Introduction to Neural Nets for Machine Learning

Sylvain Caillou (L2IT)

Workshop on LISA Data Analysis - November 21-25, Toulouse

LISA data analysis: from classical methods to machine learning

What are we going to talk about today ?

General Machine Learning concepts

Introduction to Neural Networks and Deep Learning

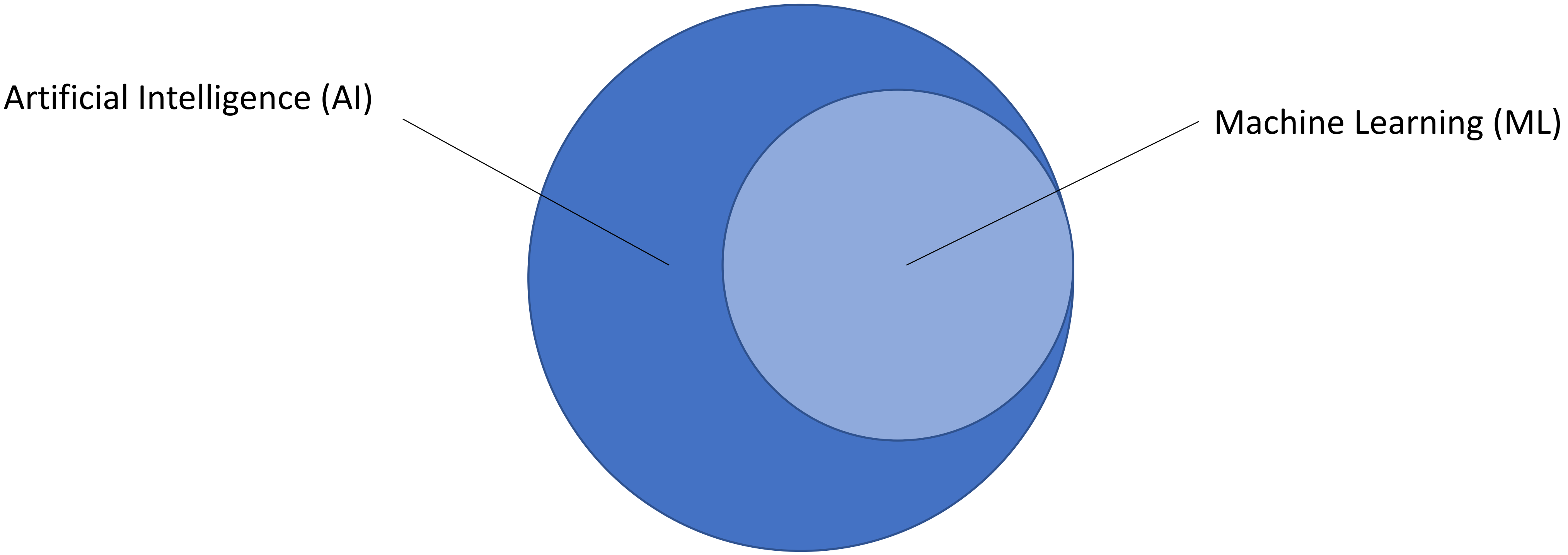
Deep Learning in practice: introduction to pytorch

A few recent (sucess?) stories of Deep Learning

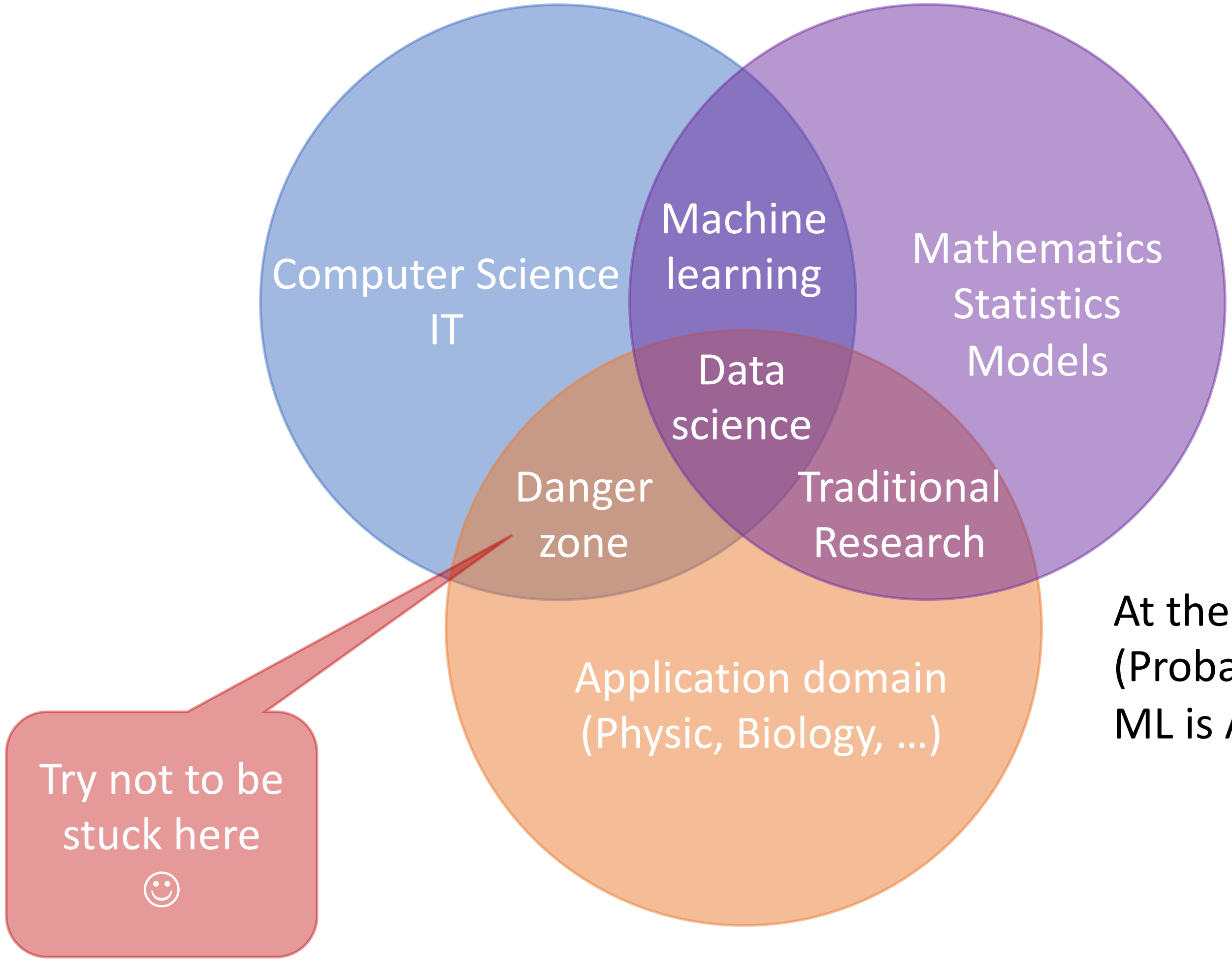
General Machine Learning concepts

What is Machine Learning ?

Machine Learning is a part of **Artificial Intelligence**. It is a set of algorithms based on models which can be trained to learn from **statistical patterns in data**, and improve **automatically** their performance. Once trained, models can **generalize** and make **prediction** taking unseen data as input.

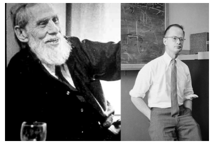


What is Machine Learning ?



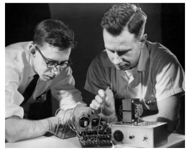
At the intersection between Mathematics (Probability and Statistics) and Computer Science.
ML is Applied in lot of application domain.

A quick history of AI

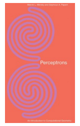


First concept of artificial Neural Networks
McCulloch, Pitts
1943

Artificial Intelligence as research field
Darmouth workshop, McCarthy
1956



Perceptron
Rosenblatt
1957



Perceptrons
Minsky, Papert
1969

“In from 3 to 8 years, we will have a machine with the general intelligence of an average human being.” Marvin Minsky, 1970

Backpropagation
Algorithm
Rumelhart
1986

Rise of Machine Learning without NNs (SVMs, …)

PyTorch
2016

TensorFlow
2015

AlphaGo
Defeats lee
seedoï
2016

AlphaFold2
Geometric ML
GNNs
2021

Diffusion models (2022)

1940 1950 1960 1970 1980 1990 2000 2010 2020

During the 1960s, DARPA provided millions of dollars for AI research

MIT (Minsky, Papert)
CM (Simon, Newel)
Stanford (McCarthy)

First AI Winter
Mansfield amendment (1969)
Lighthill report (1973)

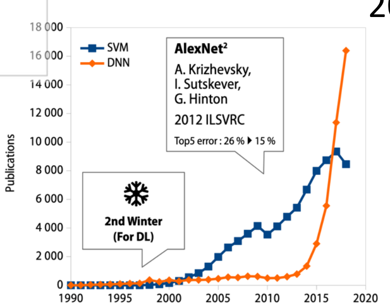
CNN
LeCun
1989

2nd Winter
(For DL)

AlexNet
(CNN)
2012

BERT
(Transformers)
2018

Publications SVM vs DNN[1]



Workshop on LISA Data Analysis - November 21-25, Toulouse

Why an acceleration of AI now ?

Algorithms

Backpropagation
CNNs,
RNNs, Transformers, VAEs,
GNNs, Diffusion models
…

Big Data

Internet
Centralization
of information
Database

Software

NNs Frameworks
Linear Algebra on accelerators
NNs dedicated

Hardware accelerators

GPUs, FPGAs, ASICs, …

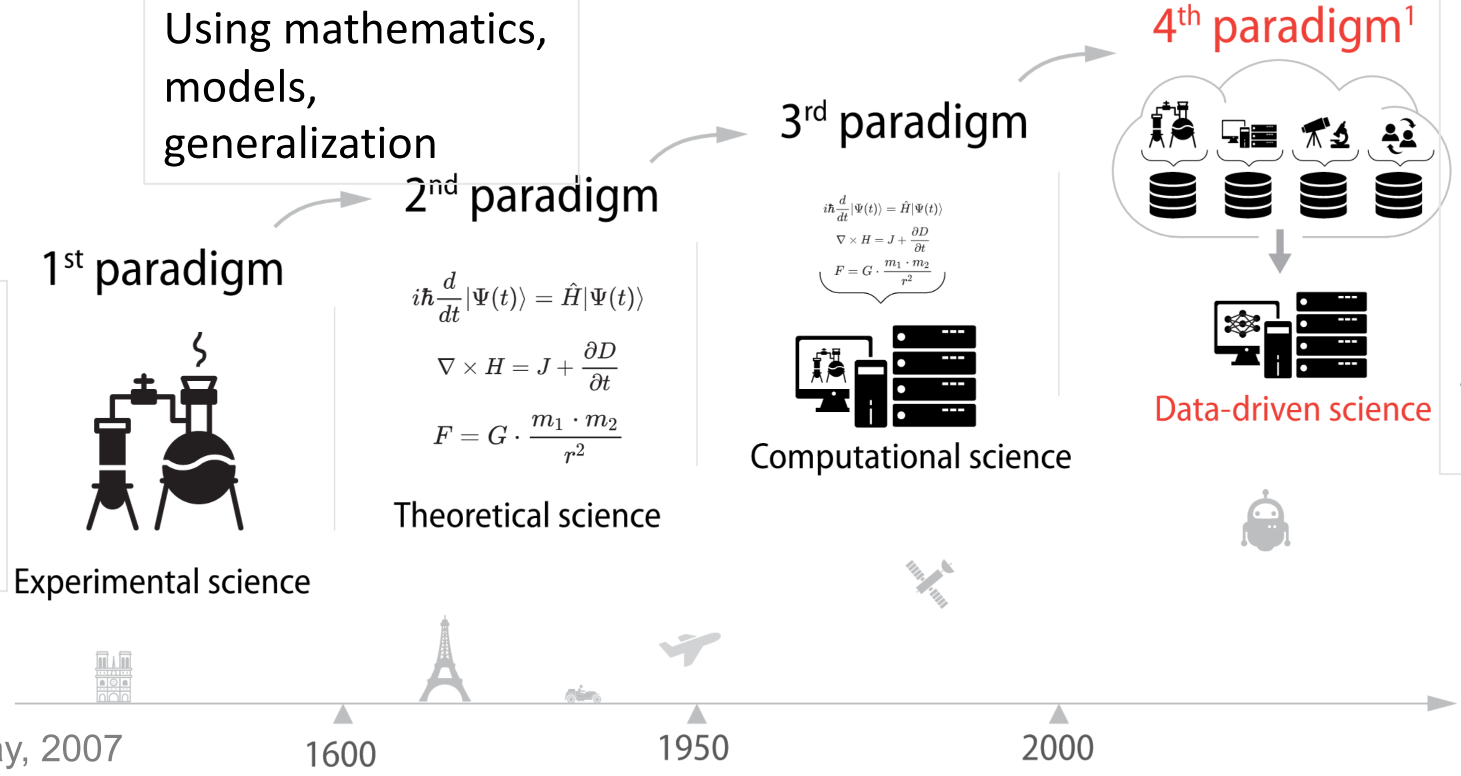
Data-driven science: A new scientific paradigm ?

Experimental science
Empirical Evidence
Observations
Describing natural phenomena

Theorical science
Scientific Theory
Using mathematics, models, generalization

Computational science
Simulating complex phenomena

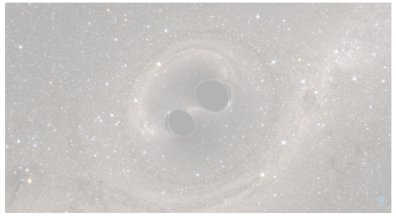
Data-driven science
Data-intensive Scientific Discovery
Unify theory, experiment and simulation
Data exploration
Statistics & Machine Learning

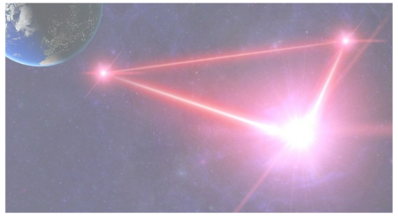


[1]Jim Gray, 2007

[From « Introduction to Deep Learning » CNRS FIDLE Formation]

Don’t use ML as black box
Don’t use data as abstract input

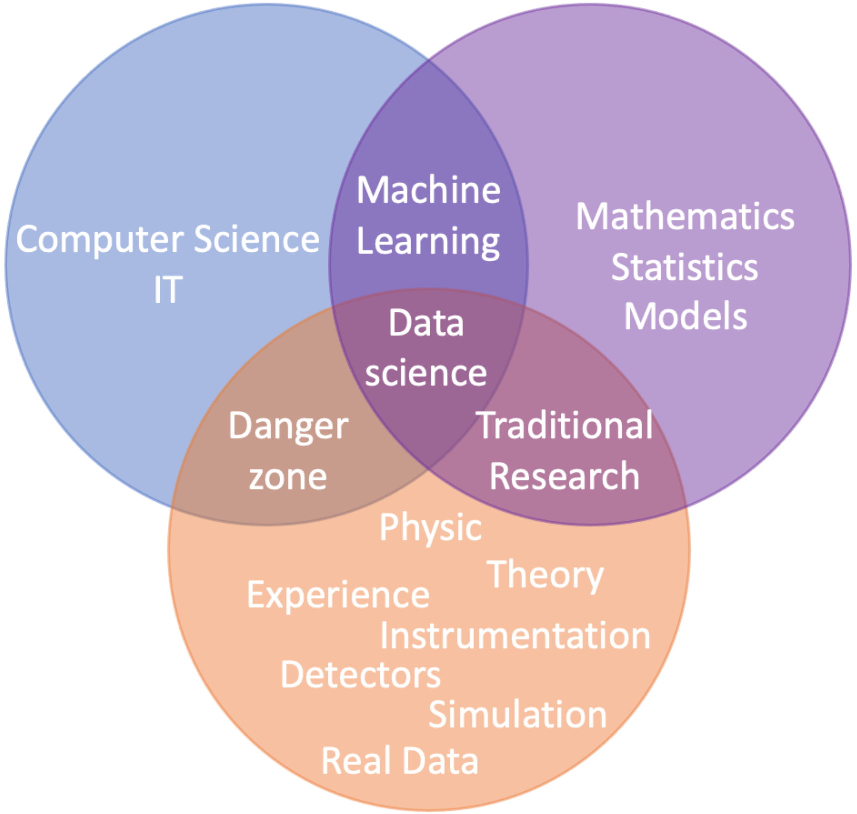




NATURE ⇒ EXPERIMENT ⇒ DATA

EXPERIMENT:
Instrumentation / detectors
Signal and noise
Simulated / real data

DATA:
Data don’t come from nowhere…



Data is link to experimental conditions and application domains
$\Rightarrow$ It is need to understand the data and where they came from
$\Rightarrow$ It is a prerequisite for applying correctly ML models

Motivation to use ML

$X =$ DATA Let's say you want to compute a variable y from the experimental data X

The *relation* between y and X can be seen as a mathematical function f : $y = f(X)$

It's *probably* a good idea to use ML if:

- We have the analytic form of f but it's highly time consuming to compute
- We don't have the analytic form

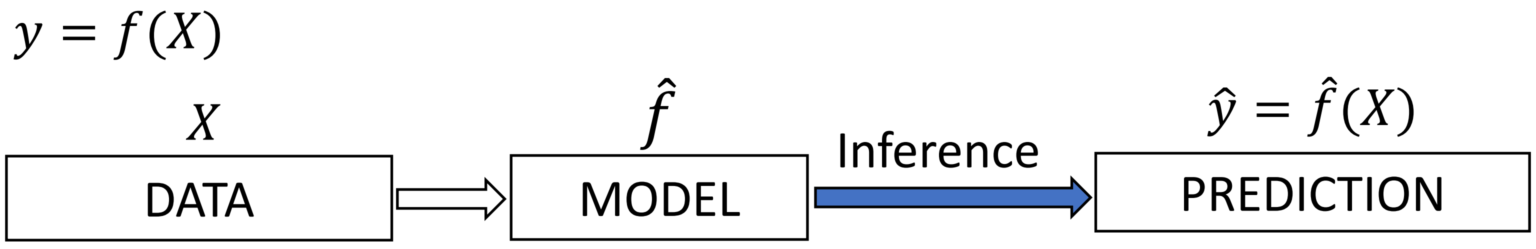
Learn from data and predict

Train a ML model to learn from **statistical patterns in data**
The model will *learn* an approximation of the function f
The model will *predict* a value $\hat{y}$ which have to be compare to y

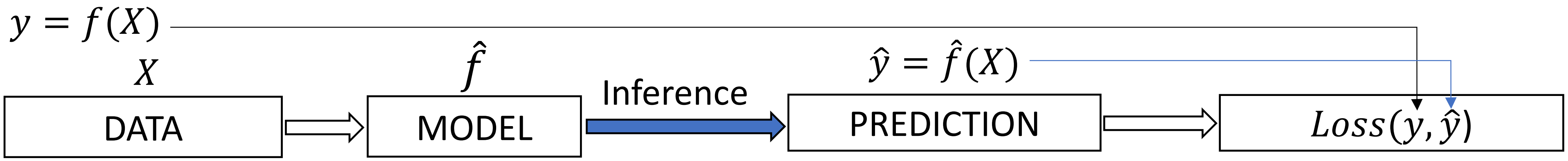
$$y = f(X)$$

$$\begin{array}{ccccc} X & & \hat{f} & & \hat{y} = \hat{f}(X) \\ \boxed{\text{DATA}} & \Rightarrow & \boxed{\text{MODEL}} & \Rightarrow & \boxed{\text{PREDICTION}} \end{array}$$

Prediction = model inference

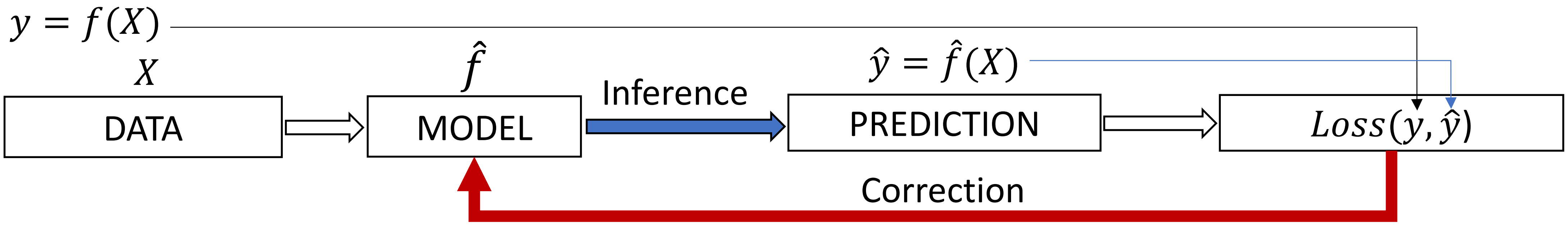


Estimation of the error between prediction and truth



We define a Loss function to evaluate the difference between prediction and truth

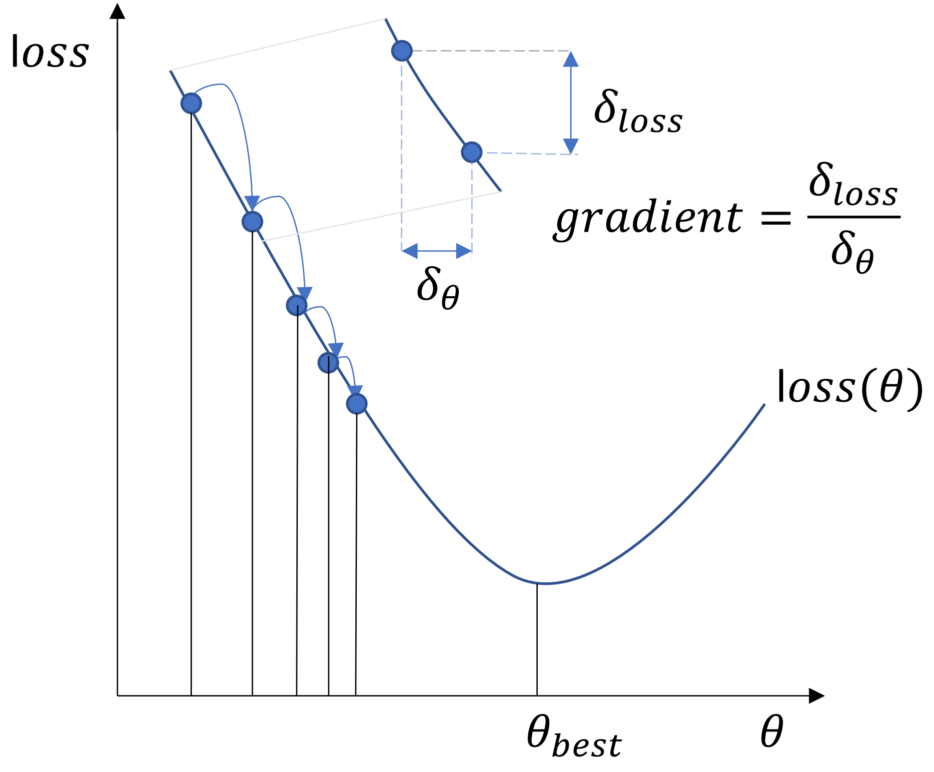
Loss optimization



All ML is here: Correction of the model parameters to minimize the Loss

Gradient descent

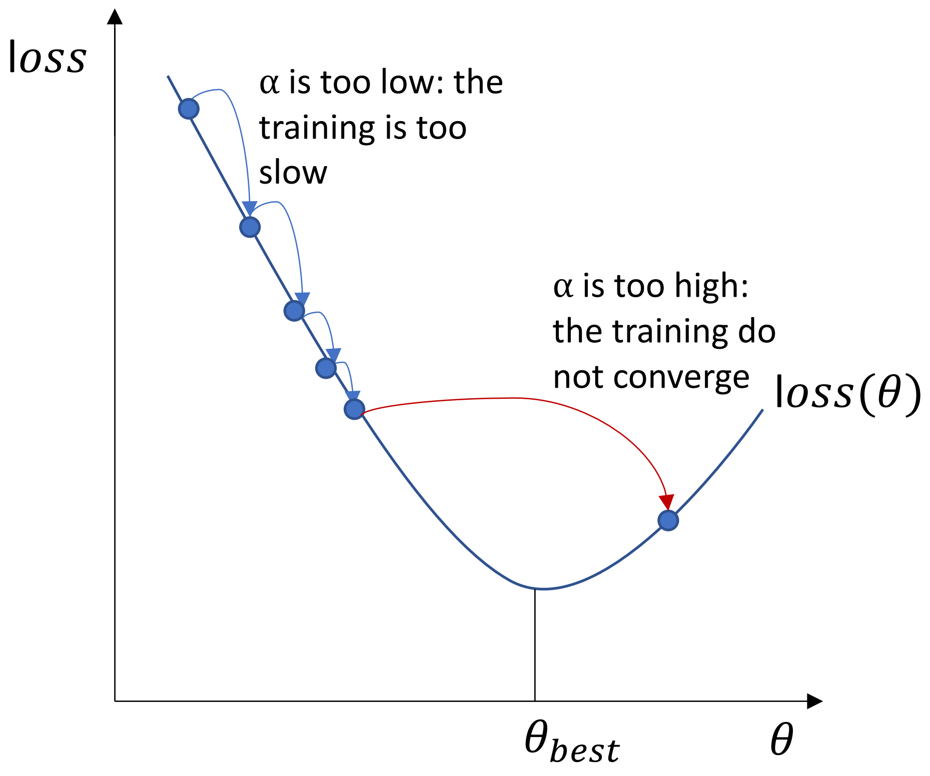
General optimization technique. Can not be applied on all ML algorithms but it can on Neural Nets (hopefully)…



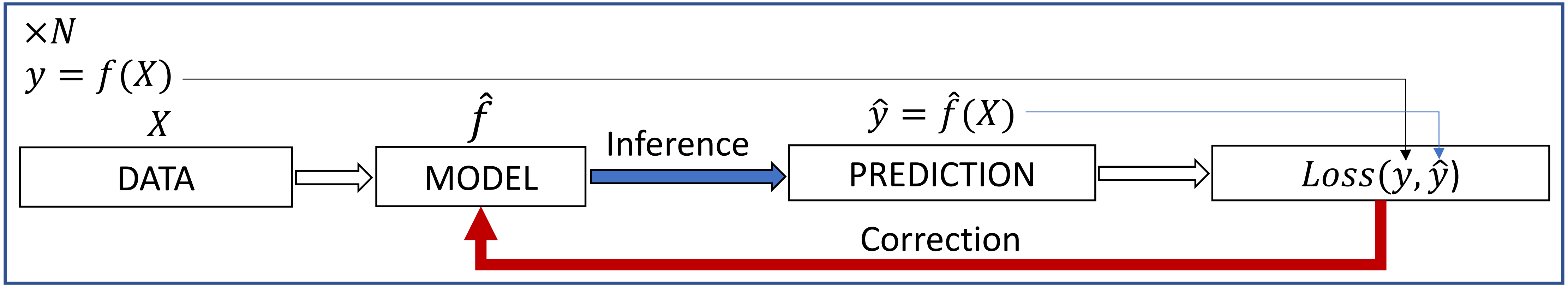
One iterative solution:

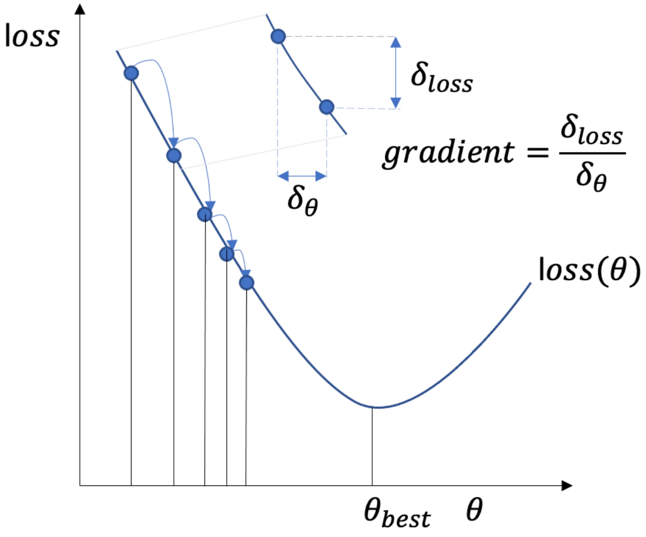
$$
repeat: \left| \begin{array}{l} \hat{y} \leftarrow \widehat{f_\theta}(X) \\ loss \leftarrow Loss(\hat{y}, y) \\ \theta \leftarrow \theta - \alpha \frac{\delta_{loss}}{\delta_\theta} \end{array} \right.
$$

α = learning rate



Train the model





One iterative solution:

repeat:

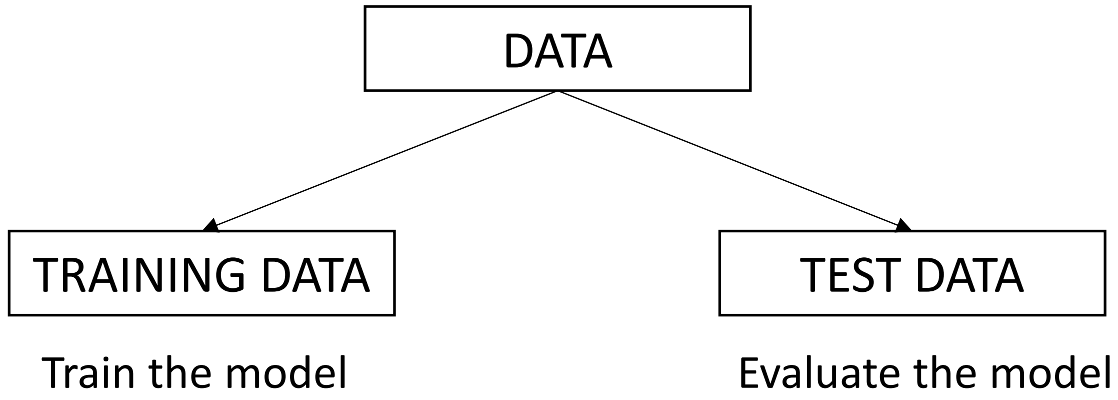
$$\hat{y} \leftarrow \hat{f}_\theta(X)$$

$$loss \leftarrow Loss(\hat{y}, y)$$

$$\theta \leftarrow \theta - \alpha \frac{\delta_{loss}}{\delta_\theta}$$

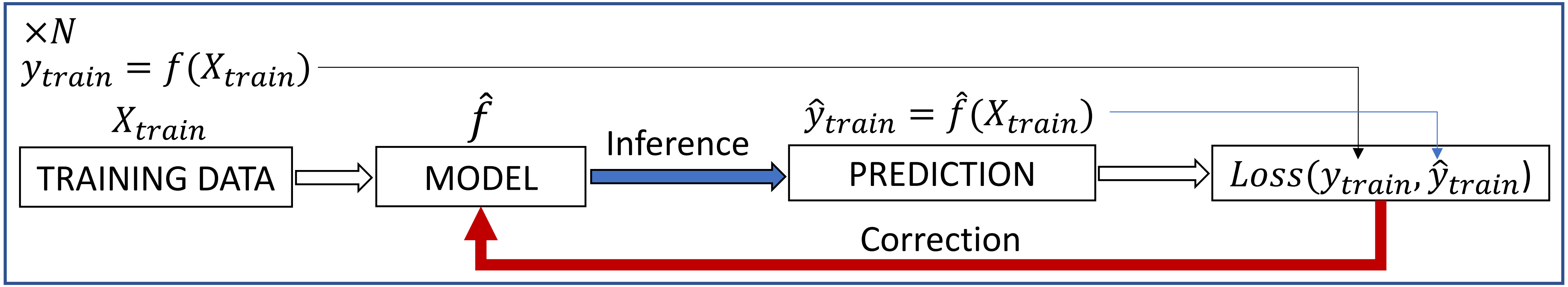
Split the data in TRAIN and TEST dataset

An important point of methodology:
It is need to *train* the model on a train dataset and *evaluate* the model (after the training) on a test dataset
As we want to evaluate the abilities of the model to generalize the train dataset and the test dataset have to be strictly different
Doing so we guarantee that the model will make predictions from the test dataset samples it has never seen during training

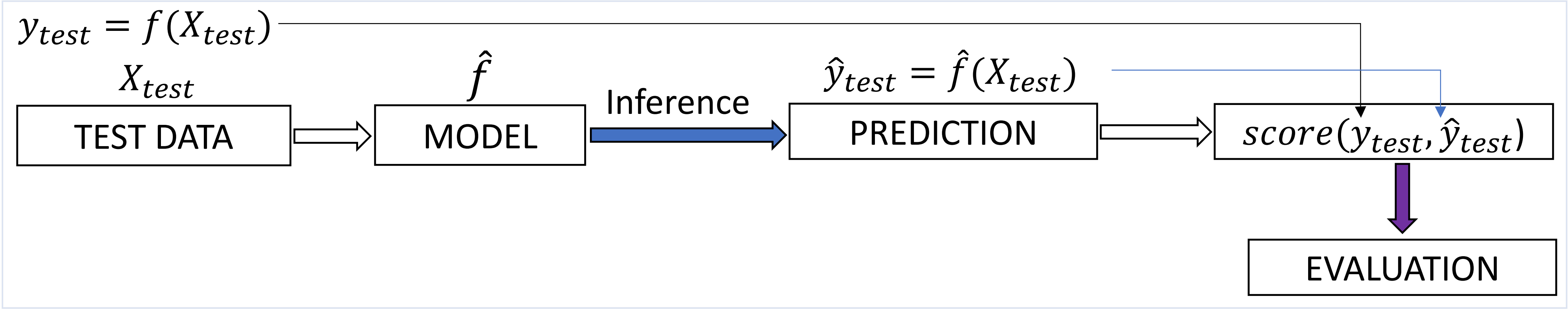


General process of model training and evaluation

TRAINING



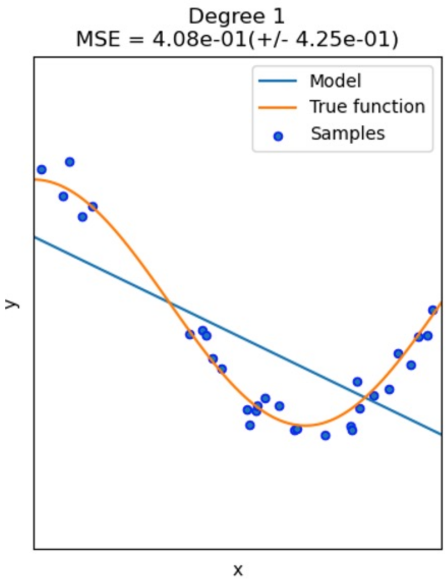
TEST (Unseen data during the training)



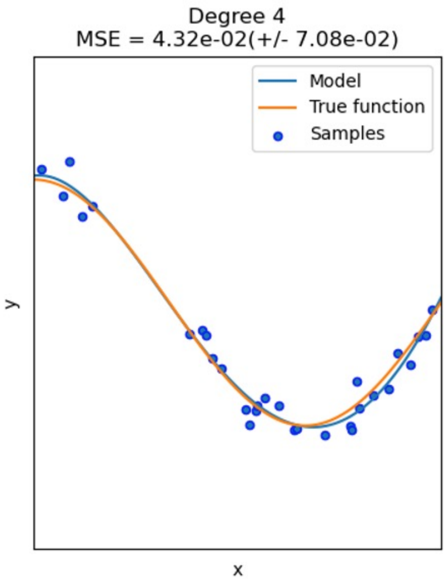
Metrics

Metric name/Evaluation method	Defintion
Accuracy	Out of 100 predictions, how many does your model get correct? E.g. 95% accuracy means it gets 95/100 predictions correct.
Precision	Proportion of true positives over total number of samples. Higher precision leads to less false positives (model predicts 1 when it should've been 0).
Recall	Proportion of true positives over total number of true positives and false negatives (model predicts 0 when it should've been 1). Higher recall leads to less false negatives.
F1-score	Combines precision and recall into one metric. 1 is best, 0 is worst.
Confusion matrix	Compares the predicted values with the true values in a tabular way, if 100% correct, all values in the matrix will be top left to bottom right (diagnol line).
Classification report	Collection of some of the main classification metrics such as precision, recall and f1-score.

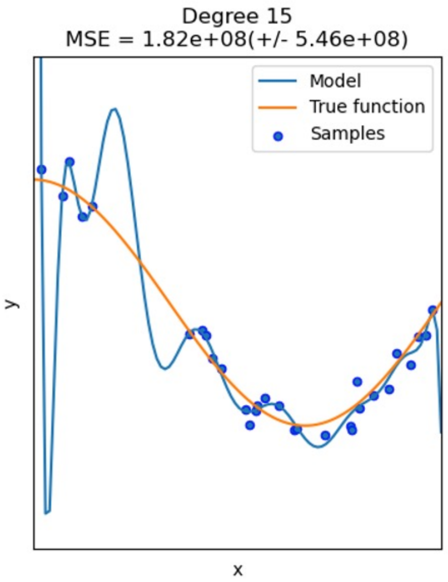
Underfitting and Overfitting



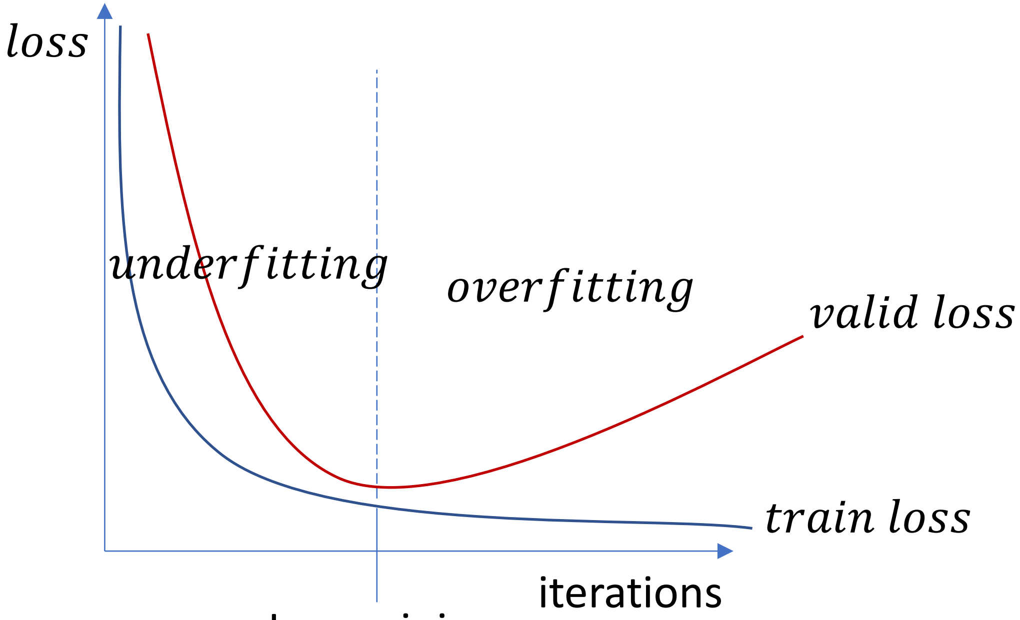
Underfitting
Model complexity
too low



Good fit / robustess
Model complexity ok



Overfitting
Model complexity
too high



Loss minima
for valid:
Early stopping

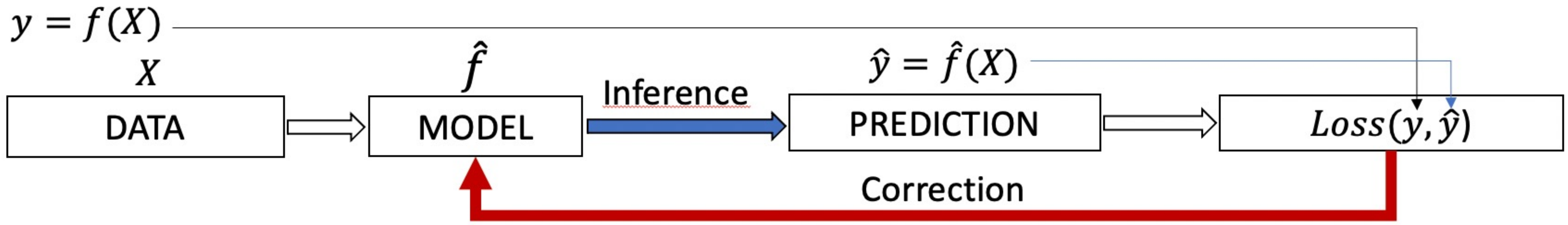
$\Rightarrow$ Split DATA between TRAIN, VAL, TEST datasets
$\Rightarrow$ VAL datasets is small and use to evaluate the model on non trained-on data *during* the training
$\Rightarrow$ Stop the training before overfitting

[*-learning]

- Machine Learning
  - Supervized Learning
    - Classification
      Identifying which category an object belongs to
    - Regression
      Predicting a continuous-valued attribute associated with an object
  - Unsupervized Learning
    - Clustering
      Automatic grouping of similar objects into sets
    - Dimensionality reduction
      Reducing the number of random variables to consider
  - Reinforcement Learning
    Semi-supervised Learning
    Self-supervized Learning

Supervized learning

Data consists of labelled examples : each data point contains features (covariates) and an associated label / target
Learn the mapping function between a sample features and its label

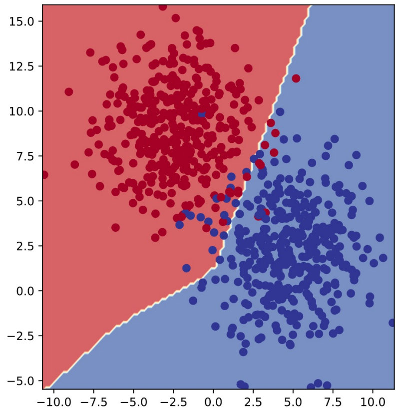


$$y = f(X) \begin{cases} \Rightarrow \text{We do have the analytic form: Compute directly or via a complete simulation of a complex system} \\ \Rightarrow \text{We do not have the analytic form: Human observation and annotation} \end{cases}$$

Supervized learning - Classification

A classification problem involves predicting whether something is one thing or another

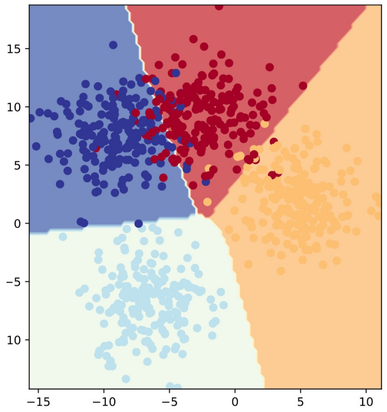
Identifying which category an object belongs to => Predict discrete values



Binary classification

Typical loss:

$$Loss_{BCE} = -\frac{1}{N}\sum_{i=0}^{N} y_i.\log(\hat{y}_i) + (1-y_i).\log(1-\hat{y}_i)$$



Multi Class classification

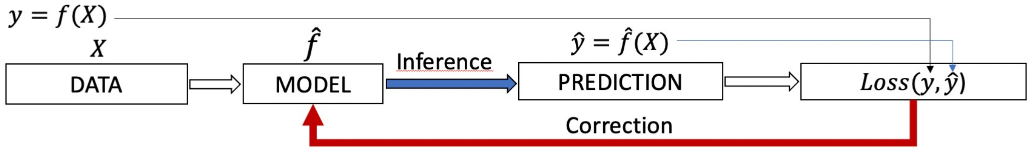
Typical loss:

$$Loss_{CE} = \ell(x,y) = \begin{cases} \frac{\sum_{n=1}^{N} l_n}{N}, & \text{if reduction} = \text{'mean'}; \\ \sum_{n=1}^{N} l_n, & \text{if reduction} = \text{'sum'}. \end{cases}$$

$$\ell(x,y) = L = \{l_1, \ldots, l_N\}^\top, \quad l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C}\exp(x_{n,i})} y_{n,c}$$

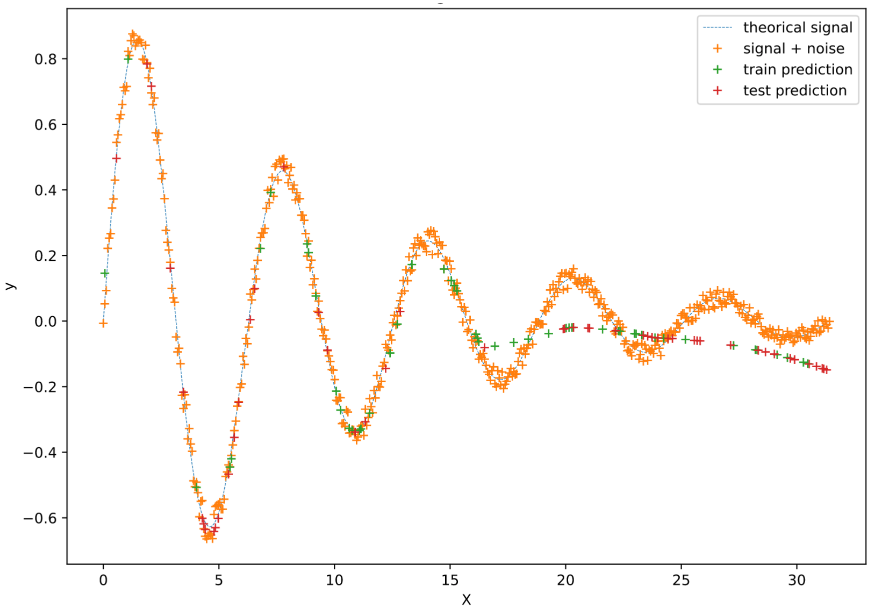
Applications: Spam detection, image recognition…

Algorithms: SVM, nearest neighbors, random forest, NNs



Supervized learning - Regression

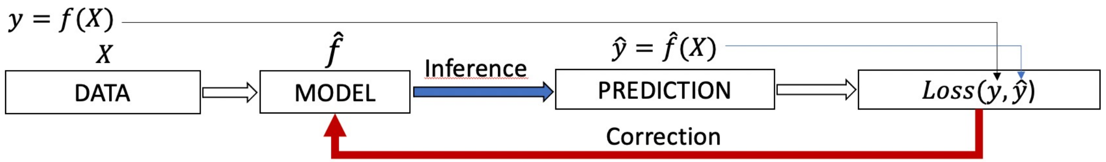
Predicting a continuous-valued attribute associated with an object.



Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, NNs

Typical loss:

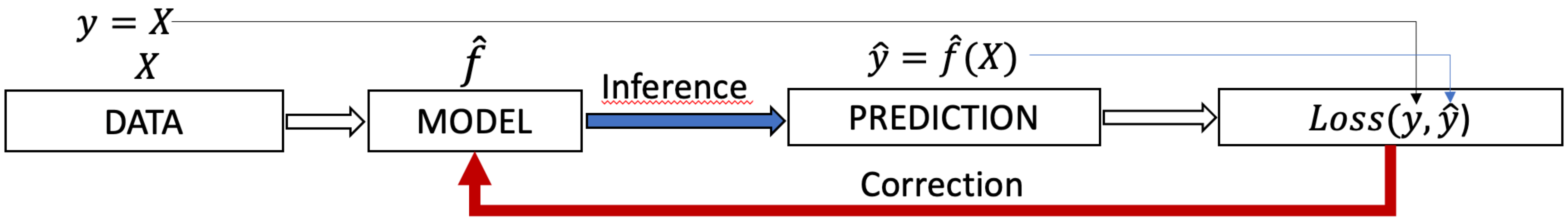
$$Loss_{MSE} = \frac{1}{N_{nodes}} \sum_{i=0}^{N_{nodes}} (y - \hat{y})^2$$



Unsupervized learning

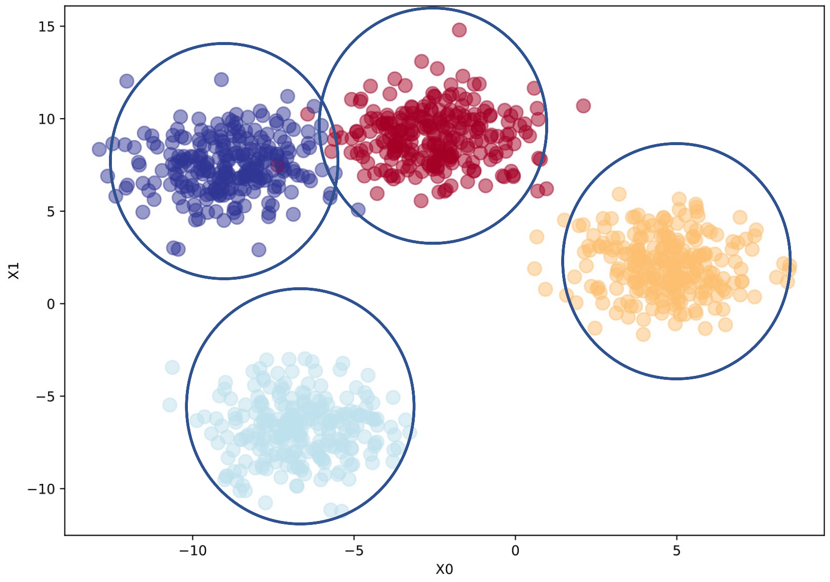
Uses machine learning algorithms to analyze and cluster unlabeled datasets.
These algorithms discover hidden patterns or data groupings without the need for human intervention

$$y = f(X) = X$$



Unsupervized learning - Clustering

Automatic grouping of similar objects into sets



.

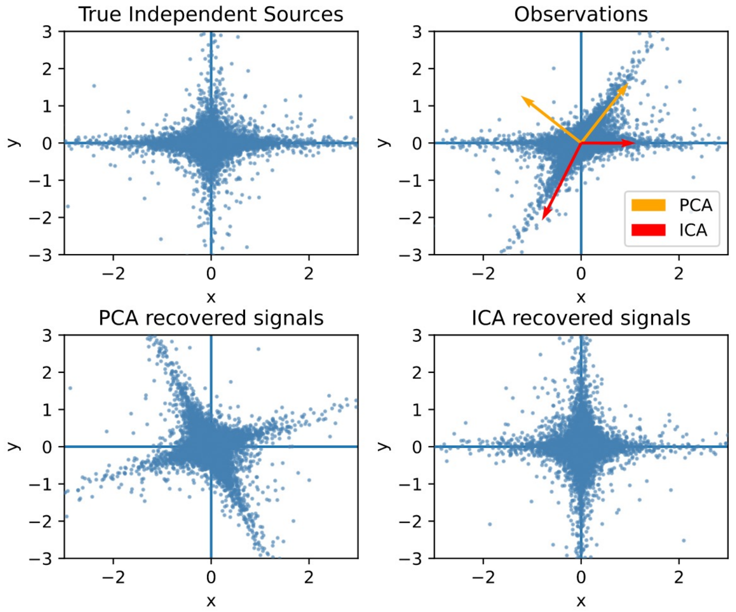
Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...

Unsupervized learning – Dimension reduction

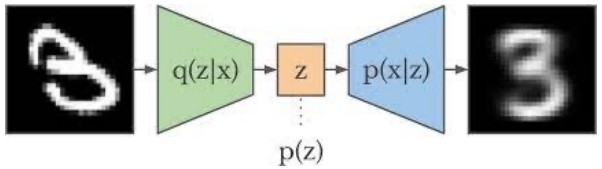
Reducing the number of random variables to consider.

Transformation of data from a high-dimensional space into a low-dimensional space
low-dimensional representation retains some meaningful properties of the original data



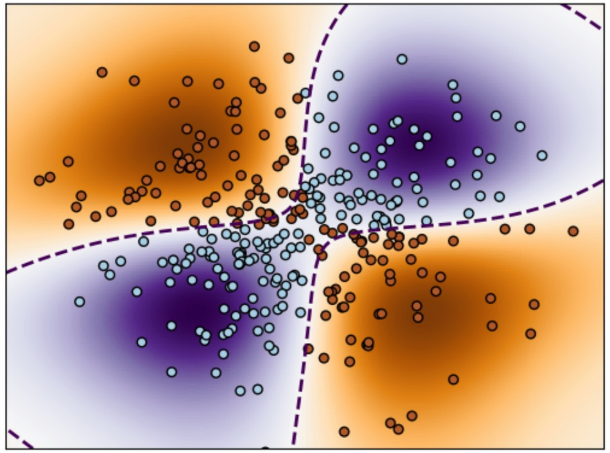
[Independent component analysis (ICA)]() vs
[Principal component analysis (PCA)]()
[FastICA on 2D point clouds]()
Authors: Alexandre Gramfort, Gael Varoquaux

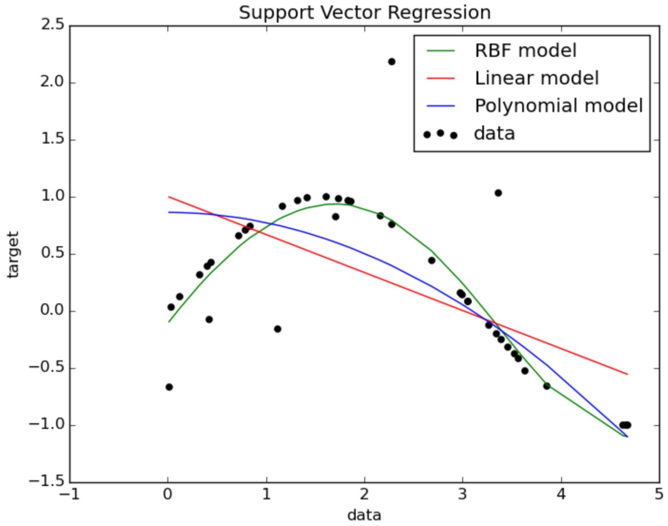
Variational Auto Encoders (VAEs)



Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, NNs

Non linearity





Projection of data in a higher dimentional space where samples are lineray separable (classification) or can fit linearly (regression)

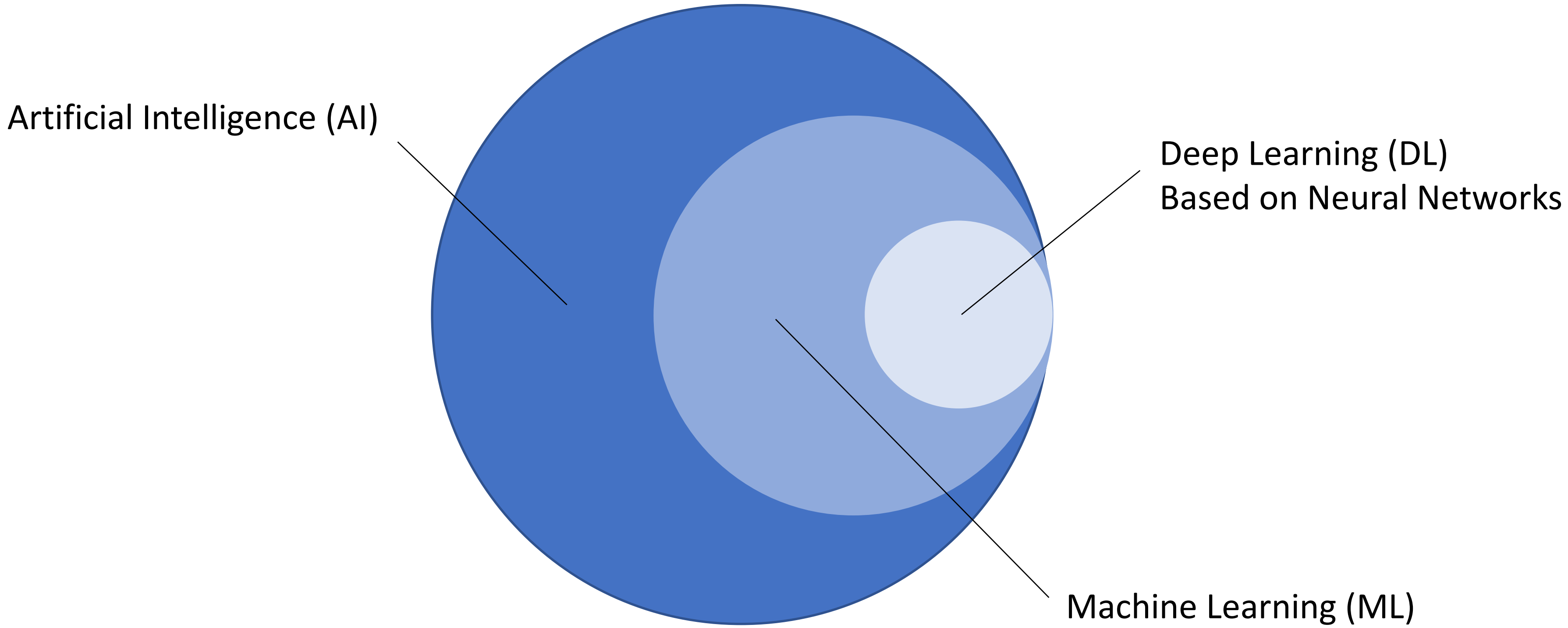
- Polynomial model: $X_1, X_2 \Rightarrow X_1, X_2, X_1X_2, X_1^2, X_2^2, \ldots \Rightarrow \hat{y} = \sigma(b + w_1x_1 + \ w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2 + \ \ldots)$
- Radial Basis Function model

$\Rightarrow$ Huge time computation cost for high dimension

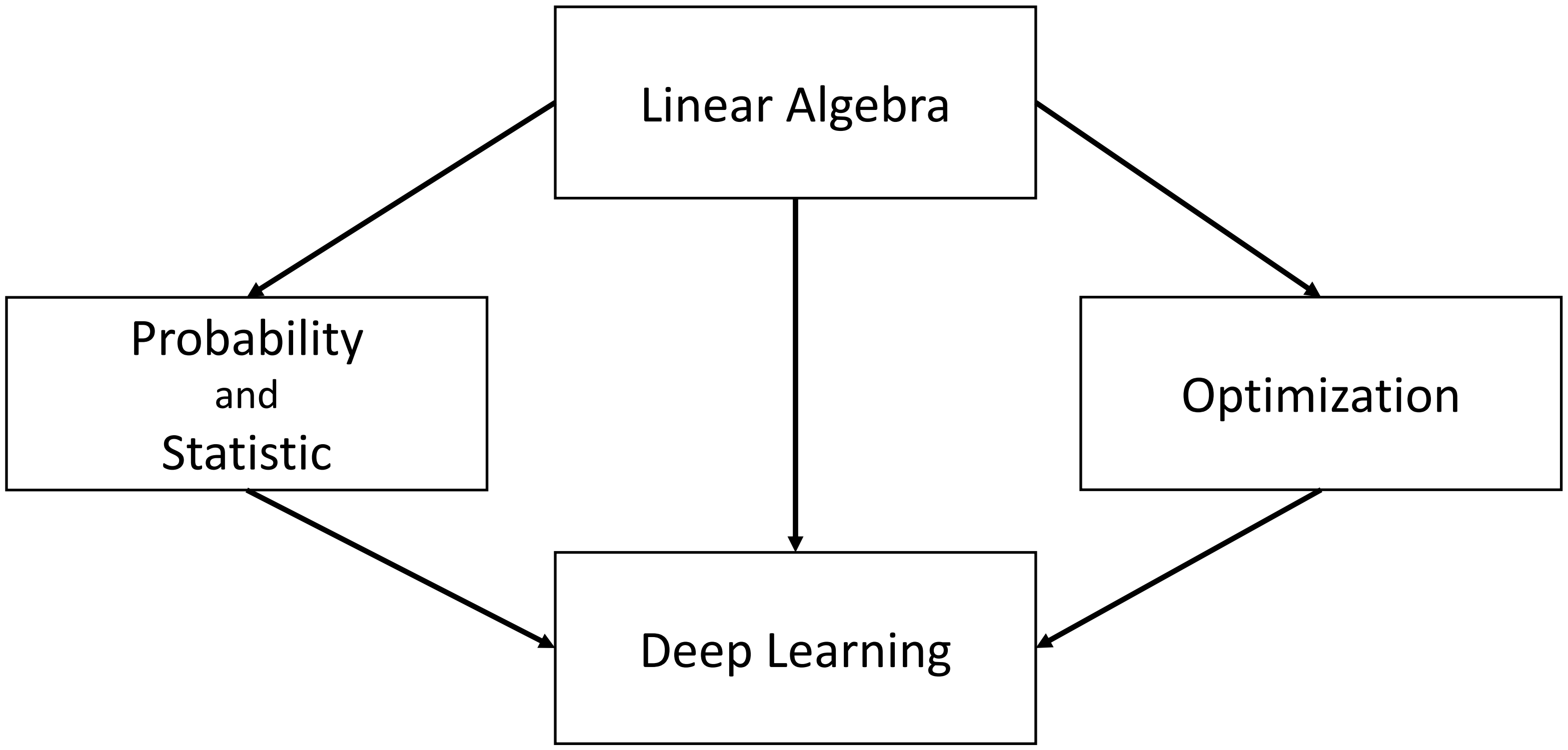
Here's come the powerness of Neural Nets !

Introduction to Neural Networks and Deep Learning

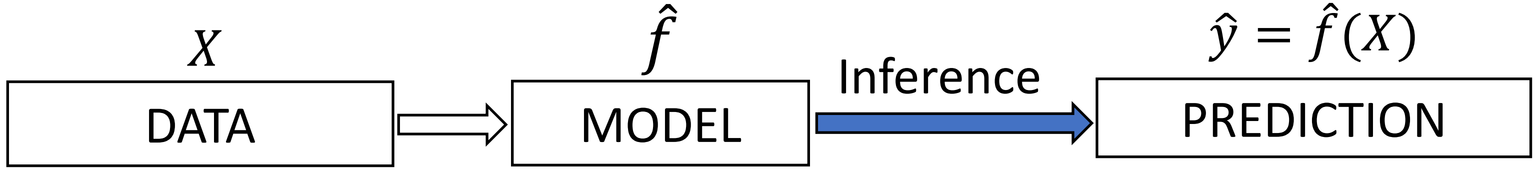
Deep learning



Deep learning



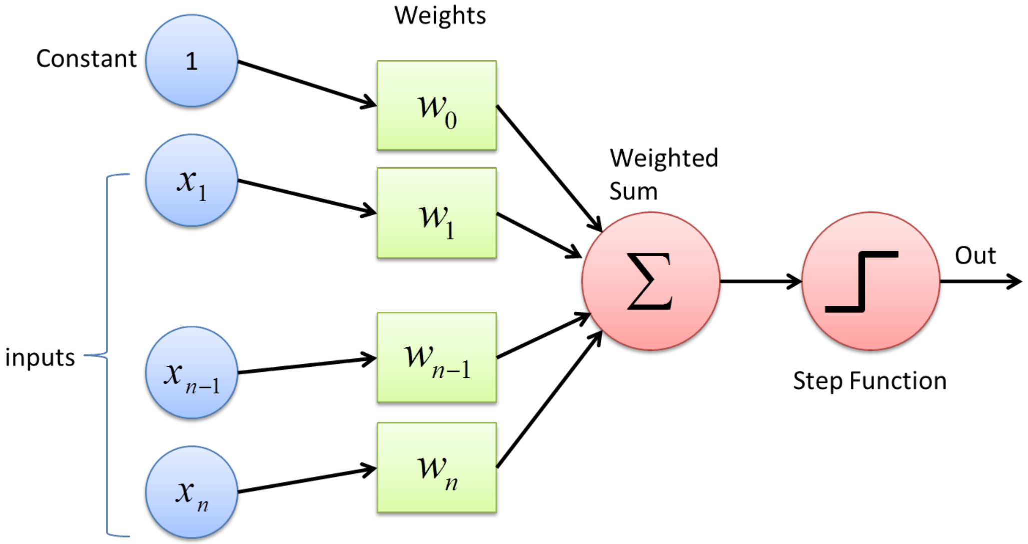
$$y = f(X)$$

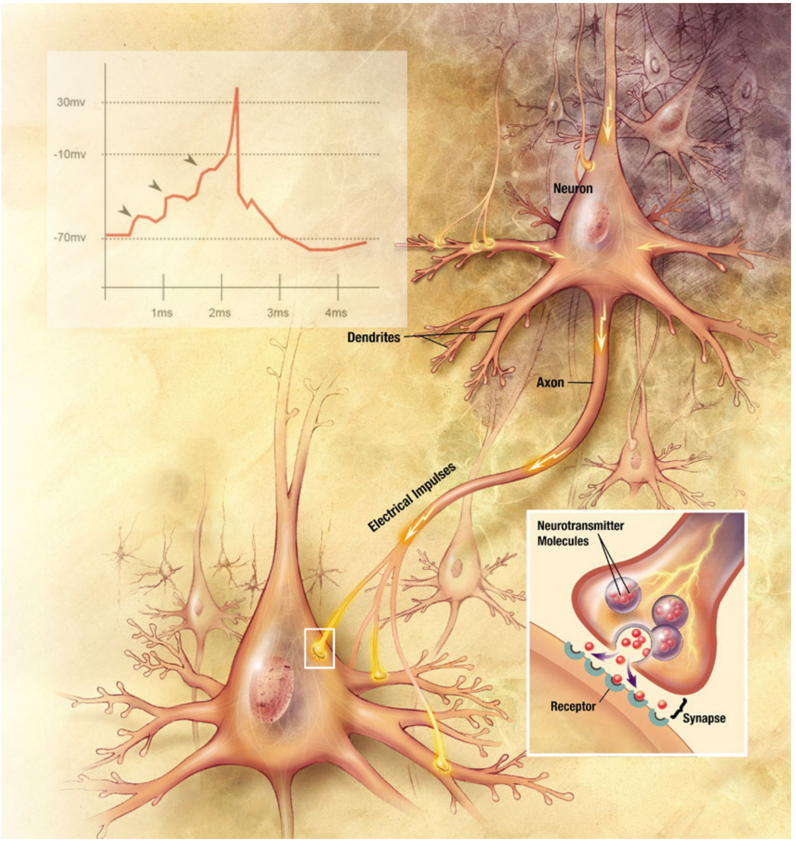


$\hat{f}$ is an approximation of f

$\hat{f}$ is implemented as a model which is a (Deep) Neural Networks

Perceptron: the first artificial neuron





Bio inspired:

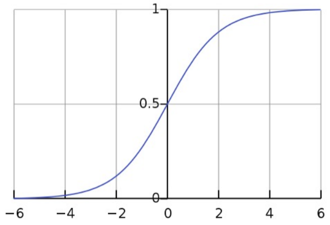
Weights correction Synaptic plasticity
Threshold for activation
Computing power comes from connexions with other neurons

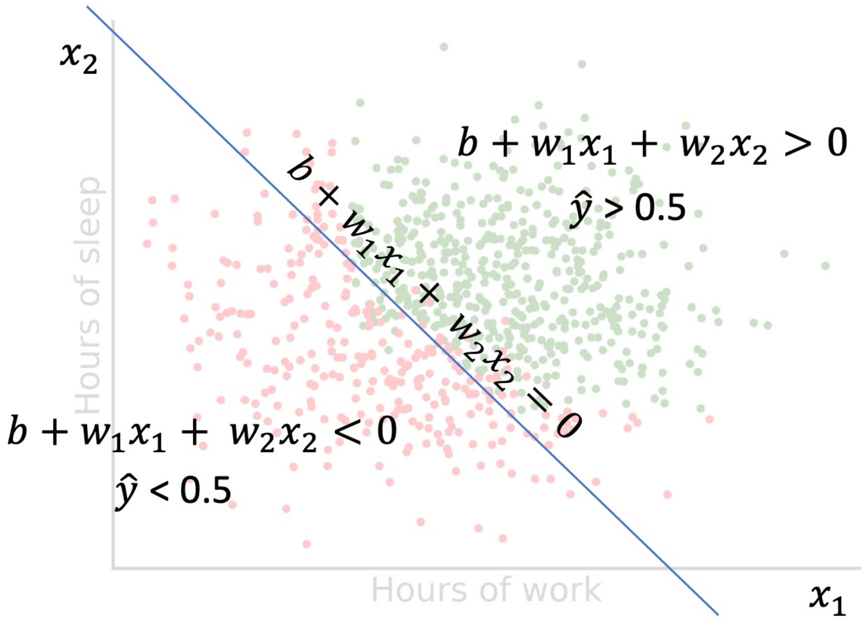
Huge simplification of real neurons…

Perceptron and logistic regression

$$\hat{y} = \sigma(b + w_1 x_1 + w_2 x_2)$$

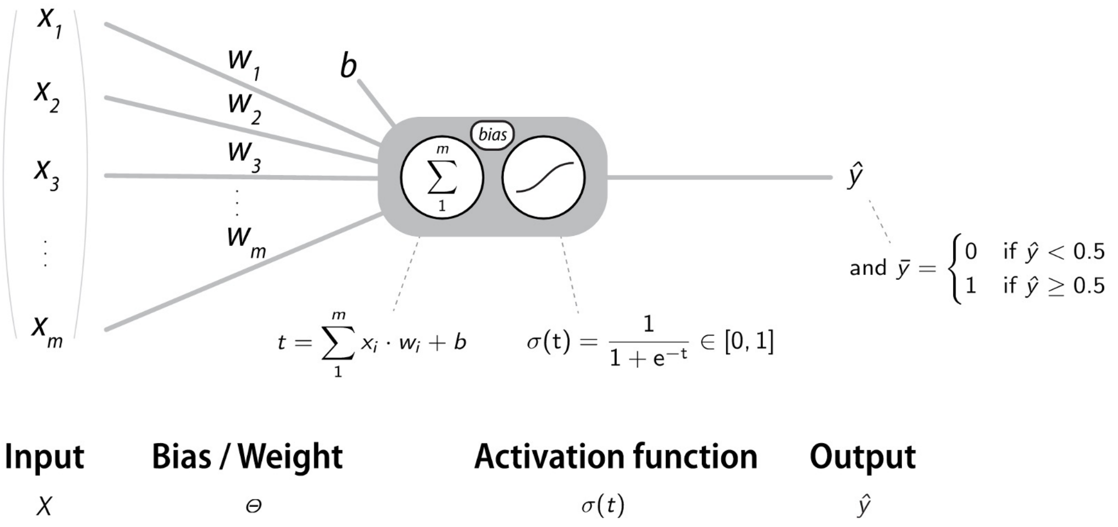
$$\sigma(z) = 1/(1 + e^{-z})$$





Generalization to m dimension:

$$\hat{y} = \sigma(\Theta^T \cdot X + b)$$

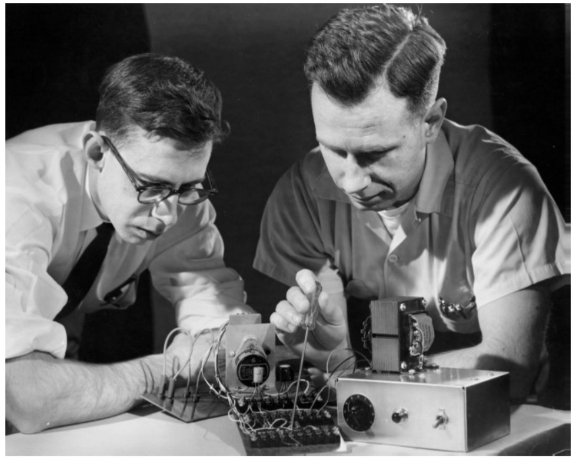


Perceptron: the first artificial neuron

$$data:\ X = \begin{bmatrix} x_0 & \cdots & x_{in\_features-1} \end{bmatrix}$$

$$weight:\ W = \begin{bmatrix} w^0 \\ \vdots \\ w^{in\_features-1} \end{bmatrix} \qquad bias:\ b$$

$$y = \sigma\left(\left(\sum_{i=0}^{in\_features-1} x_i \times w^i\right) + b\right) = \sigma(WX + b)$$



Division of Rare and Manuscript Collections

Frank Rosenblatt, left, and Charles W. Wightman work on part of the unit that became the first perceptron in December 1958.

Neural Network Linear layer

$data:\ X = [x_0 \quad \cdots \quad x_{in\_features-1}]$

Generalization of the *perceptron* to $out\_features$ dimension:

$$weights:\ W = \begin{bmatrix} w_0^0 & \cdots & w_{out\_features-1}^0 \\ \vdots & \ddots & \vdots \\ w_0^{in\_features-1} & \cdots & w_{out\_features-1}^{in\_features-1} \end{bmatrix} \qquad bias:\ b = \begin{bmatrix} b^0 \\ \vdots \\ b^{out\_features-1} \end{bmatrix}$$

σ : Activation function (ReLU, tanh, Sigmoid…)

=>Project input as linear combination of its features in a new latent space of dimension $out\_features$

$$X' = \sigma(WX + b) = \begin{bmatrix} \sum_{i=0}^{in\_features-1} x_i \times w_0^i \\ \vdots \\ \sum_{i=0}^{in\_features-1} x_i \times w_{out\_features-1}^i \end{bmatrix} + \begin{bmatrix} b^0 \\ \vdots \\ b^{out\_features-1} \end{bmatrix}$$

Neural Network Linear layer (N samples)

$$X = \begin{bmatrix} x_0^0 & \cdots & x_{in\_features-1}^0 \\ \vdots & \ddots & \vdots \\ x_0^{N-1} & \cdots & x_{in\_features-1}^{N-1} \end{bmatrix} \quad w = \begin{bmatrix} w_0^0 & \cdots & w_{out\_features-1}^0 \\ \vdots & \ddots & \vdots \\ w_0^{in\_features-1} & \cdots & w_{out\_features-1}^{in\_features-1} \end{bmatrix} \quad b = \begin{bmatrix} b^0 \\ \vdots \\ b^{out\_features-1} \end{bmatrix}$$

$$X' = WX + b = \begin{bmatrix} \sum_{i=0}^{in\_features-1} x_i^0 \times w_0^i & \cdots & \sum_{i=0}^{in\_features-1} x_i^0 \times w_{out\_features-1}^i \\ \vdots & \ddots & \vdots \\ \sum_{i=0}^{in\_features-1} x_i^{N-1} \times w_0^i & \cdots & \sum_{i=0}^{in\_features-1} x_i^{N-1} \times w_{out\_features-1}^i \end{bmatrix} + \begin{bmatrix} b^0 \\ \vdots \\ b^{out\_features-1} \end{bmatrix}$$

Why several samples ?

$\Rightarrow$ Structured data : we want the neural net to capture structural patterns
(typically CNNs in images, RNN in time series, Transformers in text, GNNs in graph)

and / or

$\Rightarrow$ Give in input a batch of samples to optimize GPU parallelization

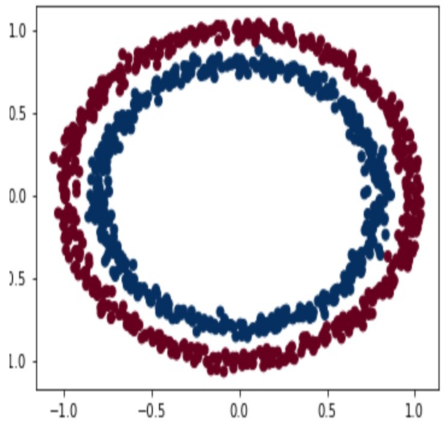
Multi Layer Perceptron (MLP)

$$\hat{y} = \hat{f}(X)$$

$$X_0 \xrightarrow{f_0 = Linear} X_1 \xrightarrow{f_1 = Linear} \dots \; X_n \xrightarrow{f_n = Linear} X_{n+1} \xrightarrow{f_{n+1} = Linear} \dots \xrightarrow{f_{final}} \hat{y} = X_{N\_1}$$

$\Rightarrow$ Sequence of Linear layers: Sucessive layers project the data features in successive latent spaces

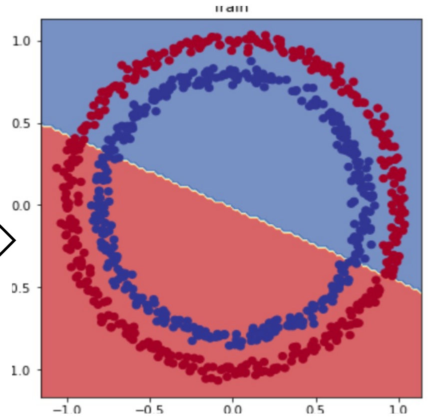
$\Rightarrow$ BUT VERY IMPORTANT: without non-linear activation function, the combination of Linear Layers will be linear so $\hat{f}$ will be linear, doesn't matter how *deep* is the Network



$$\hat{f}$$

$\Longrightarrow$

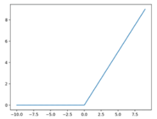
Linear Layers with no activation

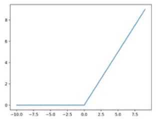


Induce non linearity with (non-linear) activation function

$$\hat{y} = \hat{f}(X)$$

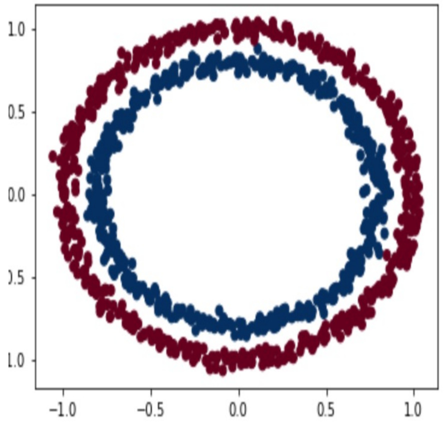
$$\overbrace{\boxed{X_0} \xrightarrow{f_0 = Linear} \boxed{X_1} \xrightarrow{f_1 = ReLU} \cdots \boxed{X_n} \xrightarrow{f_n = Linear} \boxed{X_{n+1}} \xrightarrow{f_{n+1} = ReLU} \cdots \xrightarrow{f_{final}}}^{} \boxed{\hat{y} = X_{N\_1}}$$





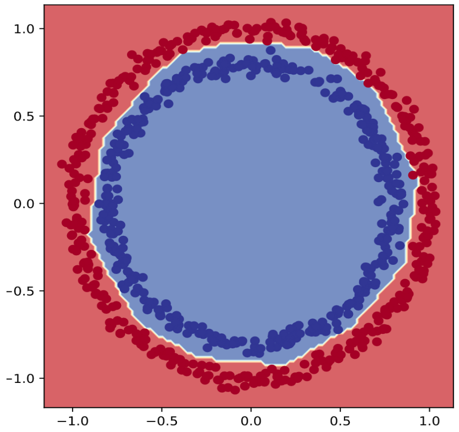
$\Rightarrow$ Non linearity is induced by non-linear activation functions, typically ReLU, tanh, etc

The Neural Network is now able to learn a non-linear function $\hat{f}$ by non linearly projecting features in non successive latent spaces. In the last latent space the sample are linearly separable.



$$\hat{f}$$

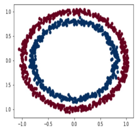
Linear Layer + non-linear activation

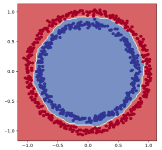


Last layer latent space representation

$$\hat{y} = \hat{f}(X)$$

$$X_0 \xrightarrow{f_0} X_1 = f_0\,(X_0) \rightarrow \cdots \quad X_n \xrightarrow{f_n} X_{n+1} = f_n(X_n) = \sigma(W_n X_n + b_n) \rightarrow \cdots \xrightarrow{f_{final}} \hat{y} = X_{N\_1}$$





So that's it? A succession of linear projection and simple non linear activation function, that's the secret of Deep Learning ?

That's the beauty and the strangeness of the thing that something so simple is so powerful.

That’s one of the secret but not the only one...
In particular the strength of DL come also from other architectures of Neural Networks which can learn from patterns in structured data

Final projection and loss computation

$$\hat{y} = \hat{f}(X)$$

$$\boxed{X_0} \xrightarrow{f_0} \boxed{X_1 = f_0\,(X_0)} \rightarrow \cdots \quad \boxed{X_n} \xrightarrow{f_n} \boxed{X_{n+1} = f_n(X_n) = \sigma(W_n X_n + b_n)} \rightarrow \cdots \xrightarrow{f_{final}} \boxed{\hat{y} = X_{N\_1}}$$

Depending on the task we want, we can then apply different *final activation* and compute *specific Loss function*

Binary classification

Final activation: $sigmoid$
Loss: $Binary\ Cross\ Entropy\ Loss$

$$f_{final} = sigmoid$$

$$Loss_{BCE} = -\frac{1}{N}\sum_{i=0}^{N} y_i.\log(\hat{y}_i) + (1 - y_i).\log(1 - \hat{y}_i)$$

Multiclass classification

Final activation: $sofmax$
Loss: $Cross\ Entropy\ Loss$

Regression

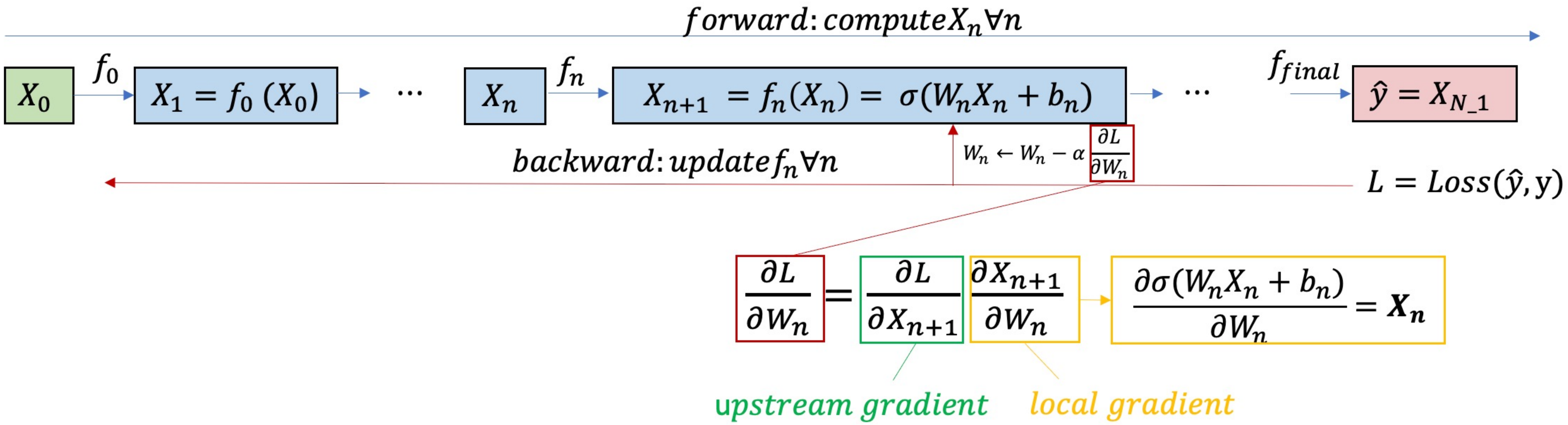
Final activation: $no\ activation$
Loss: $MSE\ Loss$

$$f_{final} = Linear$$

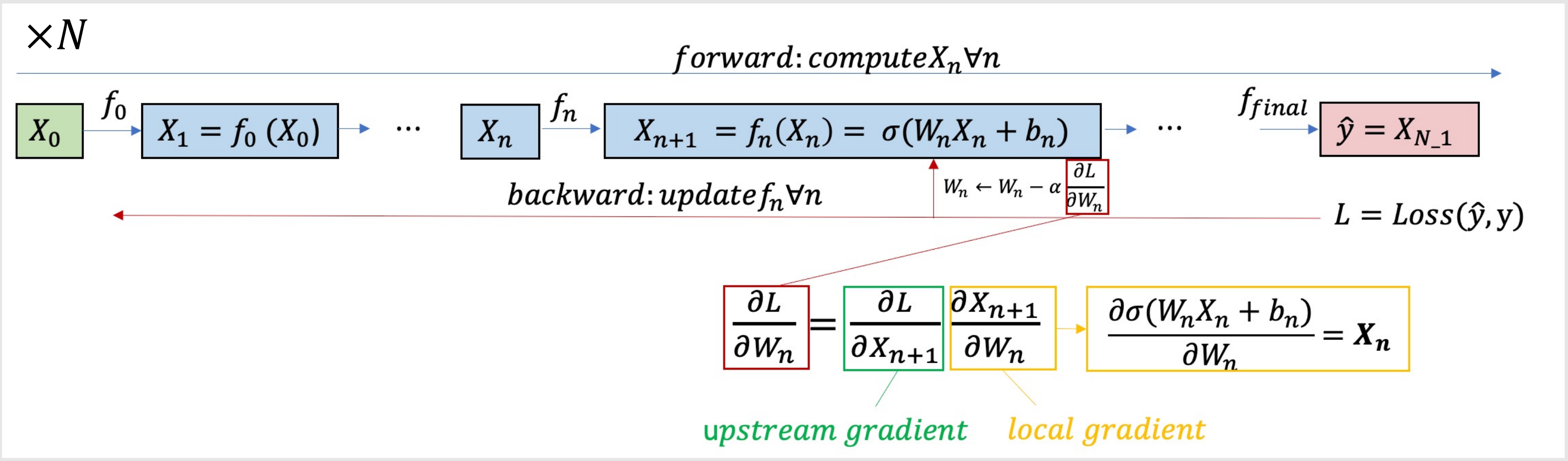
$$Loss_{MSE} = \frac{1}{N_{nodes}} \sum_{i=0}^{N_{nodes}} (y - \hat{y}\,)^2$$

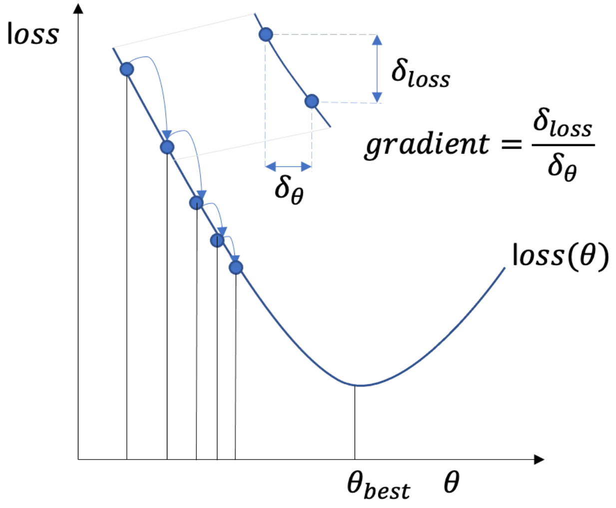
...

Loss optimization thanks to backpropagation



Neural Network training: repeat on all TRAIN dataset





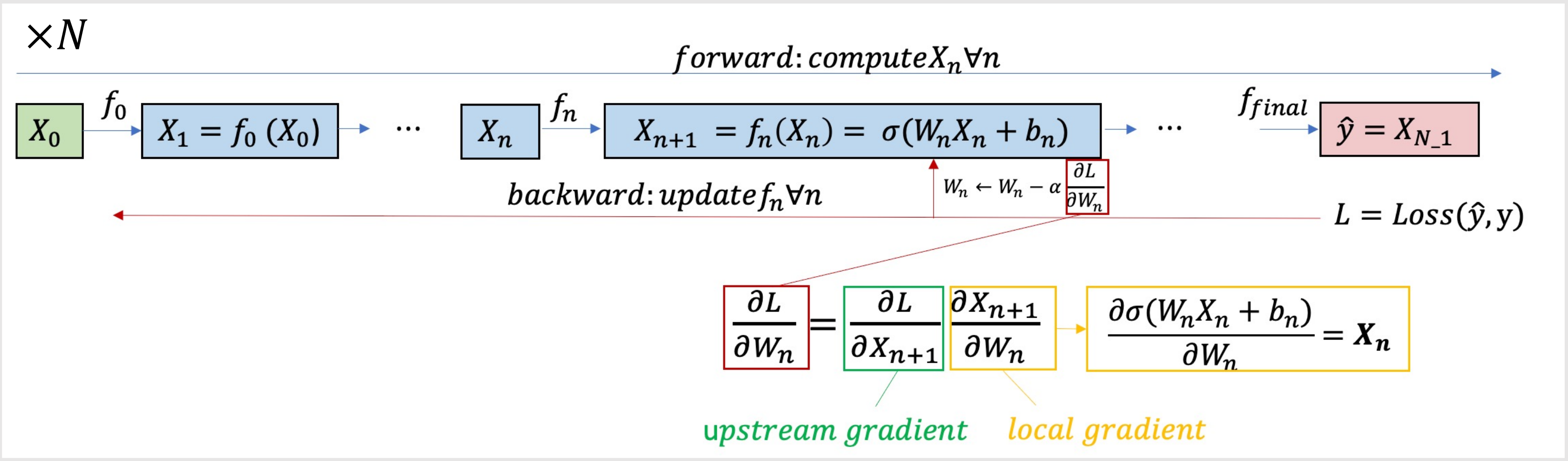
$repeat:$

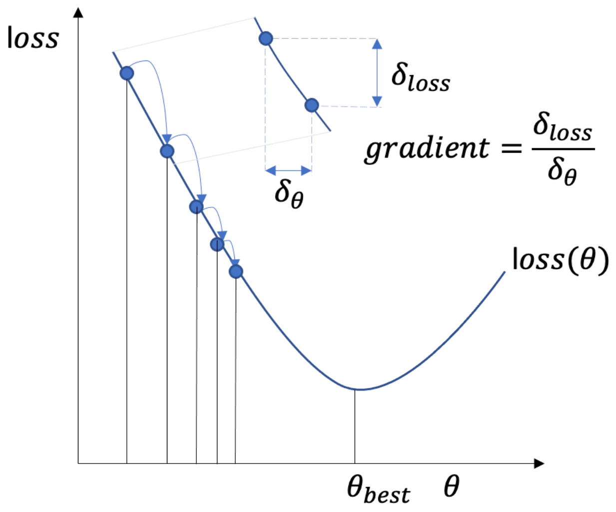
$$\hat{y} \leftarrow \widehat{f_\theta}(X)$$

$$loss \leftarrow Loss(\hat{y}, y)$$

$$\theta \leftarrow \theta - \alpha \frac{\delta_{loss}}{\delta_\theta}$$

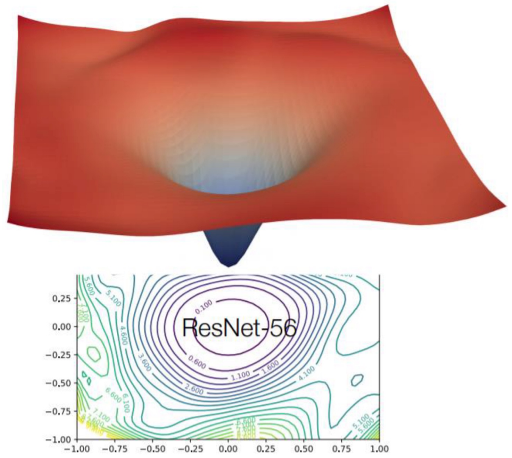
Neural Networks training challenge:
Find (the best?) local minima in the model’s parameters space

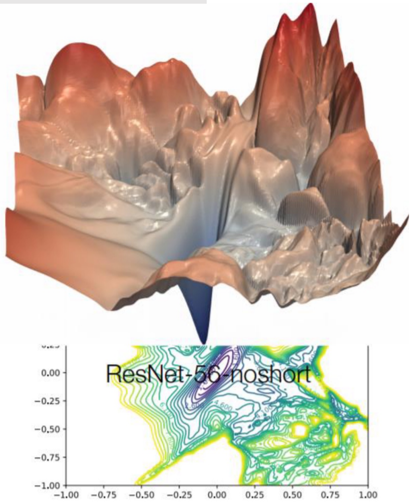




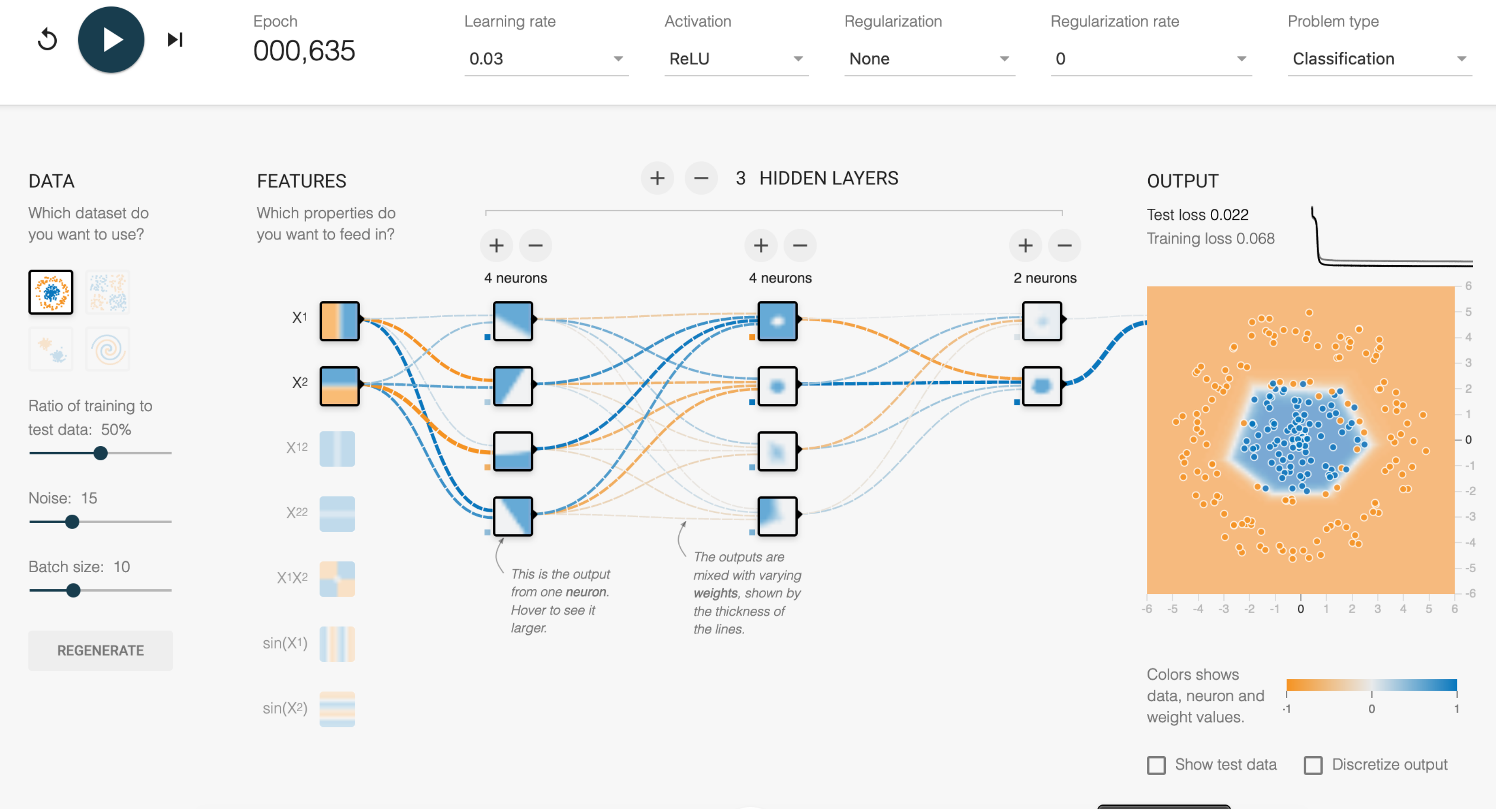
$repeat:$

$$
\begin{aligned}
\hat{y} &\leftarrow \widehat{f_\theta}(X) \\
loss &\leftarrow Loss(\hat{y}, y) \\
\theta &\leftarrow \theta - \alpha \frac{\delta_{loss}}{\delta_\theta}
\end{aligned}
$$





Let’s play a little



[TensorFlow Playground website]

Deep learning Neural Networks architectures

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs), Long Short Term Memory
- Transformers
- Variational Auto Encoders (VAEs)
- Generative Adversial Networks (GANs)
- Graph Neural Networks (GNNs)
- Diffusion models

Deep Learning in practice: introduction to pytorch

Classification of non-linearly separable data

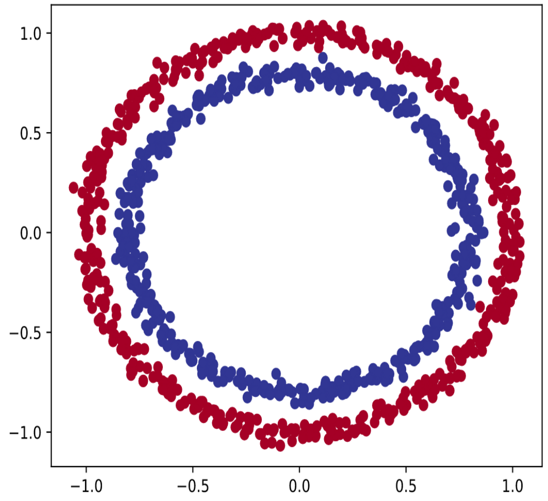
```python
# Make 1000 samples
n_samples = 1000

# Create circles
X, y = make_circles(n_samples,
                    noise=0.03,
                    random_state=42)
```

```python
print(y[:10])
```

```
[1 1 1 1 0 1 1 1 1 0]
```

```python
print(X[:10])
```

```
[[ 0.75424625  0.23148074]
 [-0.75615888  0.15325888]
 [-0.81539193  0.17328203]
 [-0.39373073  0.69288277]
 [ 0.44220765 -0.89672343]
 [-0.47964637  0.67643477]
 [-0.01364836  0.80334872]
 [ 0.77151327  0.14775959]
 [-0.16932234 -0.79345575]
 [-0.1214858   1.02150905]]
```



$X = \boxed{\text{DATA}}$

$y = f(X)$ f: Classification function

Train a model to approximate f and classify between the first circle and the second one

Define the model

```python
input_size, n_layers, hidden = 2, 2, 10

# Build model with non-linear activation function
from torch import nn
class Classifier(nn.Module):

    def __init__(self, input_size, n_layers, hidden):
        super().__init__()
        layers = []
        layers.append(nn.Linear(in_features=input_size, out_features=hidden))
        layers.append(nn.ReLU())
        for i in range(n_layers):
            layers.append(nn.Linear(in_features=hidden, out_features=hidden))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(in_features=hidden, out_features=1))
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

$\hat{f}$

MODEL

Design the Neural Network

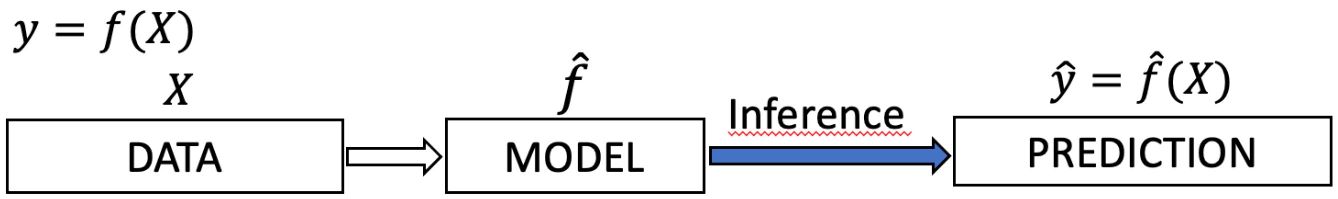
```python
model = Classifier(input_size, n_layers, hidden).to(device)
print(model)
```

Instanciate the model (call the init function of the class Classifier)

Inference

```python
input_size, n_layers, hidden = 2, 2, 10

# Build model with non-linear activation function
from torch import nn
class Classifier(nn.Module):

    def __init__(self, input_size, n_layers, hidden):
        super().__init__()
        layers = []
        layers.append(nn.Linear(in_features=input_size, out_features=hidden))
        layers.append(nn.ReLU())
        for i in range(n_layers):
            layers.append(nn.Linear(in_features=hidden, out_features=hidden))
            layers.append(nn.ReLU())
        layers.append(nn.Linear(in_features=hidden, out_features=1))
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)

model = Classifier(input_size, n_layers, hidden).to(device)

y_pred = model(X)
```

Once instantiated the model can be call to compute inference
The method forward of the object is called



Define Loss

Binary classification task => Binary Cross Entropy Loss

```
loss_fn = nn.BCEWithLogitsLoss() # BCEWithLogitsLoss = sigmoid built-in
```

With logits means sigmoid have not been apply as last activation of the model. It will be applied inside the Loss function

$$Loss_{BCEWithLogitsLoss} = -\frac{1}{N}\sum_{i=0}^{N} y_i . \log(\sigma(\hat{y}_i)) + (1 - y_i) . \log(1 - \sigma(\hat{y}_i))$$

Define Optimizer

```
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.001)
```

Model parameters

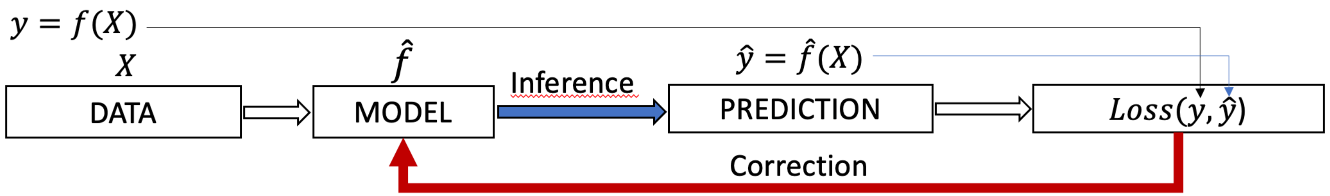
Learning rate

It’s the optimizer which will update model parameters

One step of training

```
# 1. Forward pass
y_pred = model(X)
# 2. Calculate loss
loss = loss_fn(y_pred, y_train_batch)
# 3. Optimizer zero grad
optimizer.zero_grad()
# 4. Loss backwards
loss.backward()
# 5. Optimizer step
optimizer.step()
```

1.Forward pass - The model goes through all of the training data once, performing its forward() function calculations
2.Calculate the loss - The model's outputs (predictions) are compared to the ground truth and evaluated to see how wrong they are
3.Zero gradients - The optimizers gradients are set to zero (they are accumulated by default) so they can be recalculated for the specific training step
4.Perform backpropagation on the loss - Computes the gradient of the loss with respect for every model parameter to be updated (each parameter with requires_grad=True). This is known as **backpropagation**, hence "backwards"
5.Step the optimizer (gradient descent) - Update the parameters with requires_grad=True with respect to the loss gradients in order to improve them

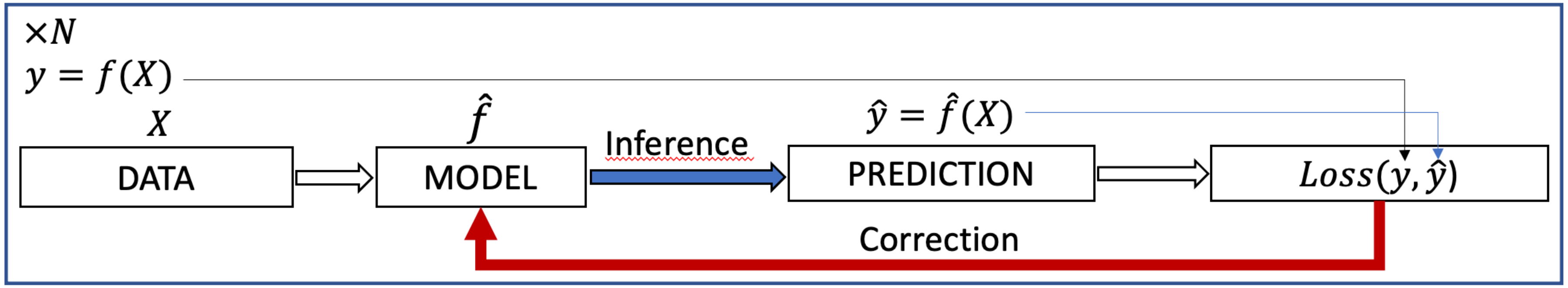


Training loop

```
for epoch in range(epochs):
    model.train()
    for X_train_batch, y_train_batch in train_dataloader:
        # 1. Forward pass
        y_pred = model(X)
        # 2. Calculate loss
        loss = loss_fn(y_pred, y_train_batch)
        # 3. Optimizer zero grad
        optimizer.zero_grad()
        # 4. Loss backwards
        loss.backward()
        # 5. Optimizer step
        optimizer.step()
```

Loop on number of epochs. For each epoch all the sample of the train dataset is given as input to the model

Loop on all the train dataset



Tips: model improvement techniques

Model improvement technique	What does it do?
Add more layers	Each layer *potentially* increases the learning capabilities of the model with each layer being able to learn some kind of new pattern in the data, more layers is often referred to as making your neural network *deeper*.
Add more hidden units	Similar to the above, more hidden units per layer means a *potential* increase in learning capabilities of the model, more hidden units is often referred to as making your neural network *wider*.
Fitting for longer (more epochs)	Your model might learn more if it had more opportunities to look at the data.
Changing the activation functions	Some data just can't be fit with only straight lines (like what we've seen), using non-linear activation functions can help with this (hint, hint).
Change the learning rate	Less model specific, but still related, the learning rate of the optimizer decides how much a model should change its parameters each step, too much and the model overcorrects, too little and it doesn't learn enough.
Change the loss function	Again, less model specific but still important, different problems require different loss functions. For example, a binary cross entropy loss function won't work with a multi-class classification problem.

A lot more to learn

- Methodology
- Hyperparameters search
- Visualization tools

Science and Deep Learning

- Interpretability
- Reproductibilty
- Convergence
- Ethics

Good questions to ask yourself

- What are my data ?
- What is my problem ?
- Do I need ML ?
- Do I need DL ?
- What do I want the model to learn ?
- What task I want to train the model for ?

A few recent (sucess?) stories of Deep Learning

Explainable AI and uncertainty quantification in CV

Grad CAM++ algorithm applied to a CNN. Colors represent filter activations. A hotter color means more emphasis was given on those pixels by the model.

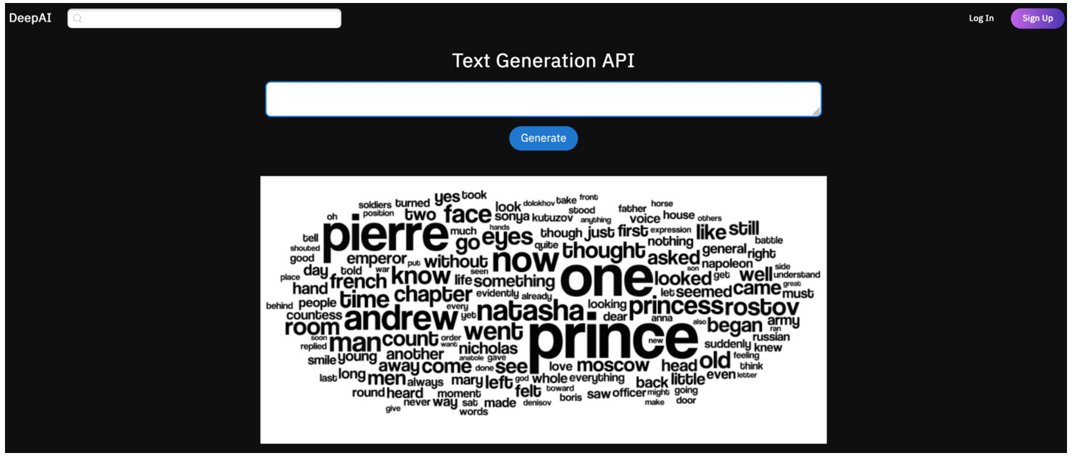
Natural Language Processing

Revolution since 2018 and the use of Transformers architecture based on attention mecanism

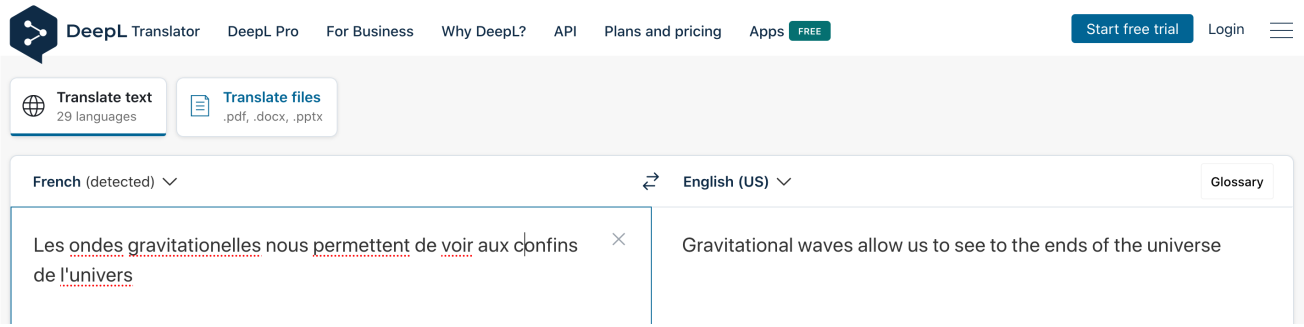
Google's BERT and OpenAI's GPT-2 and GPT-3.

The text-encoder is responsible for capturing the complexity and semantic meaning of an arbitrary input sentence. It captures these features by projecting the text sequence in a high dimensional embedding space

Automatic Text Generation



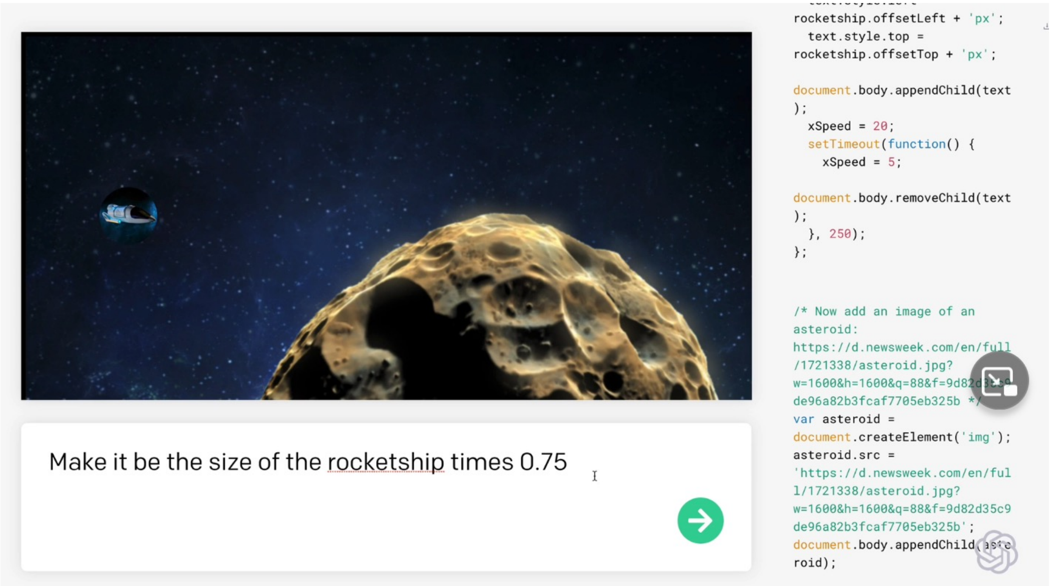
Automatic Translation

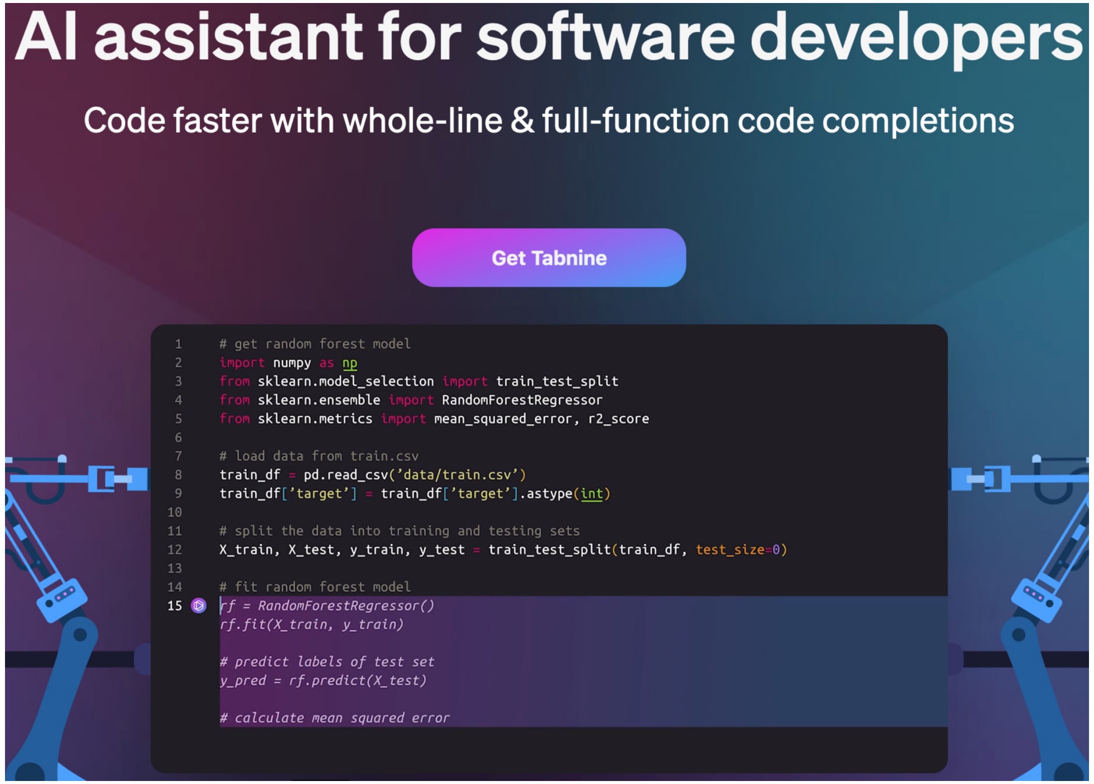


10 best language models in 2022

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
GPT2: Language Models Are Unsupervised Multitask Learners
XLNet: Generalized Autoregressive Pretraining for Language Understanding
RoBERTa: A Robustly Optimized BERT Pretraining Approach
ALBERT: A Lite BERT for Self-supervised Learning of Language Representations
T5: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer
GPT3: Language Models Are Few-Shot Learners
ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators
DeBERTa: Decoding-enhanced BERT with Disentangled Attention
PaLM: Scaling Language Modeling with Pathways

Automatic code generation

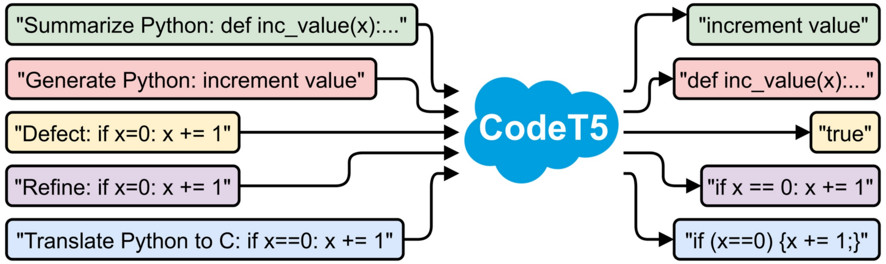




CodeT5: The Code-aware Encoder-Decoder based Pre-trained Programming Language Models

Yue Wang Steven Hoi

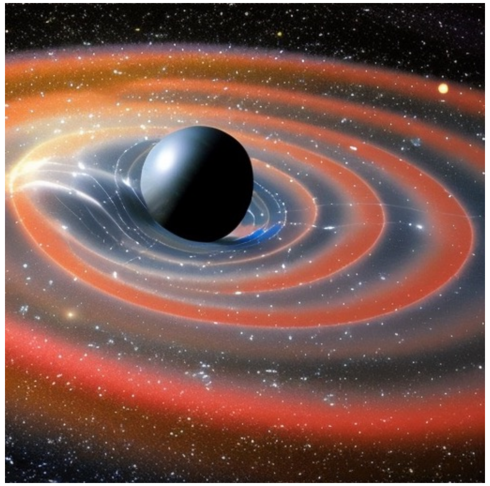
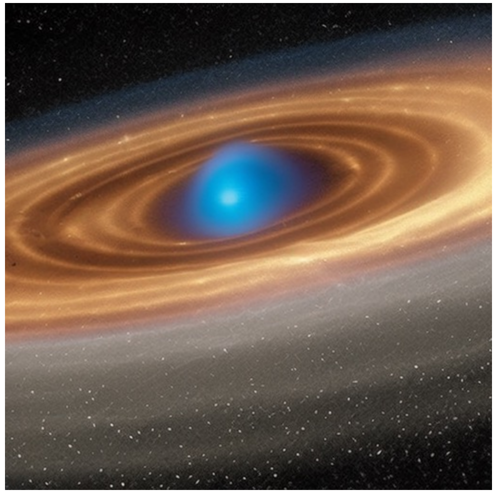
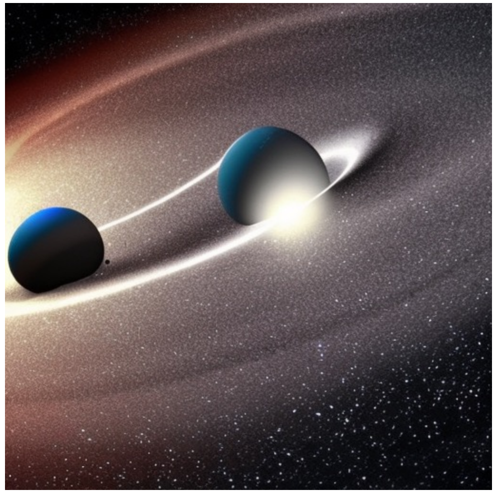
September 03, 2021 · *7 min read*



Soon computer science engineer useless ? ☺

Text-to-Image with Diffusion Models

"Gravitational waves allow us to see to the ends of the universe, But what will we see ?"

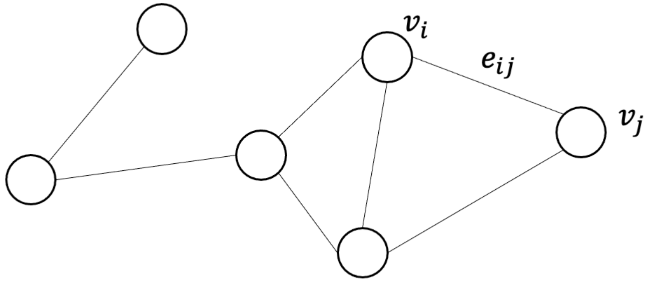





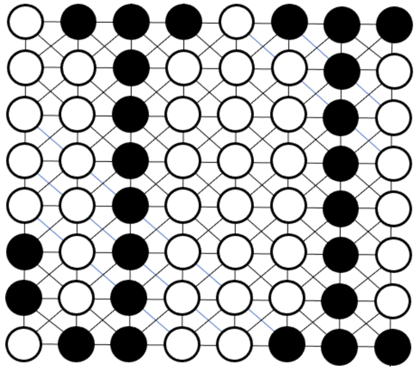
Danger of (stupid or serious) deep fake

The rise of geometric ML and representation learning

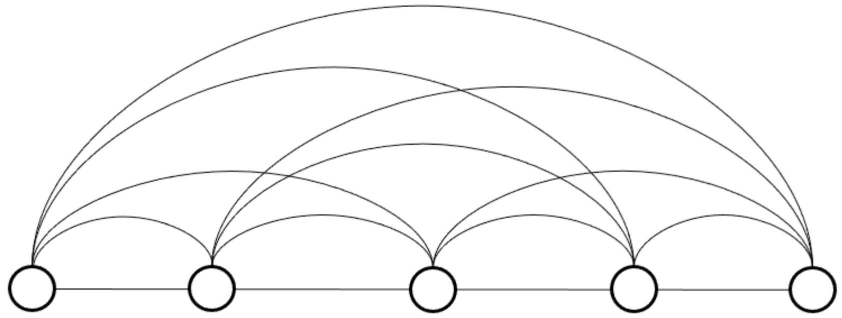
⇒ Geometric and graph-based ML methods have become one of the hottest fields of AI research
⇒ Graph Neural Networks (GNNs) capture deep geometric and structural patterns in data represented as graph

[What does 2022 hold for Geometric & Graph ML?](#)
Michael Bronstein





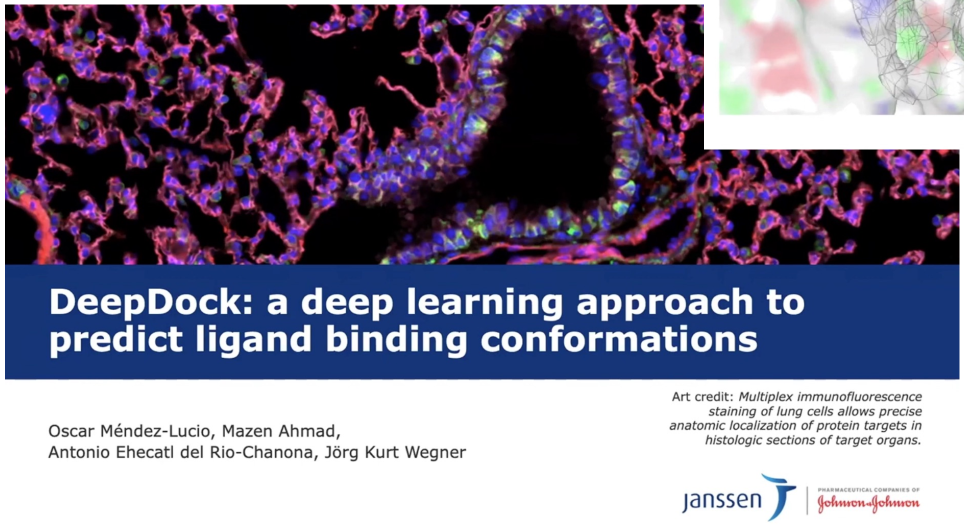
CNNs can be seen as a specific use case of GNN on regular grid graph

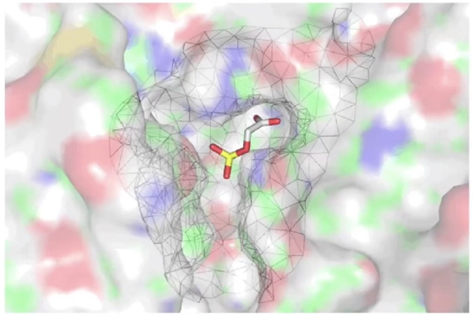


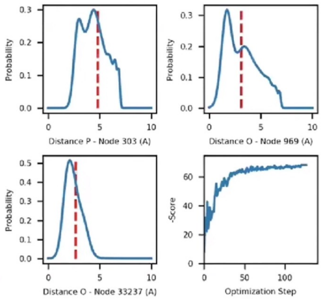
Transformers architectures in Natural Language Processing operate on fully connected graph

Bioactive molecule design with geometric deep learning

Geometric deeplearning is a promising direction in molecular design and drug screening.

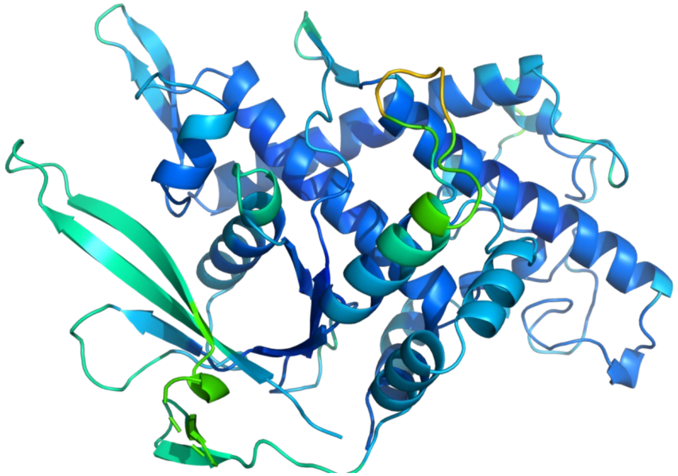




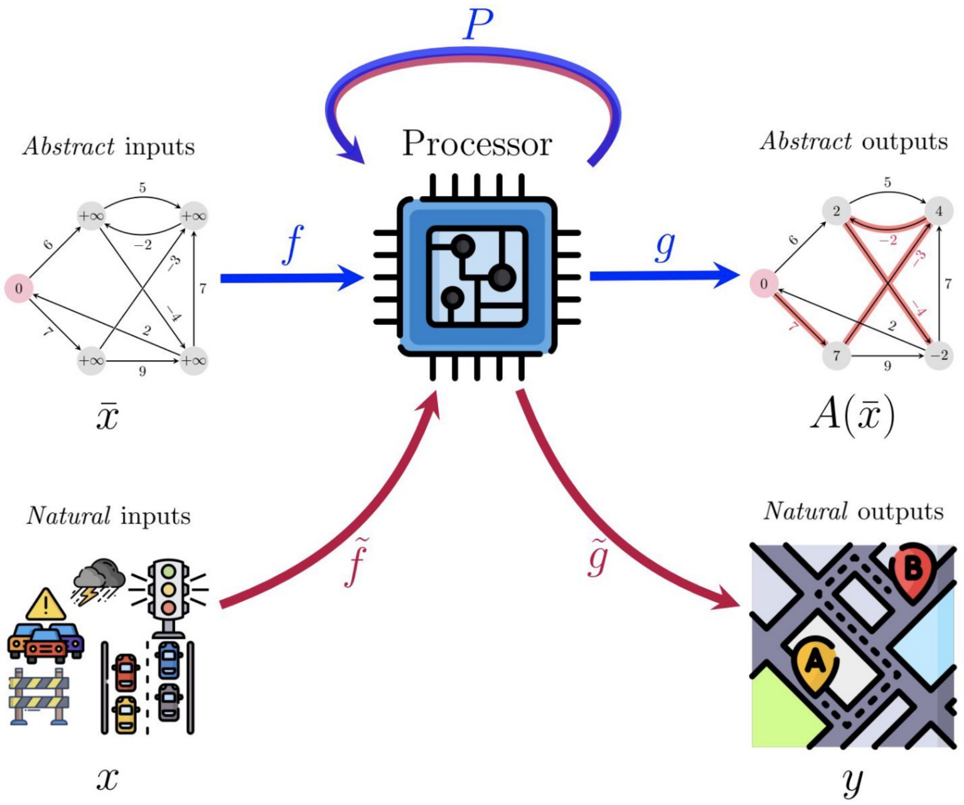


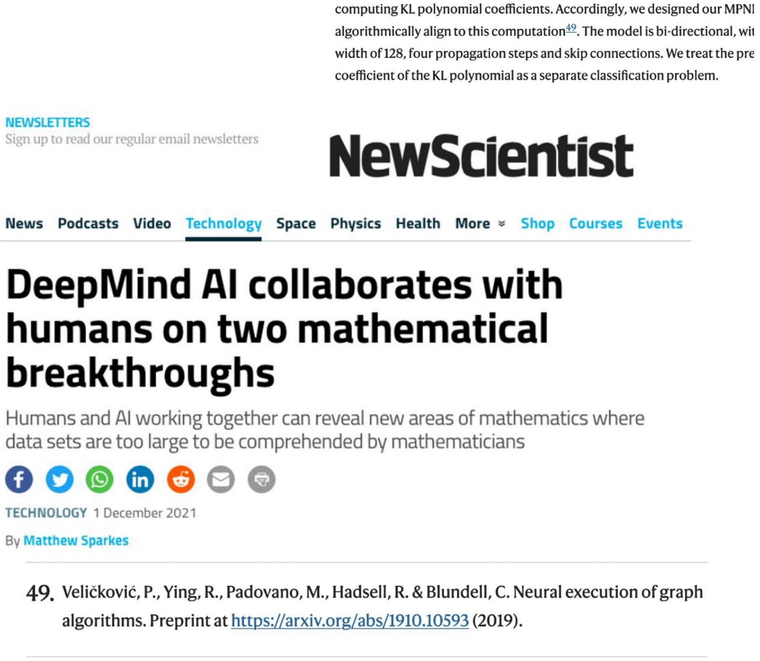
Prediction of 3D folding structures of proteins

In 2021 triumph of Geometric ML and a paradigm shift in structural biology
$\Rightarrow$ Breakthrough in prediction of the 3D folding structure of a protein by AlphaFold 2 (deepmind)



Learn Neural Nets Algorithmic and Mathematics!




computing KL polynomial coefficients. Accordingly, we designed our MPNN to algorithmically align to this computation[49]. The model is bi-directional, with a hidden layer width of 128, four propagation steps and skip connections. We treat the prediction of each coefficient of the KL polynomial as a separate classification problem.

NEWSLETTERS
Sign up to read our regular email newsletters

NewScientist

News Podcasts Video Technology Space Physics Health More Shop Courses Events

DeepMind AI collaborates with humans on two mathematical breakthroughs

Humans and AI working together can reveal new areas of mathematics where data sets are too large to be comprehended by mathematicians

TECHNOLOGY 1 December 2021

By Matthew Sparkes

49. Veličković, P., Ying, R., Padovano, M., Hadsell, R. & Blundell, C. Neural execution of graph algorithms. Preprint at https://arxiv.org/abs/1910.10593 (2019).

Retrieve fundamentals physic laws ?

arXiv > cs > arXiv:2005.07724

Computer Science > Machine Learning

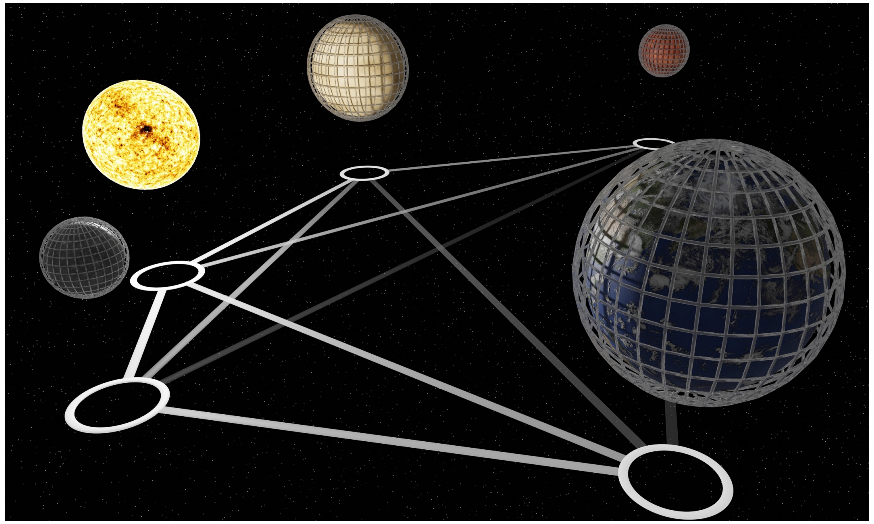
[Submitted on 15 May 2020]

Learning the gravitational force law and other analytic functions

Atish Agarwala, Abhimanyu Das, Rina Panigrahy, Qiuyi Zhang

Large neural network models have been successful in learning functions of importance in many branches of science, incluc
wide networks and kernel methods on some simple classes of functions, but not on more complex functions which arise ir
sphere for any kernel method or equivalent infinitely-wide network with the corresponding activation function trained witl
number of samples proportional to the derivative of a related function. Many functions important in the sciences are there
gravitational force function given by Newton's law of gravitation. Our theoretical bounds suggest that very wide ReLU netw
kernel learning with Gaussian kernels. We present experimental evidence that the many-body gravitational force function i

Subjects: **Machine Learning (cs.LG)**; Machine Learning (stat.ML)
Cite as: arXiv:2005.07724 **[cs.LG]**
(or arXiv:2005.07724v1 **[cs.LG]** for this version)
https://doi.org/10.48550/arXiv.2005.07724

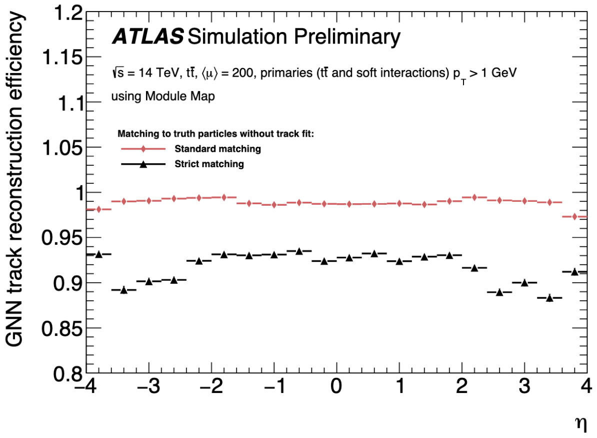


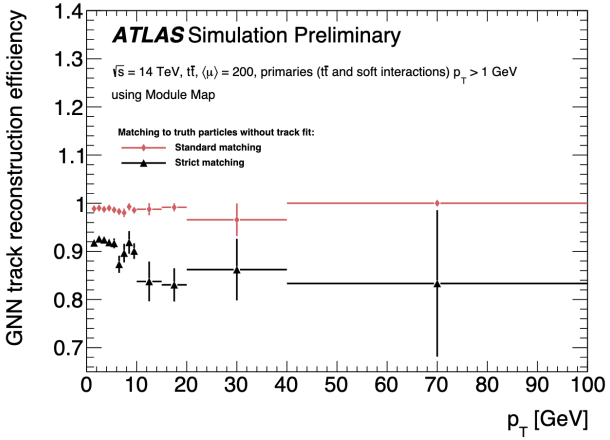
Soon theorical physicist useless ?? ☺

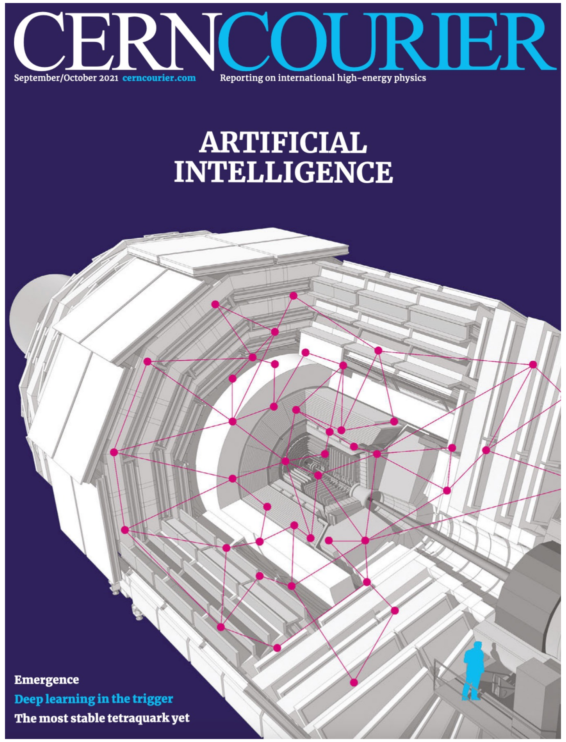
Graph-based ML for HEP @ CERN LHC

Since 2020 becoming increasingly popular for a large number of LHC physics tasks

⇒Collaboration L2IT ATLAS team & ExatTrkX Project to construct a GNN-based track reconstruction algorithm for ATLAS ITk (futur Inner Tracker of ATLAS for HL-LHC)







September/October 2021

⇒GNN-based algorithms now appear as a very competitive solution for the next generation track reconstruction algorithms

⇒Now working to integrate these GNN-based algorithms in production for in-line and off-line data processing systems

reference

- Deep Learning course
  - [MIT Introduction to Deep Learning | 6.S191]
  - [MIT OpenCourseWare : Course Introduction of 18.065 by Professor Strang]
  - [FIDLE Formation (videos in french, slides and supports in english)] FIDLE

- pytorch tutorial
  - [Learn pytorch from examples]
  - [Learn PyTorch for Deep Learning: Zero to Mastery book]