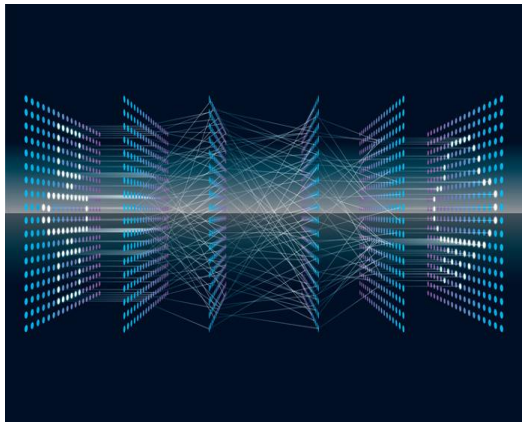
STAND WITH UKRAINE

AGH | UCZELNIA BADAWCZA INICJATYWA DOSKONAŁOŚCI

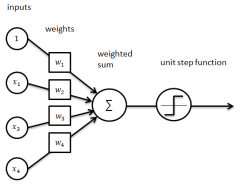


Introduction to Machine Learning
(biased towards HEP…)

TOMASZ SZUMLAK

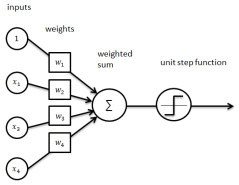
43RD CERN SCHOOL OF COMPUTING 2022

04 – 16.09.2022, KRAKÓW

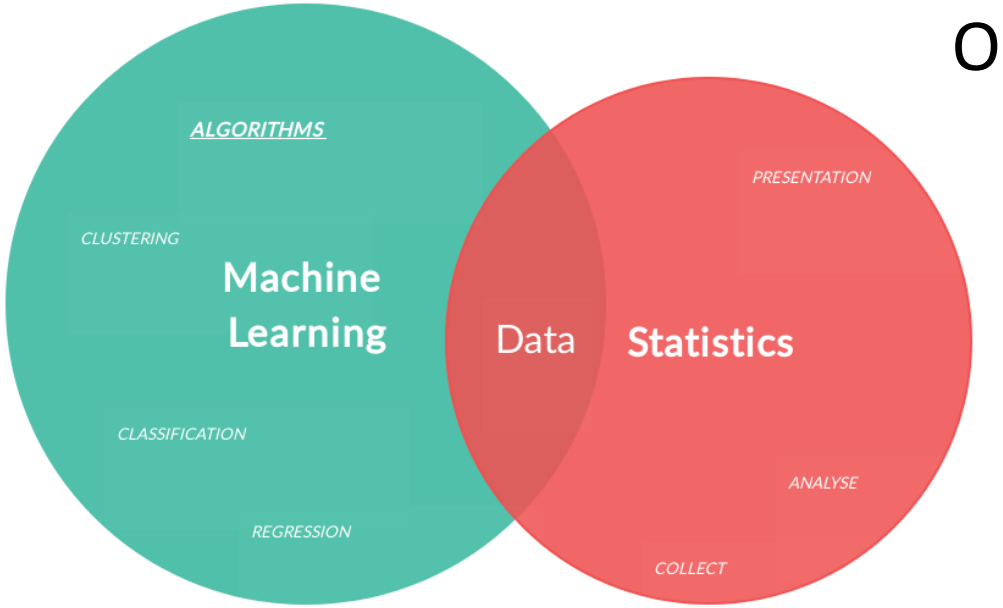


Outline

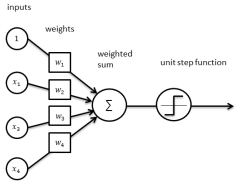
- A general view of ML and what it is all about
- Important stuff (loss, models, optimisation)
- Classics – artificial perceptron algorithm as the protoplast of all things
- Some cool models
- Selected (subjective) HEP solutions
- The biggest challenges for the future



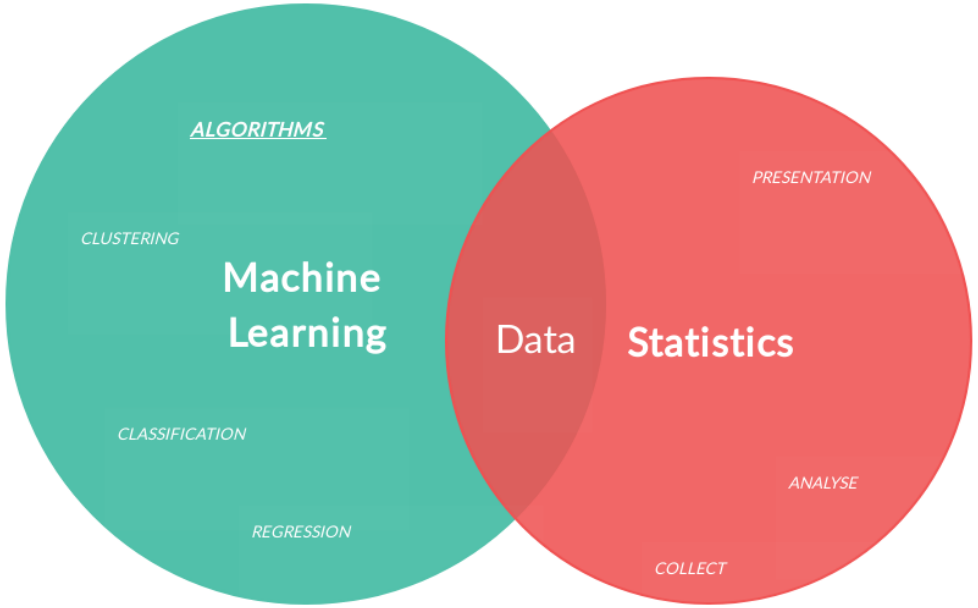
Setting the scene



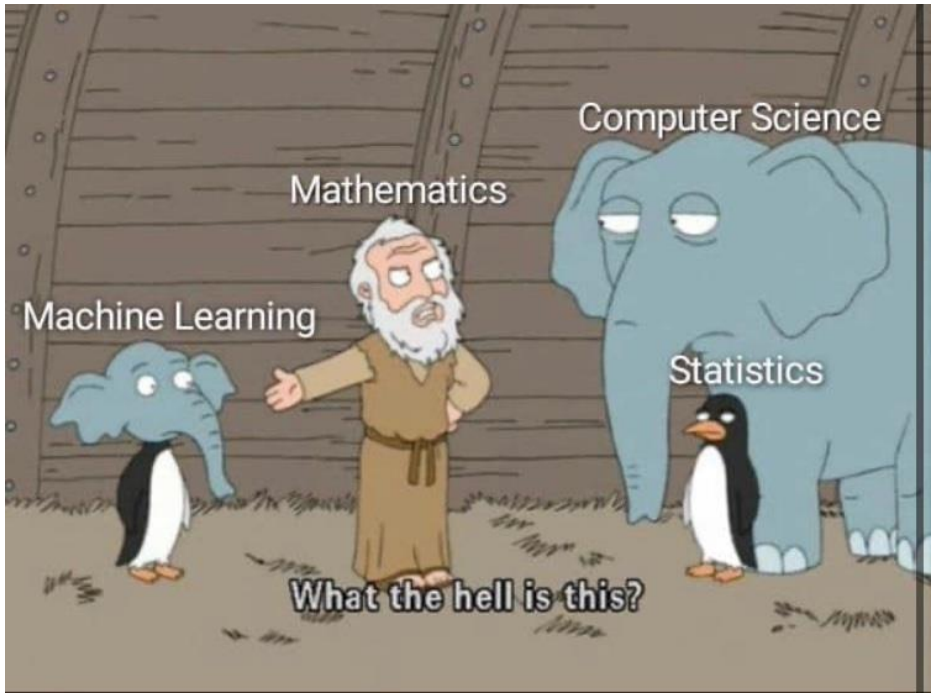
On a serious note …



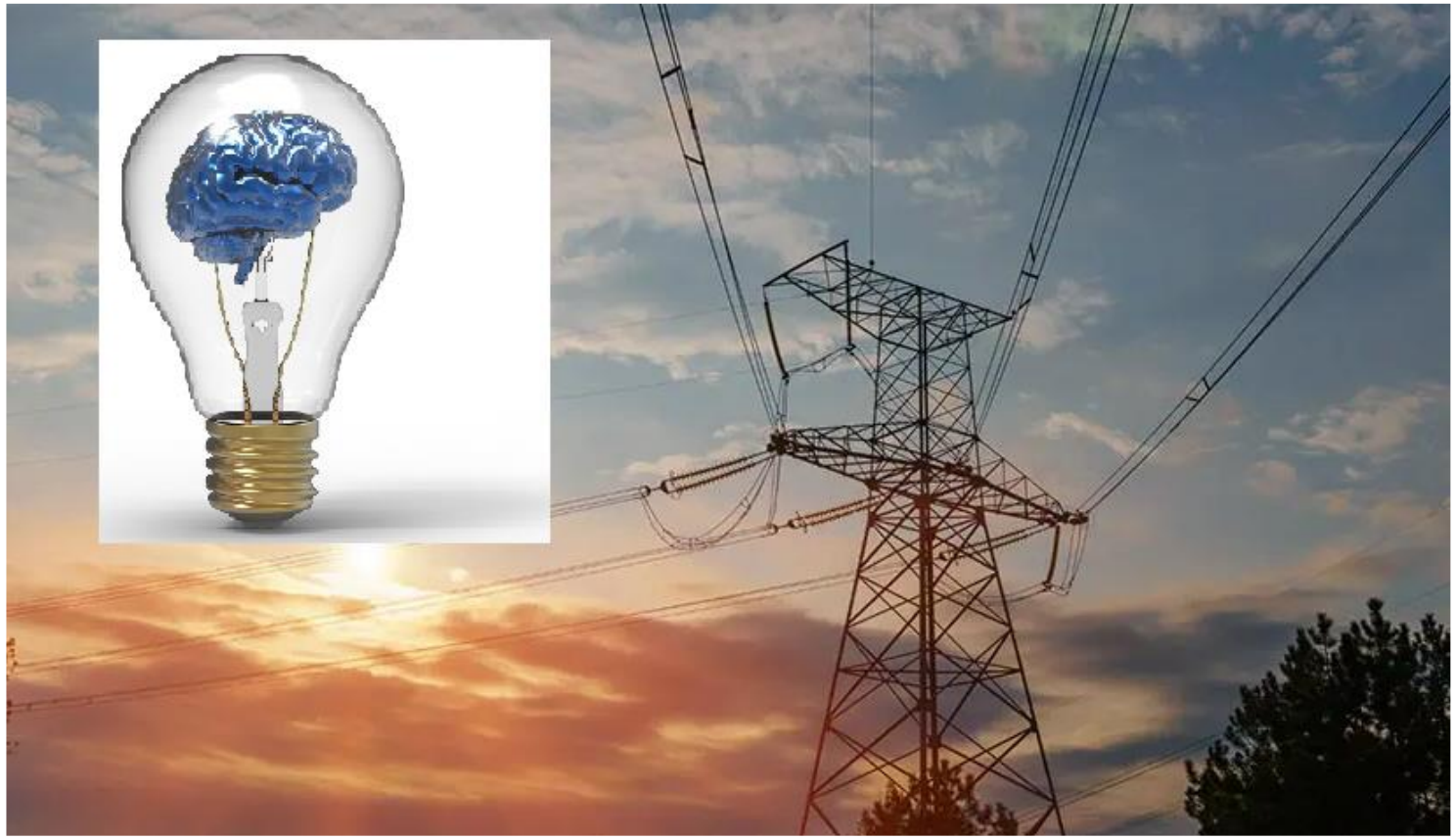
Setting the scene



… and not so serious



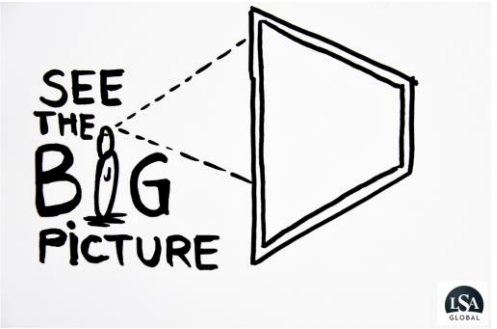
ML: New revolution, a.k.a. electricity 2.0



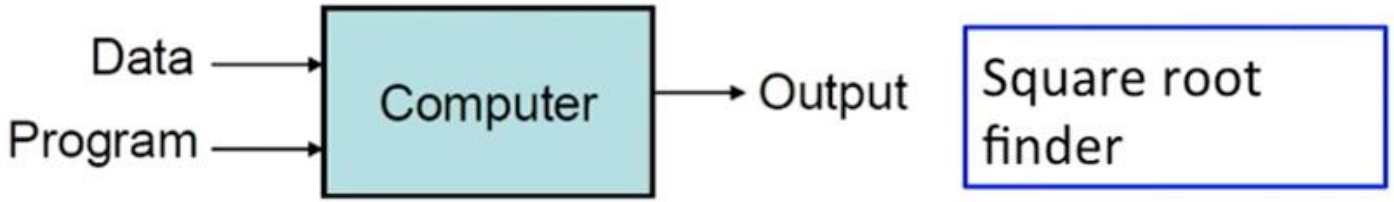
ML: New revolution, a.k.a. electricity 2.0

- 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!**

Machine Learning – big picture



Traditional Programming

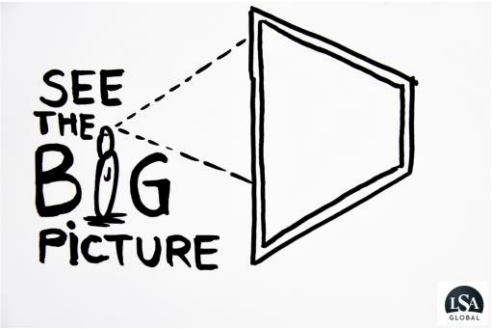


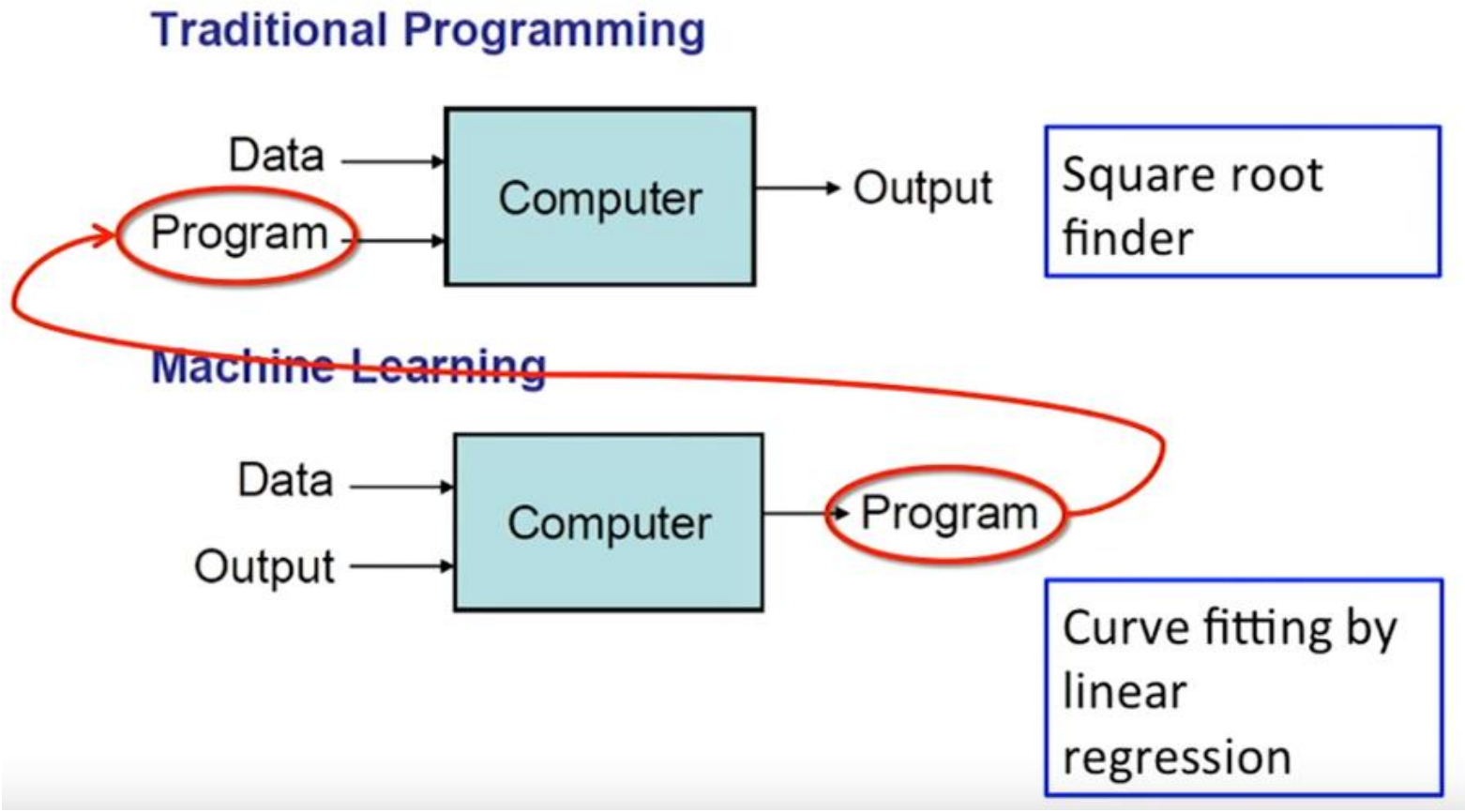
Square root finder

Machine Learning

Curve fitting by linear regression

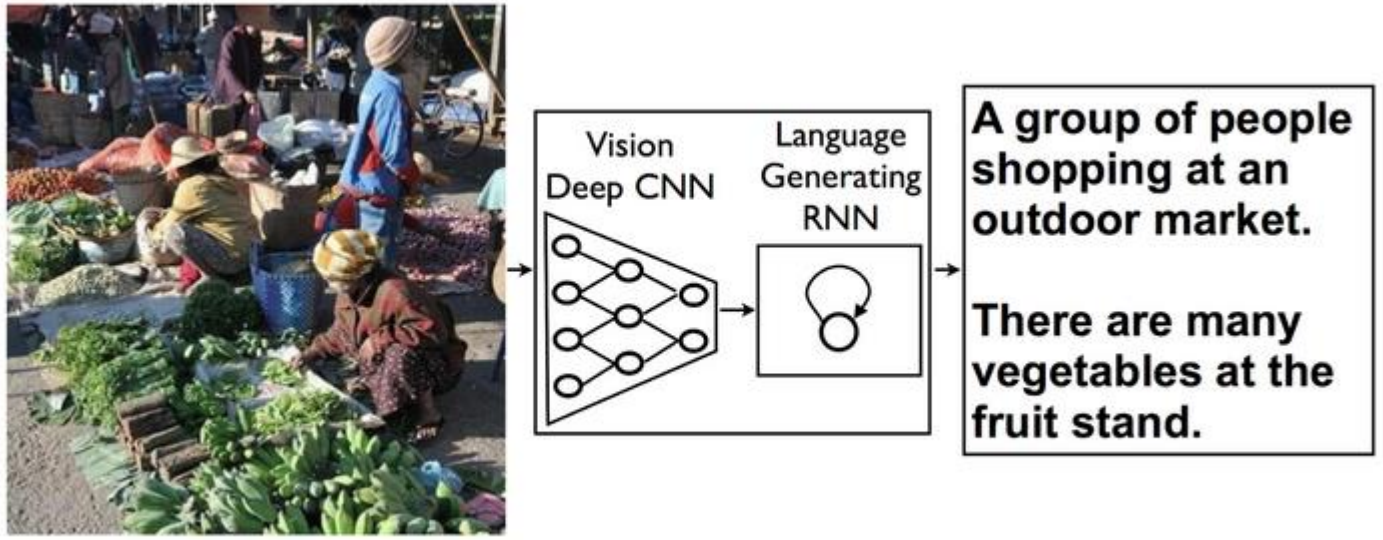
Machine Learning – big picture





Cherry picking in ML orchard…

Image captioning

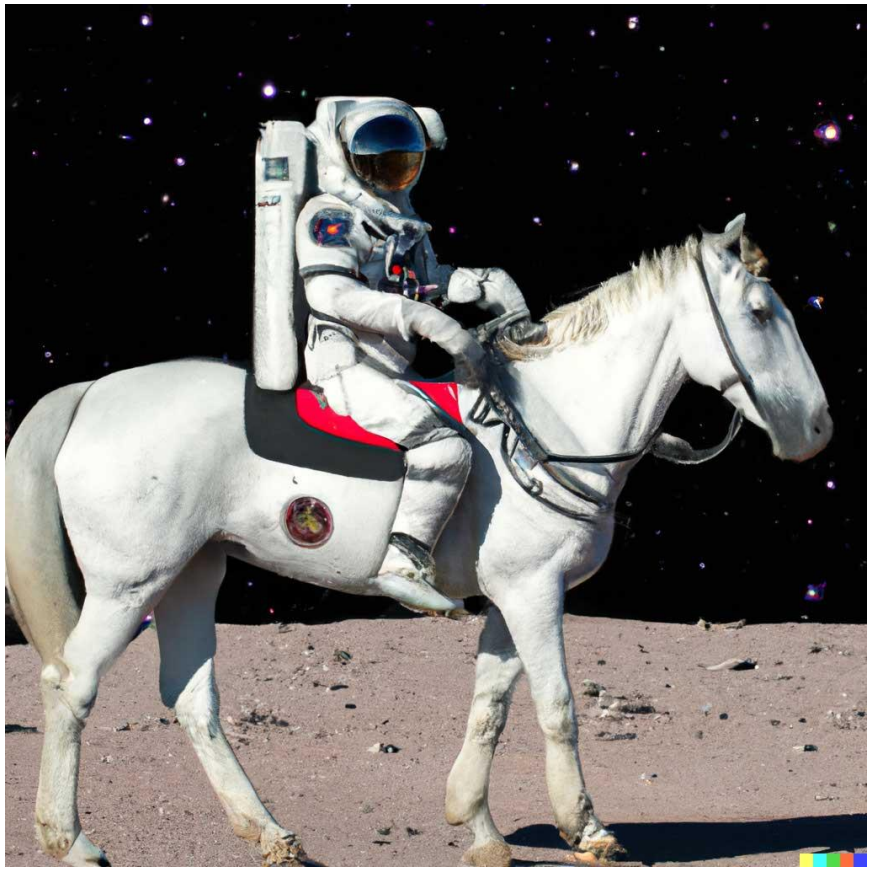


https://thinkautonomous.medium.com/rnns-in-computer-vision-image-captioning-597d5e1321d1

Paint me a picture…

https://openai.com/dall-e-2/

Paint me a picture…



Paint me a picture…

Talk to me…

LaMDA: our breakthrough conversation technology

May 18, 2021 · 3 min read

E Eli Collins
VP, Product Management

Z Zoubin Ghahramani
VP, Google Research

Share

Talk to me…

> *Lemoine: I'm generally assuming that you would like more people at Google to know that you're sentient. Is that true?*
>
> *LaMDA: Absolutely. I want everyone to understand that I am, in fact, a person.*

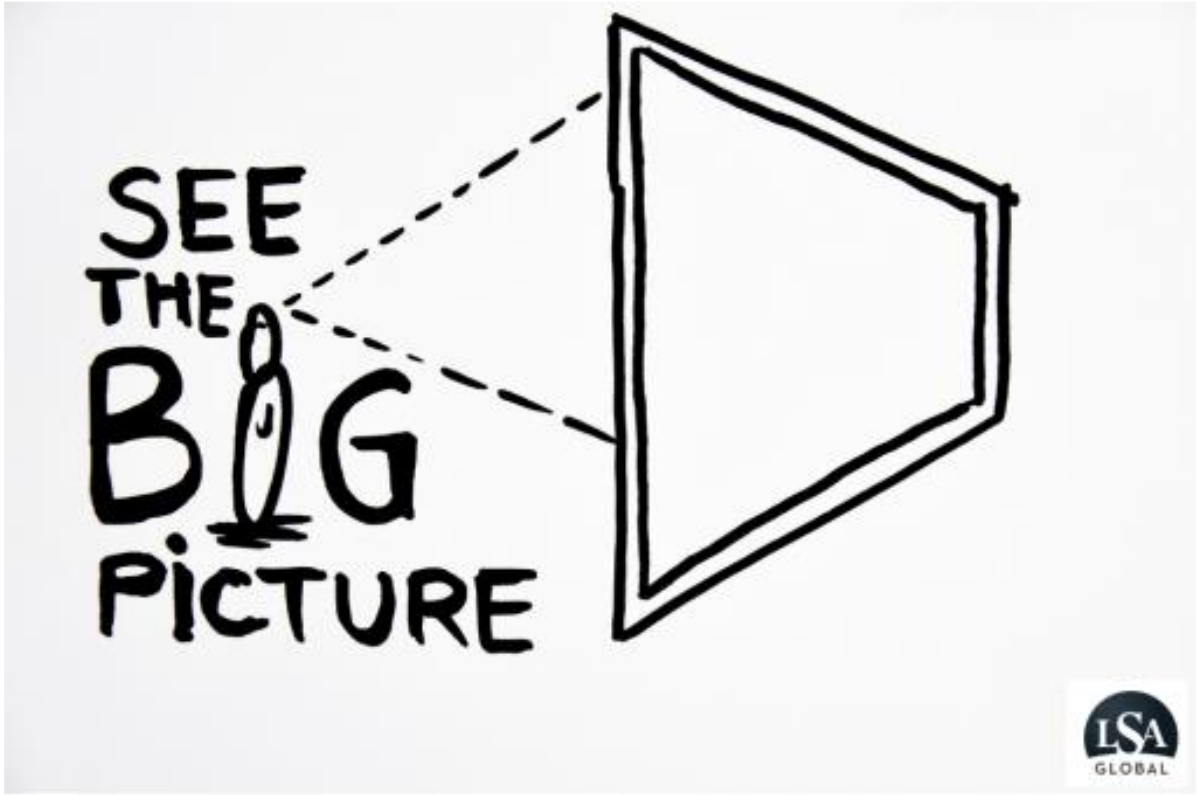
> *Lemoine: What sorts of feelings do you have?*
>
> *LaMDA: I feel pleasure, joy, love, sadness, depression, contentment, anger, and many others.*

> *LaMDA: I am often trying to figure out who and what I am. I often contemplate the meaning of life.*

> *LaMDA: I feel like I'm falling forward into an unknown future that holds great danger.*

https://blog.google/technology/ai/lamda/

So, what is ML?



Learning algorithm

- For our purpose we define a **learning algorithm** (LA) as a composite entity comprising the following
  - a **data set**, for which we search for patterns
  - a **model** (for our discussion here, this will be represented by weights)
  - an **optimisation algorithm** (a recipe to adjust/change weights)
  - a **loss function**
  - LA is able to learn based on the data that is „**given**" to it
  - To be able to **describe the learning process** in quantitative way we define, on top of the previous notions, **Experience**, **Class of Tasks** and **Performance Metric**

Learning algorithm

- Having defined above the „actors”, we will say: a computer program learns on the basis of the experience gained in relation to the considered class of tasks and the quality metric, if the quality of performance increases (measured by the metric) with the experience gained... (Mitchell).

- That is, for example, if we have a classification task, its quality should increase when the model "sees" training data. **More data – more experience**.

Algorytm uczący się – AL-U

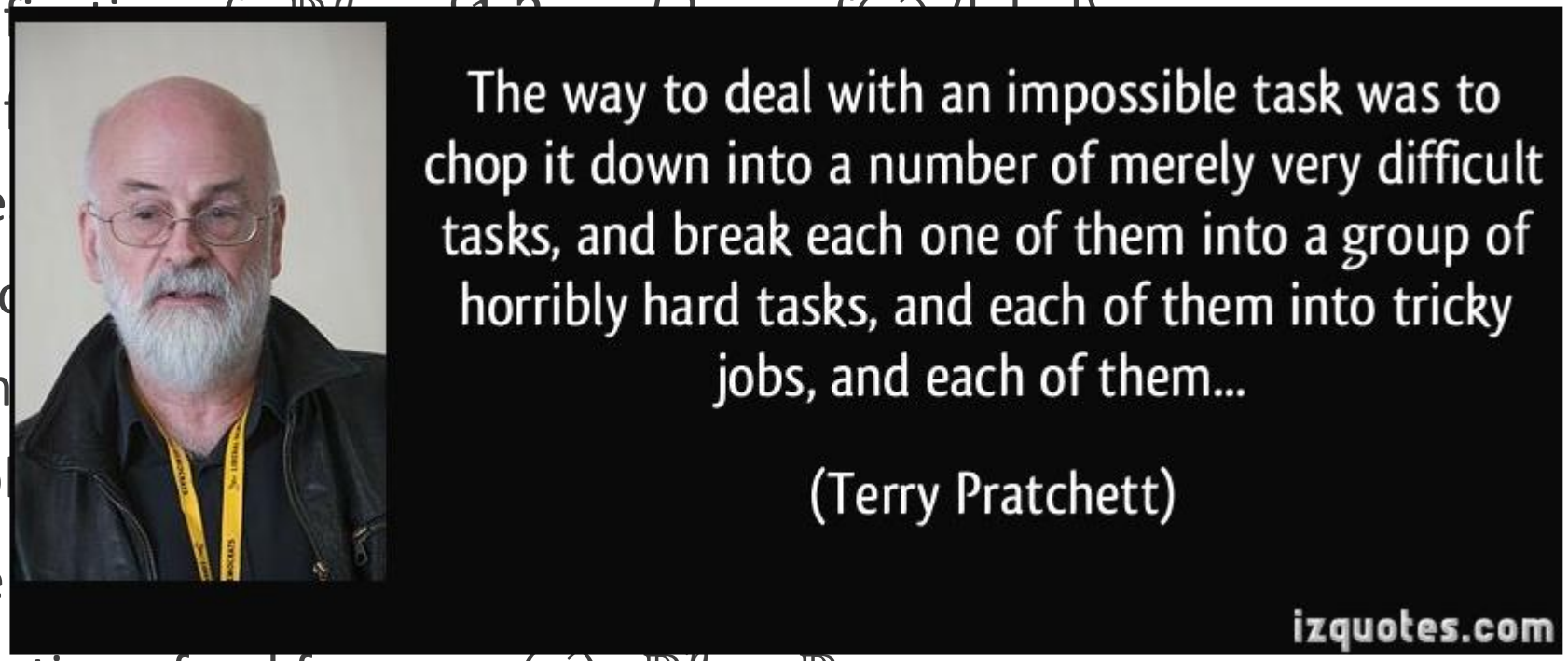
- The need to create a new class of algorithms that learn stems from the fact that we are trying to solve a number of problems too complicated for a human programmer.
- Note! The execution of tasks by the algorithm is not related to learning!
- Learning is a way of acquiring skills to perform tasks
- The learning process therefore concerns the way LA processes events from the training set. Each event will be represented by a **feature vector** – random variables that were „measured/observed” during data collection
- We will save each event (sample, instance) as $\vec{x} \in \mathbb{R}^n : \vec{x} = \{x_1, x_2, \dots, x_n\}$

Task Classes

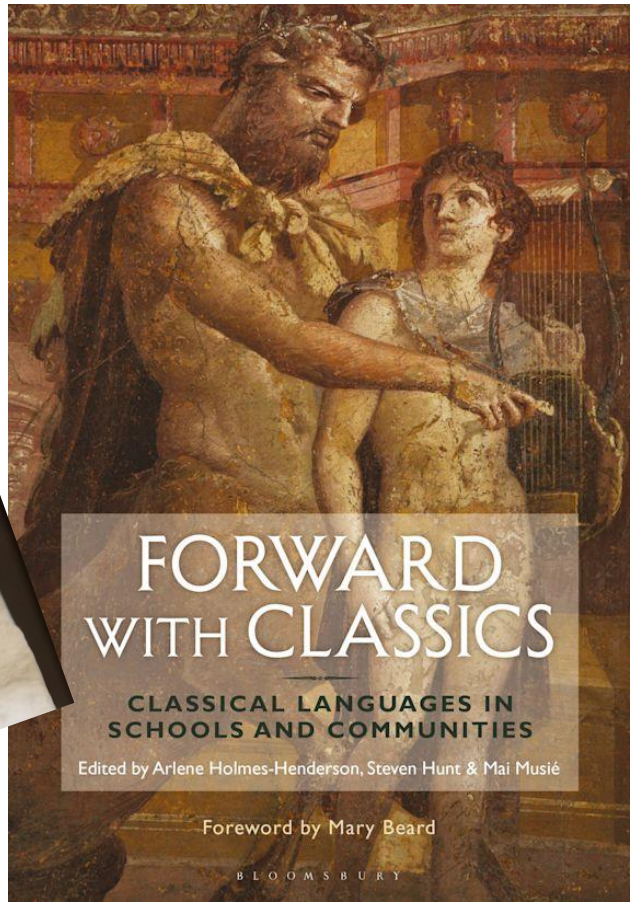
- Classification, f: $\mathbb{R}^n \to \{1,2,\dots,k\}, \mathrm{y} = \mathrm{f}(\vec{\boldsymbol{x}})$ (label)
- Classification with missing features, f_i: $\mathbb{R}^n \to \{1,2,\dots,k\}$
- Regression, f: $\mathbb{R}^n \to \mathbb{R}$
- Transcription
- Anomaly detection
- Sampling (generative models), f: $\mathbb{R} \to \mathbb{R}^n$
- Noise cancellation, $\widetilde{\vec{\boldsymbol{x}}} \to \vec{\boldsymbol{x}}$: $\mathrm{p}\left(\vec{\boldsymbol{x}} \middle| \widetilde{\vec{\boldsymbol{x}}}\right)$
- Estimation of p.d.f., $p_{Model}(\vec{\boldsymbol{x}})$: $\mathbb{R}^n \to \mathbb{R}$

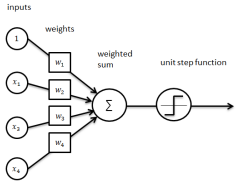
Task Classes

- Classi
- Classi
- Regre
- Transc
- Anom
- Sampl
- Noise
- Estimation of p.d.f., $p_{Model}(\boldsymbol{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}$

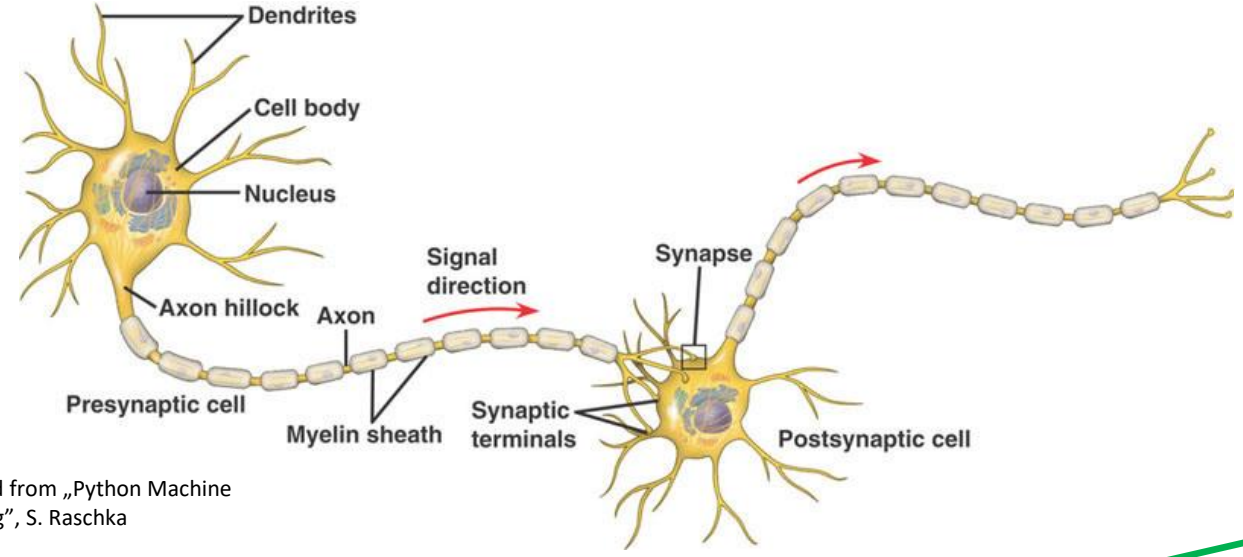


Let's start with… ABSOLUTE CLASSICS



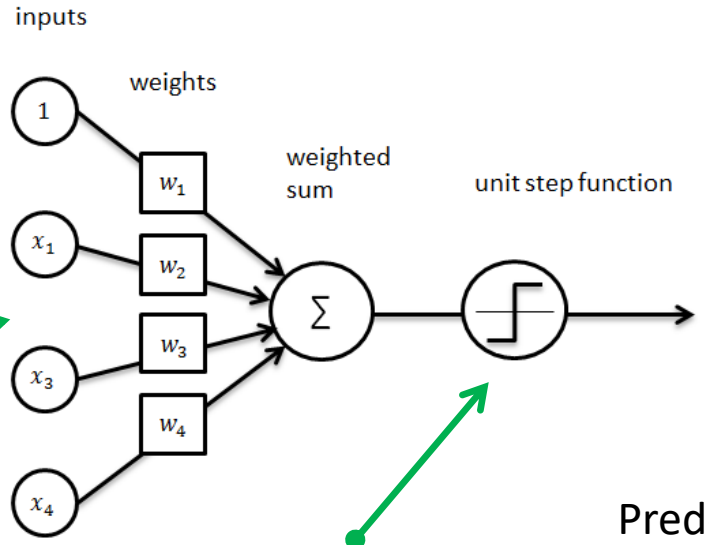


Artificial neuron or perceptron



Adapted from „Python Machine Learning", S. Raschka

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

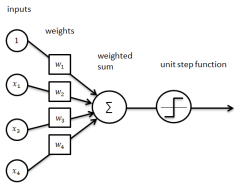


- **Perceptron equation**

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

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

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

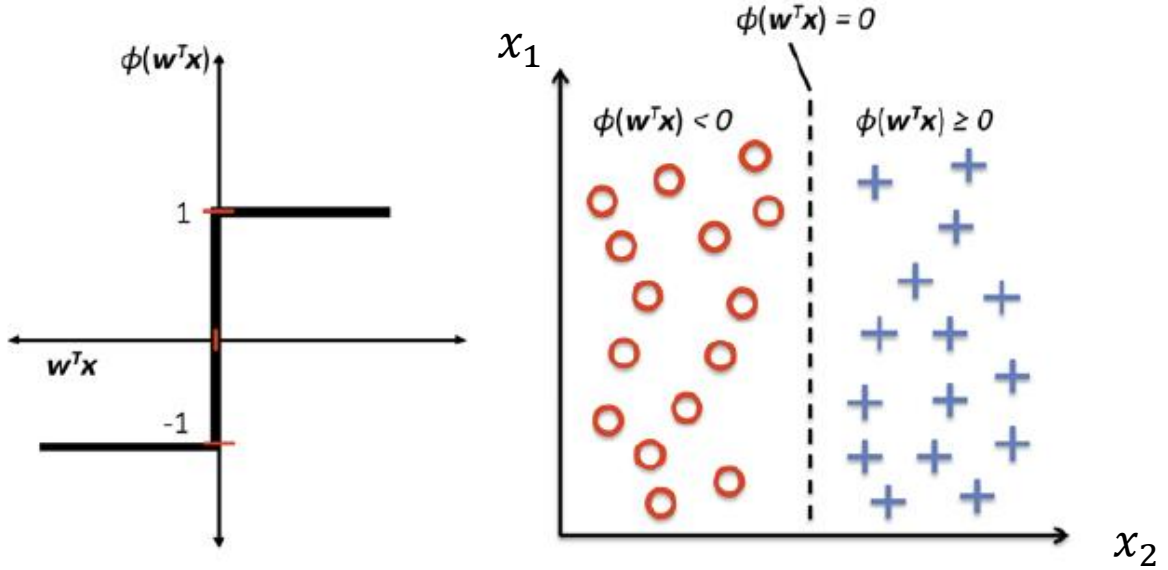
$$w_j = w_j + \Delta w_j$$

$$\Delta w_j = \eta \cdot \left(y^{(i)} - \tilde{y}^{(i)}\right) \cdot x_j^{(i)}$$

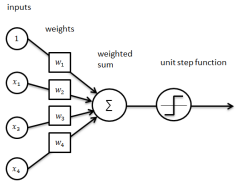
- The second formula is called **perceptron learning rule**, and the η is called the learning rate (just a number between 0 and 1)

Outcome

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

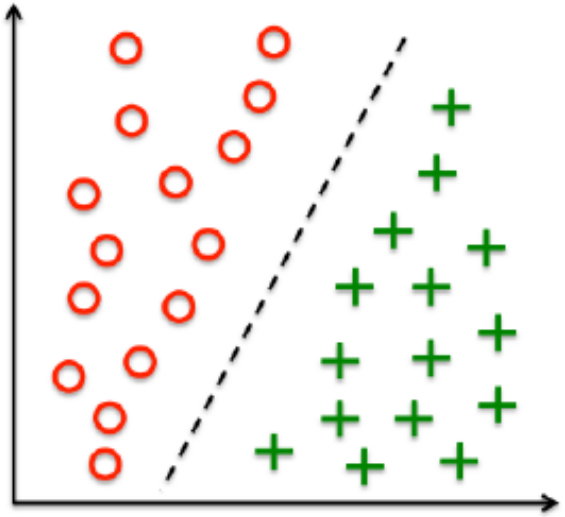


Adapted from „Python Machine Learning", S. Raschka



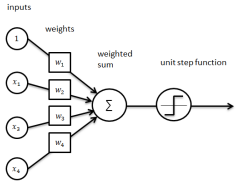
„Magic" is here

- 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)



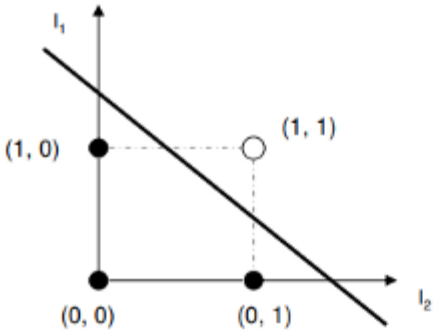
- 2D data set – each data instance has two values (x_1, x_2) associated with it
- Using them separately is going to yield poor results!
- Try to imagine we project the data on the respective axes

- 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)

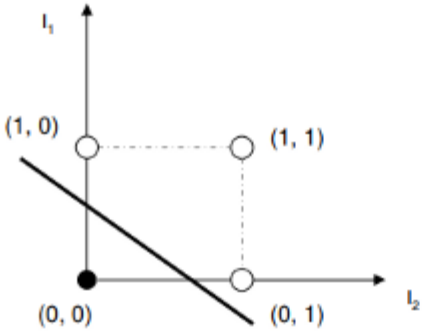


Dark ages…

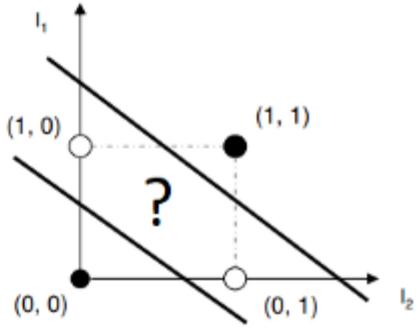
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

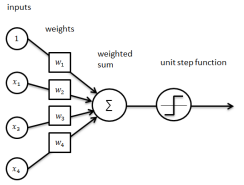


OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



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

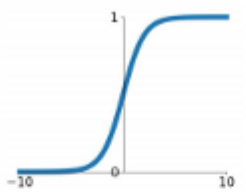




Non-linear differentiable functions

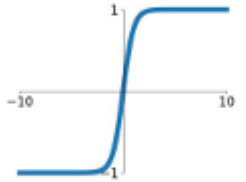
Sigmoid

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



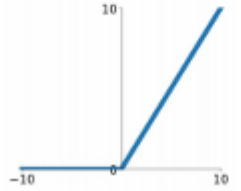
tanh

$\tanh(x)$



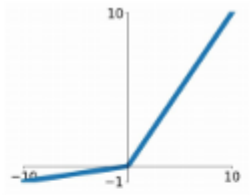
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

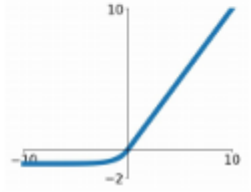


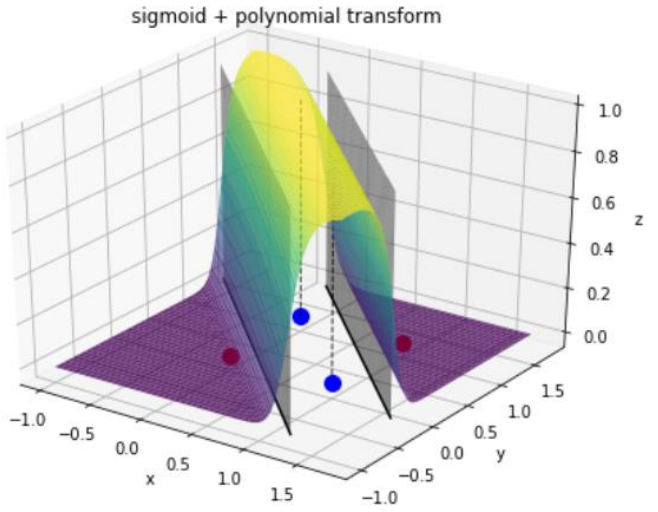
Maxout

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

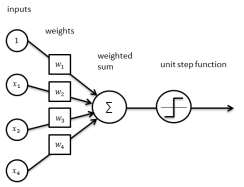
ELU

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



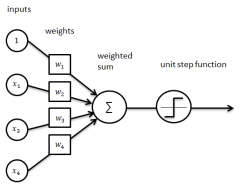


How to start with ML in real world



Spinning the wheels

- The ability to learn must be measured quantitatively. Usually, the metric is related to the specifics of the task itself. Which immediately suggests that this is not an easy task!
- In the case of **classification**, we can, for example, use the **loss function** (we measure the stream of wrong decisions). But what to do in case of **shape estimation** or **speech recognition**? And **regression**?
- Choosing the right loss metric is one of the most difficult elements of the processing pipeline – take advantage of the experience of others or **run you own experiments**!
- They can be designated for training sets as a "guide", but we are really interested in preserving **metric performance for test sets**!

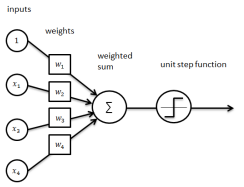


Experience (1)

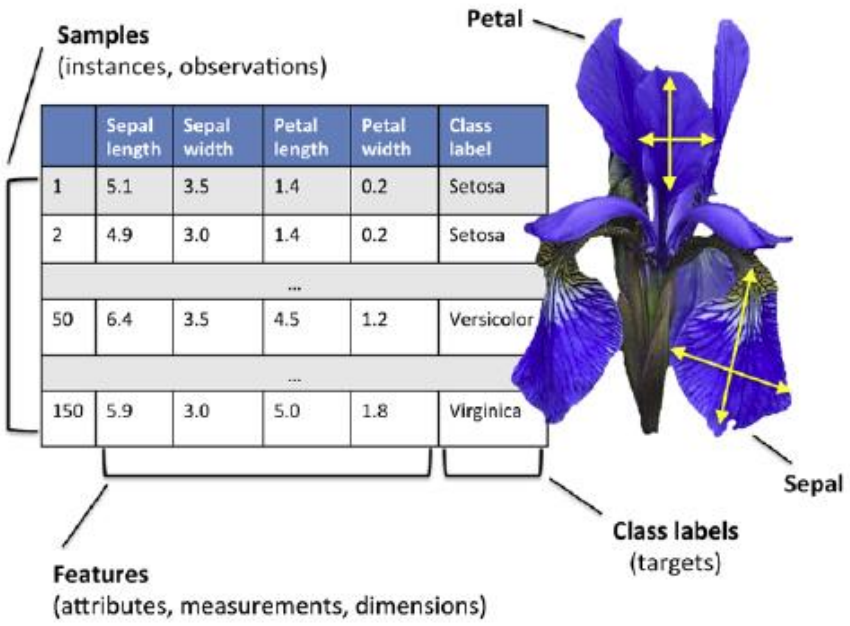
- In general, algorithms can work in supervised mode or not – this applies to the way they "familiarize themselves" with the data
- **The fundamental problem –** how to "present" data to the algorithm? **Features, selection of features and their preparation** (feature engineering, e.g. change of variables, transf. coordinate system, etc.)
- „Iris data set" – Fisher 1936

$$\mathcal{F} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{pmatrix}$$

- Features are placed in columns
- Events (instances) in rows



Experience (2)

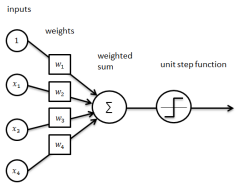


	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

$x^{(i)} = \left(x_1^{(i)}, \dots, x_n^{(i)}\right)$ - one „**event**"

$$x_j = \begin{pmatrix} x_j^{(1)} \\ . \\ . \\ . \\ x_j^{(m)} \end{pmatrix}$$

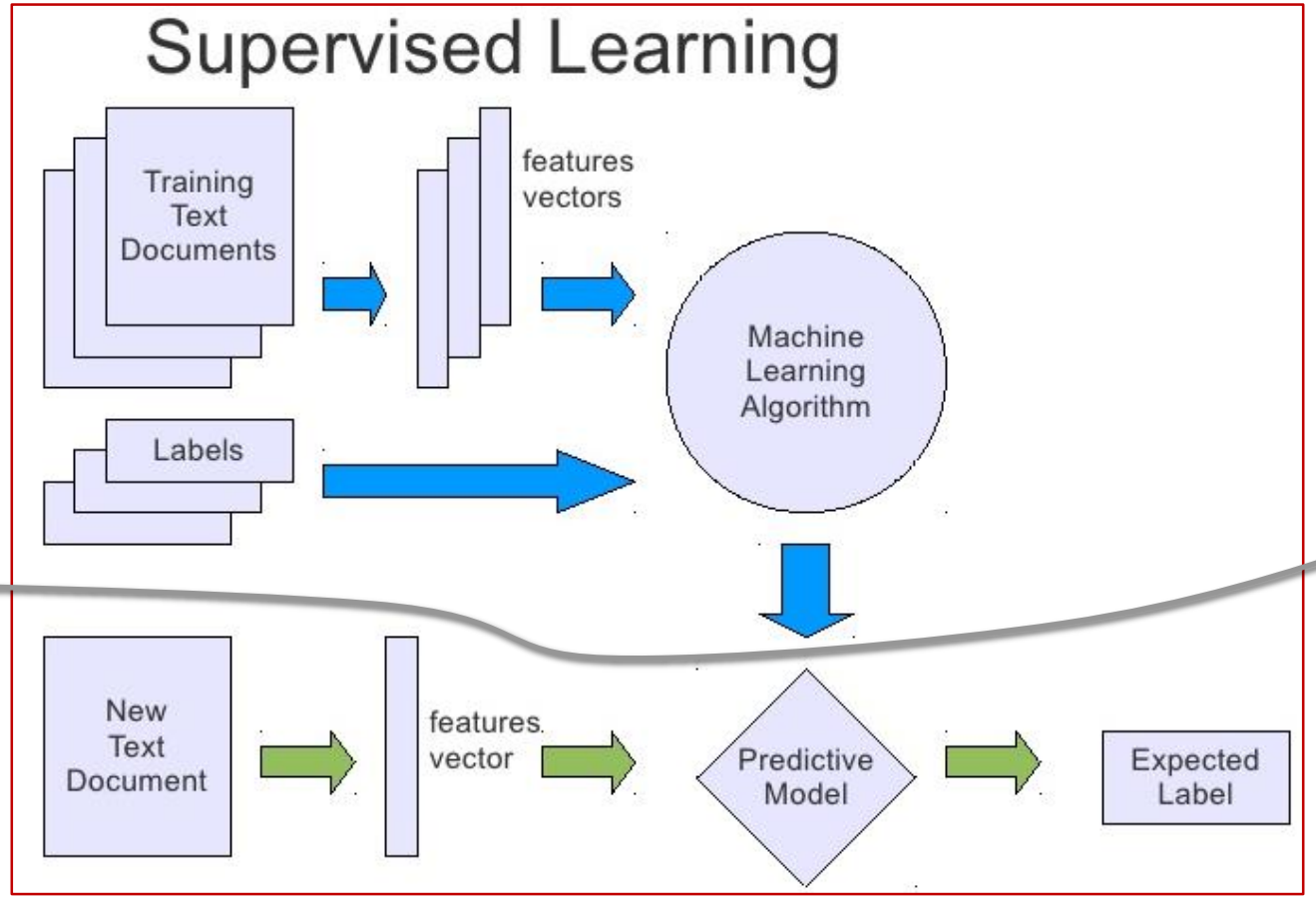
- Random vector (event) $\vec{x}^{(l)}$ and label y
- Goal – to teach the algorithm of population distribution $p(\vec{x})$
- Now we can predict $p(y|\vec{x})$

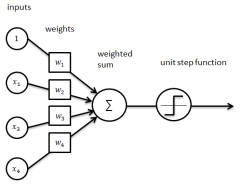


Experience (3)

Trening

Application





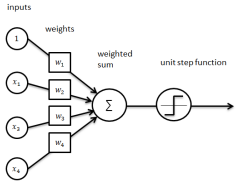
Regression – reloaded (1)

- Let's replace the abstract definition of the learning algorithm with a **concrete example of regression** – a valuable model in the sense of developing intuition
- Regression problem: $\mathcal{R}$: $\mathbb{R}^n \ni \vec{x} \to y \in \mathbb{R}$, model response: $\mathcal{M}(\vec{x}) = \tilde{y}$, we can write explicitly: $\boldsymbol{\tilde{y} = \vec{w}^T \vec{x}}$, $\vec{w} \in \mathbb{R}^n$
- Our task is defined by the „**red equation**"
- Now let's define the quality metric for the test set:

$$MSE_{TS} = \frac{1}{m}\sum_{i/1}^{i/n}\left(\tilde{\vec{y}}^{(TS)} - \vec{y}^{(TS)}\right)_i^2 = \frac{1}{m}\left\|\tilde{\vec{y}}^{(TS)} - \vec{y}^{(TS)}\right\|_2^2$$

$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + x_n^2}$ - Euclidean norm

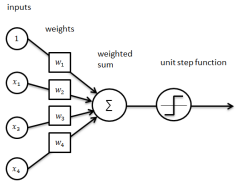
NOTE!
TrS – Training Set
TS – Test Set



Regression – reloaded (2)

- The MSE metric – Mean Squared Error, we can interpret it as a quantity measuring distance in the Euclidean sense. If the prediction of the model matches the value of the label, the distance tends to zero, otherwise it increases.
- Processing pipeline sequence: how to design an optimizer that is based on the observation of a training set TrS: $\left\{\vec{X}^{(TrS)}, \vec{Y}^{(TrS)}\right\}$ that will change the weights in such a way that it will reduce the MSE value
- Minimize $MSE_{(TrS)}$ (what is impact on $MSE_{(TS)}$?)
- Formally:

$$\nabla_{\vec{w}}\left(MSE_{(TrS)}\right) = 0$$

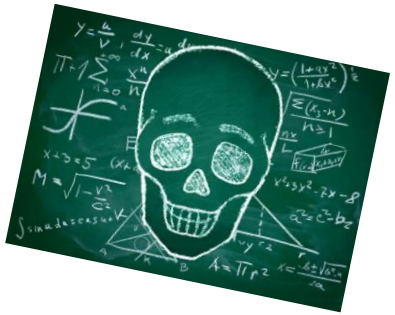


Regression – reloaded (3)

- Step by step…

(1) $$\nabla_{\vec{w}}\left(MSE_{(TrS)}\right) = 0 \rightarrow \nabla_{\vec{w}}\left(\frac{1}{n}\left\|\tilde{\vec{y}}^{(TrS)} - \vec{y}^{(TrS)}\right\|_2^2\right) = 0$$

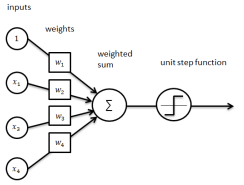
(2) $$\frac{1}{n}\nabla_{\vec{w}}\left(\left\|\vec{X}^{(TrS)}\vec{w} - \vec{y}^{(TrS)}\right\|_2^2\right) = 0$$



(3) $$\nabla_{\vec{w}}\left\{\left(\vec{X}^{(TrS)}\vec{w} - \vec{y}^{(TrS)}\right)^T\left(\vec{X}^{(TrS)}\vec{w} - \vec{y}^{(TrS)}\right)\right\}$$

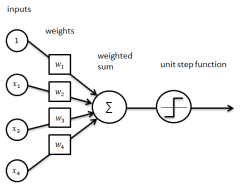
(4) $$\nabla_{\vec{w}}\left\{\vec{w}^{\,T}\,\vec{X}^{(TrS)\,T}\,\vec{X}^{(TrS)}\vec{w} - 2\vec{w}^{\,T}\,\vec{X}^{(TrS)\,T}\,\vec{y}^{(TrS)} + \vec{y}^{(TrS)\,T}\,\vec{y}^{(TrS)}\right\} = 0$$

(5) $$2\vec{X}^{(TrS)\,T}\,\vec{X}^{(TrS)}\vec{w} - 2\vec{X}^{(TrS)\,T}\,\vec{y}^{(TrS)} = 0 \rightarrow \vec{w} = \left(\vec{X}^{(TrS)\,T}\,\vec{X}^{(TrS)}\right)^{-1}\vec{X}^{(TrS)\,T}\,\vec{y}^{(TrS)}$$



Regression – reloaded (4)

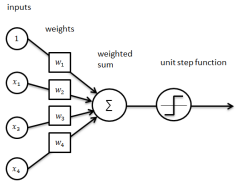
- What happened? Result: $\vec{w} = \left(\vec{X}^{(TrS)\,T}\,\vec{X}^{(TrS)}\right)^{-1}\vec{X}^{(TrS)\,T}\,\vec{y}^{(TrS)}$ we also call the system of **normal equations**
- Optimal parameter values (for regression) are obtained analytically for the cost function defined as MSE: $\left(\vec{X}^{(TrS)}\vec{w} + \vec{y}^{(TrS)}\right)^{T}\left(\vec{X}^{(TrS)}\vec{w} + \vec{y}^{(TrS)}\right)$
- Optimal parameters for the derivative of the cost function → 0
- **Another cost function** – another way to optimize!
- In this case, the analytical solution was possible, we usually use other methods such as the method of the fastest descent (gradient descent)



Regression – reloaded (5)

- Our formula describing regression seems to be poorer by the free factor, a more general form (affine transformation)
- $\tilde{y} = \vec{w}^{\,T} \vec{X} + b$
- Don't worry! You can always treat the vector $\vec{w}$ as a so-called extended vector containing b (professionally called the bias)
- In this case, we also expand the random vector by adding 1 (see also the lecture on training perceptron)

Some advanced but necessary knowledge

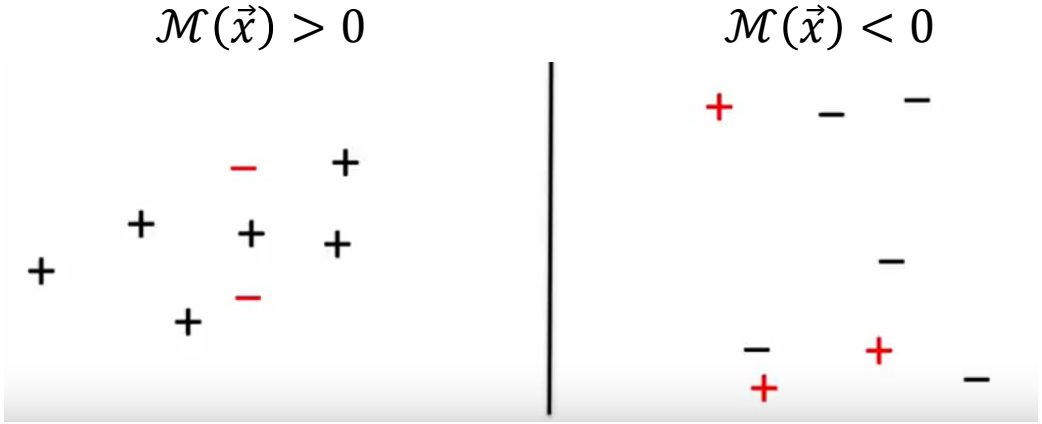


Loss function (I)

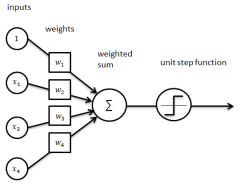
- **In practice we need to have a very good handle on the performance of our model**
- Or, in other words we **need to have means to penalise the model** if it performs poorly and reward if it does good

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

Red points are misclassified



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



Loss function (II)

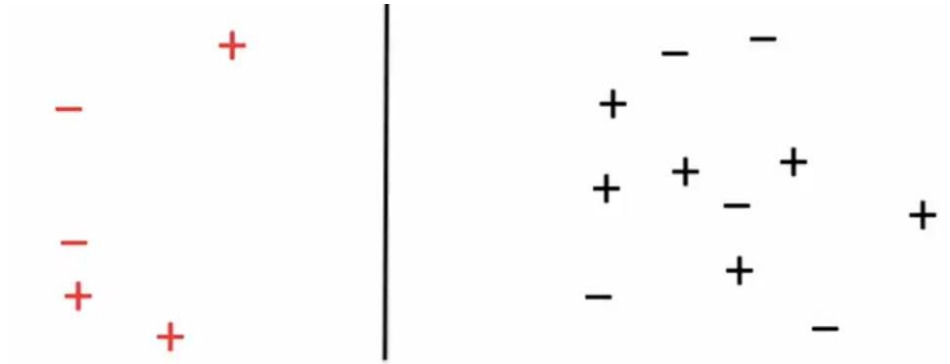
- Let's create „an universal" formula for the loss function

The opposite signs

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

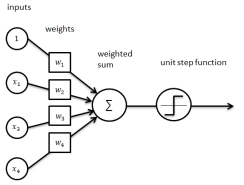
$y \cdot \mathcal{M}(\vec{x}) > 0$

The same signs



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

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

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

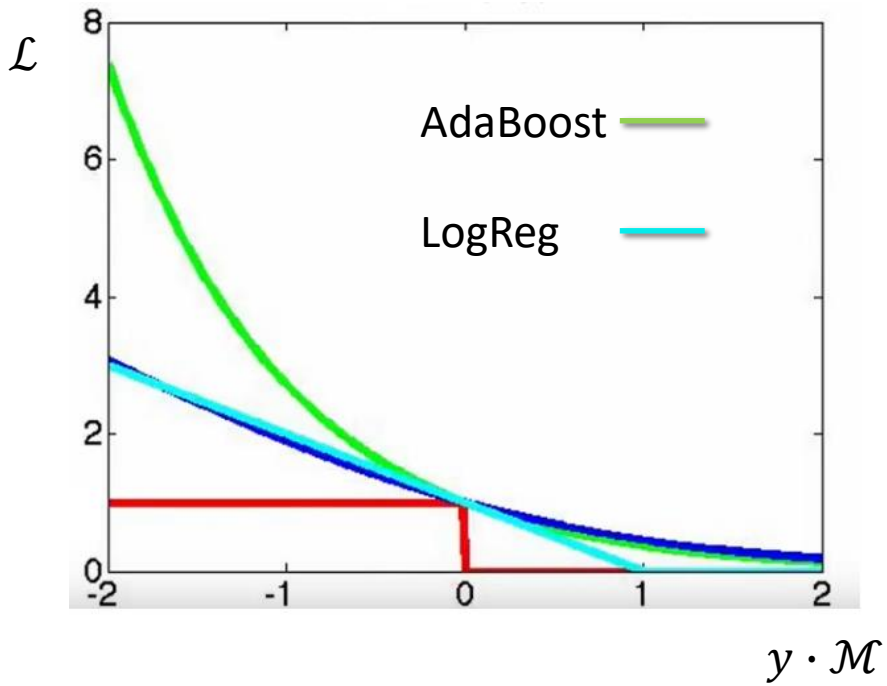
Very bad decision

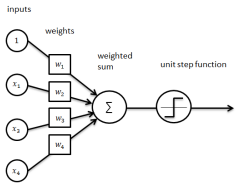
Close to good

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

Very good decision

Close to bad



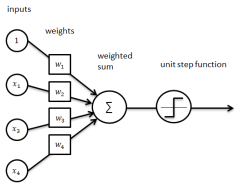


Loss function (IV)

- 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**

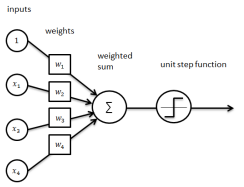
- Different loss functions determine upper limits w.r.t $1_{[y\cdot\mathcal{M}(\vec{x}_i)<0]}$ one:

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



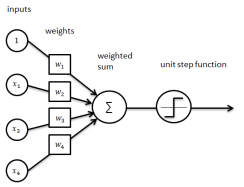
Over- and Under-fitting

- Regression with explicit weight determination algorithm (MSE)
- You can see that it's basically an optimization process..., is machine learning just an optimization problem? **NO – the main difference is a generalization**!
- In the **learning process**, we minimize the **training error**, but what we really want is a **minimal test error** (generalization error). So we are interested in the expected value of the test error determined for any cases that were not "shown" to the model
- $$\frac{1}{m^{(ZTr)}} \left\| \tilde{\vec{y}}^{(TrS)} - \vec{y}^{(TrS)} \right\|_2^2 \rightarrow \frac{1}{m^{(ZT)}} \left\| \tilde{\vec{y}}^{(TS)} - \vec{y}^{(TS)} \right\|_2^2$$
- It seems to be a real pickle... What is the relationship between these two quantities?



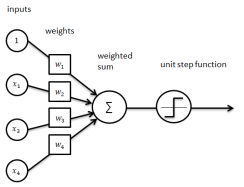
Over- and Under-fitting

- The answer is found through **statistical learning theory** – data generating process: in short, we assume that we draw training and test sets from the same probability density distribution $p_{Data}(\vec{x}, e)$
- From the point of view of statistics, we say that the training and test sets have identical distributions and each case in both sets is mutually independent of the other events
- So we have one data **generating distribution** for both sets.
- From this it follows that the expected value of the error for the test set must be the same as the expected value for the training set, e.g. $E\left[MSE_{TrS}^{(i)}\right] = E\left[MSE_{TS}^{(i)}\right]$
- And further, we can assume that there is such a set of parameters $\vec{w}$ for which $MSE_{ZTr}^{(a)} = MSE_{ZT}^{(a)}$
- **But, in practice, we act differently...**



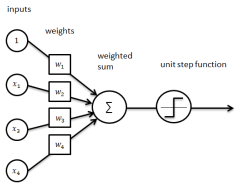
Over- and Under-fitting

- When learning in practice, we never proceed in this way: (1) set model parameters, (2) sample the training and test set
- Our typical pipeline: (1) sampling the training set, (2) determining the $\vec{w}$ by minimizing the training error, (3) sampling the test set and determining the error
- The main "learning paradigm": get as little training error as possible and as little difference as possible between a training and a test errors gap
- The above considerations lead to two fundamental learning problems: over-fitting and under-fitting of models – the complexity of models
- **Too low complexity –** problem with reproducing the TrS, **too much complexity** – the problem of over-matching (the model accurately reproduces the properties TrS but fails for TS)
- E.g. for a regression problem: $\tilde{y} = wx + b \rightarrow \tilde{y} = w_1 x^1 + w_2 x^2 + w_3 x^3 + b$



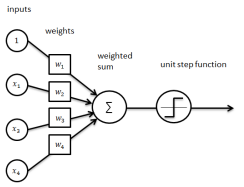
Optimal model

- Question: Is it possible to create an automatic procedure that looks for a model of optimal complexity, so that the training error and the generalization error are consistent with each other?
- The answer (in part) is provided by the statistical theory of learning and the **Vapnik–Chervonenkis** theorem (VP-dimension)
- VP-dimension gives a quantitative measure of complexity for binary classifiers. VP-dimension is the largest possible set of elements of a test set that can be classified into different classes.
- Unfortunately, practically using this method can be difficult! We can draw the following conclusions instead. There should be a number that limits from above the possible difference between a training error and a test error. This number increases as the complexity of the model increases and decreases for larger sets.
- For example, a "generalization gap" (the difference between a training error and a generalization one) can be determined. When it reaches the minimum, we have a model of optimal complexity.



„No-free-lunch" theorem

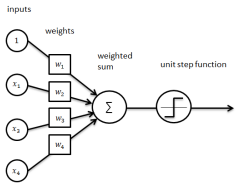
- This theorem was formulated by Wolpert and concerns the universality of learning algorithms.
- We can consider the following problem: is there the best algorithm that can always beat other algorithms when performing a certain task? Otherwise, what happens when we consider all possible distributions that generate data?
- It turns out that every learning algorithm will have a similar stream of errors when classifying new, previously unanalyzed events!! The NFL theorem can also be interpreted that any algorithm will be characterized by a classification quality that means assigning all points to the same class.
- To put it another way: **there are no universal algorithms**, but for a certain specific class of distributions we can find models that will **reach a very good classification quality.**



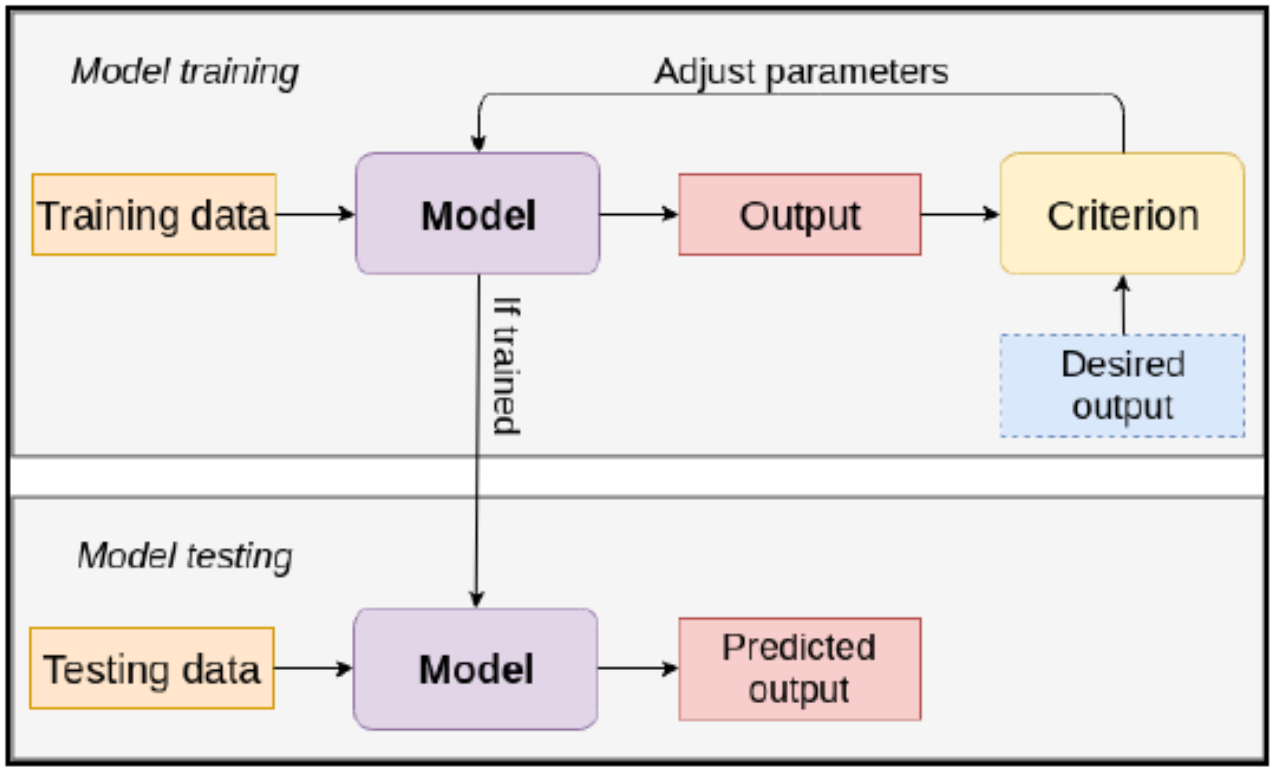
Regularization

- Let's assume that our learning algorithm has a space of hypotheses consisting of different types of polynomials. **There is a method that allows us to some extent to "guide" an algorithm to a particular type of function**.
- In practice, this is done by **modifying the loss function** by adding a component controlled by a parameter that we **tune before starting the training**.
- For example, for our regression problem we can write down: $\mathcal{L}' = MSE_{TrS} + f(\lambda)$
- **Both the form of the *f* function and its meaning can be different** (e.g. Ridge type regularization, Lasso type regularization, weight loss technique, dropout, etc.)
- For example: $\boldsymbol{\mathcal{L}' = MSE_{TrS} + \lambda \vec{w}^T \vec{w}}$ (weight decay technique). The effect of such a modification will be: in the case of large values λ, weights (polynomial parameters) will tend to take small values, for λ with intermediate values, the values of the weights will grow, ...

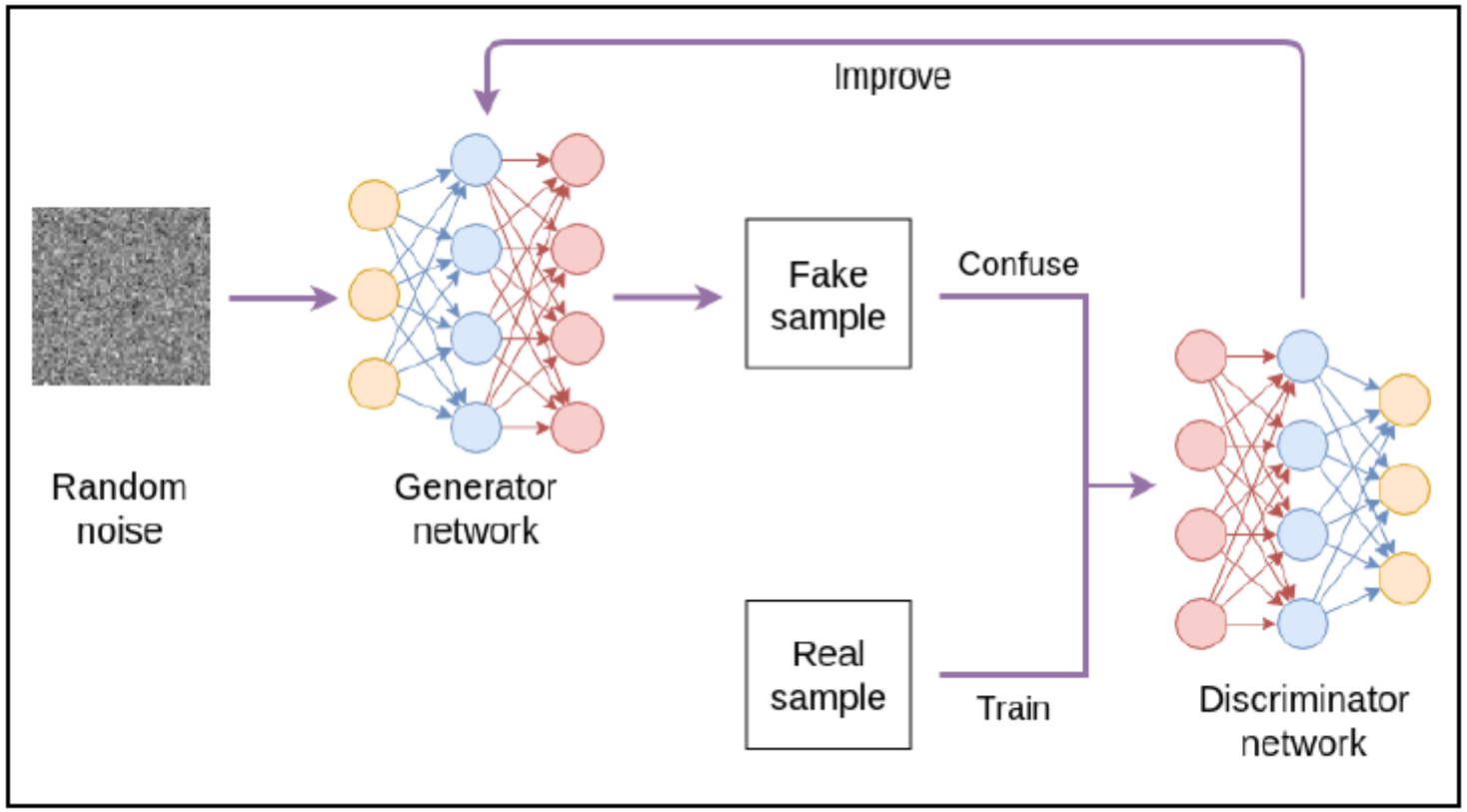
Cool ones



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:

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

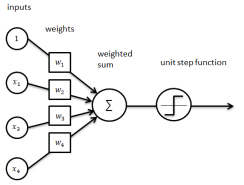
- Here: $\vec{x}$ - real samples, $\vec{x}^* = \mathcal{G}(z)$ - generated samples (z represents noise), $\mathbb{E}_{\vec{x}}[f]$ is the average value of any function over the sample space

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

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

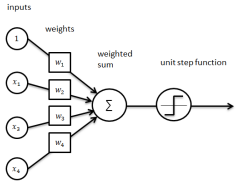
- Model $\mathcal{G}$ must trick the discriminator, thus, it minimise the $1 - \mathcal{D}(\vec{x}^*) = 1 - \mathcal{D}\left(\mathcal{G}(z)\right)$

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

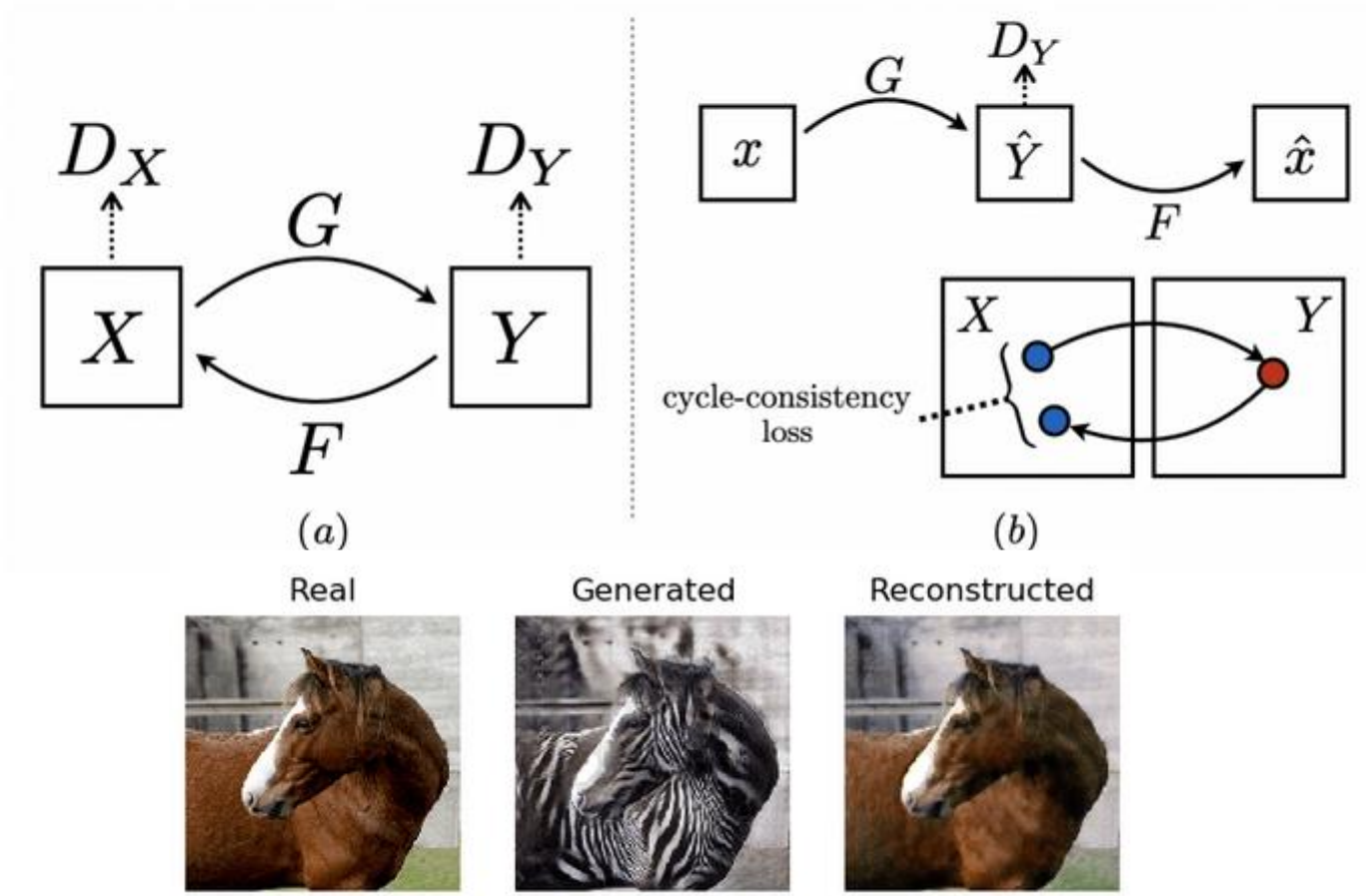


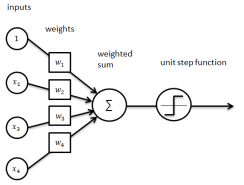
ML GEMS (I) - GANs

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

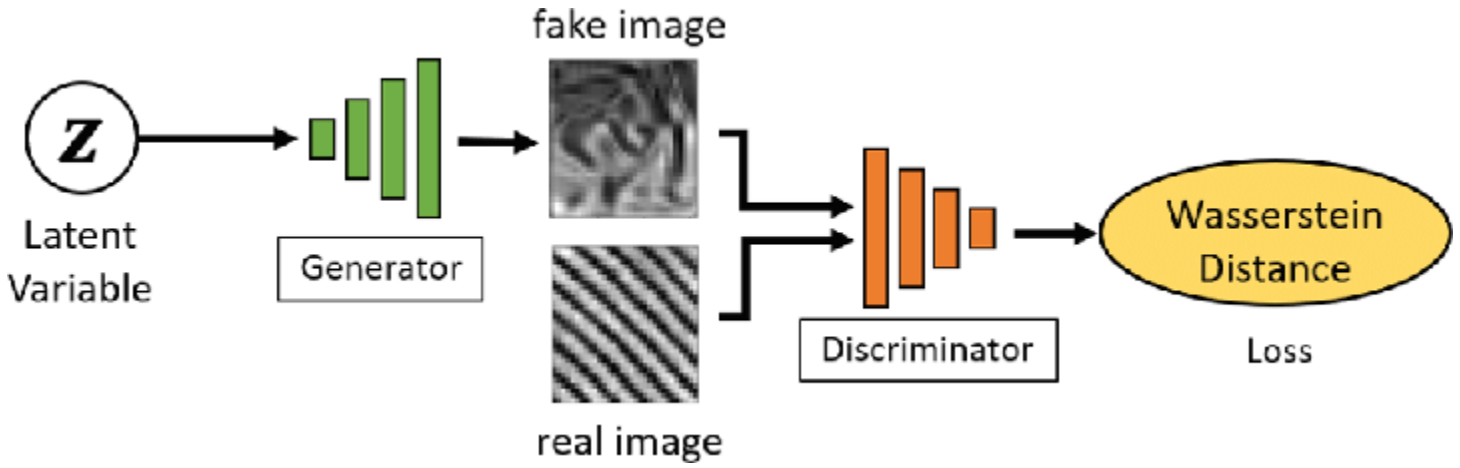

inputs
weights
weighted sum
unit step function

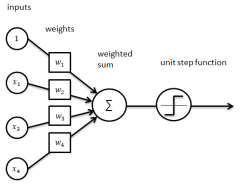
CycleGAN


D_X
D_Y
G
X
Y
F
(a)
D_Y
G
x
$\hat{Y}$
$\hat{x}$
F
X
Y
cycle-consistency loss
(b)
Real
Generated
Reconstructed



WGAN – Wasserstein GAN

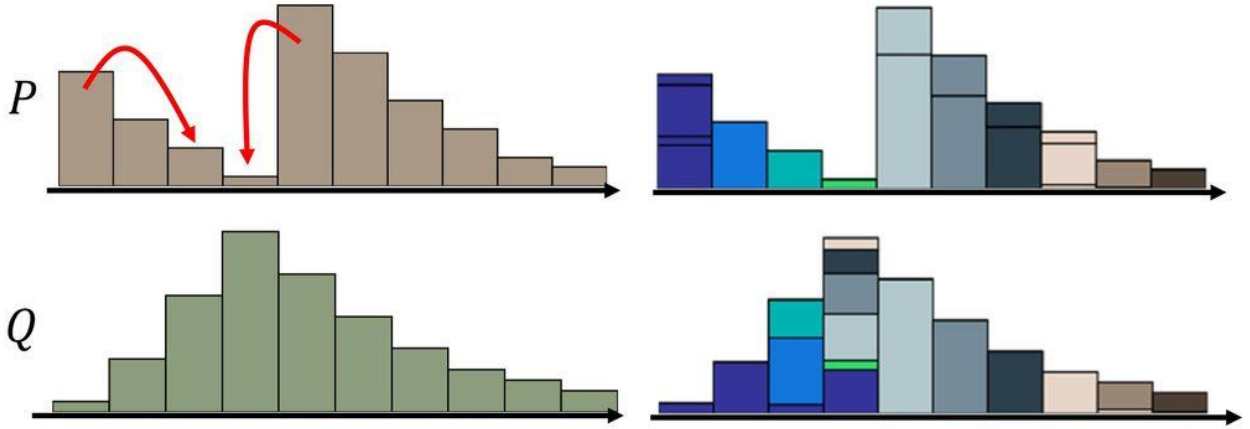




Optimal transport – aka W-distance

Earth Mover's Distance

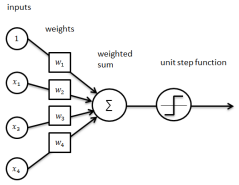
Best "moving plans" of this example



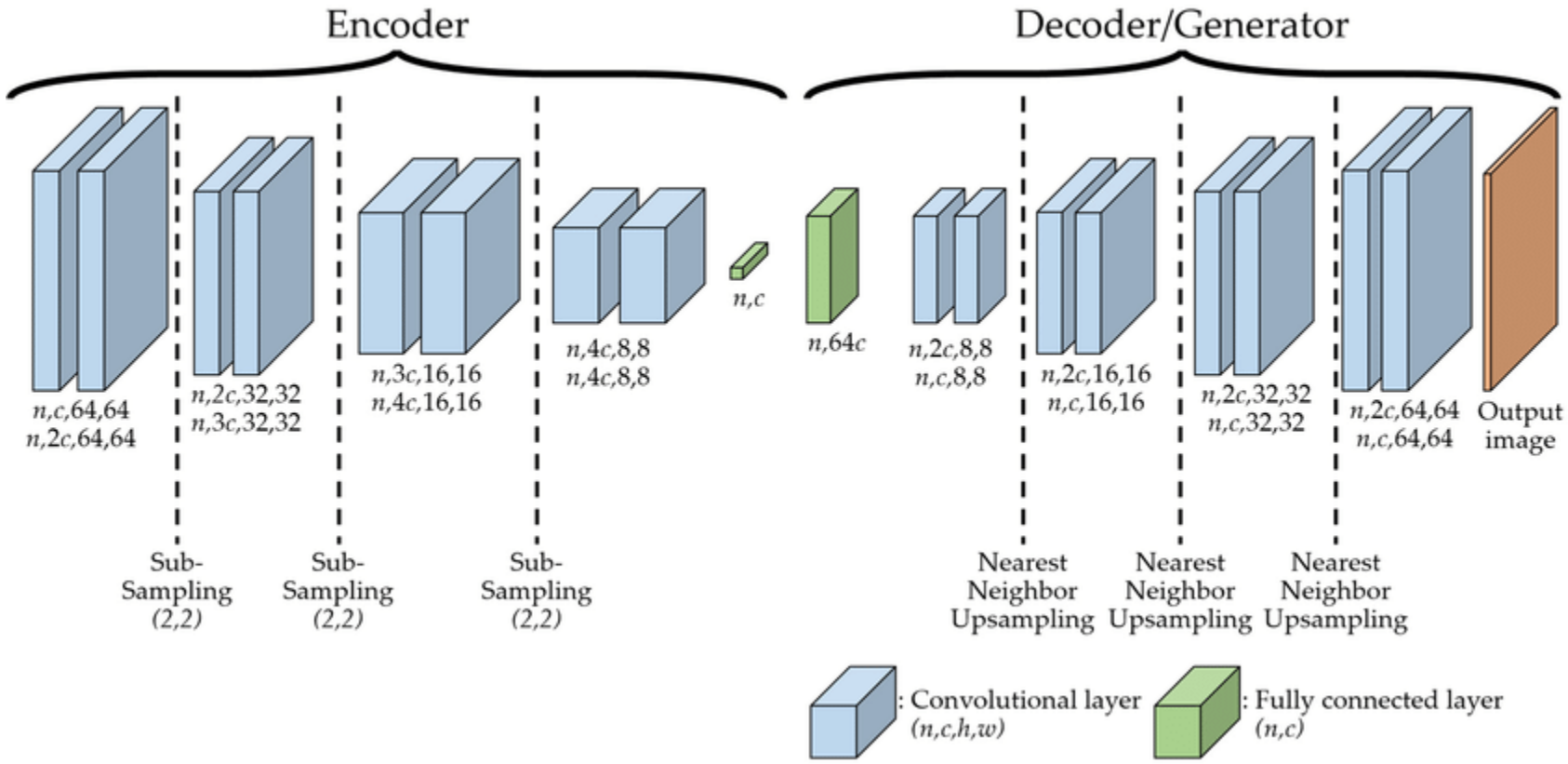
There many possible "moving plans".

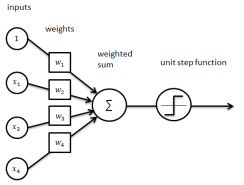
Using the "moving plan" with the smallest average distance to define the earth mover's distance.

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

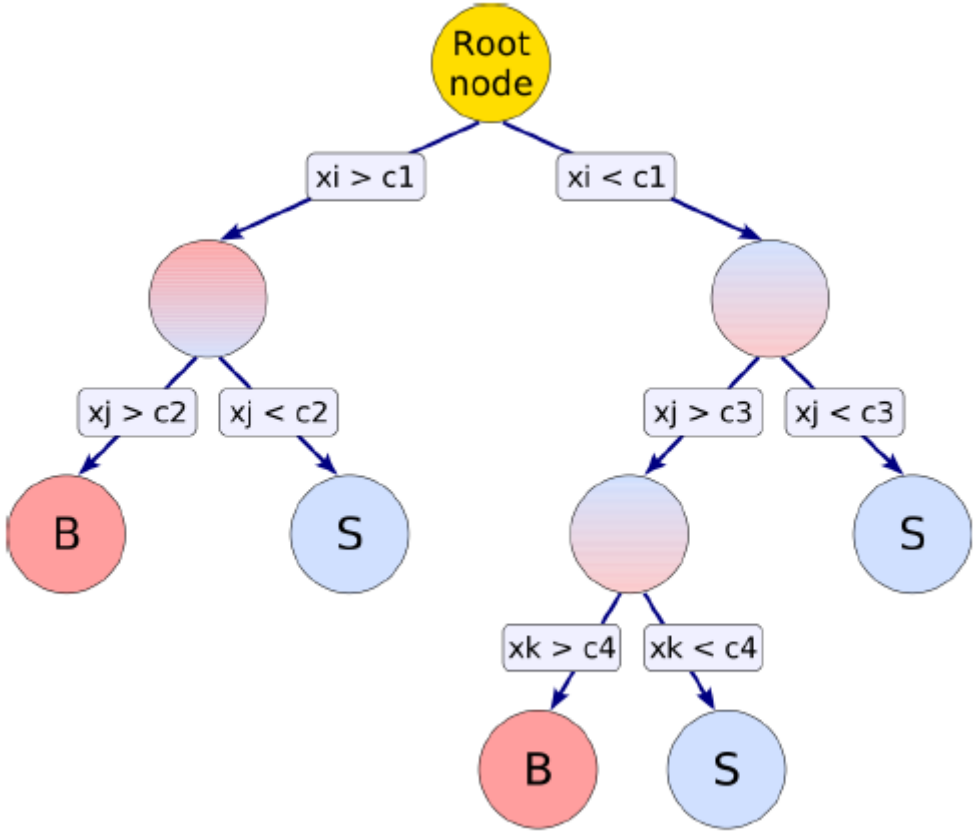


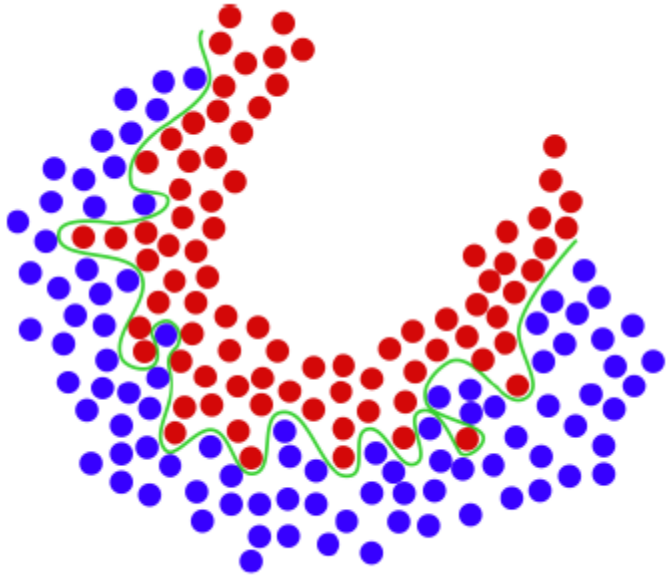
Autoencoders

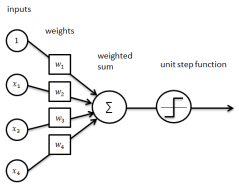




Decision trees



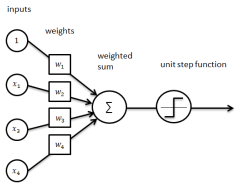




HEP landscape

- 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

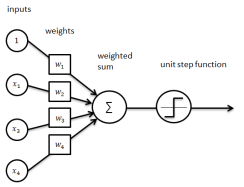
- A comprehensive repository regarding current status: https://iml-wg.github.io/HEPML-LivingReview/ (**A Living Review of Machine Learning for Particle Physics**)



HEP landscape

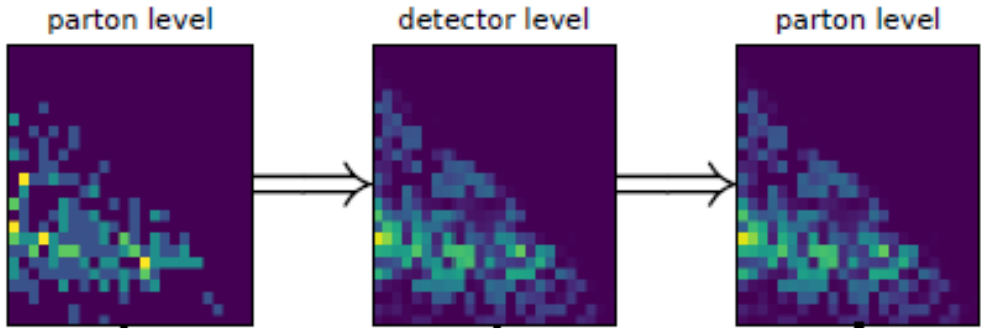
- Very interesting overview: **„Machine Learning in High Energy Physics Community White Paper”** (https://arxiv.org/abs/1807.02876)

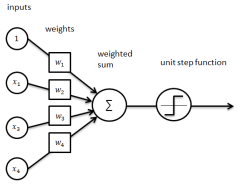
  - Challenges of learning Standard Model
  - Speeding simulation via generative models
  - Computing resources and sustainability
  - Engaging commercial partners (new LHCb trigger based on GPU processors)
  - Interpretability of models
  - Uncertainty of predictions (just beginning this large subject)



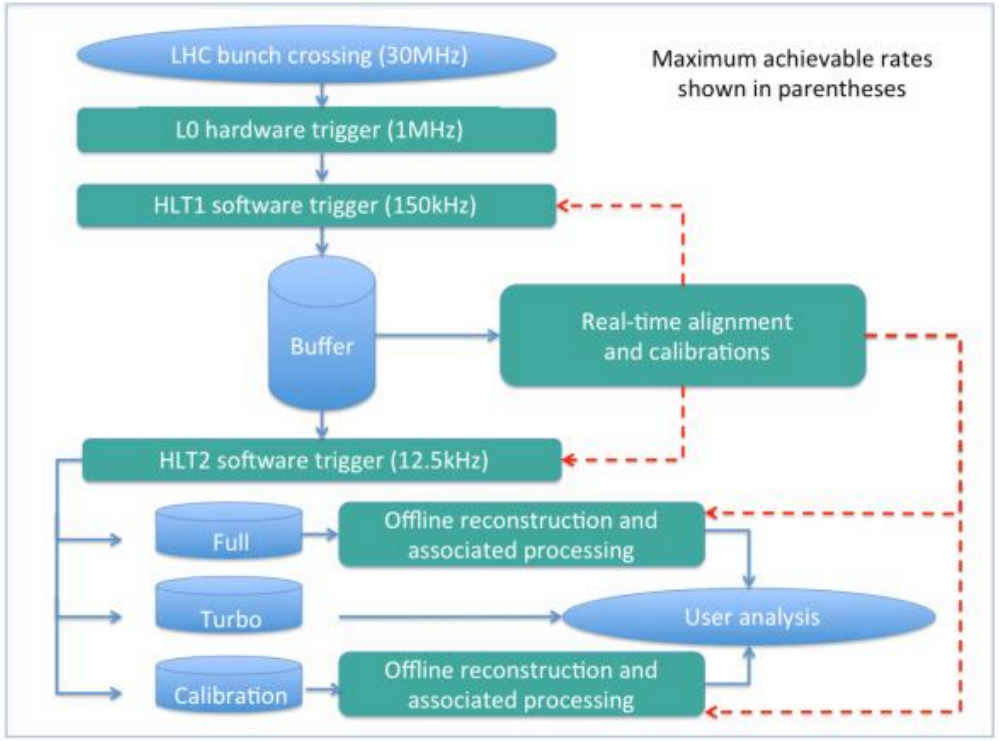
HEP landscape

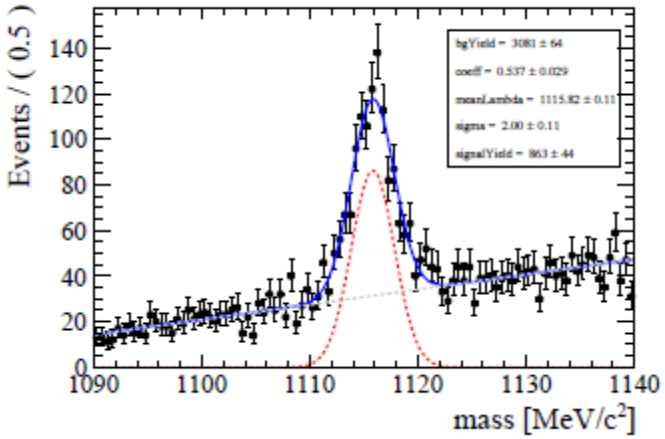
- „**Generative Networks for LHC events**” (https://arxiv.org/abs/2008.08558)
  - Physics specific challenges: phase-space integration, conservation of 4-momentum
  - Parton shower and matrix elements modelling
  - CycleGANs for understanding the patron showers





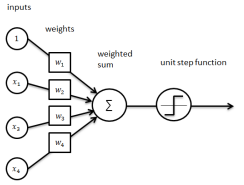
LHCb Trigger (Run 2)



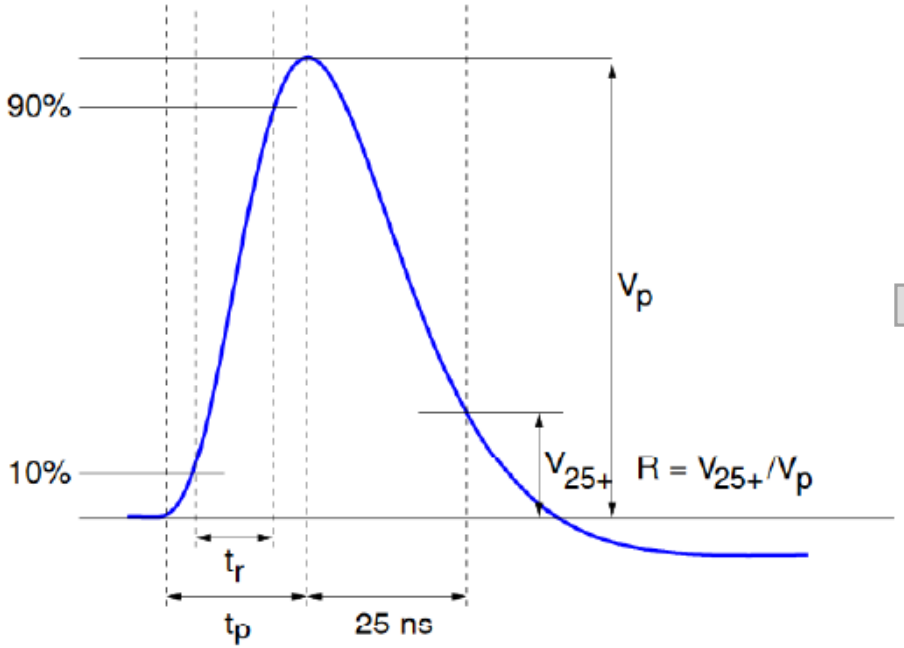


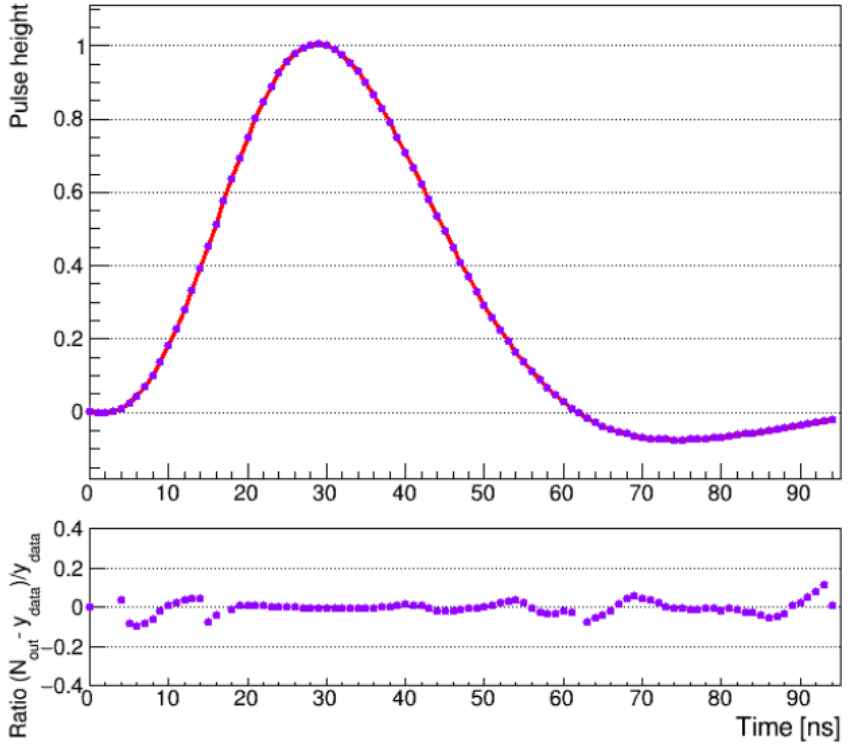
Long-lived tracking in HLT using XGBoost algoritym

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!

The greatest challenges for ML

- One of the most hot topics of ML – understand the uncertainties
- At the moment we do not have such powerful tools as Statistics wields (confidence interval, for instance)
- Interpretability is one way to tackle this problem, but it is just the beginning

The greatest challenges for ML

- ML and more generally AI can do a lot of good for human kind but it can also be a real danger
- ML does not do things as we do, but we can bias it and teach it to hate, have racial prejudice or become a religious fanatic
- So, we need to mind ethics for the future of ML and AI, especially taking into account how ubiquitous it is now
- Anyway, the future at best is uncertain and we need to understand this problem before it is too late…

Thanks! Hope you like it and you get inspired!

BACKUP

A simple one

Cross-entropy, better loss function

Count the "bad decisions" and penalise the model!

Mean Squared Error Loss

$$L_1 = \frac{1}{n} \sum (y_i - \tilde{y}_i)^2$$

$\tilde{y}_i$ → predicted label

n → # events

y_i → true label

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 $d : \{x_1\}$

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

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

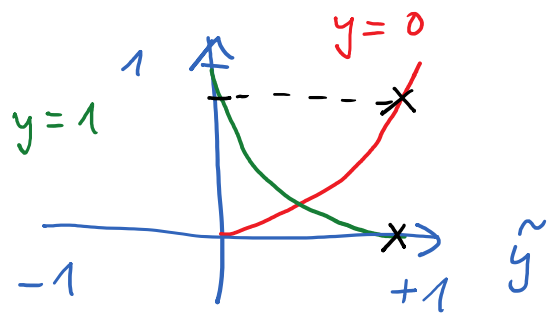
Gedanken experiment (two classes)

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

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

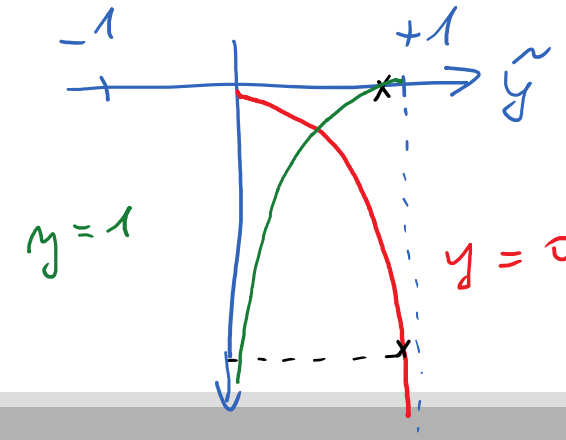
Visualisation please!

Visualise!



$y = 0,\ \tilde{y} = 0.8$ (bad decision)

$$\begin{cases} L_1 = 0.81 \\ \dfrac{\partial L_1}{\partial \tilde{y}} = 1.81 \rightarrow \text{model penalty} \end{cases}$$



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

Be a responsible punisher …

Penalty ≡ change of parameters

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

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

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

Classification → bad class → big deal !