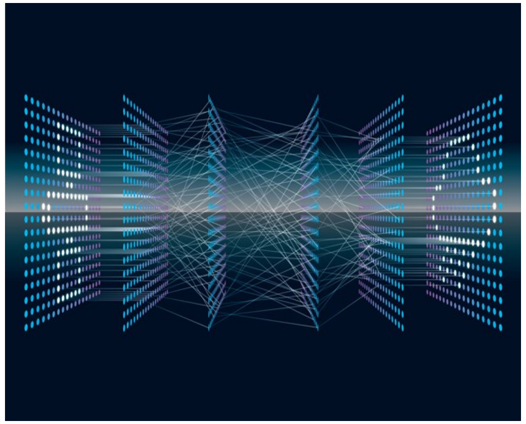
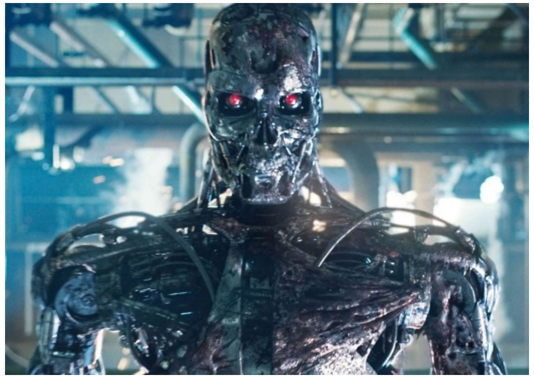
STAND WITH UKRAINE

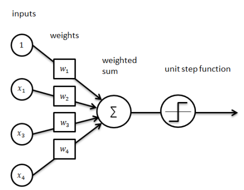
Future of Machine Learning in HEP

TOMASZ SZUMLAK

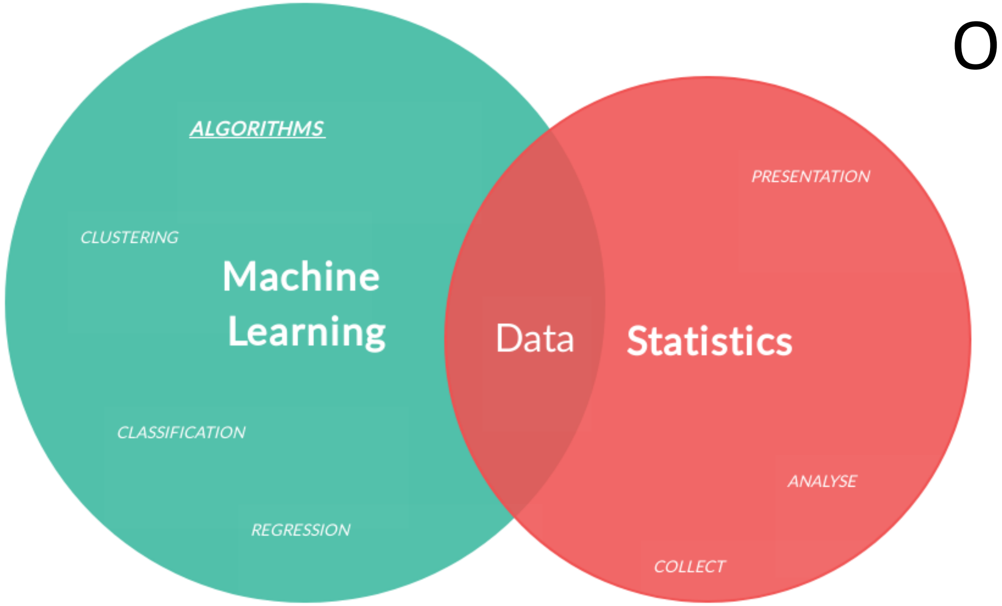




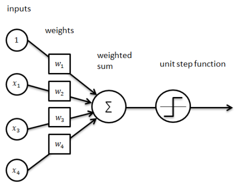
BEACH 2022, 05 – 10.06.2022, KRAKÓW



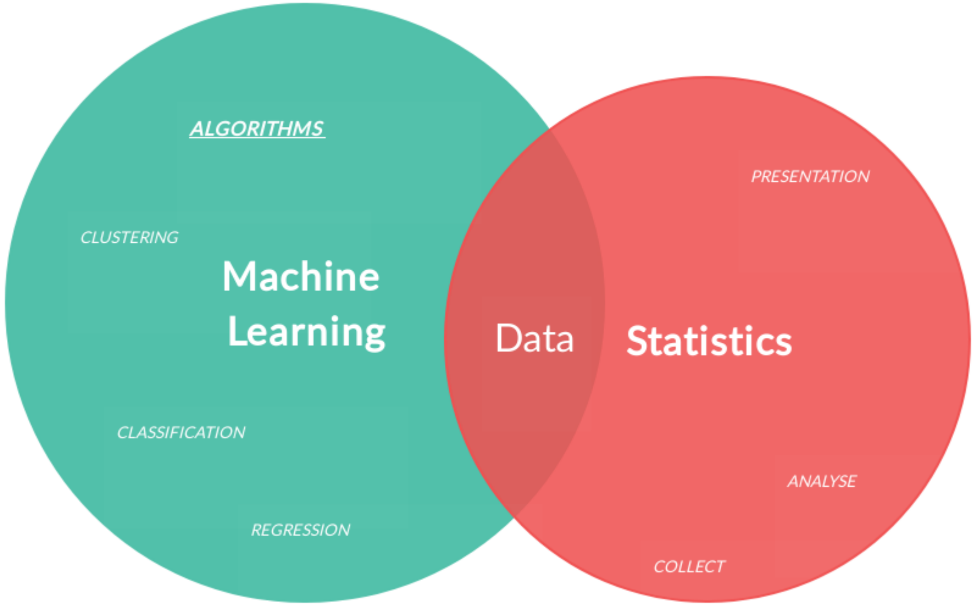
Setting the scene



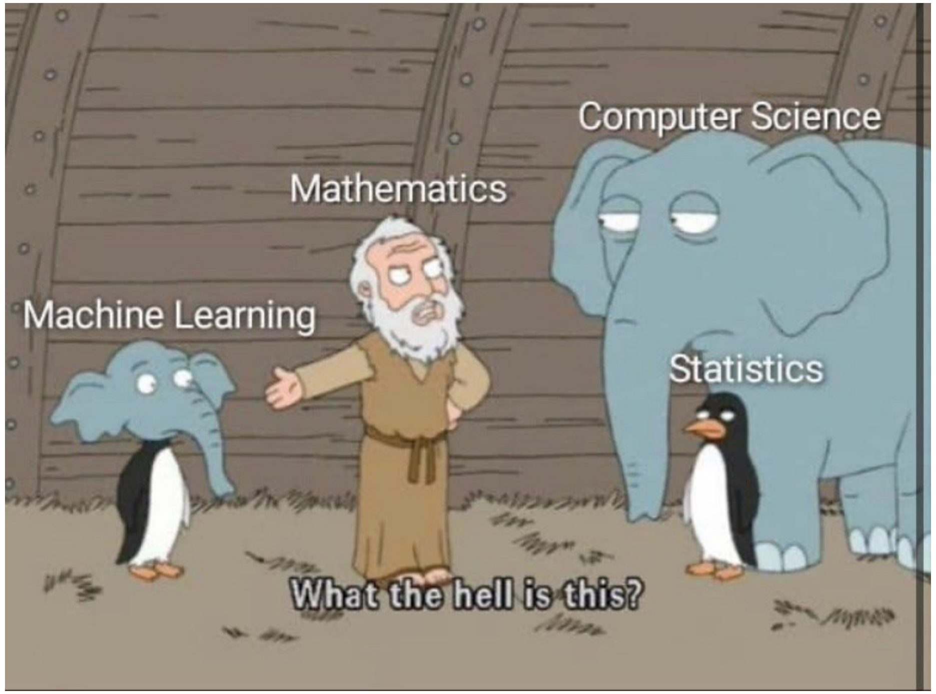
On a serious note …

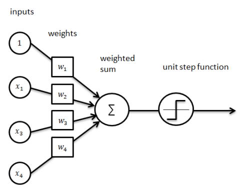


Setting the scene



… and not so serious



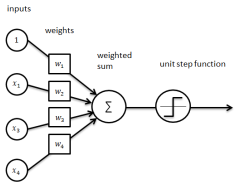


Outline

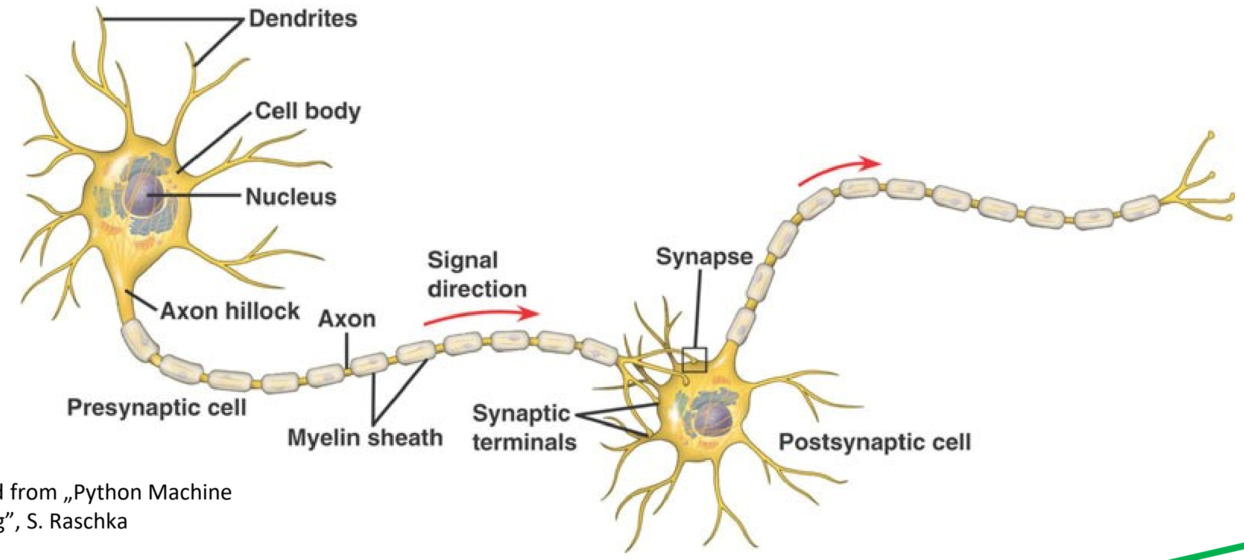
- ❑ What is ML all about
- ❑ Loss and the crucial bit
- ❑ Popular models
- ❑ Current landscape
- ❑ Selected (subjective) HEP solutions
- ❑ The biggest challenges for the future

ML: New revolution, a.k.a. electricity 2.0

- We are living in interesting times – data come in **abundance** and ability to **process** them and **gain knowledge** is of great value: data is **very precious resource** (like iron, gold or water)
- We want to process the data fast and in a robust way
- Machine Learning (ML), which is a part of data mining business, allows us to use **computer algorithms** to **make sense of data** or to turn them into knowledge
- What is more exciting we have a lot of **open source** libraries that implements the most sophisticated algorithms on the market and **they are free**!
- **Convergence of technologies made it possible!**

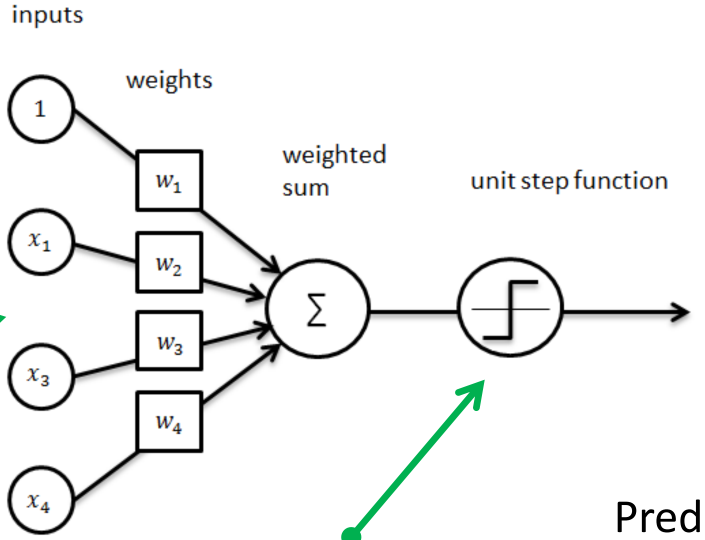


Artificial neuron or perceptron



Adapted from „Python Machine Learning", S. Raschka

- **1943** with McCullock-Pitts **neuron model**
- **Motivated by biological studies**

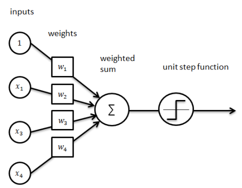


- **Perceptron equation**

$$z^{(i)} = w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_k x_k^{(i)} = \sum_{j=1}^{j=k} w_j x_j^{(i)} = \vec{w}^T \vec{x}^{(i)}$$

$$\phi(z) = \begin{cases} +1 \;\; if \;\; z \geq \;\; \theta \\ -1 \;\; if \;\; z < \theta \end{cases}$$

Predefined threshold



The algorithm

- The perceptron algorithm, then goes like that:
  - **Initialise the weights vector to 0 or „something small”**
  - **For each training data sample $\vec{x}^{(i)}$ do:**
    - **Get the output value (class label) $\tilde{y}^{(i)}$, using the unit step function**
    - **Update the weights accordingly (update concerns all the weights in one go)**
- We can write

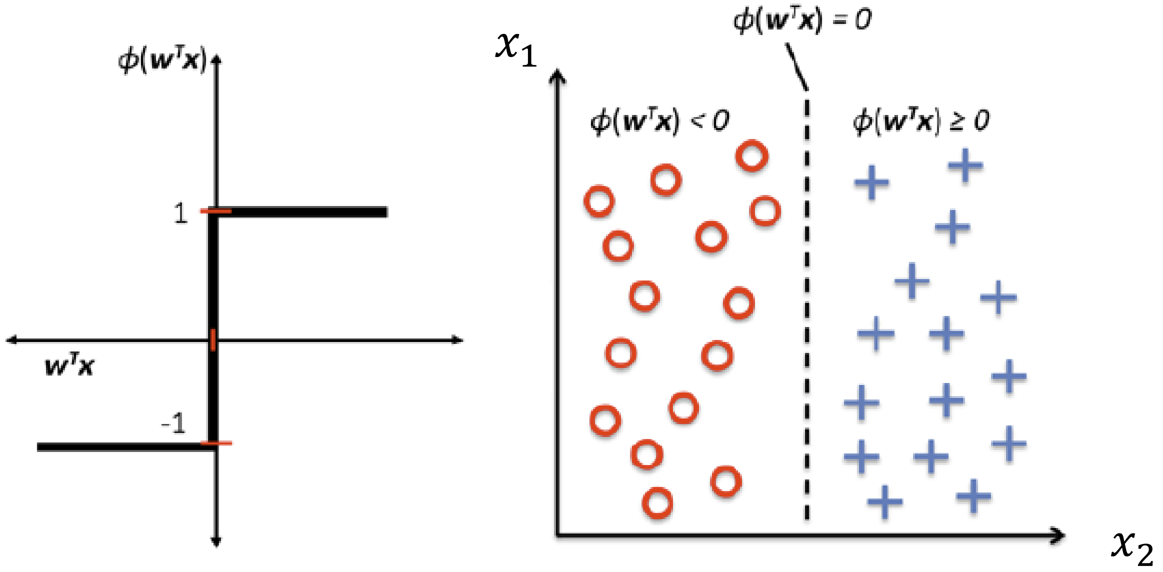
$$w_j = w_j + \Delta w_j$$

$$\Delta \boldsymbol{w_j} = \boldsymbol{\eta} \cdot \left(\boldsymbol{y}^{(\boldsymbol{i})} - \tilde{\boldsymbol{y}}^{(\boldsymbol{i})}\right) \cdot \boldsymbol{x}_{\boldsymbol{j}}^{(\boldsymbol{i})}$$

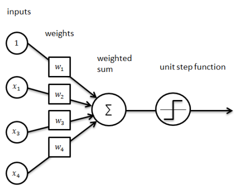
- The second formula is called **perceptron learning rule**, and the $\boldsymbol{\eta}$ is called the learning rate (just a number between 0 and 1)

Outcome

- For classification tasks we can provide an intuitive representation of the training outcome

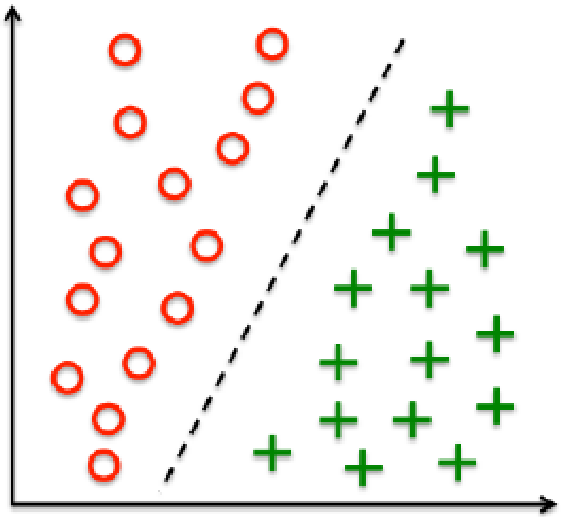


Adapted from „Python Machine Learning", S. Raschka



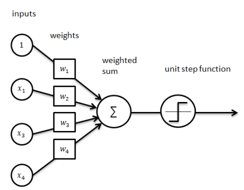
„Magic” is here

- The idea of a binary classification can be understood using the following example: say, we have given 30 training samples – half of them is **negative** (noise) and half positive (signal)



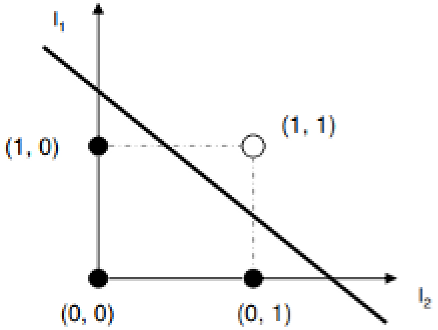
- 2D data set – each data instance has two values (x_1, x_2) associated with it
- Using them separately is going to yield poor results!
- Try to imagine we project the data on the respective axes

- Our algorithm must learn a rule to separate these two classes and classify a new instance into one of these classes given values x_1, x_2
- This rule is also called **decision boundary** (black dashed line)

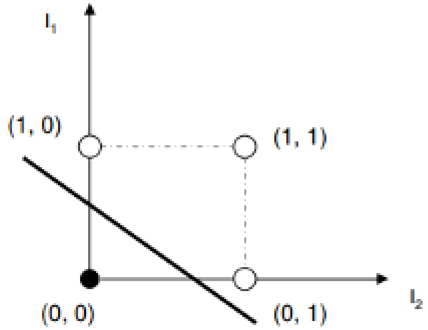


Dark ages…

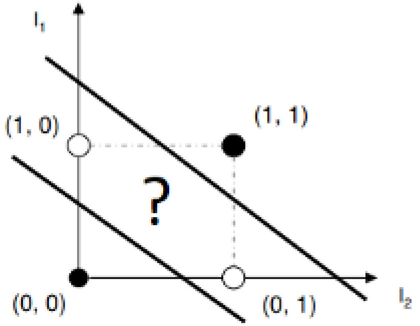
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

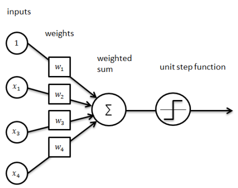


OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

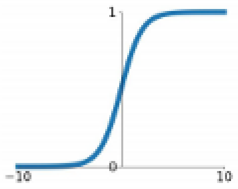




Non-linear differentiable functions

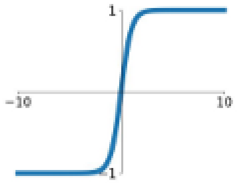
Sigmoid

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



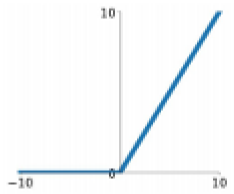
tanh

$\tanh(x)$



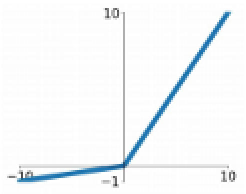
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

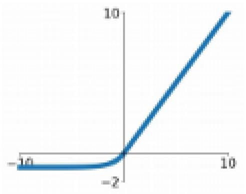


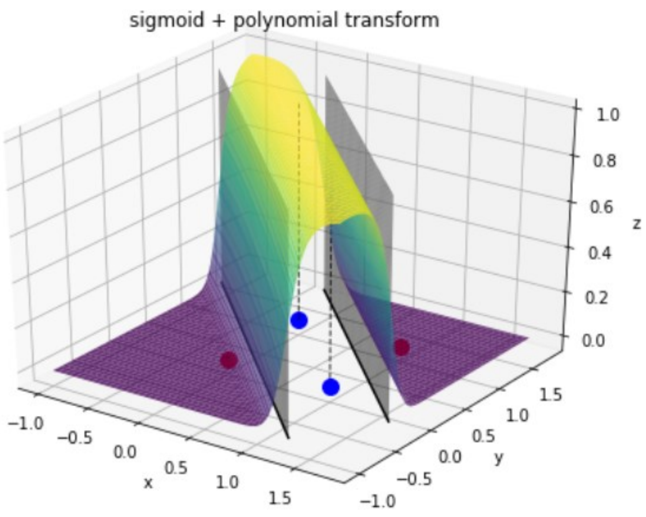
Maxout

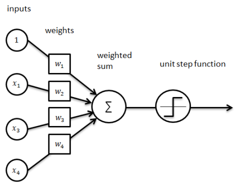
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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







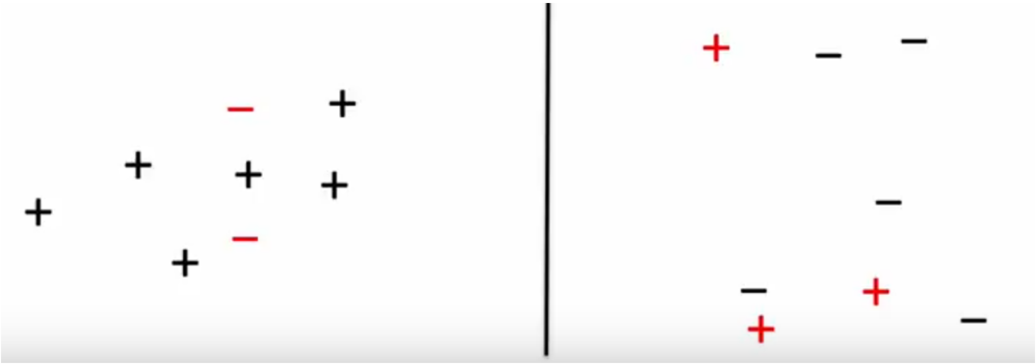
Loss function (I)

- **In practice we need to have a very good handle on the performance of our model**
- Or, in other words we **need to have means to penalise the model** if it performs poorly and reward if it does good

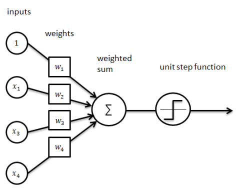
$\mathcal{M}(\vec{x}) = 0$

$\mathcal{M}(\vec{x}) > 0$ $\mathcal{M}(\vec{x}) < 0$

Red points are misclassified



$$\mathcal{L}(y_i, \mathcal{M}(\vec{x}_i)) = \frac{1}{n}\sum_i [y_i \neq sign(\mathcal{M}(\vec{x}_i))]$$



Loss function (II)

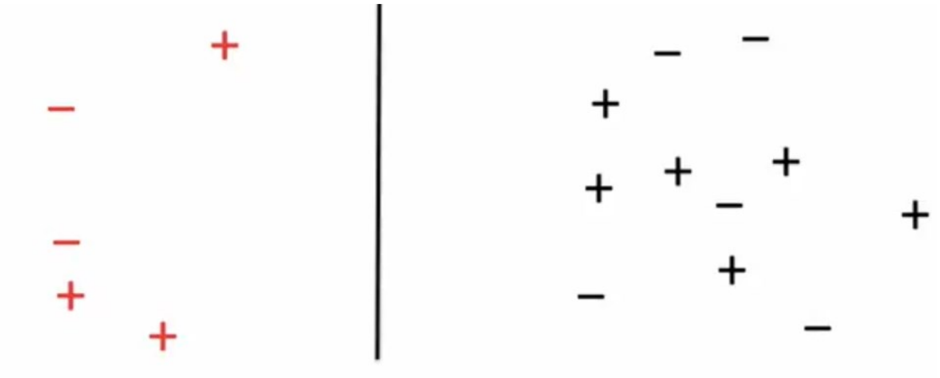
- Let’s create „an universal” formula for the loss function

The opposite signs

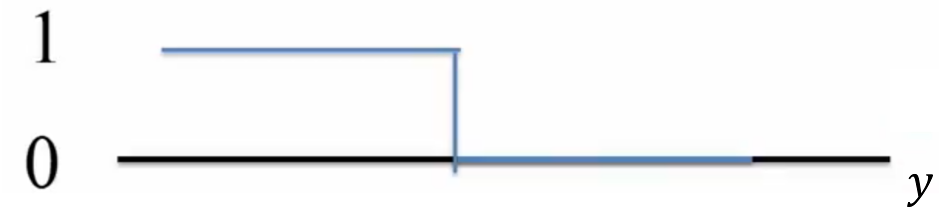
$y \cdot \mathcal{M}(\vec{x}) < 0$

$y \cdot \mathcal{M}(\vec{x}) > 0$

The same signs

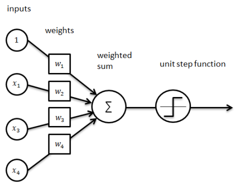


$$\mathcal{L} = \frac{1}{n}\sum_i 1_{[y\cdot\mathcal{M}(\vec{x}_i)<0]}$$



$y \cdot \mathcal{M}(\vec{x})$

Max penalty each time!



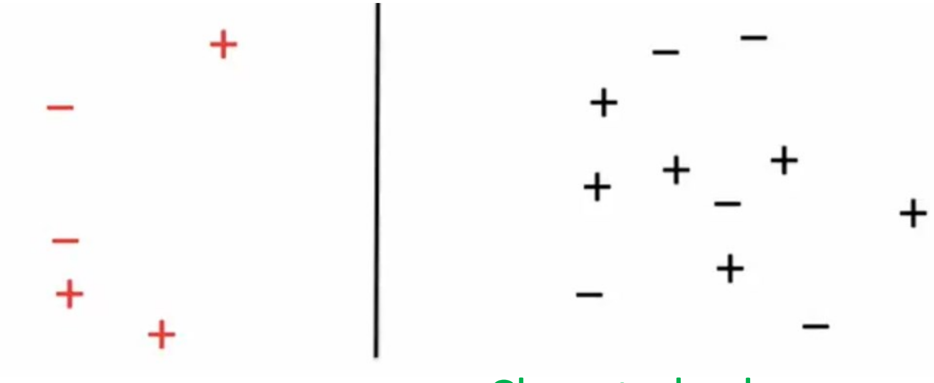
Loss function (III)

- In theory such loss function is very powerfull, but **in practice we cannot optimise such expression in any easy way** and on top of this it **has no sensitivity on how bad the decision was**, i.e., each time the penalty is maximal

$y \cdot \mathcal{M}(\vec{x}) < 0$

$y \cdot \mathcal{M}(\vec{x}) > 0$

Very bad decision

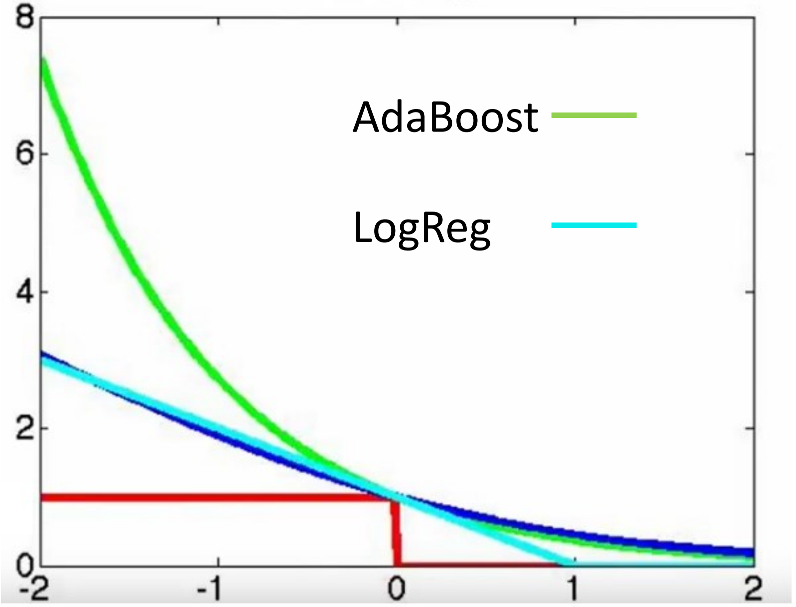


Very good decision

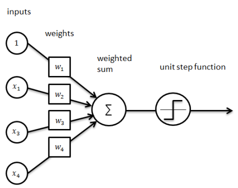
Close to good

Close to bad

$\mathcal{L}$



$y \cdot \mathcal{M}(\vec{x})$

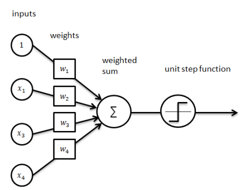


Loss function (IV)

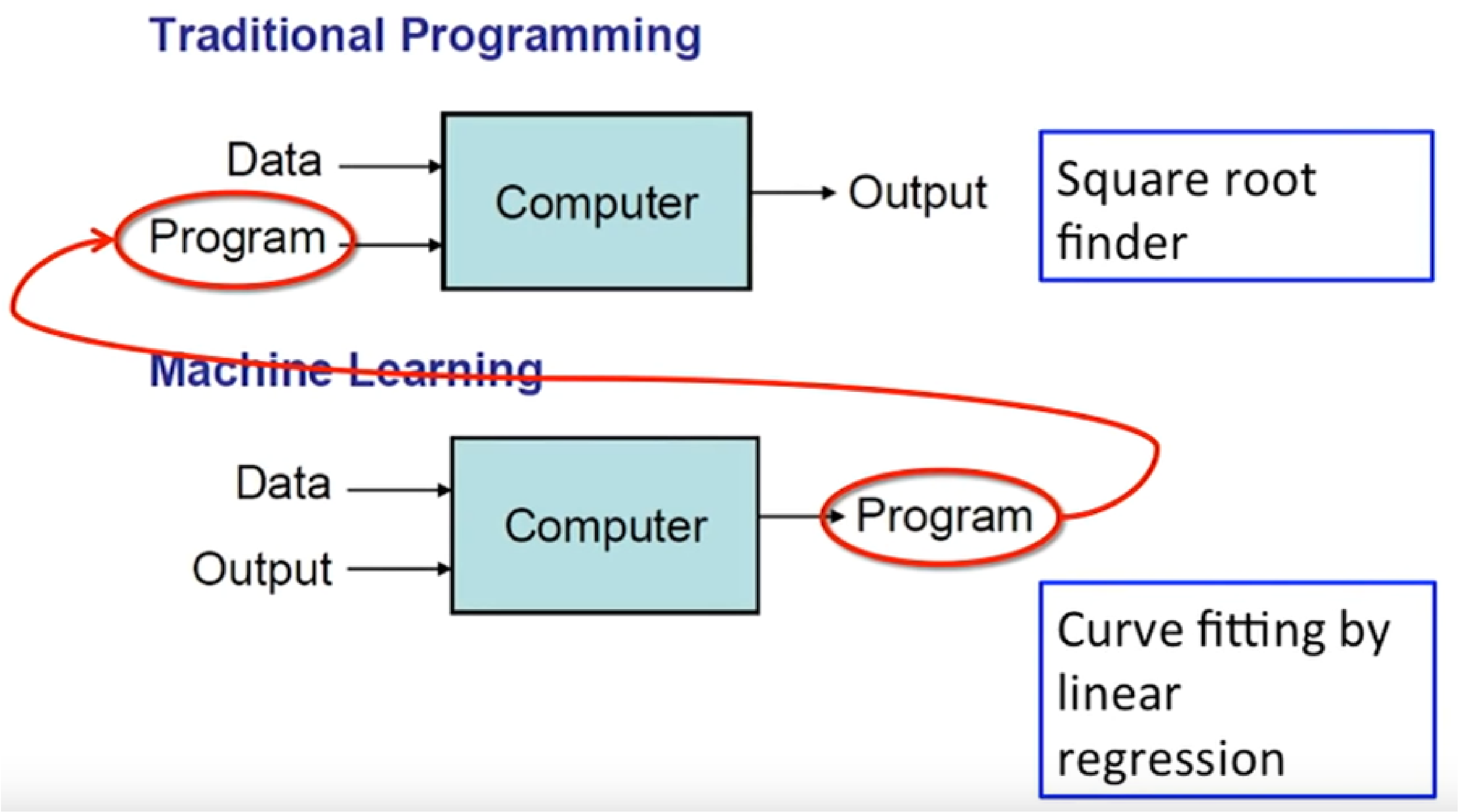
- There are some **tantalising facts regarding the loss function**: the whole training process depends on the way we measure its performance – more aggressive approach may be more beneficial, it may determine **how long the training process take and if it will be successful at all – how interesting**

- Different loss functions determine upper limits w.r.t $1_{[y \cdot \mathcal{M}(\vec{x}_i)<0]}$ one:

$$\mathcal{L}\big(y_i, \mathcal{M}(\vec{x}_i)\big) = \frac{1}{n}\sum\nolimits_i \big[y_i \neq sign\big(\mathcal{M}(\vec{x}_i)\big)\big] = \frac{1}{n}\sum\nolimits_i 1_{[y \cdot \mathcal{M}(\vec{x}_i)<0]} \leq \frac{1}{n}\sum\nolimits_i f_{\mathcal{M}}\big(y \cdot \mathcal{M}(\vec{x}_i)\big)$$


inputs
weights
weighted sum
unit step function

What is ML?


Traditional Programming
Data
Program
Computer
Output
Square root finder
Machine Learning
Data
Output
Computer
Program
Curve fitting by linear regression

ML pipeline (I)

- For our purposes we can define a ML pipeline (ML-P) or ML algorithm (ML-A) as a composite object consisting of:
  - **data set(s)**, we look for patterns/knowledge here
  - **a model**
  - **an optimising algorithm** (fitting/weights change)
  - **a loss function**
- ML-A is able to gain knowledge based on data
  - The pipeline components are: experience (E), class of tasks (T) and performance metric (PM)

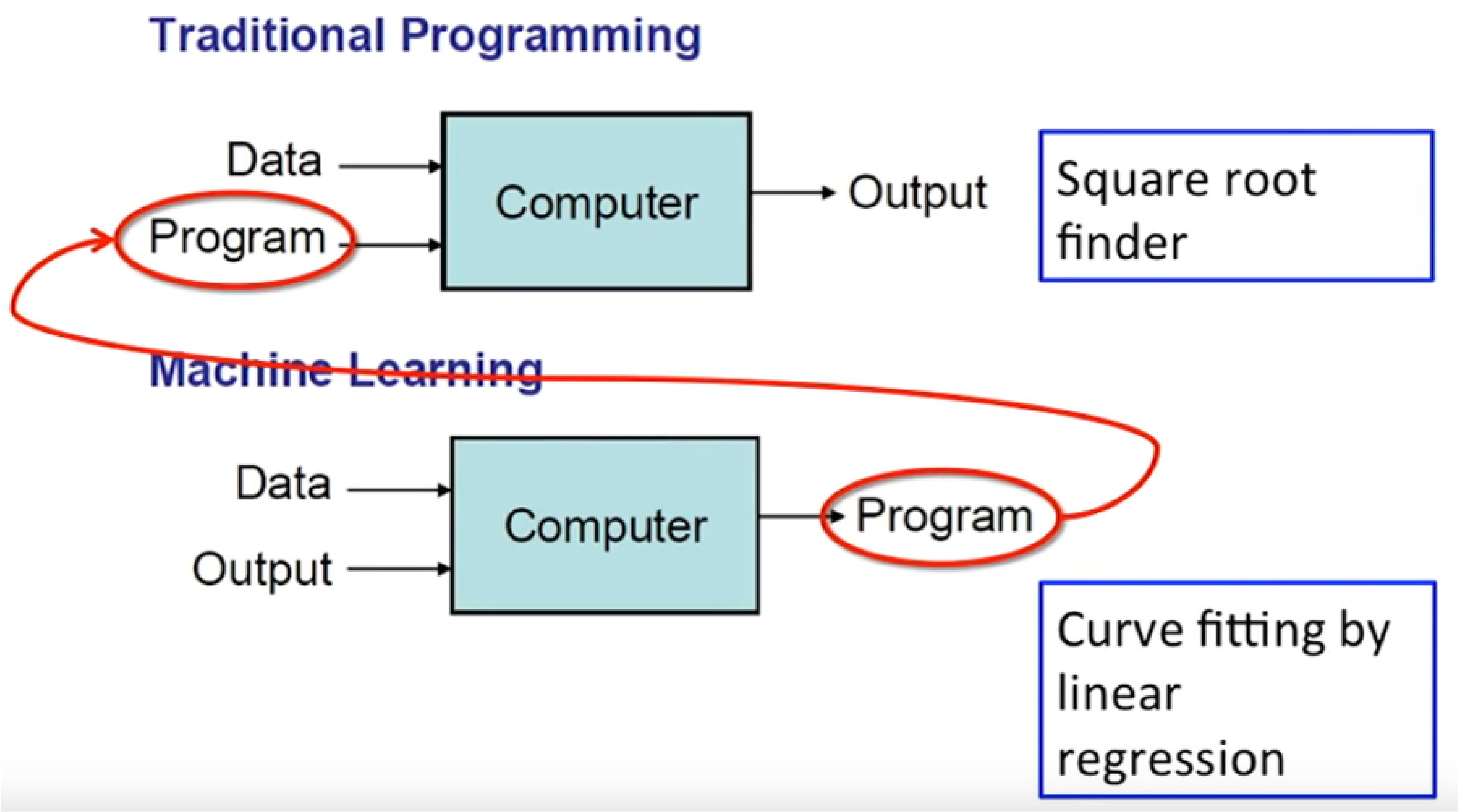
ML pipeline (II)

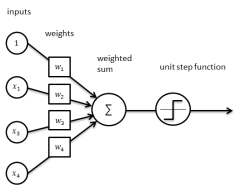
- **A general statement on ML (Mitchell): a computer program learns based on gained experience (E) for a particular class of tasks (T), the learning process is checked by the performance metric (PM)**
- So, if we have a binary classification task its performance should increase when we expose the model to more and more data. More data – more experience
- **Data quality and representation is critically important**

Selected Tasks

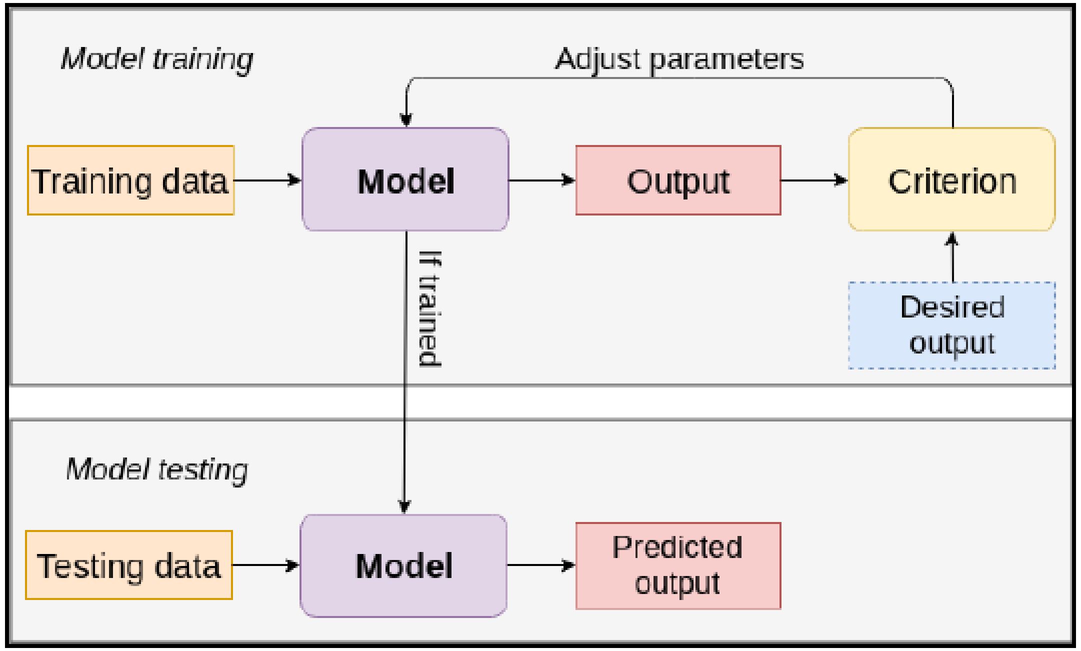
- Classification, $f\colon\ \mathbb{R}^n \to \{1,2,\dots,k\}, \mathrm{y} = \mathrm{f}(\vec{\boldsymbol{x}})$ (data labelling)
- Classification with missing features, $f_i\colon\ \mathbb{R}^n \to \{1,2,\dots,k\}$
- Regression, $f\colon\ \mathbb{R}^n \to \mathbb{R}$
- Natural Language Processing
- Anomaly detection
- Sampling (generative models), $f\colon \mathbb{R} \to \mathbb{R}^n$
- Denoising, $\tilde{\vec{\boldsymbol{x}}} \to \vec{\boldsymbol{x}}\colon \mathrm{p}\left(\vec{\boldsymbol{x}} \middle| \tilde{\vec{\boldsymbol{x}}}\right)$
- Estimation of P.D.F.s, $p_{Model}(\vec{\boldsymbol{x}})\colon\ \mathbb{R}^n \to \mathbb{R}$

Classical programming vs. ML

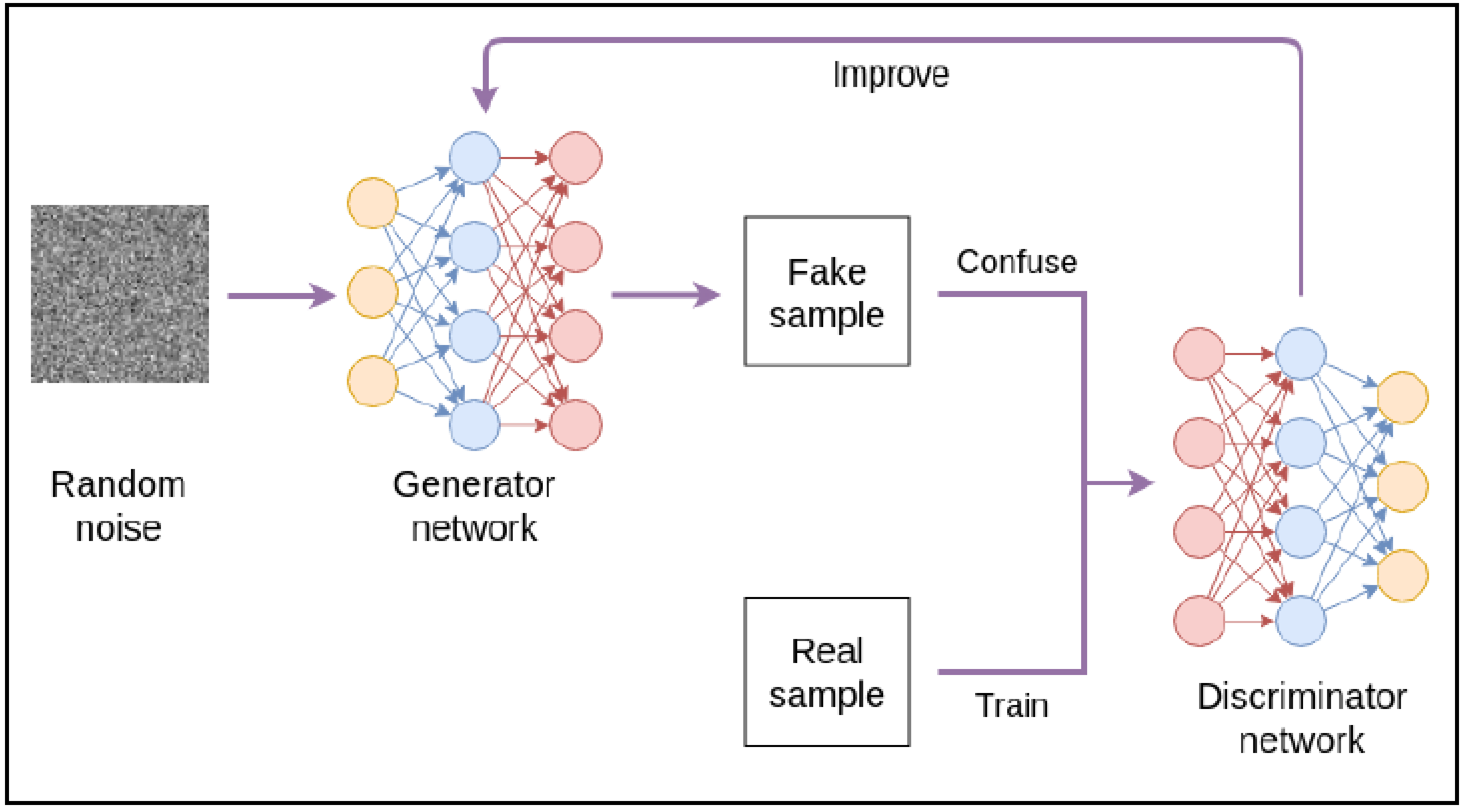




GAN – Generative Adversarial Networks



GAN – Generative Adversarial Networks



GAN optimisation rules

- Let set $\mathcal{G}$ and $\mathcal{D}$ to represent the generator and discriminator models respectively, the performance function is $\mathcal{V}$. The optimisation objective can be written as follow:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{V}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\vec{x}}[log\mathcal{D}(\vec{x})] + \mathbb{E}_{\vec{x}^*}\left[log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$

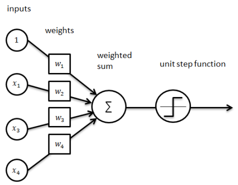
- Here: $\vec{x}$ - real samples, $\vec{x}^* = \mathcal{G}(z)$ - generated samples (z represents noise), $\mathbb{E}_{\vec{x}}[f]$ is the average value of any function over the sample space

- Model $\mathcal{D}$ should maximise the „good" prediction for the real sample - we are looking for the max – gradient ascent update rule

$$\vec{\theta}_{\mathcal{D}} \leftarrow \vec{\theta}_{\mathcal{D}} + r \cdot \frac{1}{m} \nabla_{\vec{\theta}_{\mathcal{D}}} \sum_{i/1}^{i/m} \left[log\mathcal{D}(\vec{x}) + log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$

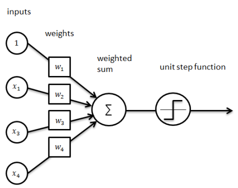
- Model $\mathcal{G}$ must trick the discriminator, thus, it minimise the $1 - \mathcal{D}(\vec{x}^*) = 1 - \mathcal{D}\left(\mathcal{G}(z)\right)$

$$\vec{\theta}_{\mathcal{G}} \leftarrow \vec{\theta}_{\mathcal{G}} - r \cdot \frac{1}{m} \nabla_{\vec{\theta}_{\mathcal{G}}} \sum_{i/1}^{i/m} \left[log\left(1 - \mathcal{D}(\vec{x}^*)\right)\right]$$

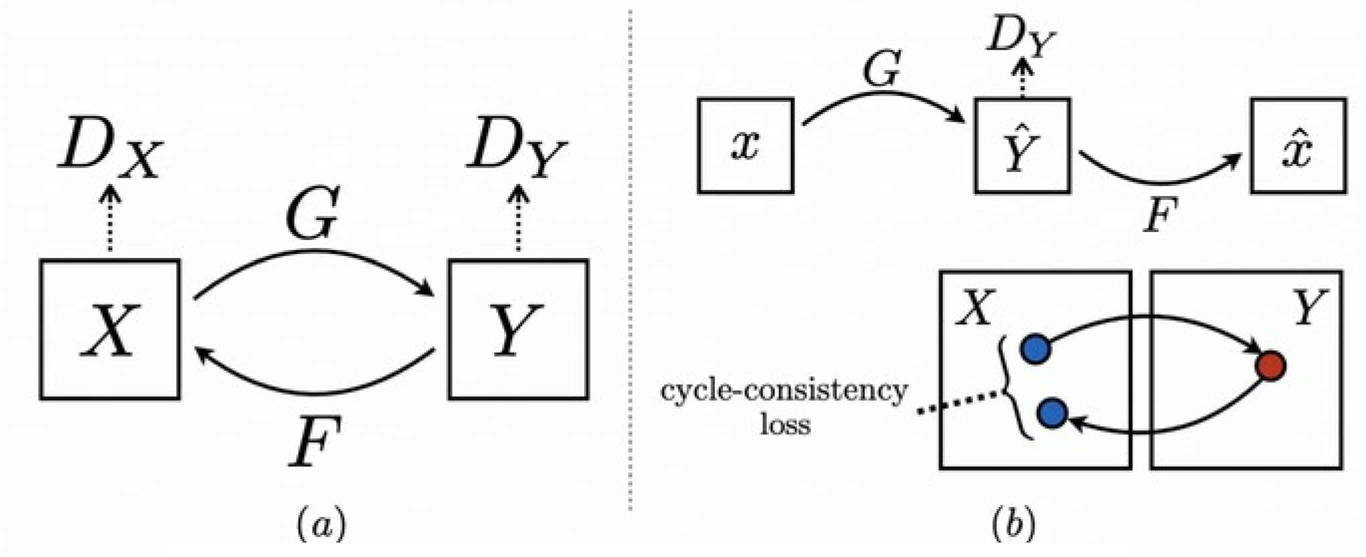


ML GEMS (I) - GANs

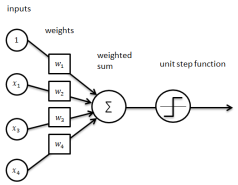
https://syncedreview.com/2019/02/09/nvidia-open-sources-hyper-realistic-face-generator-stylegan/



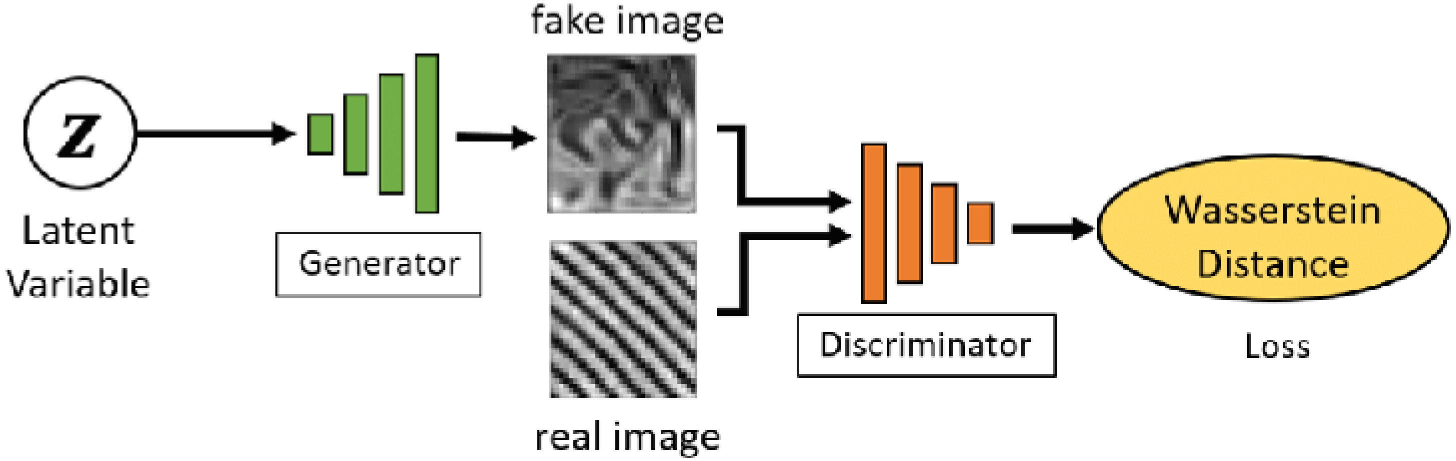
CycleGAN

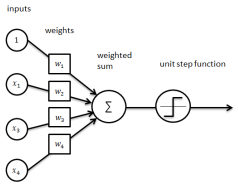


Real Generated Reconstructed

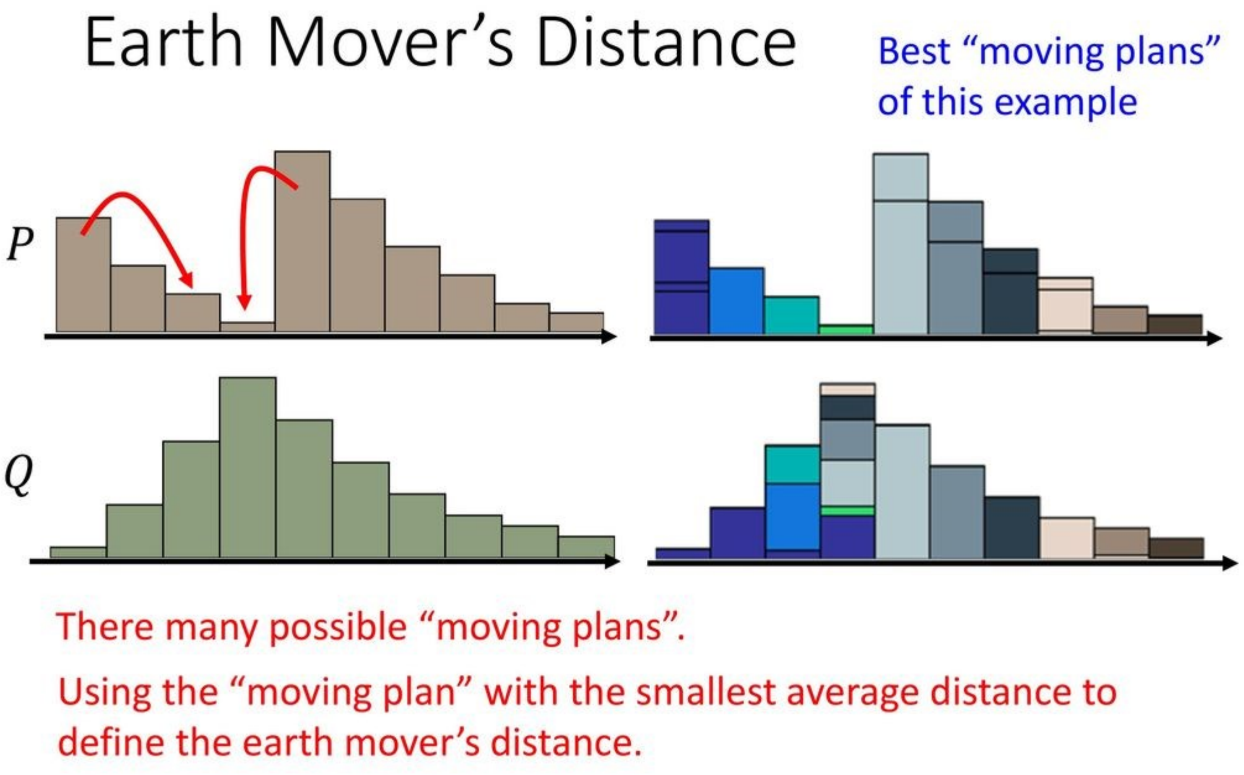


WGAN – Wasserstein GAN

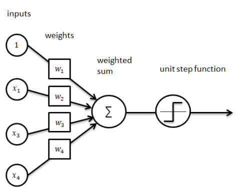




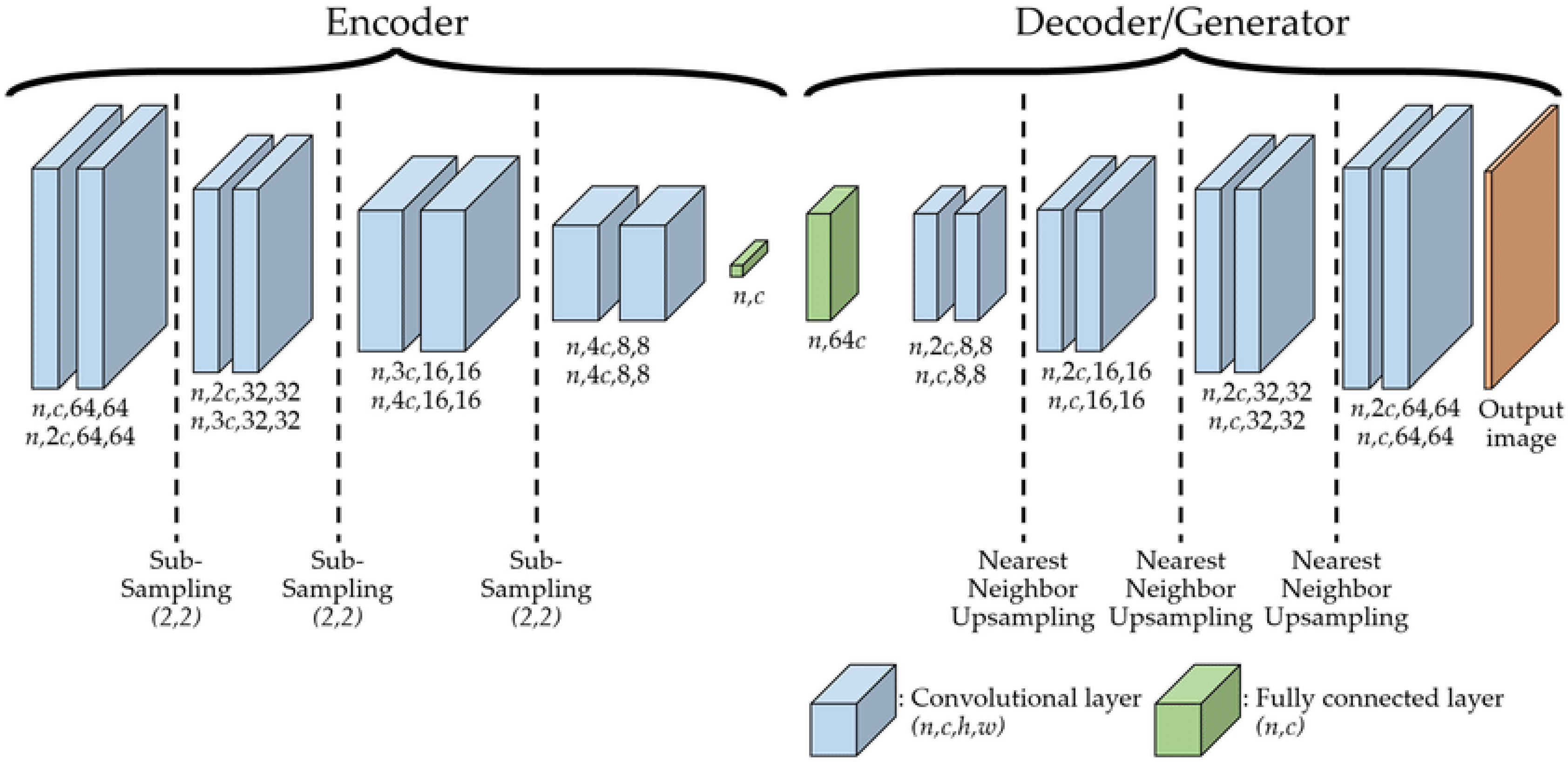
Optimal transport – aka W-distance

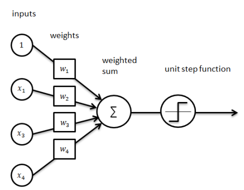


Source of image: https://vincentherrmann.github.io/blog/wasserstein/

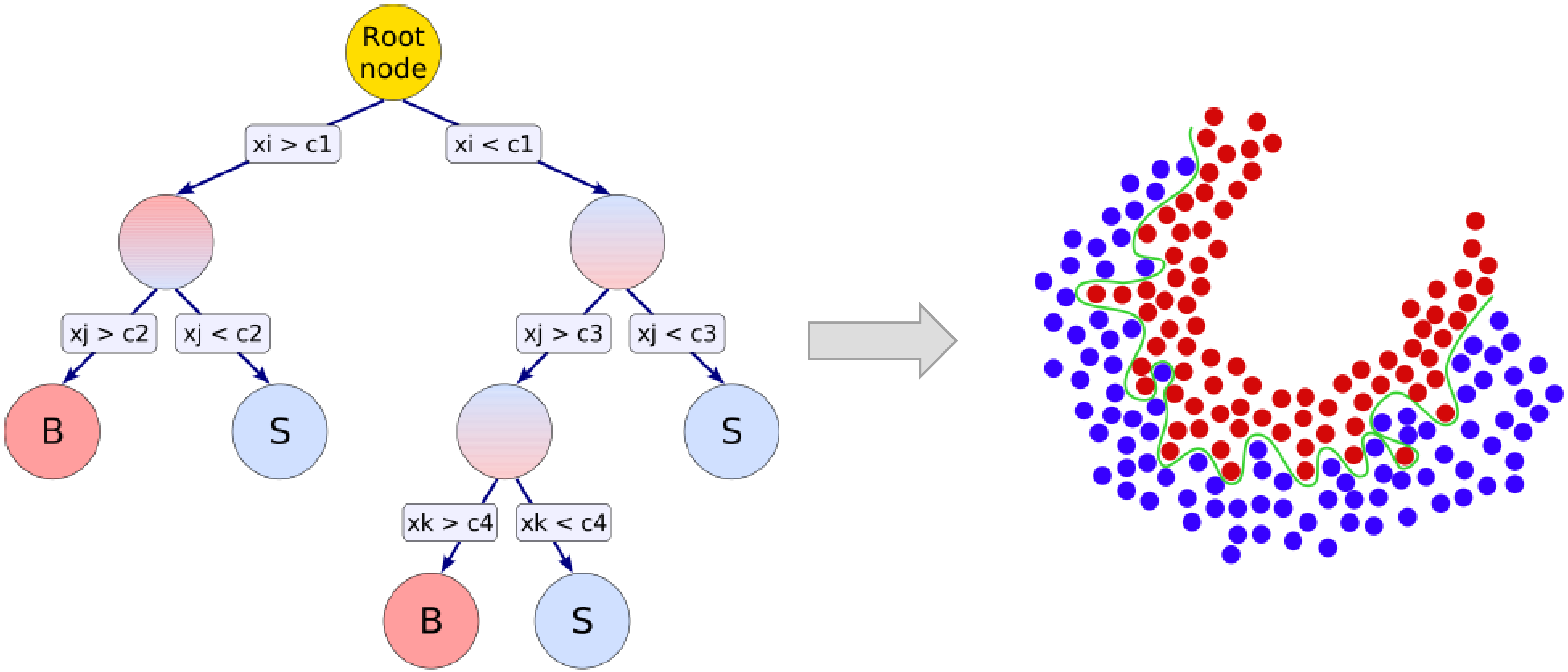


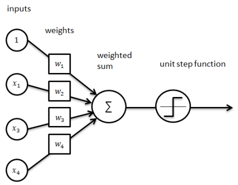
Autoencoders





Decision trees

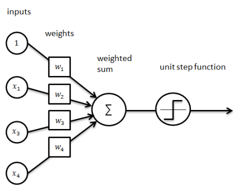




HEP landscape

- BDT models for binary classification of events – online trigger systems, **offline selections**
- ANN models – PID enhancements (crucial for flavour physics, precise measurements), P.D.F. reconstruction
- Generative models based on GANs and Autoencoders – event generators, data augmentation

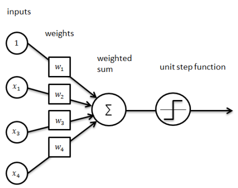
- A comprehensive repository regarding current status: https://iml-wg.github.io/HEPML-LivingReview/ (**A Living Review of Machine Learning for Particle Physics**)



HEP landscape

- Very interesting overview: **„Machine Learning in High Energy Physics Community White Paper”** (https://arxiv.org/abs/1807.02876)

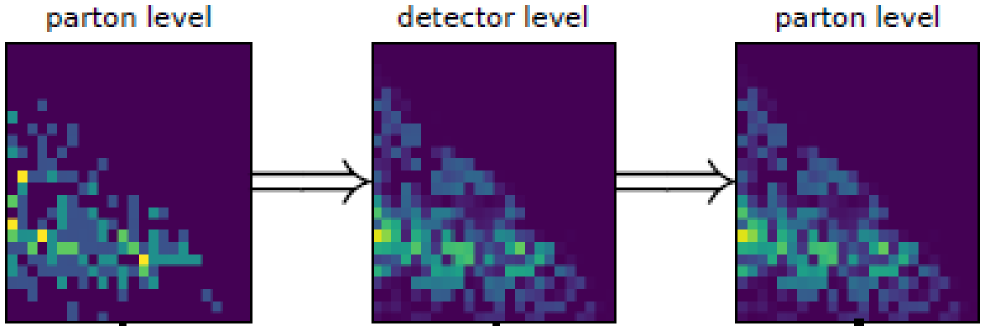
  - Challenges of learning Standard Model
  - Speeding simulation via generative models
  - Computing resources and sustainability
  - Engaging commercial partners (new LHCb trigger based on GPU processors)
  - Interpretability of models
  - Uncertainty of predictions (just beginning this large subject)

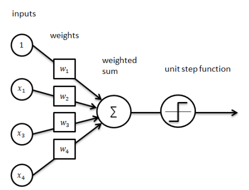


HEP landscape

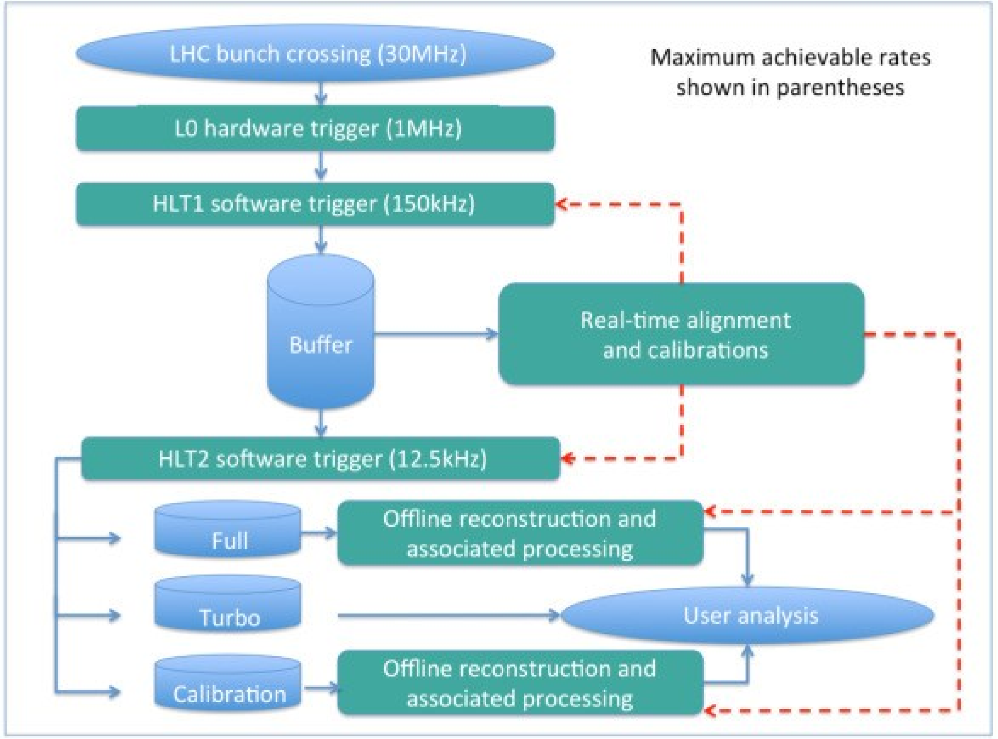
- „**Generative Networks for LHC events**” (https://arxiv.org/abs/2008.08558)

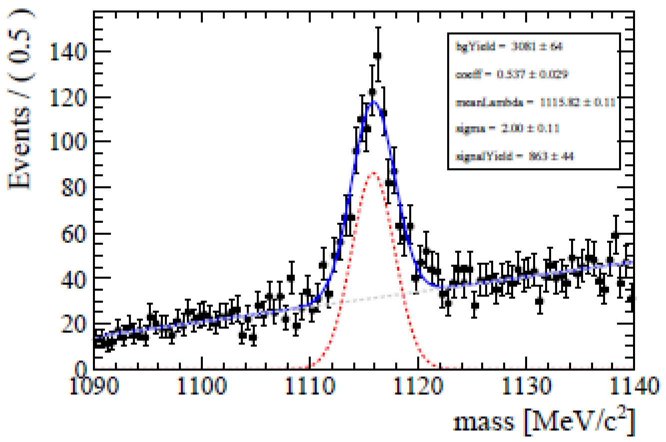
  - Physics specific challenges: phase-space integration, conservation of 4-momentum
  - Parton shower and matrix elements modelling
  - CycleGANs for understanding the patron showers





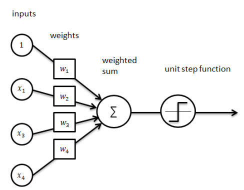
LHCb Trigger (Run 2)



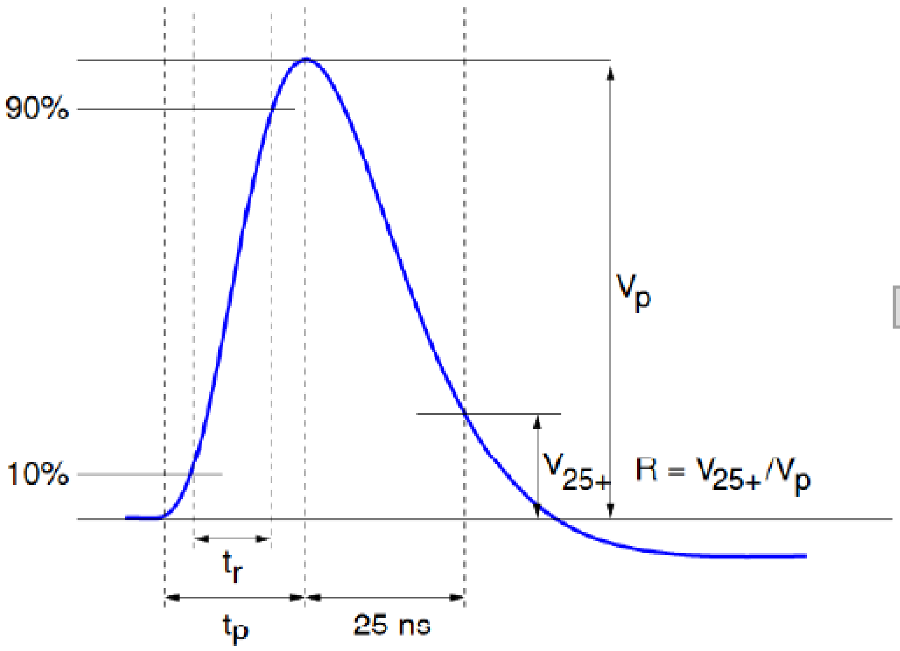


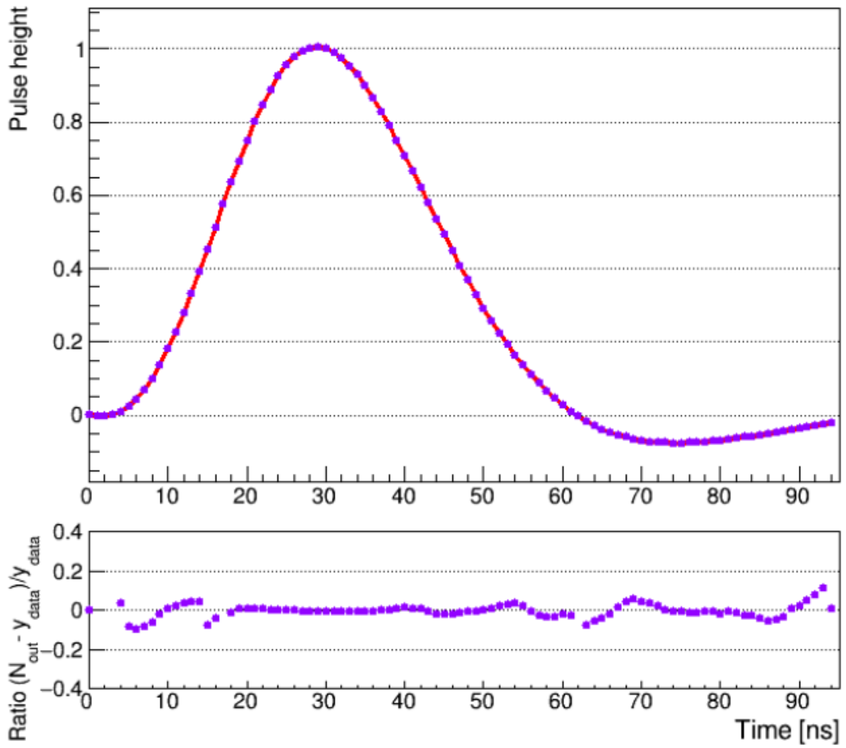
Long-lived tracking in HLT using XGBoost algoritym

Adam Dendek LHCb Thesis
http://cds.cern.ch/record/2772792?ln=en



Readout electronics response with ANN





Predicting the future for HEP

- HEP challenges are definitely closely coupled with the recent trends in ML
- Use more sustainable code (share/use the latest and greatest)
- Interpretability – critical especially for selection algorithms (SHAP and LIME)
- Prediction error – when looking for New Physics we should now it!
- Use latest hardware developments – GPU clusters, tensor cores, hardware ANN
- More models!

Thanks!

A simple one

Cross-entropy, better loss function

Count the "bad decisions" and penalise the model!

Mean Squared Error Loss

$$L_1 = \underbrace{\frac{1}{n}}_{\#\text{ events}} \sum (\underbrace{y_i}_{\text{true label}} - \underbrace{\tilde{y}_i}_{\text{predicted label}})^2$$

Binary Cross-entropy loss

$$L_2 = -\frac{1}{n} \sum_i \{ y_i \ln(\tilde{y}_i) + (1 - y_i) \ln(1 - \tilde{y}_i) \}$$

$$L_2(y_i, \tilde{y}_i) \neq L_2(\tilde{y}_i, y_i)$$

A simple one

Try to see how it works, again let's have small data sample $d: \{x_1\}$

$L_1 = (y - \tilde{y})^2 \rightarrow$ good for regression

$|L_2| = y \ln(\tilde{y}) + (1-y) \ln(1-\tilde{y}) \rightarrow$ good for classification

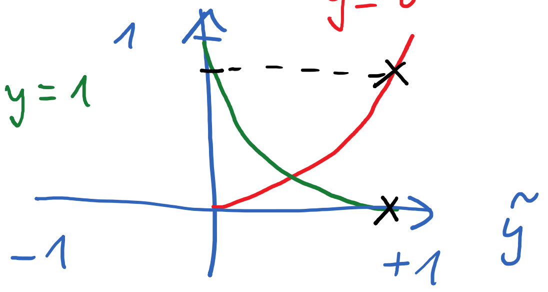
Gedanken experiment (two classes)

$L_1(y=0, \tilde{y}) = \tilde{y}^2, \quad L_1(y=1, \tilde{y}) = (1-\tilde{y})^2$

$L_2(y=0, \tilde{y}) = \ln(1-\tilde{y}), \quad L_2(y=1, \tilde{y}) = \ln(\tilde{y})$

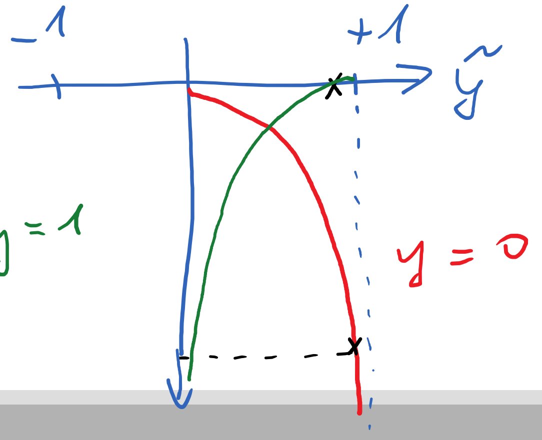
Visualisation please!

Visualise!



$y = 0$, $\tilde{y} = 0.5$ (bad decision)

$$\begin{cases} L_1 = 0.81 \\ \dfrac{\partial L_1}{\partial \tilde{y}} = 1{,}81 \rightarrow \text{model penalty} \end{cases}$$



$$\begin{cases} L_2 = 2.3 \\ \dfrac{\partial L_2}{\partial \tilde{y}} \approx 10.0 \rightarrow \text{huge penalty!} \end{cases}$$

Be a responsible punisher …

Penalty ≡ change of parameters

$$\Delta w_1 \rightarrow \frac{\partial L_1}{\partial w} = \underbrace{\frac{\partial L_1}{\partial \tilde{y}}} \times \frac{\partial \tilde{y}}{\partial w}$$

$$\Delta w_2 \rightarrow \frac{\partial L_2}{\partial w} = \underbrace{\frac{\partial L_2}{\partial \tilde{y}}} \times \frac{\partial \tilde{y}}{\partial w}$$

Regression → cont. variables: do not be very angry …

Classification → bad class → big deal !

Algorytm uczący się – AL-U

- Potrzeba stworzenia nowej klasy algorytmów, które się uczą wynika z tego, że próbujemy rozwiązać szereg problemów **zbyt skomplikowanych** dla programisty człowieka
- Uwaga! **Wykonywanie zadań** przez algorytm **nie jest związane z uczeniem się**!
- Uczenie to sposób **nabywania umiejętności** do wykonywania zadań
- Proces uczenia dotyczy więc, **sposobu przetwarzania** przez AL-U przypadków ze zbioru treningowego. Każdy przypadek będzie reprezentowany przez **wektor cech – zmienne losowe**, które zostały zmierzone podczas zbierania danych
- Każdy przypadek (próbka, egzemplarz) zapiszemy $\vec{\boldsymbol{x}} \in \mathbb{R}^n : \vec{\boldsymbol{x}} = \{x_1, x_2, \dots, x_n\}$