



Lecture 1: Introduction

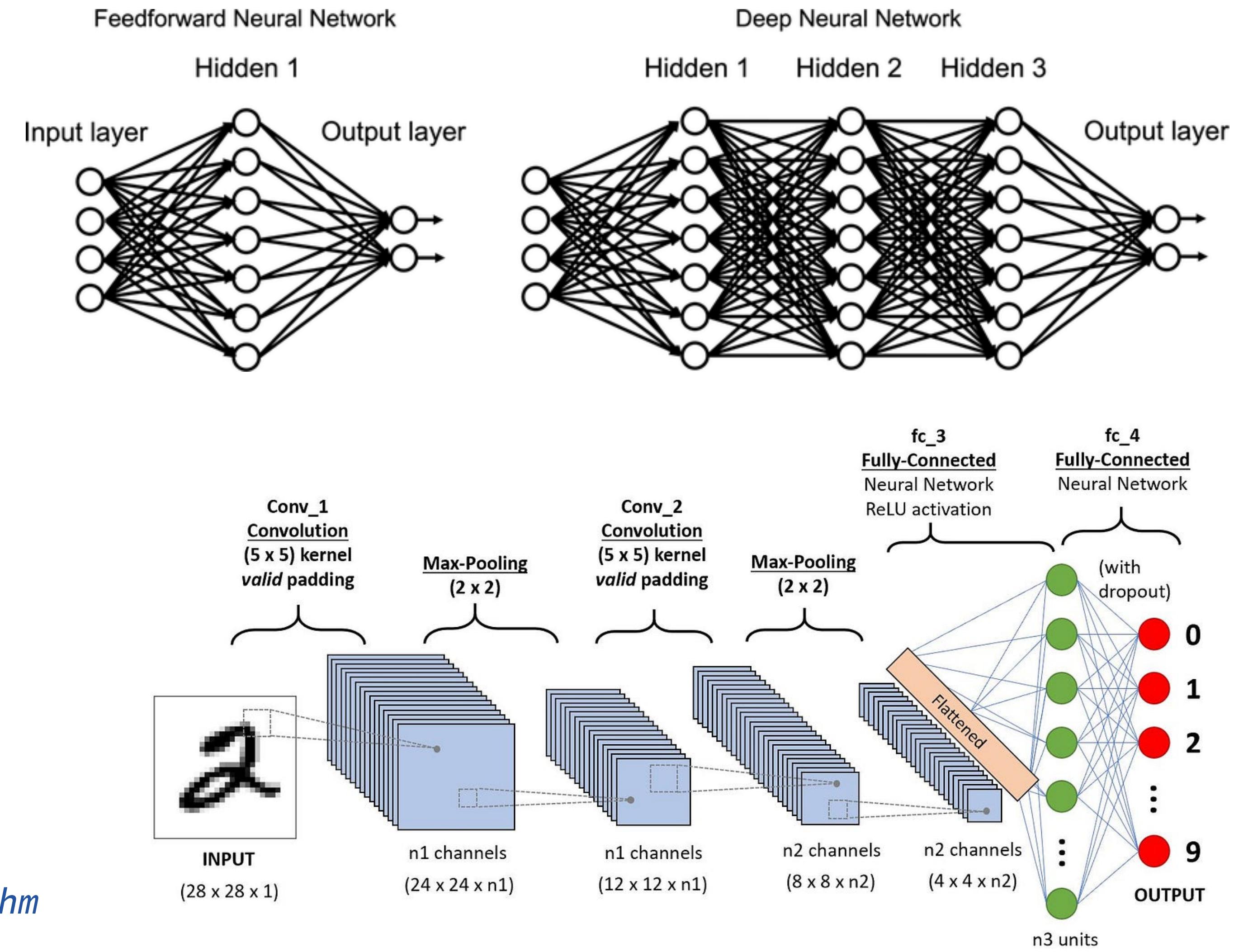
Class Overview

- This series of lectures will give you an entry point on the quickly evolving domain of Deep Learning, which is driving the recent boost of AI-related research
- The focus is on applications: we will be light on formalism and will devote substantial time to code writing
- No scientific background is required: we will use standard datasets like [MNIST](#) for most of the activity. Physics concepts, when needed, will be introduced
- Nevertheless, this is a physics course, you will see some (particle) physics



The topics

- General introduction to Machine Learning
- Deep Learning Architectures
 - Fully-connected NNs (aka Dense NNs)
 - Convolutional NNs
 - Recurrent NNs
 - Graph NNs
 - Transformers
 - Normalizing Flows
- Learning processes and applications
 - Supervised
 - Unsupervised
 - Adversarial
 - Anomaly Detection
- Deep Learning in practice
 - Training: How to optimize a model
 - Inference: How to deploy and use an algorithm
 - Model Compression and edge computing





The topics

● General introduction to Machine Learning

● Deep Learning Architectures

● Fully-connected NNs (aka Dense NNs)

● Convolutional NNs

● Recurrent NNs

● Graph NNs

● Transformers

● Normalizing Flows

● Learning processes and applications

● Supervised

● Unsupervised

● Adversarial

● Anomaly Detection

● Deep Learning in practice

● Training: How to optimize a model

● Inference: How to deploy and use an algorithm

● Model Compression and edge computing

Lectures in blue

Tutorials in red italic

Date	Topic	Tutorial
Sep 17	Intro & class description	Linear Algebra in a nutshell + prob and stat
Sep 24	Basic of machine learning + Dense NN	<i>Basic jupyter + DNN on mnist (give jet dnn as homework)</i>
Oct 1	Convolutional NN	<i>Convolutional NNs with MNIST</i>
Oct 8	Training in practice: regularization, optimization, etc	<i>Practical methodology</i>
Oct 15		<i>Tensor Flow tutorial (tbc)</i>
Oct 22	Recurrent NN	<i>Tutorial on RNNs</i>
Oct 29	Graph NNs	<i>Tutorial on Graph NNs</i>
Nov 5	Unsupervised learning and anomaly detection	<i>Autoencoders with MNIST</i>
Nov 12	Generative models: GANs, VAEs, etc	Normalizing flows
Nov 19	Spiking Neural Network	<i>Tutorial on neuromorphic chips & spiking NNs (tbc)</i>
Nov 26	Network compression (pruning, quantization, Knowledge Distillation)	
Dec 3		<i>Tutorial on hls4ml/qkeras</i>
Dec 10		Transformers
Dec 17		tbd



Teaching Style: lectures

- We like to adapt the lectures to the audience
- MP gave some of these lectures to PhD students and high-school teachers/students with equally good results
- To do so, we need feedback: **interrupt at any time with questions.**
- If we hear nothing, we assume that everything is clear and we keep going (fast)
- We don't have to get to the end of the lecture series: we prefer doing 1/2 of the program well than rushing to the end



Teaching Style: tutorials

- We envisioned full hands-on tutorials but time is limited

- Most of the time, it will be a walkthrough pre-written code

- Notable exceptions:

- On Oct 8th we will do a full exercise in class from scratch
- (tentatively) on Oct 15th we will have a long tutorial on Google TensorFlow [Pending confirmation by Google TF team in Zurich]
- On Oct 19th we are planning a special tutorial on Neuromorphic Chips (we are waiting for confirmation). As a backup, we might do a Quantum ML tutorial instead.
- (tentatively) on Dec 3rd we will have a long tutorial on hls4ml [special lecturers from CERN]

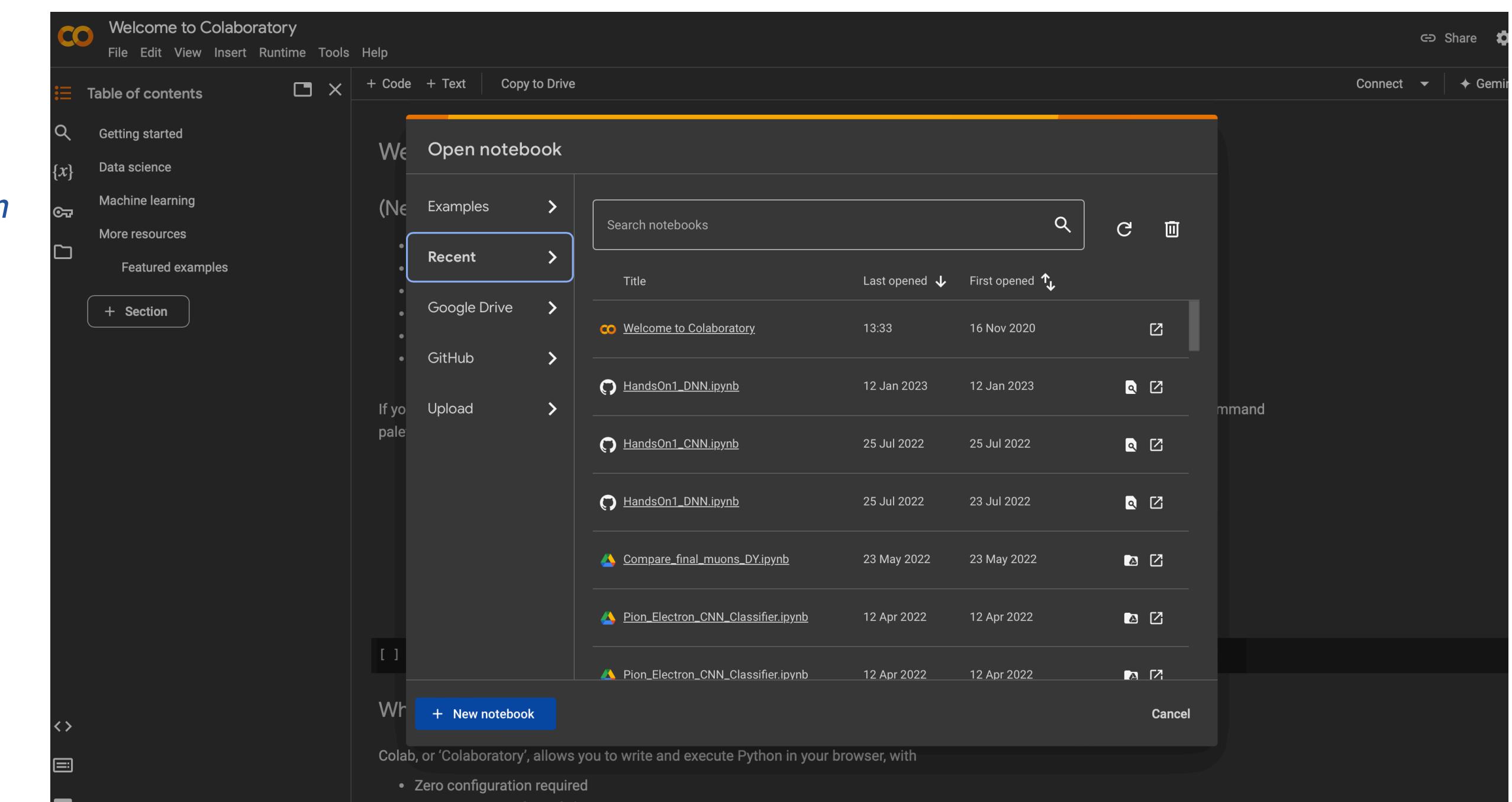
- We will use PYTHON jupyter notebooks in COLAB

- You need to have a working google account on COLAB for next week
- The notebooks run on the cloud, so you can use any device with a network (from your smartphone to your laptop)

- To familiarize with the tutorials, you will have to practice at home

- Repeat the tutorial, change things, and try to get a better result

<https://colab.research.google.com/>





Python & Jupyter

- We are assuming some previous knowledge of python
- But we will use very minimal python knowledge
- Most of what you need to know will be available on keras.io and looking for examples on googles
- It might help to refresh your jupiter/matplotlib skills with some online tutorial
- You could try [this](#) (or similar) on colab directly ([instructions here](#))
- Similarly [here](#) for matplotlib

DNN model building

```
In [ ]:  
# keras imports  
from tensorflow.keras.models import Model  
from tensorflow.keras.layers import Dense, Input, Dropout, Flatten, Activation  
from tensorflow.keras.utils import plot_model  
from tensorflow.keras import backend as K  
from tensorflow.keras import metrics  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, TerminateOnNaN
```

```
In [ ]:  
input_shape = X_train.shape[1]  
dropoutRate = 0.25
```

```
In [ ]:  
####  
inputArray = Input(shape=(input_shape,))  
#  
x = Dense(40, activation='relu')(inputArray)  
x = Dropout(dropoutRate)(x)  
#  
x = Dense(20)(x)  
x = Activation('relu')(x)  
x = Dropout(dropoutRate)(x)  
#  
x = Dense(10, activation='relu')(x)  
x = Dropout(dropoutRate)(x)  
#  
x = Dense(5, activation='relu')(x)  
#  
output = Dense(5, activation='softmax')(x)  
####  
model = Model(inputs=inputArray, outputs=output)
```

```
In [ ]:  
model.compile(loss='categorical_crossentropy', optimizer='adam')  
model.summary()
```

We now train the model

```
In [ ]:  
batch_size = 128  
n_epochs = 50
```

```
In [ ]:  
# train  
history = model.fit(X_train, y_train, epochs=n_epochs, batch_size=batch_size, verbose = 2,  
validation_data=(X_val, y_val),  
callbacks =[  
EarlyStopping(monitor='val_loss', patience=10, verbose=1),  
ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2, verbose=1),  
TerminateOnNaN()])
```

```
In [ ]:  
# plot training history  
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.yscale('log')  
plt.title('Training History')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['training', 'validation'], loc='upper right')  
plt.show()
```



Final Exam

- *The final exam will consist of two parts*
- *PRACTICAL: you will be asked to demonstrate your capability to design and train a Deep Learning algorithm to solve a task*
- *We will assign you a problem on a new dataset*
- *You will have to submit a notebook with a solution by the exam*
- *The first part of the oral exam will consist in a discussion of your work*
- *THEORETICAL: you will have to demonstrate familiarity with the topics discussed in class*
- *After the exercise discussion, some Q&A on various topics discussed in classes, not necessarily related to the exercise*

Textbook

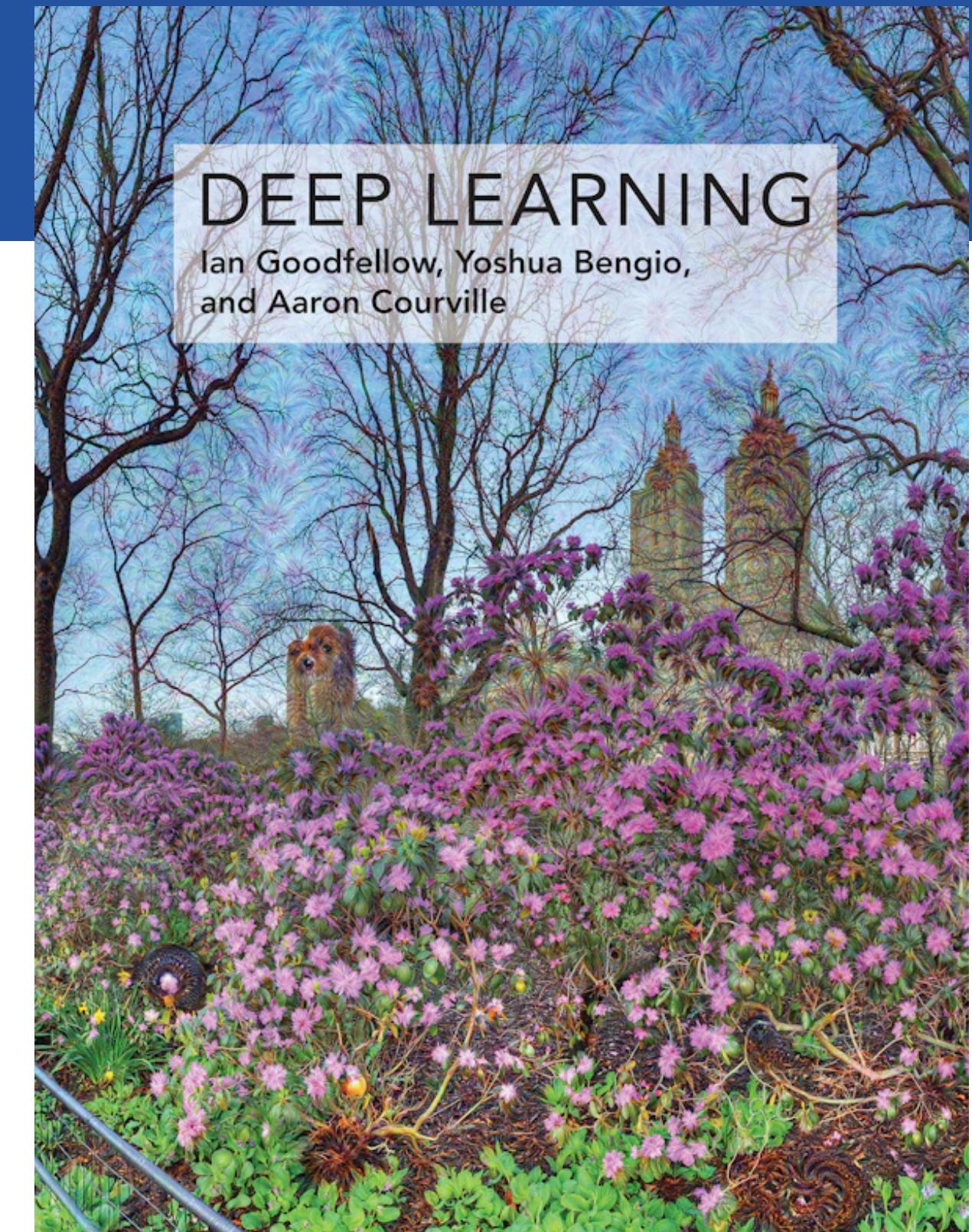
- The main reference for you will be the electronic materials

- Slides
- Tutorial notebooks

The material will be posted on MP's github progressively

- In addition, we will loosely follow “Deep Learning” by Goodfellow et al. (MIT Press)

- You can find an html version by the authors here
- You can find it in the library
- **BEWARE:** This book is (relatively) old (2016) so some of the most recent topics (e.g., transformers) are not covered.
- We will provide dedicated material for those



Melden Sie sich für Bestellungen und weitere Dienstleistungen an  Anmelden / Registrieren  SCHLIESSEN

Zurück Liste der Versionen anzeigen

Aktive Filter
Liste der Versionen 
 0 ausgewählt 2 Ergebnisse
 Alle Filter merken
 Filter zurücksetzen

Ergebnisse optimieren

Sortieren nach Datum - neuestes 

Zeige nur  In swisscovery-Bibliotheken verfügbar

ZB / Universität Zürich 
UB Zürich, Politikwissenschaft 
UB Zürich, Volkswirtschaft 
UB Zürich, Naturwissenschaften 

BUCH Deep learning
Goodfellow, Ian, Cambridge, Massachusetts; London, England : The MIT Press, [2016]; ©2016
 Online verfügbar >

BUCH Deep learning
Goodfellow, Ian 1987- (Autor_in); Bengio, Yoshua (Autor_in); Courville, Aaron (Autor_in)
Cambridge, Massachusetts : London, England : The MIT Press, [2016]; © 2016
 Verfügbar bei UB Zürich, Volkswirtschaft. Frei zugänglich (ECON V16 0391) und anderen Standorten >

Ergebnisse pro Seite 25 50



Textbook

- The main reference for you will be the electronic materials

- Slides
- Tutorial notebooks

The material will be posted on [MP's github](#) progressively

- In addition, we will loosely follow “Deep Learning” by [Goodfellow et al.](#) ([MIT Press](#))

- You can find an [html version by the authors](#) here
- You can find it in the library
- **BEWARE:** This book is (relatively) old (2016) so some of the most recent topics (e.g., transformers) are not covered.
- We will provide dedicated material for those

- [Table of Contents](#)
- [Acknowledgements](#)
- [Notation](#)
- [1 Introduction](#)
- [Part I: Applied Math and Machine Learning Basics](#)
 - ✓ [2 Linear Algebra](#)
 - ✓ [3 Probability and Information Theory](#)
 - ✓ [4 Numerical Computation](#)
 - ✓ [5 Machine Learning Basics](#)
- [Part II: Modern Practical Deep Networks](#)
 - ✓ [6 Deep Feedforward Networks](#)
 - ✓ [7 Regularization for Deep Learning](#)
 - ✓ [8 Optimization for Training Deep Models](#)
 - ✓ [9 Convolutional Networks](#)
 - ✓ [10 Sequence Modeling: Recurrent and Recursive Nets](#)
 - ✓ [11 Practical Methodology](#)
 - ✓ [12 Applications](#)
- [Part III: Deep Learning Research](#)
 - [13 Linear Factor Models](#)
 - ✓ [14 Autoencoders](#)
 - [15 Representation Learning](#)
 - [16 Structured Probabilistic Models for Deep Learning](#)
 - [17 Monte Carlo Methods](#)
 - [18 Confronting the Partition Function](#)
 - [19 Approximate Inference](#)
 - ✓ [20 Deep Generative Models](#)
- [Bibliography](#)
- [Index](#)



✓: Parts that we will cover

Additional material

- Depending on your interest for various topics, we can provide you with further readings:
- e.g., reviews on specific topics, e.g. ["Graph Neural Networks in Particle Physics"](#)
- For generic topics, you can find very good introductory material on
 - [wikipedia.org](#)
 - [townrdsdatascience.org](#)
- For code issues
 - You are not the first having the problem you have: check on [stackoverflow.co](#) or just [google your problem](#)

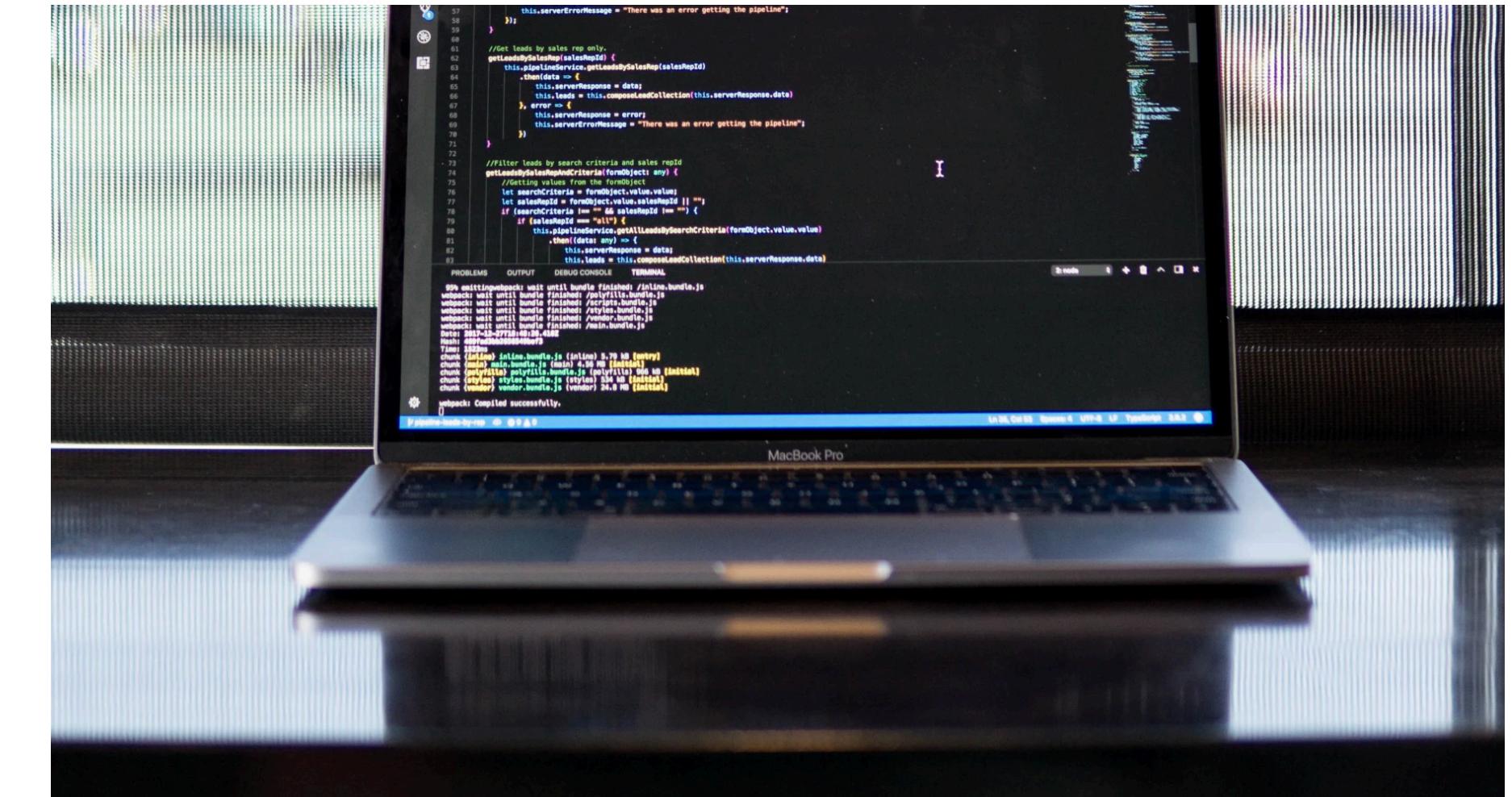


Photo by Maxwell Nelson on [Unsplash](#)

Introduction To Autoencoders

A Brief Overview

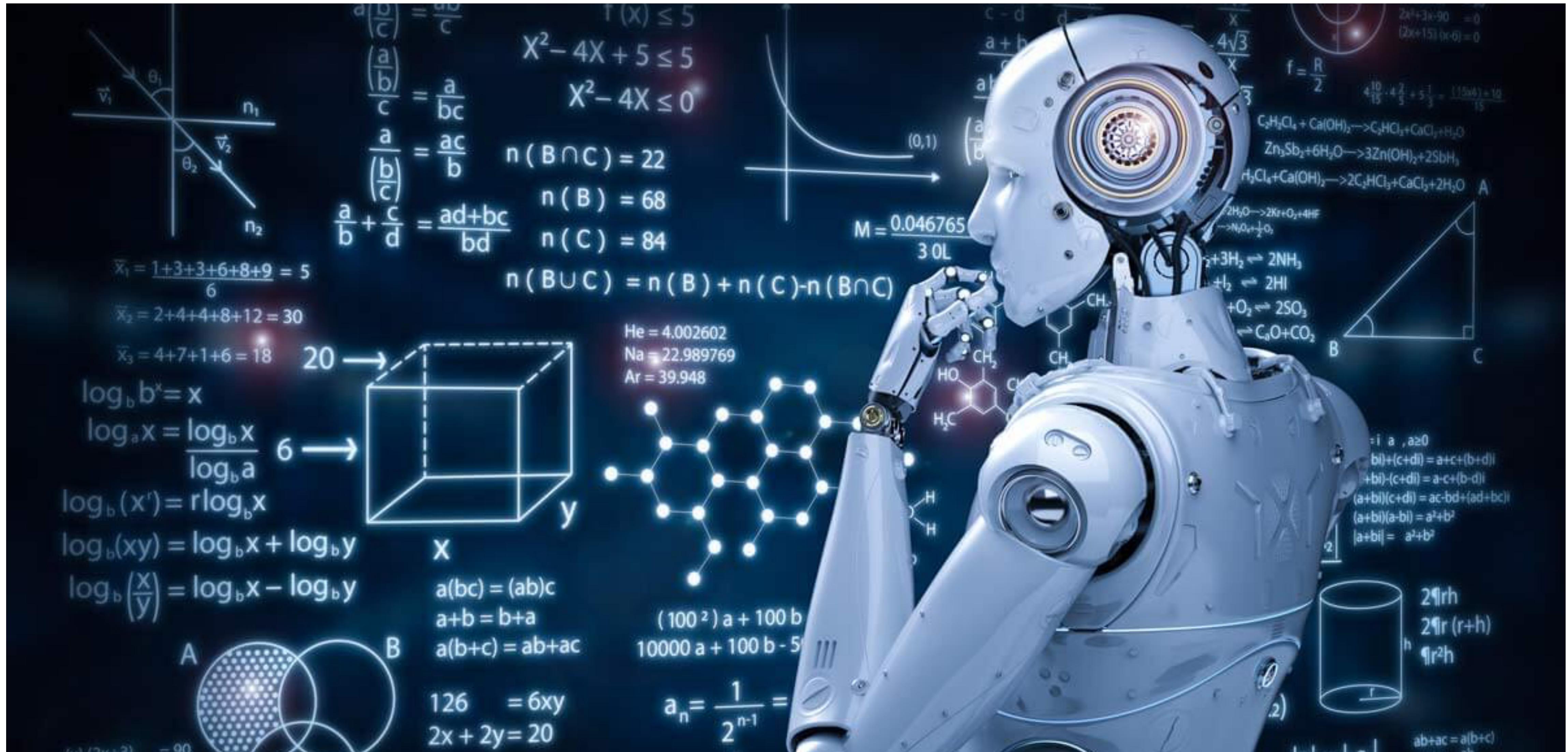
 Abhijit Roy · Follow
Published in Towards Data Science · 14 min read · Dec 12, 2020

98 1 ⌂ ⌂ ⌂

Autoencoders are neural network-based models that are used for unsupervised learning purposes to discover underlying correlations among data and represent data in a smaller dimension. The autoencoders frame unsupervised learning problems as supervised learning problems to train a

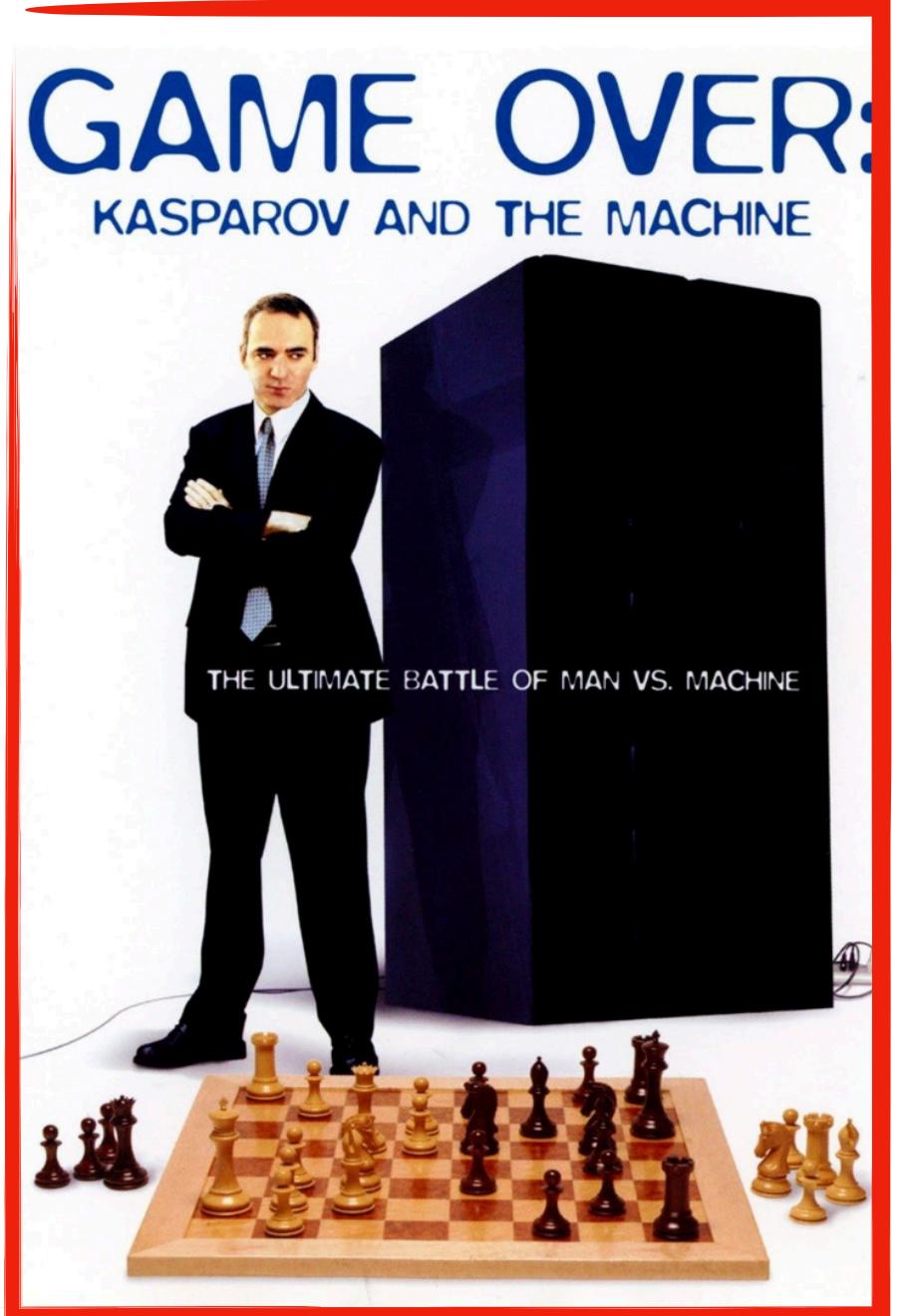


What is Machine Learning?



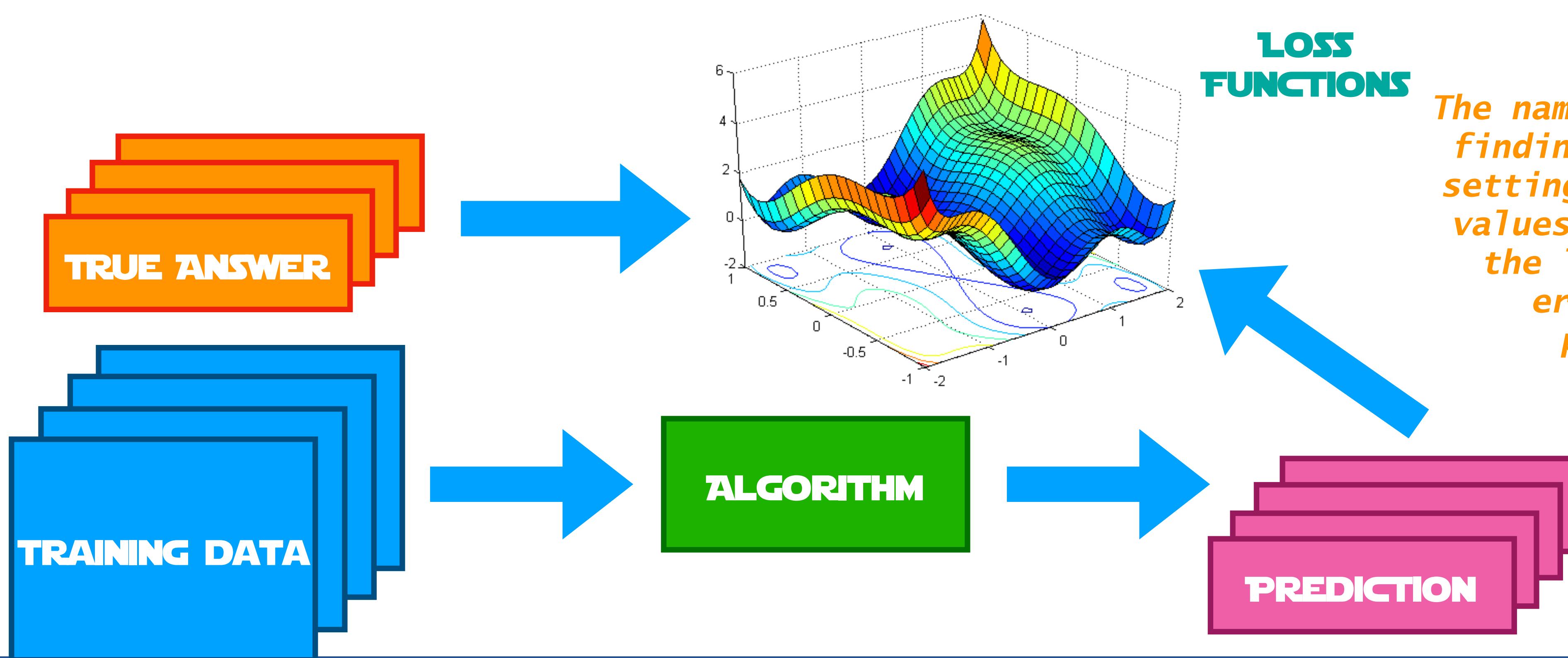
A historical perspective

- Since (at least) ancient Greece, humankind has been dreaming about the concept of a thinking machine
- This idea was revamped in '800, when programmable computers appear
- This fuelled research in AI during/after WWII
 - Very quickly, it was possible to solve human hard problems that are trivial for computers (e.g., when formulated as simple mathematical rules). Most of these solutions are so-called "**rule-based**" algorithms
 - The challenge of AI is in solving human hard problems which are difficult to formalise as a list of mathematical rules
 - Machine Learning is the field of research between AI, Computer Science and Statistics that addresses this issue with a **Learn-by-example methodology**



A definition

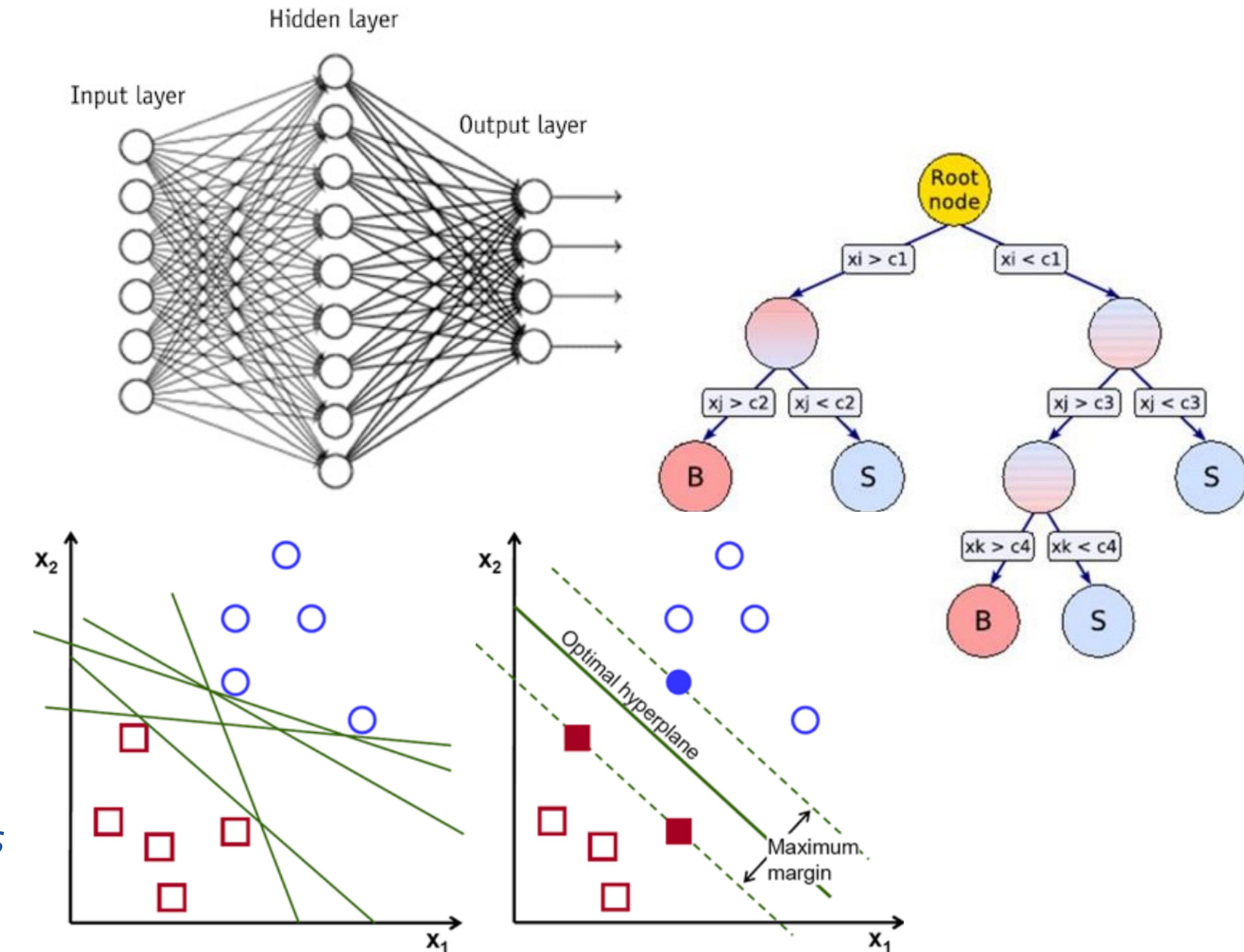
Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.



The name of the game is finding the algorithm setting (its parameter values) that minimise the loss, i.e. the error made in prediction

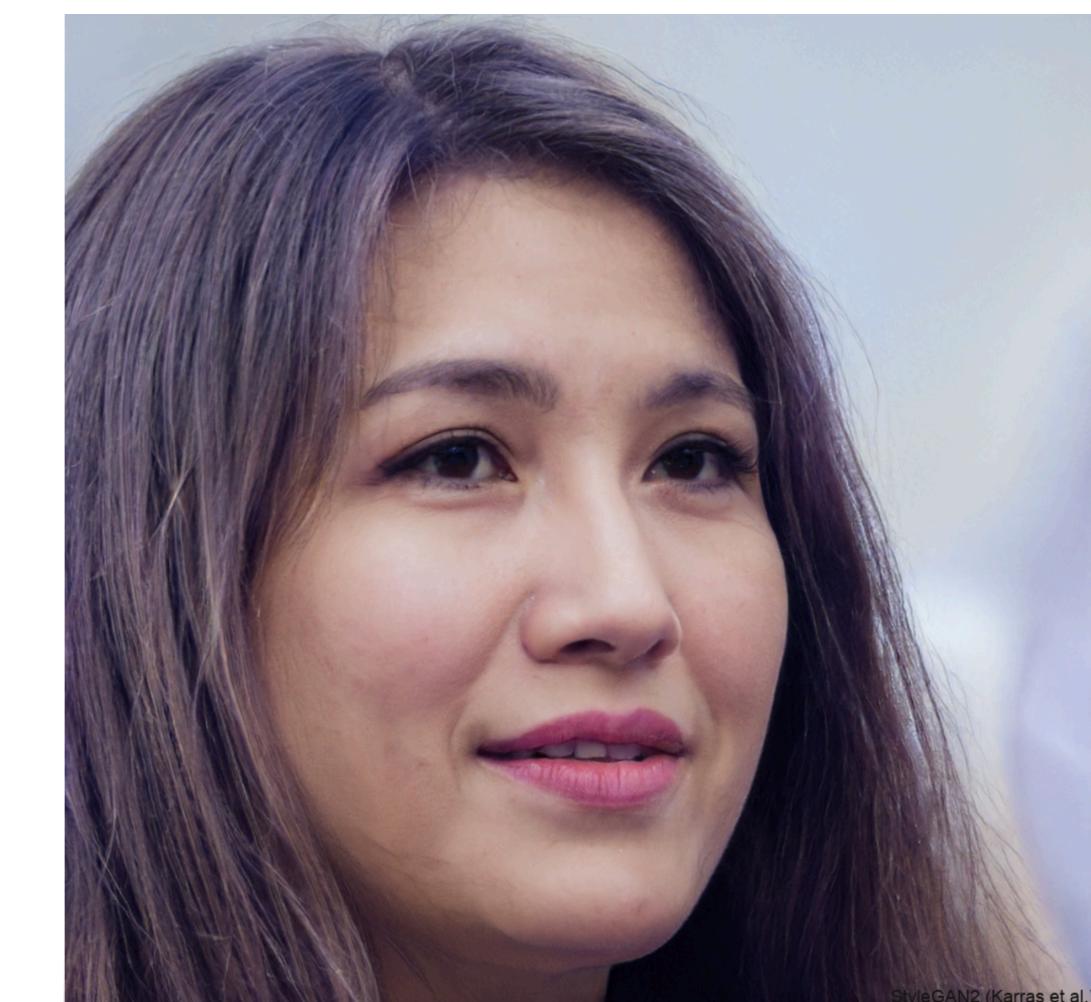
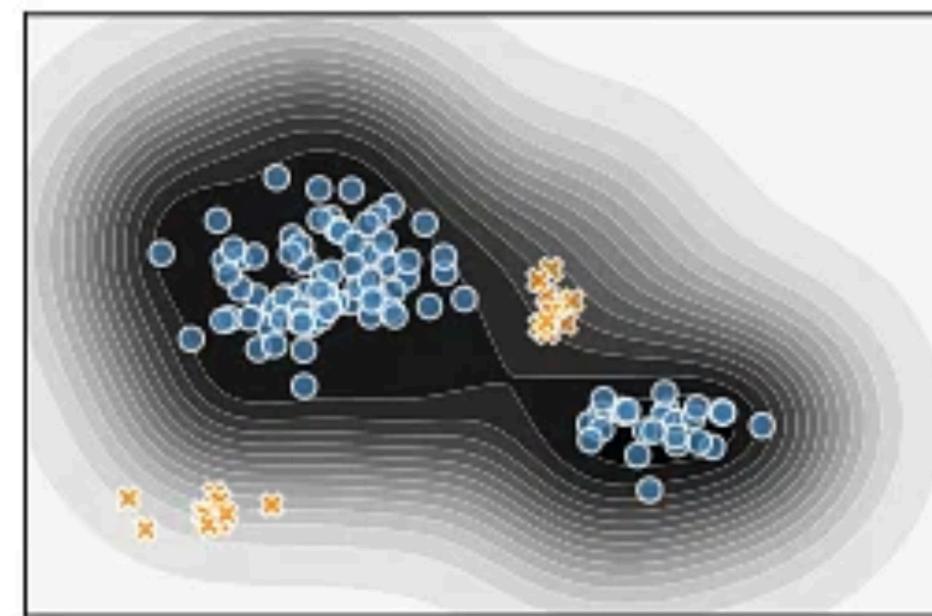
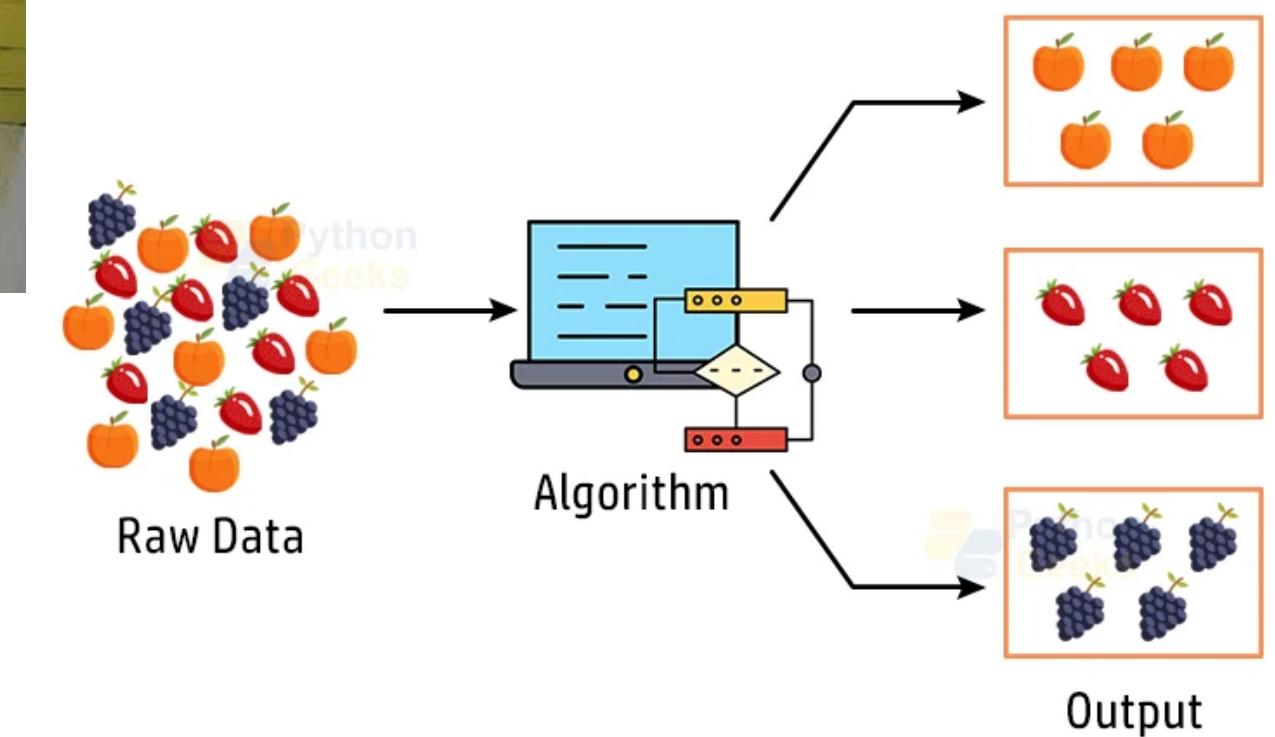
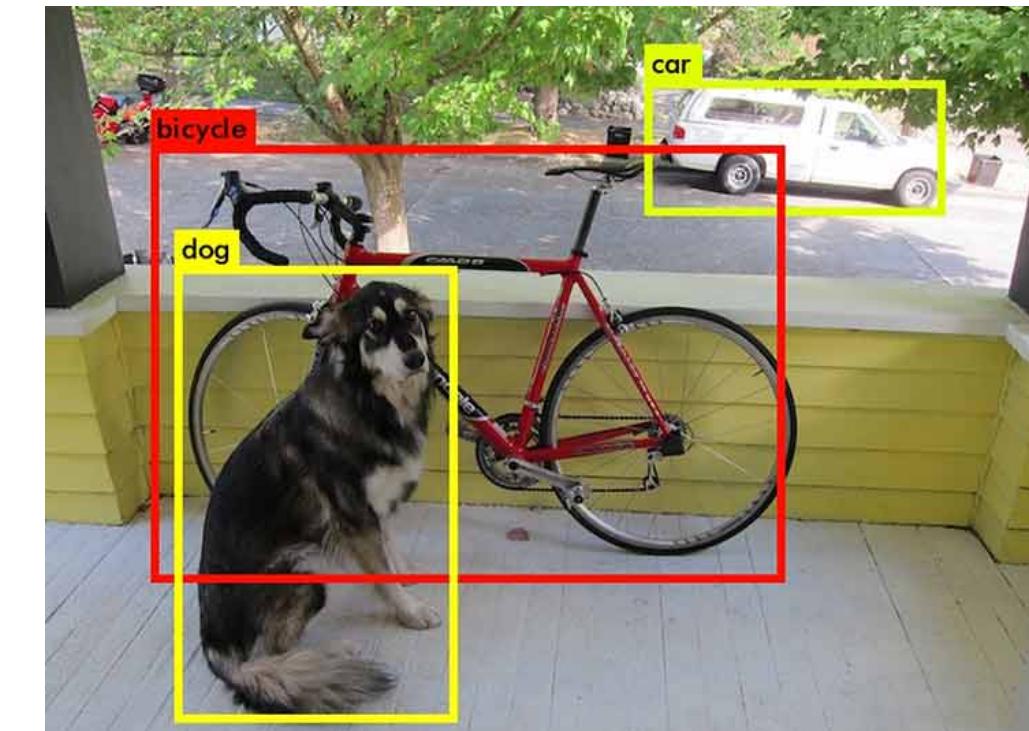
many ML algorithms

- In different moments, different algorithms were at the edge of ML research
- (Shallow) neural networks dominated the scene up to the 80's
- Alternatives emerged in the 90's
- Support vector machine
- Boosting of decision trees
- These classes of ML algorithms can be used for various tasks



Many Problems to Solve

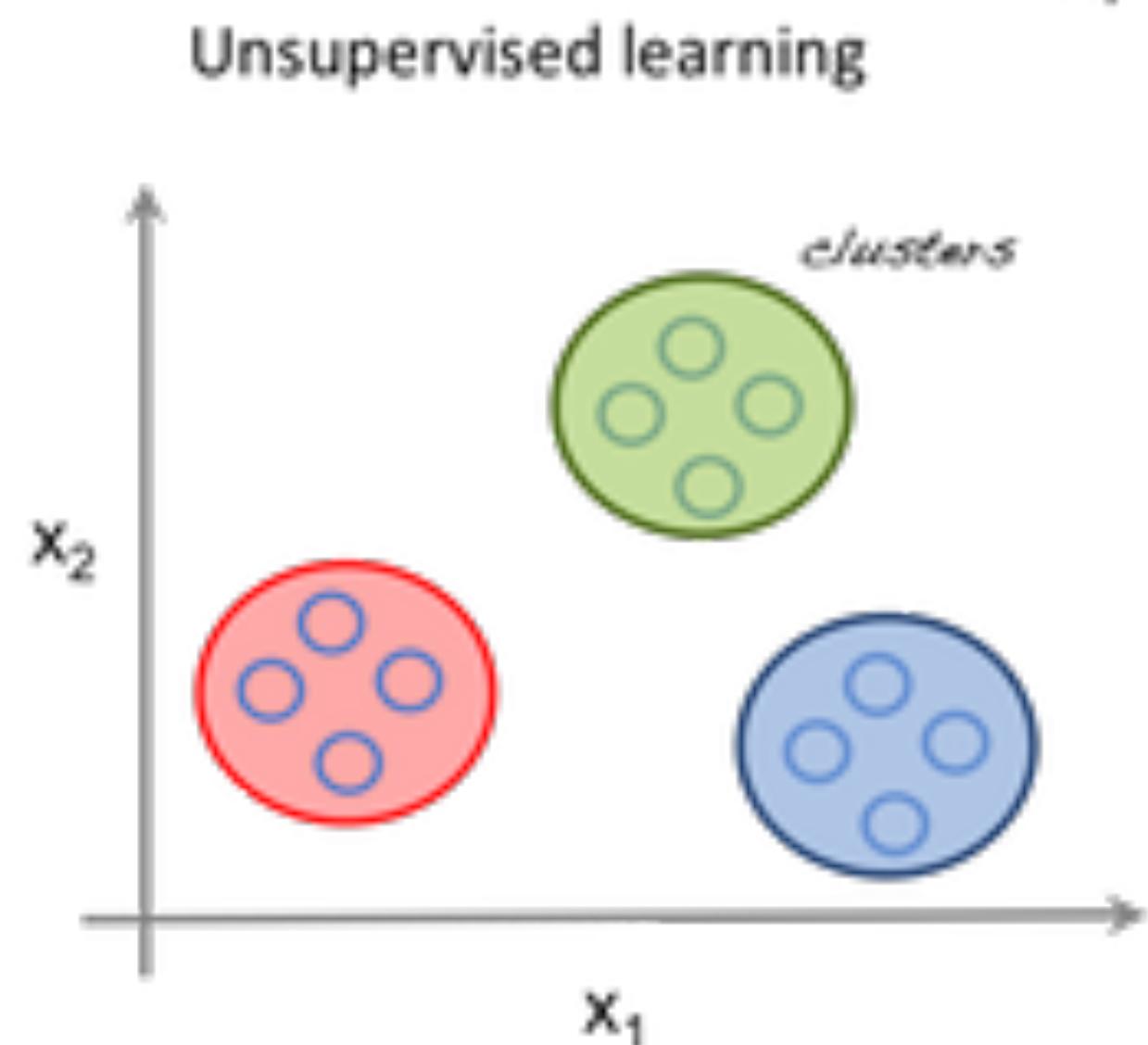
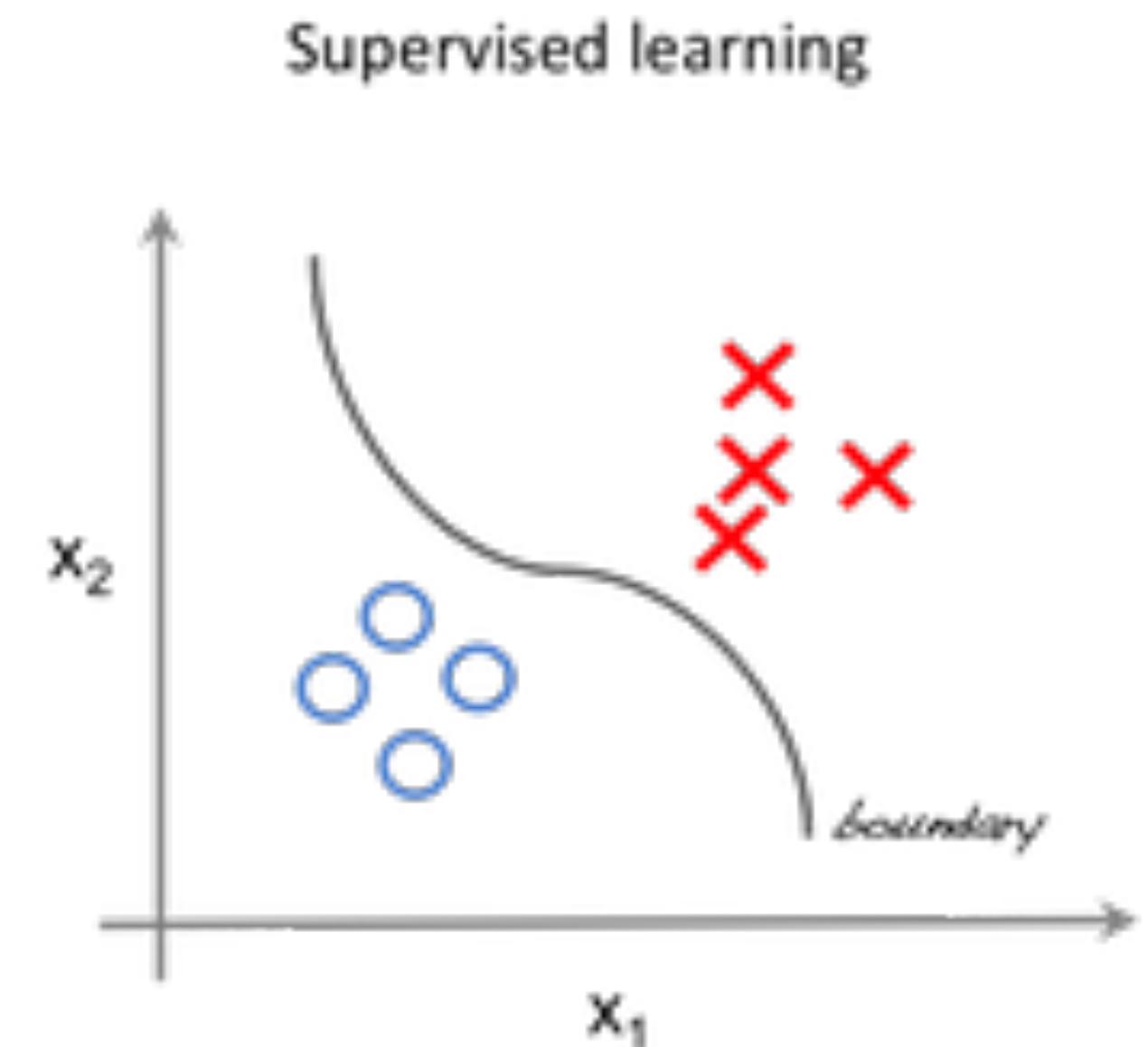
- **Classification:** given an image, identify the object represented
- **Regression:** given an image with an object, estimate its coordinates
- **Clustering:** given a dataset, group dataset entries which are alike
- **Anomaly/outlier detection:** given a reference dataset and a list of test entries, identify the entries not belonging to the dataset
- **Generation:** given a dataset, generate new entries that belong to the dataset



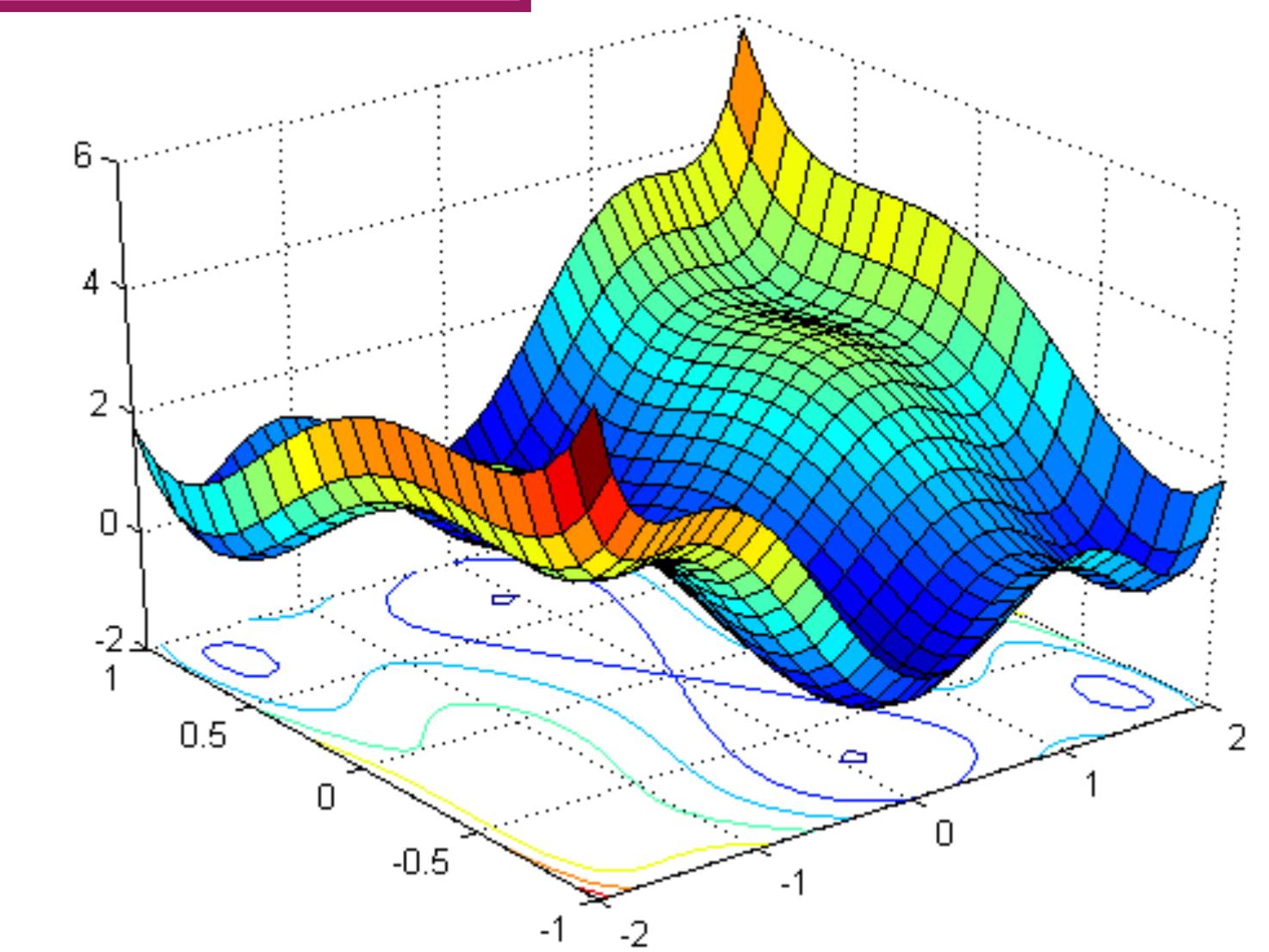
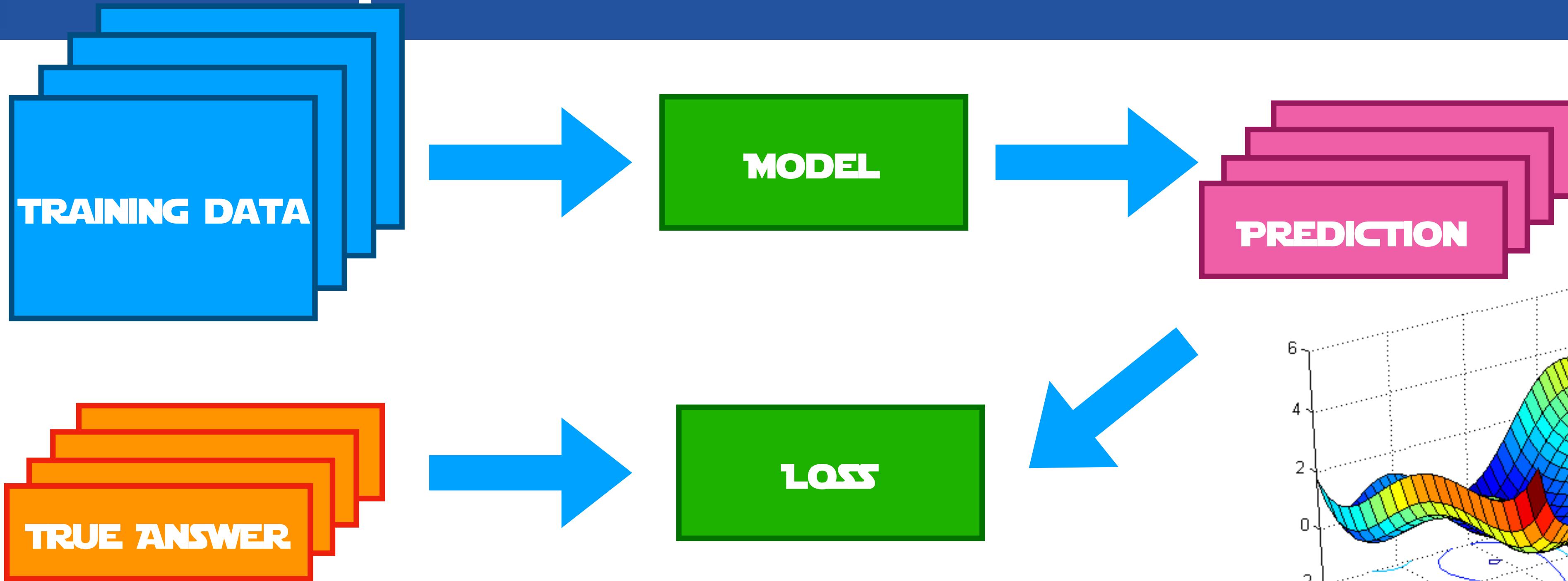
<https://thispersondoesnotexist.com/>

many kinds of learning

- **Supervised:** the dataset X comes with the right answer y (right class in a classification problem). The algorithm learns the function
- **Unsupervised:** the dataset X comes with no label. The algorithm learns structures in the data (e.g., alike events in a clustering algorithm)
- **Various intermediate flavors** (weakly supervised, semisupervised, etc.)
- **Adversarial** train a network against another network design for a competing task
- **Reinforcement Learning:** learn a series of actions and develop a decision-taking algorithm, based on some action/reward model (we will not cover this)

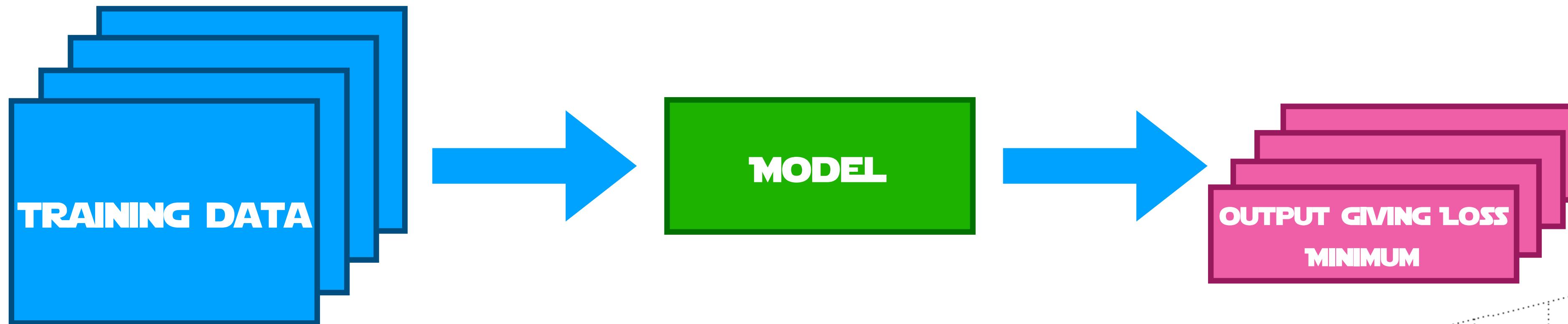


Supervised Learning

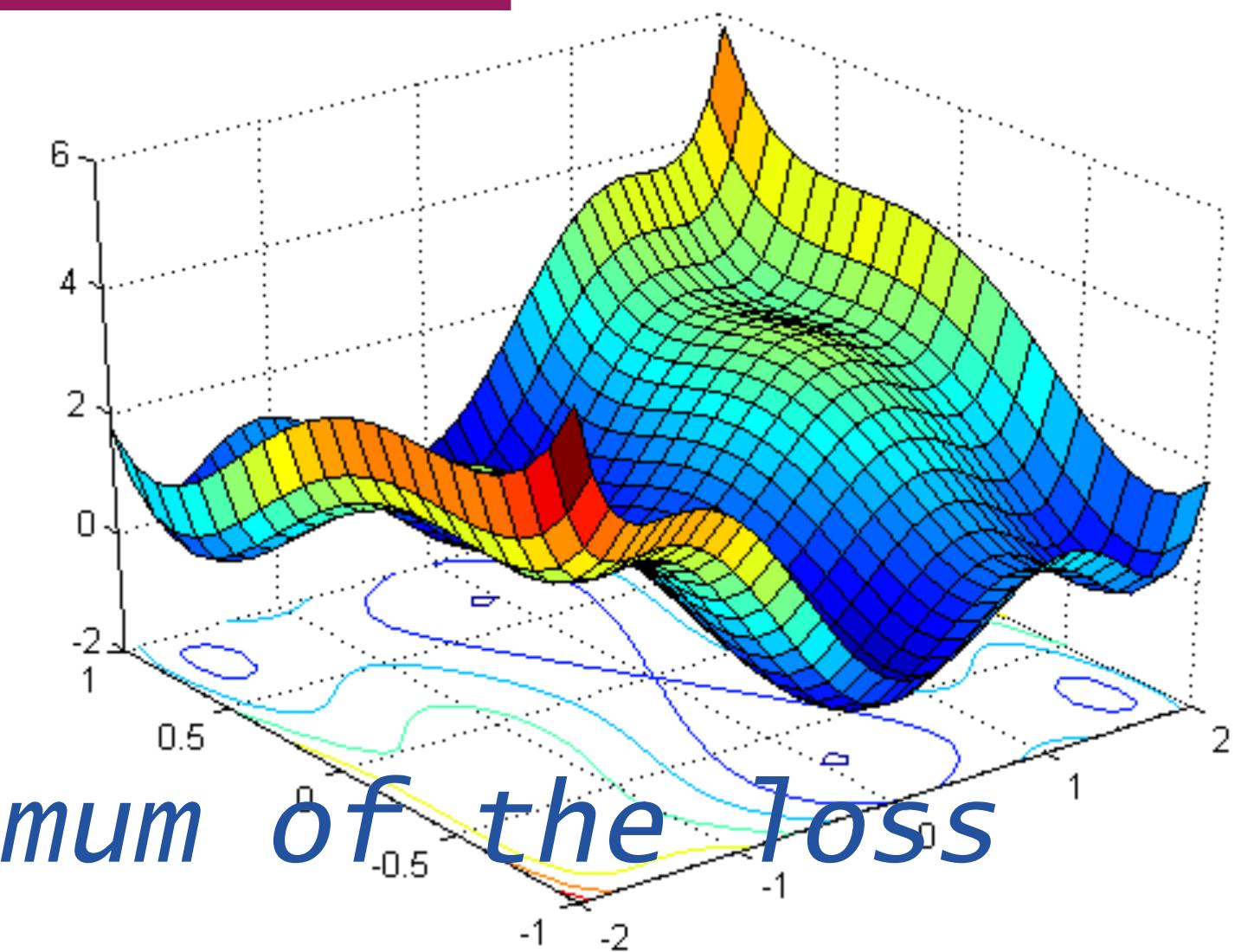


- A *training dataset* x
- A *target* y
- A *model* to go from x to y
- A *loss function* quantifying how wrong the model is
- A *minimisation algorithm* to find the model h that corresponds to the minimal loss

Unsupervised Learning

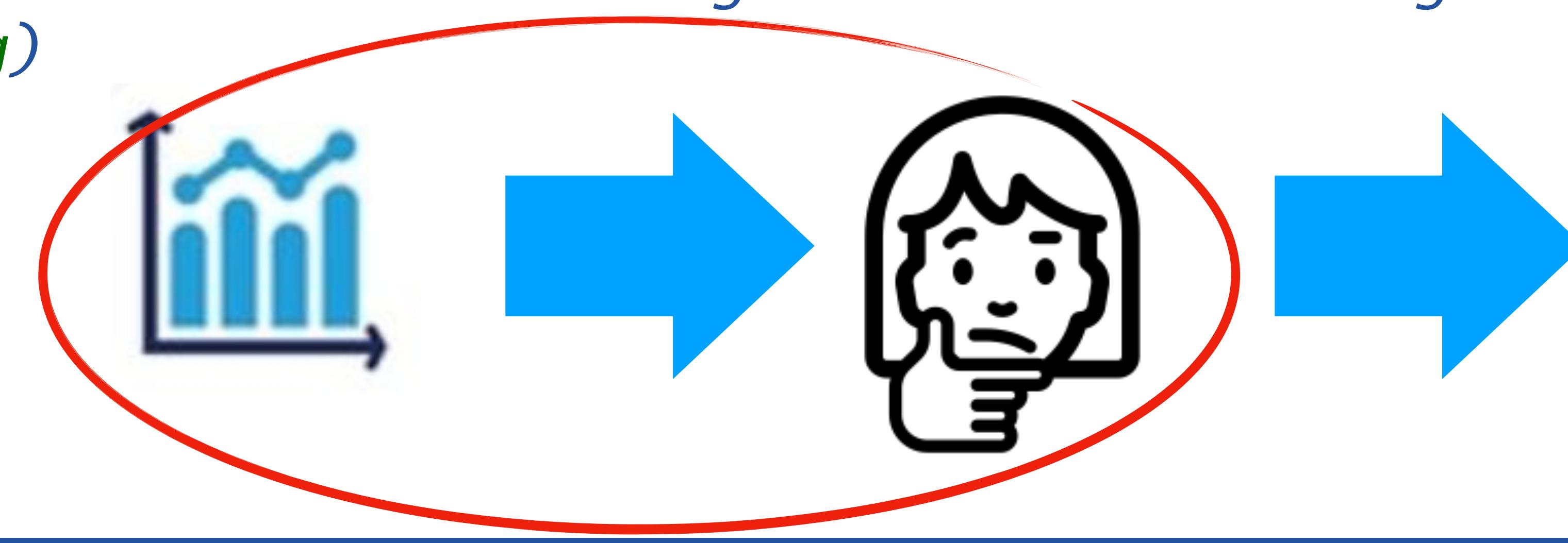


- A *training dataset* x
- No *target* y
- A *model* providing an *output* y at the *minimum of the loss*
- A *loss function* of x and y specifying the task
- e.g., clustering: group similar objects together



A pre-DL workflow

- A ML practitioner is given task and a dataset
- She would
- compute from the data a set of quantities that would be relevant to solve the problem (*feature engineering by domain knowledge*)
- pass these quantities to a ML algorithm for training (*task solving*)

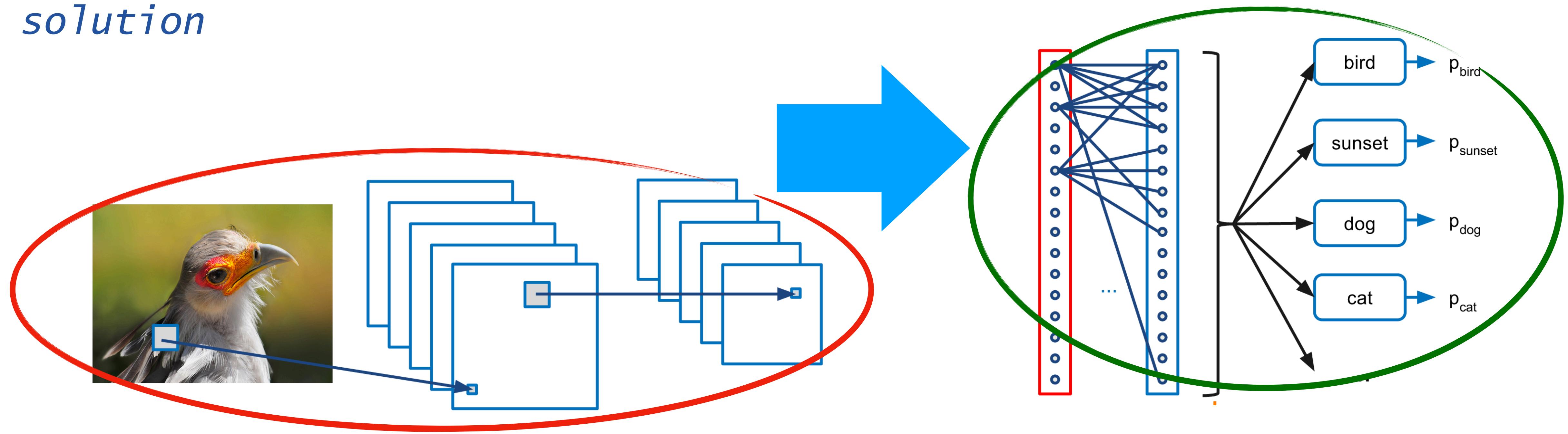


A DL workflow

- The first part of the network processes the raw data and learns the **feature engineering**

- The second part of the network performs the **task solving**

- The simultaneous training of the two steps guarantees optimal solution



Representation Learning

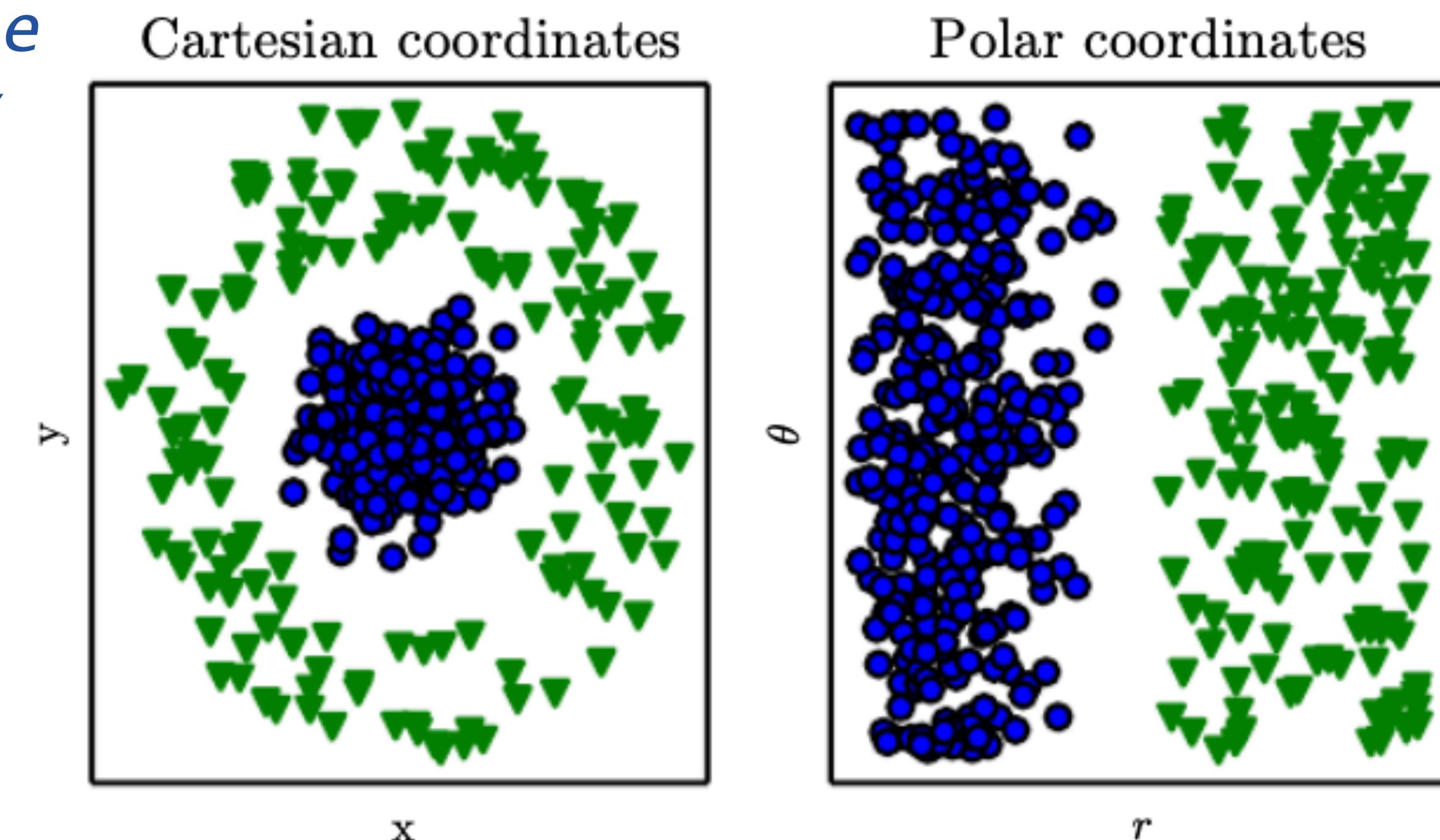
- *Performance may change depending on how inputs are presented*

- *Imagine the following problem: separate two populations with a linear boundary*

- *Depending on the coordinate system, this might be possible or not. One might have to look at data from the right angle*

- *When the right angle is unknown, one can learn it*

- *Use NNs to define functions f of the input x such that the problem is easier when starting from $f(x)$*



Feed-Forward NNs

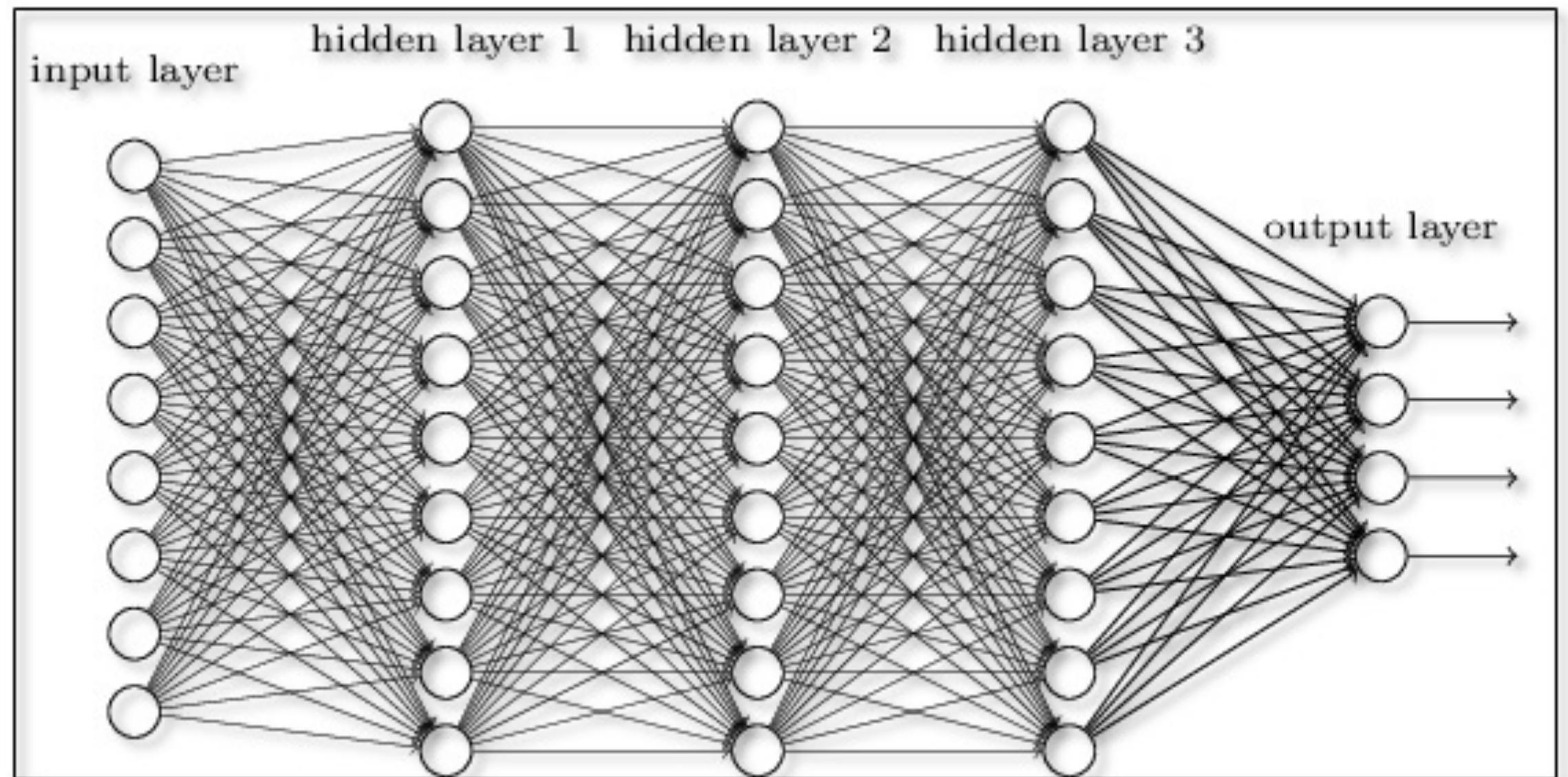
- Feed-forward neural networks have hierarchical structures:

- inputs enter from the left and flow to the right

- no closed loops or circularities

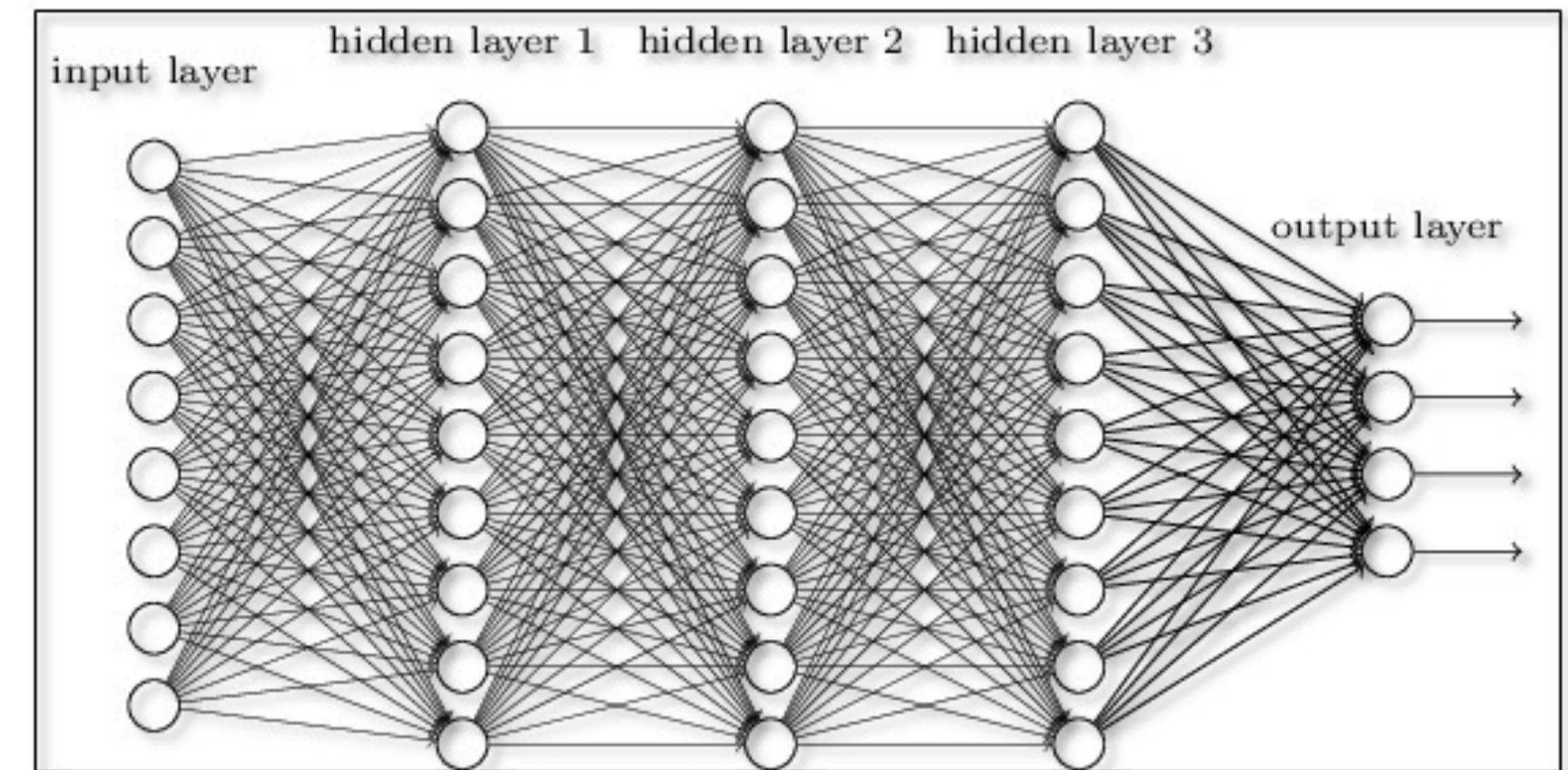
- Deep neural networks are feed-fwd NNs with more than one hidden layer

- Out of this “classic idea, new architectures emerge, optimised for computing vision, language processing, etc



Feed-Forward nnS

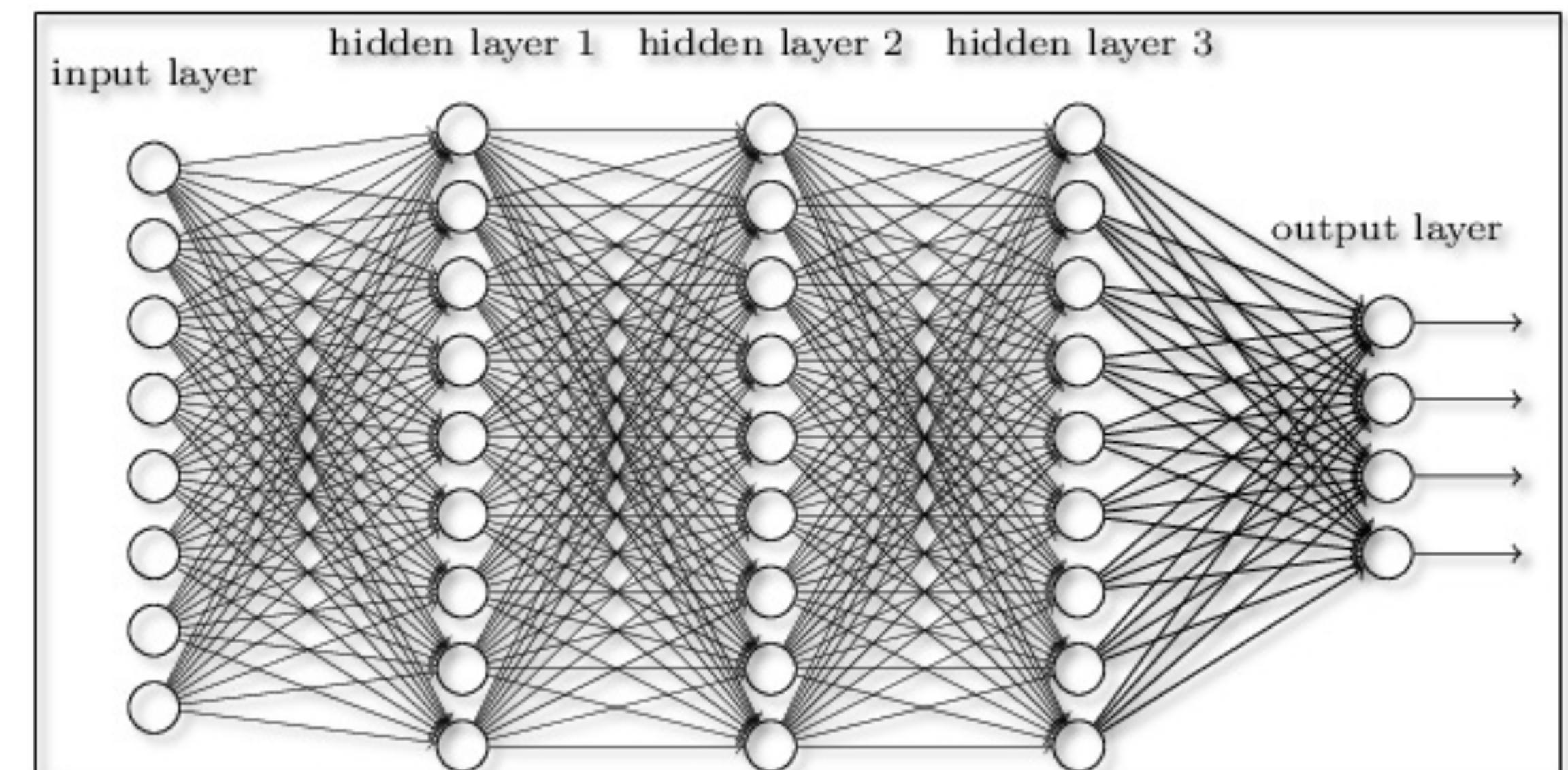
- *Each input is multiplied by a weight*
- *The weighted values are summed*
- *A bias is added*
- *The result is passed to an activation function*



$$w_{ij}x_j$$

Feed-Forward nnS

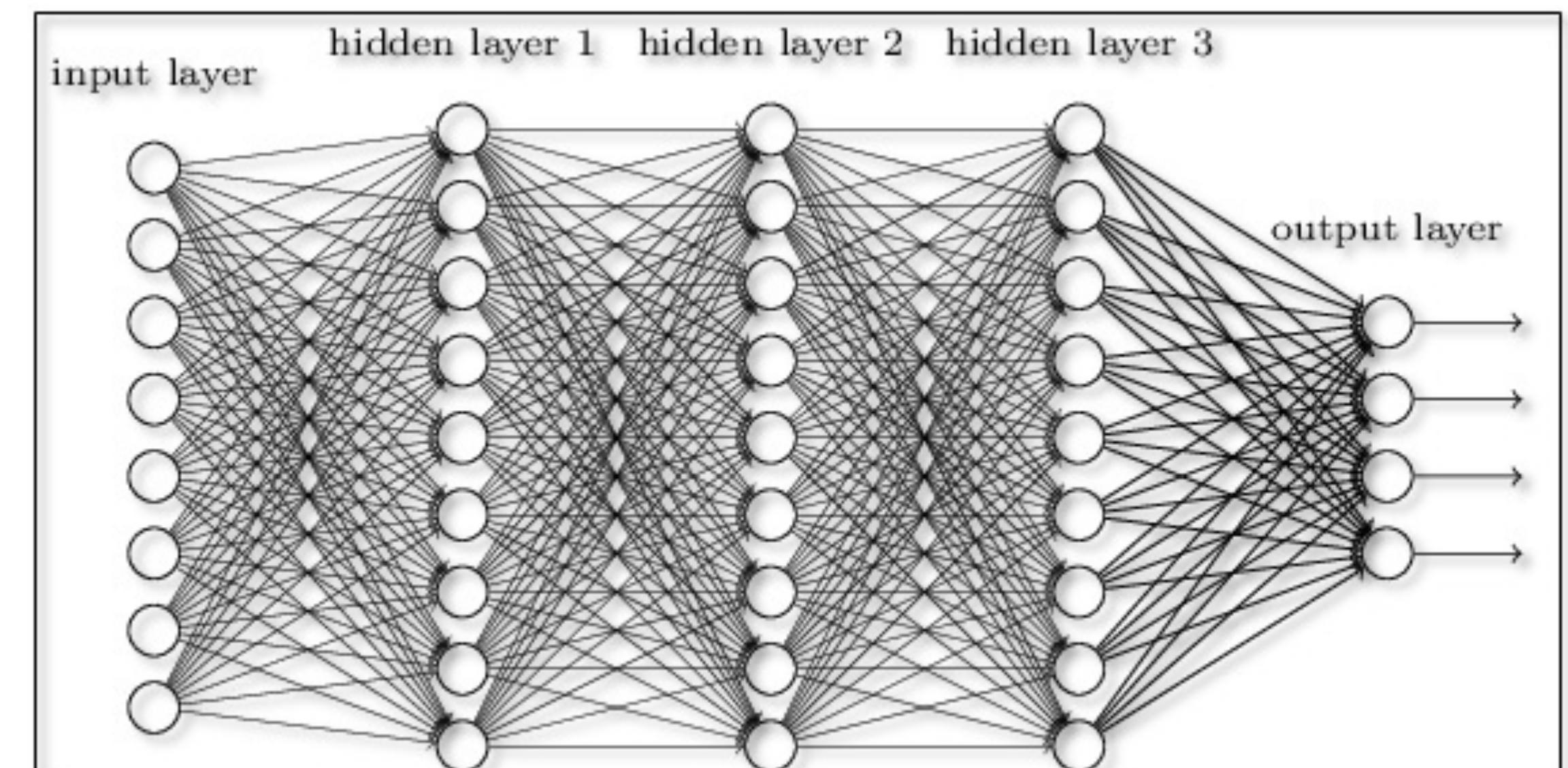
- *Each input is multiplied by a weight*
- ***The weighted values are summed***
- *A bias is added*
- *The result is passed to an activation function*



$$\sum_j w_{ij} x_j$$

Feed-Forward nnS

- *Each input is multiplied by a weight*
- *The weighted values are summed*
- **A bias is added**
- *The result is passed to an activation function*



$$\sum_j w_{ij}x_j + b_i$$

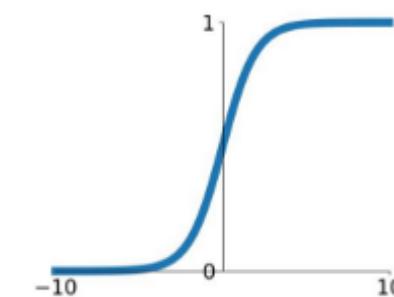
Feed-Forward nnS

- Each input is multiplied by a weight
- The weighted values are summed
- A bias is added
- The result is passed to an activation function

Activation Functions

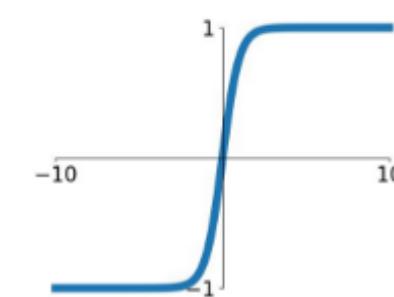
Sigmoid

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



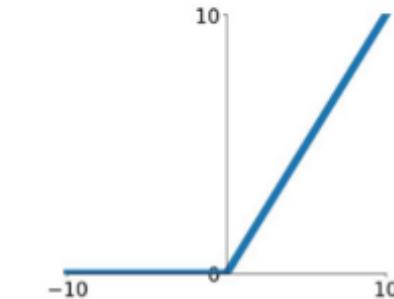
tanh

$$\tanh(x)$$



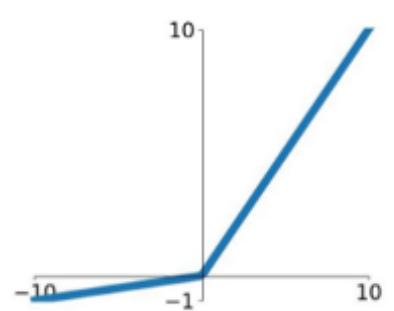
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

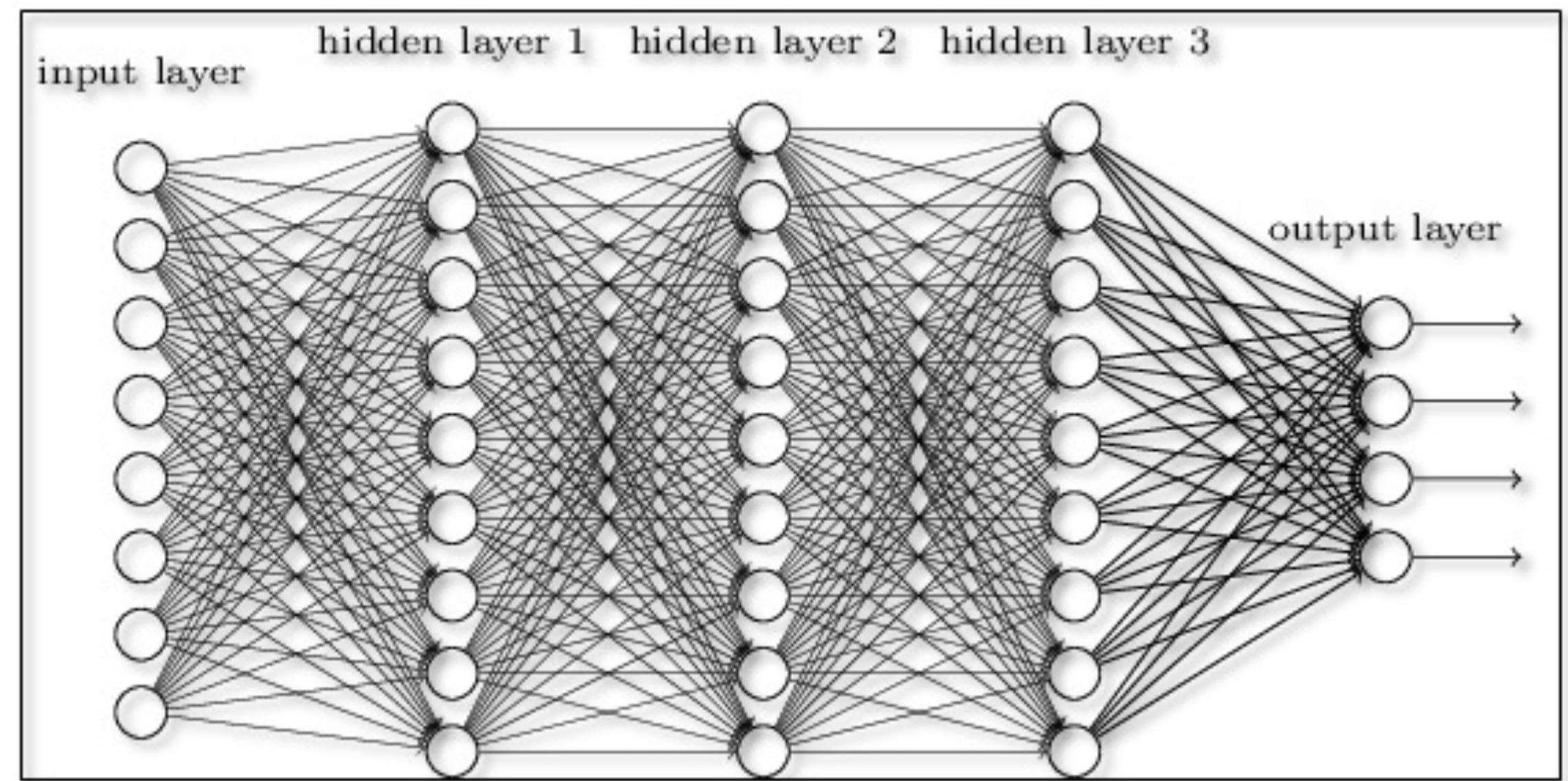
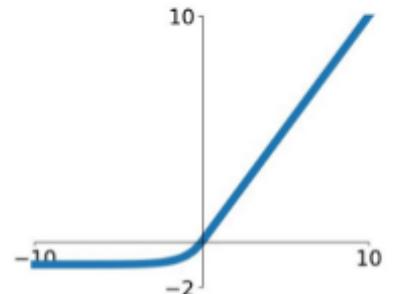


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

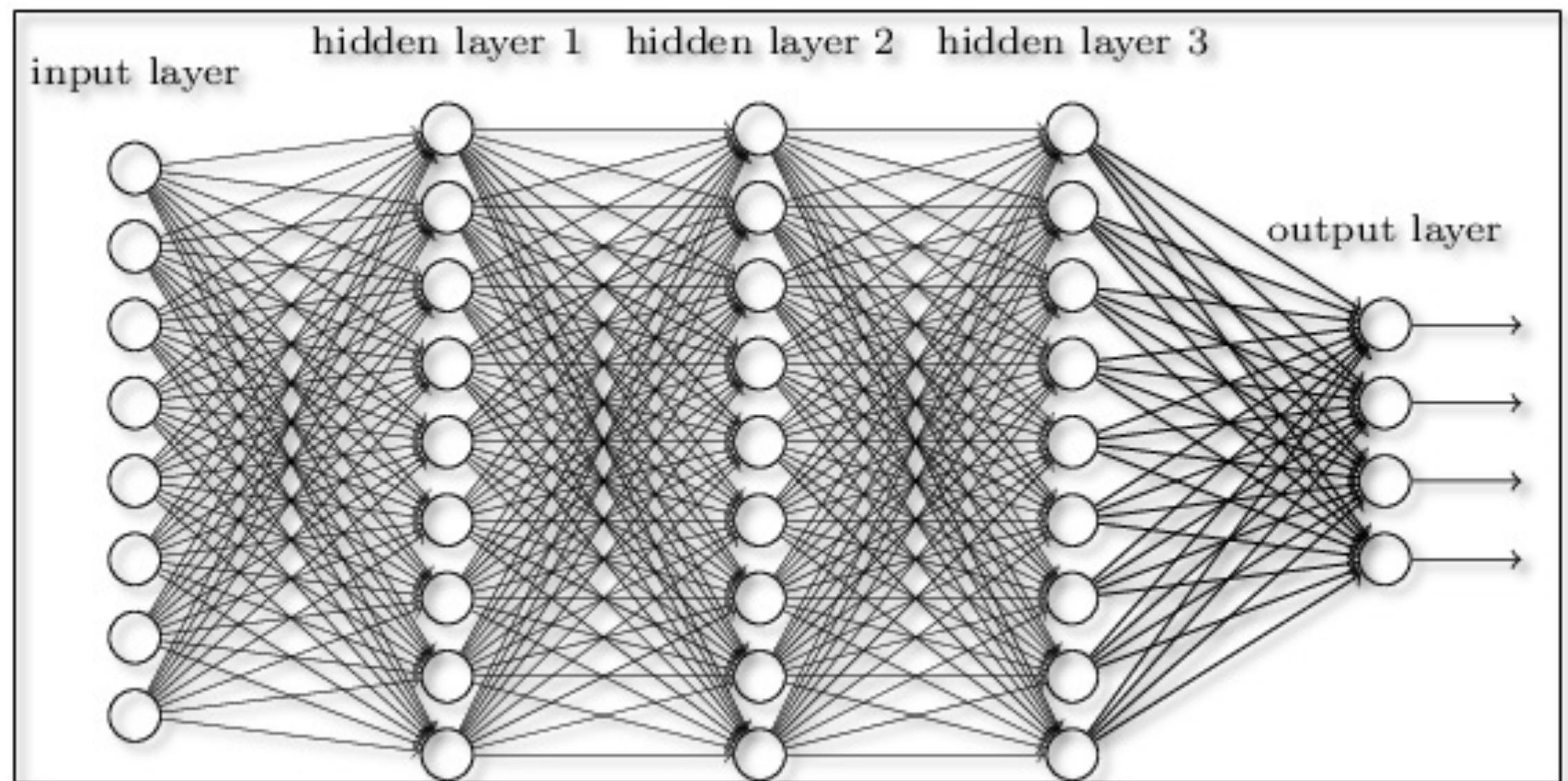
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



$$\hat{y}_i = f(\sum_j w_{ij} x_j + b_i)$$

Feed-Forward nnS

- In a feed-forward chain, each node processes what comes from the previous layer
- The final result (depending on the network geometry) is K outputs, given N inputs

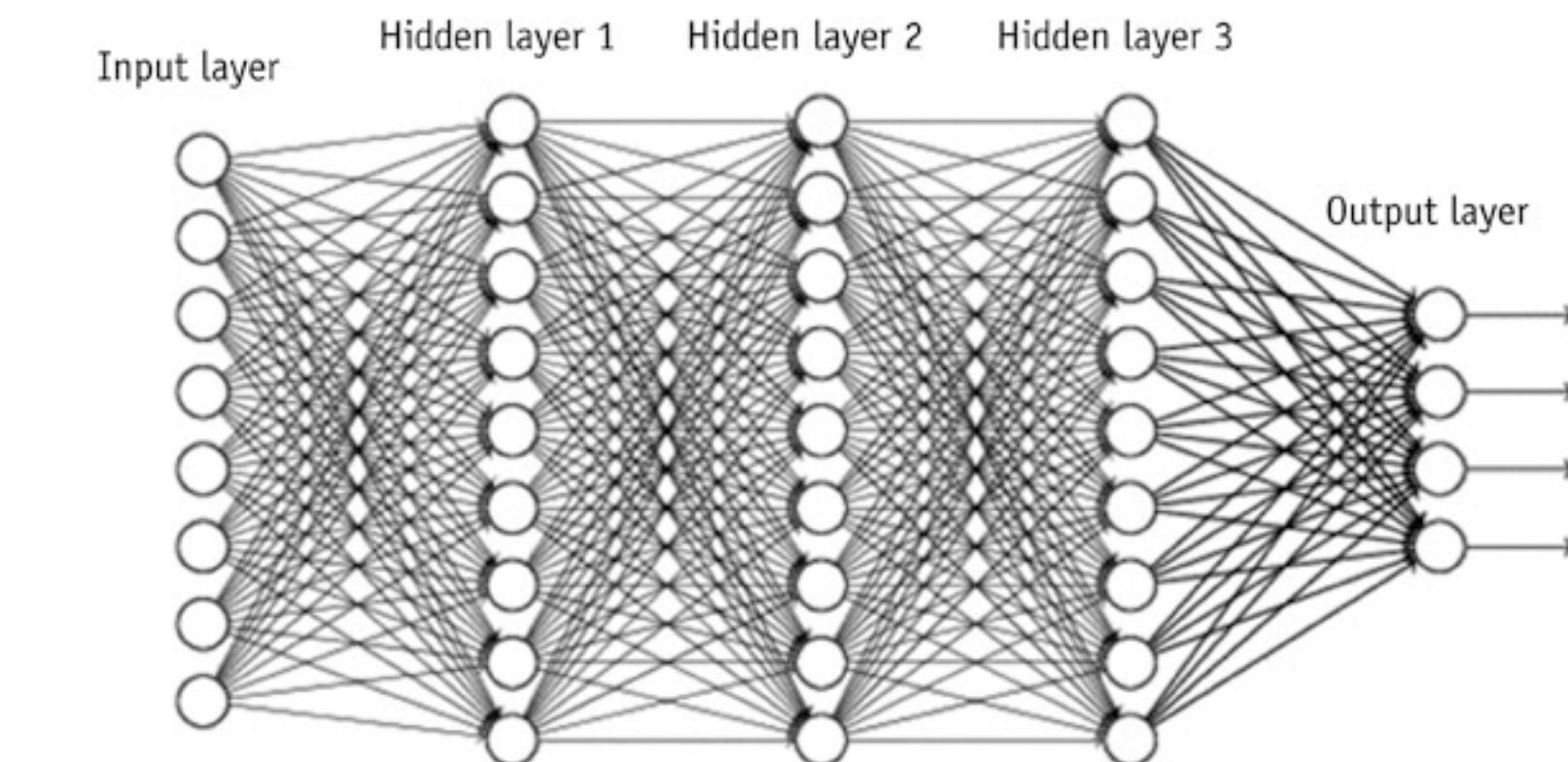


$$\hat{y}_3 = f^{(3)} \left(\sum_l w_{jl}^{(3)} f^{(2)} \left(\sum_k w_{lk}^{(2)} f^{(1)} \left(\sum_i w_{ki}^{(1)} x_i + b_k^{(1)} \right) + b_l^{(2)} \right) + b_j^{(3)} \right)$$

- One can show that such a mechanism allows to learn generic $\mathbb{R}^N \rightarrow \mathbb{R}^K$ smooth functions

A technology-driven revolution

- Deep neural networks have >1 inner layer, hence more complexity thanks to more parameters
- Thanks to GPUs, it is now possible to train them efficiently, which boosted the revival of neural networks in the years 2000
- In addition, new architectures emerged, which better exploit the new computing power

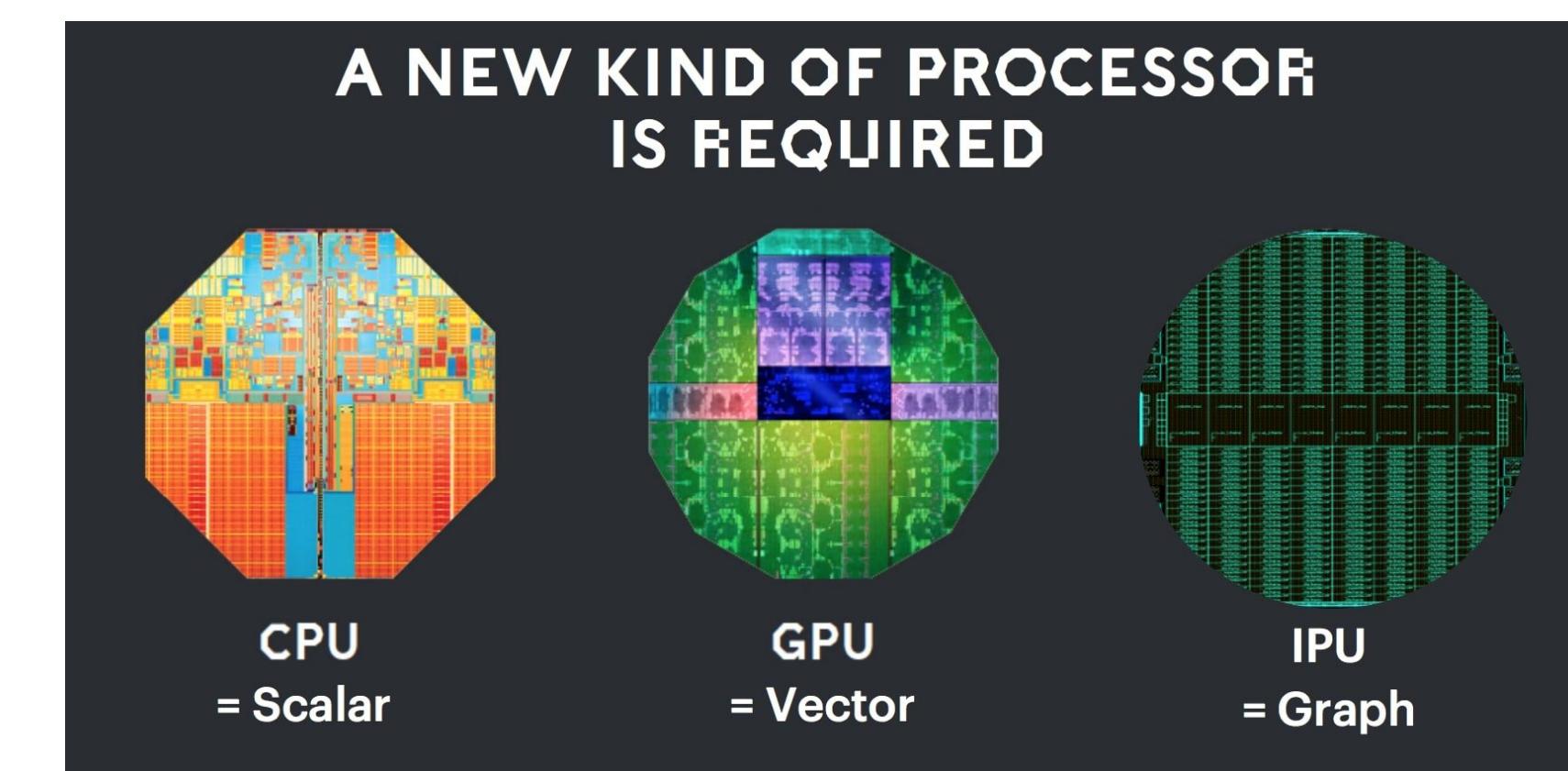


Large-scale Deep Unsupervised Learning using Graphics Processors

Rajat Raina
Anand Madhavan
Andrew Y. Ng

Computer Science Department, Stanford University, Stanford CA 94305 USA

RAJATR@CS.STANFORD.EDU
MANAND@STANFORD.EDU
ANG@CS.STANFORD.EDU



Training in practice

- The network score (i.e., the output) is a function of the network parameters (\vec{w}, \vec{b}) and the data

$$\hat{y}_3 = f^{(3)}(\sum_l w_{jl}^{(3)} f^{(2)}(\sum_k w_{lk}^{(2)} f^{(1)}(\sum_i w_{ki}^{(1)} x_i + b_k^{(1)}) + b_l^{(2)}) + b_j^{(3)})$$

- The loss is a function of the score and the data (the input x and, for supervised learning, the truth y)

$$\mathcal{L} = \sum_i |y^i - \hat{y}^i(x, w, b)|^2$$

- One can then think that, for a given dataset (and its truth), the loss is a function of the weights

$$\mathcal{L} = f(w, b | x, y)$$

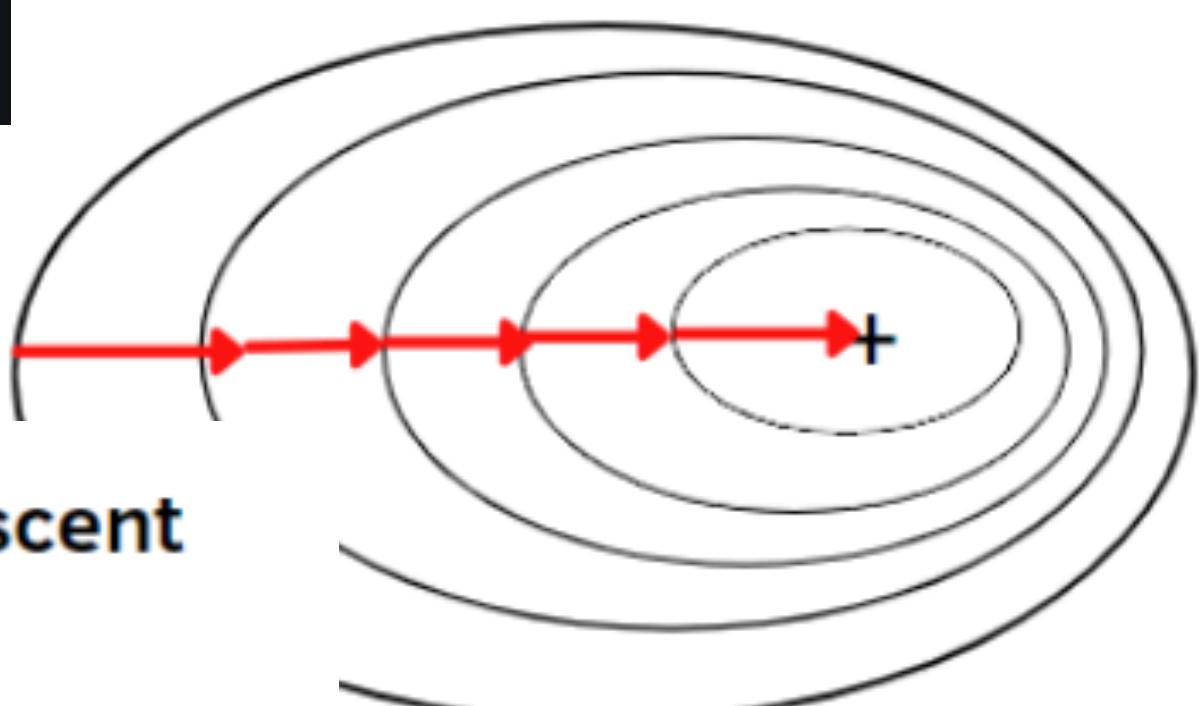
Training in practice

- The training is then a minimisation problem: one starts from a random point in the (w, b) space and tries to walk down towards the minimum
- Gradient descent:** one computes the maximum gradient and takes a step in the opposite direction
- Various kinds of gradient descents:
- batch gradient descent:** compute the gradient in parallel for small chunks of data and take the average. Update the model once all data are processed
- stochastic gradient descent:** update the model after each example. More noisy, less prone to get stuck to a local minimum, but more computationally expensive
- mini-batch gradient descent:** like sgd, but update the model only after a batch of examples. Compromise between the previous two

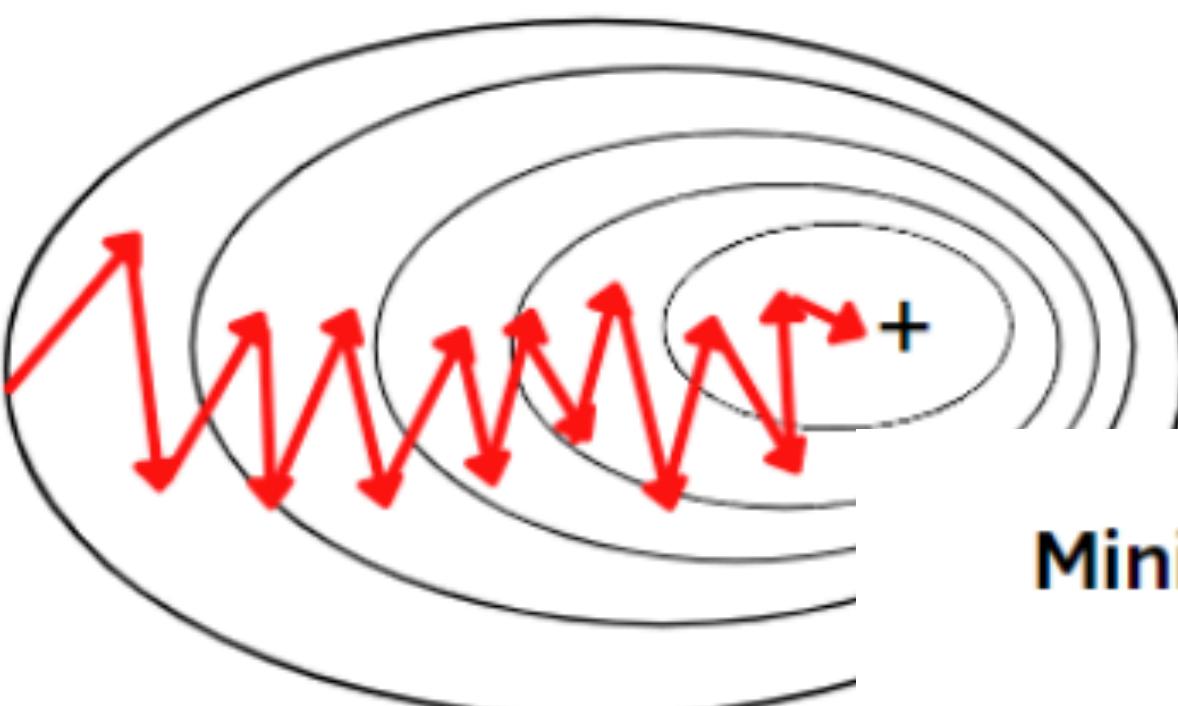
$$w := w - \eta \nabla Q(w)$$

Learning rate

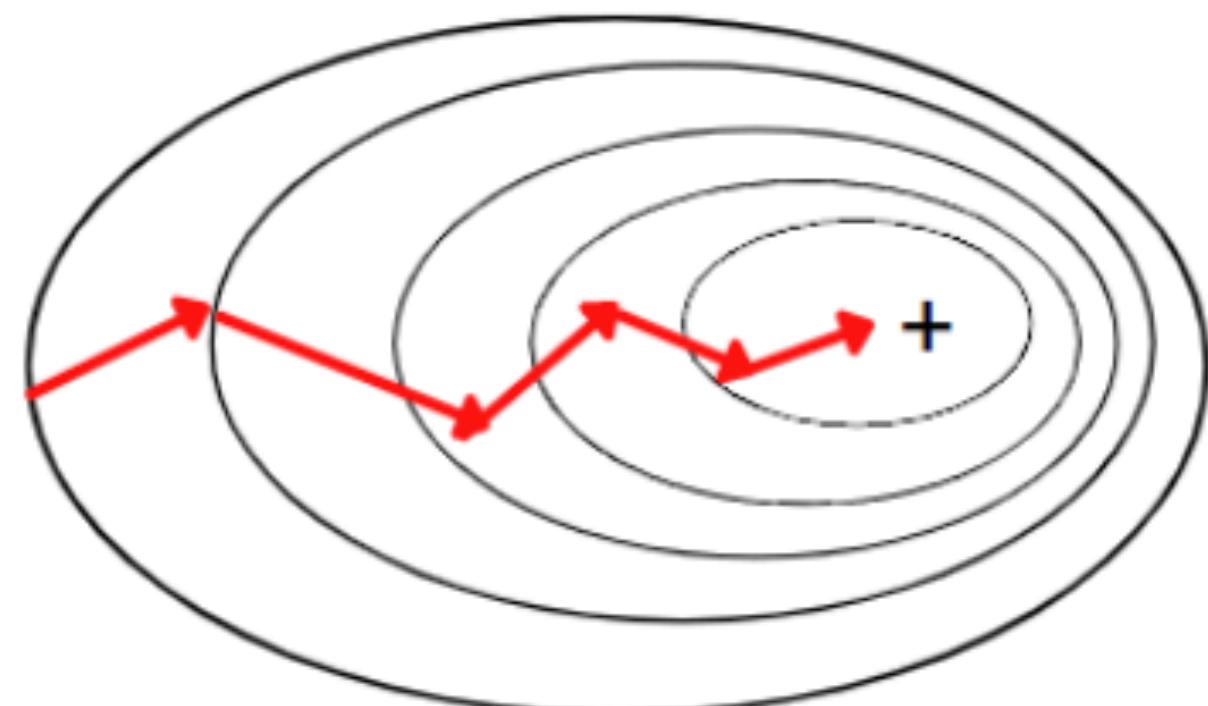
Batch Gradient Descent



Stochastic Gradient Descent

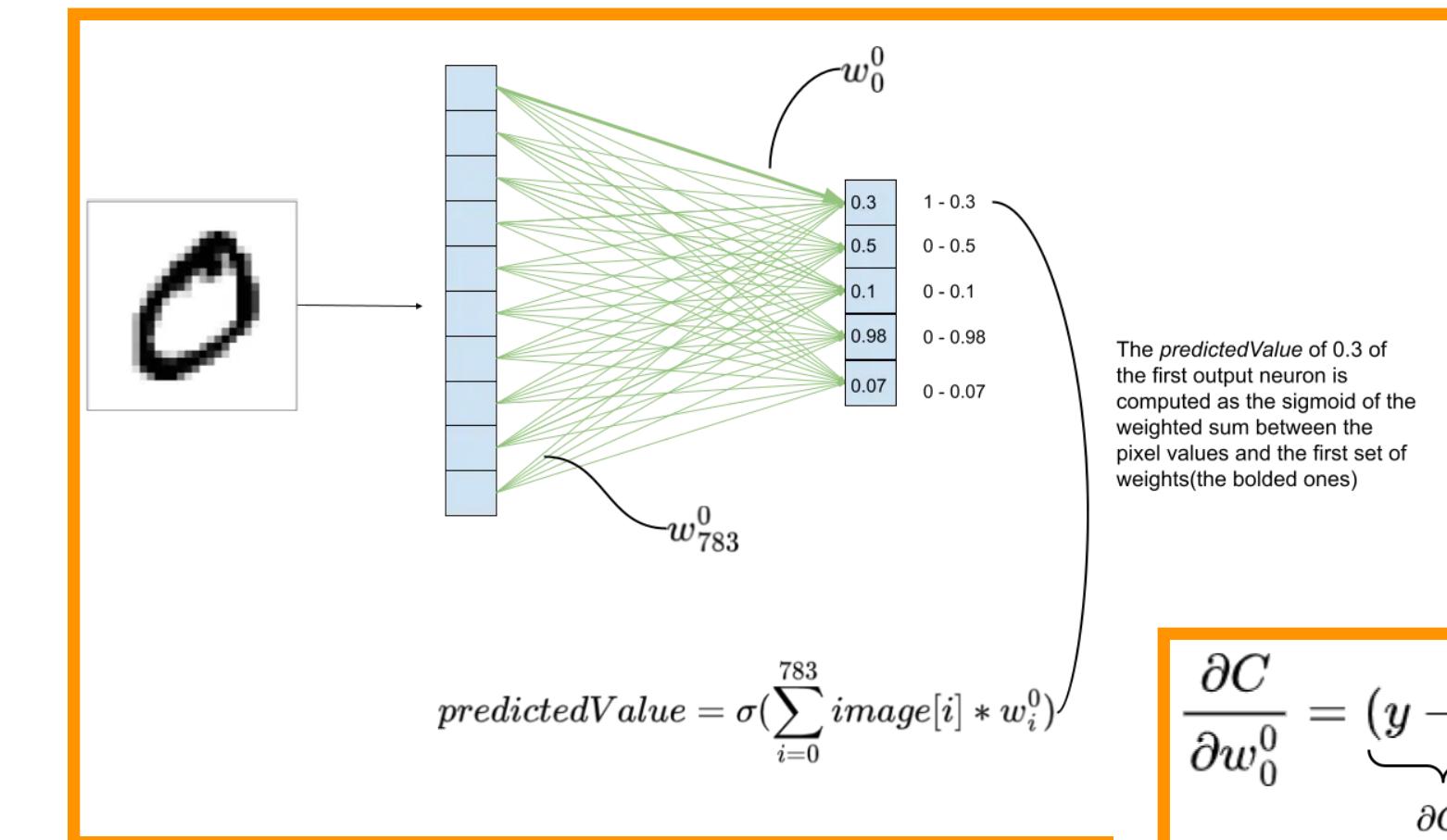


Mini-Batch Gradient Descent

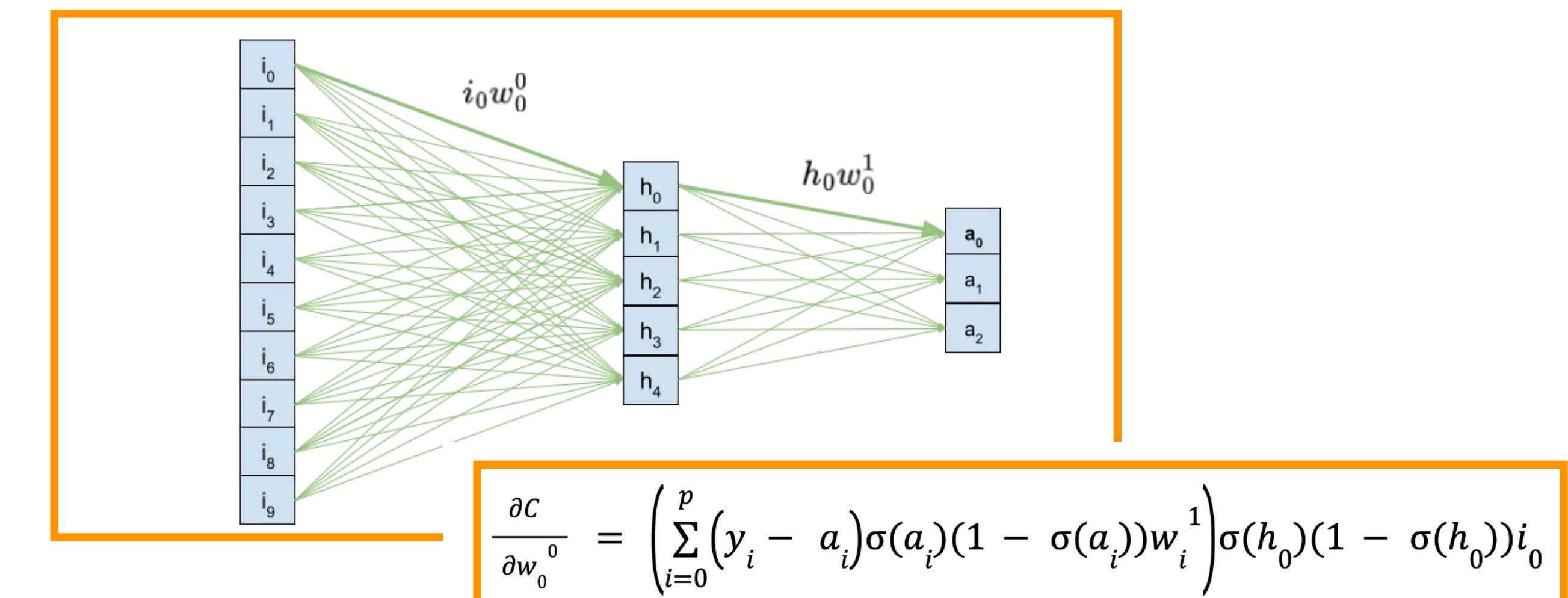


Backpropagation

- Backpropagation allows to speed up sgd exploiting the gradient chain rule
- one goes back from the activation function to the argument, to the input, using derivative chain rule
- Adding layers just makes the chain longer, but the concept is the same
- You can find [here](#) a useful walkthrough, that will become more clear in a few lectures (when you will be familiar with all the ingredients)
- with many useful tips, e.g., on the choice of the activation function



$$\frac{\partial C}{\partial w_0^0} = \underbrace{(y - a)}_{\frac{\partial C}{\partial a}} \underbrace{\sigma(z)}_{\frac{\partial a}{\partial z}} \underbrace{(1 - \sigma(z))}_{\frac{\partial z}{\partial w_0^0}} \text{image}[0]$$



Summary

- *Deep Learning is the latest evolution of Machine Learning*
- *Technological progress with gradient computation on distributed computing architectures allowed to use higher complexity architectures*
- *Various kinds of problems, architectures, and learnings*
- *Dense Neural Networks are the simplest architecture*
- *More than brute force: one needs insight for best architecture choice, best choice of hyperparameters, and best data representations to facilitate the feature extraction*