# Machine learning applied to theoretical high-energy physics

Stefano Carrazza

3 April 2019, ICTP-SAIFR, São Paulo

Università degli Studi di Milano (UNIMI and INFN Milan)

Acknowledgement: This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement no. 740006.


# Why talk about machine learning?

## Why talk about machine learning?

*because*

- it is an essential set of algorithms for building models in science,
- fast development of new tools and algorithms in the past years,
- nowadays it is a requirement in experimental and theoretical physics,
- large interest from the HEP community: *IML*, *conferences*, *grants*.

## When apply machine learning in theoretical physics?

## When apply machine learning in theoretical physics?

*at least in two situations:*

- Ambiguous choices.
- Lack of information.

# Outline

## Part I

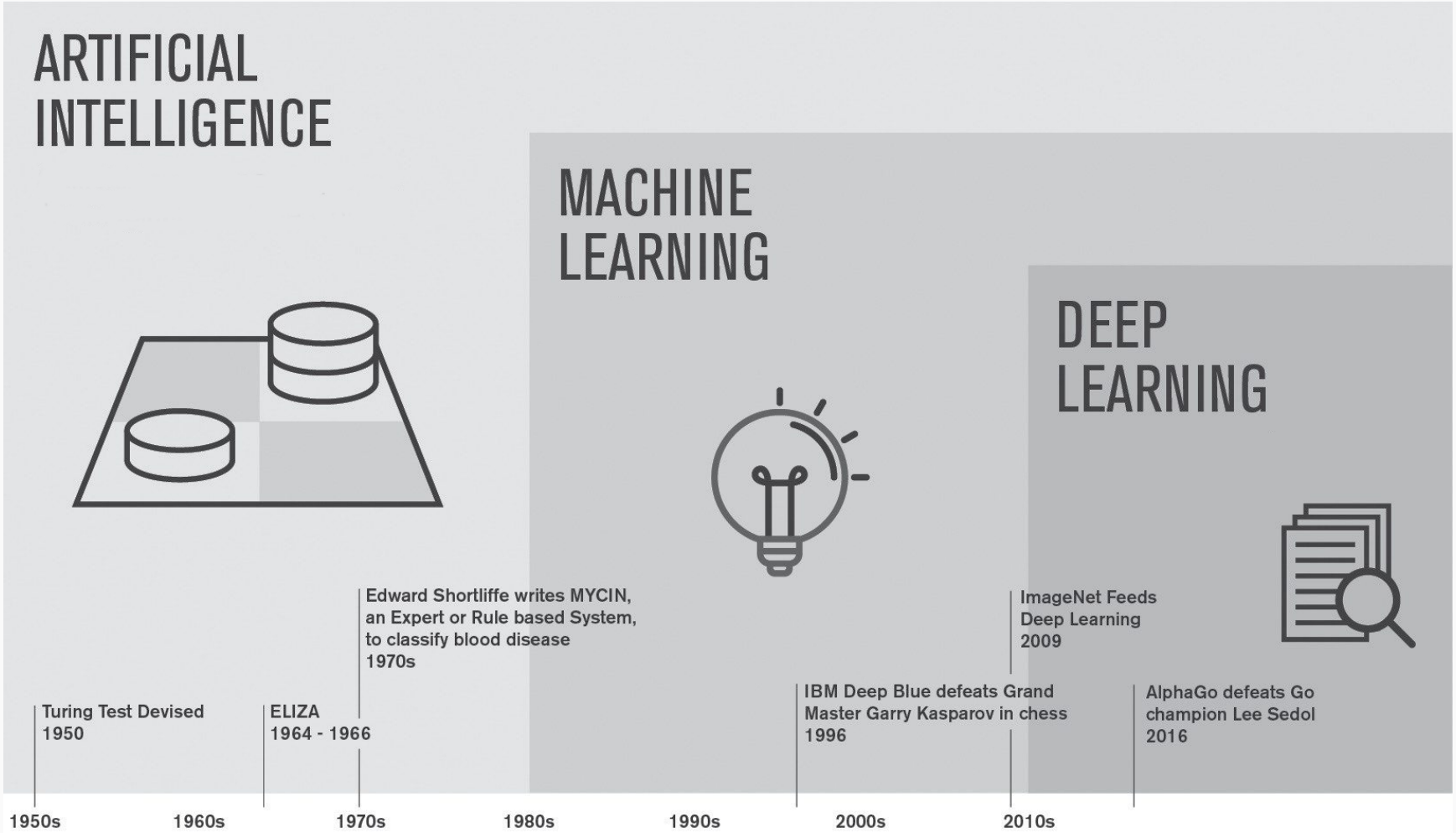
- A.I. and M.L. overview
- Non-linear models
- Learning basics

## Part II

- Theoretical physics inspiring ML models
- Examples in HEP-TH:
  - ML applied to parton model
  - ML in jet physics
  - ML examples in Monte Carlo simulation

# Artificial Intelligence

# Artificial intelligence timeline

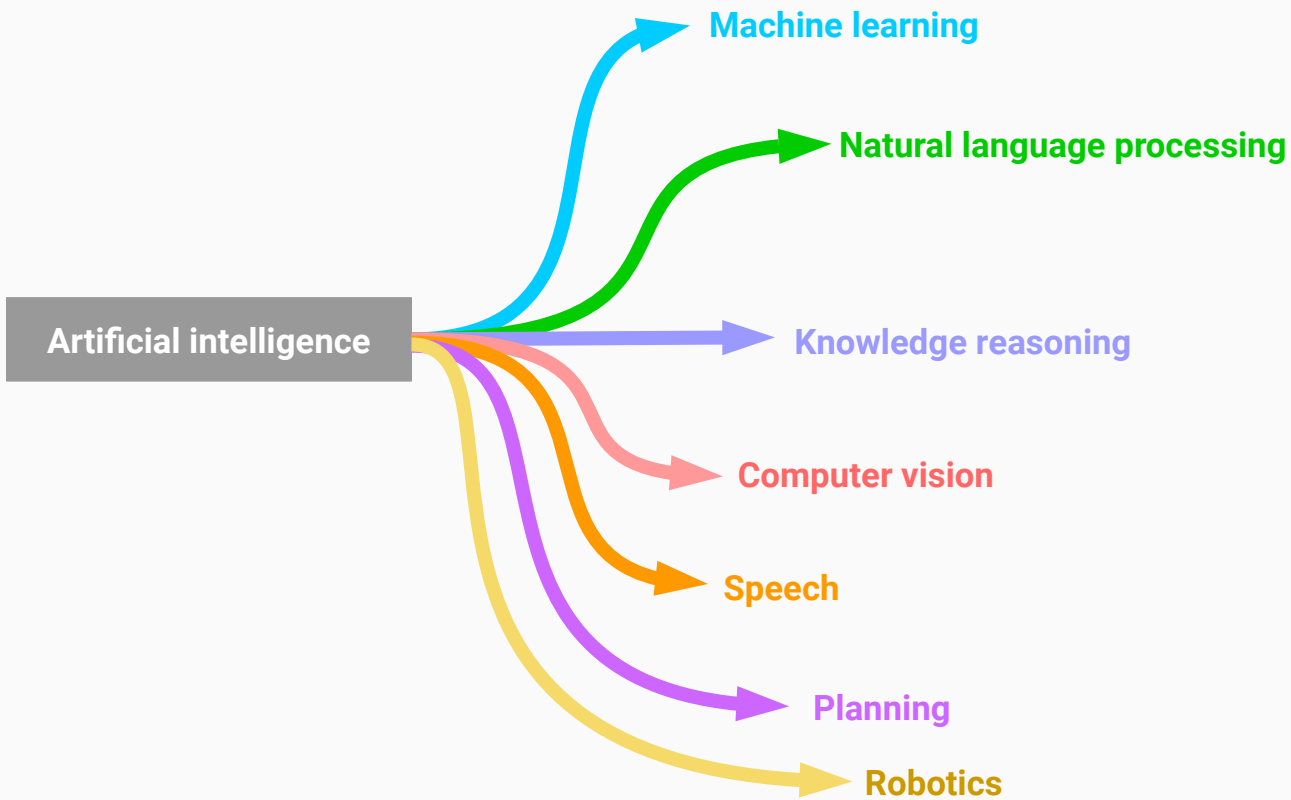


# Defining A.I.

**Artificial intelligence** (A.I.) is *the science and engineering of making intelligent machines*. (John McCarthy '56)

# Defining A.I.

**Artificial intelligence** (A.I.) is *the science and engineering of making intelligent machines*. (John McCarthy '56)

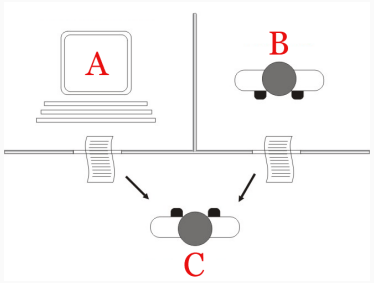



A.I. consist in the development of computer systems to perform tasks commonly associated with intelligence, such as *learning*.

# A.I. and humans

There are two categories of A.I. tasks:

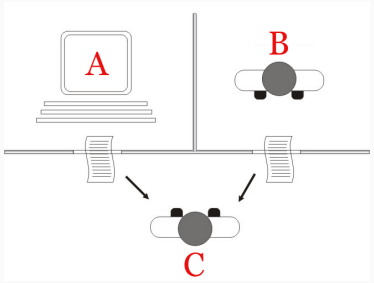
- **abstract and formal:** easy for computers but difficult for humans, *e.g.* play chess (IBM's Deep Blue 1997).
  $\rightarrow$ *Knowledge-based* approach to artificial intelligence.




# A.I. and humans

There are two categories of A.I. tasks:

- **abstract and formal:** easy for computers but difficult for humans, *e.g.* play chess (IBM's Deep Blue 1997).
  → *Knowledge-based* approach to artificial intelligence.




- **intuitive for humans but hard to describe formally:**
  *e.g.* recognizing faces in images or spoken words.
  → *Concept* capture and generalization

# A.I. technologies

Historically, the *knowledge-based* approach has not led to a major success with intuitive tasks for humans, because:

- requires human *supervision* and hard-coded *logical inference rules*.
- lacks of *representation learning* ability.

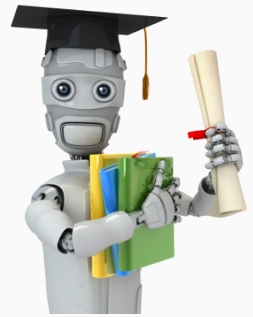
# A.I. technologies

Historically, the *knowledge-based* approach has not led to a major success with intuitive tasks for humans, because:

- requires human *supervision* and hard-coded *logical inference rules*.
- lacks of *representation learning* ability.

**Solution:**

The A.I. system needs to *acquire its own knowledge*.
This capability is known as **machine learning** (ML).
$\rightarrow$ *e.g.* write a program which learns the task.



# Machine Learning

# Machine learning definition

**Definition from A. Samuel in 1959:**
Field of study that gives computers the ability to learn without being explicitly programmed.

# Machine learning definition

**Definition from A. Samuel in 1959:**
Field of study that gives computers the ability to learn without being explicitly programmed.

**Definition from T. Mitchell in 1998:**
A computer program is said to *learn* from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance on $T$, as measured by $P$, improves with experience $E$.

# ML applications in our “day life”

# Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records

# Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)

# Machine learning examples

Thanks to work in A.I. and new capability for computers:

- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)

# Machine learning examples

Thanks to work in A.I. and new capability for computers:

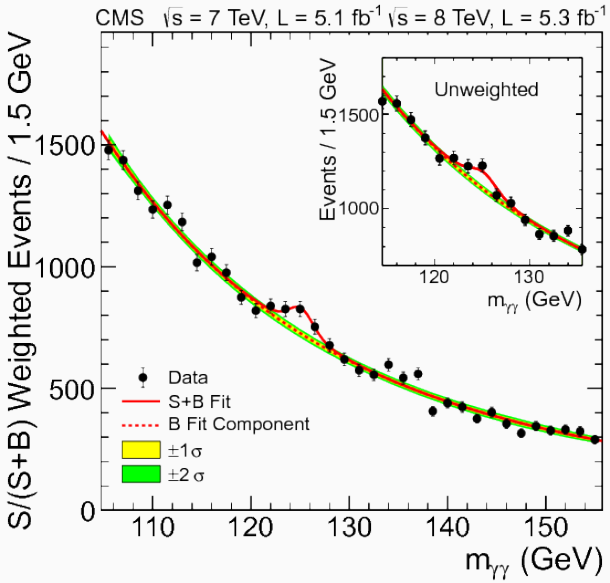
- **Database mining:**
  - Search engines
  - Spam filters
  - Medical and biological records
- **Intuitive tasks for humans:**
  - Autonomous driving
  - Natural language processing
  - Robotics (reinforcement learning)
  - Game playing (DQN algorithms)
- **Human learning:**
  - Concept/human recognition
  - Computer vision
  - Product recommendation

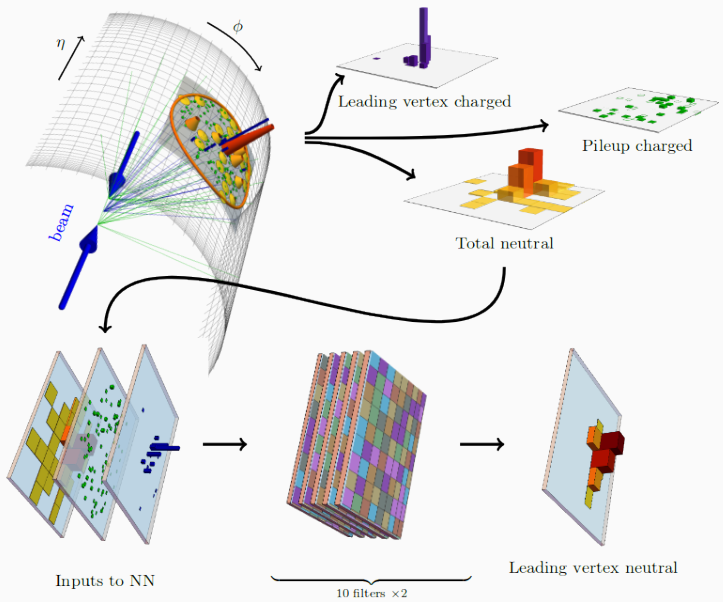
# ML applications in HEP

# ML in experimental HEP

Some remarkable examples are:

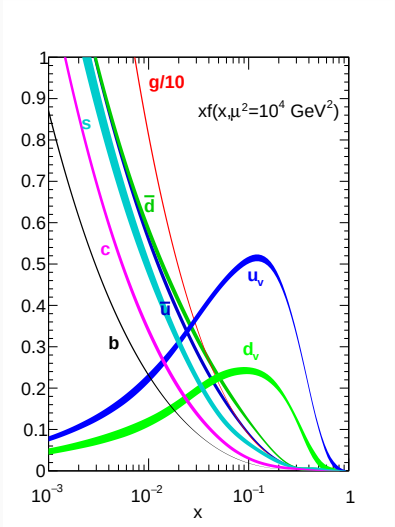
- **Signal-background detection:**
  Decision trees, artificial neural networks, support vector machines.
- **Jet discrimination:**
  Deep learning imaging techniques via convolutional neural networks.
- **HEP detector simulation:**
  Generative adversarial networks, *e.g.* LAGAN and CaloGAN.

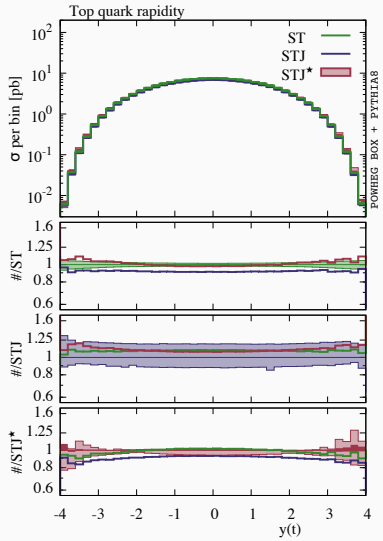


# ML in theoretical HEP

- **Supervised learning:**
  - The structure of the proton at the LHC
    - parton distribution functions
  - Theoretical prediction and combination
  - Monte Carlo reweighting techniques
    - neural network Sudakov
  - BSM searches and exclusion limits
- **Unsupervised learning:**
  - Clustering and compression
    - PDF4LHC15 recommendation
  - Density estimation and anomaly detection
    - Monte Carlo sampling

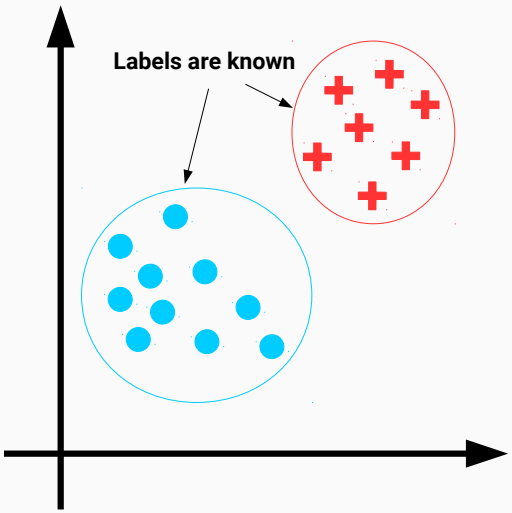


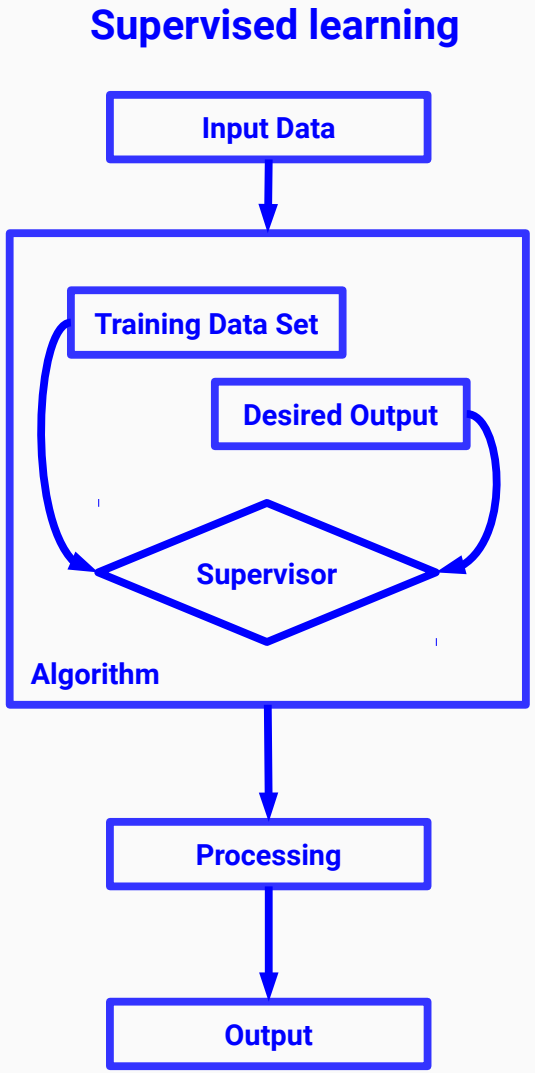



# Machine learning algorithms

## Machine learning algorithms:

- Supervised learning:
  regression, classification, ...

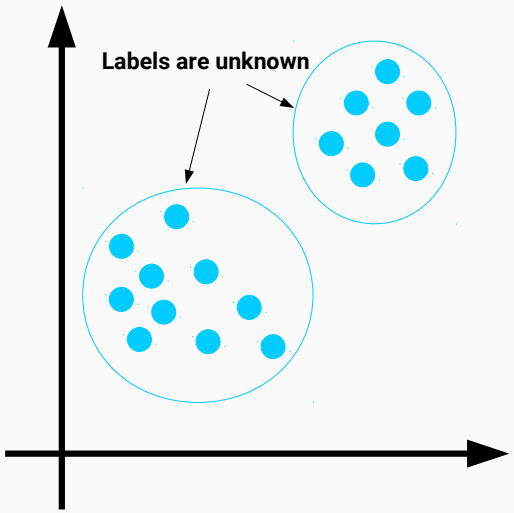


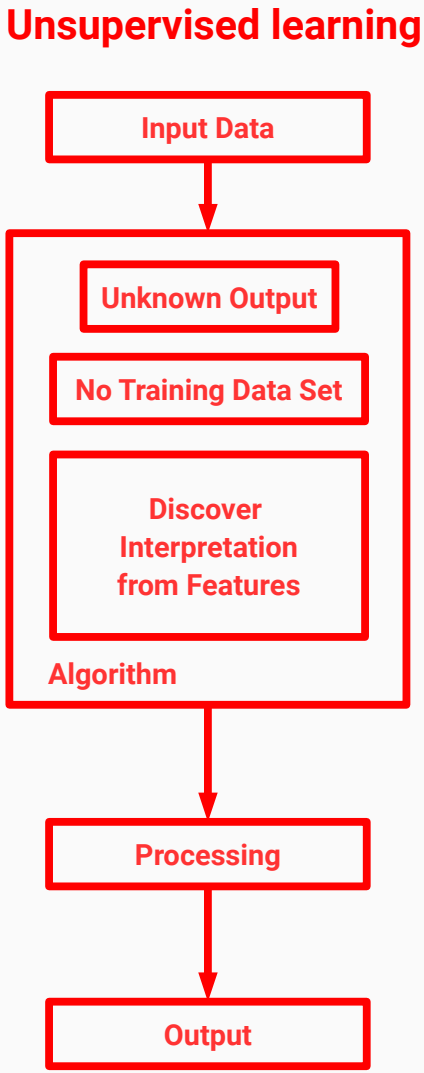



# Machine learning algorithms

**Machine learning algorithms:**

- Supervised learning:
  regression, classification, ...
- Unsupervised learning:
  clustering, dim-reduction, ...

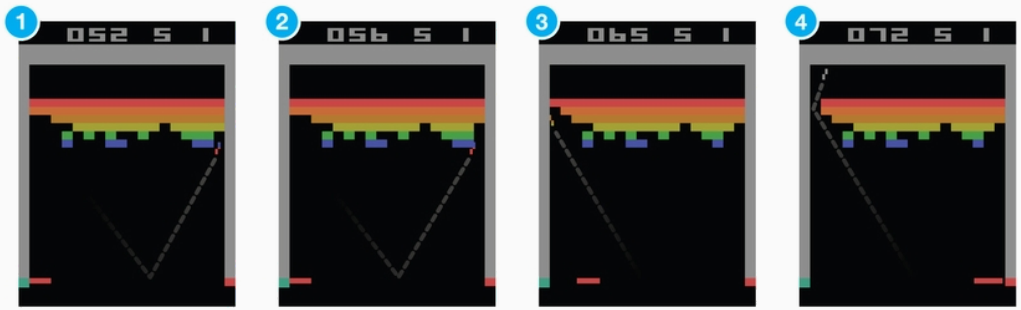


# Machine learning algorithms

**Machine learning algorithms:**

- Supervised learning:
  regression, classification, ...
- Unsupervised learning:
  clustering, dim-reduction, ...
- Reinforcement learning:
  real-time decisions, ...



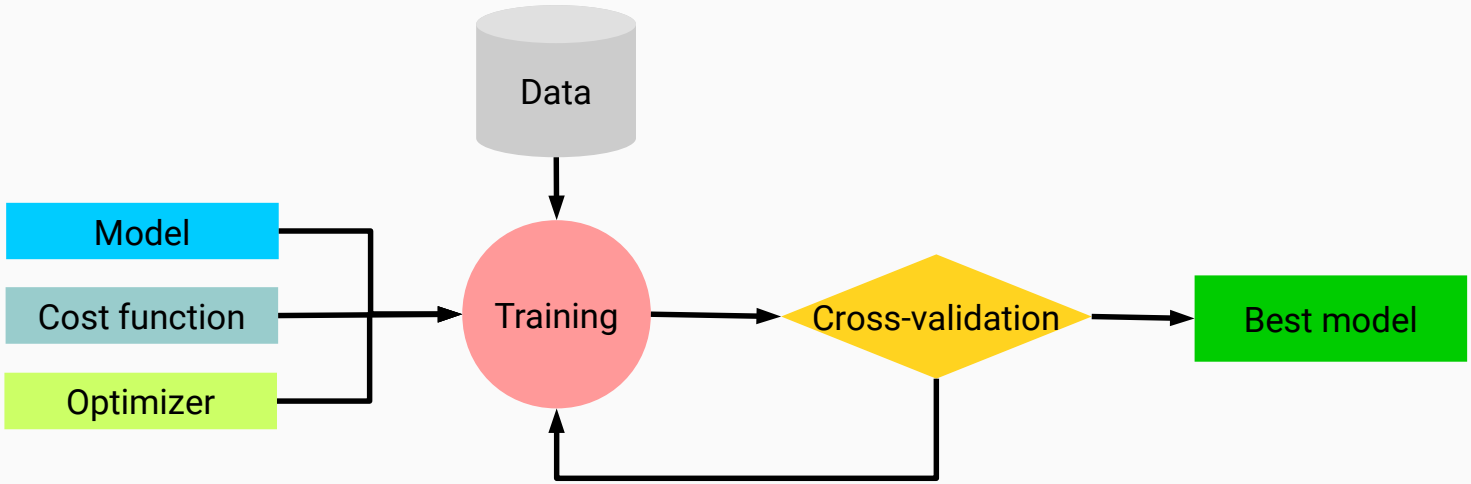

# Machine learning algorithms


More than 60 algorithms.

# Models and metrics

# Workflow in machine learning

The operative workflow in ML is summarized by the following steps:




The best model is then used to:

- supervised learning: make predictions for new observed data.
- unsupervised learning: extract features from the input data.

# Model representation in supervised learning

**Examples of models:**

$\rightarrow$ Linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and $b$ a constant.

# Model representation in supervised learning

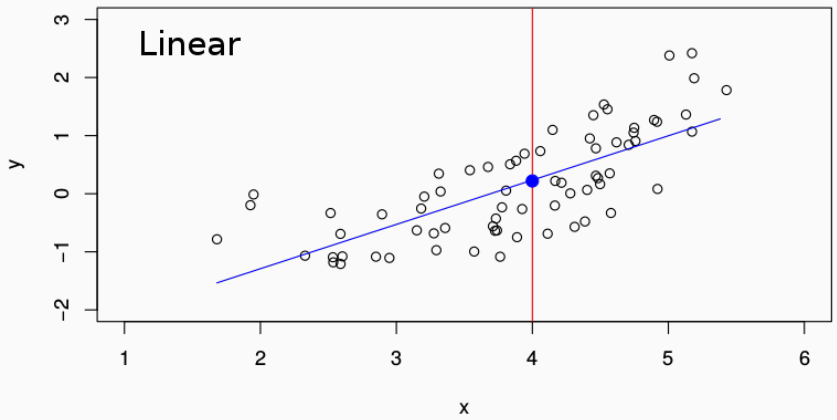
**Examples of models:**

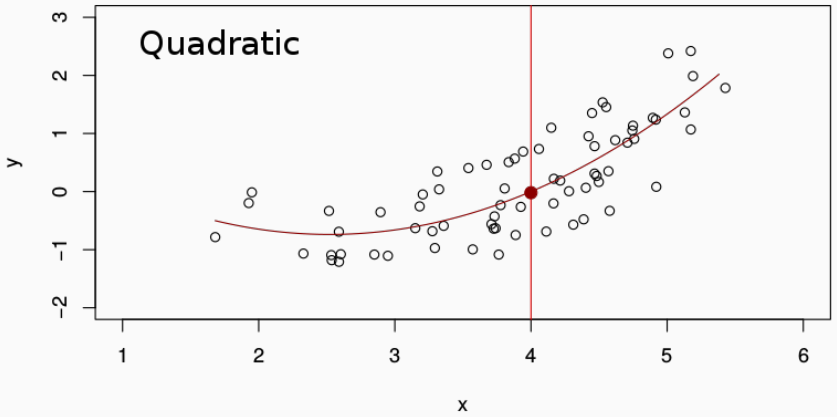
$\rightarrow$ Linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and $b$ a constant.

$\rightarrow$ Generalized linear models are also available increasing the power of linear models:



# Model representation in supervised learning

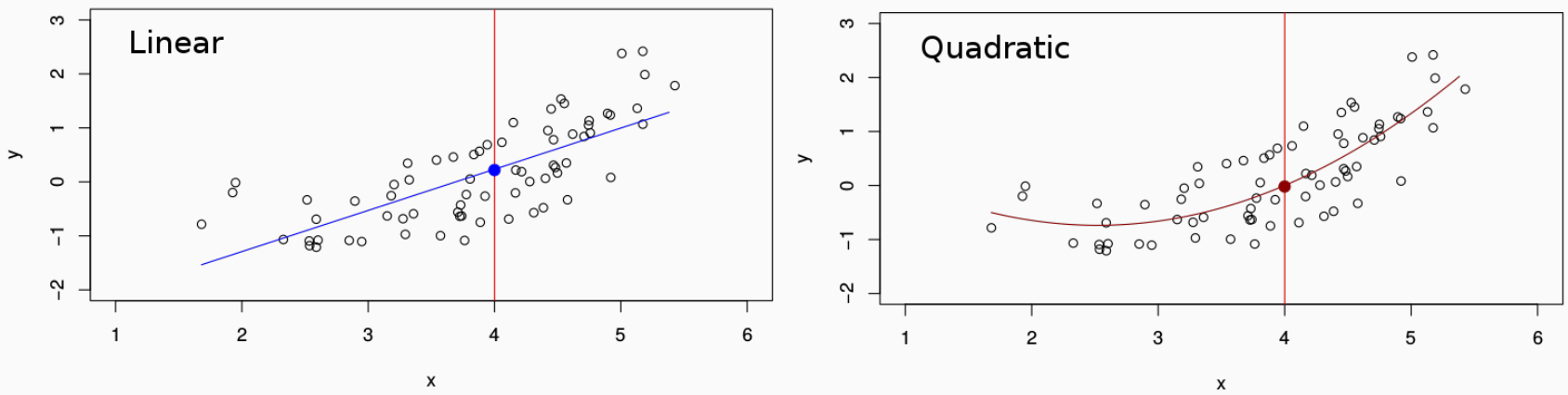
**Examples of models:**

$\rightarrow$ Linear regression we define a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output:

$$\hat{y}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters and $b$ a constant.

$\rightarrow$ Generalized linear models are also available increasing the power of linear models:

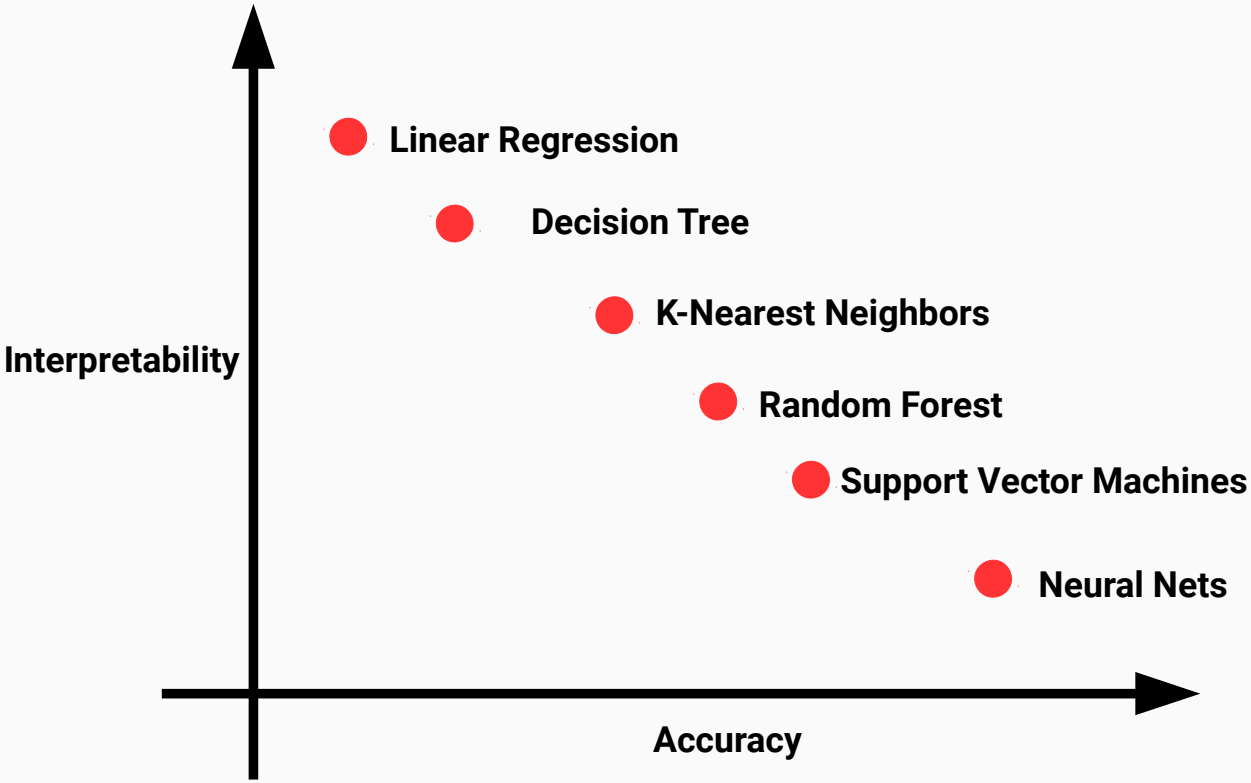



$\rightarrow$ Non-linear models: neural networks (talk later).

# Model representation trade-offs

However, the selection of the appropriate model comes with trade-offs:

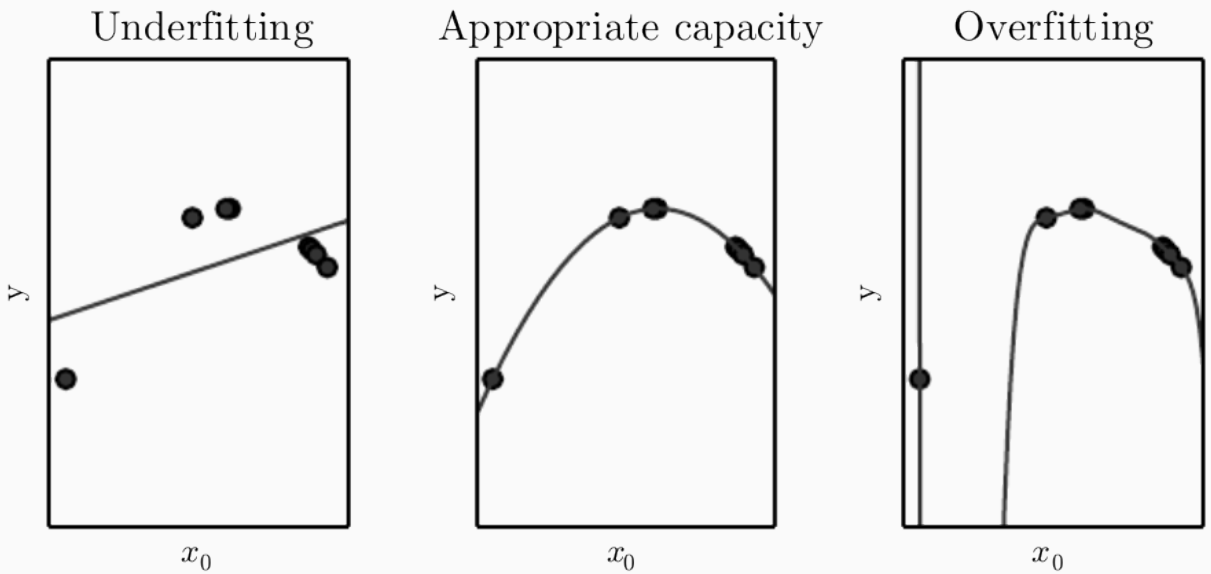
- **Prediction accuracy vs interpretability**:
  $\rightarrow$ *e.g.* linear model vs splines or neural networks.



# Model representation trade-offs

However, the selection of the appropriate model comes with trade-offs:

- **Prediction accuracy vs interpretability**:
  $\rightarrow$ *e.g.* linear model vs splines or neural networks.
- **Optimal capacity/flexibility:** number of parameters, architecture
  $\rightarrow$ deal with overfitting, and underfitting situations



# Assessing the model performance

**How to check model performance?**

$\rightarrow$ define metrics and statistical estimators for model performance.

**Examples:**

- Regression: cost / loss / error function,
- Classification: cost function, precision, accuracy, recall, ROC, AUC

# Assessing the model performance - cost function

To access the model performance we define a cost function $J(\boldsymbol{w})$ which often measures the difference between the target and the model output.

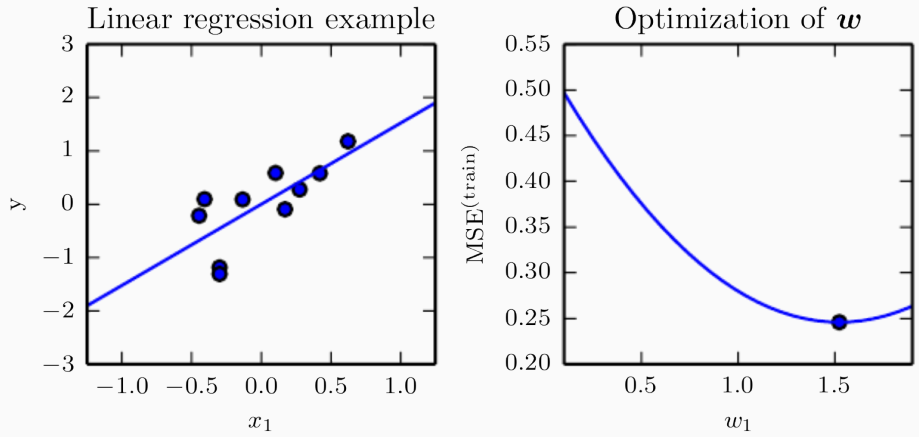
In a optimization procedure, given a model $\hat{y}_{\boldsymbol{w}}$, we search for:

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$

The mean square error (MSE) is the most commonly used for regression:

$$J(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_{\boldsymbol{w}}(\boldsymbol{x}_i))^2$$

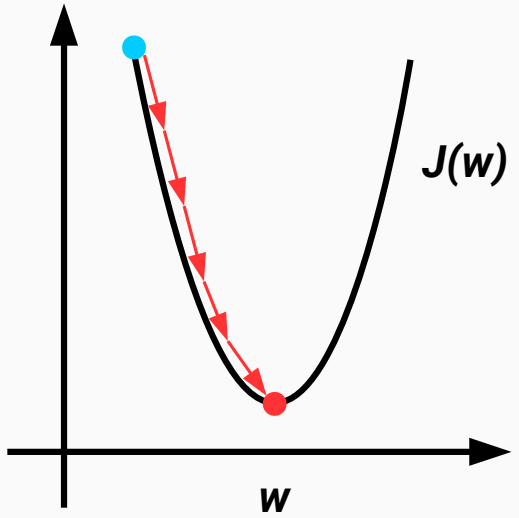
a quadratic function and convex function in linear regression.



# Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

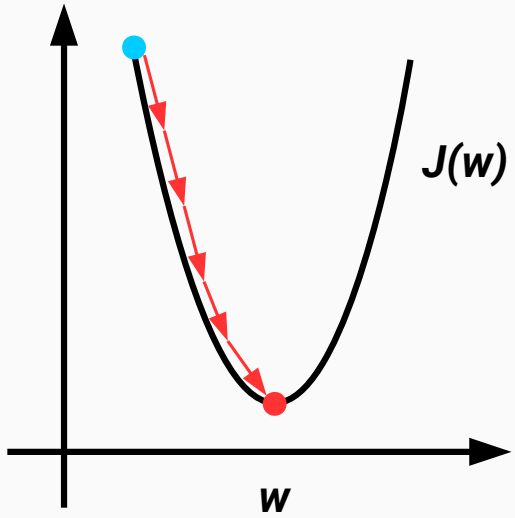
$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$



# Cost function minimization

Optimization algorithms **minimize an objective function**, $J(\boldsymbol{w})$, that depends on the model internal learnable parameters $\boldsymbol{w}$.

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$$




The most popular techniques are:

- normal equations (least squares)
- derivative based optimization
- evolutionary algorithms

The choice of a technique depends on the model and problem employed.

# Training and test sets

Another common issue related to model capacity in supervised learning:

- The model should not learn **noise** from data.
- The model should be able to **generalize** its output to new samples.

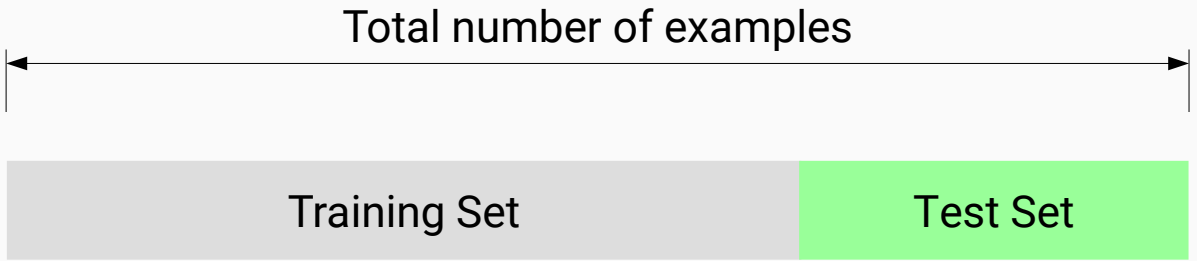
# Training and test sets

Another common issue related to model capacity in supervised learning:

- The model should not learn **noise** from data.
- The model should be able to **generalize** its output to new samples.

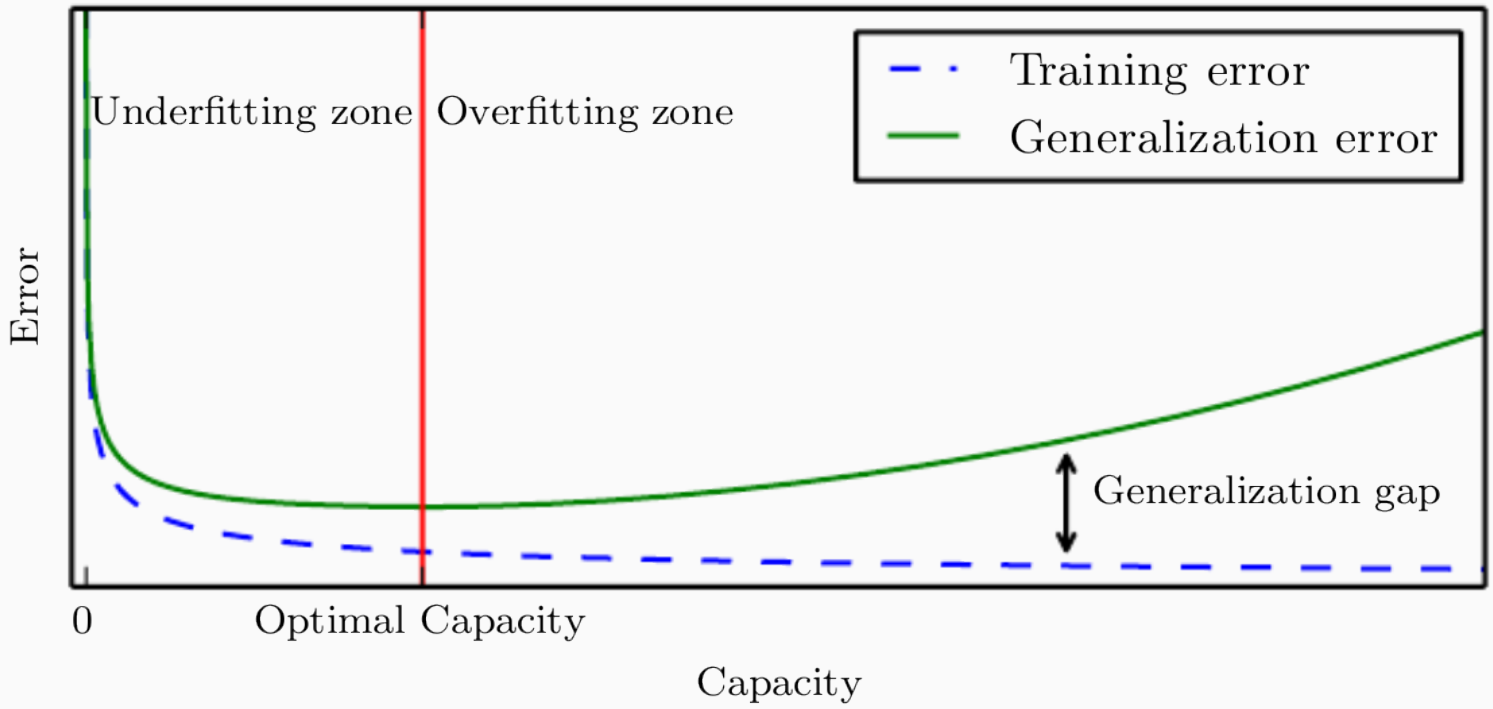
To observe this issue we split the input data in training and test sets:

- training set error, $J_{\text{Tr}}(\boldsymbol{w})$
- test set/generalization error, $J_{\text{Test}}(\boldsymbol{w})$



# Bias-variance trade-off

From a practical point of view dividing the input data in training and test:



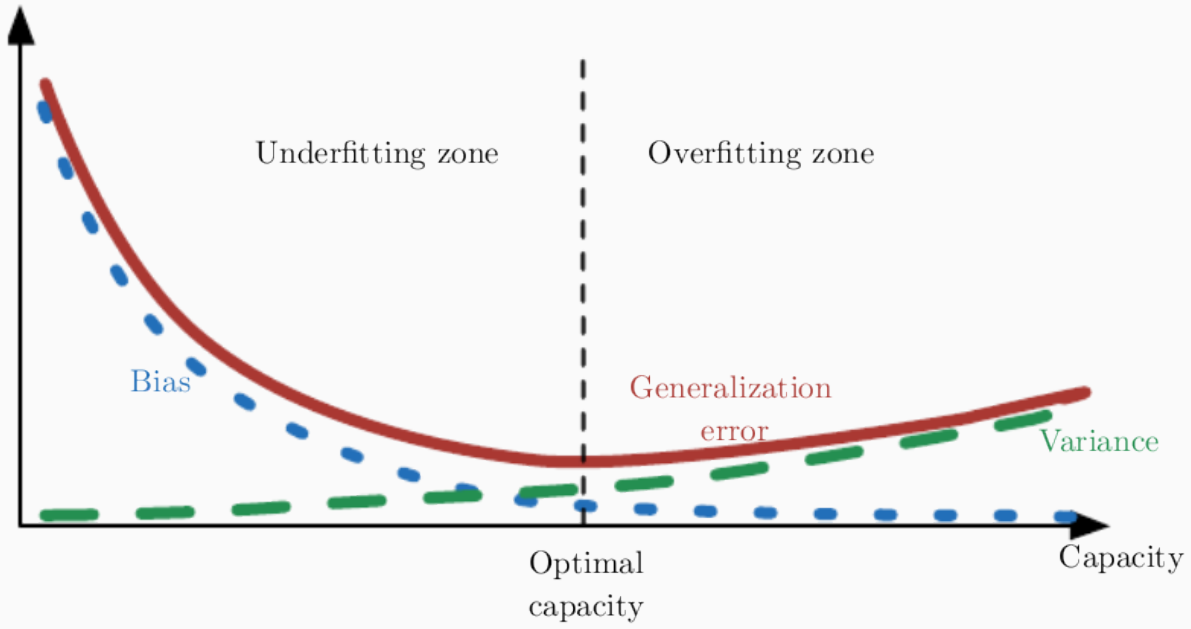

The training and test/generalization error conflict is known as bias-variance trade-off.

# Bias-variance trade-off

If $\hat{y}$ increases **flexibility**, its variance increases and its biases decreases.

Choosing the flexibility based on average test error amounts to a bias-variance trade-off:

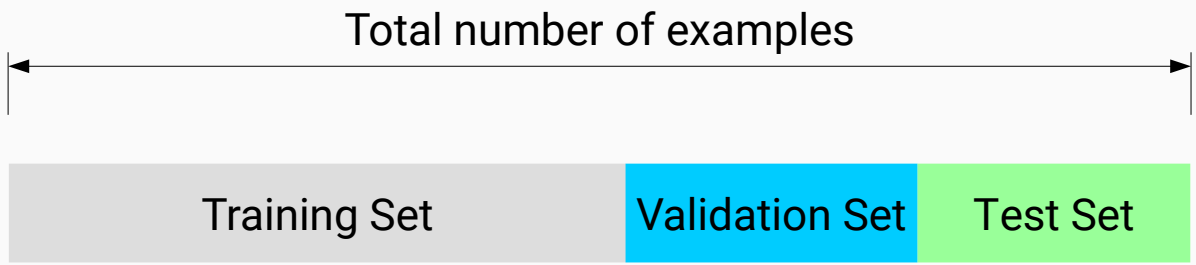
- **High Bias** $\rightarrow$ underfitting:
  erroneous assumptions in the learning algorithm.
- **High Variance** $\rightarrow$ overfitting:
  erroneous sensitivity to small fluctuations (noise) in the training set.



# Solution for the bias-variance trade off

A common way to reduce the bias-variance trade-off and choose the proper learning hyperparamters is to create a validation set that:

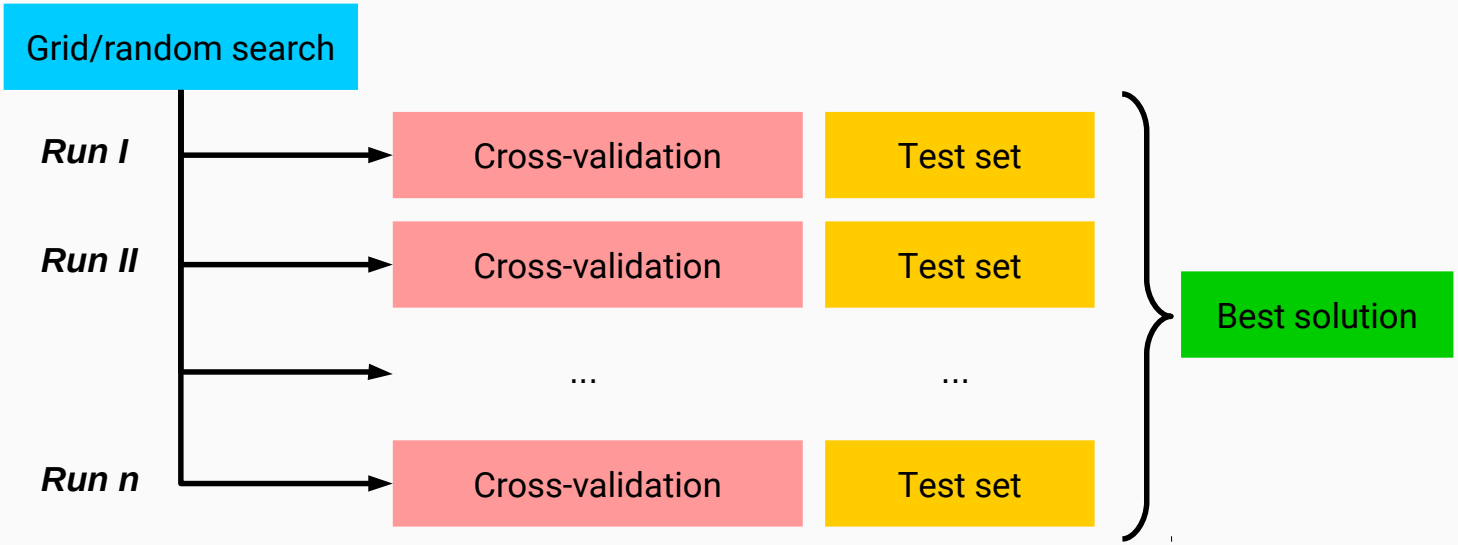
- not used by the training algorithm
- not used as test set




- **Training set**: examples used for learning.
- **Validation set**: examples used to tune the hyperparameters.
- **Test set**: examples used only to access the performance.

# ML in practice

Perform hyperparameter tune coupled to cross-validation:




Easy parallelization at search and cross-validation stages.

# Artificial neural networks

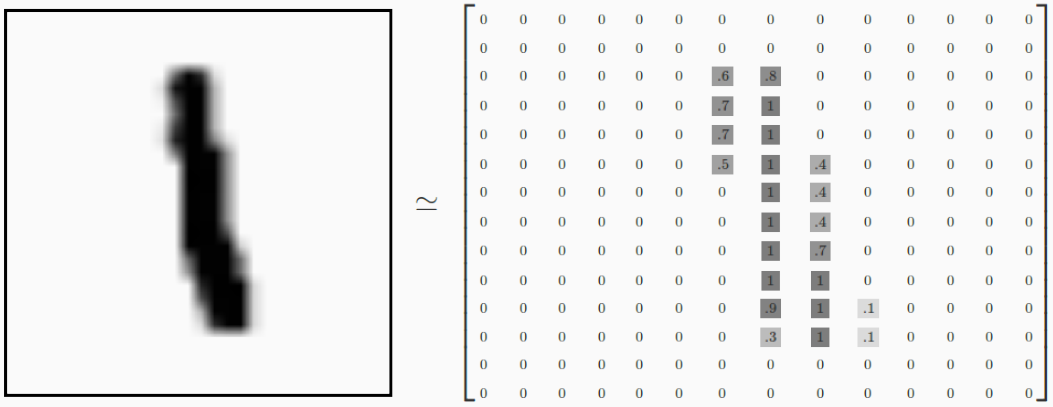
# Limitations of linear models

Why not linear models everywhere?

# Limitations of linear models

Why not linear models everywhere?

**Example:** consider 1 image from the MNIST database:



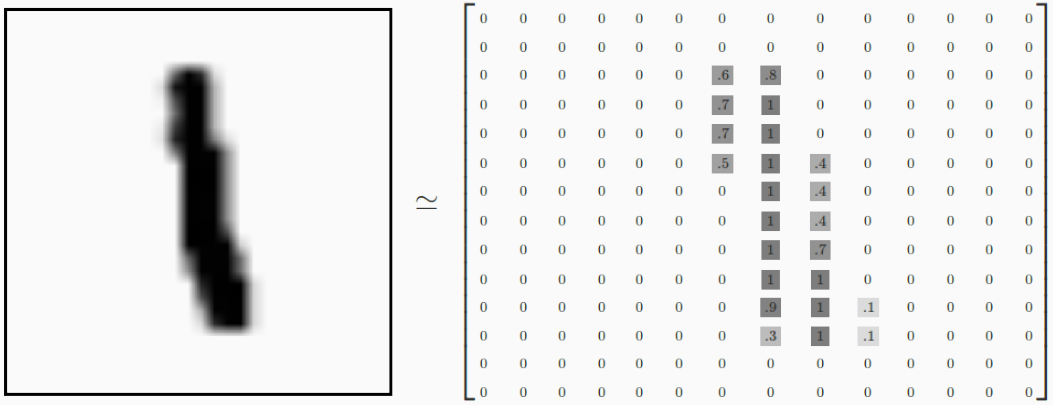
Each image has 28x28 pixels $= 785$ features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

# Limitations of linear models

Why not linear models everywhere?

**Example:** consider 1 image from the MNIST database:

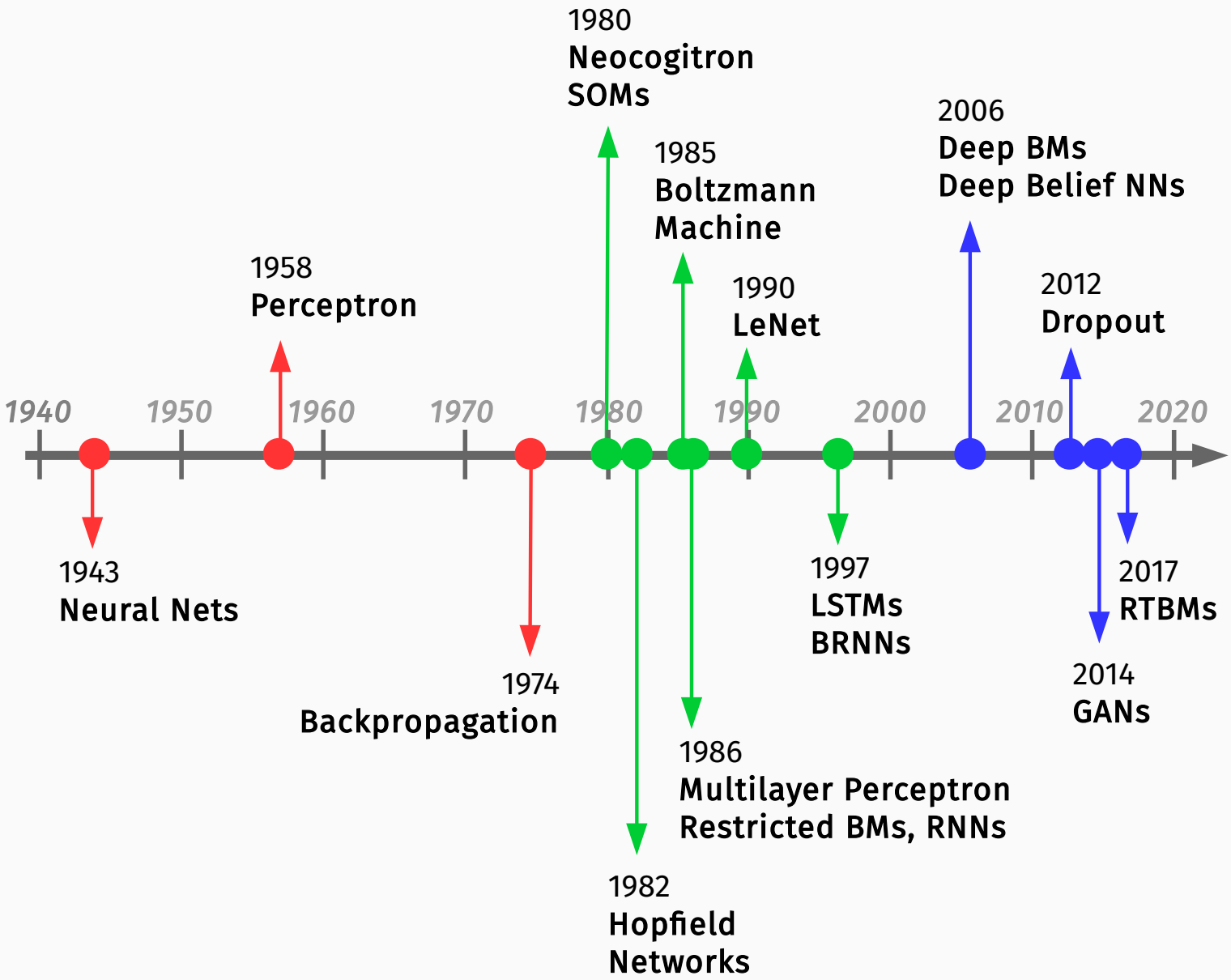


Each image has 28x28 pixels $= 785$ features (x3 if including RGB colors).

If consider quadratic function $\mathcal{O}(n^2)$ so linear models are impractical.

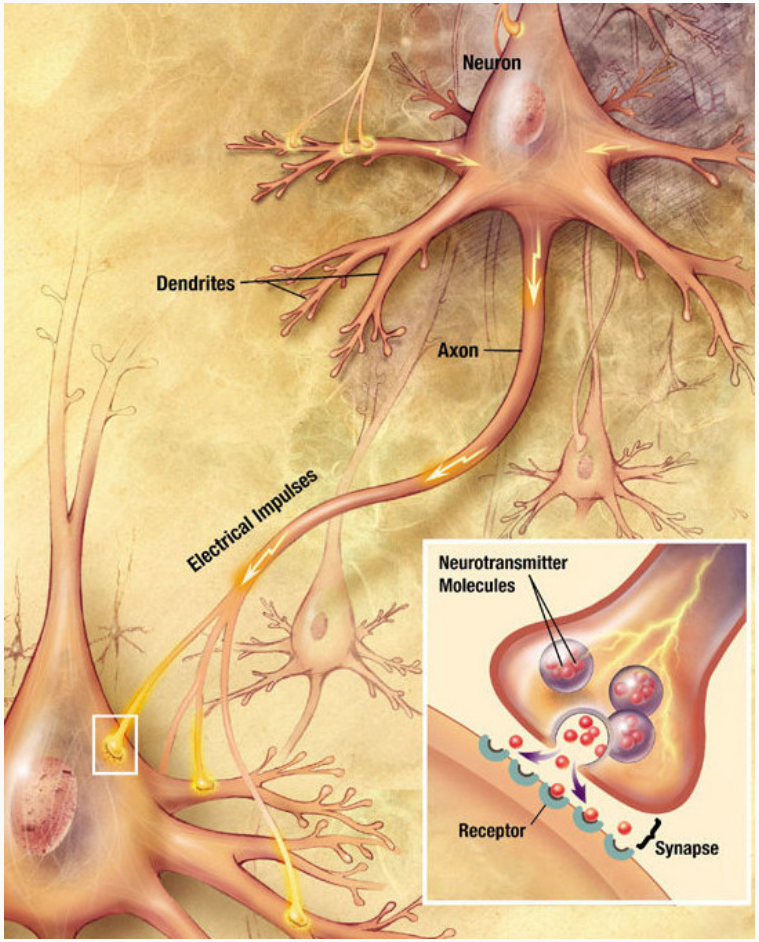
**Solution:** use non-linear models.

# Non-linear models timeline



# Neural networks

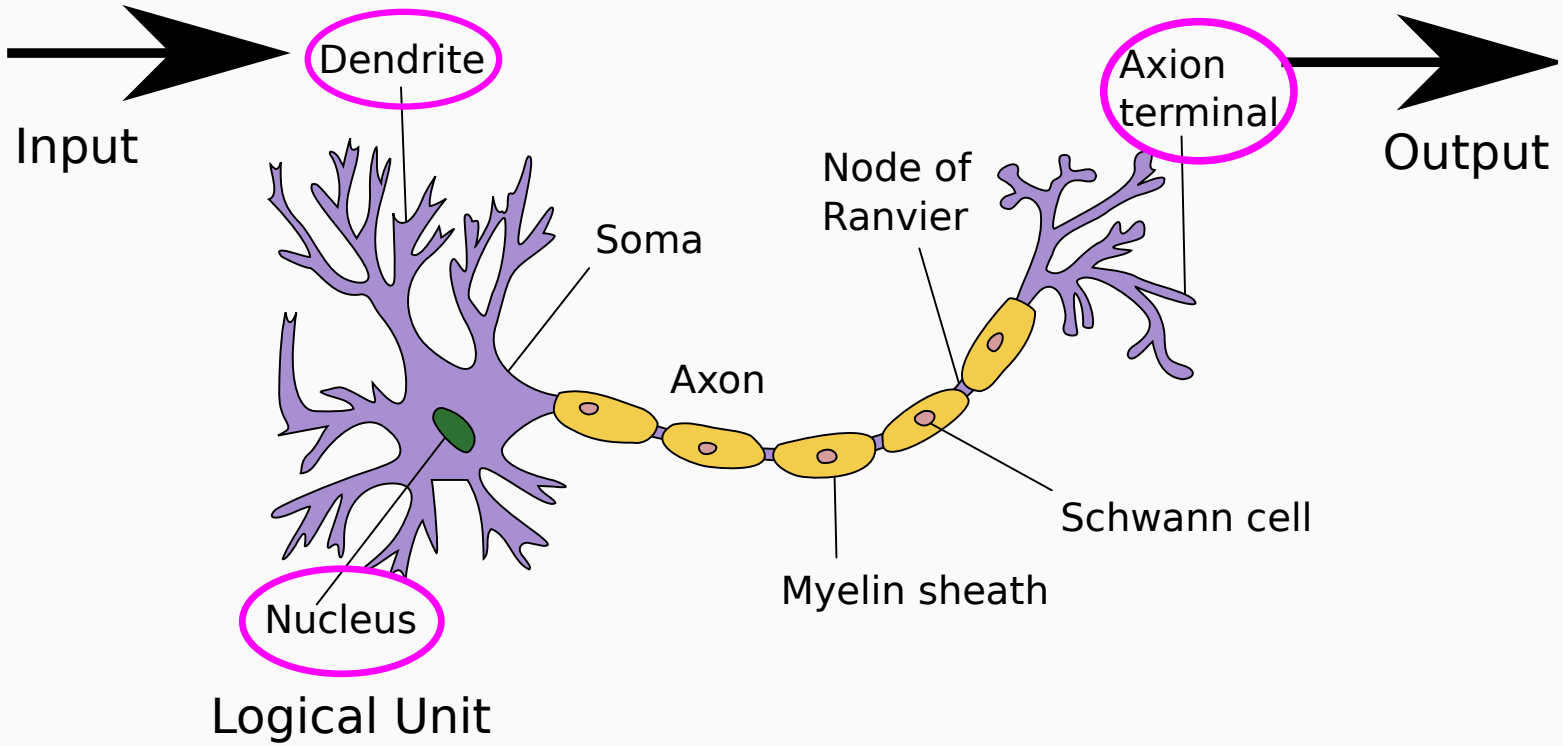
Artificial neural networks are computer systems inspired by the biological neural networks in the brain.




Currently the state-of-the-art technique for several ML applications.

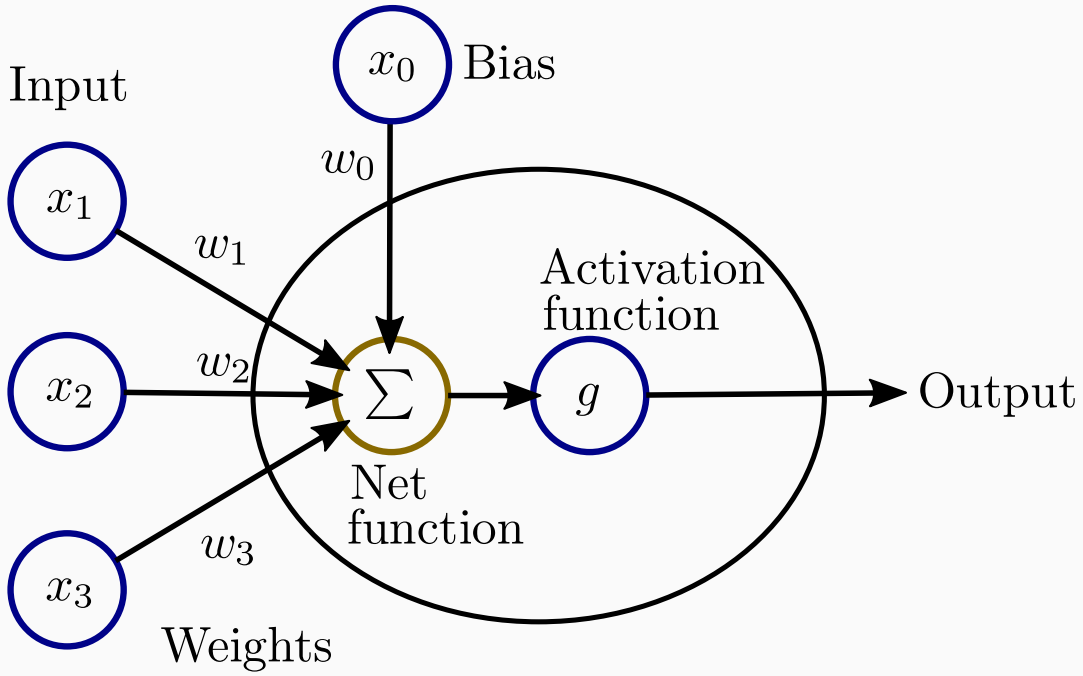
# Neuron model

We can imagine the following data communication pattern:



# Neuron model

Schematically:




where

- each **node** has an associate weights and bias $w$ and inputs $x$,
- the output is modulated by an **activation function**, $g$.

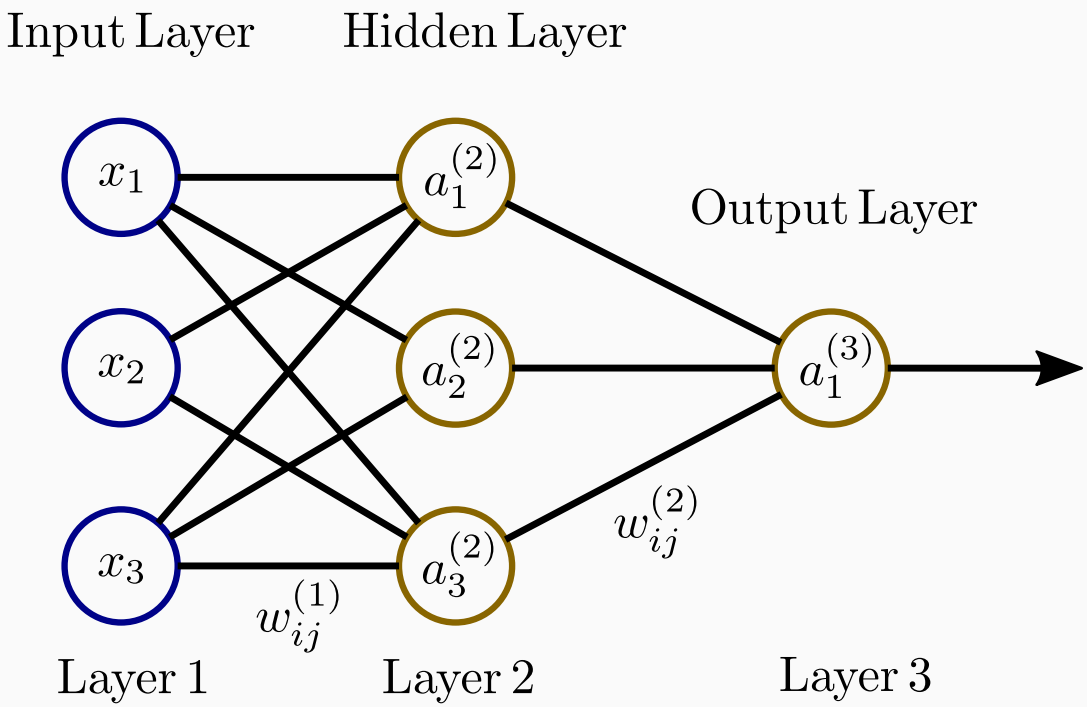
Some examples of activation functions: sigmoid, tanh, linear, ...

$$g_w(x) = \frac{1}{1+e^{-w^T x}}, \quad \tanh(w^T x), \quad x.$$

# Neural networks

In practice, we simplify the bias term with $x_0 = 1$.

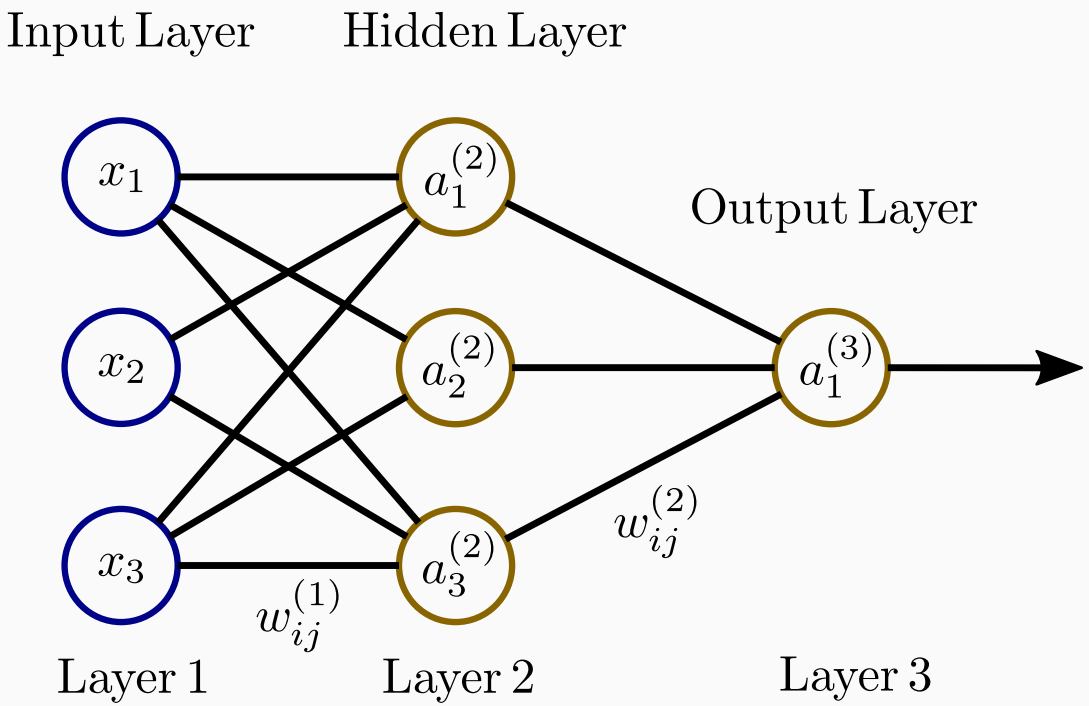
Neural network $\rightarrow$ connecting multiple units together.




where

- $a_i^{(l)}$ is the activation of unit $i$ in layer $l$,
- $w_{ij}^{(l)}$ is the weight between nodes $i, j$ from layers $l, l+1$ respectively.

# Neural networks

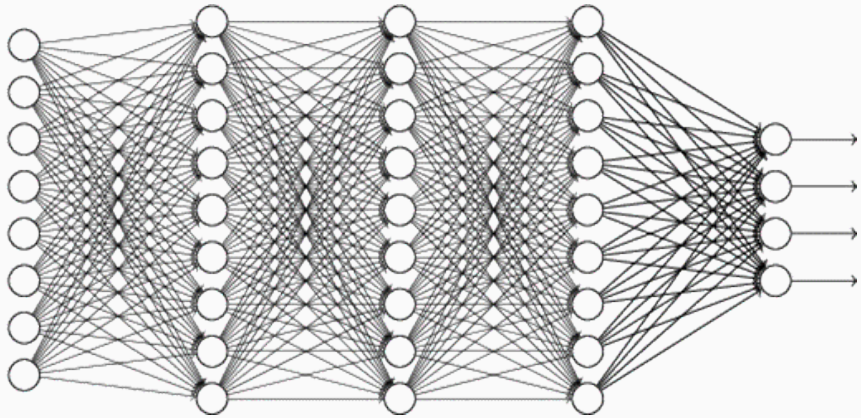



- $a_1^{(2)} = g(w_{10}^{(1)} + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3)$
- $a_2^{(2)} = g(w_{20}^{(1)} + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3)$
- $a_3^{(2)} = g(w_{30}^{(1)} + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3)$
- **Output** $\rightarrow a_1^{(3)} = g(w_{10}^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)})$

# Neural networks

Some useful names:

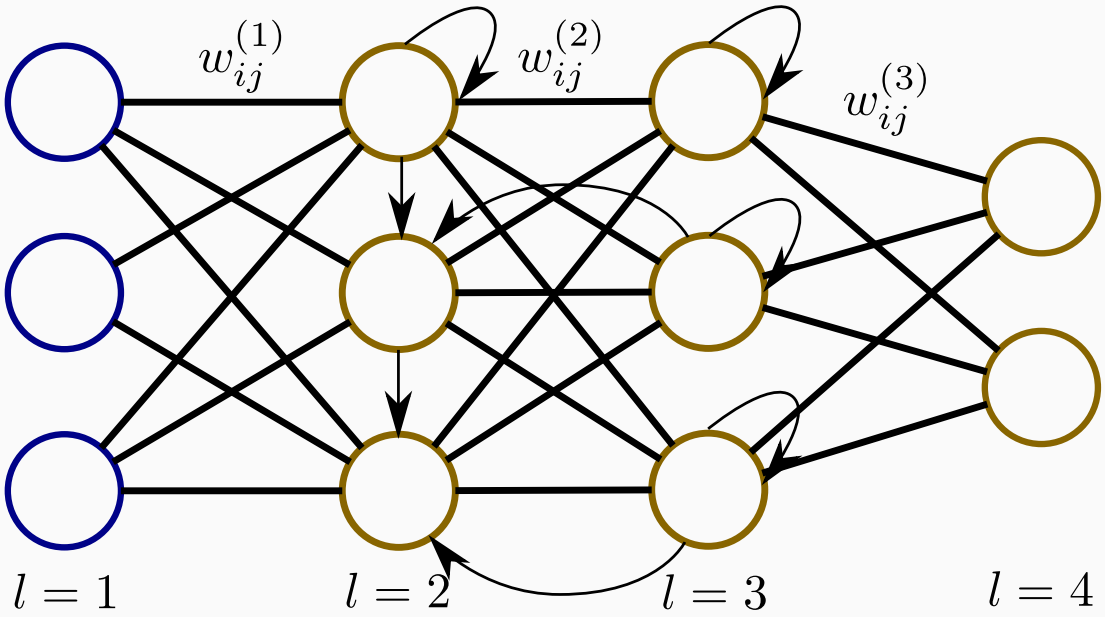
- Feedforward neural network: no cyclic connections between nodes from the same layer (previous example).
- Multilayer perceptron (MLP): is a feedforward neural network with at least 3 layers.
- Deep neural networks: term referring to neural networks with more than one hidden layer.



# Artificial neural networks architectures

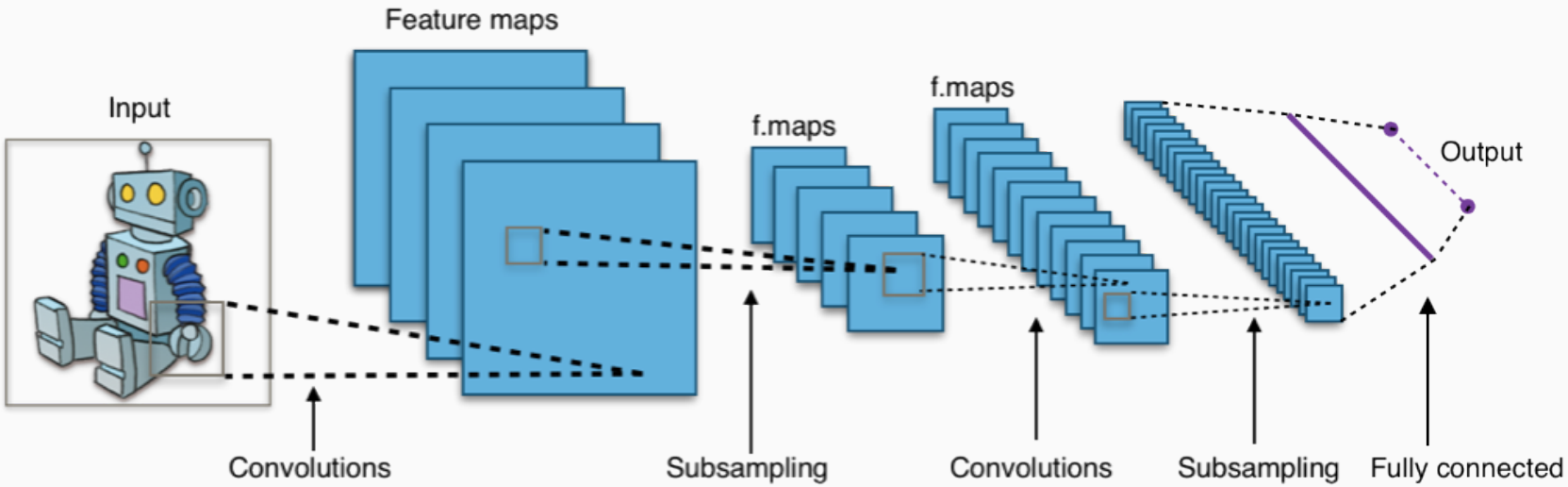
Some examples of neural network popular architectures:

- Recurrent neural networks: neural networks where connections between nodes form a directed cycle.
  - built-in internal state memory
  - built-in notion of time ordering for a time sequence



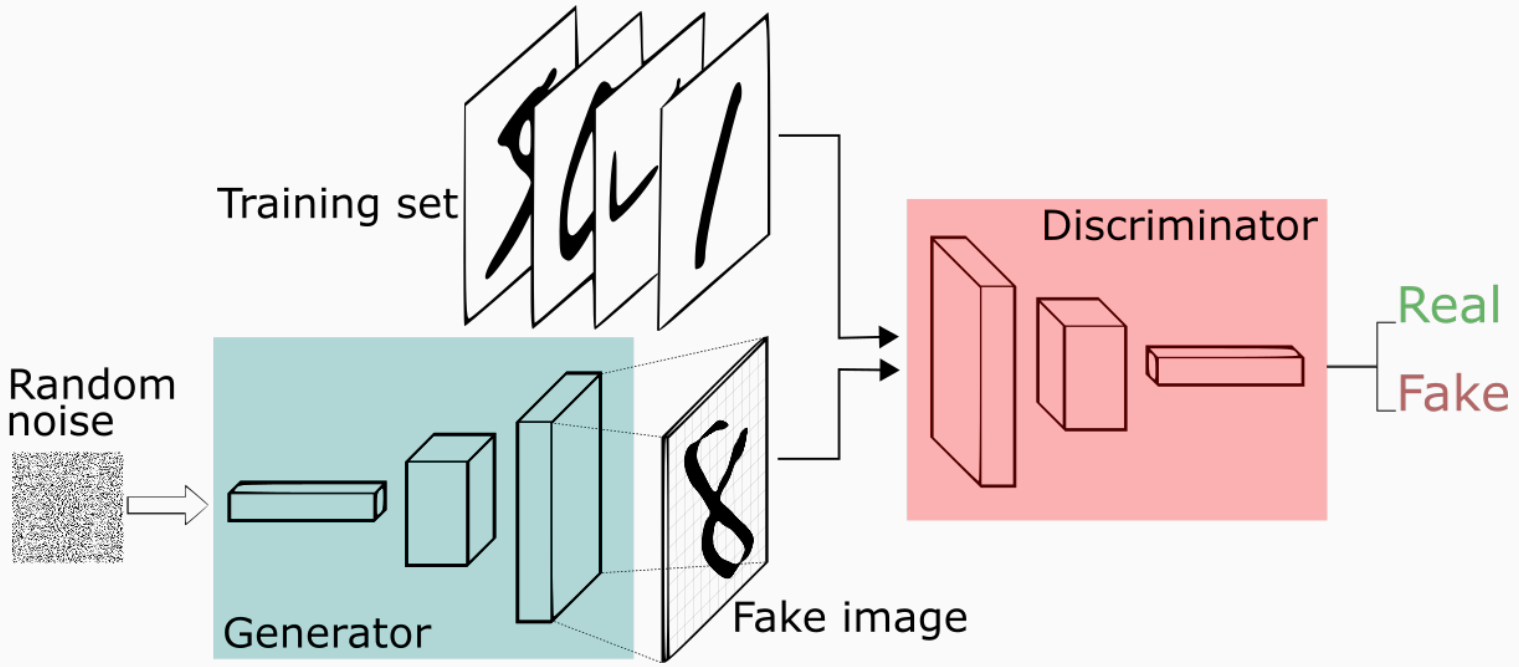
# Artificial neural networks architectures

- Convolutional neural networks: multilayer perceptron designed to require minimal preprocessing, *i.e.* space invariant architecture.
  - the hidden layers consist of convolutional layers, pooling layer, fully connected layers and normalization layers
  - great successful applications in image and video recognition.



# Artificial neural networks architectures

- Generative adversarial network: unsupervised machine learning system of two neural networks contesting with each other.
  - one network generate candidates while the other discriminates.



# Artificial neural networks architectures

Other popular examples:

- Recursive neural networks: a variation of recurrent neural network where pairs of layers or nodes are merged recursively.
  - successful applications on natural language processing.
  - some recent applications for model inference.
- Long short-term memory: another variation of recurrent neural networks composed by custom units cells:
  - LSTM cells have an input gate, an output gate and a forget gate.
  - powerful when making predictions based on time series data.
- Boltzmann Machines: is a generative stochastic recursive artificial neural network.
  - comes with energy-based model features and advantages.

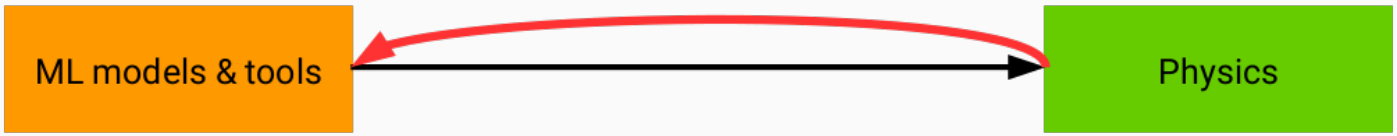
# Theoretical physics inspiring ML

# Introduction

We started this project aiming to build a model with:

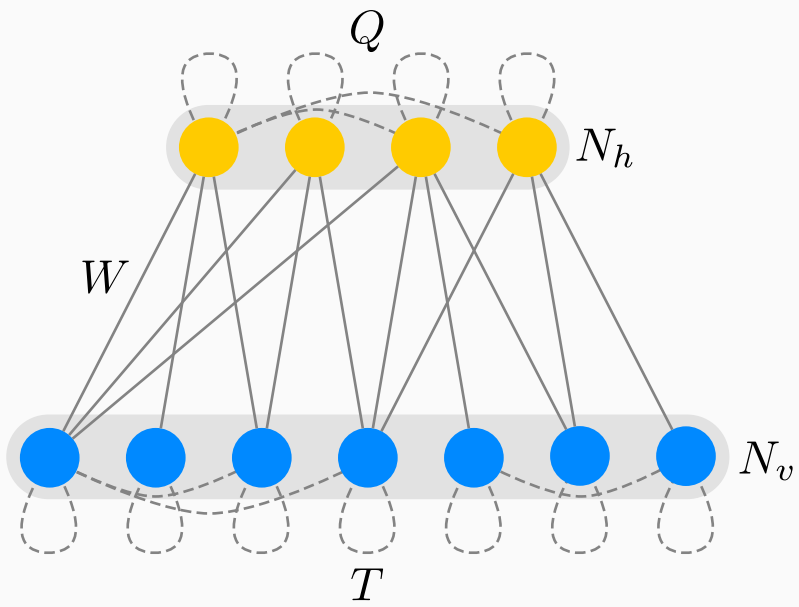
- well suited for pdf estimation and pdf sampling
- built-in pdf normalization (close form expression)
- very flexible with a small number of parameters

We decided to look at energy models, specifically Boltzmann Machines.



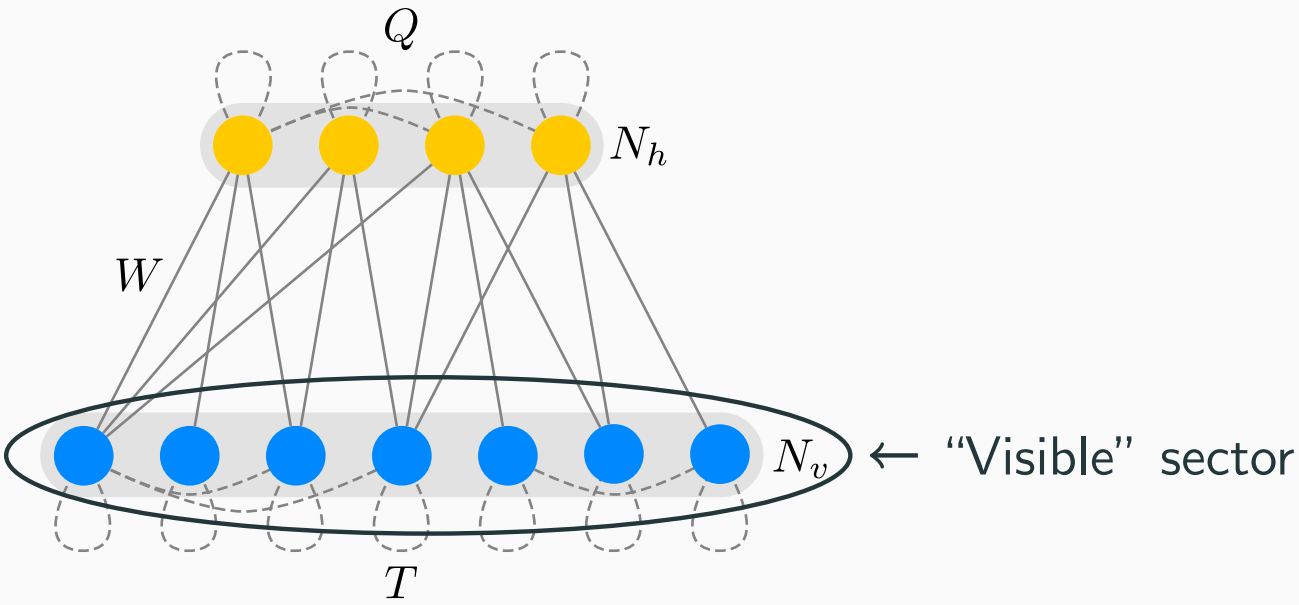
# Boltzmann machine

**Graphical representation:**



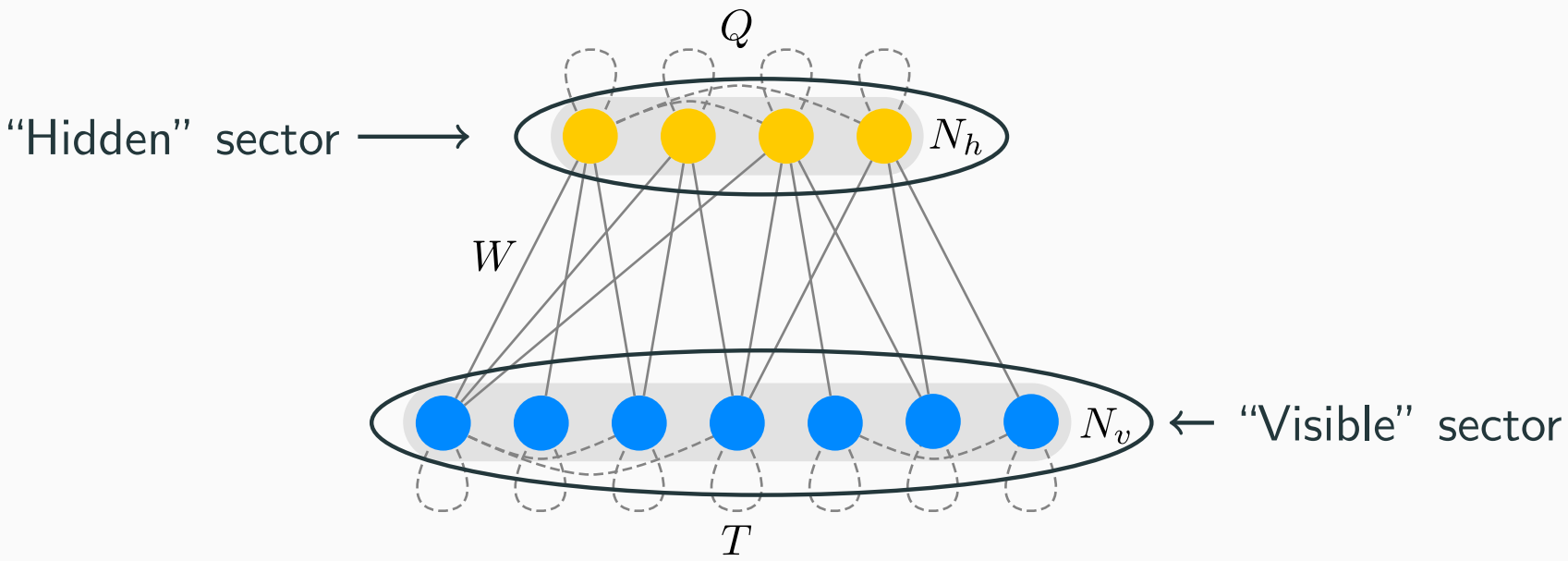
# Boltzmann machine

**Graphical representation:** **[Hinton, Sejnowski '86]**



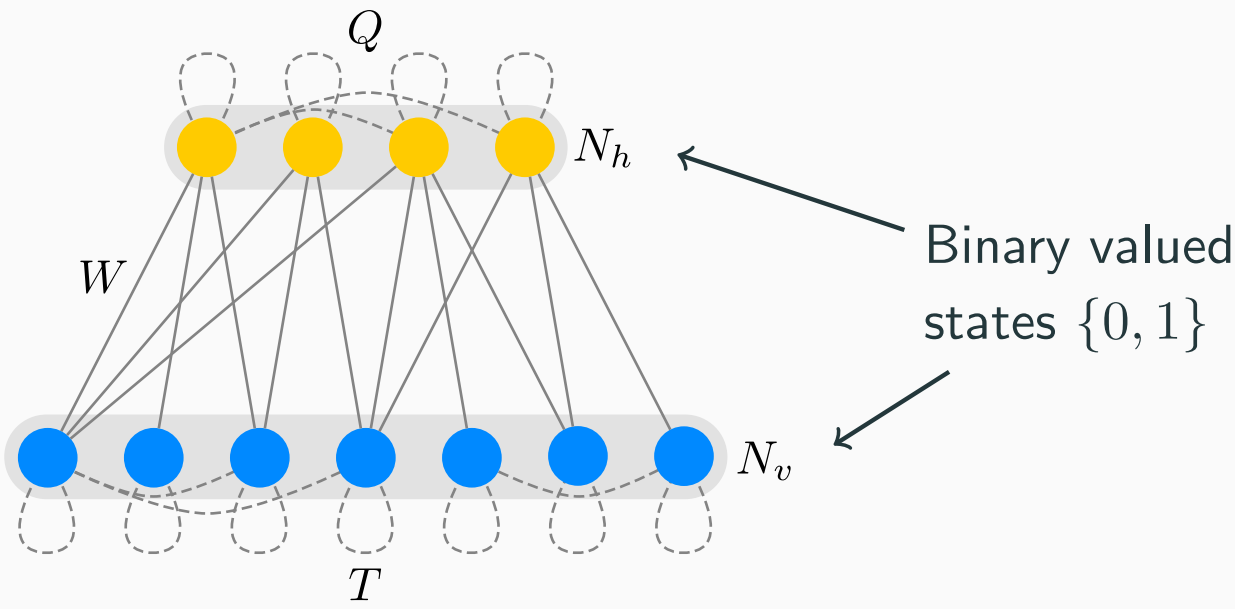
# Boltzmann machine

**Graphical representation:** **[Hinton, Sejnowski '86]**



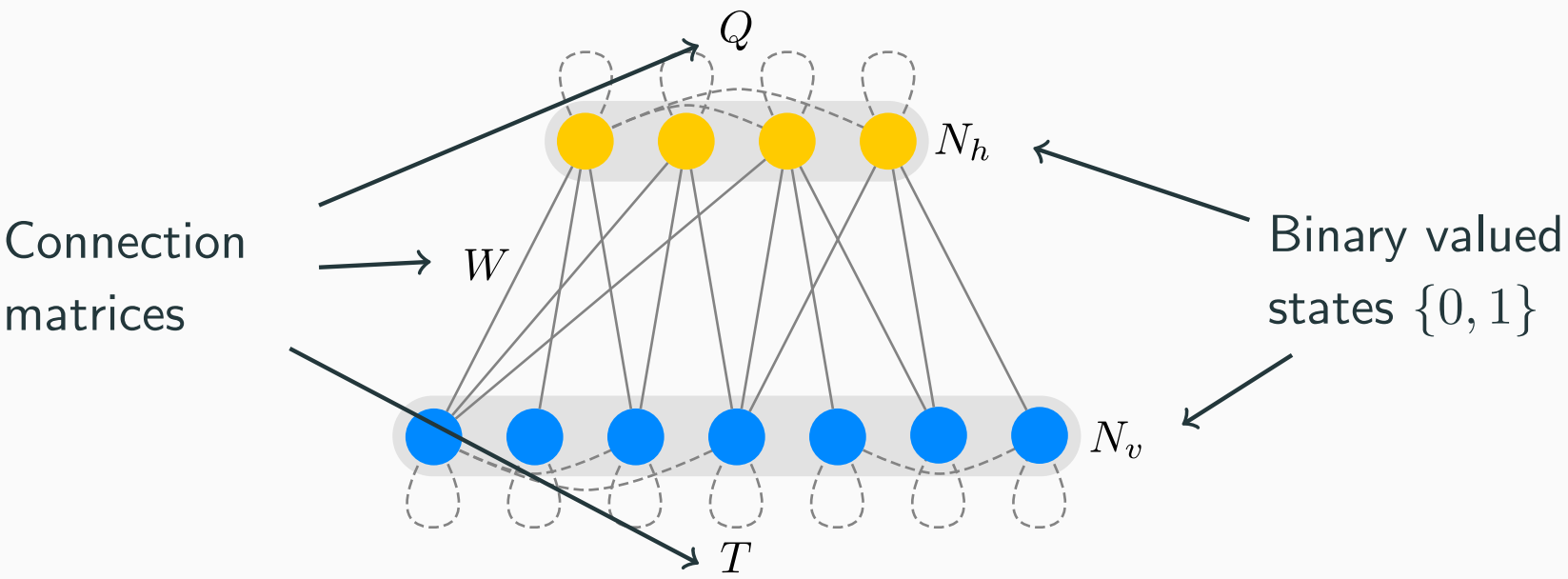
# Boltzmann machine

**Graphical representation:** **[Hinton, Sejnowski '86]**



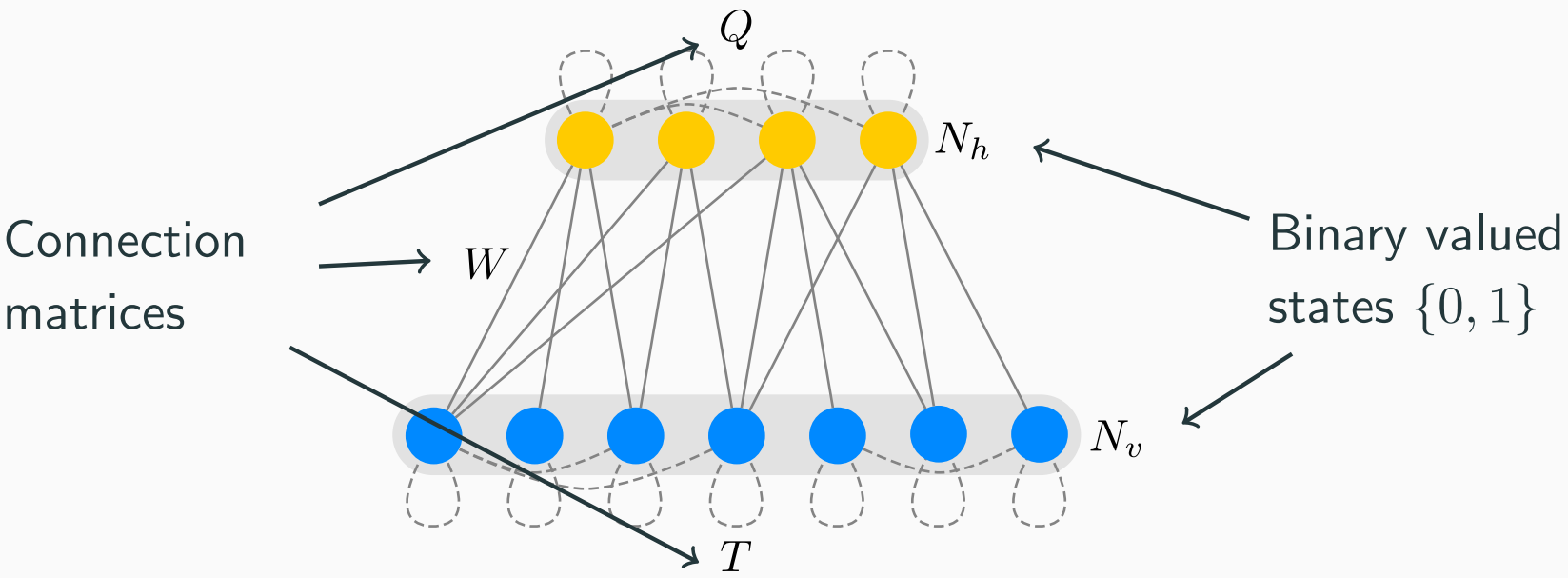
# Boltzmann machine

**Graphical representation:** **[Hinton, Sejnowski '86]**



# Boltzmann machine

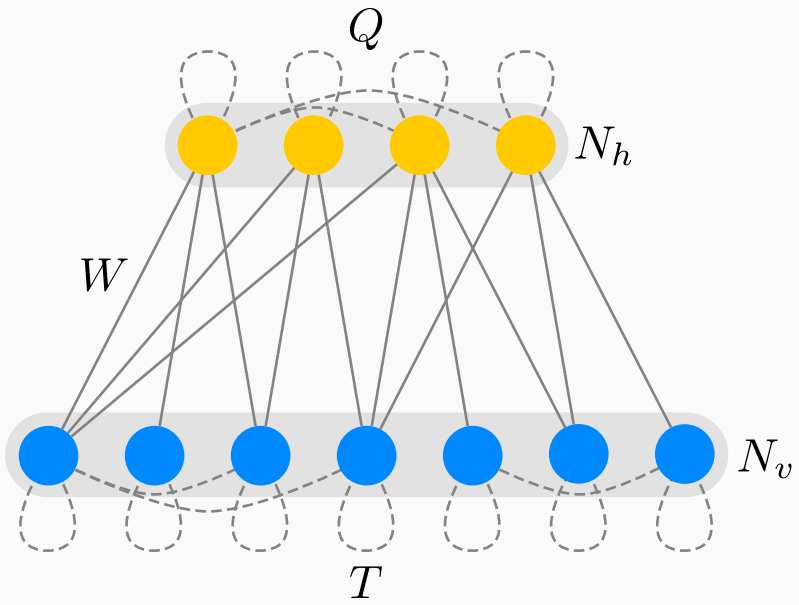
**Graphical representation:** **[Hinton, Sejnowski '86]**




- Boltzmann machine (BM): $T$ and $Q \neq 0$.
- Restricted Boltzmann machine (RBM): $T = Q = 0$.

# Boltzmann machine

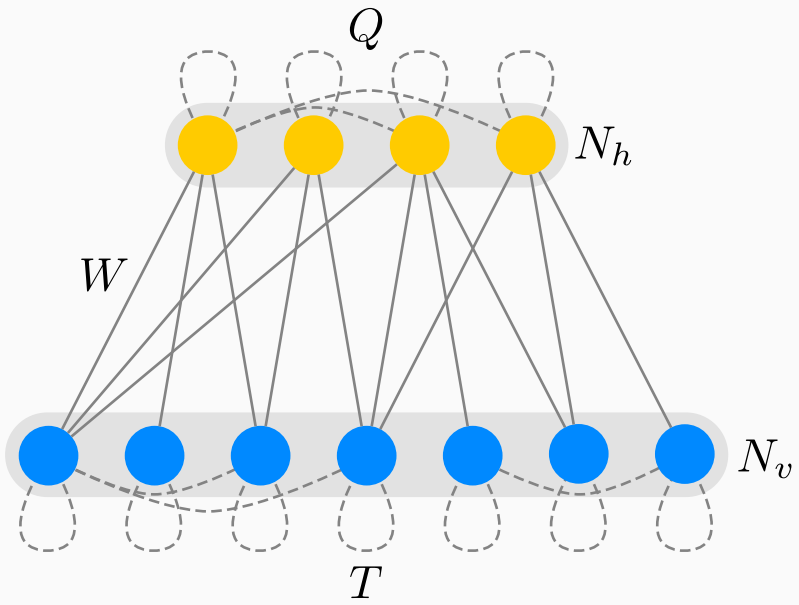
**Energy based model:** **[Hinton, Sejnowski '86]**




View as statistical mechanical system.

# Boltzmann machine

**Energy based model:** **[Hinton, Sejnowski '86]**



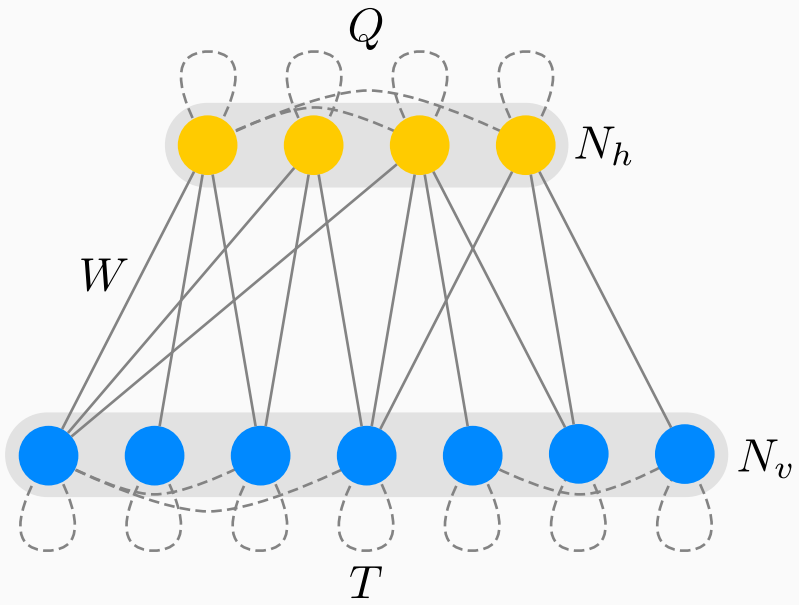

View as statistical mechanical system.

The system energy for given state vectors $(v, h)$:

$$E(v,h) = \frac{1}{2}v^t T v + \frac{1}{2}h^t Q h + v^t W h + B_h h + B_v v$$

# Boltzmann machine

**Energy based model:** **[Hinton, Sejnowski '86]**




View as statistical mechanical system.

The system energy for given state vectors $(v, h)$:

$$E(v,h) = \frac{1}{2}v^t T v + \frac{1}{2}h^t Q h + v^t W h + B_h h + B_v v$$

State vectors — Connection matrices — Biases

# Boltzmann machine

**Energy based model:** **[Hinton, Sejnowski '86]**

Starting from the system energy for given state vectors $(v, h)$:

$$E(v,h) = \frac{1}{2}v^t T v + \frac{1}{2}h^t Q h + v^t W h + B_h h + B_v v$$

The canonical partition function is defined as:

$$Z = \sum_{h,v} e^{-E(v,h)}$$

Probability the system is in specific state given by Boltzmann distribution:

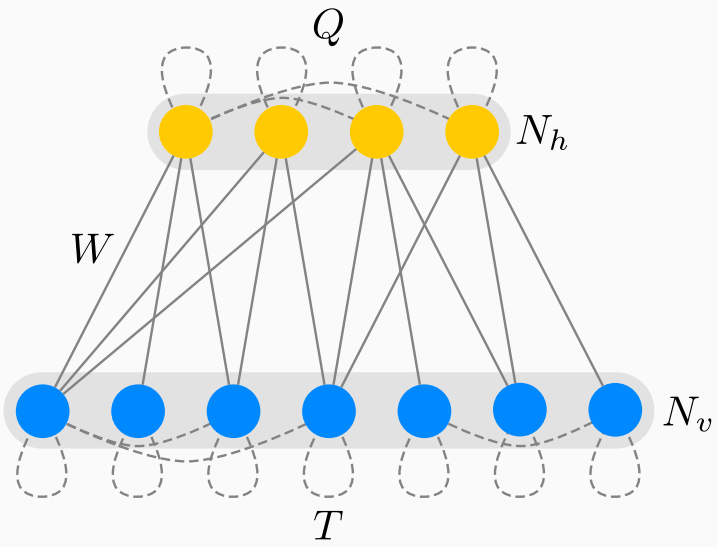
$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

with marginalization:

$$P(v) = \frac{e^{-F(v)}}{Z} \leftarrow \text{Free energy}$$

# Boltzmann machine

**Learning:** **[Hinton, Sejnowski '86]**



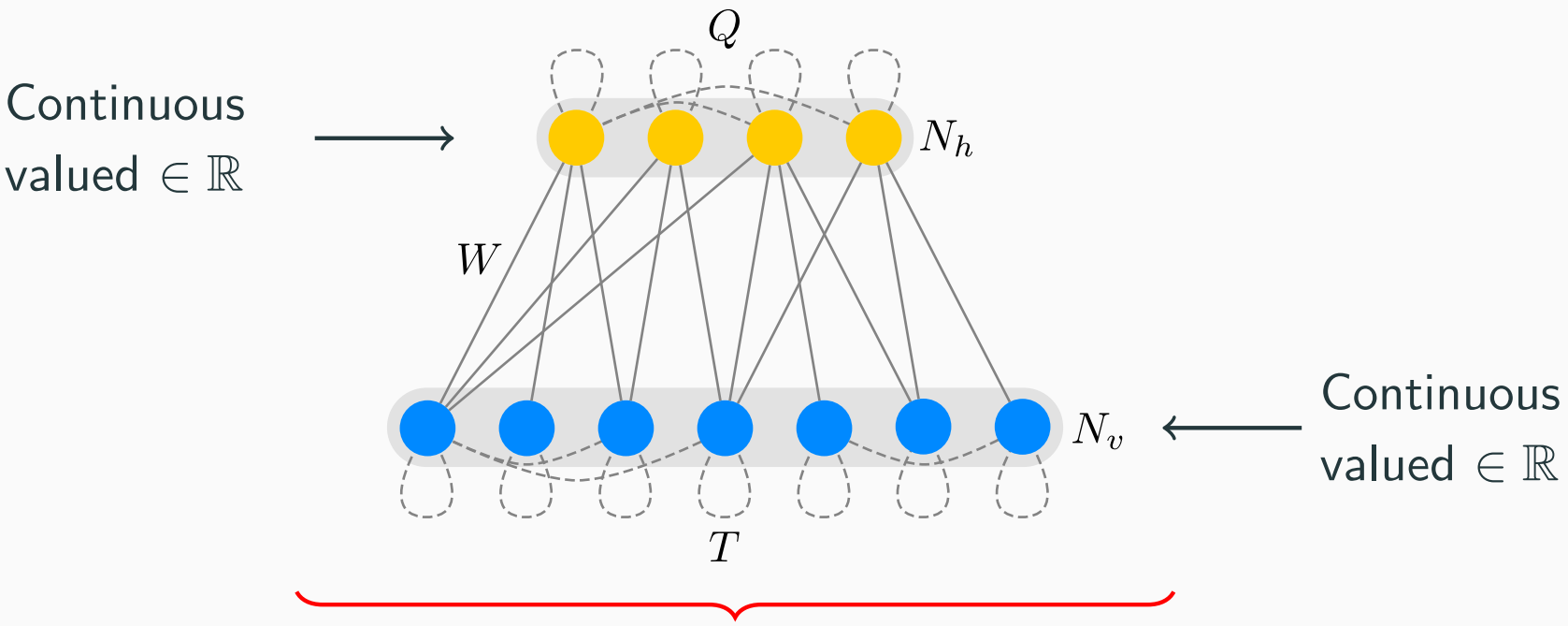

Theoretically, general compute medium.
Via adjusting $W, T, Q, B_h, B_v$ able to learn the underlying probability distribution of a given dataset.

**However: practically not feasible**

For applications only RBMs have been considered.

# Riemann-Theta Boltzmann machine

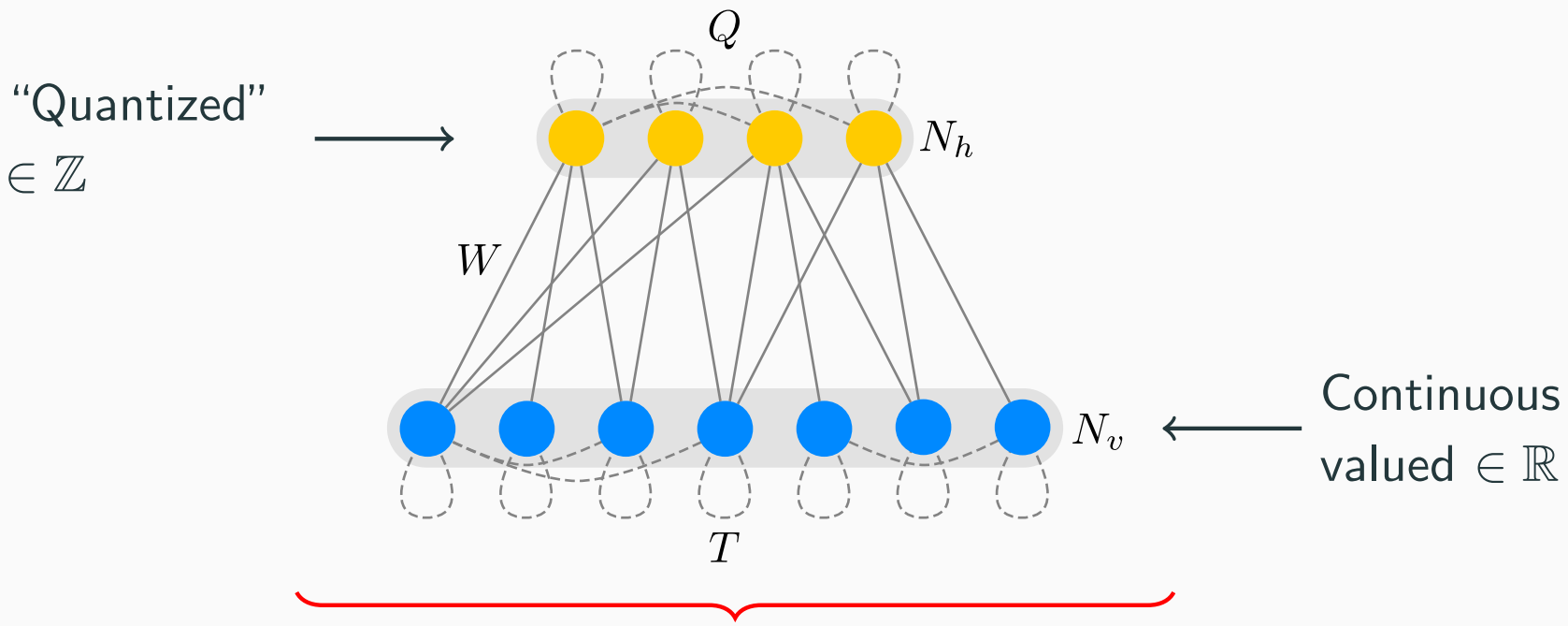
**How to change the status quo?** **[Krefl, S.C., Haghighat, Kahlen '17]**
Keep the inner sector couplings non-trivial, but the machine solvable?




$P(v) \equiv$ multi-variate gaussian (*too trivial*)

# Riemann-Theta Boltzmann machine

**How to change the status quo?** **[Krefl, S.C., Haghighat, Kahlen '17]**
Keep the inner sector couplings non-trivial, but the machine solvable?

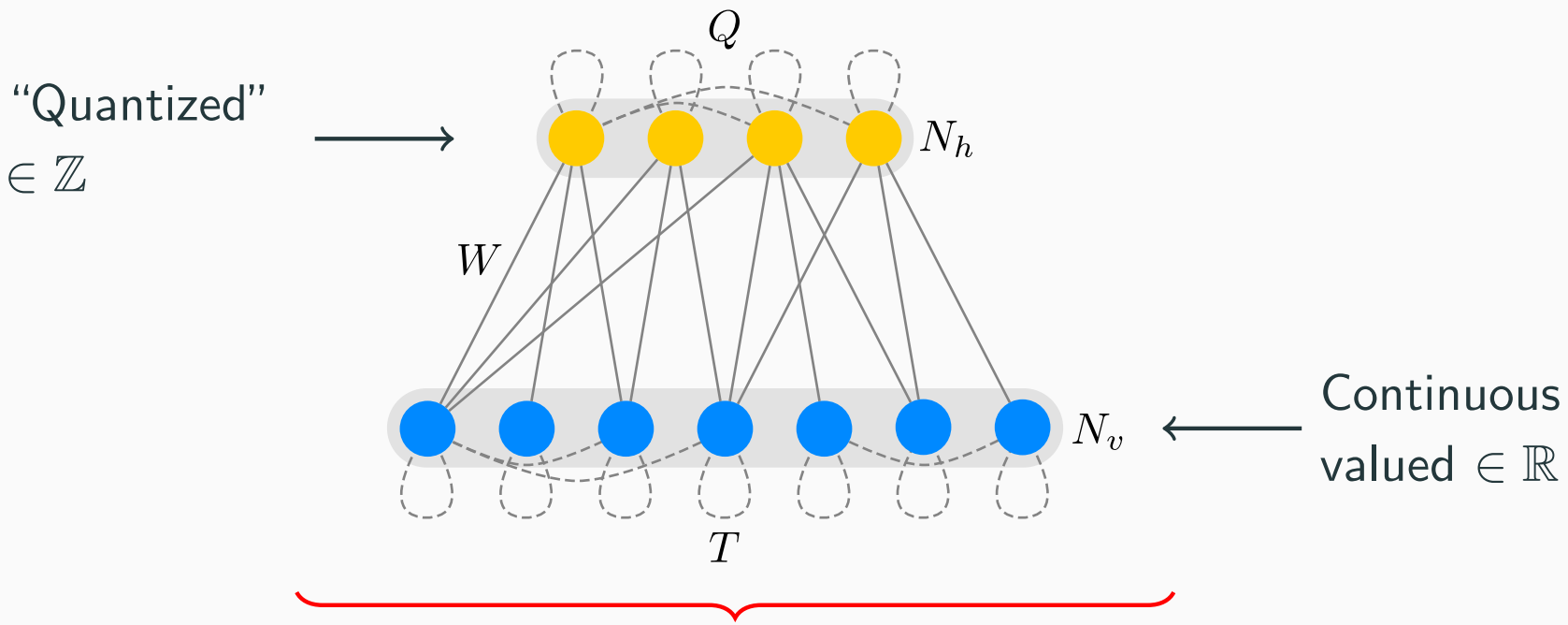



**Something interesting happens**

Under mild constraints on connection matrices (positive definiteness,...)

# Riemann-Theta Boltzmann machine

**How to change the status quo?** **[Krefl, S.C., Haghighat, Kahlen '17]**
Keep the inner sector couplings non-trivial, but the machine solvable?




$$P(v) \equiv \sqrt{\frac{\det T}{(2\pi)^{N_v}}} e^{-\frac{1}{2}v^t T v - B_v^t v - B_v^t T^{-1} B_v} \frac{\tilde{\theta}(B_h^t + v^t W | Q)}{\tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)}$$

**Closed form analytic solution still available!**

# Riemann-Theta Boltzmann machine

**RTBM** **[Krefl, S.C., Haghighat, Kahlen '17]**

Novel very generic probability density:

$$P(v) \equiv \sqrt{\frac{\det T}{(2\pi)^{N_v}}} e^{-\frac{1}{2}v^t T v - B_v^t v - B_v^t T^{-1} B_v} \frac{\tilde{\theta}(B_h^t + v^t W | Q)}{\tilde{\theta}(B_h^t - B_v^t T^{-1} W | Q - W^t T^{-1} W)}$$

Damping factor

Riemann-Theta function

The Riemann-Theta definition:

$$\theta(z, \Omega) := \sum_{n \in \mathbb{Z}^{N_h}} e^{2\pi i \left(\frac{1}{2} n^t \Omega n + n^t z\right)}$$

**Key properties:** Periodicity, modular invariance, solution to heat equation, etc.

**Note:** Gradients can be calculated analytically as well so gradient descent can be used for optimization.

# RTBM applications

# Riemann-Theta Boltzmann machine

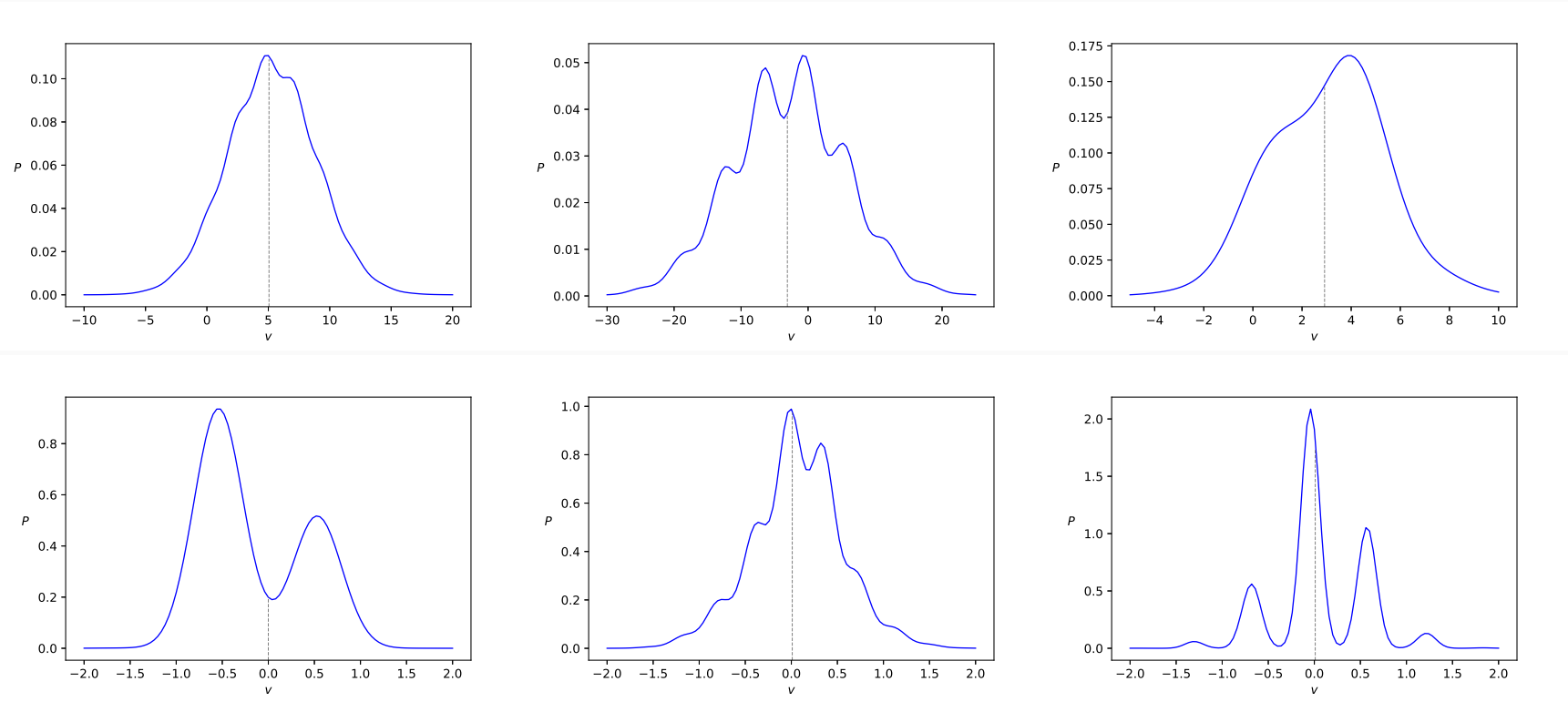
In the next we show examples of RTBMs for

- Probability determination
- Probability sampling
- Conditional probability
- Feature detection for data classification
- Data regression

# Probability determination

# Riemann-Theta Boltzmann machine

**RTBM $P(v)$ examples:** **[Krefl, S.C., Haghighat, Kahlen '17]**



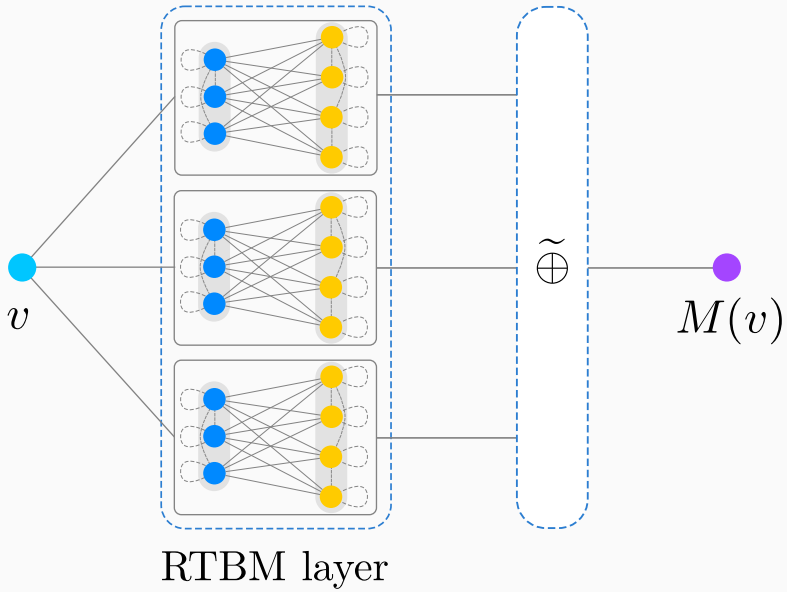

For different choices of parameters (with hidden sector in 1D or 2D).

# Riemann-Theta Boltzmann machine

**Mixture model:**

**Expectation:**
As long as the density is well enough behaved at the boundaries it can be learned by an RTBM mixture model.

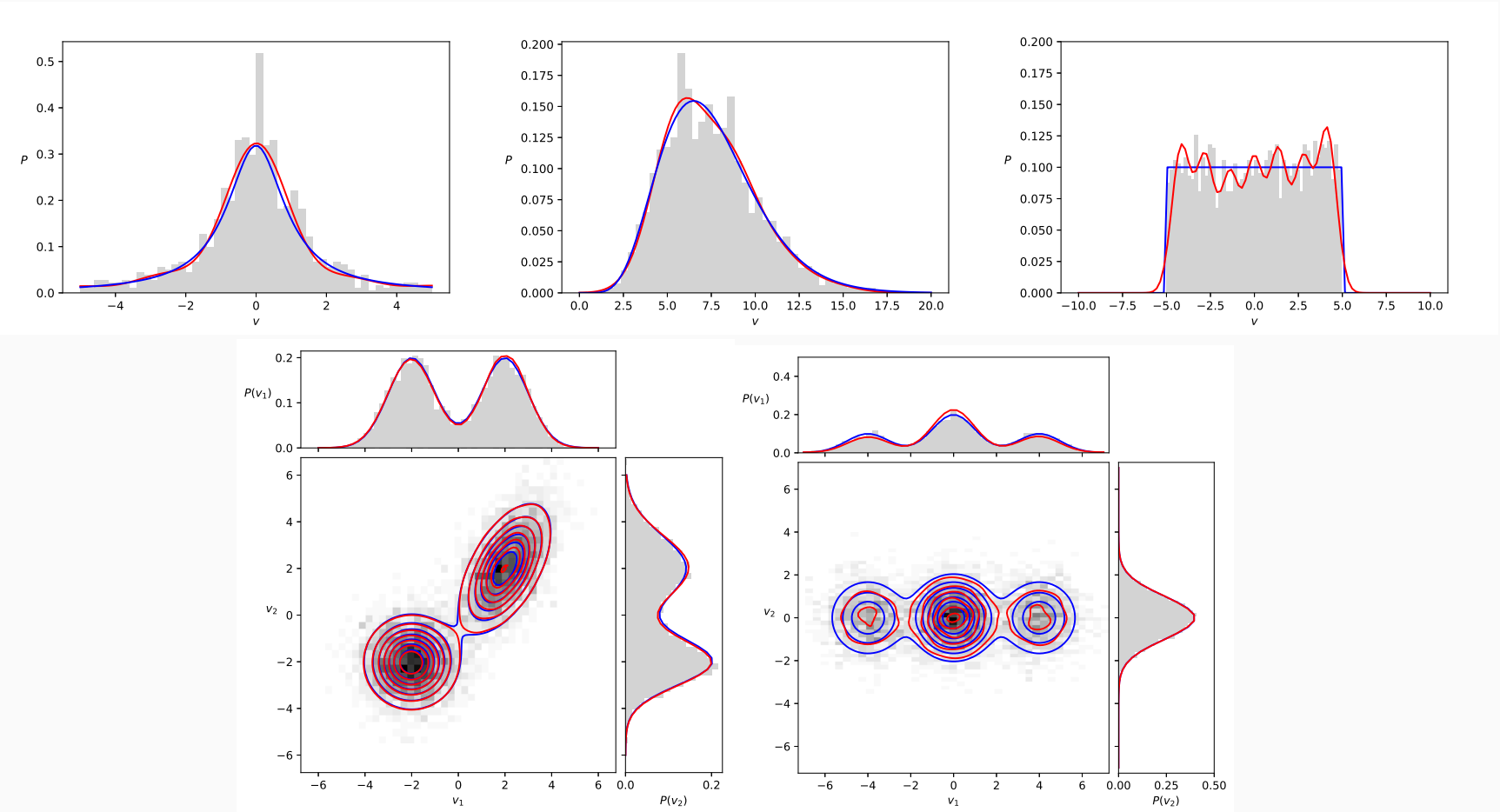
**[Krefl, S.C., Haghighat, Kahlen '17]**



# Riemann-Theta Boltzmann machine

**Examples:**

**[Krefl, S.C., Haghighat, Kahlen '17]**




Top $N_v = 1,\ N_h = 3, 2, 3,$ button $N_v = 2,\ N_h = 1\,(2\text{x RTBM}), 2.$

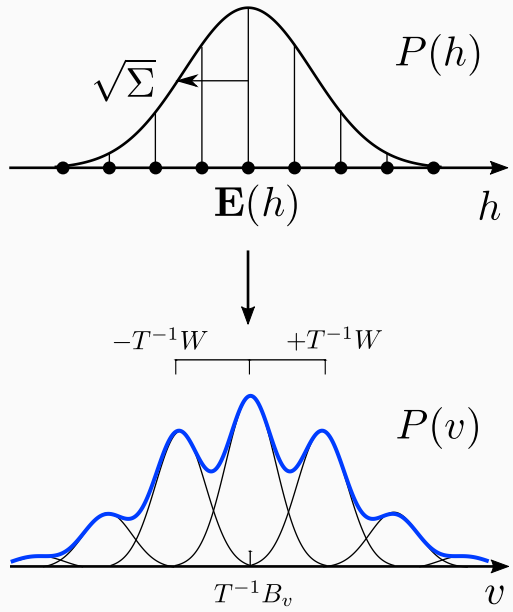
# Probability sampling

# RTBM sampling algorithm

The probability for the visible sector can be expressed as:

$$P(v) = \sum_{[h]} P(v|h)P(h)$$

where $P(v|h)$ is a multivariate gaussian. The $P(v)$ sampling can be performed easily by:



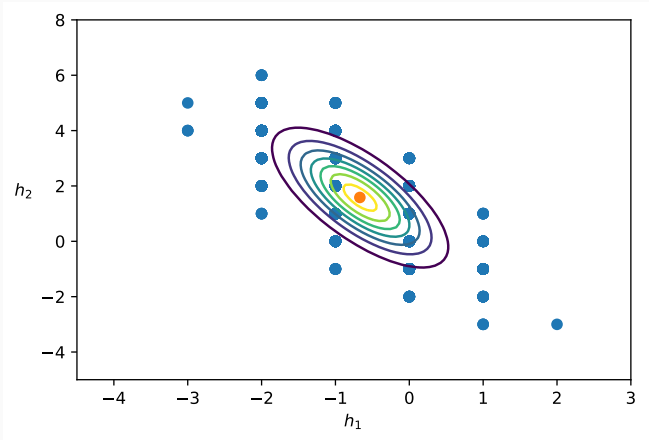

- sampling $\mathbf{h} \sim P(h)$ using the RT numerical evaluation $\theta = \theta_n + \epsilon(R)$ with ellipsoid radius $R$ so

$$p = \frac{\epsilon(R)}{\theta_n + \epsilon(R)} \ll 1$$

is the probability that a point is sampled outside the ellipsoid of radius $R$, while

$$\sum_{[h](R)} P(h) = \frac{\theta_n}{\theta_n + \epsilon(R)} \approx 1$$

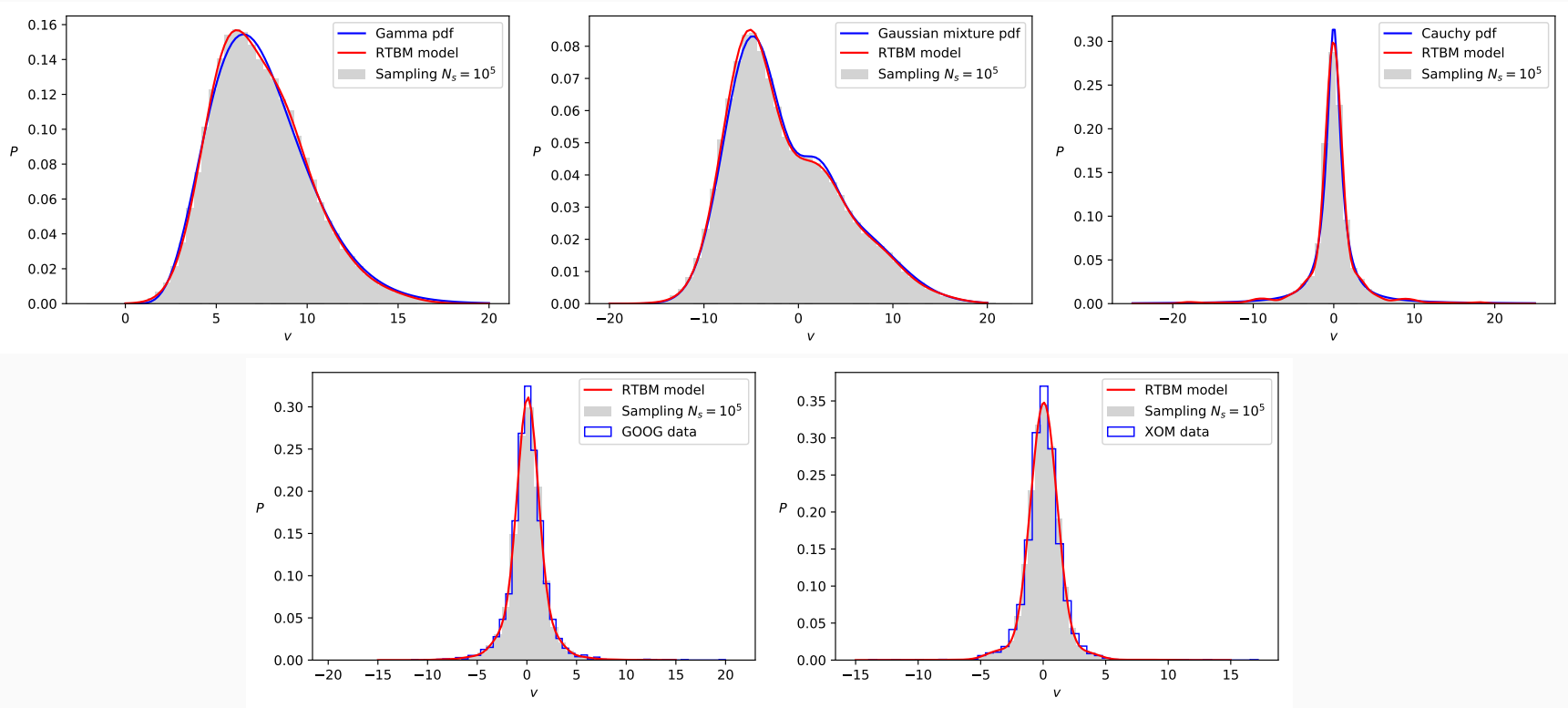
*i.e.* sum over the lattice points inside the ellipsoid.




- then sampling $\mathbf{v} \sim P(v|\mathbf{h})$

# Sampling examples

## RTBM $P(v)$ sampling examples: [S.C. and Krefl '18]




Top $N_v = 1$, $N_h = 2, 3$ (2x RTBM), $3$, button $N_v = 1$, $N_h = 3$.

# Sampling distance estimators

| **Distribution** | $\chi^2_{\mathrm{RTBM}}/N_{\mathrm{bins}}$ | $\mathrm{MSE}^{\mathrm{sampling}}_{\mathrm{RTBM}}$ | $\mathrm{MSE}^{\mathrm{sampling}}_{\mathrm{pdf}}$ | $\mathrm{MSE}^{\mathrm{pdf}}_{\mathrm{RTBM}}$ | KS distance |
|---|---|---|---|---|---|
| Gamma | 0.02/50 | $2 \cdot 10^{-5}$ | $2.6 \cdot 10^{-5}$ | $3.4 \cdot 10^{-4}$ | 0.01 |
| Cauchy | 0.12/50 | $2.9 \cdot 10^{-4}$ | $3.7 \cdot 10^{-4}$ | $1.5 \cdot 10^{-3}$ | 0.02 |
| Gaussian mixture | 0.01/50 | $6.7 \cdot 10^{-6}$ | $1.4 \cdot 10^{-5}$ | $9.3 \cdot 10^{-5}$ | 0.01 |
| GOOG | 0.10/50 | $2.7 \cdot 10^{-4}$ | $9.5 \cdot 10^{-3}$ | $2.5 \cdot 10^{-4}$ | 0.02 |
| XOM | 0.09/50 | $2.6 \cdot 10^{-4}$ | $6.7 \cdot 10^{-3}$ | $3.7 \cdot 10^{-4}$ | 0.02 |

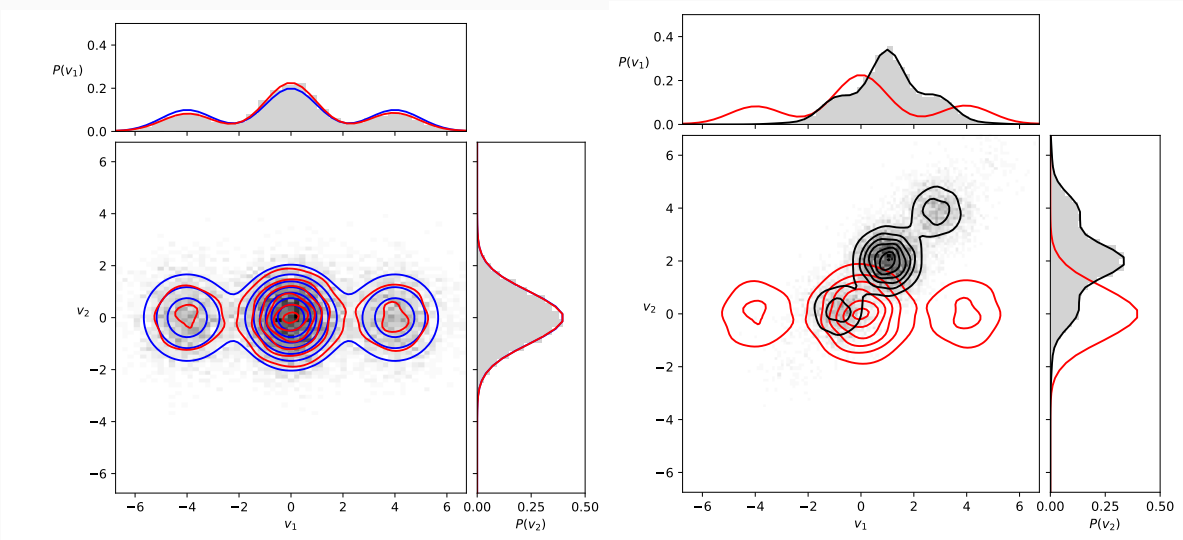
TABLE I: Distance estimators for the sampling examples in figures 3 and 4. Exact definitions for all distance estimators are given in section VII. The mean squared error (MSE) is taken between the sampling, the RTBM model and the underlying distribution (pdf). The Kolmogorov-Smirnov (KS) distance is shown in the last column of the table. For GOOG and XOM the empirical distribution is employed as underlying pdf.

| **Distribution** | Mean | 2nd moment | 3th moment | 4th moment |
|---|---|---|---|---|
| Gamma | 7.43 (7.43) [7.49] | 6.91 (6.89) [7.41] | 10.03 (10.03) [13.79] | 154 (153.23) [195.8] |
| Cauchy | -0.057 (-0.057) [-] | 11.64 (11.64) [-] | -4.63 (-4.97) [-] | 1749.8 (1753) [-] |
| Gaussian mixture | -1.48 (-1.48) [-1.31] | 34.45 (34.45) [34.29] | 134.35 (136.67) [131.78] | 3558.7 (3571.8) [3569.1] |
| GOOG | 0.06 (0.06) [0.08] | 3.28 (3.23) [3.58] | 1.52 (1.42) [6.04] | 117 (108) [191] |
| XOM | 0.02 (0.02) [0.03] | 2.13 (2.15) [2.36] | -0.42 (-0.18) [1.44] | 38.3 (40.2) [97.1] |

TABLE II: Mean and central moments for the sampling data, the RTBM model (round brackets) and the underlying true distribution (square brackets). Note that the moments of the Cauchy distribution are either undefined or infinite. The given values correspond to the RTBM model approximation and its sampling, which are defined and finite, *cf.*, (4). For the GOOG and XOM distributions the true moments (square brackets) are evaluated from the underlying empirical distribution.

# Sampling examples with affine transformation

**RTBM $P(v)$ sampling with affine transformation:** **[S.C. and Krefl '18]**




For a rotation of $\theta = \pi/4$ and scaling of 2 ($N_v = 2$, $N_h = 2$).

# Conditional probability

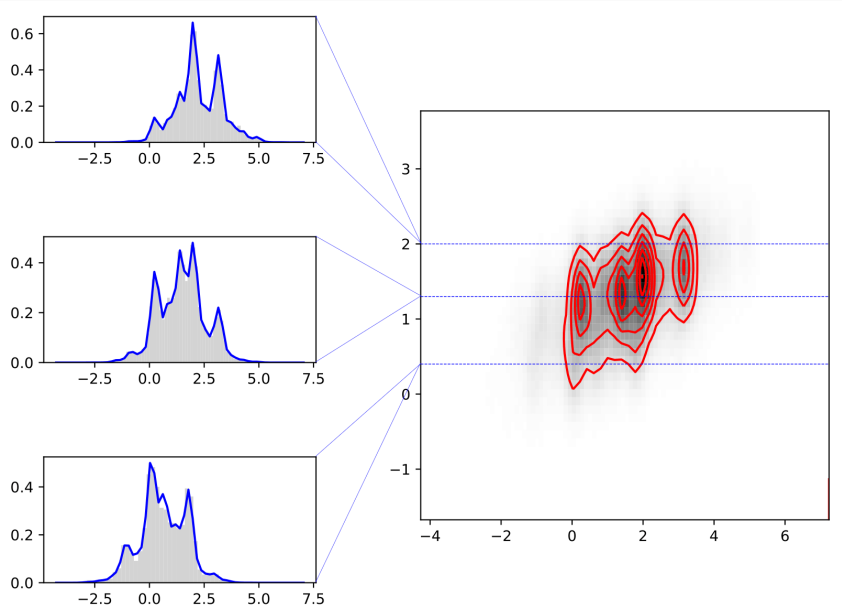
# Conditional probability estimation

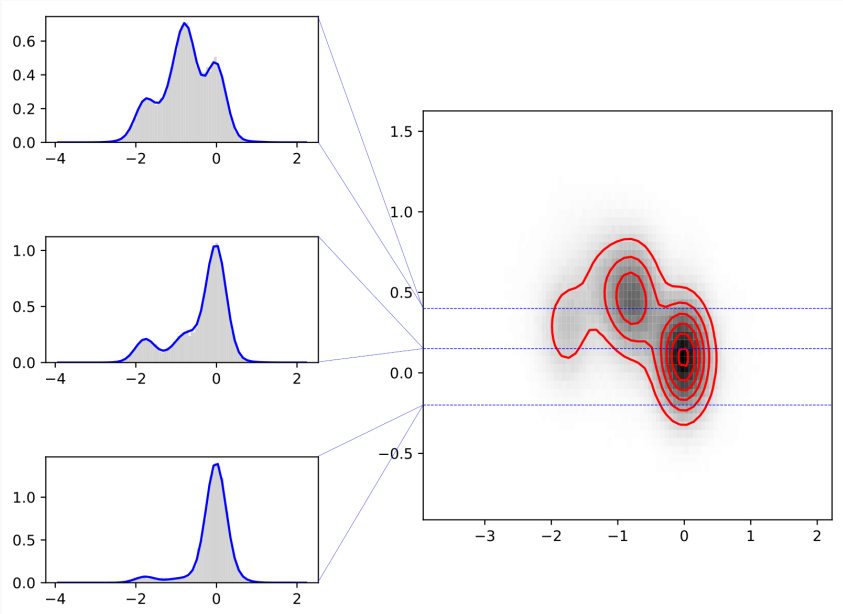
**[Papaluca, S.C., Krefl '19 in preparation]**

Considering a probability function $P(v)$ modelled by a RTBM, given some observed data $d$ and some future outcome $y$, *i.e.* $v = (y, d)$:

$$P(y|d) = \frac{P(y,d)}{P(d)} = \frac{\sqrt{t_0}}{2\pi} e^{-\frac{1}{2}t_0 y^2 - B_0 y + \frac{1}{2t_0}B_0^2} \frac{\tilde{\theta}(B_h^t - v^t W|Q)}{\tilde{\theta}(B_h^t - r^t W|Q - \frac{W_0 W_0^t}{t_0})}$$

## Examples in 2D:





# Feature dectection

# Riemann-Theta Boltzmann machine

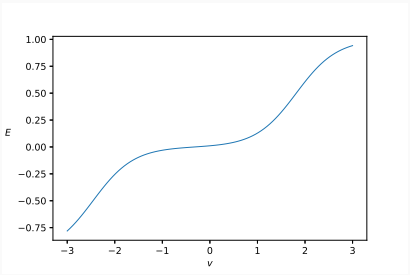
**Feature detector:**

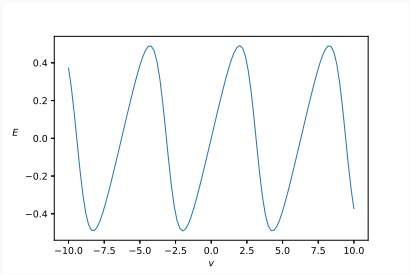
**[Krefl, S.C., Haghighat, Kahlen '17]**
**Similar to [Krizhevsky '09]**

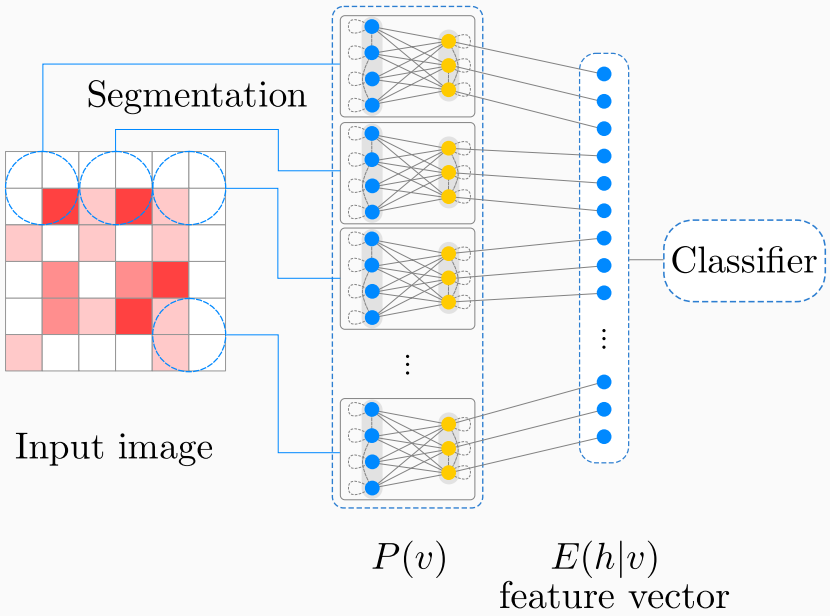
**New:**
Conditional expectations of hidden states after training

$$E(h_i|v) = -\frac{1}{2\pi i}\frac{\nabla_i \tilde{\theta}(v^t W + B_h^t|Q)}{\tilde{\theta}(v^t W + B_h^t|Q)}$$

The detector is trained in probability mode and generates a feature vector.



# Feature detector example - jet classification

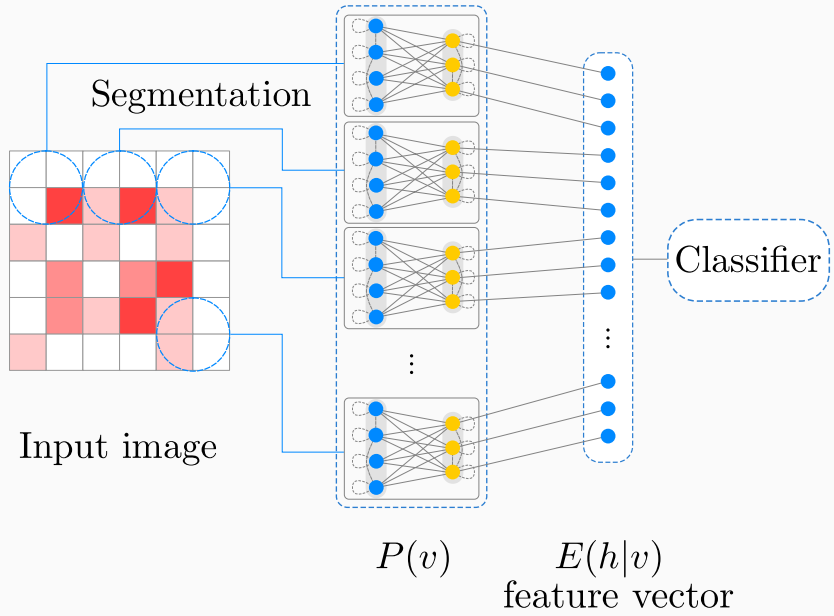
**Jet classification:**

**[Krefl, S.C., Haghighat, Kahlen '17]**
**Data from [Baldi et al. '16, 1603.09349]**

Descriminating jets from single hadronic particles and overlapping jets from pairs of collimated hadronic particles.

**Data (images of 32x32 pixels)**

- 5000 images for training
- 2500 images for testing



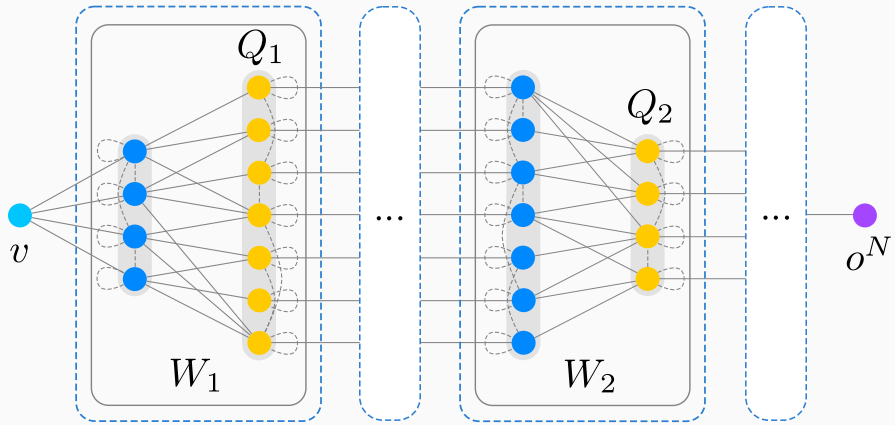

| Classifier | Test dataset precision |
|---|---|
| Logistic regression (LR) | 77% |
| RTBM feature detector + LR | 83% |

# Data regression

# Riemann-Theta Boltzmann machine

**Theta Neural Network:** **[Krefl, S.C., Haghighat, Kahlen '17]**

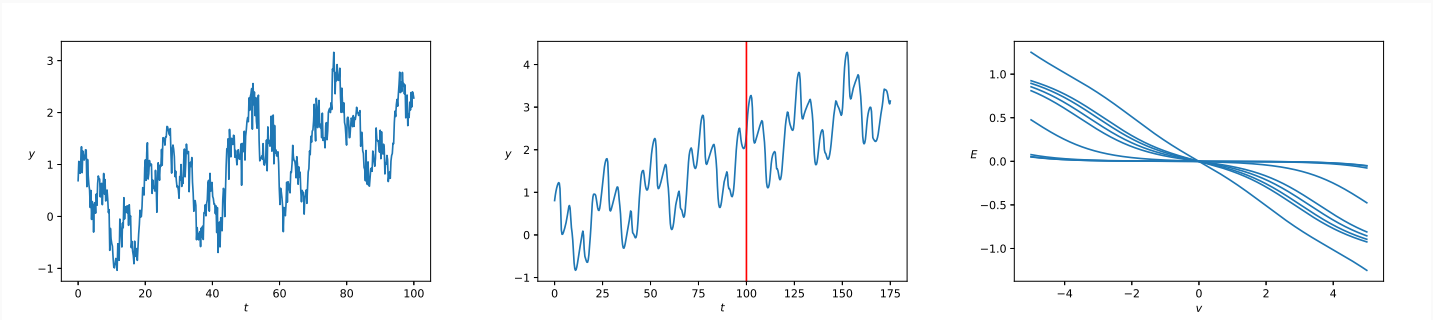
**Idea:**
Use as activation function in a standard NN. The particular form of non-linearity is learned from data.




**Key point:**
smaller networks needed but Riemann-Theta evalution is expensive.

**Example (1:3-3-2:1):**

$$y(t) = 0.02t + 0.5\sin(t + 0.1) + 0.75\cos(0.25t - 0.3) + \mathcal{N}(0, 1)$$

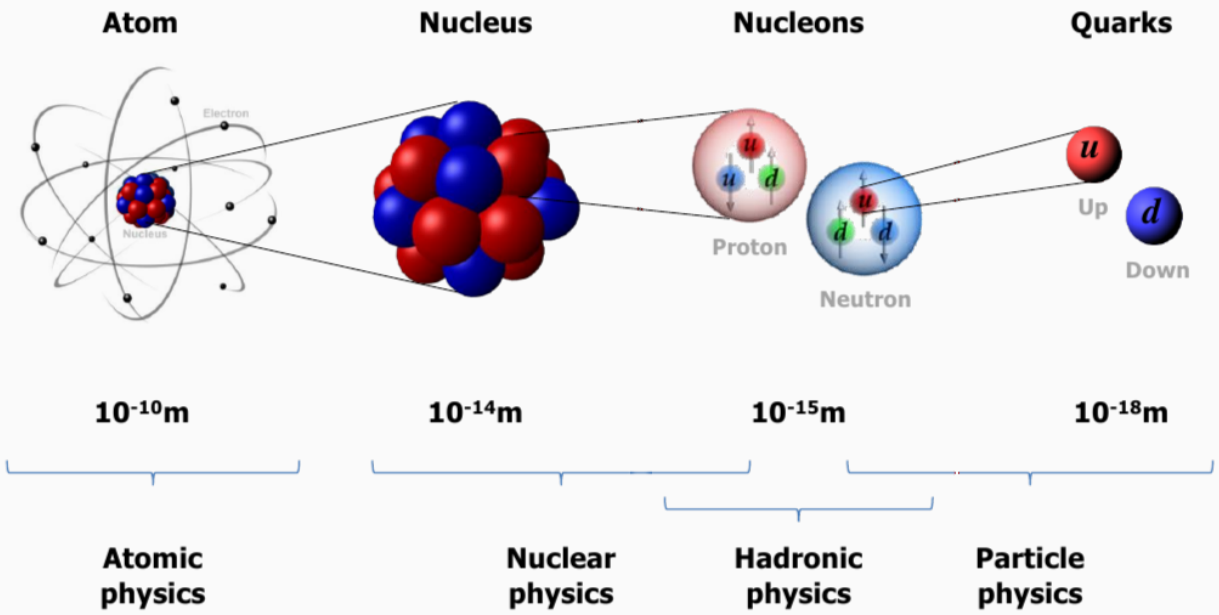


$y(t)$ TNN fit TNN activations

# ML and Parton Density functions

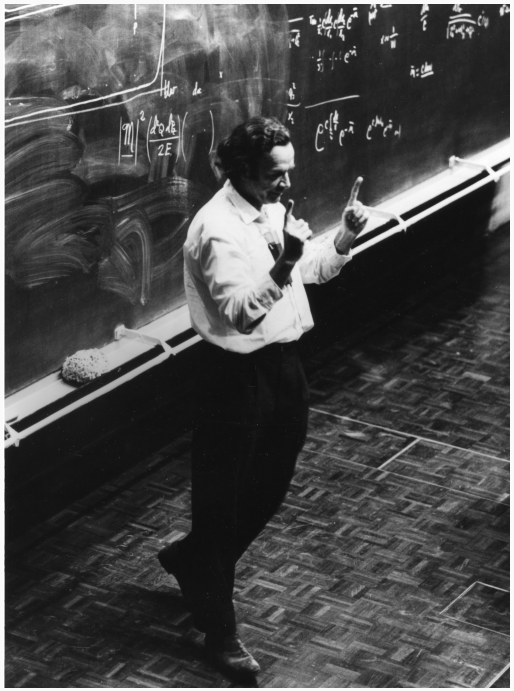
# Parton density functions

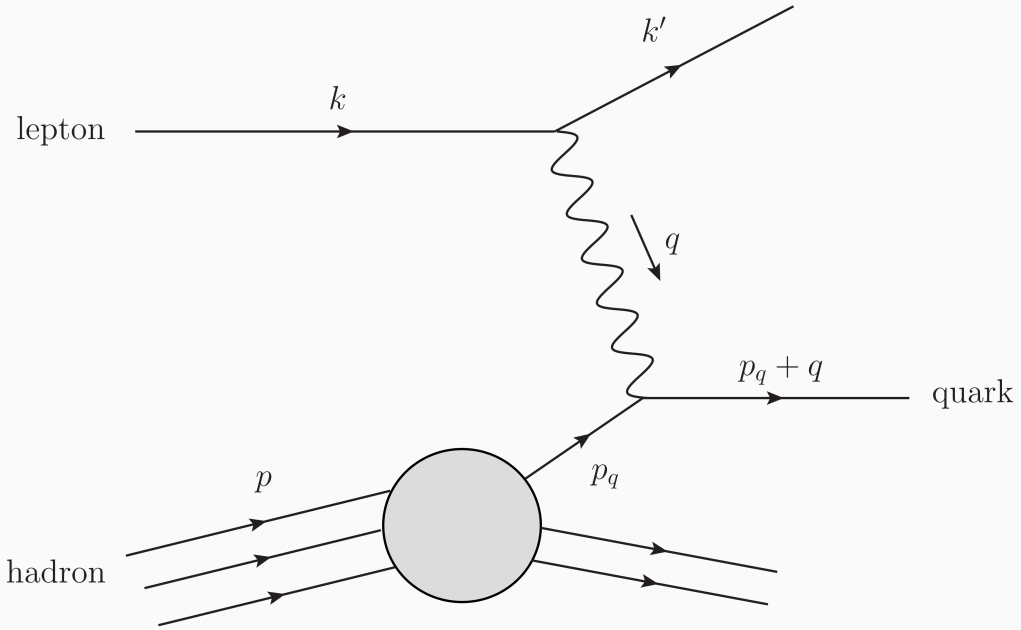
The **parton** model was introduced by Feynman in 1969 in order to characterize **hadrons** (*e.g.* protons and neutrons) in QCD processes and interactions in high energy particle collisions.




Partons are quarks and gluons characterized by a probability density functions of its nucleon momentum.

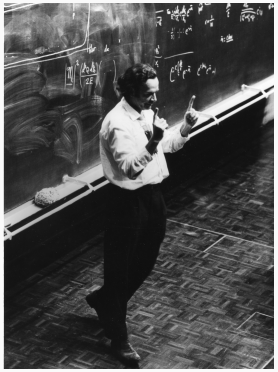
## The Feynman Parton Model






- **Photon probes the proton** by striking a **free massless "parton"** (quark, gluon) that carries a fraction $x$ of its parent proton.
- Value of $x$ is fixed by final-state kinematics.
- Cross-section proportional to probability $q_i(x)$ of finding parton of species $i$ with momentum-fraction $x$ in target proton.

# Perturbative QCD



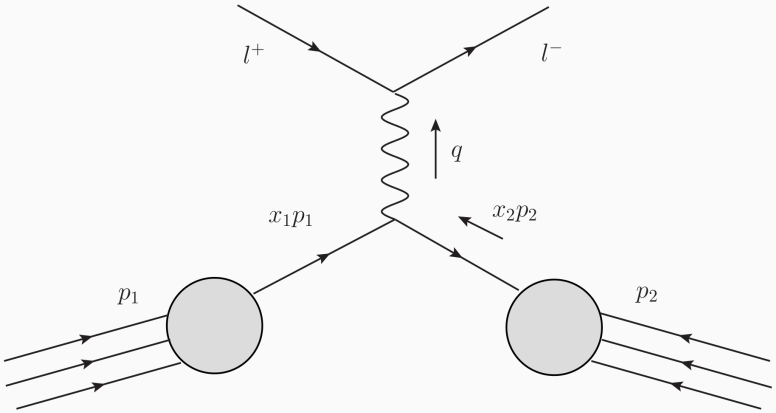

- The **Parton Model** is the first order of a perturbative expansion
- **PDFs** are **not calculable**: reflect non-perturbative physics of confinement.
- **PDFs** are **essential** for a **realistic computation** of any particle physics **observable**, $\sigma$, thanks to the factorization theorem

$$\sigma = \hat{\sigma} \otimes f,$$

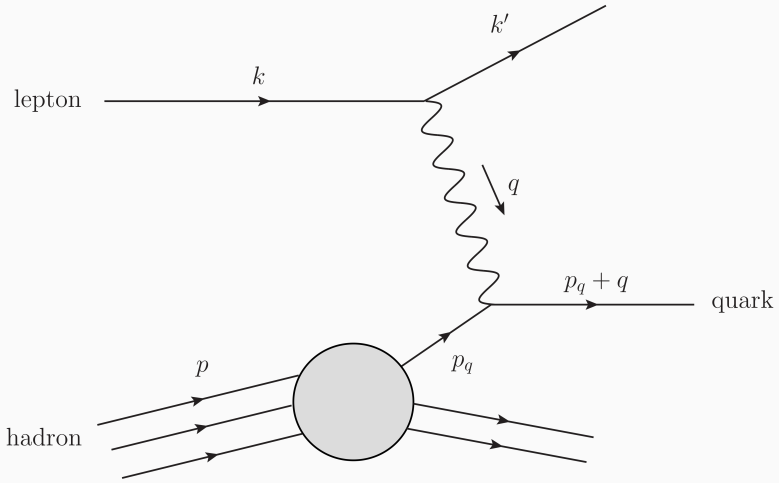
  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with $f$ the **PDF**.
- Can be **proven rigorously** using the **OPE** (Wilson expansion).

Factorization theorem is applied to several processes:

**Drell-Yan (e.g. LHC)**



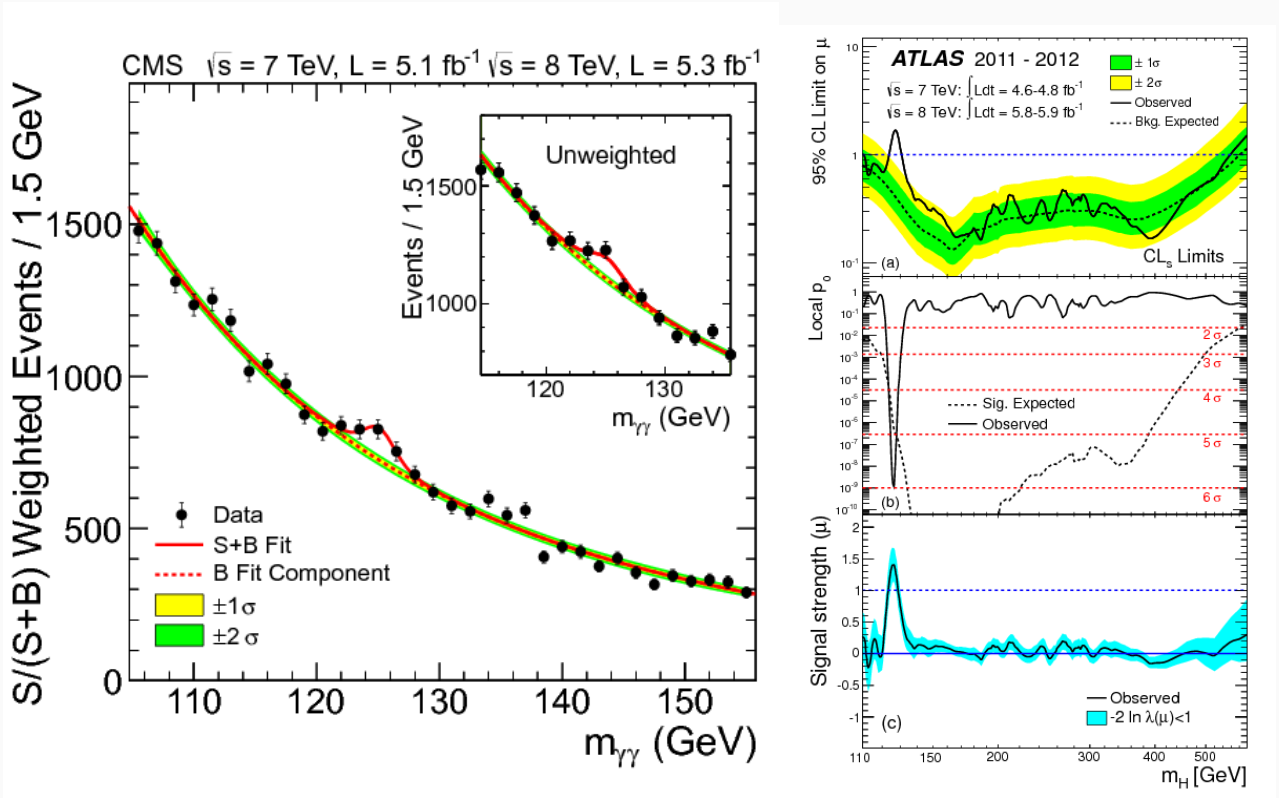

**DIS**




**PDFs** are **extracted** by comparing theoretical predictions to real data.

# Parton density functions

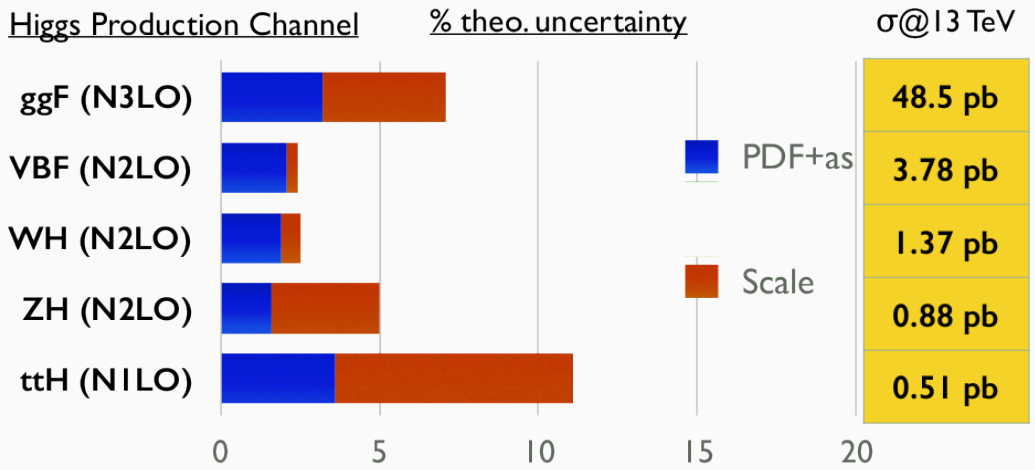
- PDFs are **necessary** to determine theoretical predictions for **signal**/**background** of experimental **measurements**.
  - *e.g. the Higgs discovery at the LHC*:



# PDF uncertainties

PDF determination requires a sensible estimate of the **uncertainty**, and not only the central value, so not a well researched topic in ML.

## CERN Yellow Report 4 (2016)




PDF uncertainties are a **limiting** factor in the accuracy of theoretical predictions for several processes at LHC.

$\Rightarrow$ Need of **precise** PDF determination and **uncertainty** estimate.

# Parton density functions

Historical examples of the first PDF models:

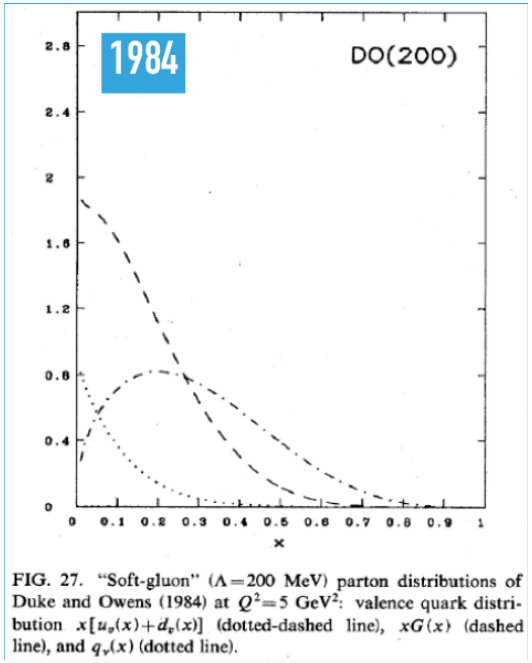
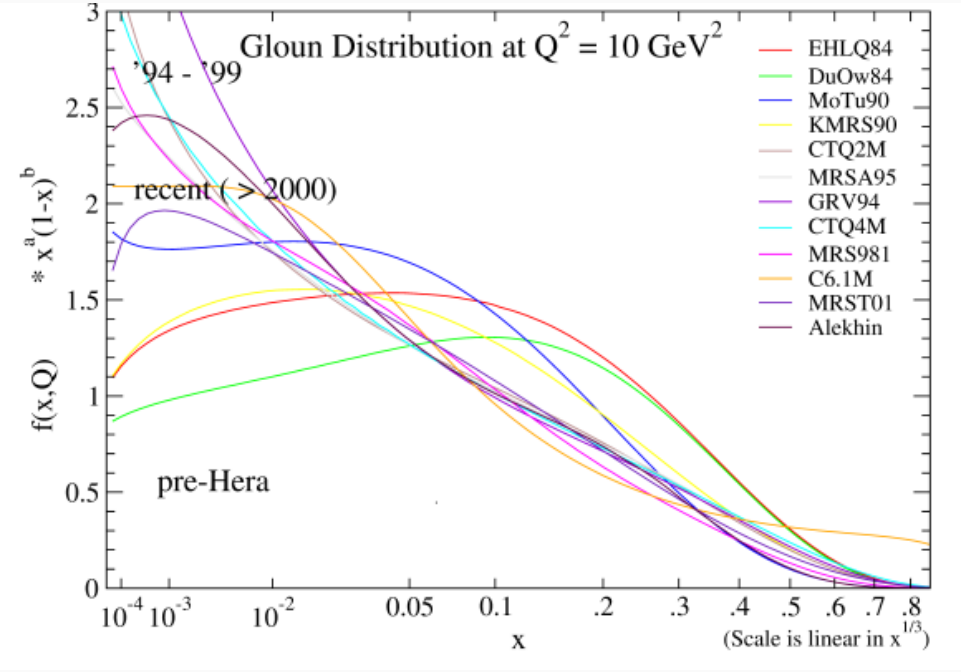



FIG. 27. "Soft-gluon" (A=200 MeV) parton distributions of Duke and Owens (1984) at $Q^2$=5 GeV$^2$: valence quark distribution $x[u_v(x)+d_v(x)]$ (dotted-dashed line), $xG(x)$ (dashed line), and $q_v(x)$ (dotted line).

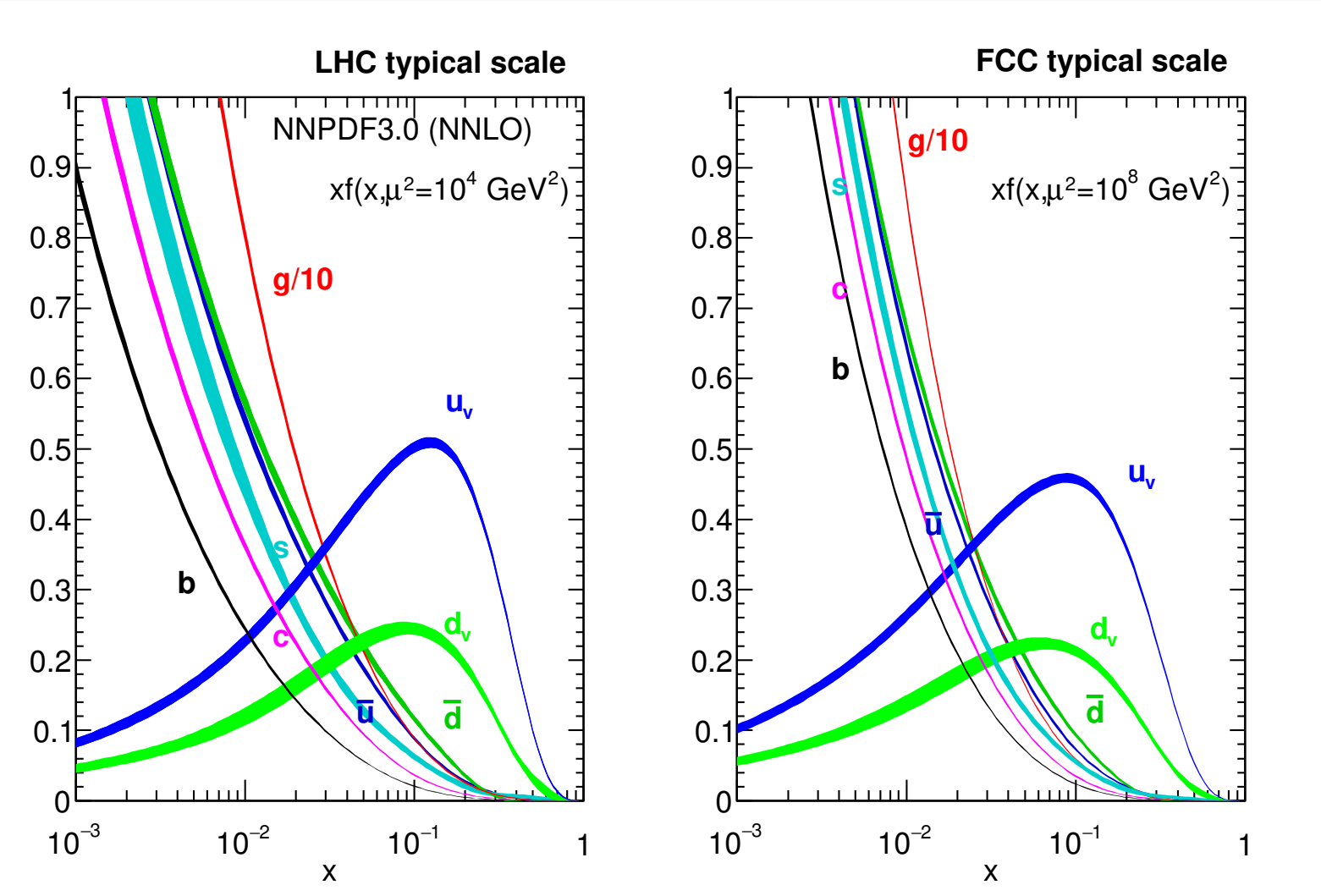



where

- PDFs are very simple functional forms (polynomials).
- PDFs are constrained by few data points and low order theory.
- No uncertainties are provided.
- No cross-validation methods are implemented.

# Parton density functions

**Possible improvement:** use ML in PDF determination.
**NNPDF** (Neural Network PDFs) created **10 years ago**.



# Why ML in PDFs determination?

- **PDFs** are **essential** for a **realistic computation** of hadronic particle physics **observable**, $\sigma$, thanks to the factorization theorem, *e.g.* in $pp$ collider:

$$\underbrace{\sigma_X(s, M_X^2)}_{Y} = \sum_{a,b} \int_{x_{\min}}^{1} dx_1 dx_2 \; \underbrace{\hat{\sigma}_{a,b}(x_1, x_2, s, M_X^2)}_{X} \; f_a(x_1, M_X^2) f_b(x_2, M_X^2),$$

  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with $f$ the **PDF**.

- $f_i(x_1, M_X^2)$ is the PDF of parton $i$ carrying a fraction of momentum $x$ at scale $M \Rightarrow$ **needs to be learned from data.**

# Why ML in PDFs determination?

- **PDFs** are **essential** for a **realistic computation** of hadronic particle physics **observable**, $\sigma$, thanks to the factorization theorem, *e.g.* in $pp$ collider:

$$\underbrace{\sigma_X(s, M_X^2)}_{Y} = \sum_{a,b} \int_{x_{\min}}^{1} dx_1 dx_2 \; \underbrace{\hat{\sigma}_{a,b}(x_1, x_2, s, M_X^2)}_{X} \; f_a(x_1, M_X^2) f_b(x_2, M_X^2),$$

  where the elementary **hard cross-section** $\hat{\sigma}$ is convoluted with $f$ the **PDF**.

- $f_i(x_1, M_X^2)$ is the PDF of parton $i$ carrying a fraction of momentum $x$ at scale $M$ $\Rightarrow$ **needs to be learned from data.**

- Constraints come in the form of convolutions:

$$X \otimes f \to Y$$

- Experimental data points is ˜5000 $\rightarrow$ not a big data problem
- Data from several process and experiments over the past decades $\Rightarrow$ deal with **data inconsistencies**

# The NNPDF methodology

The NNPDF (Neural Networks PDF) implements the Monte Carlo approach to the determination of a global PDF fit. We propose to:

1. **reduce** all sources of **theoretical bias**:
   - no fixed functional form
   - possibility to reproduce non-Gaussian behavior

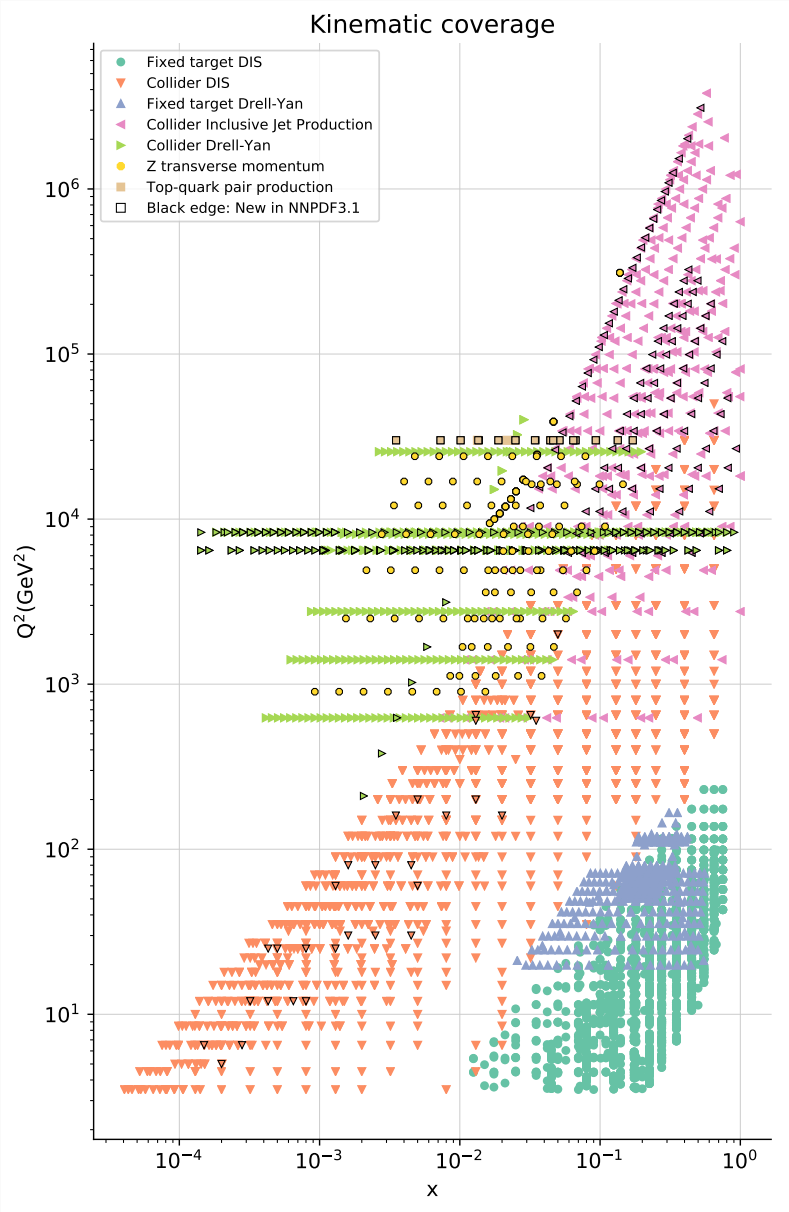
   $\Rightarrow$ use Neural Networks instead of polynomials
2. provide a sensible estimate of the **uncertainty**:
   - uncertainties from input experimental data
   - minimization inefficiencies and degenerate minima
   - theoretical uncertainties

   $\Rightarrow$ use MC artificial replicas from data, training with a GA minimizer
3. Test the setup through **closure tests**

# Experimental data

The total number of data points for the default PDF determination is

- 4175 at LO, 4295 at NLO and 4285 at NNLO.
- 7 physical processes from 14 experiments over ˜30 years (deal with data inconsistencies)
- few data points at high and low $x$ (deal with extrapolation)
- range of 5 and 7 orders of magnitude per PDF evaluation arguments $(x, Q^2)$



# DGLAP evolution

Can we reduce the PDF input size? **Yes**, thanks to DGLAP:

$$f_i(x_\alpha, Q^2) = \Gamma(Q, Q_0)_{ij\alpha\beta} f_j(x_\beta, Q_0^2)$$

We remove the $Q^2$ dependence from PDF determination thanks to the DGLAP evolution operator $\Gamma$.

$$f(x, Q^2) \to f(x, Q_0^2) := f(x)$$

- Precompute the DGLAP operator for all data points
- Apply the operator to the partonic cross section
- Store the results and perform fast convolutions

In NNPDF theoretical predictions are stored in **APFELgrid** tables:

$$\sigma = \sum_{i,j}^{n_f} \sum_{\alpha,\beta}^{n_x} W_{ij\alpha\beta} f_i(x_\alpha, Q_0^2) f_j(x_\beta, Q_0^2)$$

# Defining the ML problem

In comparison to a typical ML problem, a PDF fit

- requires a statistically sound uncertainty estimate
- is a regression problem but complex dependence on PDFs
- must satisfy physical constrains:
  - $f(x) \to 0$ for $x \to 1$ (continuity)
  - sum rules:

$$\sum_i^{n_f} \int_0^1 dx\, x f_i(x) = 1, \qquad \int_0^1 dx\, (u(x) - \bar{u}(x)) = 2$$

$$\int_0^1 dx\, (d(x) - \bar{d}(x)) = 1, \quad \int dx\, (q(x) - \bar{q}(x)) = 0,\ q = s, b, t$$

# PDF parametrizations

- Early models:

$$f_i(x) = A \cdot x^{\alpha}(1-x)^{\beta}$$

  - parameters are chosen based on Hessian minimization approach
  - Can a simple model provide a reliable uncertainty estimate?
  - Can it deal with data inconsistencies?

# PDF parametrizations

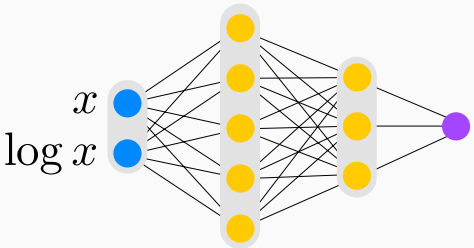
- Early models:

$$f_i(x) = A \cdot x^{\alpha}(1-x)^{\beta}$$

  - parameters are chosen based on Hessian minimization approach
  - Can a simple model provide a reliable uncertainty estimate?
  - Can it deal with data inconsistencies?

- NNPDF approach:

$$f_i(x, Q_0) = A \cdot x^{\alpha}(1-x)^{\beta} NN(x)$$




  - fully connected MLP (2-5-3-1)
  - two sigmoid hidden layers and linear output layer
  - x8 independent PDFs $\Rightarrow$ 296 free parameters

# Cost function

- We minimize the cost function:

$$\chi^2 = \sum_{ij} (D_i - O_i)\sigma_{i,j}^{-1}(D_j - O_j)$$

  - $D_i$ is the experimental measurement for point $i$
  - $O_i$ the theoretical prediction for point $i$ $(= \bar{\sigma} \otimes f)$
  - $\sigma_{ij}$ is the covariance matrix between points $i$ and $j$ with corrections for normalization uncertainties
  - supplemented by additional penalty terms for positivity observables

# Propagating experimental uncertainties

Generate artificial **Monte Carlo** data replicas from experimental data.

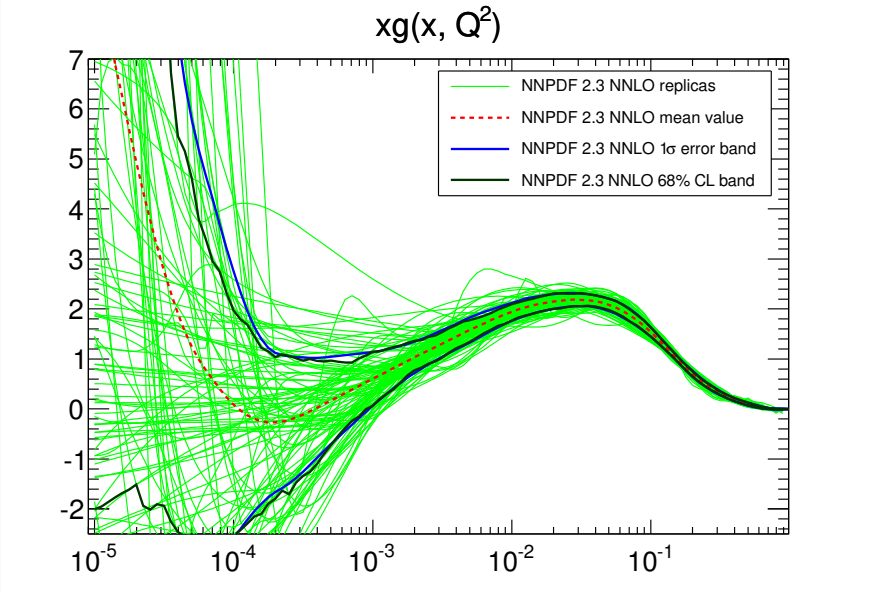
We perform $N_{\text{rep}}$ O(1000) fits, sampling pseudodata replicas:

$$D_i^{(r)} \to D_i^{(r)} + \text{chol}(\Sigma)_{i,j}\mathcal{N}(0,1), \quad i,j = 1..N_{\text{dat}},\, r = 1...N_{\text{rep}}$$

We obtain $N_{\text{rep}}$ PDF replicas. No assumptions at all about the Gaussianity of the errors.

# PDF fit example

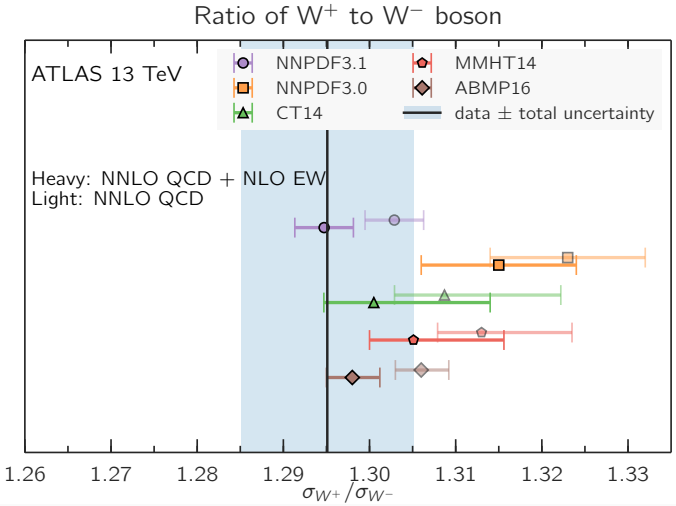
The procedure delivers a **Monte Carlo** representation of results:

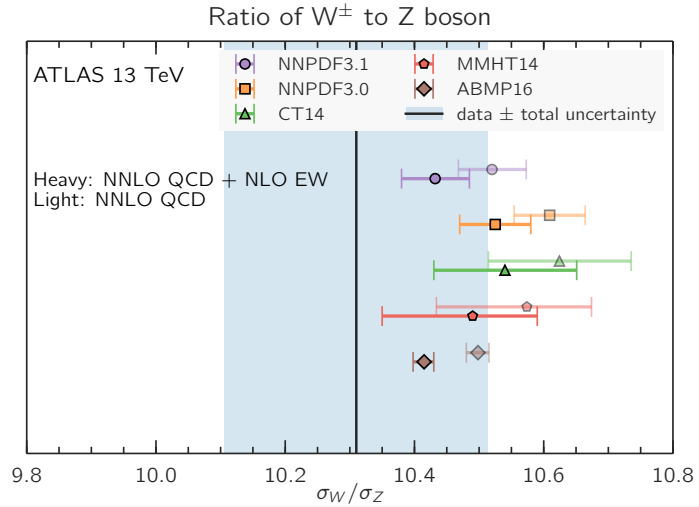


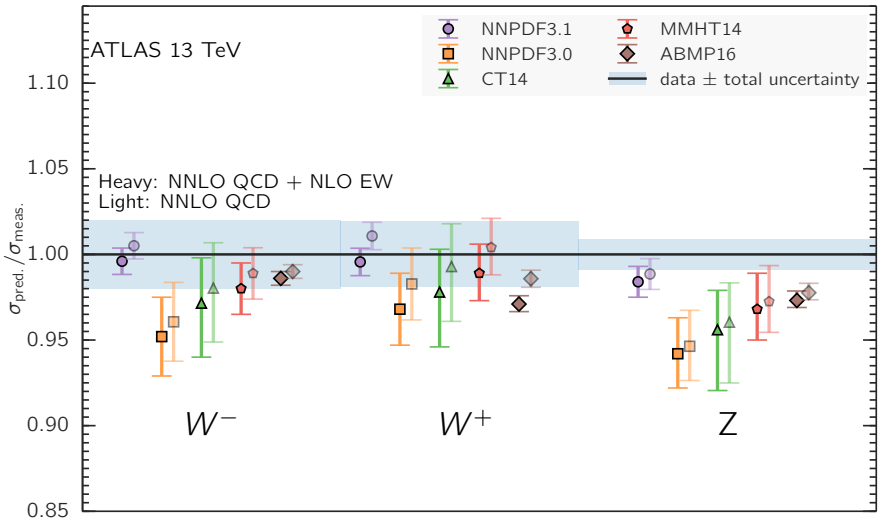

The central value of observables based on PDFs are obtained with:

$$\langle \mathcal{O}[f] \rangle = \frac{1}{N_{\rm rep}} \sum_{k=1}^{N_{\rm rep}} \mathcal{O}[f_k]$$

# W and Z production cross-sections at LHC 13 TeV

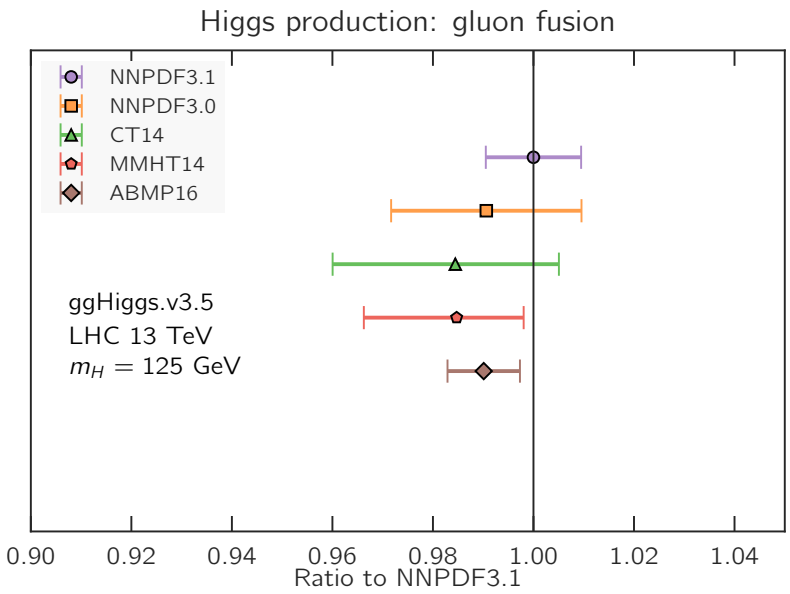


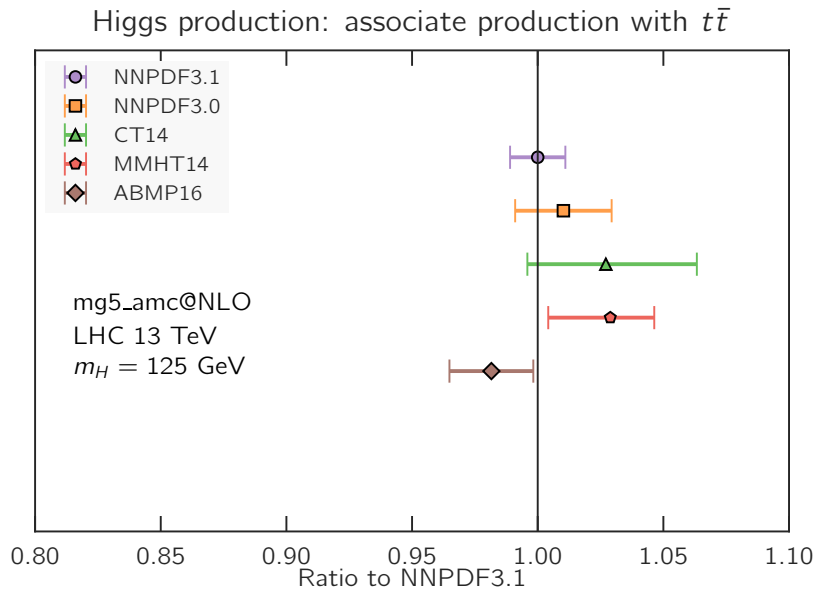



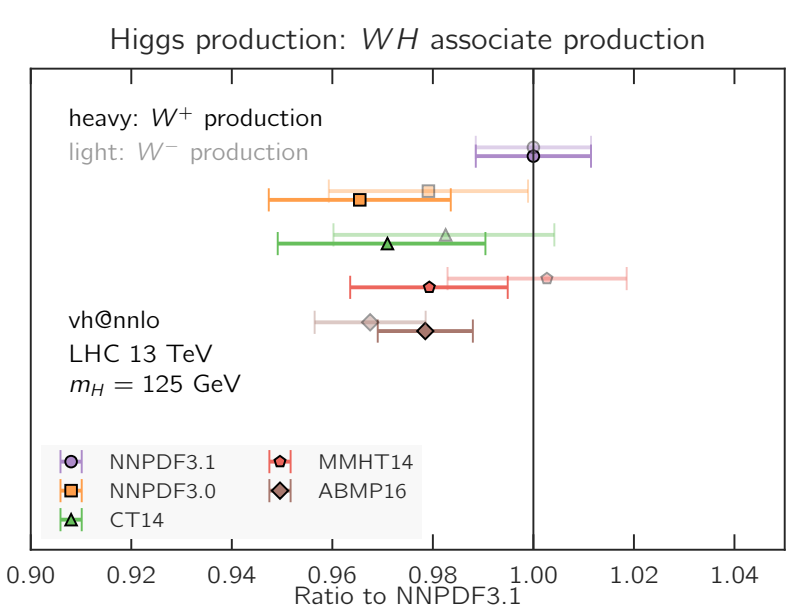



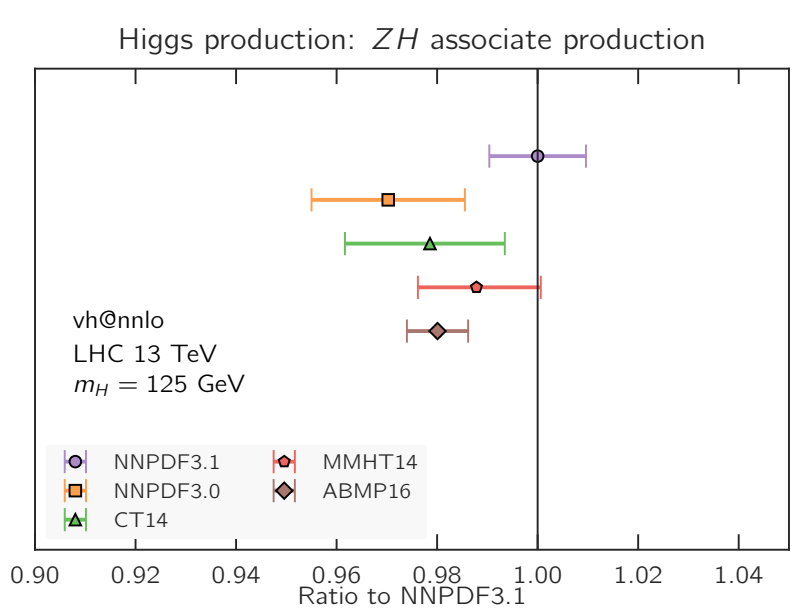

NNPDF3.1 have smaller PDF uncertainties than NNPDF3.0.

# Higgs production cross-sections



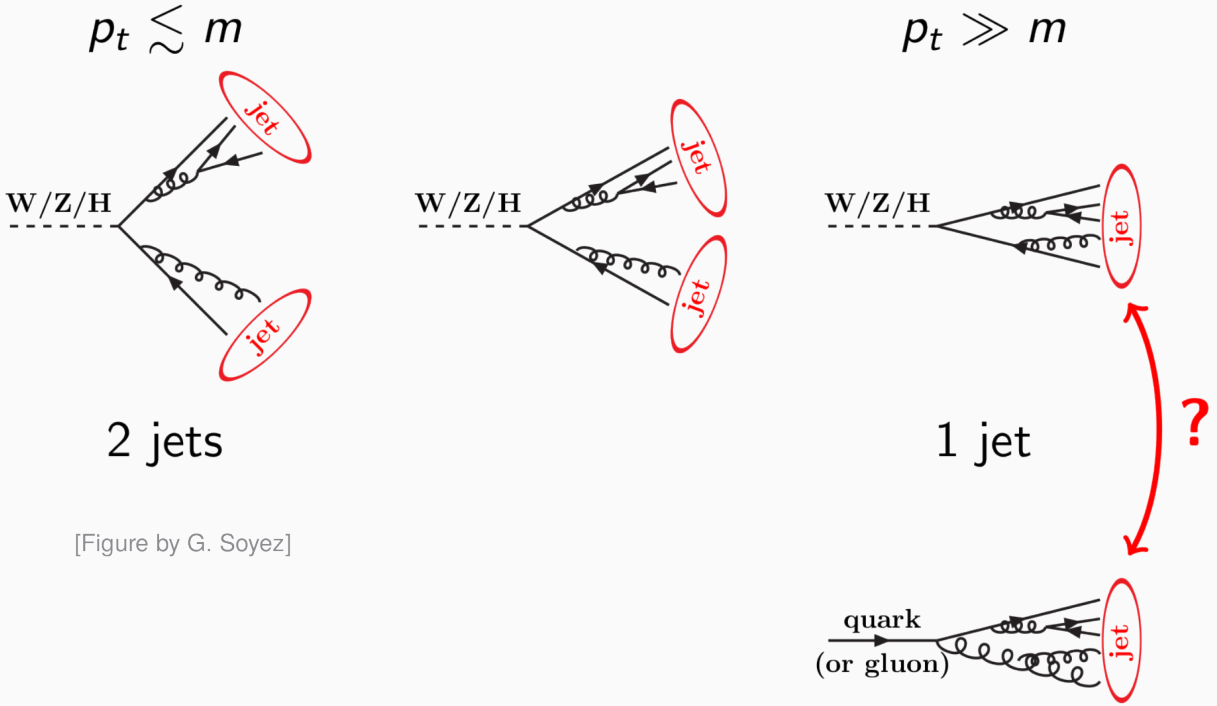



# ML in jet physics

# Jet grooming with reinforcement learning

Boosted objects at LHC energies, EW-scale particles ($W/Z/t...$) are often produced with $p_t \gg m$, leading to collimated decays.

**Problem:** hadronic decay products are often reconstructed into single jets. Identification of boosted objects by looking at the mass of the jet.




[Figure by G. Soyez]

Mass peak can be partly reconstructed by removing unassociated soft wide-angle radiation (grooming).

# Jet grooming with reinforcement learning

**[arXiv:1903.09644 - S.C. and Dreyer '19]**

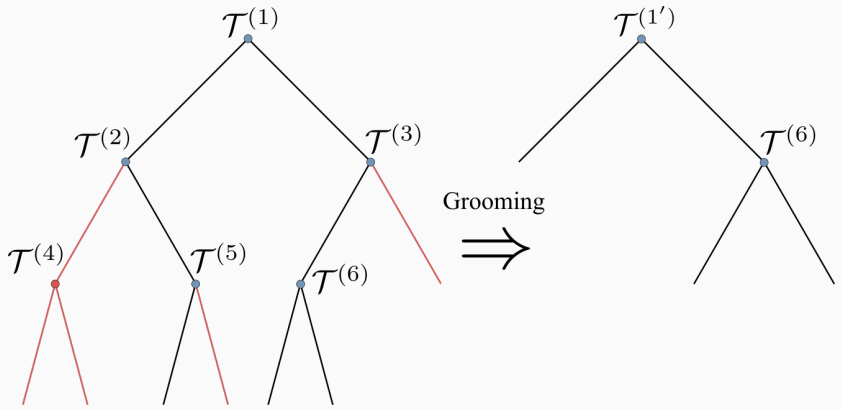
**ML idea:** use reinforcement learning to the problem of jet grooming.

- Cast jet as clustering tree where state of each note $\mathcal{T}^{(i)}$ is a tuple with kinematic information on splitting

$$s_t = \{z, \Delta_{ab}, \psi, m, k_t\}$$

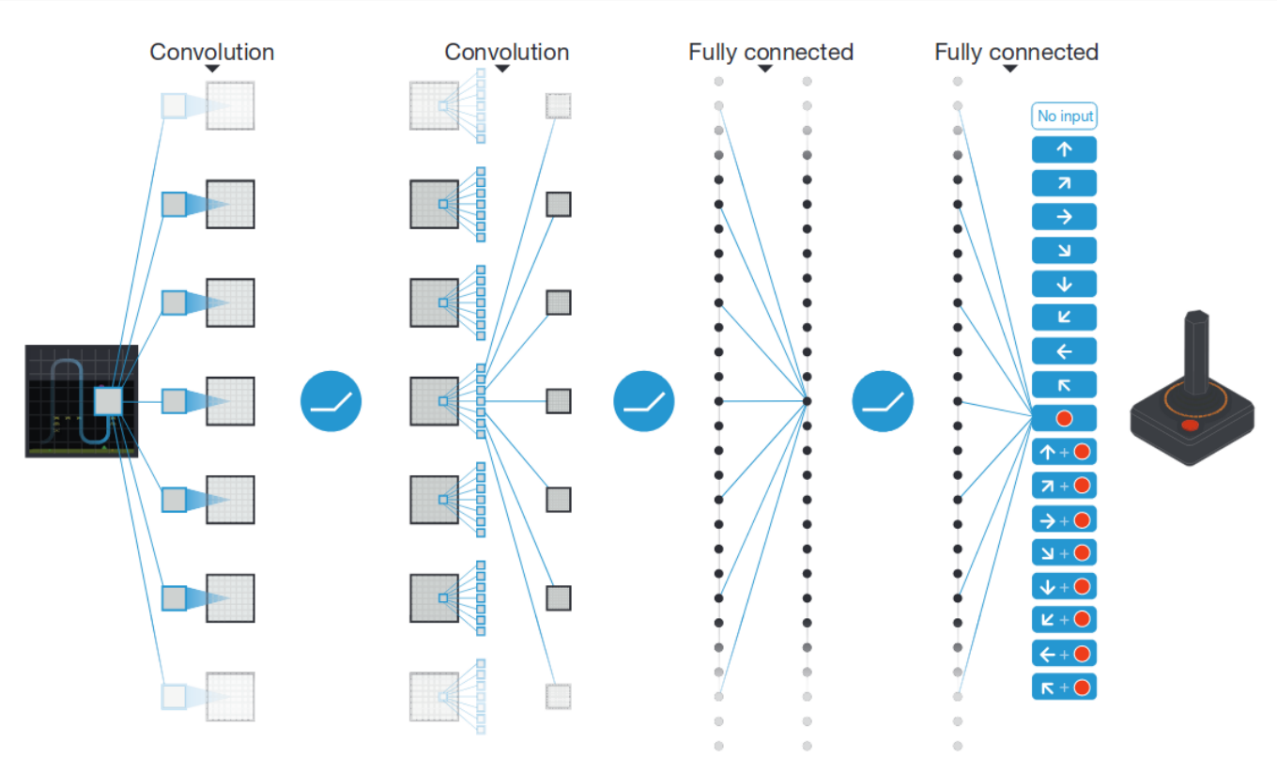
- Grooming algorithm defined as a function $\pi_g$ observing a state and returning an action $\{0, 1\}$ on the removal of the softer branch, *e.g.*

$$\pi_{\text{RSD}}(s_t) = \begin{cases} 0 & \text{if} \quad z > z_{\text{cut}} \left(\frac{\Delta_{ab}}{R_0}\right)^{\beta}, \\ 1 & \text{else} \end{cases}$$



# Jet grooming with reinforcement learning

We use a Deep Q-Network as a RL algorithm which uses a table of $Q(s, a)$, determining the next action as the one that maximizes $Q$.

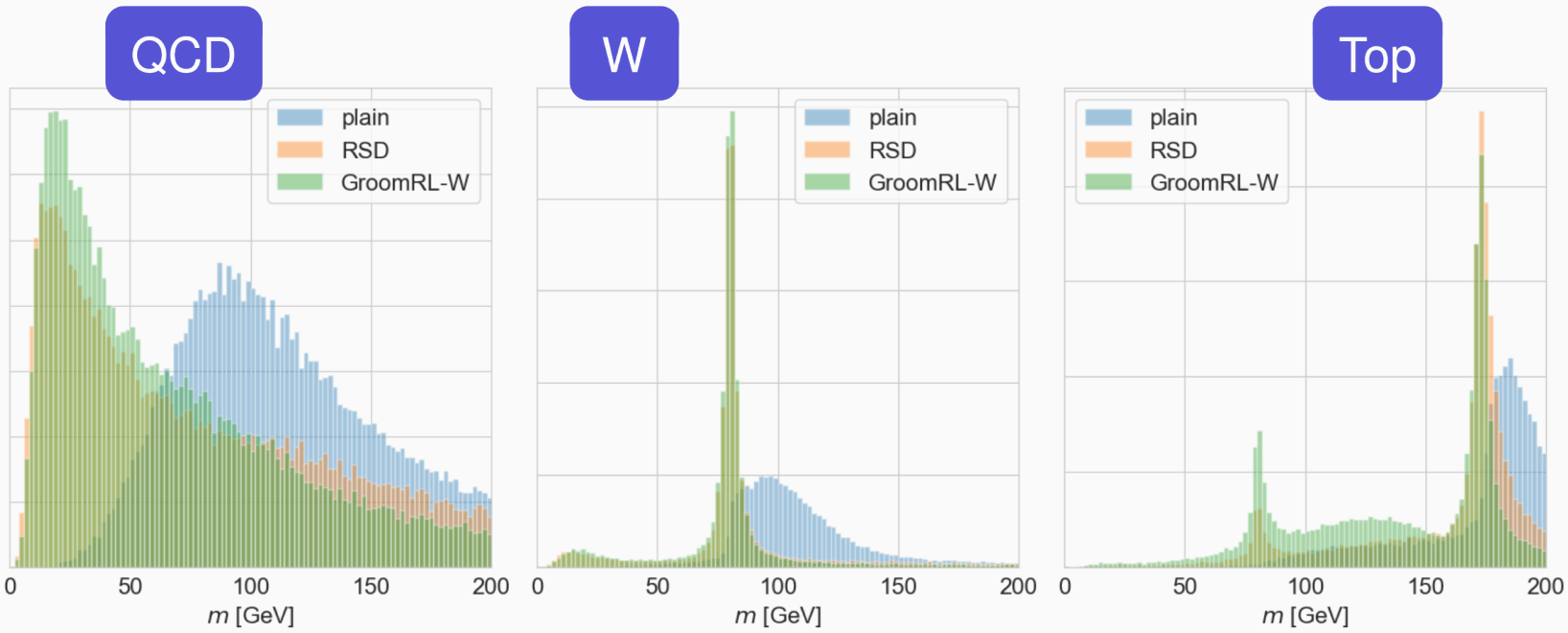



A NN is used to approximate the optimal action-value function:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + ... | s_t = s, a_t = a, \pi]$$

# Jet grooming with reinforcement learning

- To test the grooming algorithm derived from the DQN agent, we apply our groomer to three test samples: QCD, $W$ and Top jets.
- Improvement in jet mass resolution compared to heuristic methods (RSD)
- Algorithm performs well on data beyond its training range such as the top sample.



# ML in Monte Carlo simulation

# MINLO $t$-channel single-top plus jet

**[arXiv:1805.09855 - S.C., Frederix, Hamilton, Zanderighi '18]**

Use neural nets to adjust unknown higher-order resummation terms.

Use NLO-matched single-top + jet (STJ) from the POWHEG-MINLO formalism:

$$d\sigma_{\mathcal{M}} = \Delta(y_{12}) \left[ d\sigma^{\mathrm{STJ}}_{\mathrm{NLO}} - \Delta(y_{12})|_{\bar{\alpha}_S} d\sigma^{\mathrm{STJ}}_{\mathrm{LO}} \right]$$

# MINLO $t$-channel single-top plus jet

**[arXiv:1805.09855 - S.C., Frederix, Hamilton, Zanderighi '18]**

Use neural nets to adjust unknown higher-order resummation terms.

Use NLO-matched single-top + jet (STJ) from the POWHEG-MINLO formalism:

$$d\sigma_{\mathcal{M}} = \Delta(y_{12}) \left[ d\sigma^{\mathrm{STJ}}_{\mathrm{NLO}} - \Delta(y_{12})|_{\bar{\alpha}_S} d\sigma^{\mathrm{STJ}}_{\mathrm{LO}} \right]$$

**Advantage:** enhance fixed-order calculation with matched NLL Sudakov form factor.

# MINLO $t$-channel single-top plus jet

**[arXiv:1805.09855 - S.C., Frederix, Hamilton, Zanderighi '18]**

Use neural nets to adjust unknown higher-order resummation terms.

Use NLO-matched single-top + jet (STJ) from the POWHEG-MINLO formalism:

$$d\sigma_{\mathcal{M}} = \Delta(y_{12}) \left[ d\sigma^{\mathrm{STJ}}_{\mathrm{NLO}} - \Delta(y_{12})|_{\bar{\alpha}_S} d\sigma^{\mathrm{STJ}}_{\mathrm{LO}} \right]$$

**Advantage:** enhance fixed-order calculation with matched NLL Sudakov form factor.

**Issue:** this spoils the NLO accuracy of ST (single-top observables).

# MINLO $t$-channel single-top plus jet

**[arXiv:1805.09855 - S.C., Frederix, Hamilton, Zanderighi '18]**

Use neural nets to adjust unknown higher-order resummation terms.

Use NLO-matched single-top + jet (STJ) from the POWHEG-MINLO formalism:

$$d\sigma_{\mathcal{M}} = \Delta(y_{12}) \left[ d\sigma^{\mathrm{STJ}}_{\mathrm{NLO}} - \Delta(y_{12})|_{\bar{\alpha}_S} d\sigma^{\mathrm{STJ}}_{\mathrm{LO}} \right]$$

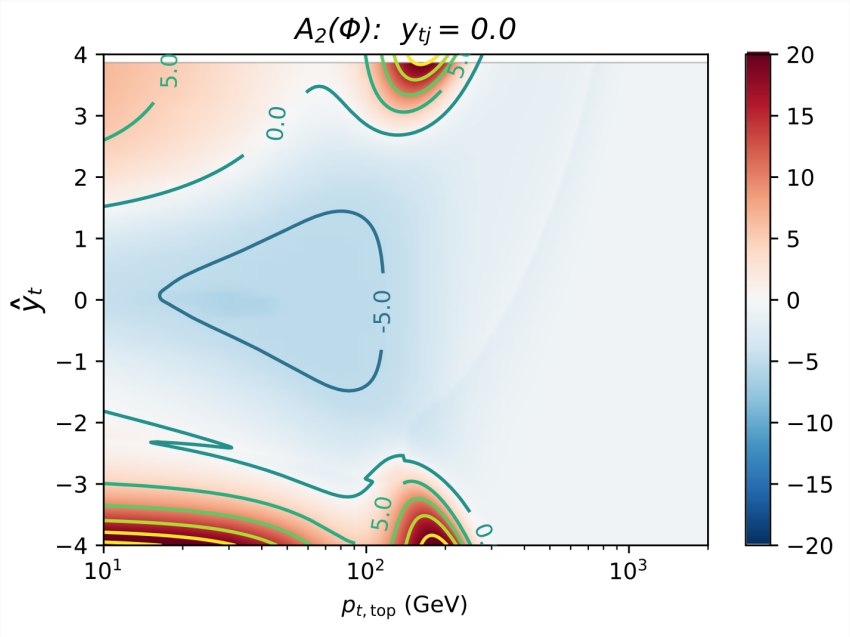
**Advantage:** enhance fixed-order calculation with matched NLL Sudakov form factor.

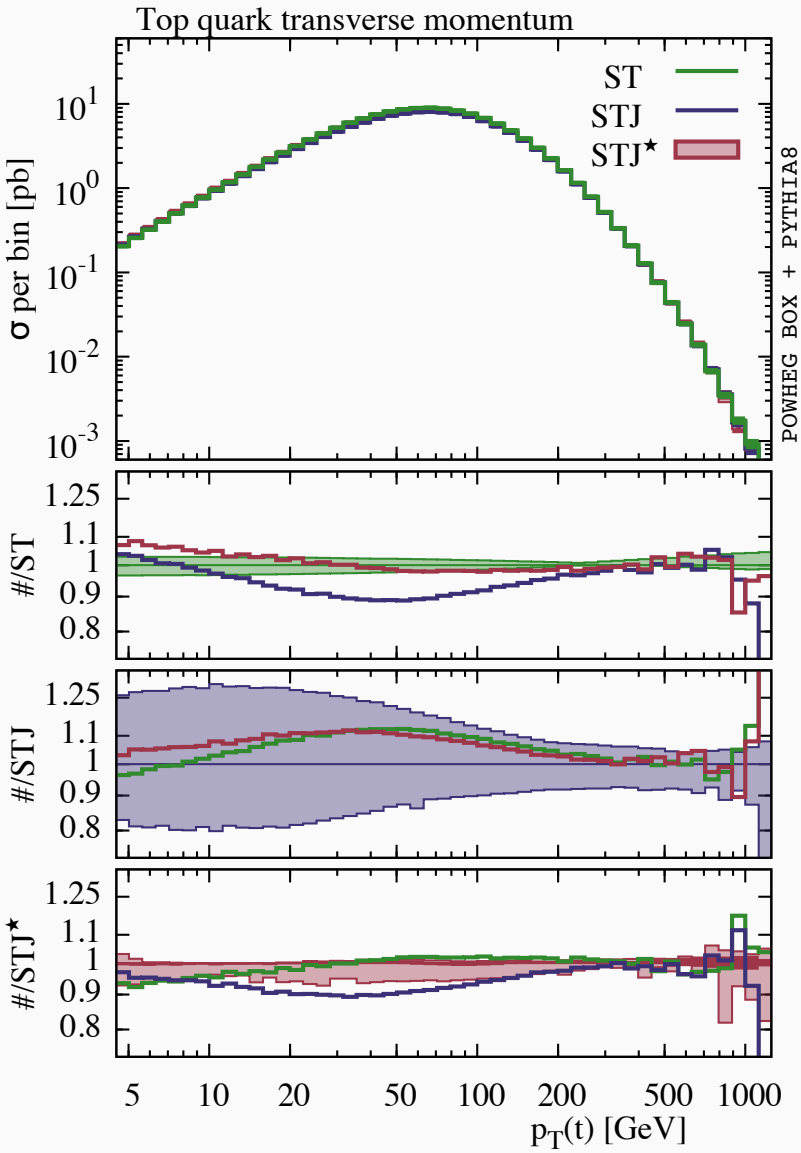
**Issue:** this spoils the NLO accuracy of ST (single-top observables).

**Solution:** fix at NNLL, fit $A_2$ with a Neural Network-based tuning of degrees of freedom, and test universality at 8 TeV.

$$\ln \delta\Delta(y_{12}) = -2 \int_{y_{12}}^{Q^2_{bt}} \frac{dq^2}{q^2} \bar{\alpha}_S^2 \mathcal{A}_2(\Phi) \ln \frac{Q^2_{bt}}{q^2}$$

# MINLO $t$-channel single-top plus jet



**Thanks for your attention!**