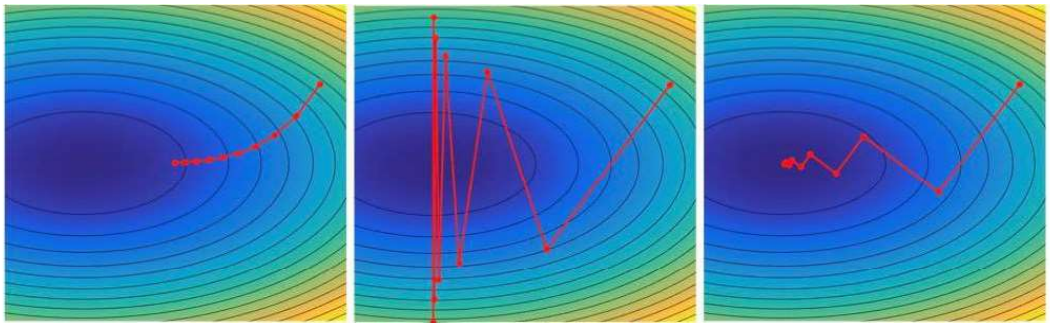

Small ϵ | Large ϵ | Optimal ϵ

Image by Gabriel Peyre

- Choosing a small ϵ is always the **safest**, but might result in **slow** convergence
- There exists many variations on gradient descent. We will cover some of them later in this chapter.

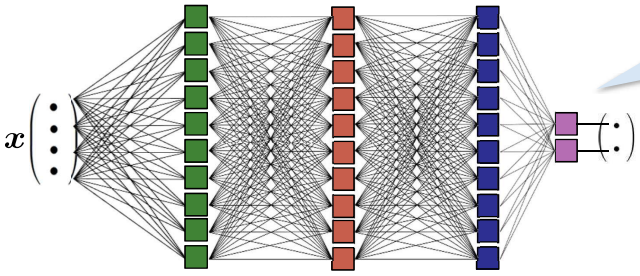
Summary of optimization techniques

1. **Brute Force / Random / Grid Search** : Useful when searching among **discrete** parameters. Quickly **explodes** in complexity.
2. **Population-Based algorithms** : Versatile but **heuristic**.
3. **Directly finding zeroes of the gradient** : Very efficient (**not** iterative) but only possible with a **limited** number of functions.
4. **Alternate minimization** : A **very general** family of principled methods. Allows **combining** multiple techniques. Often **no hyperparameters**. Needs to be designed on a **case-by-case** basis. Global convergence is generally **not guaranteed**.
5. **Gradient descent** : Works on **any differentiable functions**. Convergence to **local minima**. The learning rate is a **critical hyperparameter**.

Back to Neural Networks

- Neural network models are **fitted** using variants of **gradient descent**.

Remember: *A **deep feedforward neural network***



How to compute $\nabla_{\boldsymbol{\theta}} L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T})$ *?!* 😱

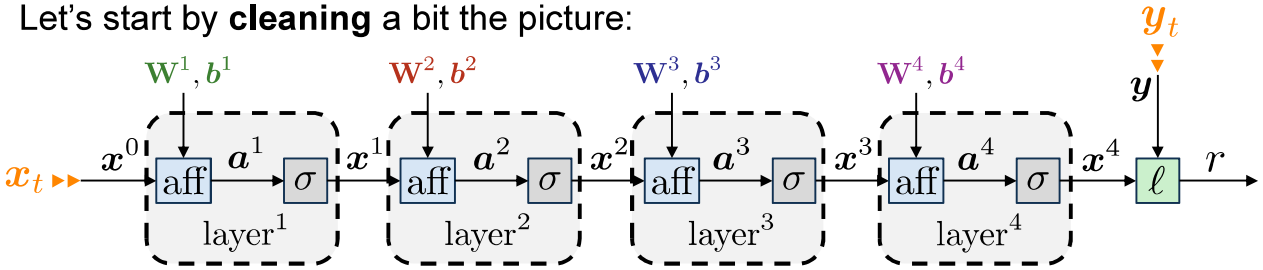
$$\boldsymbol{y} = \mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x})$$
$$= \sigma\left(\boldsymbol{b}^4 + \mathbf{W}^4\sigma\left(\boldsymbol{b}^3 + \mathbf{W}^3\sigma\left(\boldsymbol{b}^2 + \mathbf{W}^2\sigma\left(\boldsymbol{b}^1 + \mathbf{W}^1\boldsymbol{x}\right)\right)\right)\right)$$
$$\boldsymbol{\theta} = [\boldsymbol{\theta}^i]_{i=1}^4 = [(\boldsymbol{b}^i, \mathbf{W}^i)]_{i=1}^4$$

- Given a **training dataset** of *input* ↔ *output* $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^T$, the general goal is to adjust $\boldsymbol{\theta}$ so that $\boldsymbol{y}_t \approx \mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$.
- We use a **total loss** of this form: $L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right)$, where ℓ is simply called ***the loss*** of the DNN.

  For example: $\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2$ *, the so called "L2 loss" or "Euclidean loss".*
- Losses of the form L are called ***Empirical Risk***, where the ***Risk*** of the model is defined as $\mathcal{R}(\mathrm{dnn}_{\boldsymbol{\theta}}) \overset{\text{def}}{=} \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}\{\ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{Y})\} \approx L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T})$.
- We will focus next on **supervised learning**, but the approach is more general.

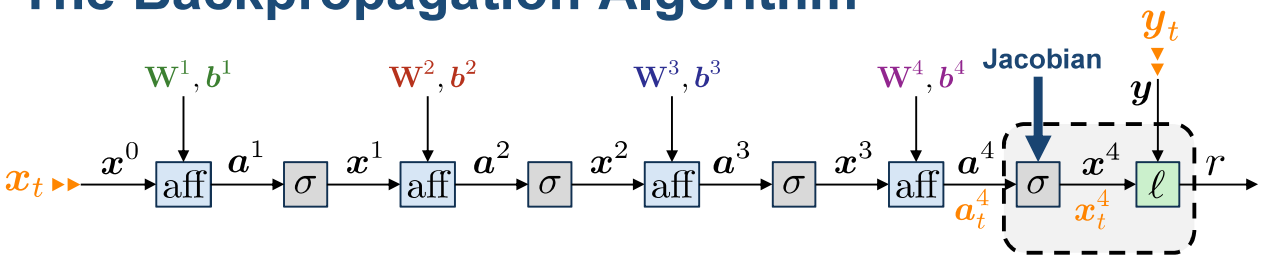
The Backpropagation Algorithm

Let's start by **cleaning** a bit the picture:



- $\boldsymbol{a}^i = \mathrm{aff}_{\boldsymbol{\theta}^i}(\boldsymbol{x}^{i-1}) = \mathbf{W}^i \boldsymbol{x}^{i-1} + \boldsymbol{b}^i$ are the **pre-activations**.
- $\boldsymbol{x}^i = \sigma(\boldsymbol{a}^i) = \sigma(\mathrm{aff}_{\boldsymbol{\theta}^i}(\boldsymbol{x}^{i-1})) = \mathrm{layer}^i(\boldsymbol{x}^{i-1})$ are the **activations**.
- The **loss** ℓ can be viewed as another layer, with **real output** r (the *"residual"*).
- By **linearity** of the gradient, we have: $\nabla_{\boldsymbol{\theta}} L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$.
- Hence, it is enough to calculate the gradient of the loss for **one sample** $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, i.e., $\boldsymbol{G}_{\boldsymbol{\theta}} \overset{\text{def}}{=} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$.
- The **Backpropagation Algorithm** (*"Backprop"*) is an efficient way to do this.

The Backpropagation Algorithm



*The trick is to **recursively calculate** the gradient of the **loss** with respect to both the **parameters** and **activations**, going **backwards** from the end, using the **chain rule**.*

0) We start by $\boldsymbol{G}_{\boldsymbol{x}^4} = \left.\frac{\partial r}{\partial \boldsymbol{x}^4}\right|_{\boldsymbol{x}_t^4}^{\top} = \nabla_{\boldsymbol{x}^4} \ell(\boldsymbol{x}_t^4, \boldsymbol{y}_t)$.

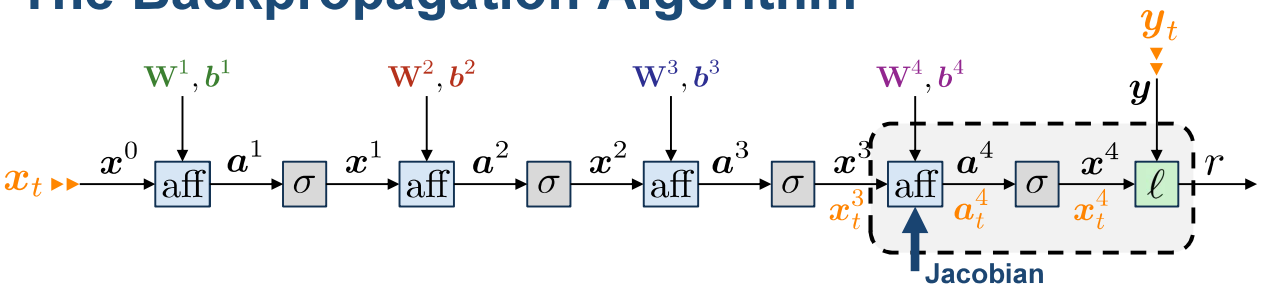
For example, for the L2 loss $\ell(\boldsymbol{x}^4, \boldsymbol{y}_t) = \|\boldsymbol{x}^4 - \boldsymbol{y}_t\|_2^2$, we have $\boldsymbol{G}_{\boldsymbol{x}^4} = 2(\boldsymbol{x}_t^4 - \boldsymbol{y}_t)$, the **difference** between the network **prediction** and the **target**.

1) Then, $\boldsymbol{G}_{\boldsymbol{a}^4} = \left.\frac{\partial r}{\partial \boldsymbol{a}^4}\right|_{\boldsymbol{a}_t^4}^{\top} = \left(\left.\frac{\partial r}{\partial \boldsymbol{x}^4}\right|_{\boldsymbol{x}_t^4} \times \left.\frac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\right|_{\boldsymbol{a}_t^4}\right)^{\top} = \left.\frac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\right|_{\boldsymbol{a}_t^4}^{\top} \times \boldsymbol{G}_{\boldsymbol{x}^4}$

$= \sigma'(\boldsymbol{a}_t^4) \odot \boldsymbol{G}_{\boldsymbol{x}^4}$.

$\left.\frac{\partial \boldsymbol{x}^4}{\partial \boldsymbol{a}^4}\right|_{\boldsymbol{a}_t^4}^{\top} = \mathrm{diag}[\sigma'(\boldsymbol{a}_t^4)]$!

The Backpropagation Algorithm

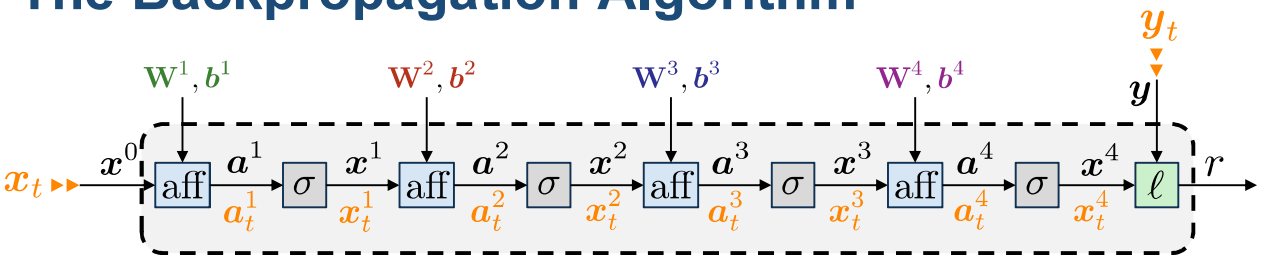


2) Then:

- $\boldsymbol{G}_{\boldsymbol{x}^3} = \dfrac{\partial r}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{x}^3}\Big|_{\boldsymbol{x}_t^3}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = \mathbf{W}^{4\top}\boldsymbol{G}_{\boldsymbol{a}^4}$.
- $\boldsymbol{G}_{\boldsymbol{b}^4} = \dfrac{\partial r}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{b}^4}\Big|_{\boldsymbol{b}^4}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = \boldsymbol{G}_{\boldsymbol{a}^4}$.
- $\boldsymbol{G}_{\boldsymbol{w}_d^4} = \dfrac{\partial r}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top = \left(\dfrac{\partial r}{\partial \boldsymbol{a}^4}\Big|_{\boldsymbol{a}_t^4} \times \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}\right)^\top = \dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top \times \boldsymbol{G}_{\boldsymbol{a}^4} = x_{t,d}^3 \boldsymbol{G}_{\boldsymbol{a}^4}$.

$\dfrac{\partial \boldsymbol{a}^4}{\partial \boldsymbol{w}_d^4}\Big|_{\boldsymbol{w}_d^4}^\top = x_{t,d}^3 \mathbf{I}$

The Backpropagation Algorithm



And so on…

3) $\boldsymbol{G}_{\boldsymbol{a}^3} = \sigma'(\boldsymbol{a}_t^3) \odot \boldsymbol{G}_{\boldsymbol{x}^3}$. 5) $\boldsymbol{G}_{\boldsymbol{a}^2} = \sigma'(\boldsymbol{a}_t^2) \odot \boldsymbol{G}_{\boldsymbol{x}^2}$. 7) $\boldsymbol{G}_{\boldsymbol{a}^1} = \sigma'(\boldsymbol{a}_t^1) \odot \boldsymbol{G}_{\boldsymbol{x}^1}$.

4) $\boldsymbol{G}_{\boldsymbol{x}^2} = \mathbf{W}^{3\top}\boldsymbol{G}_{\boldsymbol{a}^3}$.
$\boldsymbol{G}_{\boldsymbol{b}^3} = \boldsymbol{G}_{\boldsymbol{a}^3}$.
$\boldsymbol{G}_{\boldsymbol{w}_d^3} = x_{t,d}^2 \boldsymbol{G}_{\boldsymbol{a}^3}$.

6) $\boldsymbol{G}_{\boldsymbol{x}^1} = \mathbf{W}^{2\top}\boldsymbol{G}_{\boldsymbol{a}^2}$.
$\boldsymbol{G}_{\boldsymbol{b}^2} = \boldsymbol{G}_{\boldsymbol{a}^2}$.
$\boldsymbol{G}_{\boldsymbol{w}_d^2} = x_{t,d}^1 \boldsymbol{G}_{\boldsymbol{a}^2}$.

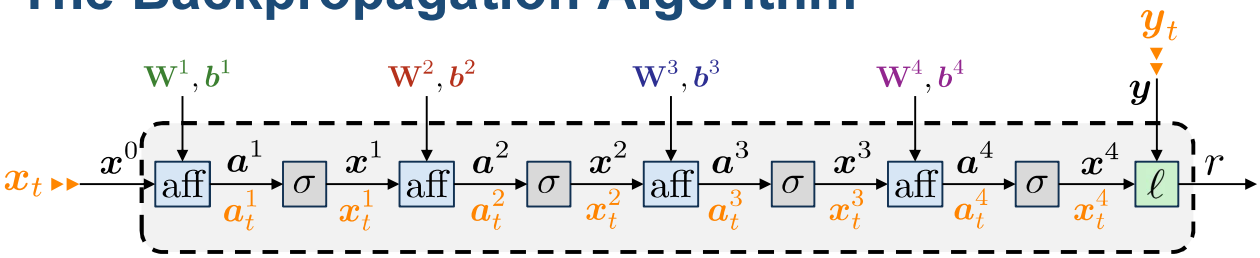
8) $\boldsymbol{G}_{\boldsymbol{x}^0} = \mathbf{W}^{1\top}\boldsymbol{G}_{\boldsymbol{a}^1}$.
$\boldsymbol{G}_{\boldsymbol{b}^1} = \boldsymbol{G}_{\boldsymbol{a}^1}$.
$\boldsymbol{G}_{\boldsymbol{w}_d^1} = x_{t,d}^0 \boldsymbol{G}_{\boldsymbol{a}^1}$.

In Fine:

$$\boldsymbol{G}_{\boldsymbol{x}^0} = \mathbf{W}^{1\top}\sigma'(\boldsymbol{a}_t^1) \odot \mathbf{W}^{2\top}\sigma'(\boldsymbol{a}_t^2) \odot \mathbf{W}^{3\top}\sigma'(\boldsymbol{a}_t^3) \odot \mathbf{W}^{4\top}\sigma'(\boldsymbol{a}_t^4) \odot \nabla_{\boldsymbol{x}^4}\ell(\boldsymbol{x}_t^4, \boldsymbol{y}_t).$$

⇨ Using the parameters' gradients, we **update** them via **gradient descent**.

The Backpropagation Algorithm



Conclusions

- The idea is **very general** and can be applied to **any feedforward** neural network architecture
- **There are 4 key ingredients**:
  - the **data** (constants)
  - the **parameters** (*free* variables to optimize)
  - the **activations** / **layer outputs** (*dependent* variables)
  - the **functions** / **layers** (layers are generally compositions of functions)
- The data flows **forwards** while the gradient propagates **backwards**, a bit like another neural network, with only vector/matrix multiplications
- All we need are the **forward operators** and **Jacobians** of each module

Back to gradient descent

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)})$$

- Remember that we train DNNs using Empirical Risk Minimization:

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right) \approx \mathbb{E}_{\boldsymbol{X},\boldsymbol{Y}}\{\ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{X}), \boldsymbol{Y})\}$$

- We compute the gradient of the **total loss** by summing the gradients of the **loss** at individual data samples:

$$\nabla_{\boldsymbol{\theta}} L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$$

- But doing so across the **entire dataset** (e.g.: 1 million images) for **every gradient step** would be very expansive.

Stochastic Gradient Descent

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)})$$

(The SGD algorithm)

- At each iteration (i), compute the gradient over a **random subset** $\mathcal{T}^{(i)} \subseteq \mathcal{T}$ and perform one step of gradient descent:

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \cdot \frac{1}{T} \sum_{(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}_i} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$$

- At the next iteration, pick another (disjoint) random subset
- When the entire dataset has passed, start over again
- Each $\mathcal{T}^{(i)}$ is called a **minibatch**
- Each pass over the entire dataset is called an **epoch**

Splitting the training set into B minibatches:

- Reduces the computation cost of one gradient by a factor of B
- Increases the **standard deviation** on the gradient estimate by a factor of $\sqrt{B}$ only.

More iterations but fewer epochs
= less total computation

Stochastic Gradient Descent

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)})$$

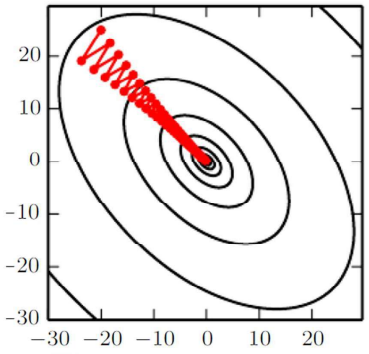
(The SGD algorithm)

- In practice the gradients $\ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t)$ of all examples $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ are computed in **parallel** using a **graphical processing unit** (GPU) and summed up within a minibatch

- The choice of the minibatch size is governed by these considerations:
  - The minibatch data and computations must fit in **GPU memory**
  - Too small minibatches do **not exploit well** GPU capabilities
  - Some kinds of hardware perform better with **power-of-2** sizes

- Typical minibatch sizes: from 32 to 256.

- **Limit of SGD:** Tends to "zigzag" when descending a "canyon", which increases the number of iterations



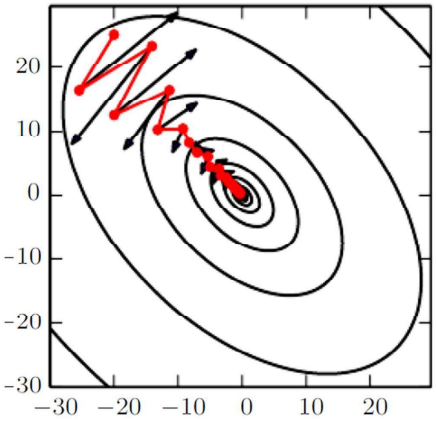
SGD with Momentum

$$\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \epsilon \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}^{(i)})$$

- **Solution:** "smooth" the gradient estimates across several iterations.
- **Momentum** = vector $\boldsymbol{v}$ representing the direction and speed at which the parameters move through parameter space.
- Defined as an ***exponentially decaying average*** of the negative gradient.
- **SGD with momentum**: initialize $\boldsymbol{v}^{(0)} = \mathbf{0}$, then replace each iteration of SGD by:

$$\begin{cases} \boldsymbol{v}^{(i+1)} \leftarrow \alpha \boldsymbol{v}^{(i)} - \epsilon \cdot \frac{1}{T} \sum_{(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}_i} \nabla_{\boldsymbol{\theta}} \ell(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t) \\ \boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} + \boldsymbol{v}^{(i)} \end{cases}$$

- The very popular **ADAM optimizer** (**140k** citations since 2014!) extends this idea by also averaging **squared** gradients.



Converges faster than SGD

Local Minima

When properly tuned (learning rate not too large nor too small), SGD converges to a **local minimum**.

How many local minima are they? Are they good or bad?

Neural networks always have multiple local minima because of **model identifiability** issues (things that do no change the value of the loss):

- **Reordering** the neurons in each layer ($N!^K$ possible orderings for K layers with N neurons each!)
- **Scaling** the incoming weights and biases of a ReLU neuron by β and its outgoing weights by $1/\beta$.

→ This creates a large or infinite number of local minima, but they are **all equivalent** to each other (not a problem).

Local Minima

For many years, people believed that large neural networks failed because of poor local minima.

Recent theoretical and experimental results suggest that, for **sufficiently large** neural networks:

- Most stationary points are **saddle points** corresponding to a **high value** of the loss function
- SGD manages to avoid them in practice
- Most local minima correspond to a **low value** of the cost function

The PyTorch framework

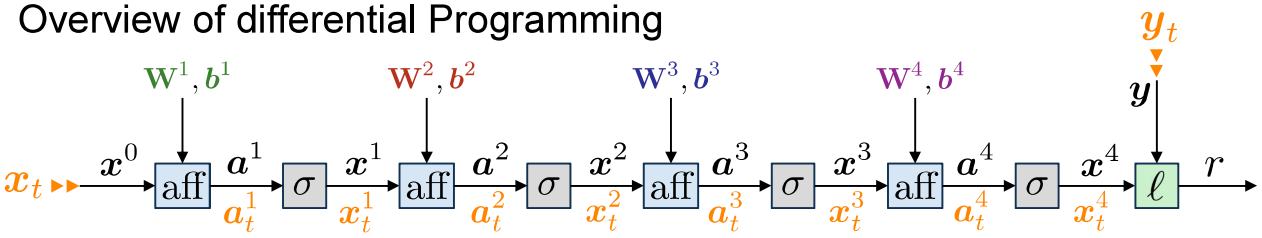
PyTorch

GOOD NEWS: You (probably) **won't ever need to implement backpropagation or SGD yourself!** ☺

- PyTorch is an opensource Python library designed to easily **design**, **train** and **test** neural networks, initially developed by Facebook (Meta), based on Torch. **Constantly evolving** thanks to a broad community
- It uses the **abstractions** allowed by Python, and in particular **object oriented programming**, in order to seamlessly manipulate all the objects we have seen: **data/constants**, **variables/parameters**, **functions**, **optimizers**, **loss…**
- It uses **differential programming**, a concept first introduced in ***Theano***. A module called ***AutoGrad*** automatically records every operations done on variables, so that the gradient of complex functions (such as DNN) can be automatically calculated using **backprop** and **elementary gradients**.
- Includes support for **GPU** and a C++ interface.
- Competing framework: **TensorFlow**, initially developed by Google Brain.
  ≈ *TensorFlow → Production / PyTorch → R&D.*

The PyTorch framework

Overview of differential Programming



- Model the network as an ***acyclic computational flow graph***
- Associate each box with a `forward` method, that computes the value of the box given its children
- Call the `forward` method of each box in **left->right** order

Similarly for backpropagation:

- Associate each box with a backward method, that computes the gradient with respect to each child box
- Call the backward method of each box in reverse**, right->left** order

The PyTorch framework

PyTorch

- Tensors (Data)

```
import torch
a = torch.Tensor([[1,2],[3,4]])
print(a)
```

```
1 2
3 4
[torch.FloatTensor of size 2x2]
```

```
print(a**2)
```

```
1 4
9 16
[torch.FloatTensor of size 2x2]
```

https://cs230.stanford.edu/blog/pytorch/

The PyTorch framework

- Variables, Functions and Autograd

```
from torch.autograd import Variable
a = Variable(torch.Tensor([[1,2],[3,4]]), requires_grad=True)
print(a)
```

```
Variable containing:
1 2
3 4
[torch.FloatTensor of size 2x2]
```

```
y = torch.sum(a**2) # 1 + 4 + 9 + 16
print(y)
```

```
Variable containing:
30
[torch.FloatTensor of size 1]
```

```
y.backward() # compute gradients of y wrt a
print(a.grad) # print dy/da_ij = 2*a_ij for a_11, a_12, a21, a22
```

```
Variable containing:
2 4
6 8
[torch.FloatTensor of size 2x2]
```

https://cs230.stanford.edu/blog/pytorch/

The PyTorch framework

- Loss

```
loss_fn = nn.CrossEntropyLoss()
loss = loss_fn(out, target)


def myCrossEntropyLoss(outputs, labels):
    batch_size = outputs.size()[0] # batch_size
    outputs = F.log_softmax(outputs, dim=1) # compute the log of softmax values
    outputs = outputs[(batch_size), labels] # pick the values corresponding to the labels
    return -torch.sum(outputs)/num_examples
```

https://cs230.stanford.edu/blog/pytorch/

The PyTorch framework

- Models / Neural Network Modules

```
import torch.nn as nn
import torch.nn.functional as F
class TwoLayerNet(nn.Module):
    def __init__(self, D_in, H, D_out):
        """ Constructor. Instantiate two nn.Linear modules and assign them as member variables.
        D_in: input dimension,  H: dimension of hidden layer,   D_out: output dimension
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = nn.Linear(D_in, H)
        self.linear2 = nn.Linear(H, D_out)

 def forward(self, x):
        """ In the forward function we accept a Variable of input data and we must return a
        Variable of output data. We can use Modules defined in the constructor as well as arbitrary
        operators on Variables.
        """
        h_relu = F.relu(self.linear1(x))
        y_pred = self.linear2(h_relu)
        return y_pred
```

https://cs230.stanford.edu/blog/pytorch/

The PyTorch framework

- Using Models / Neural Network Modules

```
#N is batch size; D_in is input dimension;
#H is the dimension of the hidden layer; D_out is output dimension.
N, D_in, H, D_out = 32, 100, 50, 10

#Create random Tensors to hold inputs and outputs, and wrap them in Variables
x = Variable(torch.randn(N, D_in)) # dim: 32 x 100

#Construct our model by instantiating the class defined above
model = TwoLayerNet(D_in, H, D_out)

#Forward pass: Compute predicted y by passing x to the model
y_pred = model(x) # dim: 32 x 10
```

https://cs230.stanford.edu/blog/pytorch/

The PyTorch framework

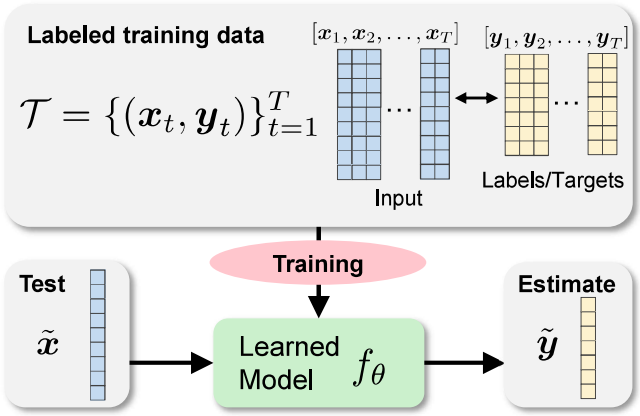
- Core Training Step

```
output_batch = model(train_batch) # compute model output
loss = loss_fn(output_batch, labels_batch) # calculate loss

#pick an SGD optimizer
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)

#or pick ADAM
optimizer = torch.optim.Adam(model.parameters(), lr = 0.0001)

optimizer.zero_grad() # clear previous gradients
loss.backward() # compute gradients of all variables wrt loss
optimizer.step() # perform updates using calculated gradients
```

https://cs230.stanford.edu/blog/pytorch/

OUTLINE

Supervised Learning

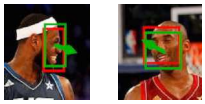


1. Discrete case: («one-hot»)
- ▸ Classification
Ex. application: *dog breed*

$\boldsymbol{y} =$ 

2. Continuous case:
- ▸ Regression
Ex. application: *head pose*

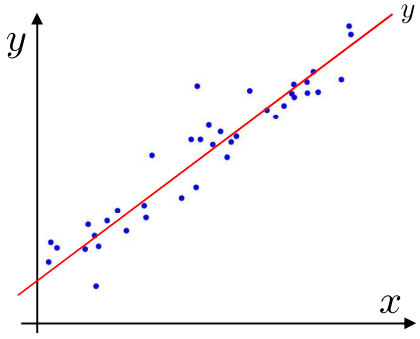
$\boldsymbol{y} =$ 



3. Sparse case:
- ▸ Multi-label classification
Ex. application: *image labelling*

$\boldsymbol{y} =$ 

Generalizing linear regression



- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t) - y_t)^2$

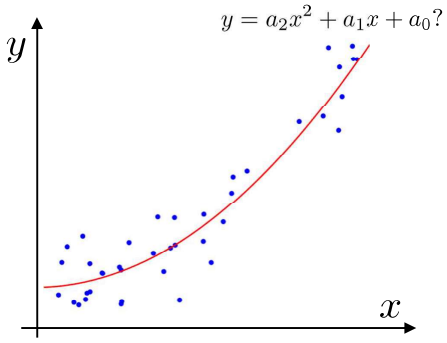
$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\Big(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix}a\\b\end{bmatrix} - y_t\Big)^2 = \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}$$

→ This generalizes to $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbf{A}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{b}) \in \mathbb{R}^{N\times D} \times \mathbb{R}^N$:

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\|\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{b} - \boldsymbol{y}_t\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \begin{bmatrix}\hat{\boldsymbol{a}}_{n,0}\\ b_{n,0}\end{bmatrix} = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}_n, \ \forall n$$

$$\mathbf{W} \in \begin{bmatrix}\boldsymbol{x}_1^\top, & 1\\ \vdots & \vdots\\ \boldsymbol{x}_T^\top, & 1\end{bmatrix} \in \mathbb{R}^{T\times(D+1)}$$

Generalizing linear regression



- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = a_2x^2 + a_1x + a_0$
- Parameters: $\boldsymbol{\theta} = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t) - y_t)^2$

→ What about polynomial regression?

- Convertible to the **same problem** using the ***lifting***: $y = [a_0, a_1, a_2] \times \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$
- More generally, 2nd degree **multivariate** polynomial regression $(\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{x} \in \mathbb{R}^D)$:

$$y_n = \sum_{i=1}^{D}\sum_{j=i}^{D} a_{ij}^{(n)} x_i x_j + \sum_{i=1}^{D} a_i^{(n)} x_i + a_0^{(n)} \;\Rightarrow\; y_n = [a_0^{(n)}, a_1^{(n)}, \ldots, a_{11}^{(n)}, a_{12}^{(n)} \ldots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ x_1x_2 \\ \vdots \\ x_D^2 \end{bmatrix}$$

→*How many parameters?* $\mathcal{O}(ND^2)$

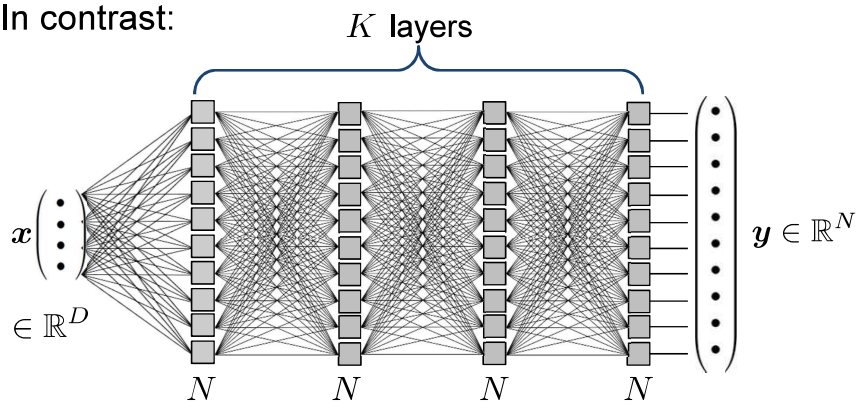
Generalizing linear regression

- K[th] degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D}\sum_{i_2=i_1}^{D}\cdots\sum_{i_K=i_{K-1}}^{D} a_{i_1 i_2 \ldots i_K}^{(n)} x_{i_1}x_{i_2}\ldots x_{i_K} + \cdots + \sum_{i_1=1}^{D}\sum_{i_2=i_1}^{D} a_{i_1 i_2}^{(n)} x_{i_1}x_{i_2} + \sum_{i_1=1}^{D} a_{i_1}^{(n)} x_{i_1} + a_0^{(n)}$$

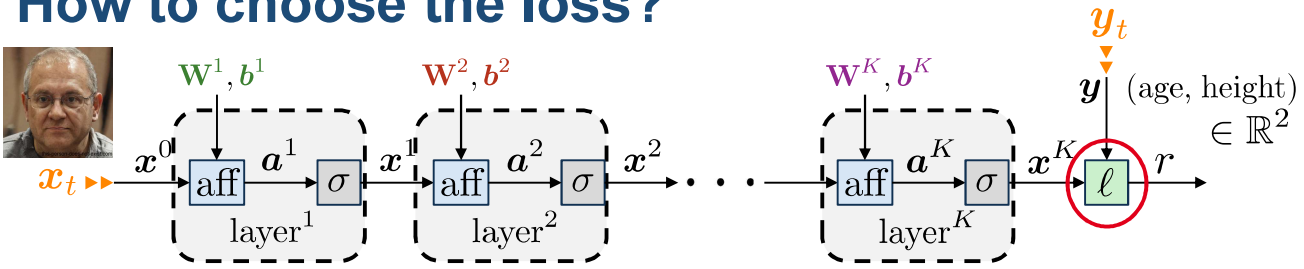
→*How many parameters?* $\mathcal{O}(ND^K)$

- In contrast:



→*How many parameters?*
$\mathcal{O}(DN + (K-1)N^2)$

How to choose the loss?



A general principled approach is to use the network to model $p(\boldsymbol{y}|\boldsymbol{x})$.

1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of $\boldsymbol{y}$ and $\boldsymbol{x}^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(\boldsymbol{y})\}_{\lambda \in \Lambda}$.

Ex: $\tilde{p}_{\boldsymbol{\mu}}(\boldsymbol{y}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|\boldsymbol{y}-\boldsymbol{\mu}\|_2^2}{2}\right)$, (Gaussian)

$\tilde{p}_b(y) = \begin{cases} b \in [0,1] \text{ for } y = 1 \\ 1 - b \text{ for } y = 0 \end{cases}$ (Bernouilli)

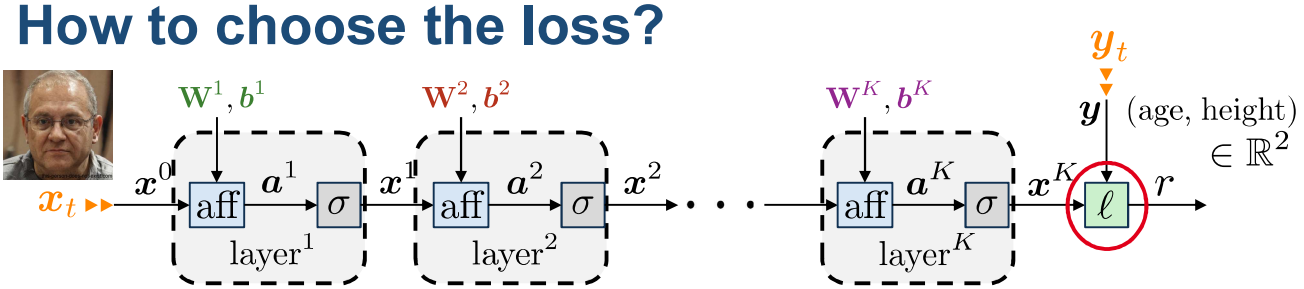
2) We then model the conditional probability as: $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = \tilde{p}_{\lambda = \text{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x})}(\boldsymbol{y})$

3) We want to find $\boldsymbol{\theta}$ that maximizes the **likelihood** over the training set:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

Note: we assume the training set examples are **independent**.

How to choose the loss?



4) Usually, we define the **total loss** as the ***negative log-likelihood***:

$$L(\text{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} -\log \tilde{p}_{\text{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t) = \sum_{t=1}^{T} -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t)$$

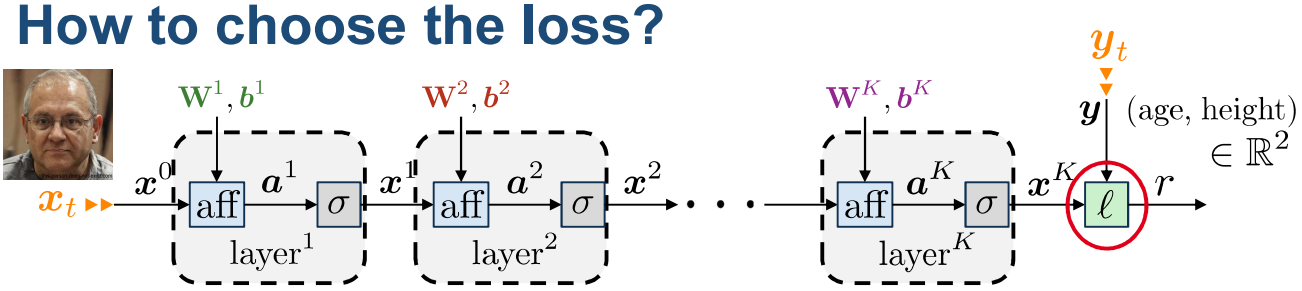
Example with the Gaussian distribution:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = -\log \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2}{2}\right) \overset{c}{=} \frac{1}{2}\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2$$

→ We recover the **L2 loss**!

Using the L2 loss is equivalent to assuming the network will make i.i.d Gaussian errors.

How to choose the loss?



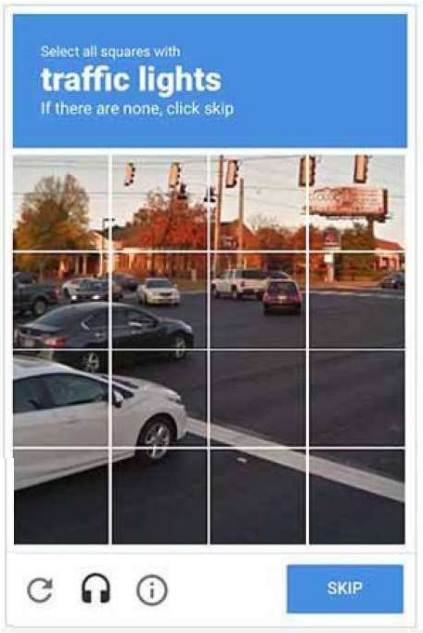
The approach is very general and can be used with a variety of parameterized probability distributions.

- For $y \geq 0$ we can use an exponential distribution $\tilde{p}_\lambda(y) = \lambda \exp(-\lambda y)$
- We can use this approach to not only estimate **the mean** but also ***the variance*** (≈uncertainty) of the network output:

$$\tilde{p}_{\boldsymbol{\mu},\sigma^2}(\boldsymbol{y}) = \frac{1}{\sqrt{2\pi\sigma^{2N}}} \exp\left(-\frac{\|\boldsymbol{y}-\boldsymbol{\mu}\|_2^2}{2\sigma^2}\right), \quad \boldsymbol{x}^K \equiv [\boldsymbol{\mu}, \sigma^2]$$

Detection

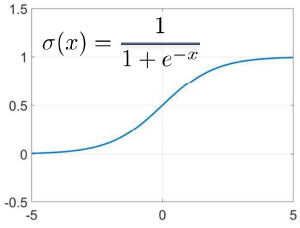
Example: Captcha $\quad \boldsymbol{x}_t \in \mathbb{R}^D, \; y_t \in \{0, 1\}$



- Let's use the same principle to design our loss.
- We can use a Bernouilli distribution:

$$\tilde{p}_b(y) = \begin{cases} b \in [0,1] \text{ for } y = 1 \\ 1 - b \text{ for } y = 0 \end{cases}$$

- Note that $b \in [0, 1]$, hence we need to constrain the output of the network in this interval => we use a **sigmoid function** at the output:



- Using the ***maximum likelihood*** approach with this distribution, we obtain the following loss:

$$\ell(x_t^K, y_t) = -\log \tilde{p}_{x_t^K}(y_t) = -y_t \log x_t^K - (1 - y_t) \log(1 - x_t^K),$$

= the ***Binary Cross-Entropy***.

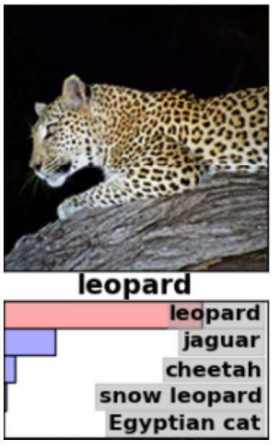
Classification

- This generalizes to **multi-class classification**

$$\boldsymbol{x}_t \in \mathbb{R}^D,\ y_t \in \{1, 2, \ldots, N\}$$

- It is convenient to represent the output as a ***"one-hot"*** vector:

$$\boldsymbol{y}_t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{index of the class}$$



Ex: ImageNet
(1000 classes)

- We use a ***categorical distribution***:

How to enforce this constraints at the network output?

$$\tilde{p}_{\boldsymbol{b}}(\boldsymbol{y}) = \begin{cases} b_1 \in [0,1] \text{ for } y_1 = 1 \\ b_2 \in [0,1] \text{ for } y_2 = 1 \\ \vdots \\ b_N \in [0,1] \text{ for } y_N = 1 \end{cases}, \text{ with } \sum_n b_n = 1$$

Classification

- The ***Soft-Max*** activation function:

$$\boldsymbol{x}^K = \sigma(-\boldsymbol{a}^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a_{n'}^K)} \begin{bmatrix} \exp(a_1^K) \\ \exp(a_2^K) \\ \vdots \\ \exp(a_N^K) \end{bmatrix}$$

- Can be viewed as a generalization of the sigmoid
- Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value
- Using the maximum likelihood approach with a categorical distribution yields the (generalized) **cross entropy loss**:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = \sum_{n=1}^{N} -y_{t,n} \log x_{t,n}^K$$

Multi-Label Classification

- Can be done by statistically aggregating **multiple binary detectors**
- Falls in the category of **ensemble methods**

Scenario:

- Imagine we have a training dataset $\mathcal{T}$ containing 10,000 images with labels (supervised learning)
- Using the backpropagation algorithm, we train a DNN to perform the multi-class classification task
- We get nearly **perfect results** on these 10,000 images, e.g., 99.9% of correct classification
- However, when we run the DNN on new images, the results are **awful,** i.e., close to random.

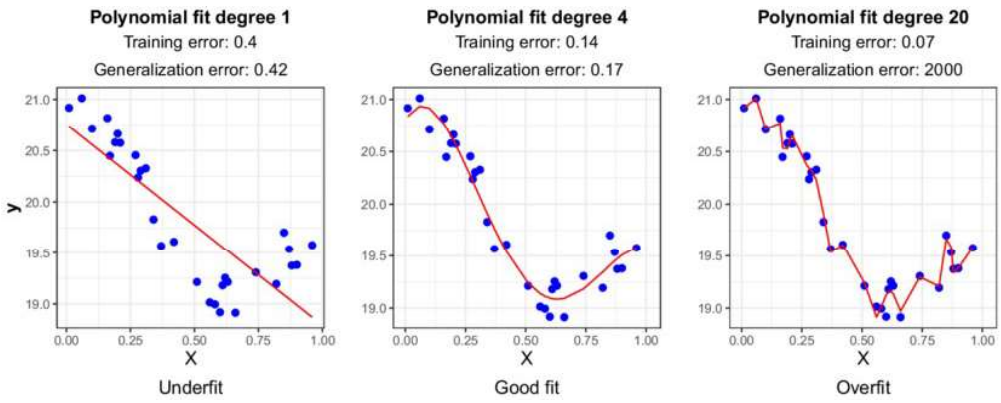
What's going on?

Overfitting

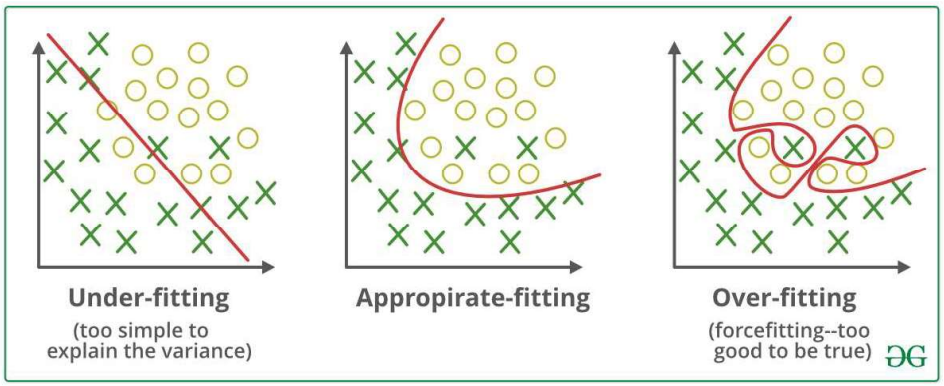
- Our algorithm is guilty of **overfitting** (*sur-apprentissage*)
- Instead of learning general features to classify the images, it **learned by heart** all the images in our training dataset!
- Remember that we often have **millions** or **billions** or parameters in a deep model. Hence, it has the **capacity** to **store/encode** large amount of data
- This may even happen for models of relatively small capacity, if the **amount of training data** is insufficient.

Overfitting

- Ex: polynomial regression

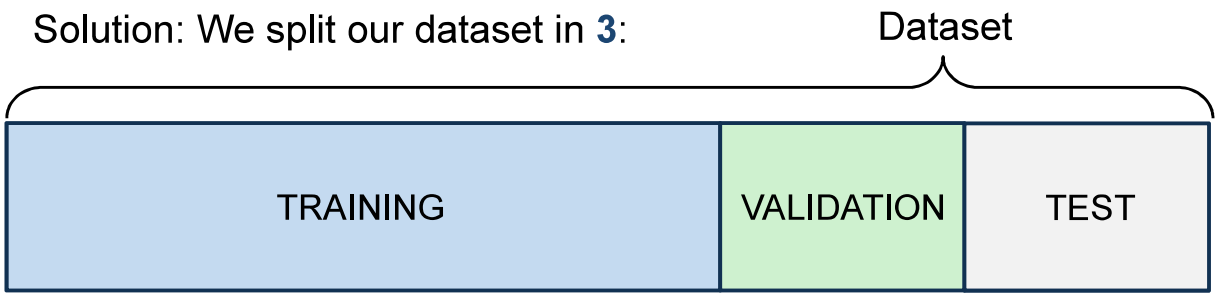


- Ex: binary classification



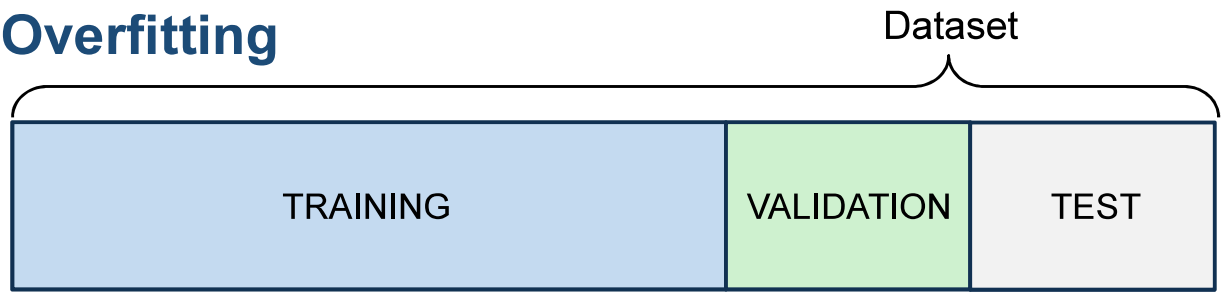
Overfitting

- In supervised learning, we are **not only interested** in a model that works perfectly on our training set. **We already have the answers anyway**, by definition of a (supervised) training set !
- We want a model that **generalizes** to **unseen** data
- Solution: We split our dataset in **3**:



- These 3 subsets must be **perfectly disjoint** and all **representative** of the data.
- To achieve this, the split is done **at random**.

Overfitting

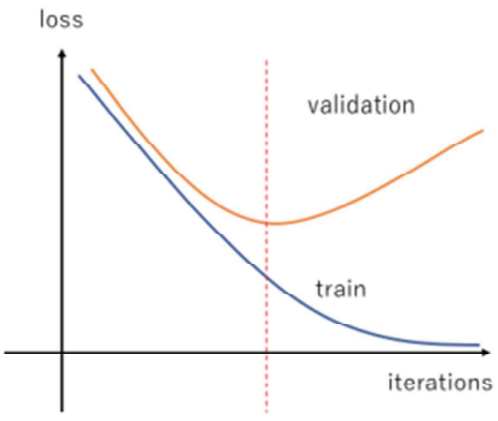


- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work
- The model parameters are only optimized over the **training set**
- We only use the validation set to:
  - At each training step, verify that the model is making progress on that set (possibly using another performance measure than the loss) => If not: we **stop**.
  - Tune hyperparameters (e.g. gradient steps), compare different families of models
- Looking at the test set if **forbidden** in any of those steps *("inverse crime")*

Overfitting

- We can detect overfitting by tracking the **total loss** over the training iterations / epochs:

GOOD. The validation loss is only slightly less good than the training one and does not increase



NOT GOOD. The validation loss increases: we are starting to learn by heart the training set !

Some vocabulary

- **Capacity:** flexibility of a model. It often (but not necessarily!) correlates with the number of parameters of the model
- **Hyper-parameter:** a parameter of a model that is not trained (specified before training)
- **Model selection:** process of choosing the best hyperparameters on the validation set
- **Underfitting:** state of model which could improve generalization with **more** training or **more** capacity
- **Overfitting:** state of model which could improve generalization with **less** training or **less** capacity

Overfitting vs. Underfitting

Quizz

- If capacity increases:
  - training error will ?
  - validation error will ?
- If training time increases:
  - training error will ?
  - validation error will ?
- If training set size increases:
  - generalization error will ?
  - difference between the training and generalization error will ?

IV. Supervised Learning ▸ Over and Underfitting

Quizz

- If capacity increases:
  - training error will ? decrease
  - validation error will ? decrease or increase
- If training time increases:
  - training error will ? decrease
  - validation error will ? decrease or increase
- If training set size increases:
  - generalization error will ? decrease (or stay the same)
  - difference between the training and generalization error will ? decrease

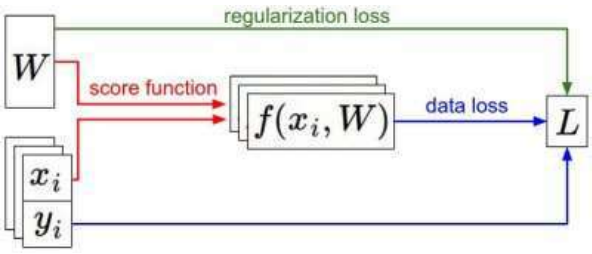
École d'ingénieurs Télécom Physique Université de Strasbourg Inria Antoine.Deleforge@inria.fr Artificial Intelligence & Deep Learning 153/200

Techniques to reduce overfitting

1) Regularization

- We add to the total loss a term that depends directly on the parmeters of the neural network:

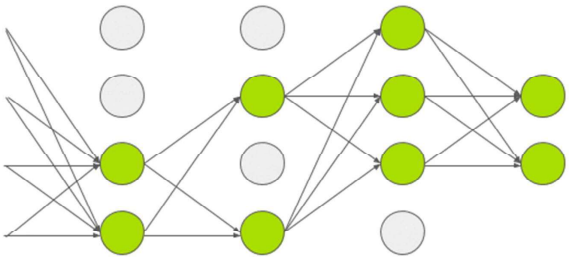
$$L(\text{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\text{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right) + \lambda \mathcal{R}(\boldsymbol{\theta})$$



- For example, we could add the **L2 norm** of the coefficients in the **weight matrices**, to avoid that they become very large (a common clue of overfitting)

Techniques to reduce overfitting

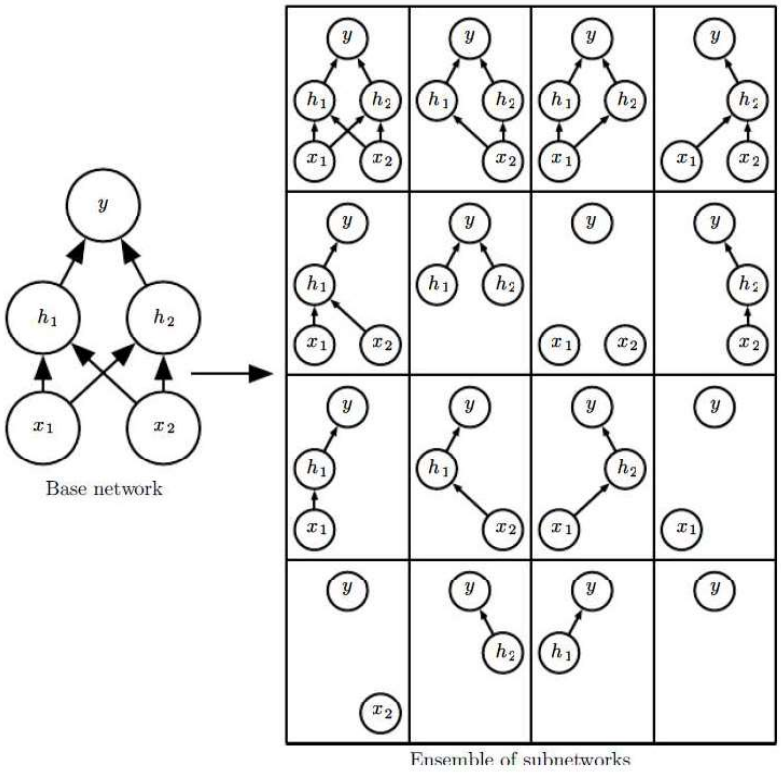
2) Dropout



- Idea: “cripple” the neural network by removing hidden units stochastically
- Each hidden unit is set to 0 with a certain probability at each gradient step
- Hidden units cannot co-adapt to other units
- Hidden units must be more generally useful
- Dropout probability typically between 0.2 and 0.5.
- At test time, replace the masks by their expectation (e.g., constant vector 0.5 if dropout probability is 0.5).
- Can be viewed as averaging an exponential number of networks.

Techniques to reduce overfitting

2) Dropout



Techniques to reduce overfitting

3) Data Augmentation

- Increase the dataset size by applying **transformations** to the input examples that does **not** affect the output (or affect it in a predictable way)
- Examples:
  - Crop an image, flip it, modify its brightness. Replace words by synonyms in sentences
  - Add noise to input signals, degrade their quality, remove imperceptible parts
- One may as well augment the data by using **simulators**: e.g. photorealistic 3D scenes, simulated acoustic scenes….or other generative machine learning models.
- Data augmentation is often key for a ML method to work

Techniques to reduce overfitting

3) Data Augmentation

OUTLINE

Supervised Learning

Labeled training data

$\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$ $[x_1, x_2, \dots, x_N]$ Input $[y_1, y_2, \dots, y_N]$ Labels/Targets

Training

Test $\tilde{\boldsymbol{x}}$ → Learned Model f_θ → Estimate $\tilde{\boldsymbol{y}}$

1. Discrete case: («one-hot»)
- ▸ Classification

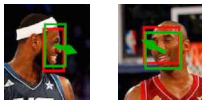
Ex. application: *dog breed*

$\boldsymbol{y} =$ 1

2. Continuous case:
- ▸ Regression

Ex. application: *head pose*

$\boldsymbol{y} =$



3. Sparse case:
- ▸ Multi-classification

Ex. application: *image labelling*

$\boldsymbol{y} =$ 0.1, 0.6, 0.3

Unsupervised Learning

Unlabeled training data

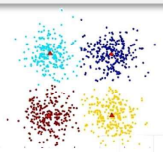
$\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$ $[x_1, x_2, \dots, x_N]$ Input

Training

Learned Model f_θ → Representation $[z_1, z_2, \dots, z_N]$

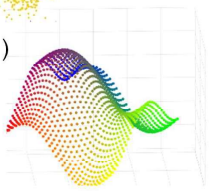
1. Discrete case:
- ▸ Clustering

$\boldsymbol{z} =$



2. Continuous case: ($\dim(\boldsymbol{z}) \ll \dim(\boldsymbol{x})$)
- ▸ Dimensionality Reduction

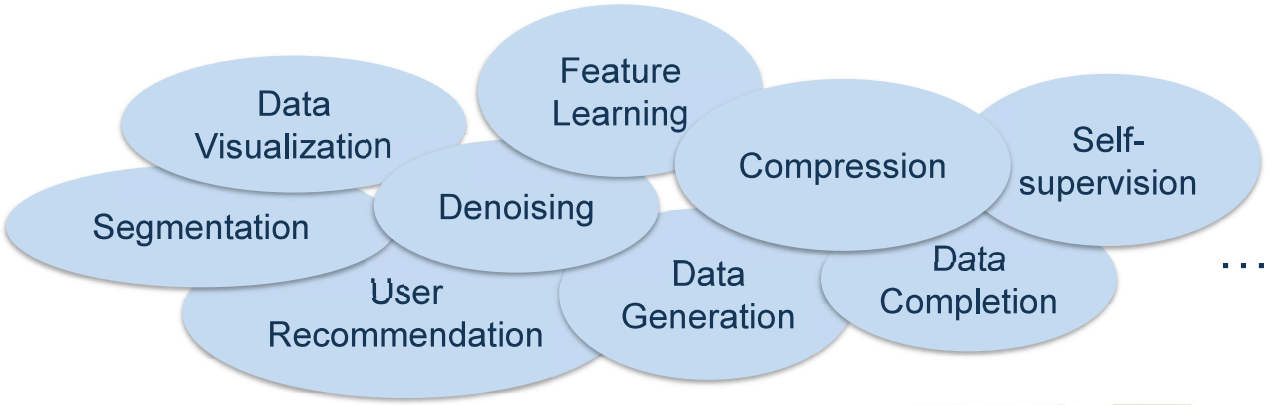
$\boldsymbol{z} =$



3. Sparse case:
- ▸ Dictionary Learning

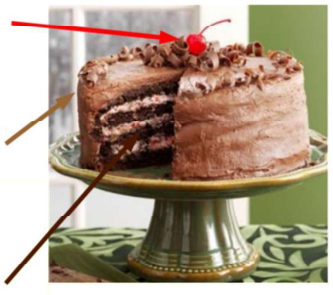
$\boldsymbol{z} =$

Applications of Unsupervised Learning



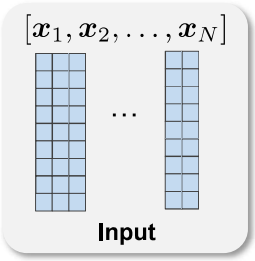
« If machine learning is a cake, then unsupervised learning is the actual cake, supervised learning is the icing, and reinforcement learning is the cherry on the top. »

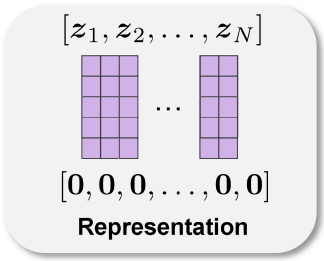
-Yann Lecun (Facebook AI) at NIPS 2016



Potential to learn from massive amount of unlabeled data to generate even more

How does it work?





- **Desirable properties**
  - Preserve information from original data
  - Can be used to reconstruct it / generalize it / efficiently learn from it
- **A unifying view :** Generative models

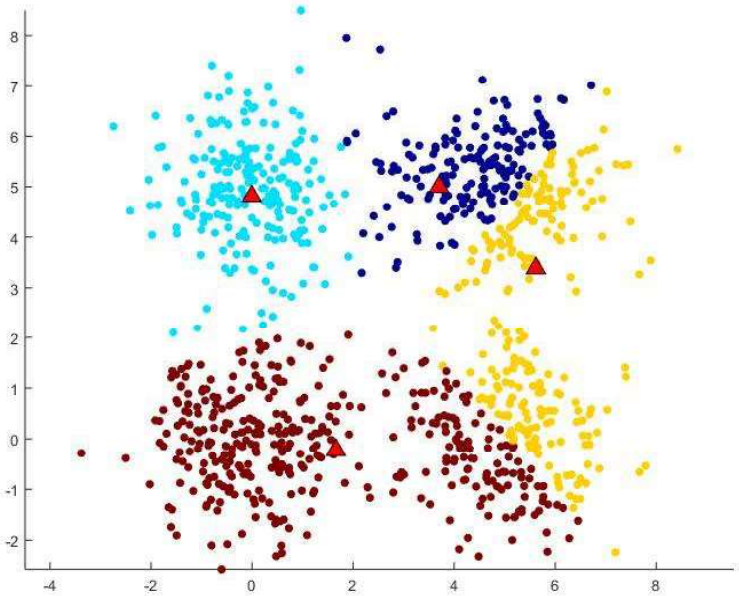
Find $\begin{cases} p_\theta(\boldsymbol{z}) & \text{(as simple as possible)} \\ p_\theta(\boldsymbol{x}|\boldsymbol{z}) & \text{(typically, transformation of } \boldsymbol{z}\text{)} \end{cases}$ so that $p_\theta(\boldsymbol{x}) = \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z}$

fits the data (e.g., maximum likelihood).

Generation	$p_\theta(\boldsymbol{z}) \rightarrow \boldsymbol{z} \rightarrow \boldsymbol{x}$
Compression	$\boldsymbol{x} \rightarrow p_\theta(\boldsymbol{z}\|\boldsymbol{x}) \rightarrow \boldsymbol{z}$
Reconstruction / Completion / Denoising	$\tilde{\boldsymbol{x}} \rightarrow p_\theta(\boldsymbol{z}\|\tilde{\boldsymbol{x}}) \rightarrow \boldsymbol{z}$ $\rightarrow p_\theta(\boldsymbol{x}\|\boldsymbol{z}) \rightarrow \hat{\boldsymbol{x}}$

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random
- The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

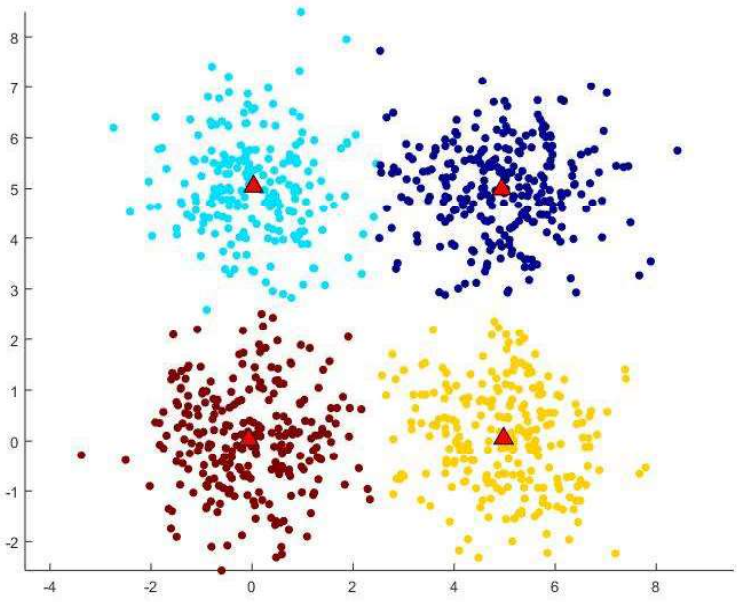
2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:
- $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

How would you cluster/group/separate these 2D points?

$\{\boldsymbol{x}_n\}_{n=1}^N \subset \mathbb{R}^2$



1. Pick $K = 4$ points at random
▸ The *centroids* $\{\boldsymbol{c}_k\}_{k=1}^K \subset \mathbb{R}^2$

2. For each point $\boldsymbol{x}_n$, find its nearest centroid $\boldsymbol{c}_k$. Place n in cluster $\mathcal{G}_k$.

3. Update each $\boldsymbol{c}_k$ as the mean of all points in $\mathcal{G}_k$:
▸ $\boldsymbol{c}_k = \frac{1}{|\mathcal{G}_k|} \sum_{n \in \mathcal{G}_k} \boldsymbol{x}_n$

4. Repeat **2.** and **3.** until convergence

The K-means Algorithm

K-means: What does it do?

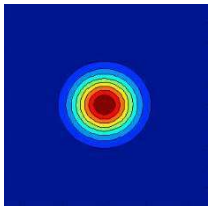
- It decreases this **total loss** at each iteration:

$$\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) = \sum_{k=1}^{K} \sum_{n \in \mathcal{G}_k} \|\boldsymbol{x}_n - \boldsymbol{c}_k\|^2, \quad \theta = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_K\}$$

▸ Non-Convex (Depends on init.)

- « Compression » interpretation: $\mathbf{X}$ is « summarized » by θ
- Probabilistic / Generative interpretation *(where is $\boldsymbol{z}$) ?*

$$\begin{cases} p(z_{n,k} = 1) = \frac{1}{K} \quad \text{where } \boldsymbol{z}_n = \begin{matrix} \square \\ \square \\ \square \\ \square \\ \square \\ k \rightarrow \boxed{1} \\ \square \end{matrix} \in \{0,1\}^K \text{ !} \\ p_\theta(\boldsymbol{x}_n | \boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{c}_k, \mathbf{I}) \text{ (Gaussian centered on } \boldsymbol{c}_k \text{)} \end{cases}$$



$$\Rightarrow p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{n=1}^{N} \sum_{k=1}^{K} z_{n,k} \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{c}_k, \mathbf{I})$$

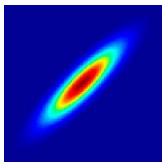
$$\boxed{\mathcal{C}(\mathbf{X}, \mathcal{G}, \theta) \overset{c}{=} -\log p_\theta(\mathbf{Z}|\mathbf{X})}$$

K-means can be seen as a *maximum a posteriori* (MAP) approach!

Gaussian mixture models: a generalization

$$\begin{cases} p(z_{n,k}=1)=\pi_k, \quad \sum_{k=1}^K \pi_k = 1 \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n \equiv k) = \mathcal{N}(\boldsymbol{x};\boldsymbol{c}_k,\Sigma_k) \end{cases}$$

$$\Rightarrow p_\theta(\boldsymbol{x}_n) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_n;\boldsymbol{c}_k,\Sigma_k)$$



$$\theta = \{\pi_1,\dots,\pi_K, \boldsymbol{c}_1,\dots,\boldsymbol{c}_K, \Sigma_1,\dots,\Sigma_K\}$$

- Find θ that maximizes the *observed data log-likelihood*:

$$\mathcal{L}_\theta(\mathbf{X}) = \log p_\theta(\mathbf{X}) = \sum_n \log p_\theta(\boldsymbol{x}_n)$$

…very hard to solve directly.

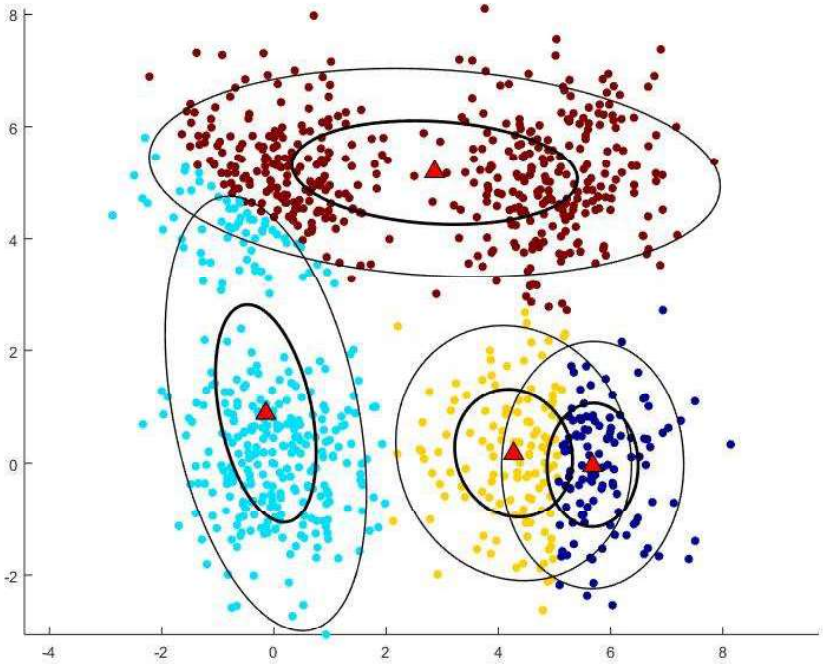
- The **Expectation-Maximization** (EM) algorithm iteratively maximizes the *expected complete-data log-likelihood* instead:

$$\theta^{(i+1)} = \underset{\theta}{\operatorname{argmax}}\ \mathbb{E}_{p_{\theta^{(i)}}(Z|X)}\{\log p_\theta(\mathbf{X},\mathbf{Z})\}$$

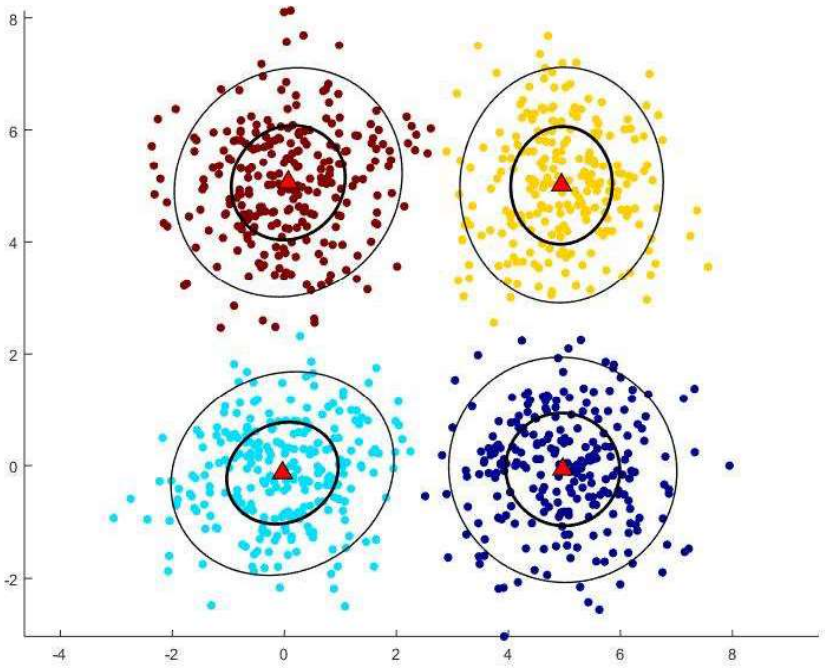
Repeat until convergence:

- $r_{n,k}^{(i)} = p_{\theta^{(i-1)}}(z_{n,k}=1|\boldsymbol{x}_n)$
- $\pi_k^{(i)} = \frac{1}{N}\sum_n r_{n,k}^{(i)} = \frac{\overline{r}_k}{N}$
- $\boldsymbol{c}_k^{(i)} = \frac{1}{\overline{r}_k}\sum_n r_{n,k}^{(i)} \boldsymbol{x}_n$
- $\Sigma_k^{(i)} = \frac{1}{\overline{r}_k}\sum_n r_{n,k}^{(i)} (\boldsymbol{x}_n - \boldsymbol{c}_k^{(i)}) \cdot (\boldsymbol{x}_n - \boldsymbol{c}_k^{(i)})^\top$

Gaussian Mixture Model Expectation-Maximization (GMM EM)



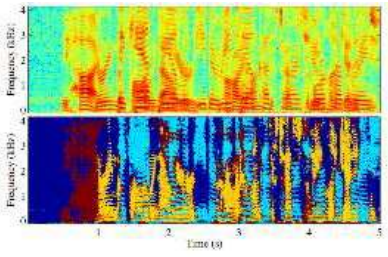
Gaussian Mixture Model Expectation-Maximization (GMM EM)



And in the *Deep Learning Era* …

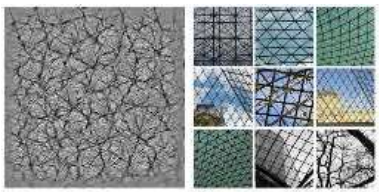
- ***Deep clustering:*** trains a **DNN** to project data to a feature space where K-means can be optimally used

- Application to blind speech source separation:



John R. Hershey, Zhuo Chen, Jonathan Le Roux, and Shinji Watanabe**. "Deep clustering: Discriminative embeddings for segmentation and separation."** In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 31-35. IEEE, 2016.

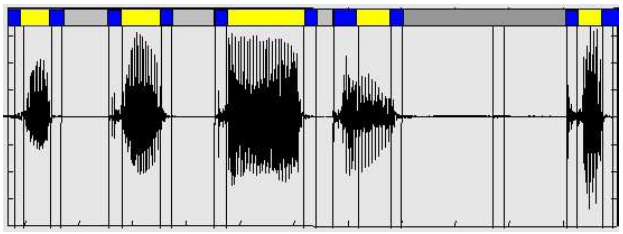
- Application to feature learning from images:



Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. **"Deep clustering for unsupervised learning of visual features."** In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132-149. 2018.

Typical applications of clustering

Image segmentation ($\{x_n\}_{n=1}^N$ are local descriptors)



Audio segmentation ($\{x_n\}_{n=1}^N$ are sound segment)

Anti-spam filters

Data Quantification

DNA sequence analysis

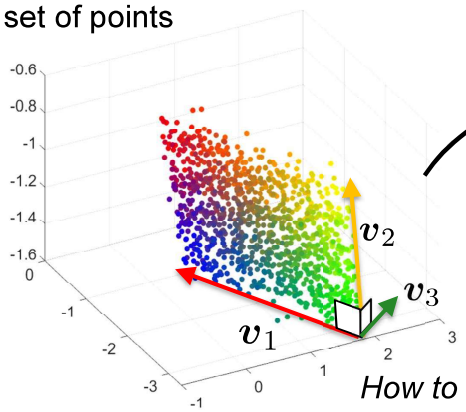
Medical imaging

Speech diarization

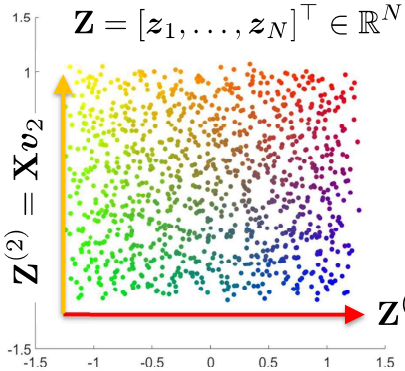
Species classificaton *(biology)*

Social network analysis

Let $\mathbf{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times 3}$ be a 3D set of points



$\mathbf{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_N]^\top \in \mathbb{R}^{N \times 2}$



How to reduce its dimensionality while preserving most of its information? ➡ Project it along axes of **maximal variance**

1. Find $\boldsymbol{v}_1 \in \mathbb{R}^3$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_1) = \mathrm{Var}([\boldsymbol{v}_1^\top \boldsymbol{x}_1, \dots, \boldsymbol{v}_1^\top \boldsymbol{x}_n]^\top) = \frac{1}{N}\sum_{n=1}^N |\boldsymbol{v}_1^\top \boldsymbol{x}_n|^2$ is largest

▸ The solution is given by the eigenvector associated to the largest eigenvalue λ_1 of the sample covariance matrix $\mathbf{C} = \frac{1}{N}\sum_{n=1}^N (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^\top \in \mathbb{R}^{3\times 3}$.

2. Find $\boldsymbol{v}_2 \perp \boldsymbol{v}_1$ such that $\mathrm{Var}(\mathbf{X}\boldsymbol{v}_2)$ is largest. ▸ Second dominant eigenvector of $\mathbf{C}$.

3. Find $\boldsymbol{v}_3 \perp [\boldsymbol{v}_1, \boldsymbol{v}_2]$ … *etc.*

▸ The **Principal Axes** of $\mathbf{X}$ are the P dominant eigenvectors of $\mathbf{C}$.

Principal Component Anaysis

Principal Component Analysis

- Probabilistic / Generative interpretation:

$$\begin{cases} p_\theta(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n; \mathbf{0}_P, \boldsymbol{\Lambda} - \sigma^2 \mathbf{I}_P), \quad \boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_P), \quad \sigma^2 \leq \lambda_P \\ p_\theta(\boldsymbol{x}_n|\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n; \mathbf{V}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_D), \quad \mathbf{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_P] \in \mathbb{R}^{D \times P}, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \theta = \{\boldsymbol{\Lambda}, \sigma^2, \mathbf{V}\} \end{cases}$$

PCA is **equivalent** to:

- $\hat{\theta} = \underset{\theta}{\text{argmax}} \log p_\theta(\mathbf{X})$ (Maximum Likelihood)
- $\hat{\boldsymbol{z}}_n = \underset{\boldsymbol{z}_n}{\text{argmax}}\, p_{\hat{\theta}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$ (Maximum a posteriori)

M.E. Tipping and C.M. Bishop. **"Probabilistic PCA."** *Journal of the Royal Statistical Society: Series B*, 61, no. 3 (1999): 611-622.

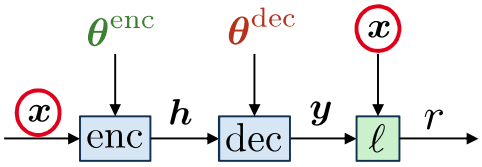
Variational Autoencoders

- Generalize PCA by replacing this by a **DNN** (the *decoder*)
- Optimized using a variational approximation of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ by another neural network (the *encoder*)

Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *ICLR* 2014.

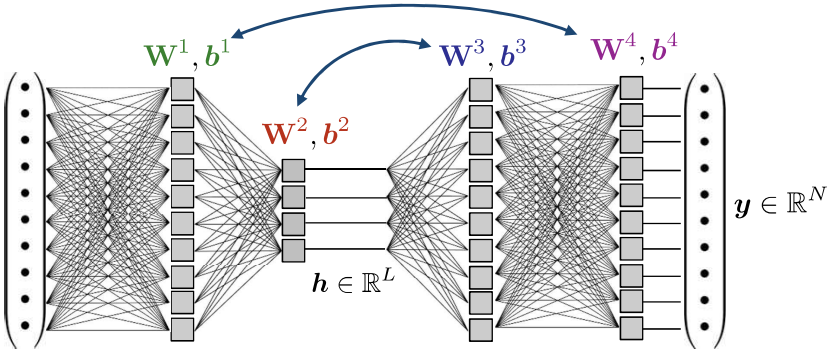
Autoencoder

- A neural network trained to predict its input: a **pretext task**
- Consists of two parts:
  - An **encoder** function $\boldsymbol{h} = \text{enc}(\boldsymbol{x})$
  - A **decoder** function $\boldsymbol{y} = \text{dec}(\boldsymbol{h})$



- The task is non-trivial if the encoder is **dimensionality-reducing**

Ex:

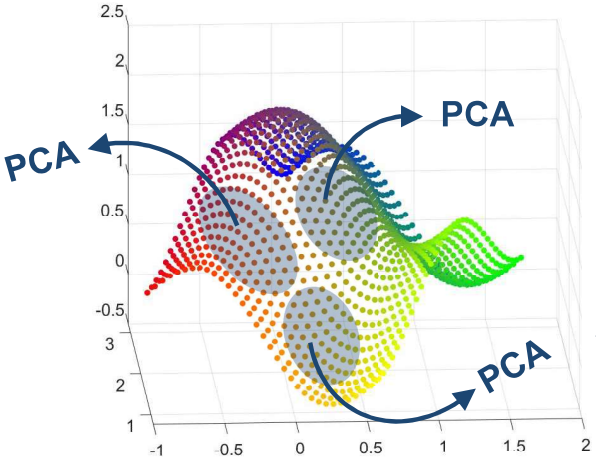


The encoder and decoder parameters can be ***tied*** together

- $\boldsymbol{h}$ is called an ***embedding*** of $\boldsymbol{x}$, i.e., a nonlinear representation of the input.

Manifold Learning

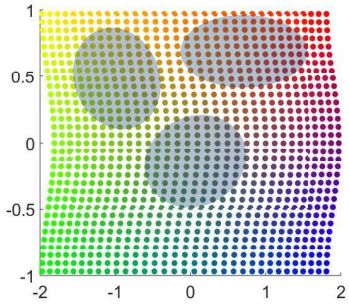
- Local Tangent Space Alignment (LTSA)



1. Builds local k-nearest neighborhoods on the data

2. Applies PCA to each neighborhood

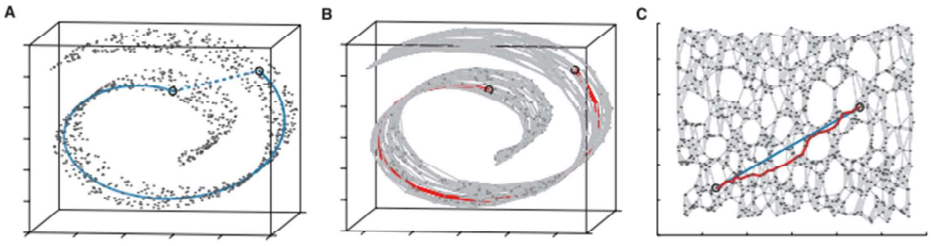
3. Patch the local PCAs together



Zhang, Zhenyue, and Hongyuan Zha. **"Principal manifolds and nonlinear dimensionality reduction via tangent space alignment."** *SIAM journal on scientific computing* 26, no. 1 (2004): 313-338.

Manifold Learning

- Graph-Based Methods: *Isomap, LLE, Laplacian Eigenmap,…*
  - Build a **neighborhood graph** from the data
  - *« Unroll »* the graph to a lower dimensional space
  - Ex: **Isomap**. Compute all ***geodesic distances*** (shortest paths) on the graph



- ***Stochastic neighbor embedding*** (**SNE**): match *neighborhood probabilities* in the high- and low-dim. spaces

Hinton, Geoffrey E., and Sam Roweis. "Stochastic neighbor embedding." *Advances in neural information processing systems* 15 (2002).

Ghodsi, Ali. **"Dimensionality reduction a short tutorial."** *Department of Statistics and Actuarial Science, Univ. of Waterloo, Ontario, Canada* 37, no. 38 (2006): 2006.

Manifold Learning

- Ex: **t-SNE**: uses a Gaussian model for similarity between data points and a *Student's t* (Cauchy) model for similarity in the latent space (2D or 3D)
- The variances of the Gaussians are fixed to attain a given ***entropy*** value.
- The latent representation is found by minimizing the ***Kullback-Leibler divergence*** wrt. these distributions using **gradient descent**.
- Easy to use thanks to the Scikit Learn implementation

sklearn.manifold.TSNE

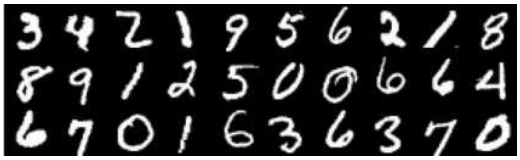
class `sklearn.manifold.TSNE`(*n_components=2, *, perplexity=30.0, early_exaggeration=12.0, learning_rate='auto', n_iter=1000, n_iter_without_progress=300, min_grad_norm=1e-07, metric='euclidean', metric_params=None, init='pca', verbose=0, random_state=None, method='barnes_hut', angle=0.5, n_jobs=None, square_distances='deprecated'*) [source]

Documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

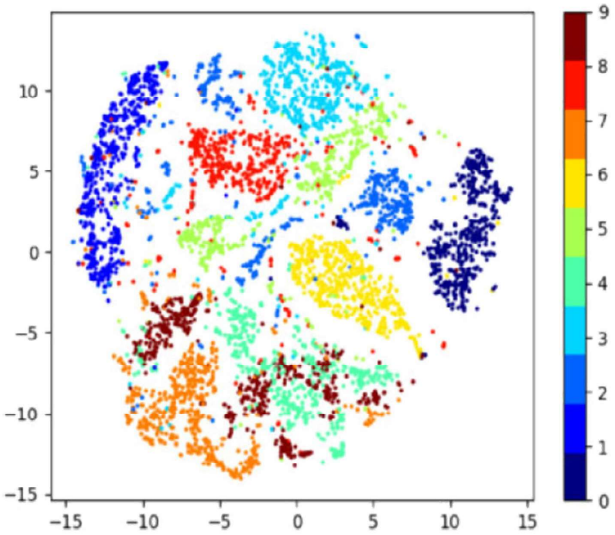
Manifold Learning

Examples of t-SNE visualizations

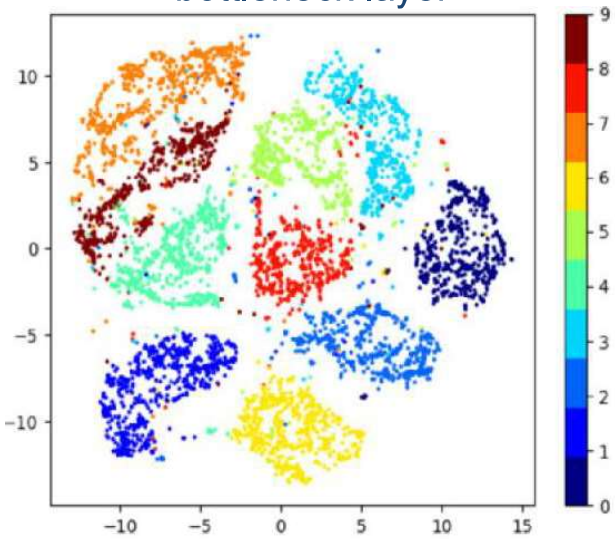
- MNIST dataset



T-SNE on image pixels



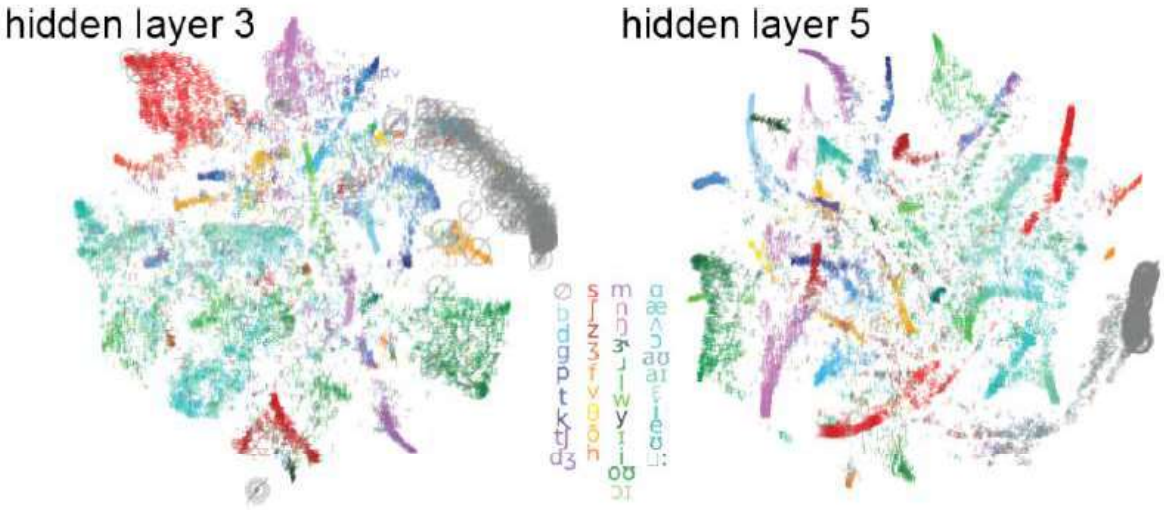
T-SNE on auto-encoder bottleneck layer



Manifold Learning

Examples of t-SNE visualizations

- Speech recognition (Wall Street Journal Dataset)



And in the Deep Learning era?

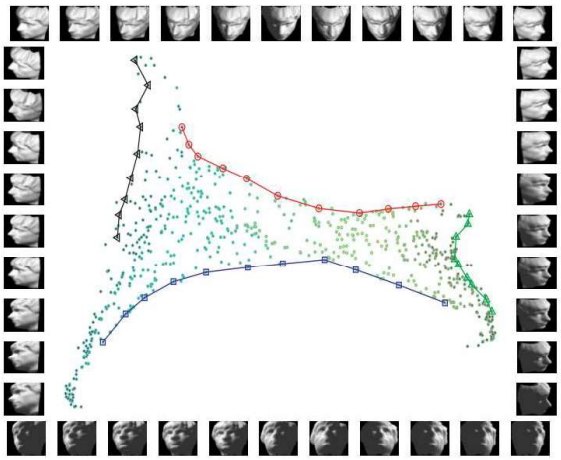
Diederik P. Kingma and Max Welling. **"Auto-encoding variational bayes."**, *arXiv:1312.6114* (2013).

Dinh, Laurent, David Krueger, and Yoshua Bengio. **"Nice: Non-linear independent components estimation."** *arXiv preprint arXiv:1410.8516* (2014). **=> *Invertible Neural Networks***

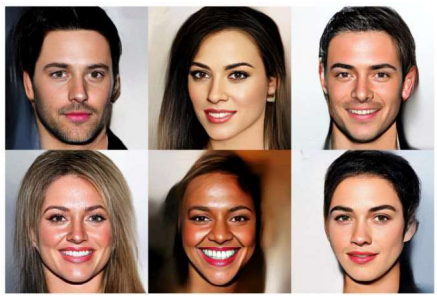
+ extensions (Normalizing Flows, Glow…)

Typical applications:

Dataset Visualization



Data generation (Glow)

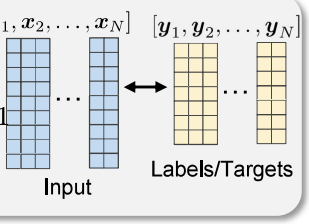


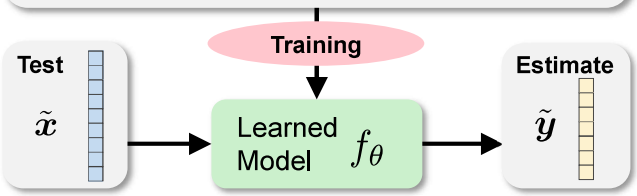
Pre-processing to speed up learning

Compression

Supervised Learning

Labeled training data $\mathcal{T} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N}$





1. Discrete case: («one-hot»)
- ▸ Classification

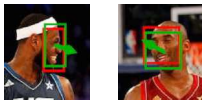
Ex. application: *dog breed*

$y =$ 

2. Continuous case:
- ▸ Regression

Ex. application: *head pose*

$y =$ 



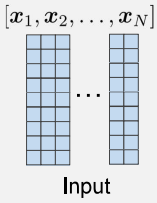
3. Sparse case:
- ▸ Multi-classification

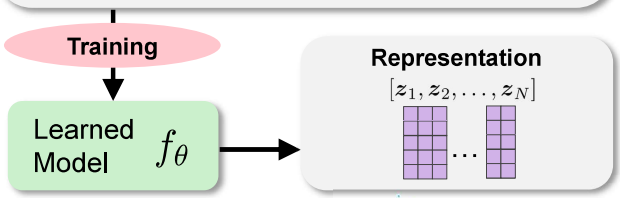
Ex. application: *image labelling*

$y =$ 

Unsupervised Learning

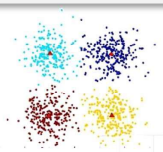
Unlabeled training data $\mathcal{T} = \{\boldsymbol{x}_n\}_{n=1}^{N}$





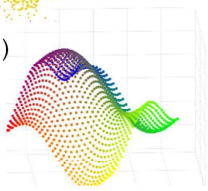
1. Discrete case:
- ▸ Clustering

$z =$ 



2. Continuous case: ($\dim(z) \ll \dim(x)$)
- ▸ Dimensionality Reduction

$z =$ 



3. Sparse case:
- ▸ Dictionary Learning

$z =$ 

OUTLINE

~~I. Introduction~~

~~II. Background~~

~~III. Fitting a Model~~

~~IV. Supervised Learning~~

~~V. Unsupervised Learning~~

VI. Convolutional Neural Networks

- Definition
- Why ConvNets ?
- CNN layers
- Famous examples

Convolutional Neural Networks

- The idea of using such networks to mimic the **human visual system** dates back to K. Fukushima (1980). Y. Lecun was the first to train a CNN using backpropagation (1989)
- **Def:** a neural network that uses a linear operation called ***convolution*** in **at least one layer**, instead of a generic linear layer
- This amounts to **constraining the weight matrix** $\mathbf{W}$ to have a special structure called ***Toeplitz***

$$\mathbf{W} = \begin{bmatrix} v_0{=}u_0 & u_1 & u_2 & \cdots & u_{N-2} & u_{N-1} \\ v_1 & u_0 & u_1 & \ddots & u_{N-3} & u_{N-2} \\ v_2 & v_1 & u_0 & \ddots & \ddots & u_{N-3} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} & \vdots \\ v_{D-1} & v_{D-2} & v_{D-3} & \ddots & \ddots & u_{N-D} \end{bmatrix} = \text{Toeplitz}(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^{D \times N}$$

- Typically, $\boldsymbol{u}$ and $\boldsymbol{v}$ only have only **a few nonzero values**, determined by the ***kernel size***
- In terms of **model fitting**, everything we saw so far remains valid!

But what ***is*** (discrete) convolution?

Discrete Convolution (1D Case)

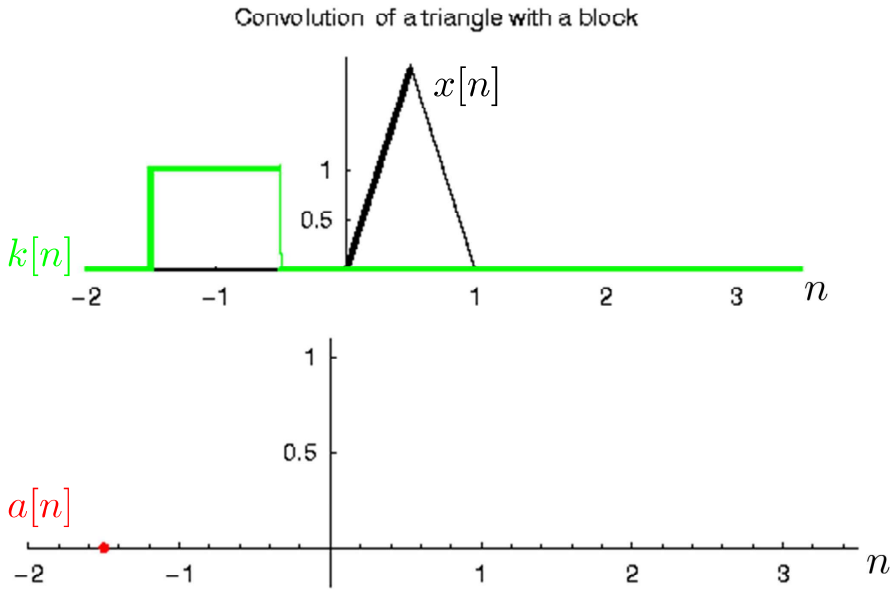
- Given two sequences $x[n]$ and $k[n]$, the convolved sequence $a[n]$ is defined by the following linear operation:

$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau], \quad \text{denoted by} \quad \boldsymbol{a} = \boldsymbol{x} * \boldsymbol{k}$$

- In practice, in CNNs, $\boldsymbol{k}$ only has a **finite support**: the sum is finite
- Terminology:
  - $\boldsymbol{x}$: **input** (or signal)
  - $\boldsymbol{k}$: convolution **kernel** (or filter)
  - $\boldsymbol{a}$: **feature map** (analog to pre-activation)
- 1D convolutions precisely describe ***Linear Time Invariant systems***
- **Ex:** moving average, smoothing, low-pass, band-pass, etc.

Discrete Convolution (1D Case)

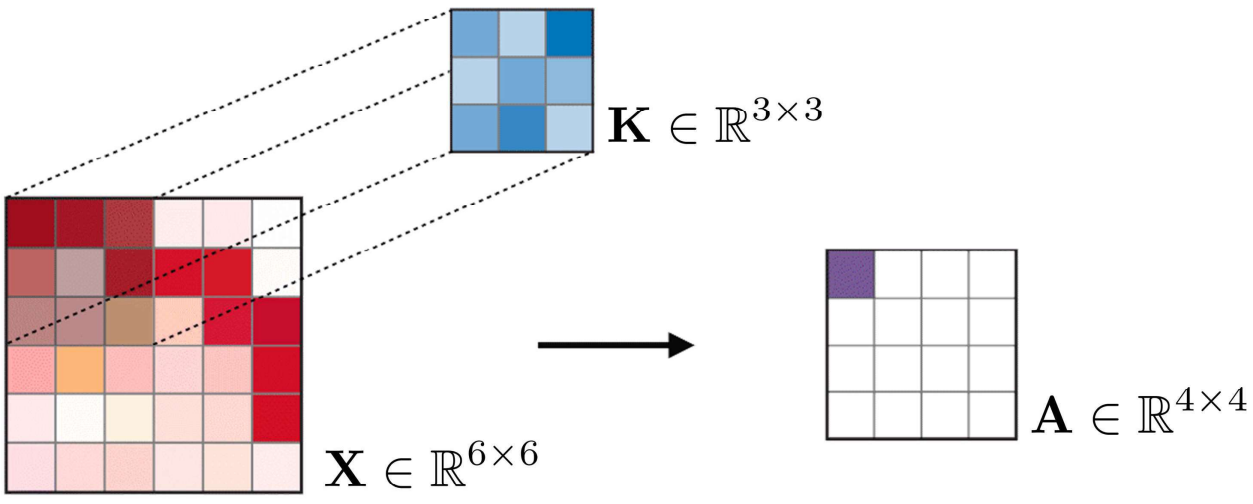
$$a[n] = \sum_{\tau=-\infty}^{+\infty} x[\tau]k[n-\tau]$$



Discrete Convolution (2D Case)

- Convolution can be generalized to **any** dimension
- For instance in 2D:

$$A[i,j] = (\mathbf{X} * \mathbf{K})[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n]K[i-m, j-n]$$



Discrete Convolution (2D Case)

- **Ex:** edge detection, sharpening, blurring, ….

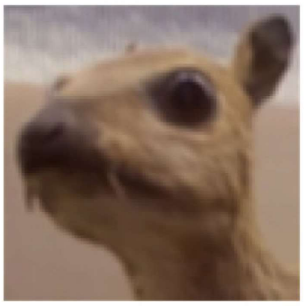
$$K = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

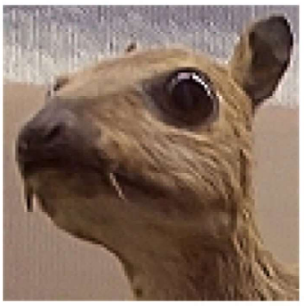


Discrete Convolution (2D Case)

- **Ex:** edge detection, sharpening, blurring, ….

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad K = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad K = \frac{1}{16}\begin{bmatrix} -1 & -2 & -1 \\ -2 & 28 & -2 \\ -1 & -2 & -1 \end{bmatrix}$$





Convolution vs. Correlation

- The convolution operation (using proper ***zero-padding*** in the discrete finite case) is **associative** and **commutative**:

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}, \quad \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c}) = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c}$$

- This is not the case of the related ***cross-correlation*** operation:

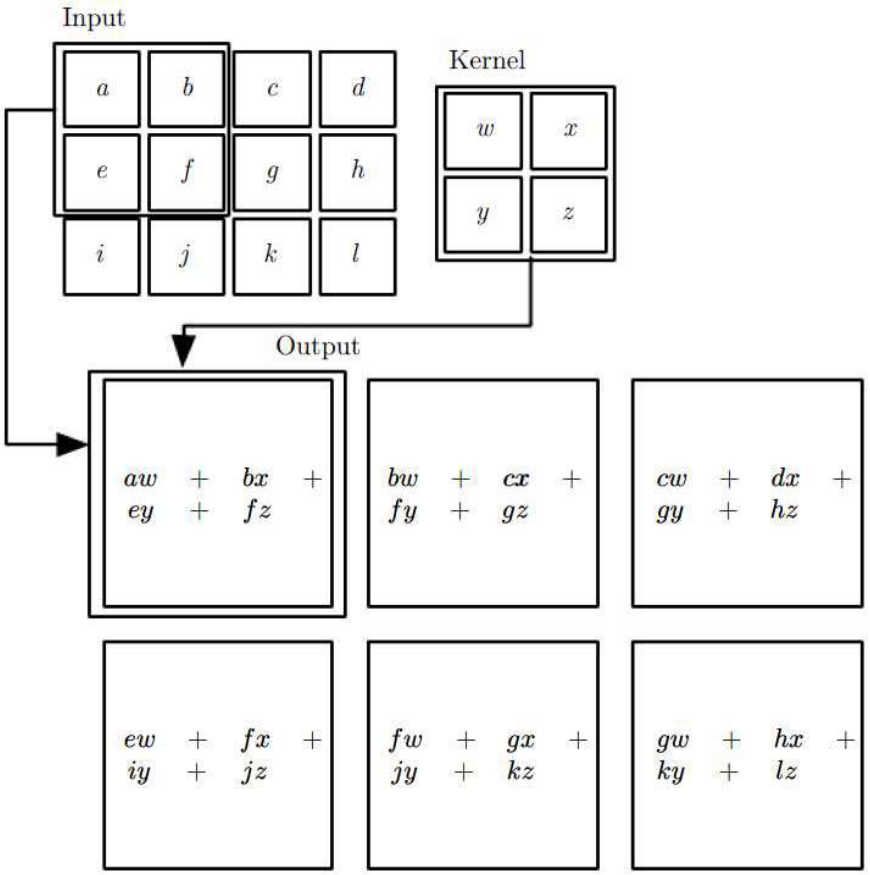
$$A[i,j] = \sum_{m=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} X[m,n]K[i+m,j+n]$$

➡ Equivalent to **flipping** the kernel along all axes

- Most machine learning libraries (eg. Pytorch, TensorFlow) implement cross-correlation but call it convolution

- In practice, since the kernel elements are **learned parameters**, it makes no difference (but worth keeping in mind!)

The maths:



Why Convolutional Neural Networks?

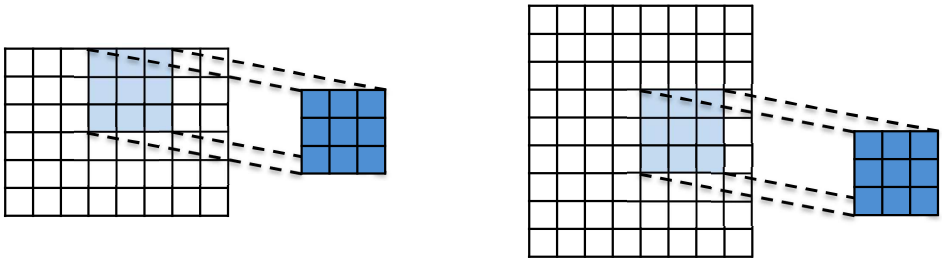
- Convolution leverages three key machine learning concepts:

  1) Sparse connectivity

  2) Parameter sharing

  3) Equivariant representation

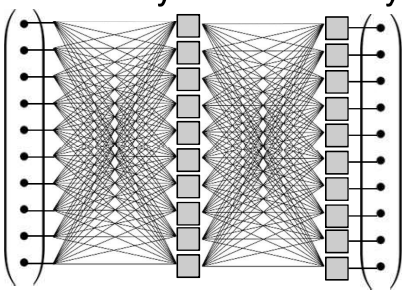
- Moreover, it provides a means for handling **variable size inputs**. The same kernel can be sled on signals/images of variable sizes:



Why Convolutional Neural Networks?

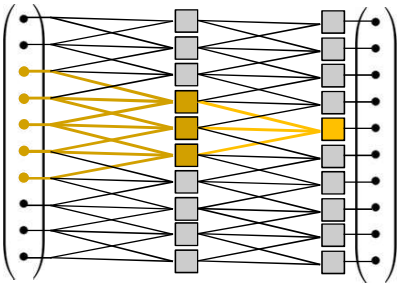
1) Sparse connectivity

Two conventional fully connected layers:

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

- Every output entry of a layer is connected to every input entry of the next

Two convolutional layers (kernel size S=3):

$\boldsymbol{x} \in \mathbb{R}^N$  $\boldsymbol{y} \in \mathbb{R}^N$

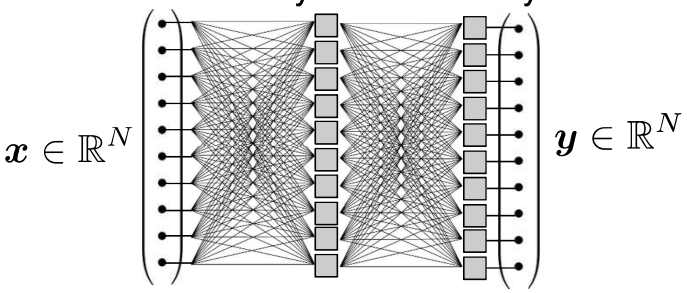
- Every output connected to only a **few neighboring inputs** due to the small kernel
- Still, in a deep CNN, each unit in late layers can interact with many inputs, forming a **receptive field**

Less connections* = *smaller model* = *less overfitting* & *faster computation

Why Convolutional Neural Networks?

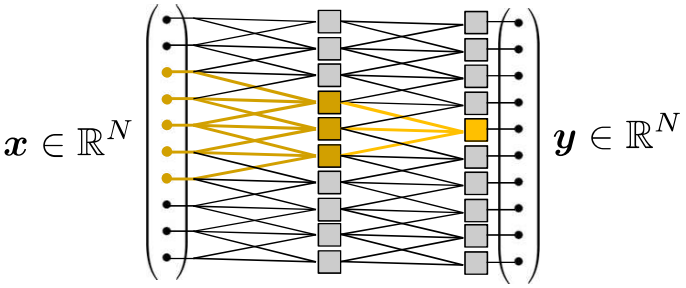
2) Parameter sharing

Two conventional fully connected layers:



- Every parameter is used **exactly once**
- ➡ $\mathcal{O}(LN^2)$ parameters

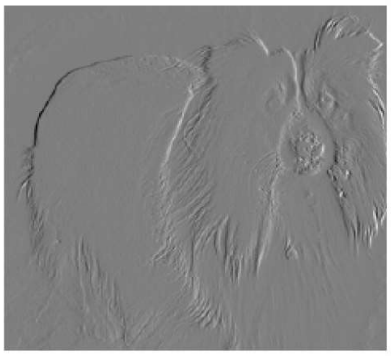
Two convolutional layers (kernel size S=3):



- Parameters **shared** (or ***tied***) across every position of the input
- ➡ $\mathcal{O}(LS)$ parameters
- Even **smaller model**, even **less overfitting**

Why Convolutional Neural Networks?

Computational efficiency



$$\mathbf{K} = \begin{array}{|c|c|}\hline \text{-1} & \text{1} \\ \hline \end{array}$$

- Example: horizontal edge detection (280 x 320 pixels)
  - **Convolutional:** 1 x 2 kernel, 280 x 319 ≈ 2 x 10^5 operations
  - **Fully connected**: 280 x 320 x 280 x 319 ≈ 8 x 10^9 weights ≈ 16 x 10^9 operations

Why Convolutional Neural Networks?

3) Equivariance

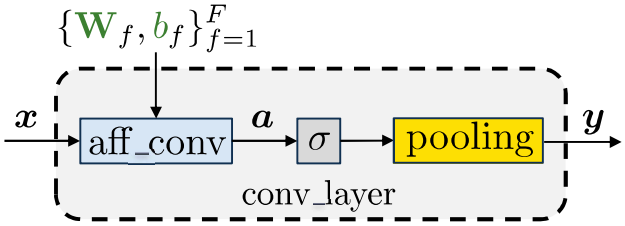
- **Definition:** If the input is transformed, then the output is **transformed the same way**
- /!\ **Different from invariance**: If the input is transformed, then the output is **unchanged**.
- Convolution is **equivariant to translation**: If the input signal is shifted (along its axes) by τ, then the output is also shifted by τ (convolution can even be **defined** by this).
- Hence, the **feature map** can be interpreted as indicating **where** some features (matching the kernels) appear in the input

Why Convolutional Neural Networks?

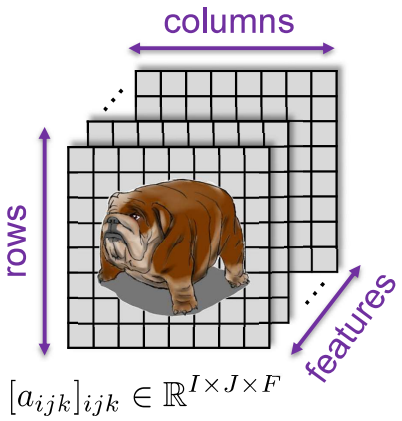
3) Equivariance

- **Example 1:** To process an **image**, because the objects will **look the same** no matter **where** they are in the image, it makes sense to use 2D convolution
- **Example 2:** To process a **speech signal**, because speech will **sound the same** no matter **when** it occurs in the signal, it makes sense to use 1D convolution
- **Note**: Convolution is **not** equivariant to rotation or scaling
- **Exercise:** Supposed my input signal is a list of incomes in a given city, sorted in ascending order? Supposed my input is a 2D map of the humidity of the soil around a given location?

CNN layers



- A **CNN layer** typically consists of 3 stages
- The **affine convolution** stage applies **multiple kernels** $\{\mathbf{W}_f\}_{f=1}^F$ to the input, yielding F feature maps, arranged in a **multi-way** tensor a .
- A **bias term** b_f is added to each feature map
- A **nonlinear activation function** σ is then applied to all feature maps (e.g. ReLU)
- There is then a **pooling** stage, that replaces every **neighborhood** in a given feature map by a **summary statistics** (ex: mean or max)



Pooling

- Pooling makes the representation **more invariant** to **small translations** of the input
- Translation invariance is useful when we care more about whether some feature is **present** than exactly **where** it is
- **Example:** to detect a face, we search for an eye on the left side and an eye on the right side, but we don't need to locate them with pixel-level accuracy.
- **Counter-example:** to denoise an image, we must preserve the location of the features. In that situation pooling is not desirable..

Pooling

- Since pooling summarizes the responses over a **neighborhood**, we often report summary statistics **every** *S* **pixels** instead of every 1 pixel
- *S* is called the **stride**. It can be different for different spatial directions
- This drastically improves the **computational efficiency**
  - Typical stride values are 2 or 3 along each axes.
  - For an image, this amounts to a **x4** or **x9** **dimensionality reduction** over feature maps
- Also useful when dealing with **variable size** input: adjust the pooling and stride to obtain a fixed-size output

Pooling

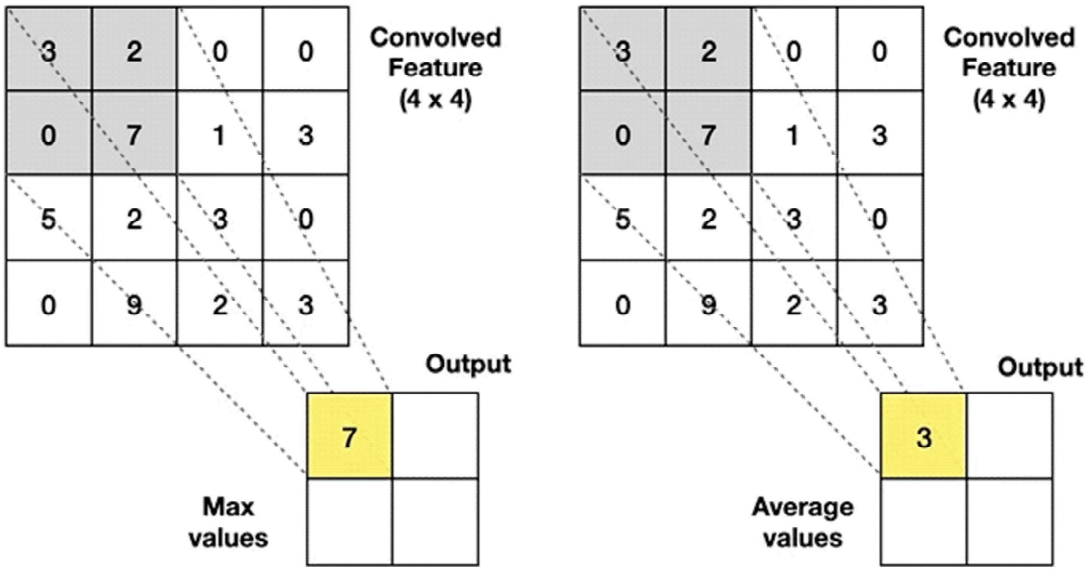
Max Pooling

Take the **highest** value from the area covered by the kernel

Average Pooling

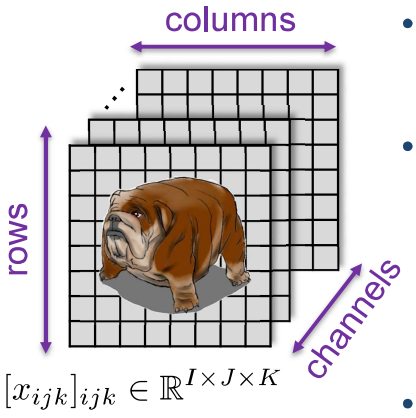
Calculate the **average** value from the area covered by the kernel

Example: Kernel of size 2 x 2; stride=(2,2)



Dealing with multichannel input

- In contrast to classical convolution, each **entry** of the input of a **convolution layer** in a CNN is usually a **vector**, not a **scalar**.
  - **Ex1**: RGB **channels** in an image
  - **Ex2**: Multiple **feature maps** obtained in a previous layer
- Hence, the **input** of a 2D conv. layer is also a **3-way tensor**:

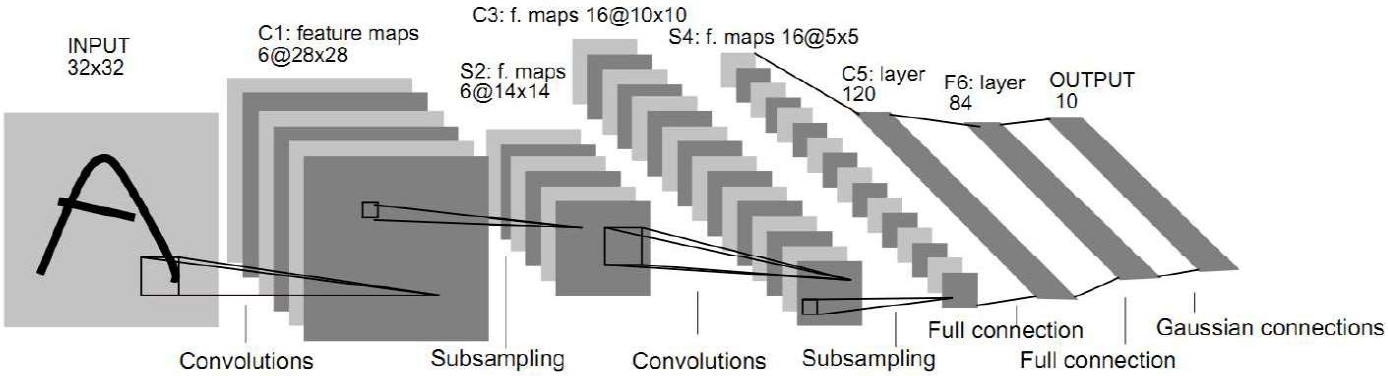


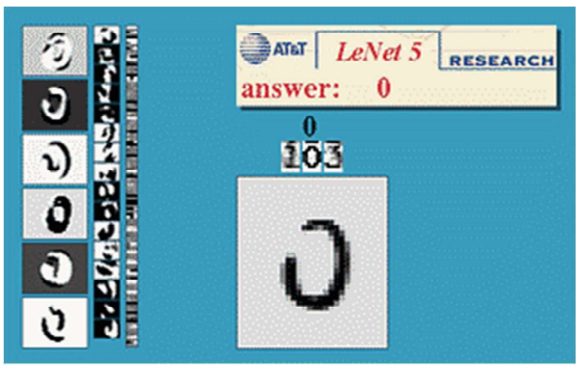
- Convolution over the channels (ie. 3D conv.) does not make sense (by def. of "channels")
- In standard multichannel convolution, each of the K **input channel** is **fully connected** to all of the F feature maps, i.e., **output channels**:

$$a_{ijf} = (\mathbf{X} *_{1,2} \mathbf{W})_{ijf} = \sum_k \sum_m \sum_n x_{ijk} w_{i+m,j+n,f,k}$$

- Each **entry** of the kernel acts as a $F \times K$ matrix: the list of kernels $\mathbf{W}$ form a **4-way** tensor

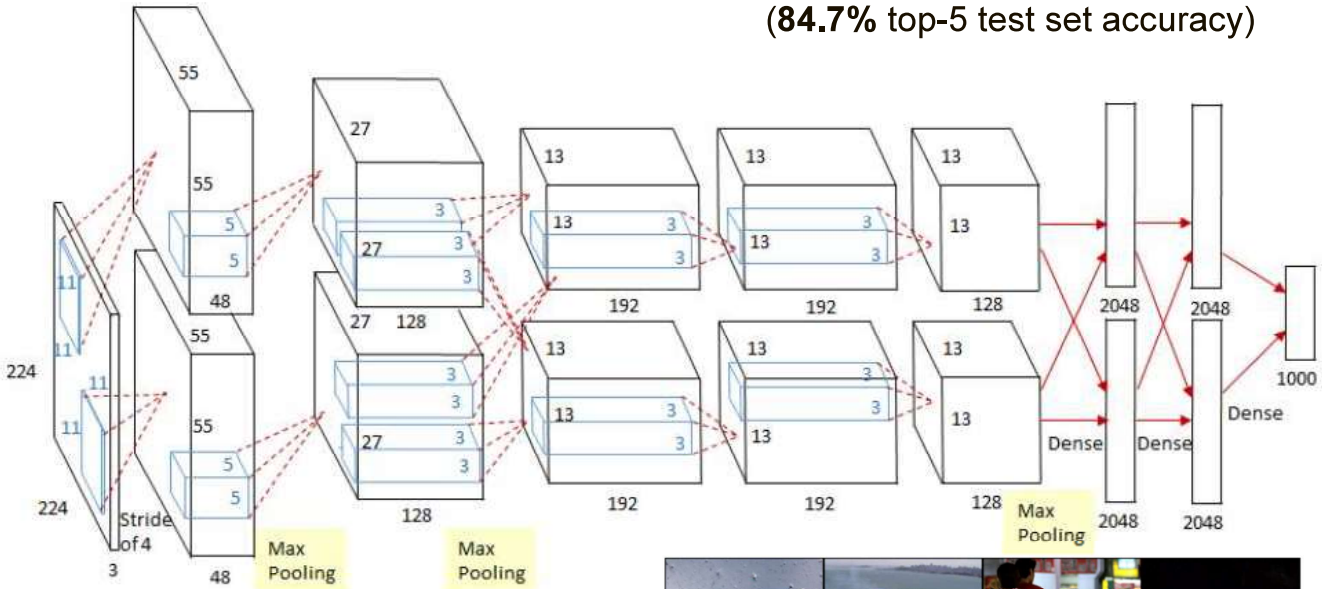
Famous example: LeNet-5 for digit recognition (1989)





Famous example: AlexNet for ImageNet classification (2012)

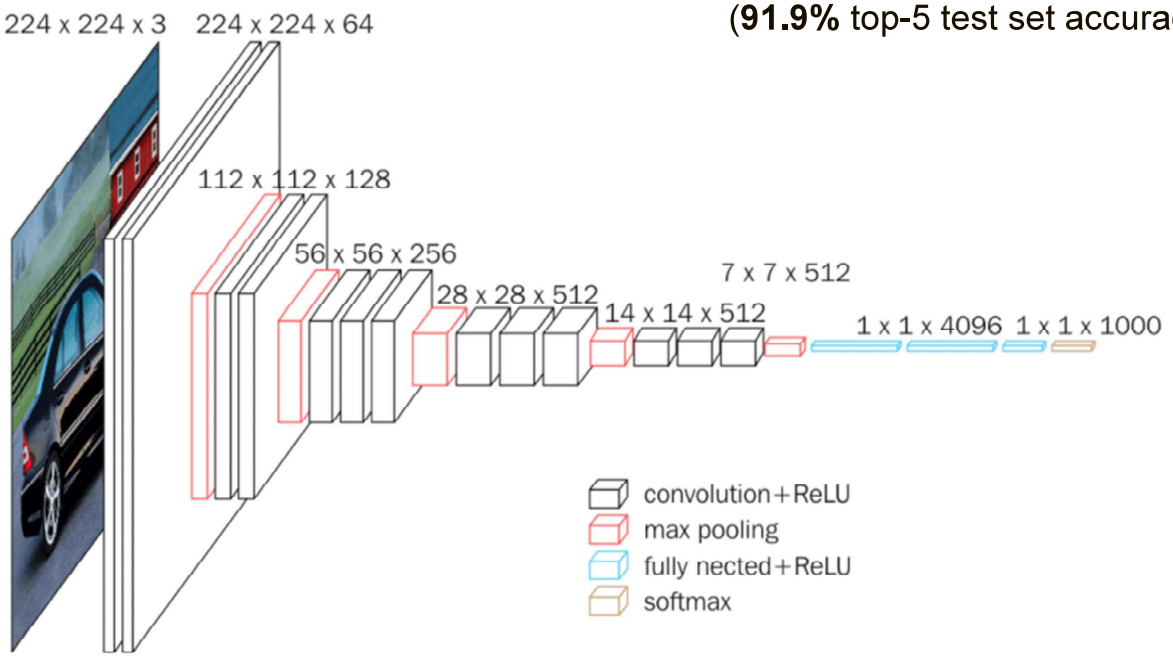
(**84.7%** top-5 test set accuracy)



Proven not so useful in the end

Famous example: VGG-16 for ImageNet classification (2014)

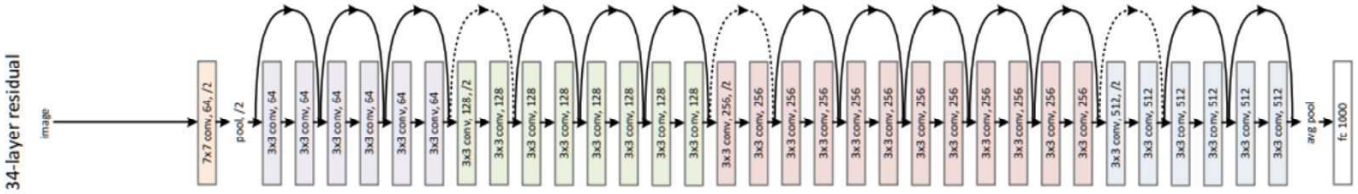
(**91.9%** top-5 test set accuracy)



- Uses only 3x3 kernels, but more depth (16 vs. 7 layers)

Famous example: ResNet for ImageNet classification (2015)

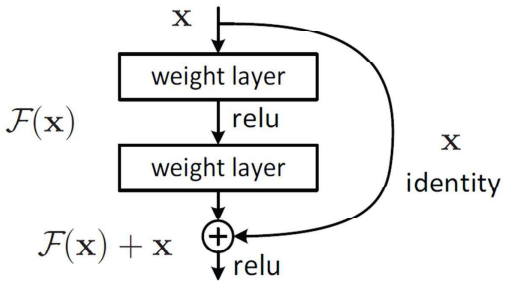
94.3% top-5 for ResNet-152 / **95.2%** for ResNet-200 (2016)



Note:
- Human top-5 accuracy reported at **94.9%**
- Perf. on ImageNet ≈plateaus around **98%** since 2018

Key idea:

- **Residual** or ***skip* connections** that "passes through" several layers
- Solves the problem of ***vanishing gradient***
- **Recall:**



$$\boldsymbol{G}_{\boldsymbol{b}^1} = \sigma'(\boldsymbol{a}_t^1) \odot \mathbf{W}^{2\top}\sigma'(\boldsymbol{a}_t^2) \odot \mathbf{W}^{3\top} \ldots \sigma'(\boldsymbol{a}_t^3) \odot \mathbf{W}^{L\top}\sigma'(\boldsymbol{a}_t^L) \odot \nabla_{\boldsymbol{x}^L}\ell(\boldsymbol{x}_t^L, \boldsymbol{y}_t)$$

OUTLINE