CERN openlab

# Neural Networks and Deep Learning: an introduction

*2021 JENNIFER2 SUMMER SCHOOL*

Sofia Vallecorsa

23/07/2021

# Outline

- Introduction
- Basic Concepts
  - Nodes
  - Feed-forward networks
  - Multi Layer Perceptrons
- The learning process
  - Learning models
  - The training process
  - Loss functions
  - Adaptive SGD methods
- Overfitting and regularization
- Alternatives to ANN
- Examples

**Material**:

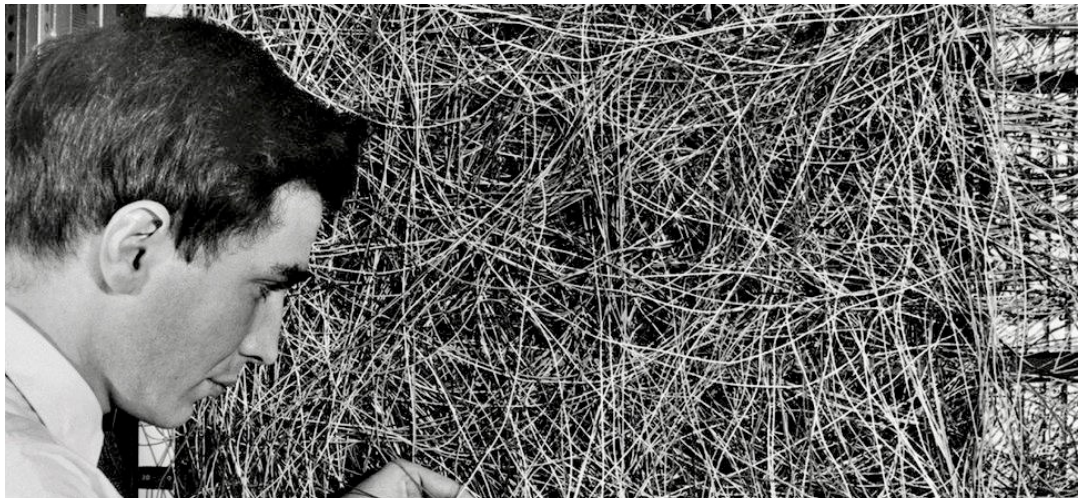
- M. Kagan, Introduction to Machine Learning, CERN openlab summer student lectures
- G. Louppe, Deep Learning, https://github.com/glouppe/info8010-deep-learning
- François Fleuret, Deep Learning Course at UniGE: https://fleuret.org/dlc/

# Machine Learning

**Mathematical models** learnt from data

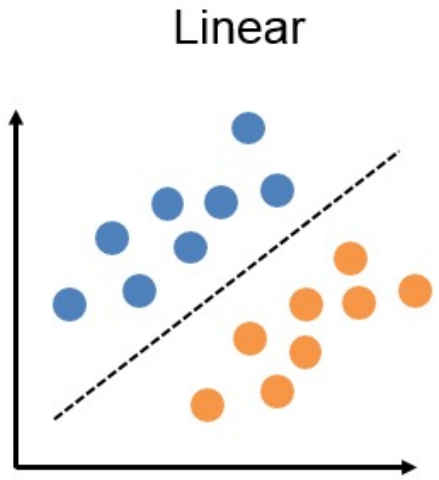
- Characterizes patterns, regularities, and relationships amongst variables
- Chosen according to task / available data

Frank Rosenblatt working on the Mark I perceptron (1956)



**1957: IBM 704 system**


Rosenblatt Perceptron could solve linear classification problems




1997:

IBM DeepBlue wins Kasparov (Chess)

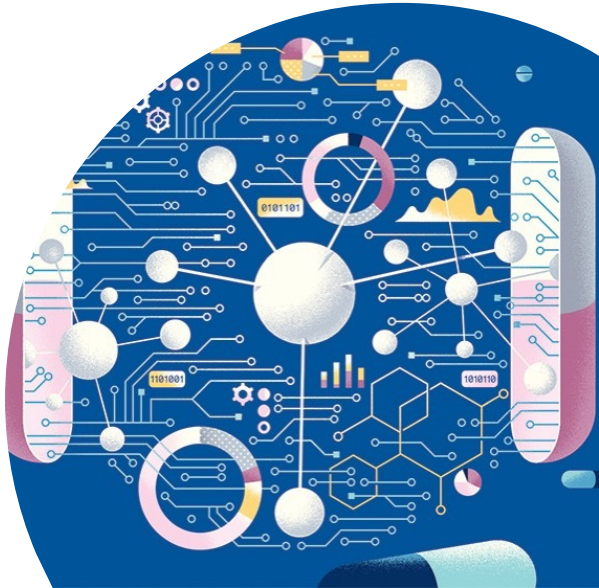
# Endgame…

2017:

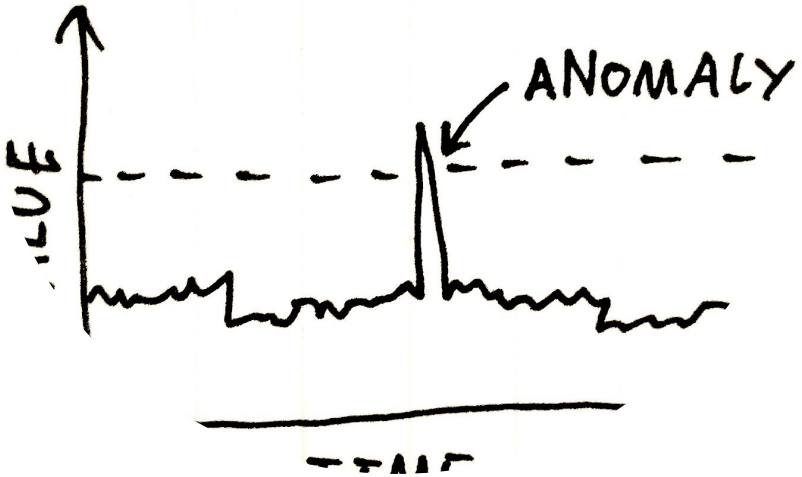
DeepMind AlphaGo wins Ke Jie (Go)

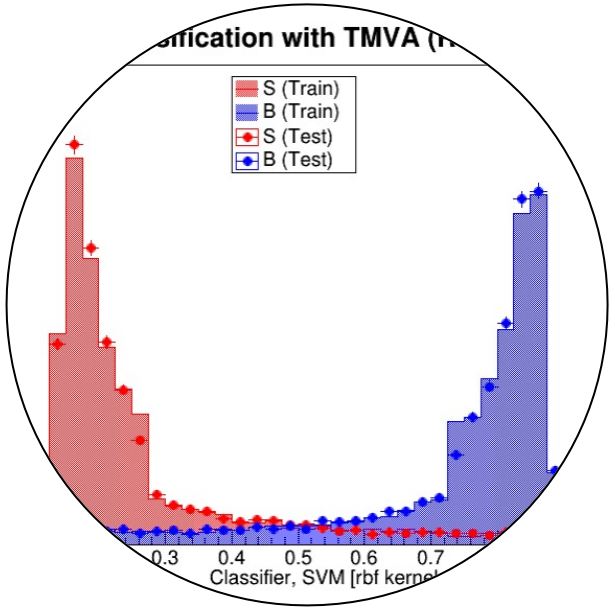



# Today’s applications…

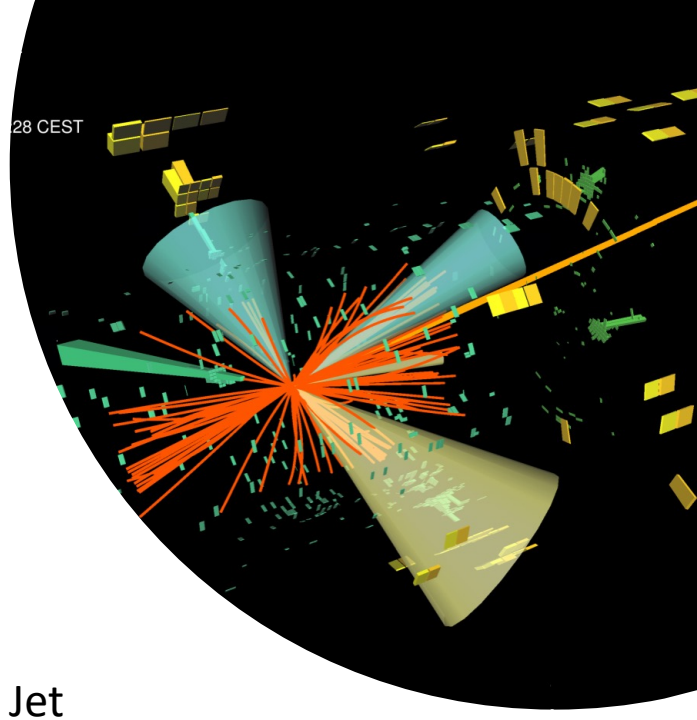


Anomaly detection

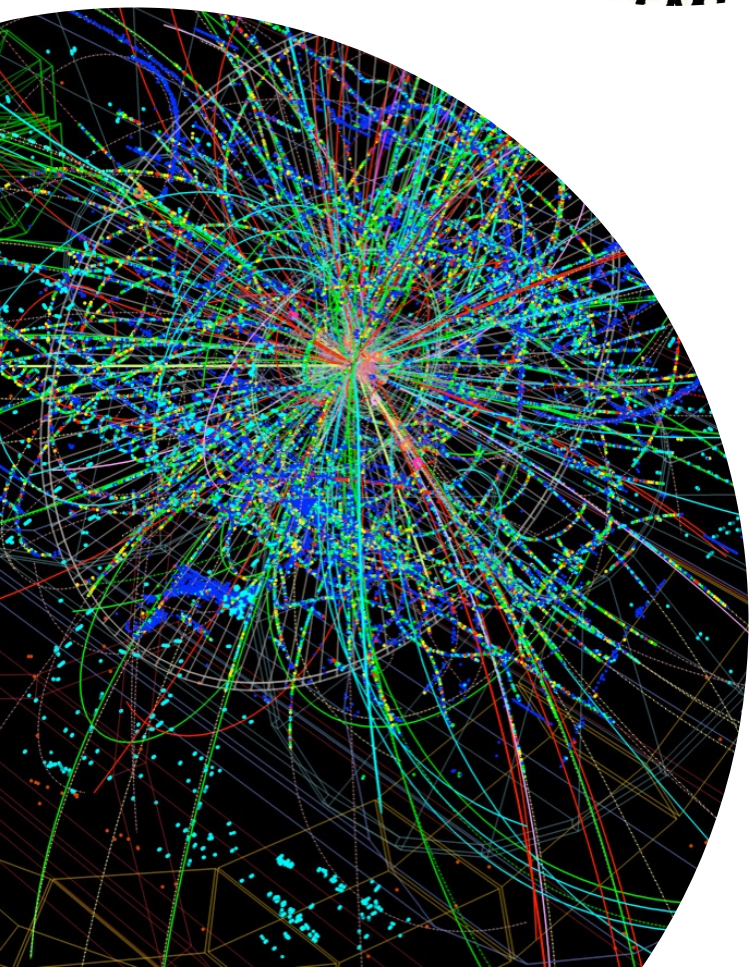


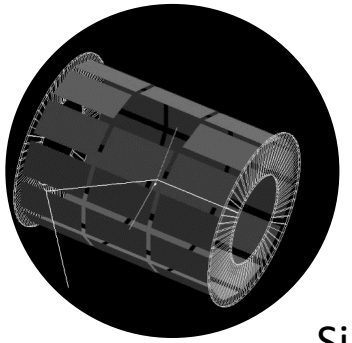




Classification




Jet reconstruction





Simulation

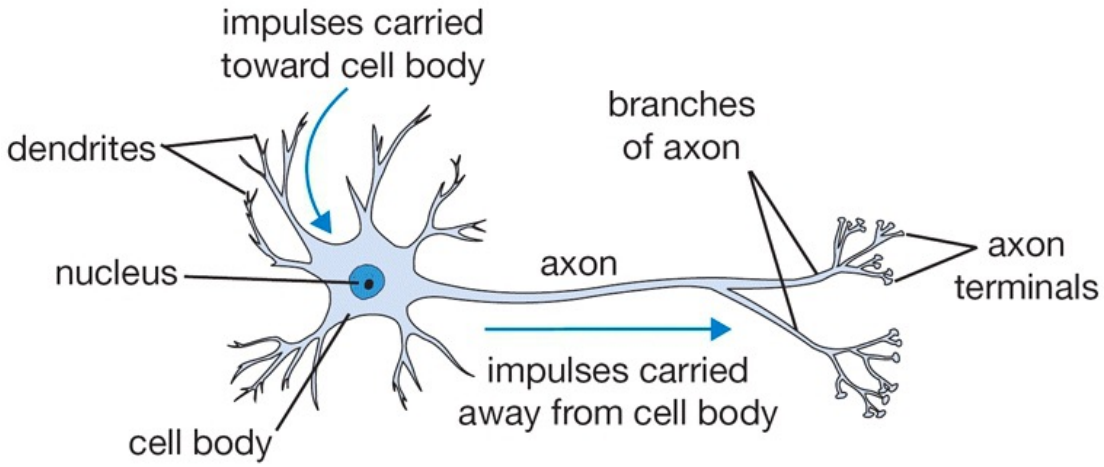
# ...at CERN

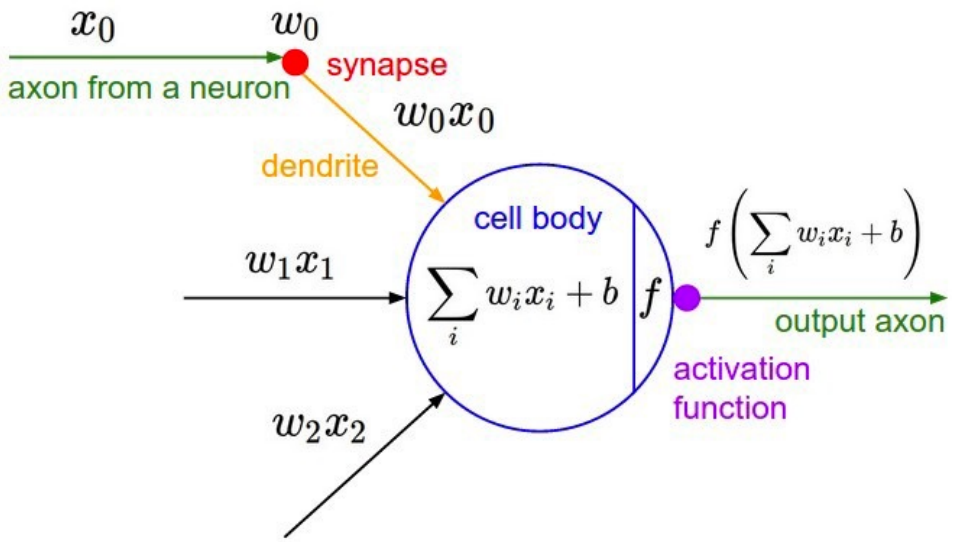
Reconstruction

Hardware monitoring and optimization

# Artificial Neural Network

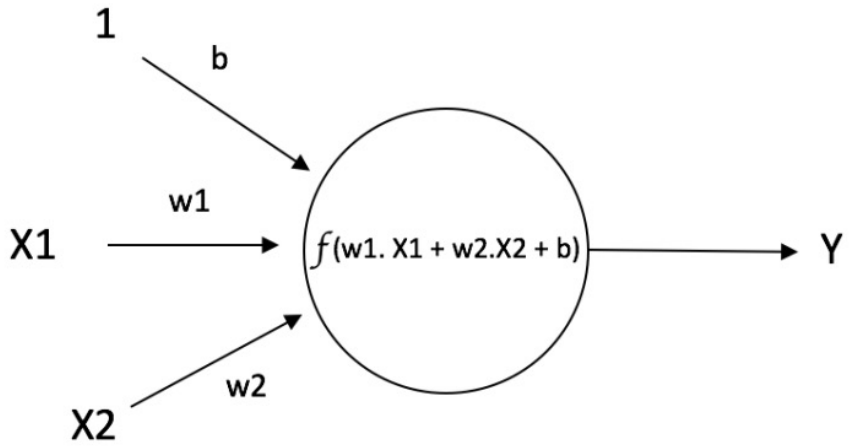
ANN are computational models inspired by biological neural networks.



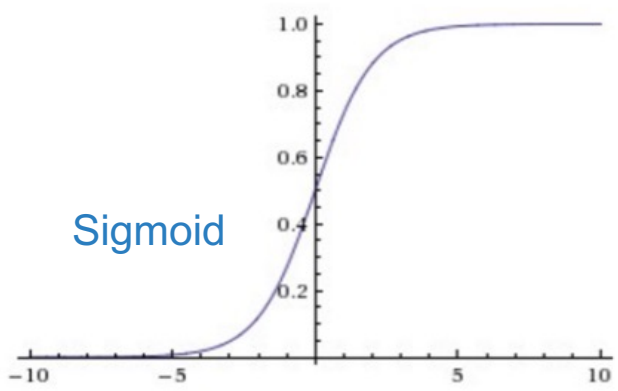



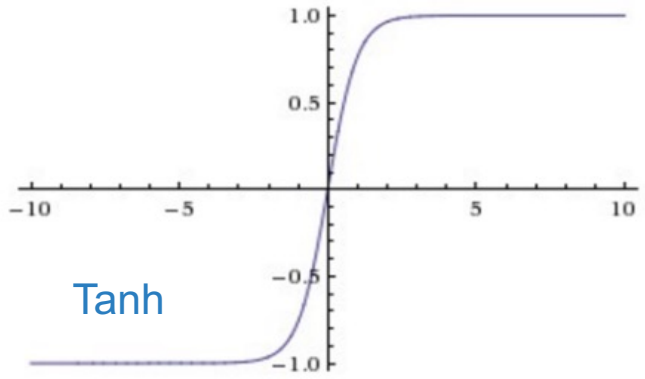
# Node

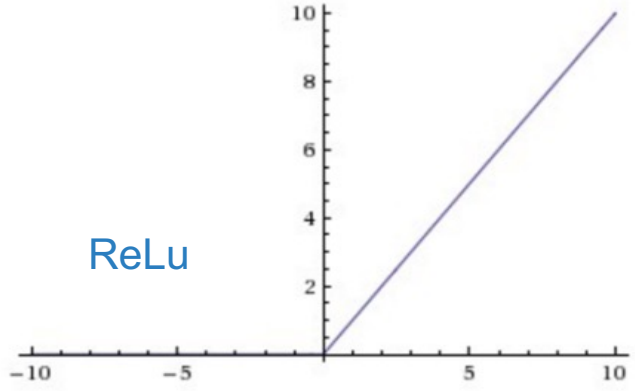
- Receives input from other nodes, or an external source
  - Each input has an associated **weight**
- Computes an output
  - Applies an **Activation Function** to the weighted sum of its inputs
- A node is characterized by its parameters




$$Y = f(w1.\,X1 + w2.X2 + b)$$



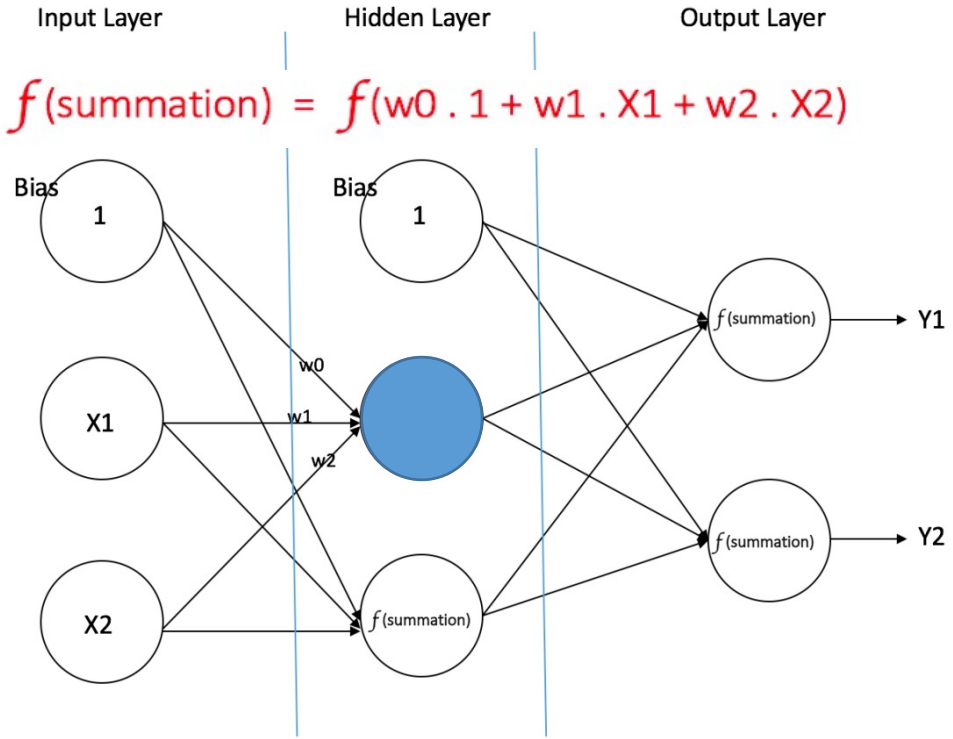
# Feed-forward networks

Multiple nodes arranged in **layers**.

Nodes from adjacent layers
have **connections** (with weights).

**Ex. fully-connected layer**

**Multi Layer Perceptron** (MLP) contains one or more hidden layers




$N_{para}$ = 3 + 2 (nodes)
= 3x3 + 2x2 (weights) + 3 + 2 (bias)
= 18 parameters

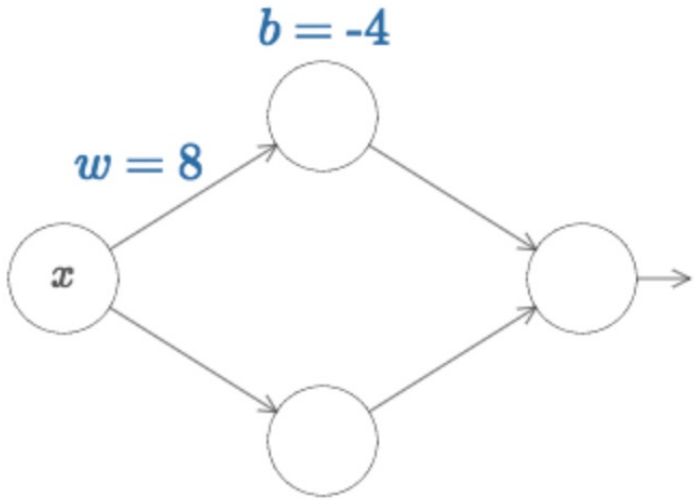
```
# forward-pass of a 2-layer NN:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # sigmoid activation function
x = np.random.randn(3, 1) # random input (3x1)
h1 = f(np.dot(W1, x) + b1) #  hidden layer activations (3x1)
out = np.dot(W2, h1) + b1 # output neuron (1x1)
```

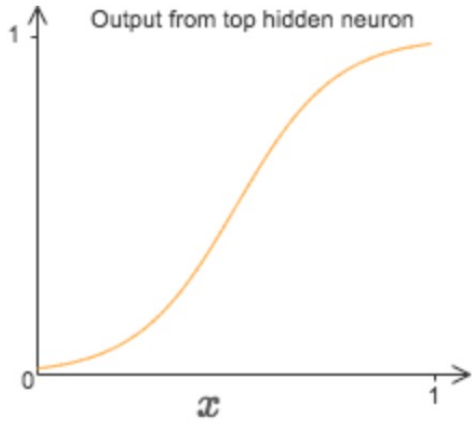
# NN as universal approximators

NN with at least one hidden layer are *universal approximators*

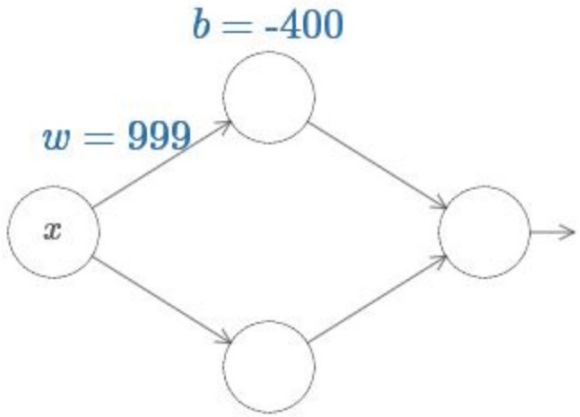
$\sigma(wx + b)$

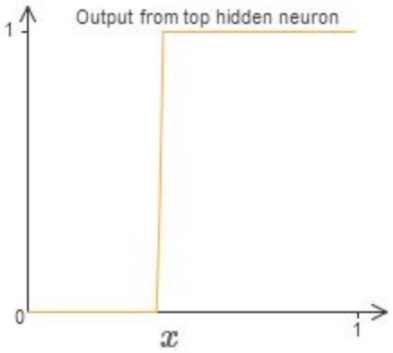
$\sigma(z) \equiv 1/(1 + e^{-z})$




Playing with the w, b parameters we can modify the shape of the sigmoid



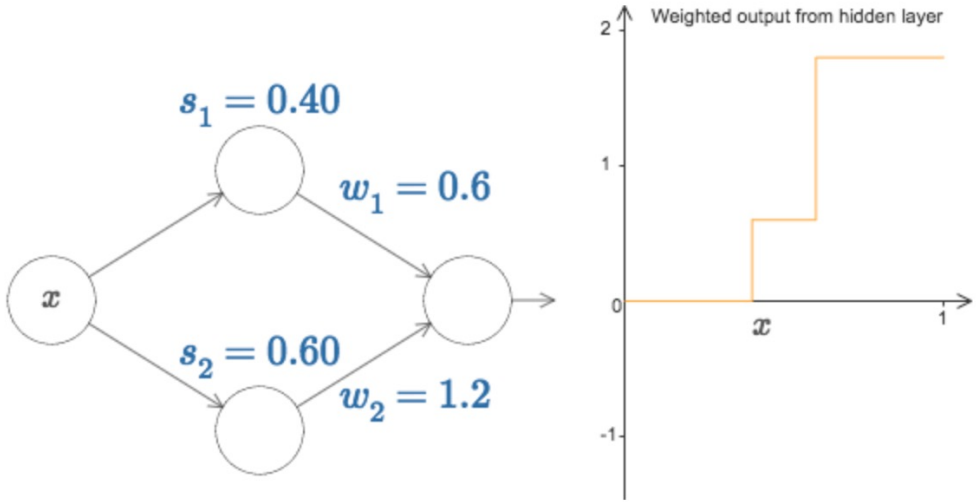




http://neuralnetworksanddeeplearning.com/chap4.html

*Or Approximation by Superpositions of Sigmoidal Function*

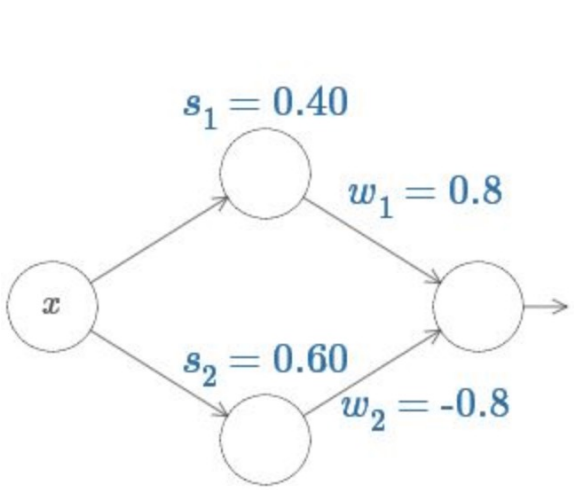
# NN as universal approximator

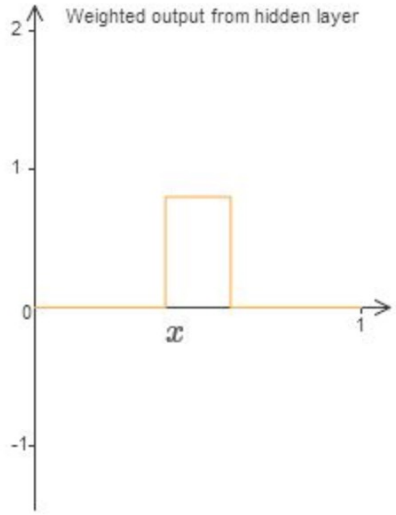
## We can add nodes and introduce “steps”









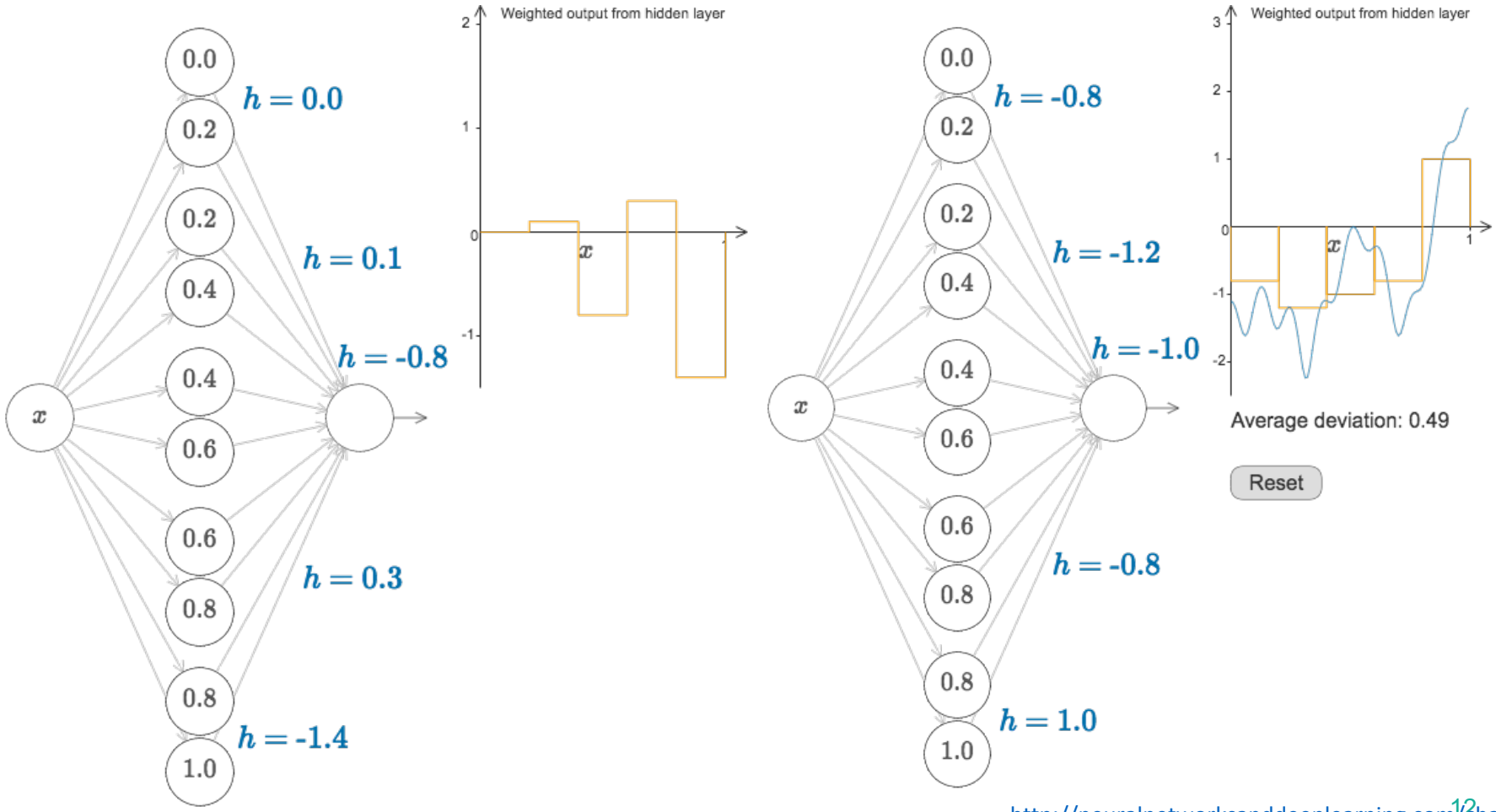




http://neuralnetworksanddeeplearning.com/chap4.html
*Or Approximation by Superpositions of Sigmoidal Function*

# NN as universal approximators

*Increasing complexity*

NN with a single hidden layer can be used to approximate any continuous function **to any desired precision**



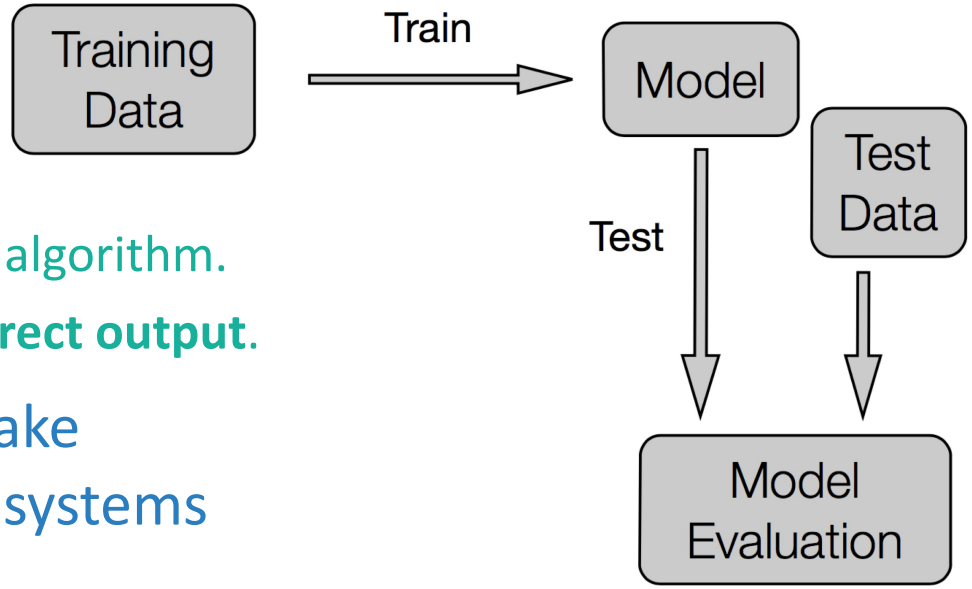

http://neuralnetworksanddeeplearning.com/chap4.html

*Or Approximation by Superpositions of Sigmoidal Function*

# (Machine) Learning

*Giving computers the ability to learn without explicitly programming them*

(A. Samuel, 1959)




**Learning:** estimate statistical model from data

A result of **altering the network's weights**, with some kind of algorithm.

Find a set of weights so that the NN can **map input to the correct output**.

**Prediction and Inference:** using statistical models to make predictions on **new data points** and infer properties of systems

**Training set** : to fit the model

**Validation set**: to check performance on independent data and tune-hyper parameters

**Test set**: final performance evaluation

# Supervised learning

The **desired output** (target) **is known** at training time.

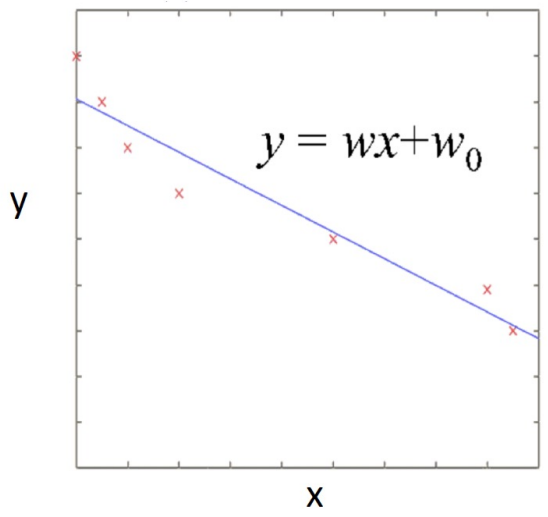
calculate an error based on target output and NN output

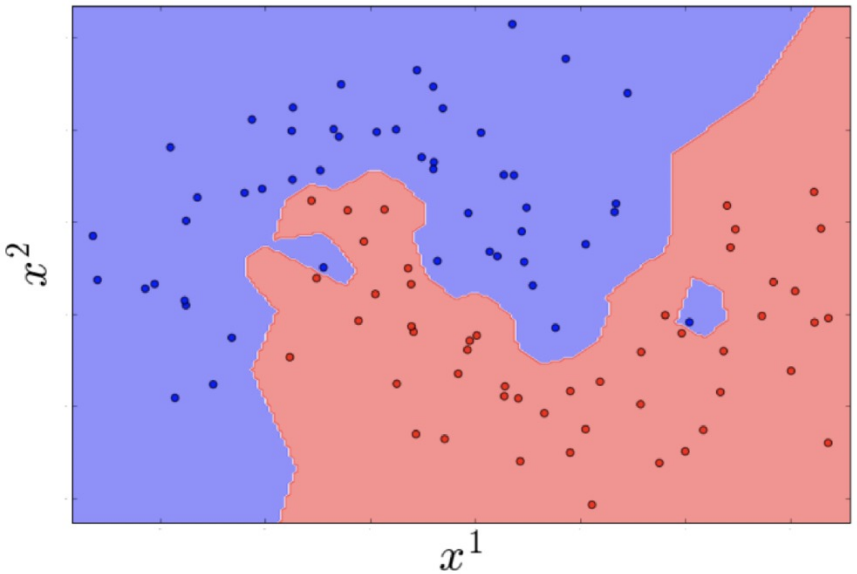
use error to make corrections by updating the weights.

Given N examples with features $\{x_i\}$ and

targets $\{y_i\}$, learn function mapping h(x)=y

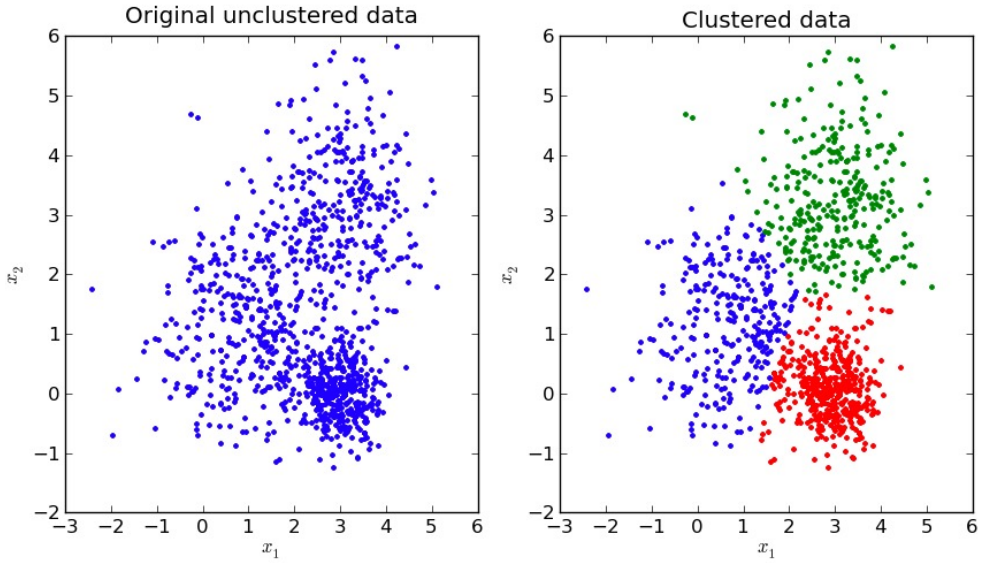
- Regression:
  Y = Real Numbers
- Classification:
  Y is a finite set of labels (i.e. classes)




CERN openlab

# Unsupervised Learning

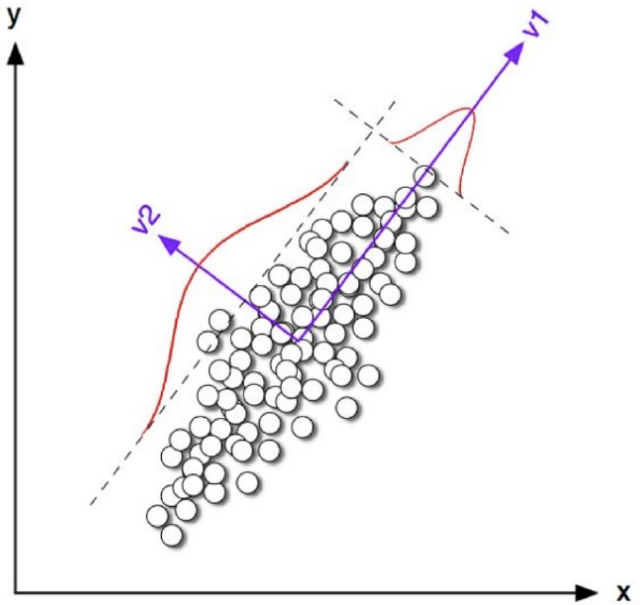



No target output is used at training time

**NN finds pattern within the inputs** (data mining)

Given some data D={$x_i$}, but no labels, find structure in the data

- Clustering:
  partition the data into groups
- Dimensionality reduction:
  find a low dimensional (less complex) representation of the data with a mapping Z=h(X)




https://lazyprogrammer.me/tutorial-principal-components-analysis-pca/

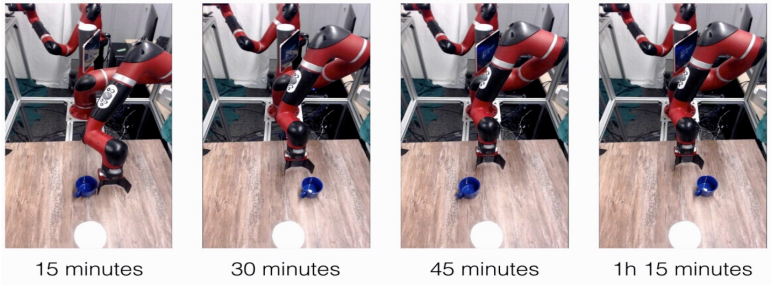
# Reinforcement learning

**Agent interacts** with environment and receives **reward** after an **action**

- Learns through **trial-and-error**

Agent follows certain **policy $\pi$**: $S \rightarrow A$

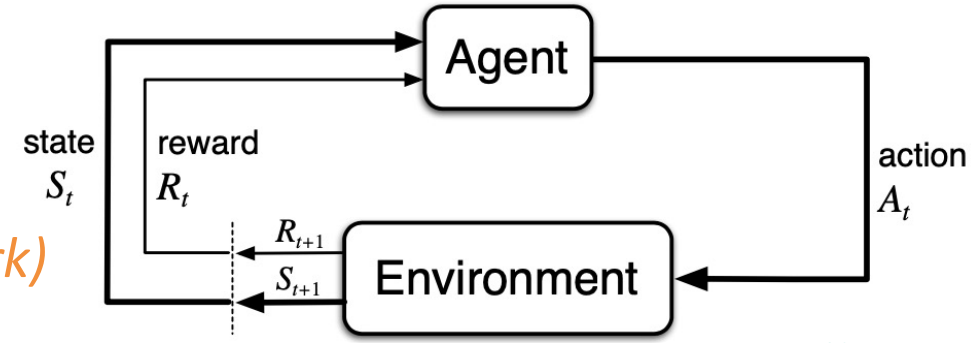
- **Find optimal policy $\pi^*$ maximizing return:** $G_t = \sum_k \gamma^k R_{t+k}$




source

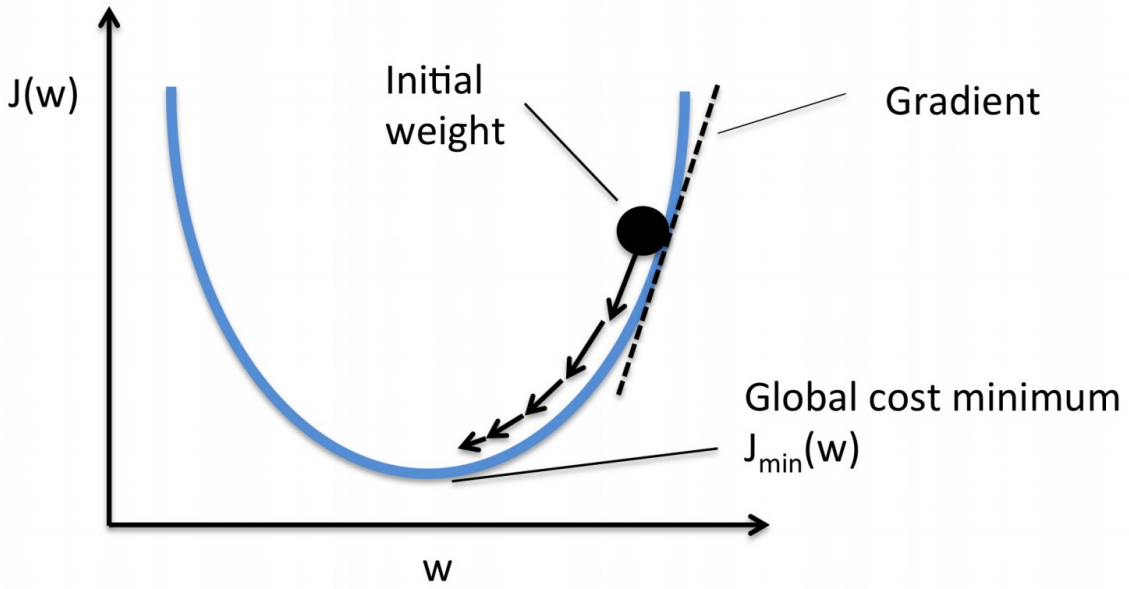
**Expected return** can be estimated through ***value function Q(s, a)***

- **"What's the best action to take in each state" *=> greedy policy:*** *take action that maximizes Q(s,a)*
- **Not a priori known, but can be learned iteratively**
- **Q-learning** – learn Q(s, a) using **function approximator**
  - DQN: Deep Q-learning *(feed-forward neural network)*



# The training process

*Learning as an optimization problem*




- Random NN **initialisation**
- Design a **Loss function**
  - Differentiable with respect to model parameters
- Find best parameters which minimize loss
  - Adjust parameters to reduce loss function
    - Calculate gradients wrt to weights
    - Weights update
  - Repeat until parameters stabilize

**Gradient evaluated over batch of data**

Too small → very noisy and scattering

Too large → information dilution and slow convergence

# Loss function

Slide from M. Kagan

Typical Supervised Learning use case:

a term comparing prediction **h** with target **y** and a regularizer, penalizing certain values of w

$$\arg\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N} L(h(\mathbf{x}_i;\mathbf{w}), y_i) + \lambda\Omega(\mathbf{w})$$

Average expected loss

Model regularization

## Example Loss Functions



- Square Error Loss:
  - Often used in regression

$$L(h(\mathbf{x};\mathbf{w}), y) = \left(h(\mathbf{x};\mathbf{w}) - y\right)^2$$

- Cross entropy:
  - With y ∈ {0,1}
  - Often used in classification

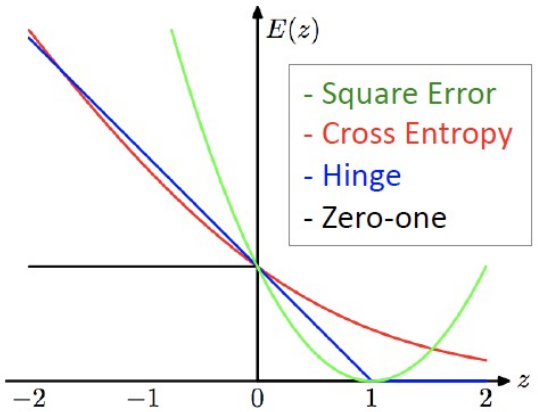
$$L(h(\mathbf{x};\mathbf{w}), y) = -\,y \log h(\mathbf{x};\mathbf{w}) - (1-y)\log(1-h(\mathbf{x};\mathbf{w}))$$

- Hinge Loss:
  - With y ∈ {-1,1}

$$L(h(\mathbf{x};\mathbf{w}), y) = \max(0, 1 - yh(\mathbf{x};\mathbf{w}))$$

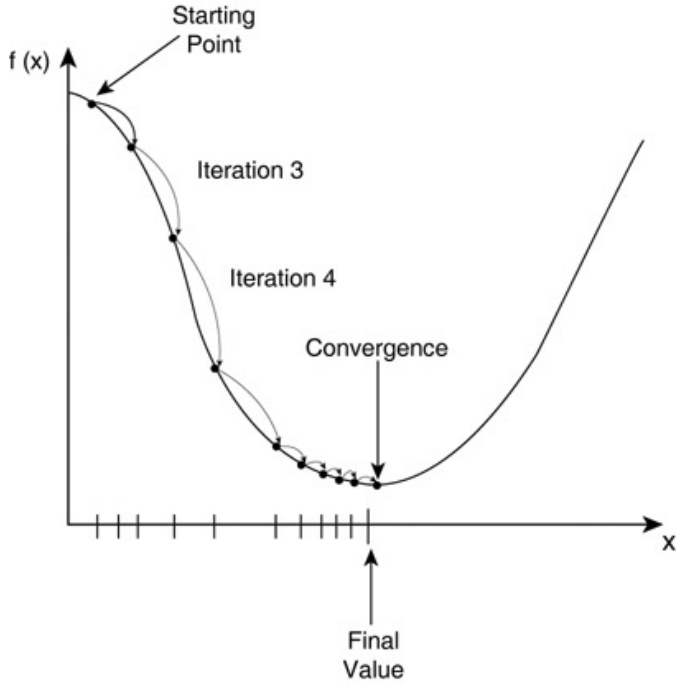
- Zero-One loss
  - With h(x; w) predicting label

$$L(h(\mathbf{x};\mathbf{w}), y) = 1_{y \neq h(\mathbf{x};\mathbf{w})}$$




[Bishop]

# Gradient Descent




**Convex** loss

- Single global minimum
- Iterations toward minimum

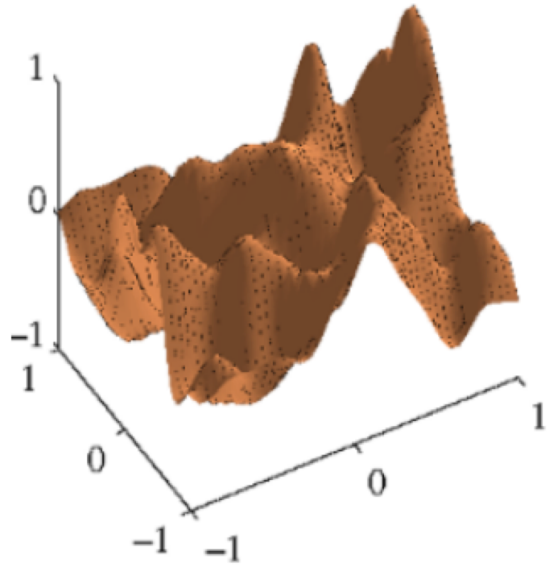
Noisy estimates average out

Scales well with dataset and model size

**Non convex** loss

- Stochastic behavior can allow “jumping” out of bad critical points
- Get stuck in non-optimal minima
- Convergence issue!

Compute gradient on one event at a time (stochastic) or a mini-batch (batch)



# SGD algorithms

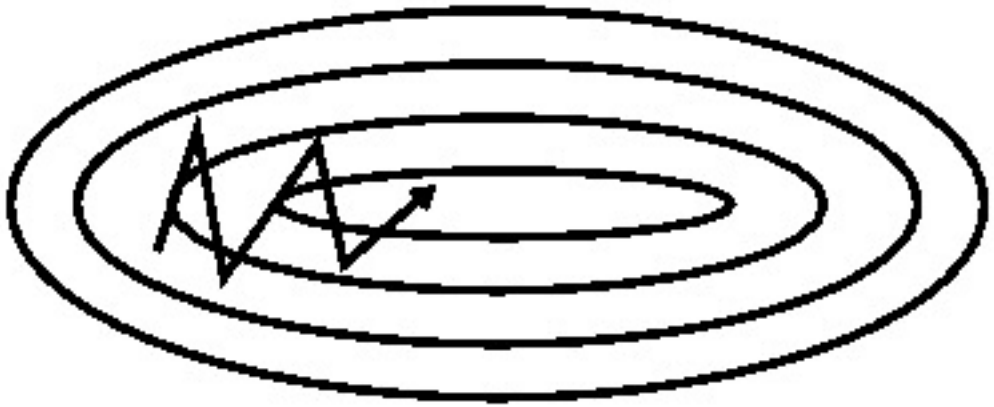
Vanilla SGD

Momentum SGD

Annealing SGD (step, exponential or 1/t decay):

```
x += - learning_rate * dx
```

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```



http://danielnouri.org/notes/category/deep-learning/

# Some adaptive SGD

**Adagrad**: adapts updates to each parameter depending on their importance..

squared gradients accumulate in the denominator → learning rate quickly drops

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

**RMSprop**: divides learning rate by an exponentially decaying average of squared gradients.

```
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

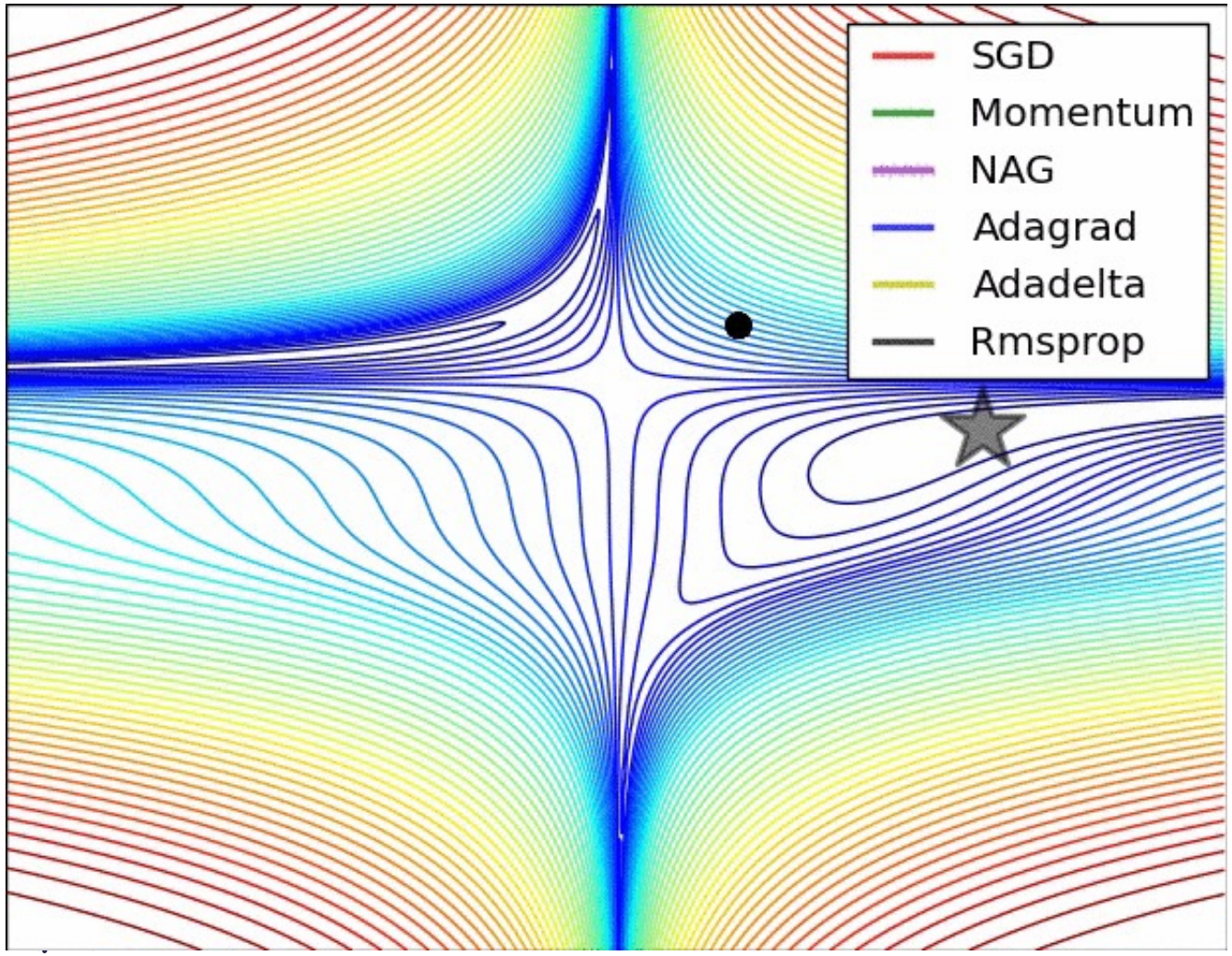
**Adam**: computes adaptive learning rates for each parameter. Similar to RMSprop also keeps an exponentially decaying average of past gradients mt, similar to momentum

```
m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)
x += - learning_rate * m / (np.sqrt(v) + eps)
```

CERN openlab

# Adaptive SGD methods

*comparison*

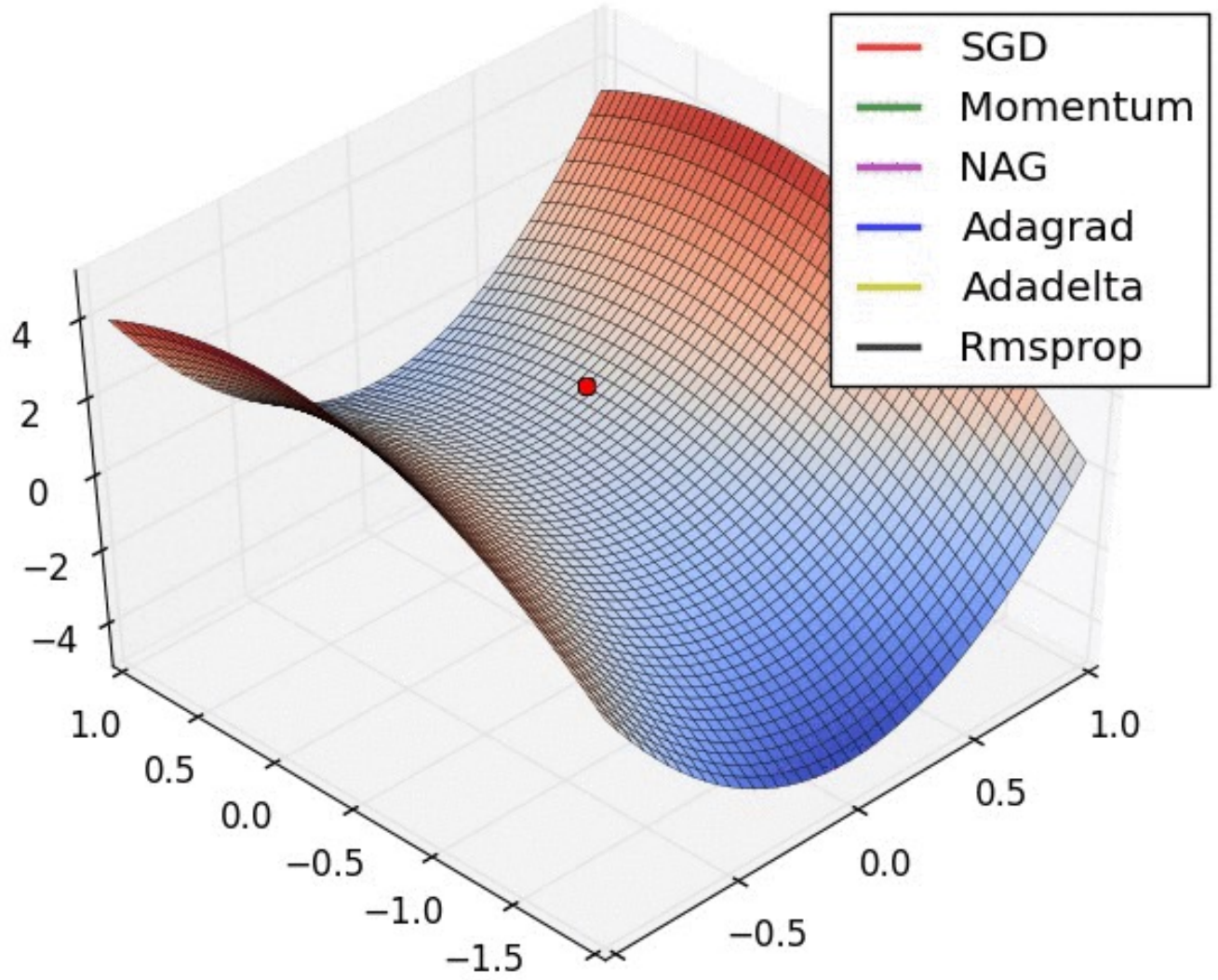



time evolution of different optimization algorithms.

"overshooting" behavior of momentum-based methods

# Adaptive SGD methods

*comparison*




A saddle point:

SGD has a very hard time breaking symmetry.

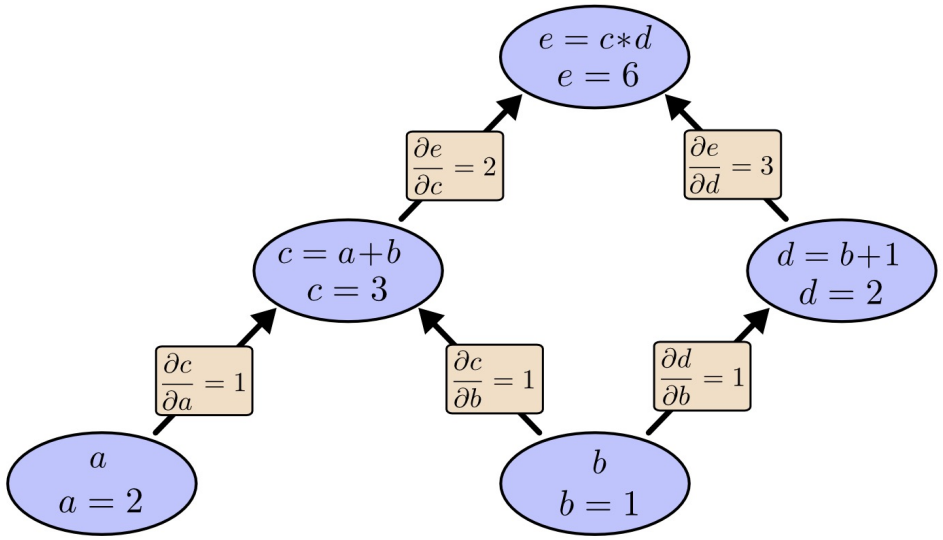
Adaptive algorithms, see very low gradients in the saddle direction. the denominator increase the effective learning rate along this direction,

Learning rate is one of the major hyper-parameters regulating training

# Back-propagation: the problem

Taking into account that even easy classification examples might involve hundreds of weights, complicated activations and losses

How do we actually implement training in a computationally efficient way for a NN??




Given a function **e** and a set of variables (**a** & **b**) :

How does a change in **a** affect **e**?

To calculate it we **follow the path through each of the connecting branches**

What if the number of layers increases

**Combinatorics explodes**!

http://colah.github.io/posts/2015-08-Backprop/

http://colah.github.io/posts/2015-08-Backprop/

# Back-propagation

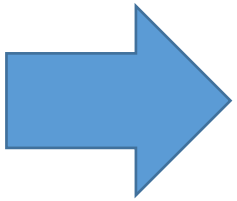
*"the difference between a model taking a week to train and taking 200,000 years"*

- the key algorithm that makes training deep models computationally tractable.
- it's a technique for calculating derivatives quickly
- It simply relies on the **derivatives chain rule**

$$u_1 = f(v)$$

$$u_2 = g(v)$$

$$t = h(u_1, u_2)$$



$$\frac{dt}{dv} = \sum_i \frac{dt}{du_i} \frac{du_i}{dv}$$

# Back-propagation

*A simple visual example*

$$f(x, y, z) = (x + y)z$$

$$q = x + y$$

$$f = qz$$

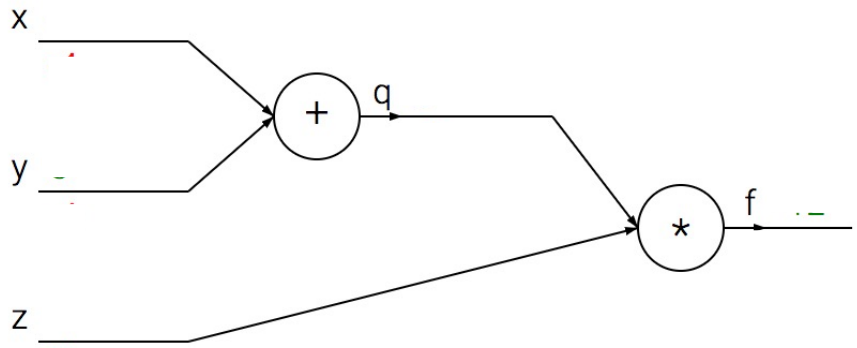
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\Rightarrow \quad \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

**Forward pass** computes values from inputs to output

X = -2, Y = 5, Z = -4




**How does a change in Y affect f?**

Calculate (forward) derivatives

OR

use **backward derivatives**: recursively applie the chain rule backward

# Back-propagation

*A visual example*

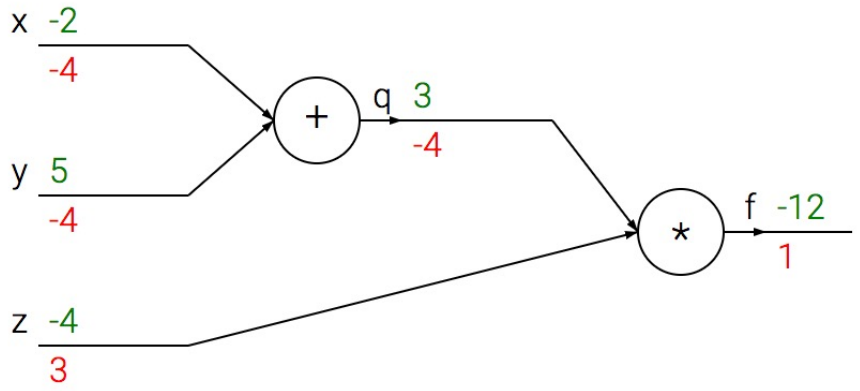
**Backward derivatives** approach is much more efficient in the case of large graphs

Because of the chain rules, at each step the derivative depends only

- On the derivatives already calculated for the parent nodes
- On the node values calculated during the forward pass

Gradients flow “backward” through the graph

Backpropagation is a special case of the **automatic differentiation** programming abstraction applied to neural networks



# Ex: Back-propagation example

*Example circuit for a 2D neuron with a sigmoid activation function*

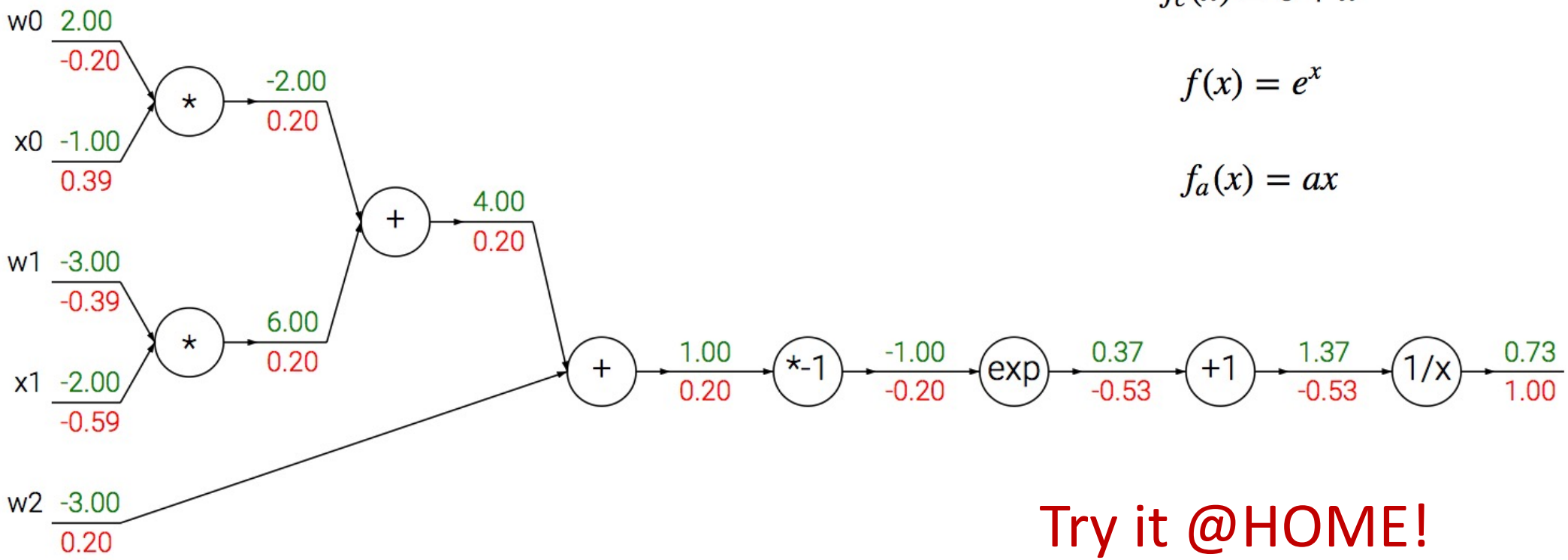
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Decompose using : $f(x) = \frac{1}{x}$

$f_c(x) = c + x$

$f(x) = e^x$

$f_a(x) = ax$




Try it @HOME!

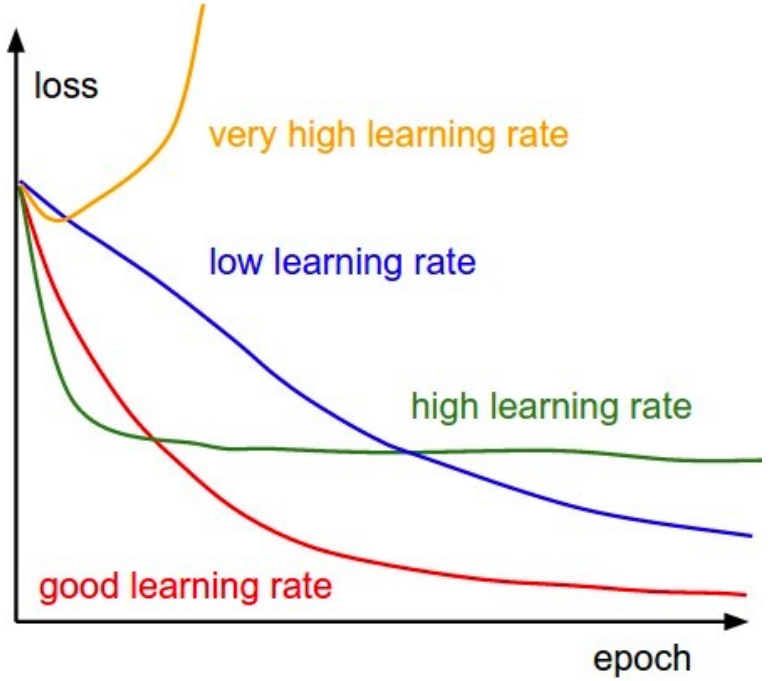
http://cs231n.github.io

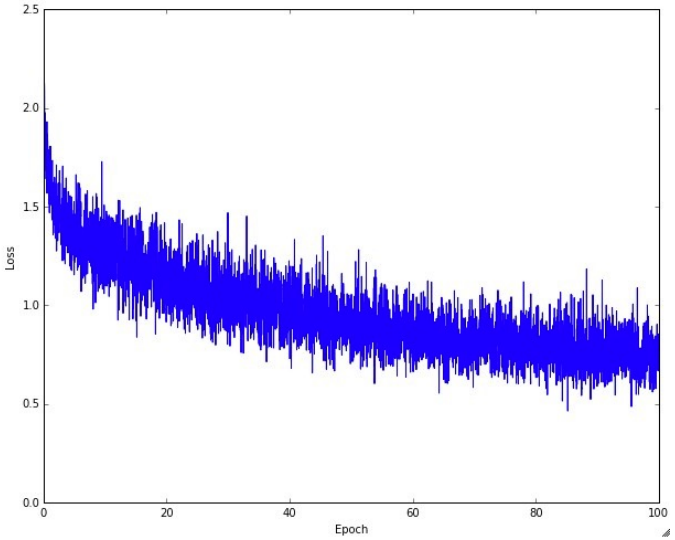
# Evaluating the learning process: loss check

High learning rate will quickly decay the loss faster

- Can  get stuck at worse values because the parameters bounce around chaotically

Low learning rate induces very low convergence speed



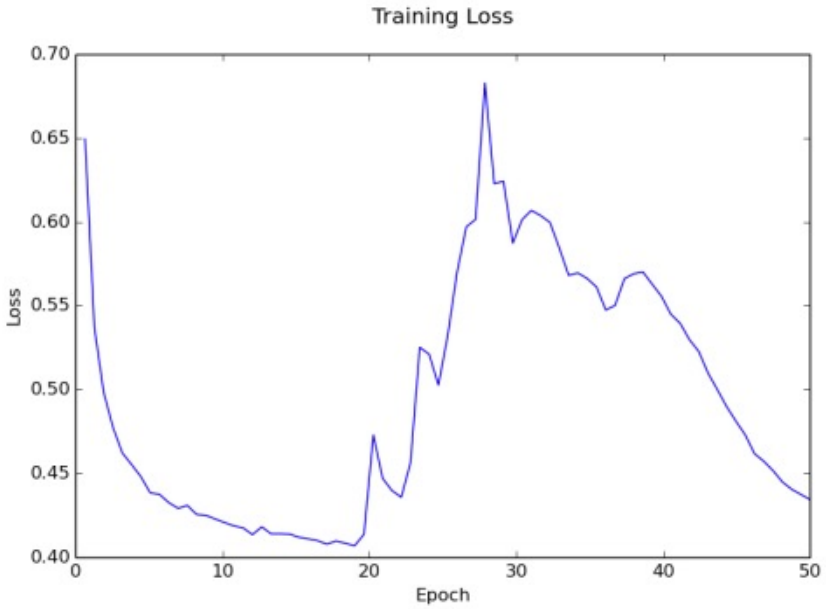




Typical loss function over time

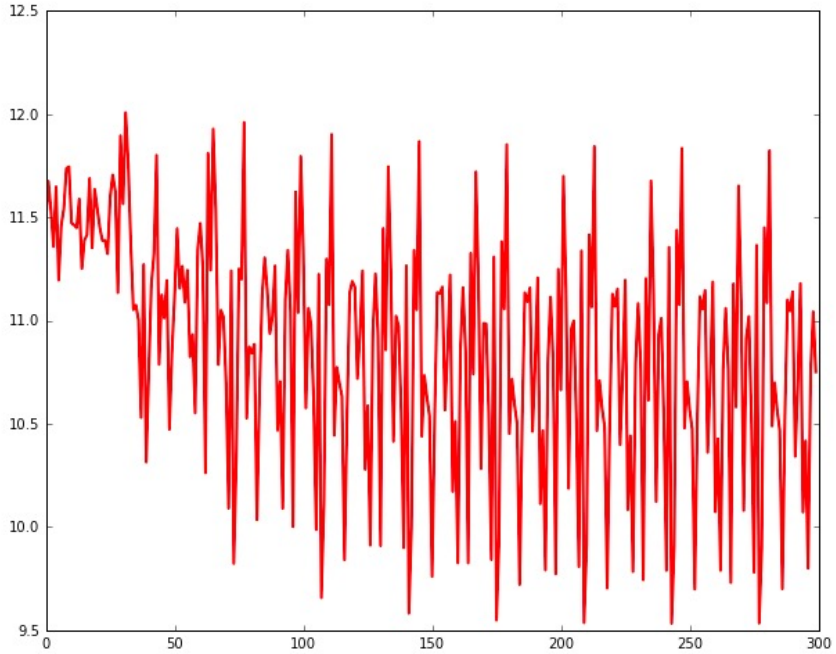
- A slightly too small learning rate ?
- Too low batch size ?

# “Art” pieces

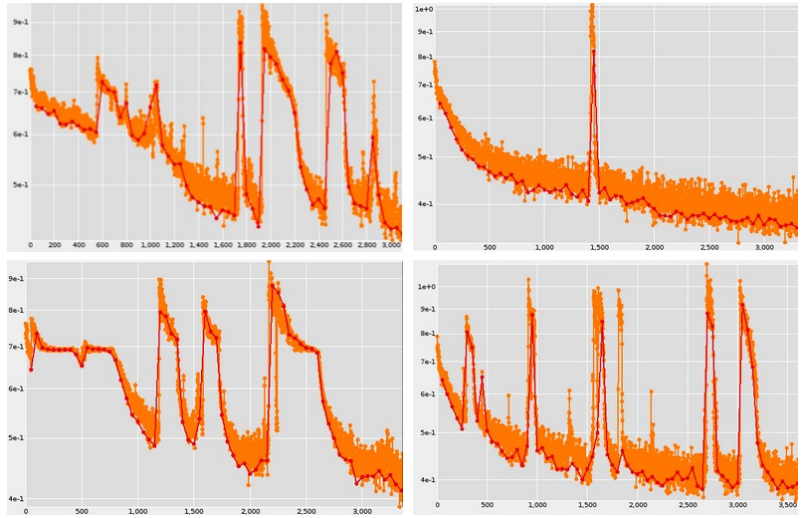
https://lossfunctions.tumblr.com




"This RNN smoothly forgets everything it has learned. God knows what happened… »

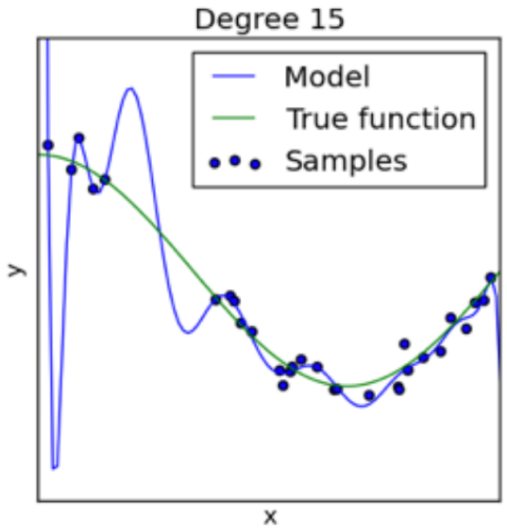



“A heart rate or a loss function? :)”



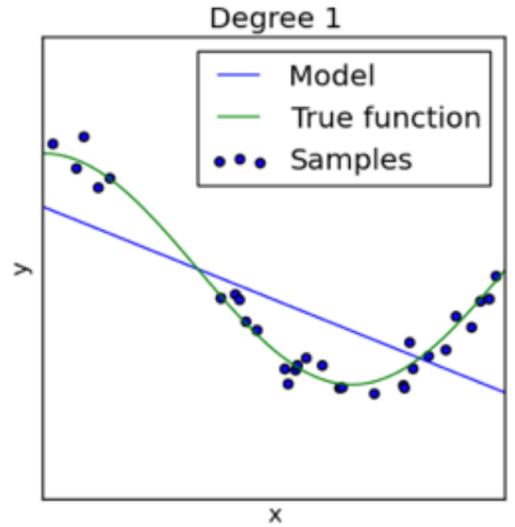
“Taming Spatial Transformer Networks, contributed by Diogo. For the record, it’s not supposed to look like that”

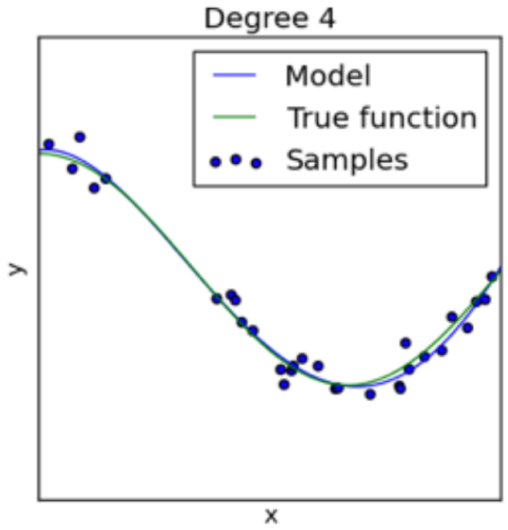
# Overfitting




**Overfitting:**

A model with high capacity fits the noise in the data instead of the underlying relationship




Simple models may **underfit**: will deviate from data but will not be influenced by its peculiarities

Correct identification of outliers (noise) leads to better generalization

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

http://scikit-learn.org/

# Overfitting

validation error curve shows very small validation accuracy compared to training

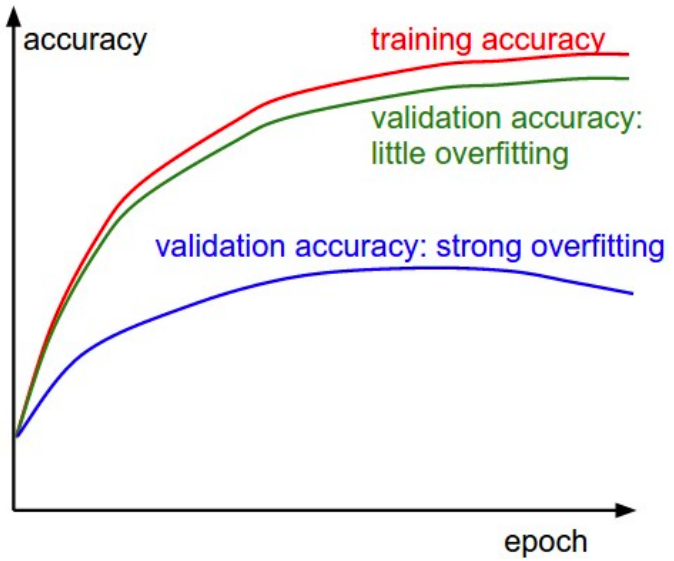
indicating strong overfitting

→ increase regularization (stronger L2 weight penalty, more dropout, etc.) or collect more data.

validation accuracy tracks the training accuracy fairly well.

model capacity is not high enough: make the

model larger by increasing the number of parameters



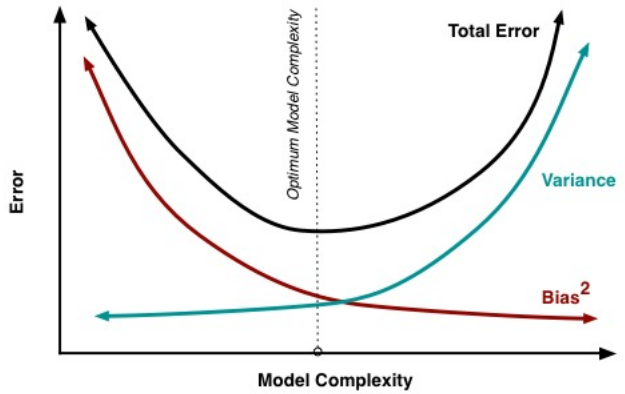

Gap between training and validation accuracy indicates the amount of overfitting

# Optimising hyper-parameters

Noise is intrinsic to system or measurements

Can not be avoided or improved with modeling

Lower bound on possible noise




## Bias Variance Tradeoff



- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

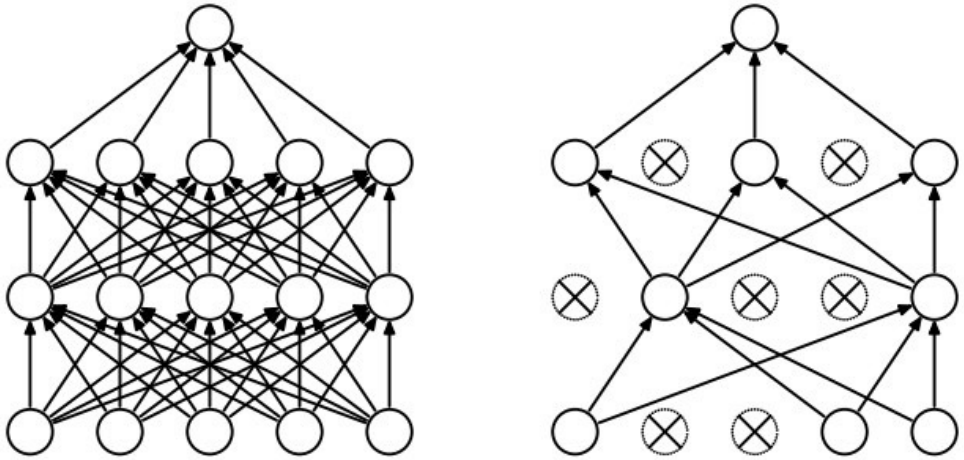
$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$
$$= \text{noise} + (\text{bias})^2 + \text{variance}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.
- **More Complexity** will make the model "move" more to capture the data points, and hence its **variance will be larger**.
  - **As dataset size grows, can reduce variance! Can use more complex model**

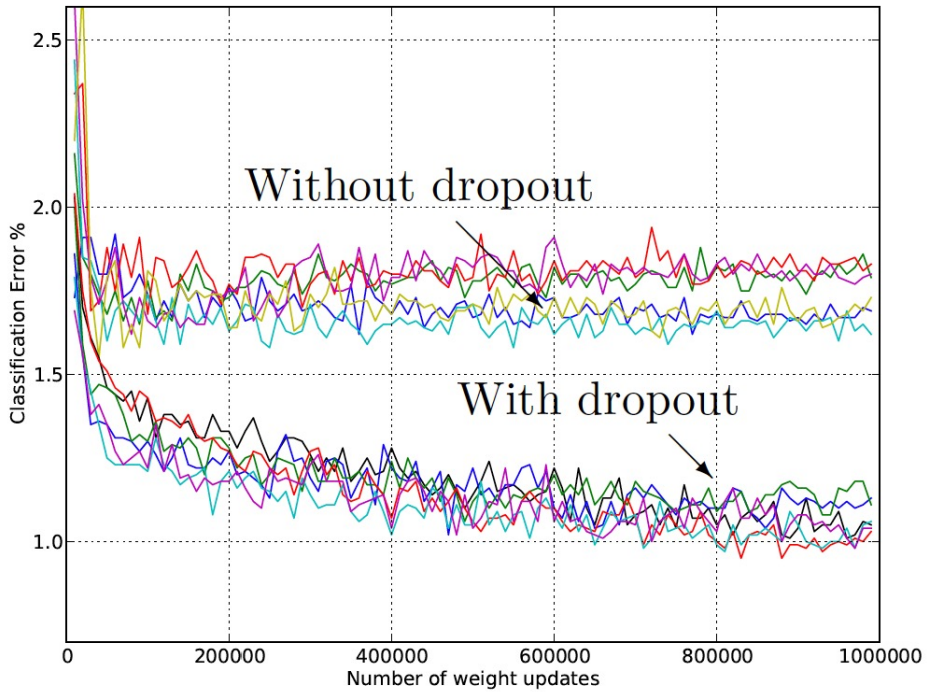
Slide from M. Kagan

# Regularisation

- L2 or L1 regularisation
  - Introduce directly in the objective function terms penalising large weights)
  - use all inputs with small contributions instead of just a few ones with larger weights
- Cap the maximum value
- Dropout



(a) Standard Neural Net (b) After applying dropout.




(http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf)

# Beyond Neural Networks

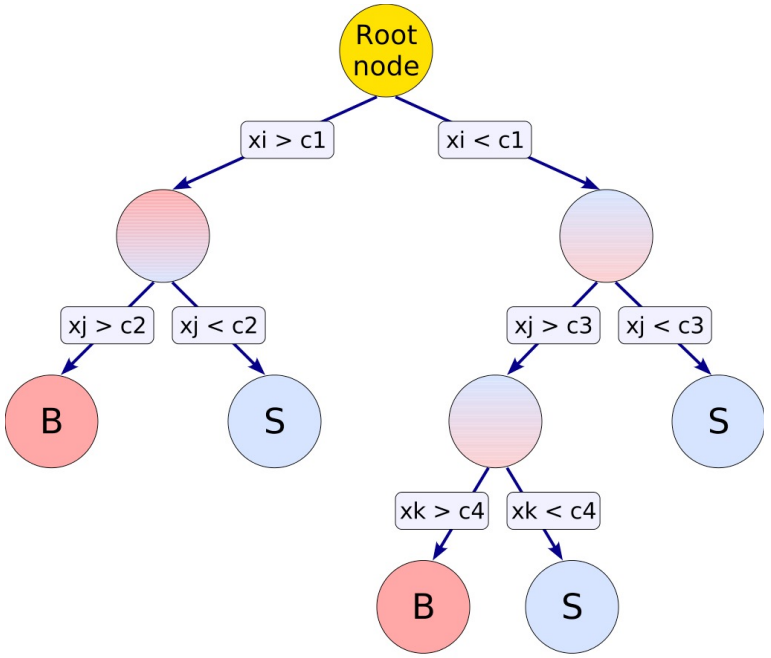
Decision Trees

Kernel Methods



# Decision Tree

L. Breiman et al., “Classification and Regression Trees” (1984)




**Consecutive set of questions** (nodes)

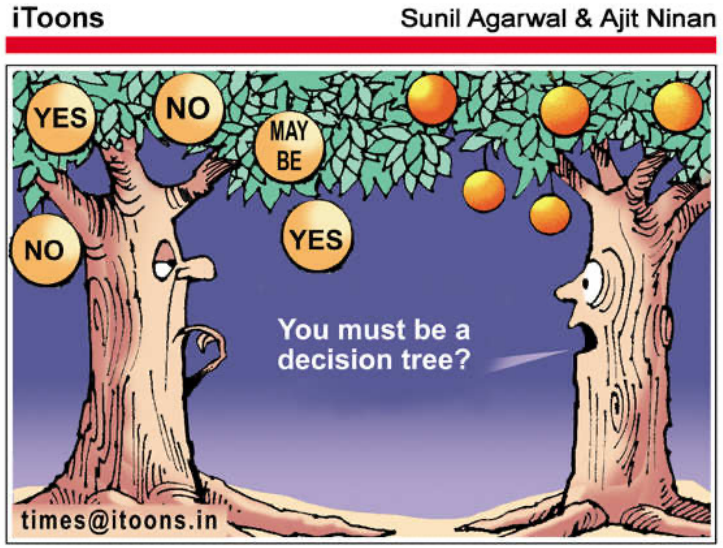
- Only two possible answers per question
- Each question depends on the formerly given answers
- Final verdict (leaf) is reached after a given maximum number of nodes

**Advantages**

- Easy to understand/interpret
- Good with multivariate data
- Fast training

**Disadvantages**

- Single tree not very strong → Random Forest




CERN openlab

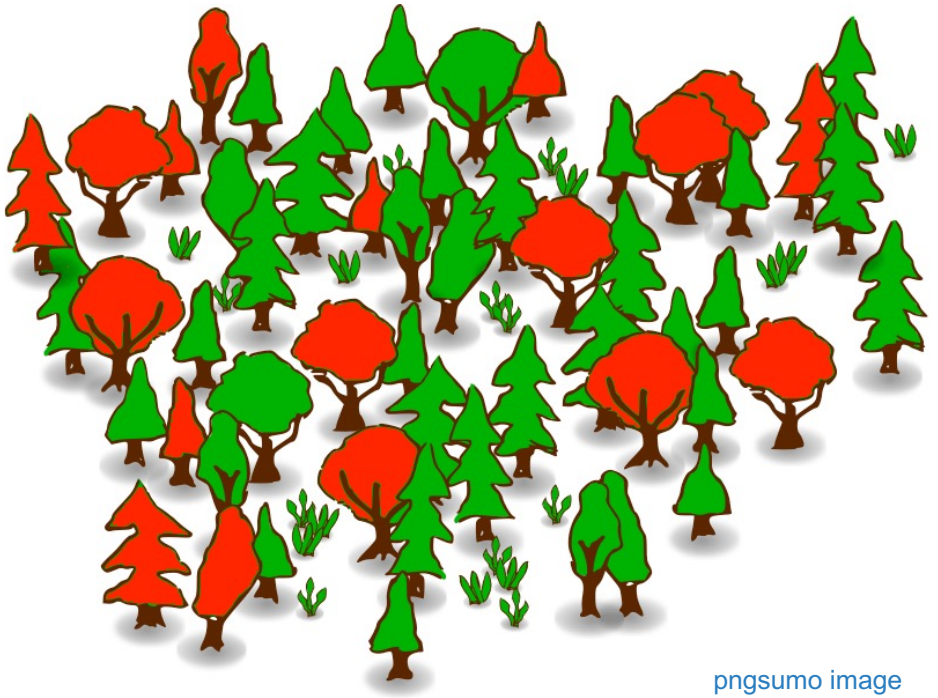
# Random Forests

**Ensemble** of trees, **based on majority vote**:

- ensemble of **uncorrelated trees** is better than only one tree even though their separation power is the same

Available methods to train Random Forests:

- Bagging
  - A subset of events is drawn from the training data with replacement
  - Tree is trained on this subset, this is repeated many times
- Boosting
  - Misclassified events are weighted higher so that future learners
  - concentrate on these
  - Most commonly used method AdaBoost

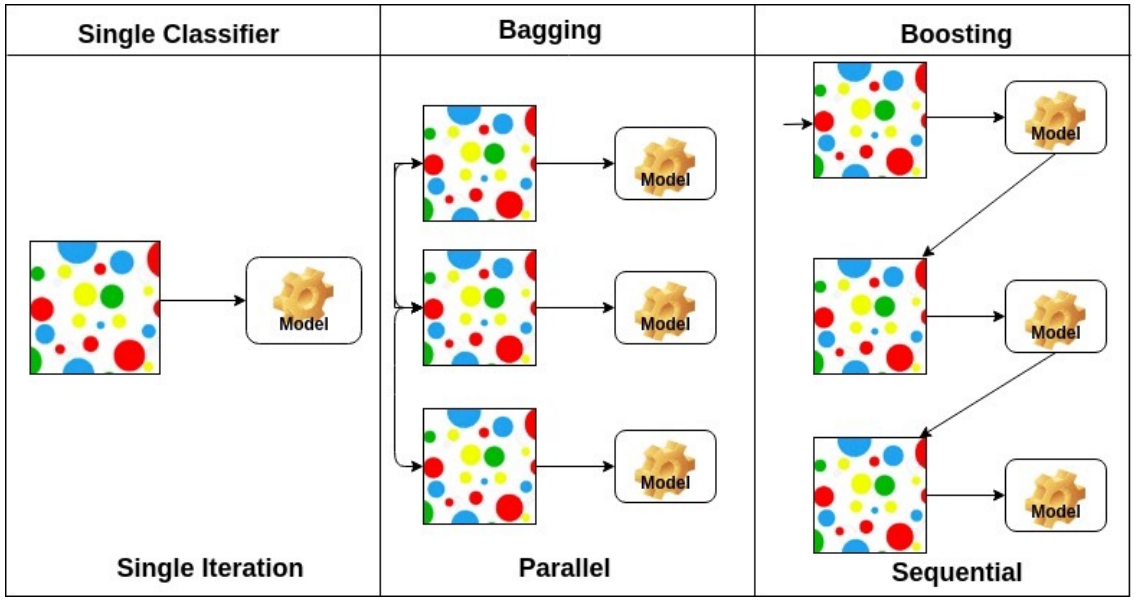


pngsumo image

# Boosting

**AdaBoost (Adaptive Boosting):**

- Enhances weights of misclassified events and reduces weight of correctly classified ones after each training so that future trees learn those better
- Iterates until weight of misclassified > 50%
- Final weight is the sum of all classifiers weighted by their errors




**Alternative algorithms: Gradient Tree Boosting, XGBoost**

# Support Vector Machines

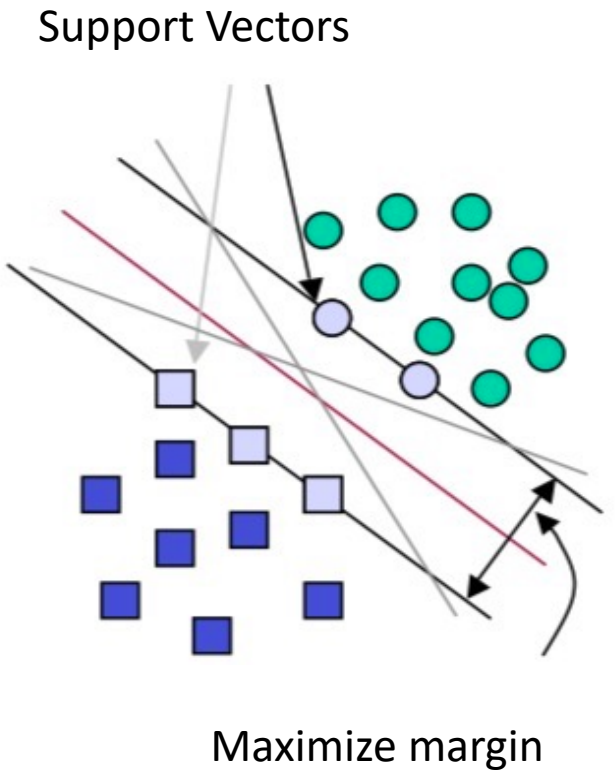
*Example of a kernel based method*

Separate two sets of points with the **widest possible margin**

Decision function is fully specified by a support vectors (subset of training samples)

- Input: set of training pair samples with a result function $y(x_i) \in [-1,1]$;
- Output: set of $w_i$ whose linear combination predicts the value of $y(x_i)$

Optimising a SVM is a convex optimization problem (global minimum)



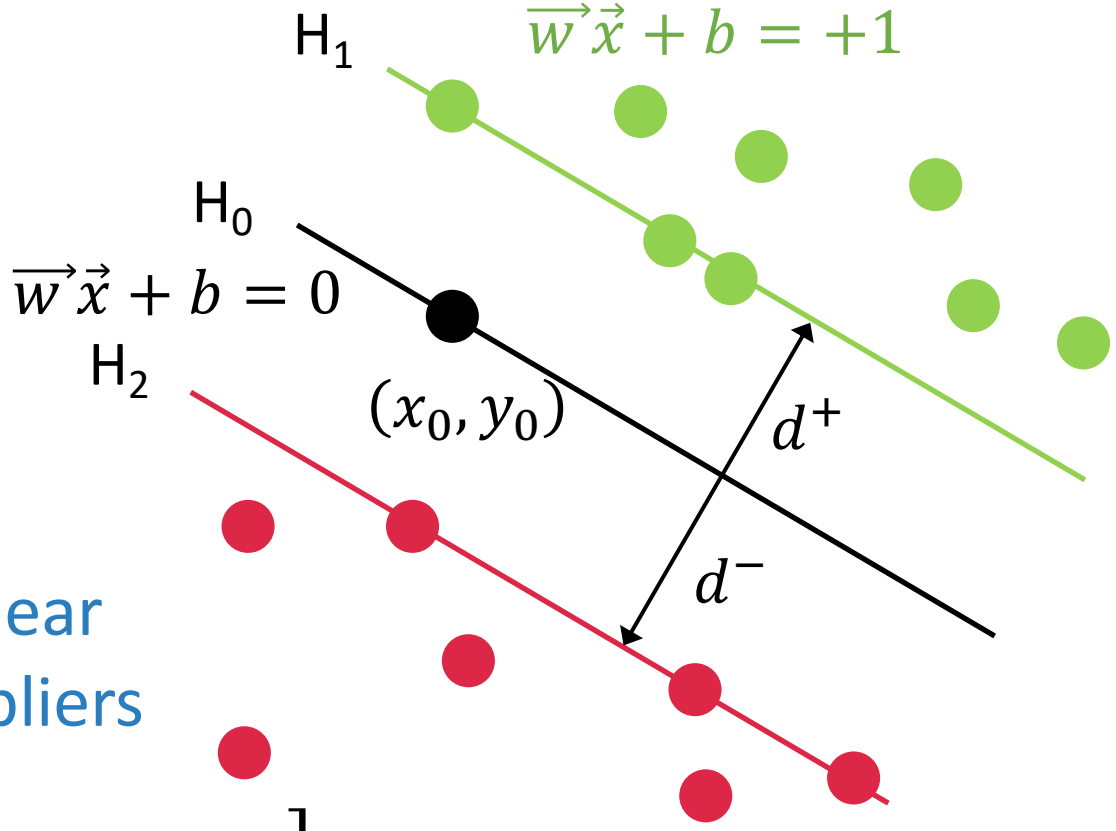
# One word on how SVM works

Distance from support point to centerline

$$d = |\vec{w}\vec{x} + b|/|\vec{w}| = 1/|\vec{w}|$$

minimize $|\vec{w}|$ and
impose no points "in between"

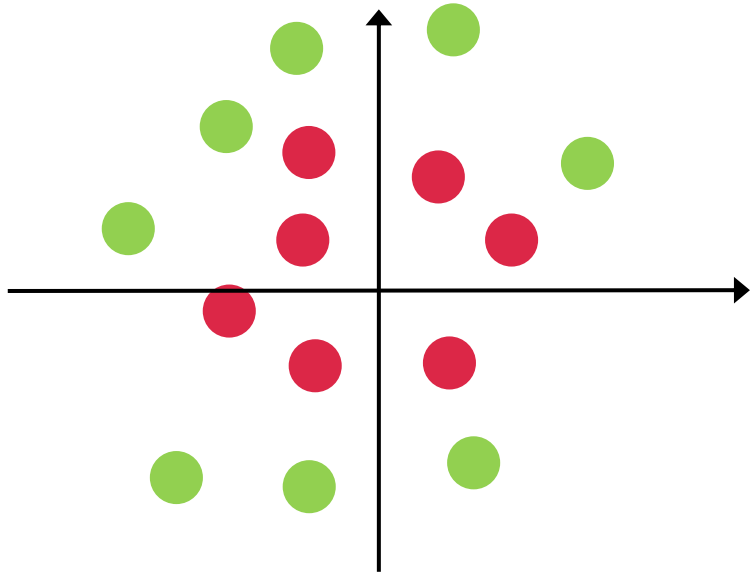
$$y_i(\vec{w}\vec{x}_i + b) \geq 1$$



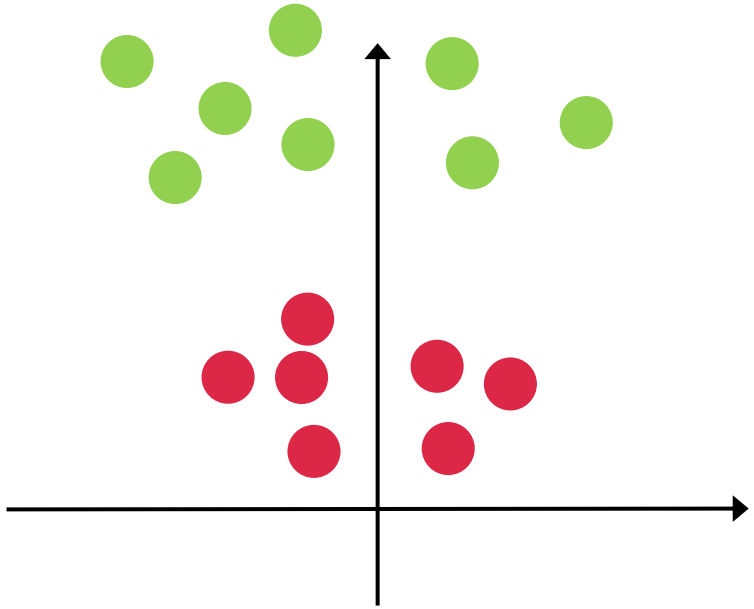

quadratic minimization problem with linear constraint solved with Lagrangian multipliers

$$\min_{\vec{w},b} \mathcal{L}(\vec{x}, \vec{a}) = \min_{\vec{w},b} \left[ 1/2|\vec{w}| + \sum_i a_i(y_i(\vec{w}\vec{x}_i + b) - 1) \right]$$

# Non-linearity



$$x_i' = x_i$$
$$y_i' = \sqrt{x_i^2 + y_i^2}$$



Introducing kernels

We do not need $\overrightarrow{x'} = \Phi(\vec{x})$ but just $\mathrm{K}\left(\overrightarrow{x_i}, \overrightarrow{x_j}\right) = \Phi(\overrightarrow{x_i}) \cdot \Phi\left(\overrightarrow{x_j}\right)$!

# Reinforcement Learning for accelerator control

Online training of an agent on 160MeV **LINAC4 trajectory** steering in horizontal plane.

- LINAC4 parameters are usually trained manually !

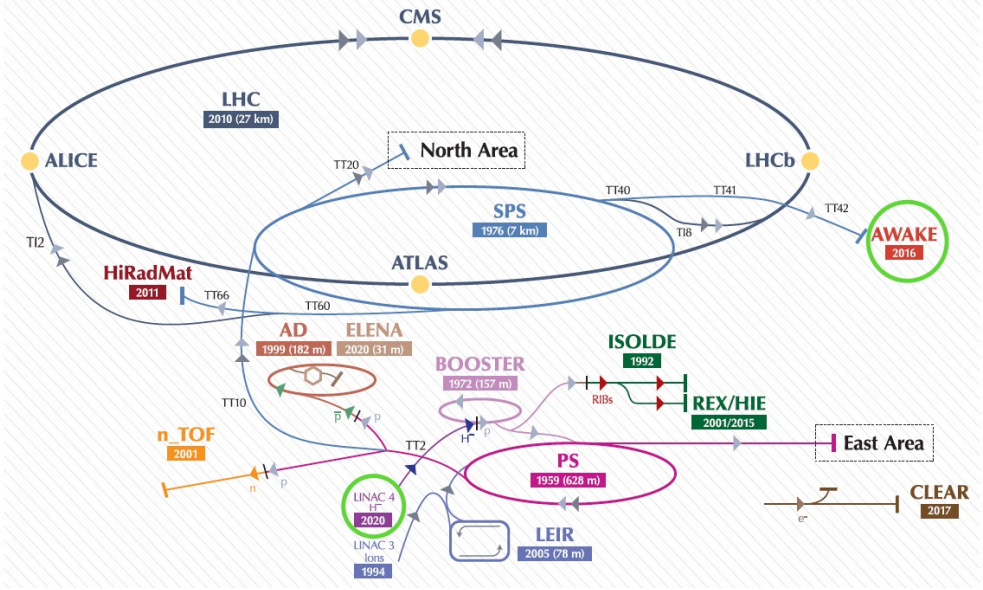
17 states, 16 possible actions

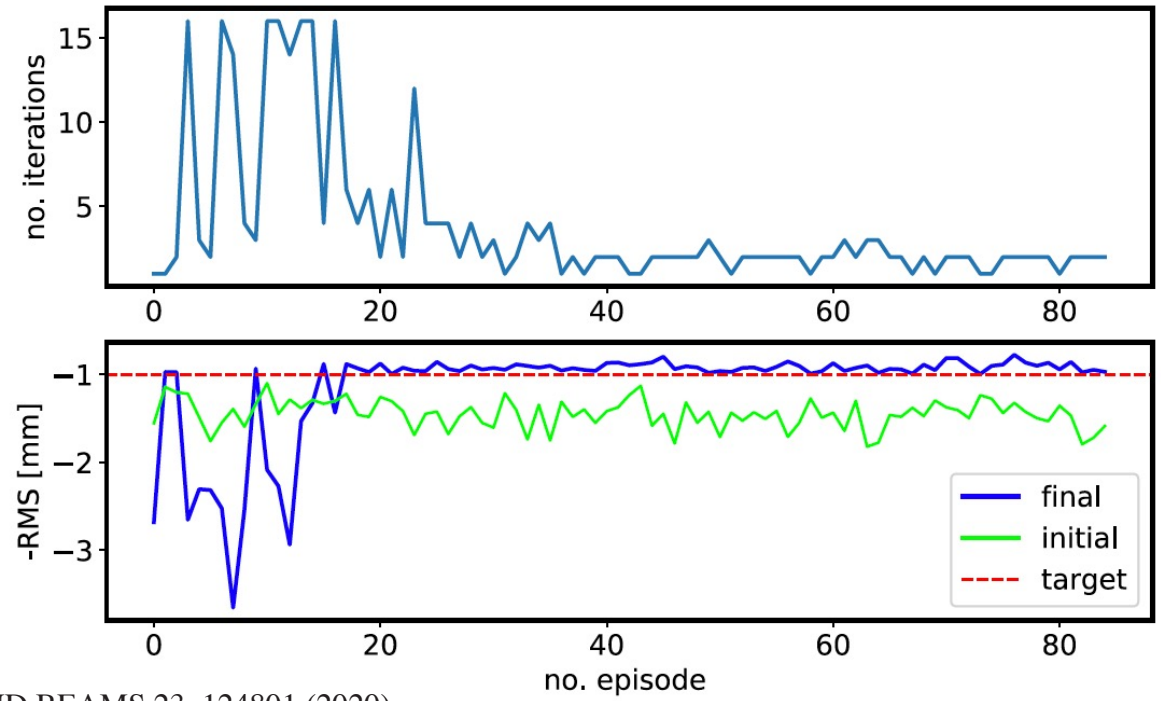
Agent: 2 fully-connected layers (32 nodes)

Online hyperparameter tuning

- Maximum allowable RMS is limited to 3 mm to protect the machine.
- The target set to reach 1 mm RMS max.

**Achieved in less than 5 iterations**





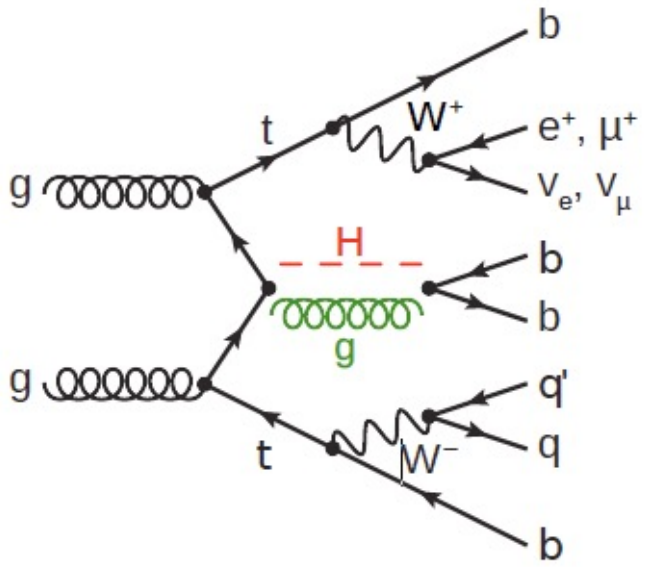

V. Kain, PHYSICAL REVIEW ACCELERATORS AND BEAMS 23, 124801 (2020)

C. Reissel, ML4Jets 2021

V. Belis, arxiv: 2104.07692

# Classification in High Energy Physics

## *Higgs classification*




Irreducible tt+bb background with final state being indistinguishable from signal

Background simulation comes with large theory modelling uncertainties

large number of jets in final state:

assignment of jets to partons not trivial

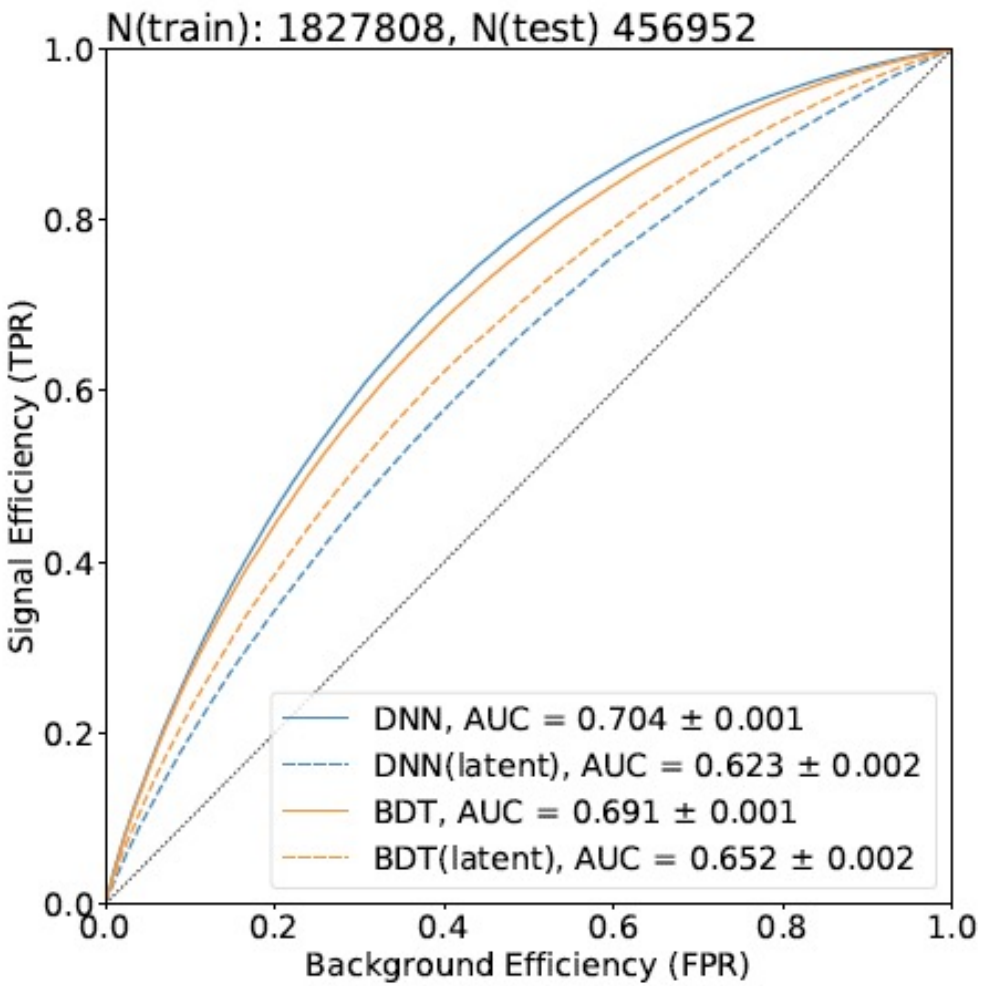
→ combinatorial background

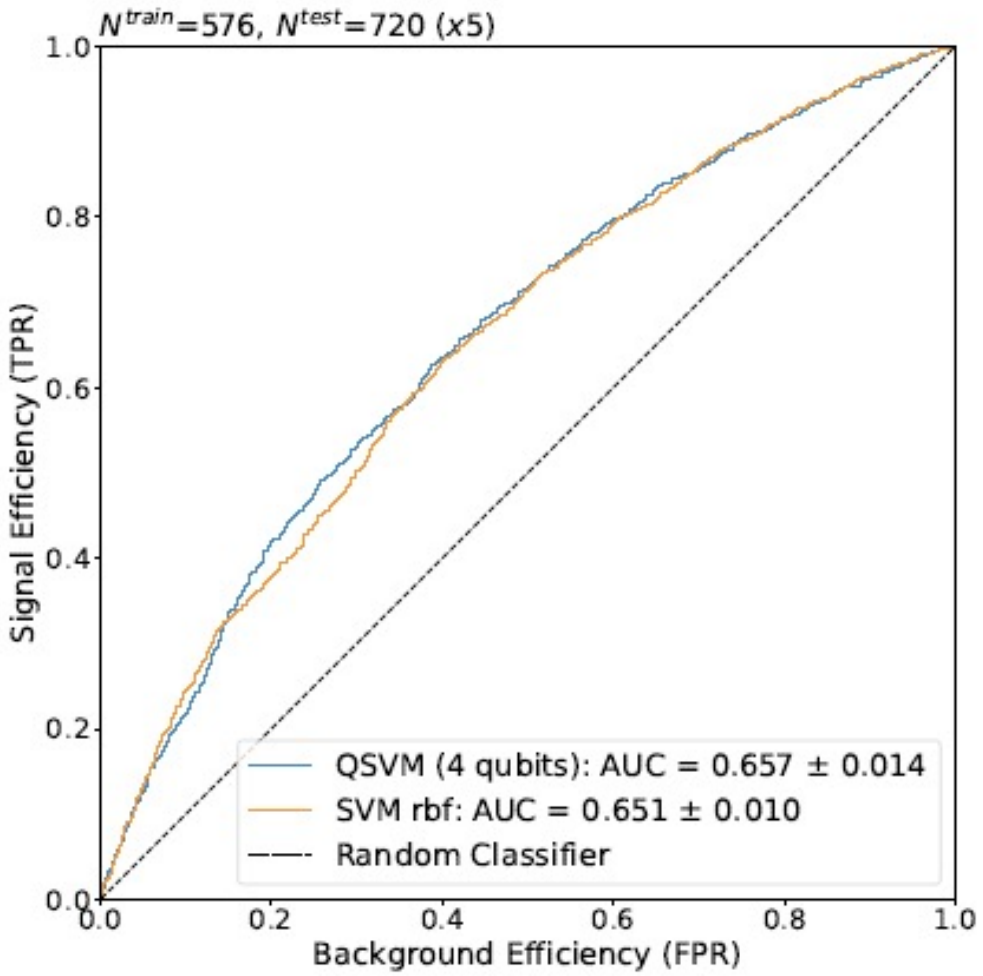
information lost: significant amount of events cannot be fully reconstructed because jets are out of acceptance

V. Belis, arxiv: 2104.07692

# Simplified analysis

*BDT vs DNN vs SVM (and QSVM…)*



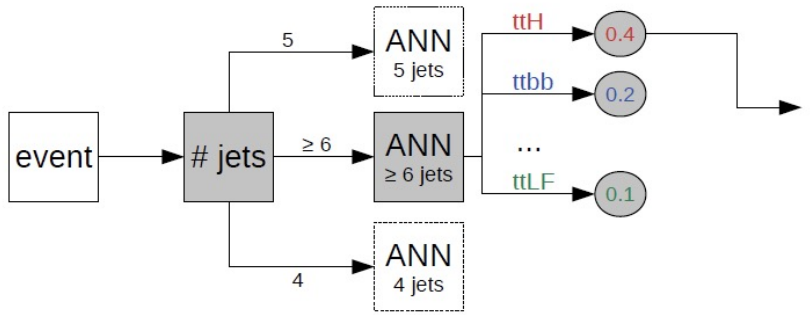
# Higgs Classification: ANN in CMS

Slide from C. Reissel

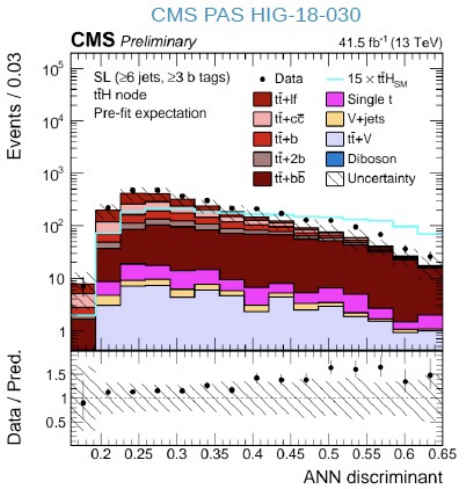
ANN discriminant used for event classification and final selection in the fit based on high-level variables

## Current Analysis Strategy (CMS)

- events are classified according to number of jets, number of b tagged jets and output node of ANN (artificial neural network)
- inputs to ANN: kinematics of leptons, jets and missing transverse energy, but also high-level variables (e.g. event shape or Matrix-Element Method discriminant)
- ANN discriminant output used also as final discriminant in fit
- evidence for $t\bar{t}H(b\bar{b})$ found (3.9σ), aiming for discovery (5σ)




CMS PAS HIG-18-030




ETH zürich Christina Reissel 19

CERN openlab

CMS PAS HIG-18-030

# Higgs Classification: BDT in ATLAS

Slide from C. Reissel

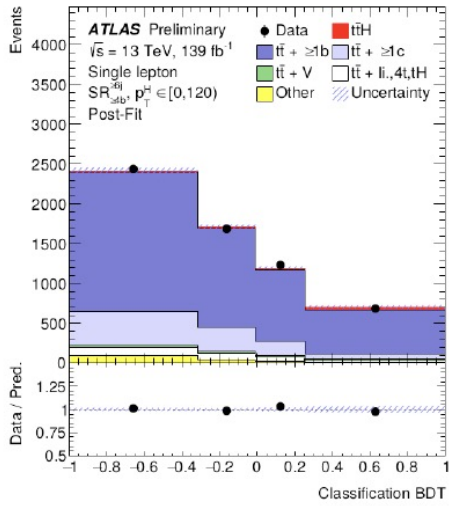
Two BDT – based classification steps

High-level variables are used as input

## Current Analysis Strategy (ATLAS)

- events are classified according to number of leptons, jets, b tagged jets and boosted Higgs boson candidates
- two type of MVA based classifiers:
  - reconstruction BDTs:
    matching reconstructed jets to partons
    → identification of Higgs boson candidates by chosing combination of jets with highest BDT output
  - classification BDTs
  - observed (expected) significance for $t\bar{t}H(b\bar{b})$: 1.3(3.0)σ

ATLAS-CONF-2020-058





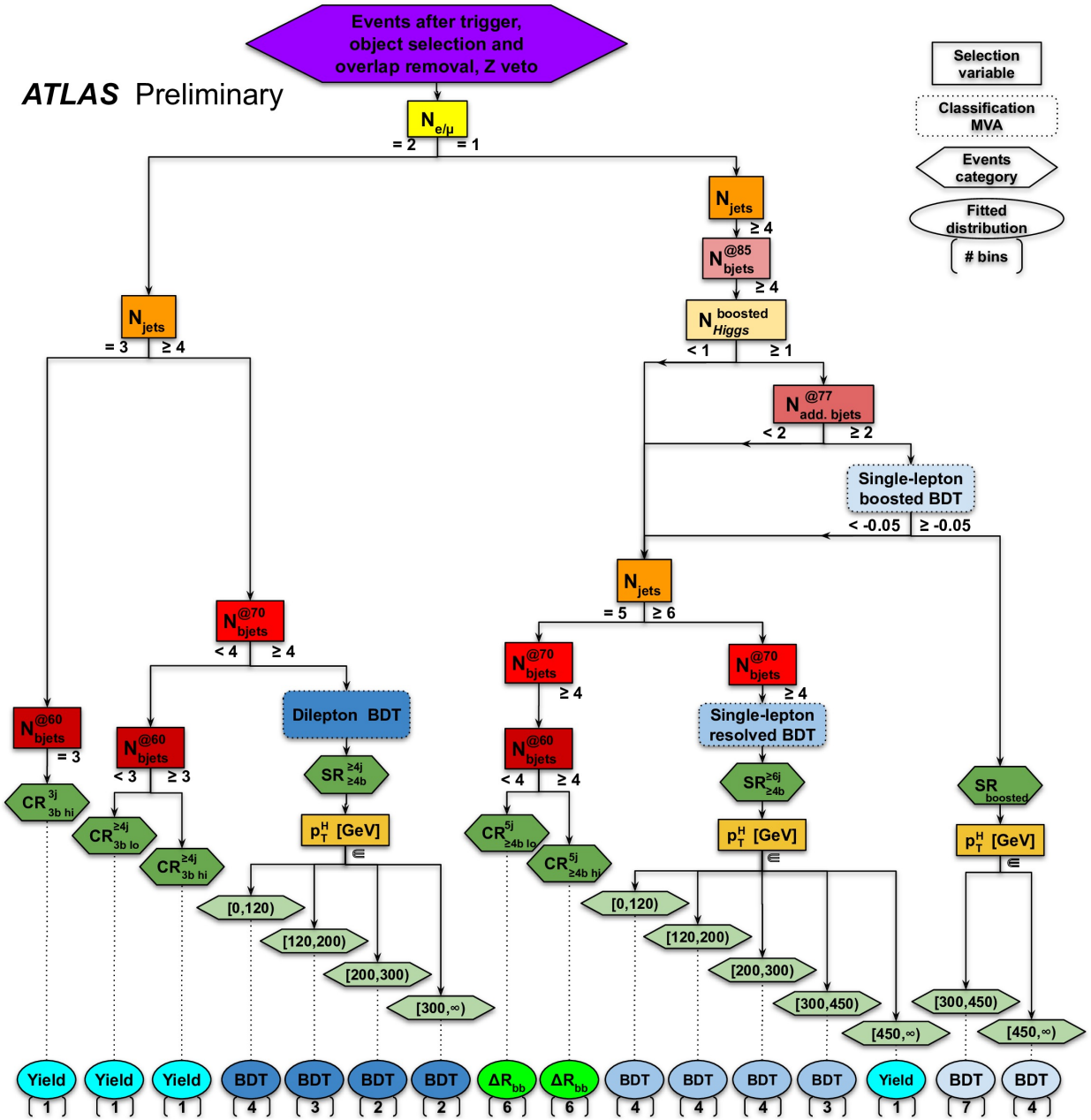

CMS PAS HIG-18-030

# Machine Learning -based analysis

"**Conventional machine-learning** techniques were **limited** in their ability to **process natural data** in their raw form.

For decades, constructing (…) a machine-learning system required **careful engineering and considerable domain expertise** to design a feature extractor … "

LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521,** 436–444 (2015).




ATLAS-CONF-2020-058 analysis flowchart

# Summary

Machine learning uses **mathematical** and **statistical models learned** from **data** to characterize patterns and relations between inputs, and use this for inference / prediction

a powerful tool for many different applications

Choosing a model for a given problem is difficult so is optimizing the training process.

Always plot the losses!

Classical Machine Learning has limitations

Deep Learning



For tomorrow:

**Think about the smartest person you know.**

**Why do you think she/he is smart ?**



# Thanks!

*Questions?*

*Sofia.Vallecorsa@cern.ch*



# References


- M. Kagan, Introduction to Machine Learning, CERN openlab summer student lectures
- G. Louppe, Deep Learning, https://github.com/glouppe/info8010-deep-learning
- François Fleuret, Deep Learning Course at UniGE: https://fleuret.org/dlc/
- http://cs231n.github.io/
- Pattern Recognition and Machine Learning, Bishop (2006)
- Elements of Statistical Learning (2nd Ed.) Hastie, Tibshirani & Friedman 2009
- Introduction to machine learning, Murray: http://videolectures.net/bootcamp2010_murray_iml/
- CS181, Parkes and Rush, Harvard University: http://cs181.fas.harvard.edu
- CS229, Ng, Stanford University: http://cs229.stanford.edu/
- http://scs.ryerson.ca/~aharley/vis
- http://cs. stanford.edu/people/karpathy/convnetjs/
- http://scikit-learn.org/