Machine learning

Yann Coadou

CPPM Marseille

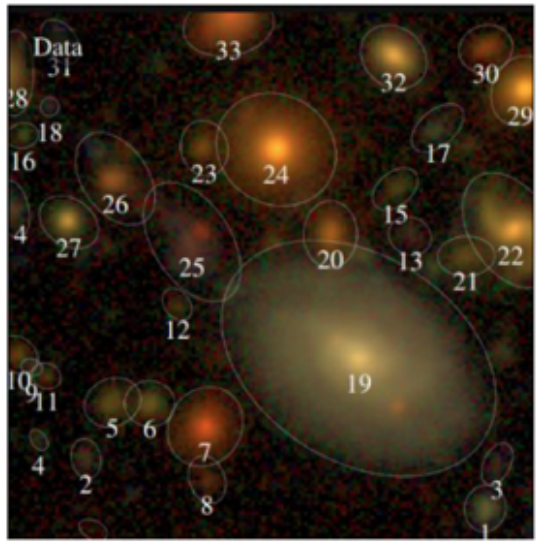
Online edition, 26 January 2022

Outline

Introduction

Typical problems

- Classification of objects
  - separate real and fake leptons/jets/etc.
- Signal enhancement relative to background
  - e.g. galaxy deblending
- Regression: best estimation of a parameter
  - lepton energy, $E_{\mathrm{T}}^{\mathrm{miss}}$ value, invariant mass, proportions of radioelements, etc.



Discrimination of signal from background in HEP

- Event level (Higgs searches, . . .)
- Cone level (tau-vs-jet reconstruction, . . .)
- Lifetime and flavour tagging (b-tagging, . . .)
- Track level (particle identification, . . .)
- Cell level (energy deposit from hard scatter/pileup/noise, . . .)

Introduction

Input information from various sources

- Kinematic variables (masses, momenta, decay angles, ...)
- Event properties (jet multiplicity, sum of charges, sphericity, ...)
- Detector response (silicon hits, dE/dx, Cherenkov angle, shower profiles, muon hits, ...)
- Telescope images in visible light/infrared/X-ray/etc.

Most data are (highly) multidimensional

- Use dependencies between $x = \{x_1, \cdots, x_n\}$ discriminating variables
- Approximate this n-dimensional space with a function $f(x)$ capturing the essential features
- f is a multivariate discriminant
- For most of these lectures, use binary classification:
  - an object belongs to one class (e.g. signal) if $f(x) > q$, where q is some threshold,
  - and to another class (e.g. background) if $f(x) \leq q$

Machine learning definitions

- "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."
  Tom Mitchell (1997)

- "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed"

Machine learning: (un)supervised learning

Supervised learning

- Labelled training events: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- Use examples' classes to know how good algorithm is doing

Machine learning: (un)supervised learning

Supervised learning

- Labelled training events: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- Use examples' classes to know how good algorithm is doing

Reinforcement learning

- Instead of labels, some sort of reward system (e.g. game score)
- Goal: maximise future payoff by optimising decision policy
- May not even "learn" anything from data, but remembers what triggers reward or punishment: learn policy

Machine learning: (un)supervised learning

Supervised learning

- Labelled training events: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ of (discriminating) feature variables x and class labels y
- Use examples' classes to know how good algorithm is doing

Reinforcement learning

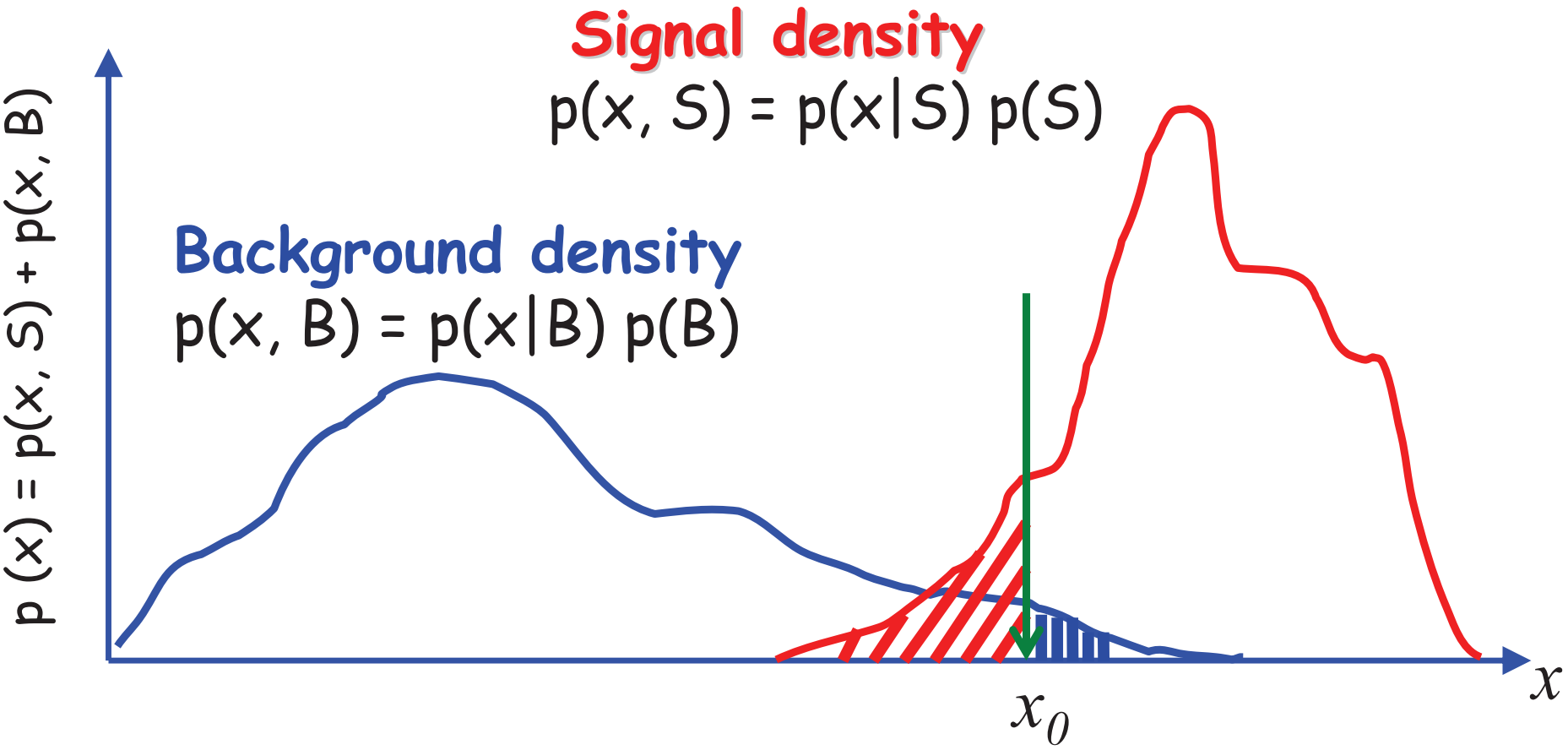
- Instead of labels, some sort of reward system (e.g. game score)
- Goal: maximise future payoff by optimising decision policy
- May not even "learn" anything from data, but remembers what triggers reward or punishment: learn policy

Unsupervised learning

- Find similarities in training sample, without having predefined categories (no labels/targets)
- Discover good internal representation of the input
- Not biased by pre-determined classes $\Rightarrow$ may discover unexpected features!

Optimal discrimination

- Where to place a cut x_0 on variable x?



- Optimal choice: minimum misclassification cost at decision boundary $x = x_0$

Optimal discrimination: cost of misclassification

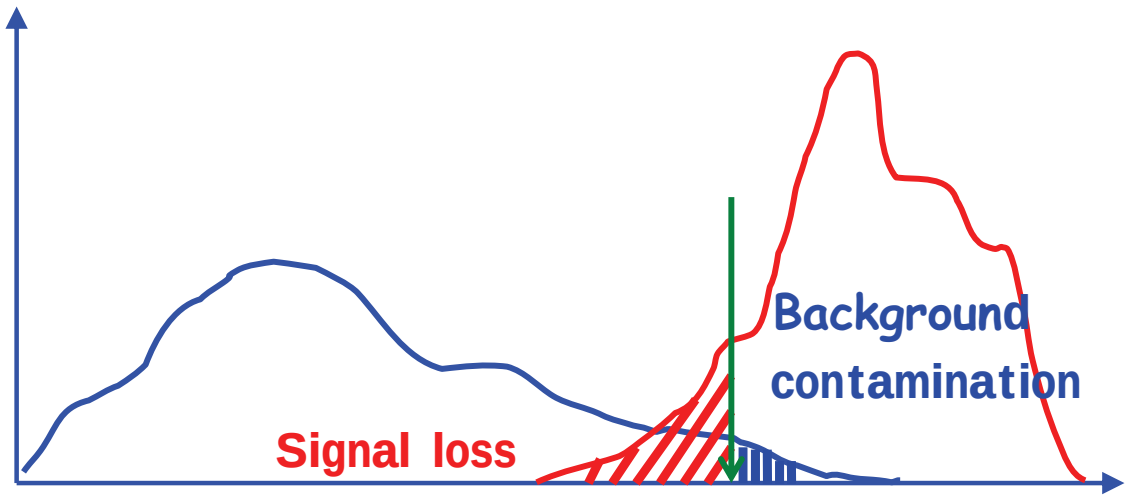
$$C(x_0) = C_S \int H(x_0 - x)p(x, S)dx + C_B \int H(x - x_0)p(x, B)dx$$

signal loss

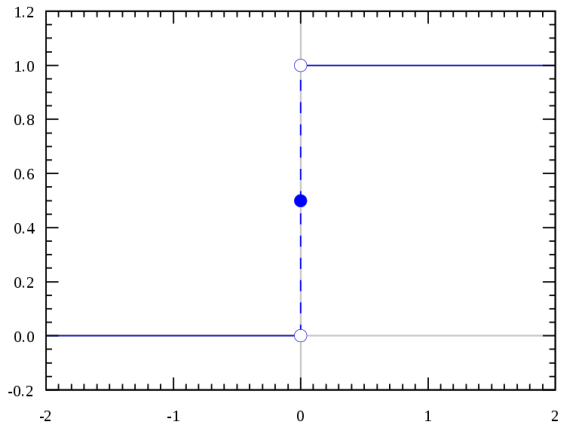
background contamination

C_S = cost of misclassifying signal as background

C_B = cost of misclassifying background as signal



- $H(x)$: Heaviside step function
  - $H(x) = 1$ if $x > 0$, 0 otherwise



- Optimal choice: when cost function C is minimum

Optimal discrimination: Bayes discriminant

Minimising the cost

- Minimise
  $C(x_0) = C_S \int H(x_0 - x)p(x, S)dx + C_B \int H(x - x_0)p(x, B)dx$
  with respect to the boundary x_0:

$$\begin{aligned} 0 &= C_S \int \delta(x_0 - x)p(x, S)dx - C_B \int \delta(x - x_0)p(x, B)dx \\ &= C_S p(x_0, S) - C_B p(x_0, B) \end{aligned}$$

- This gives the Bayes discriminant:

$$BD = \frac{C_B}{C_S} = \frac{p(x_0, S)}{p(x_0, B)} = \frac{p(x_0|S)p(S)}{p(x_0|B)p(B)}$$

Optimal discrimination: Bayes limit

Bayes discriminant

$$BD = \frac{C_B}{C_S} = \frac{p(x|S)}{p(x|B)} \times \frac{p(S)}{p(B)}$$

C_B = background contamination

C_S = signal loss

- From Bayes theorem ($p(A|B)p(B) = p(B|A)p(A)$) and sum of probabilities ($p(S|x) + p(B|x) = 1$):

$$p(S|x) = \frac{BD}{1 + BD}$$

Bayes limit

- $p(S|x) = BD/(1 + BD)$ is what should be achieved to minimise cost of misclassification, reaching classification with fewest mistakes
- Fixing relative cost of background contamination and signal loss $q = C_B/(C_S + C_B)$, $q = p(S|x)$ defines decision boundary:
  - signal-rich if $p(S|x) \geq q$
  - background-rich if $p(S|x) < q$
- Any function that approximates conditional class probability $p(S|x)$ with negligible error reaches the Bayes limit

Optimal discrimination: using a discriminant

How to construct $p(S|x)$?

- $k = p(S)/p(B)$ typically unknown
- Problem: $p(S|x)$ depends on k!
- Solution: it's not a problem...
- Define a multivariate discriminant:

$$D(x) = \frac{s(x)}{s(x) + b(x)} = \frac{p(x|S)}{p(x|S) + p(x|B)}$$

- Now:

$$p(S|x) = \frac{D(x)}{D(x) + \big(1 - D(x)\big)/k}$$

- Cutting on $D(x)$ is equivalent to cutting on $p(S|x)$, implying a corresponding (unknown) cut on $p(S|x)$

Machine learning: learning from examples

Several types of problems

- Classification/decision:
  - signal or background
  - type Ia supernova or not
  - will pay his/her credit back on time or not
- Regression: estimating a parameter value (energy of a particle, redshift of a supernova, ...) [mostly ignored in these lectures]
- Clustering (cluster analysis):
  - in exploratory data mining, finding features

Our goal

- Teach a machine to learn the discriminant $f(x)$ using examples from a training dataset
- Be careful to not learn too much the properties of the training sample
  - no need to memorise the training sample
  - instead, interested in getting the right answer for new events
    $\Rightarrow$ generalisation ability

Finding the multivariate discriminant $y = f(x)$

- Given our N examples $(x, y)_1, \ldots, (x, y)_N$ we need
  - a function class $\mathbb{F} = \{f(x, w)\}$ (w: parameters of prediction to be found)
  - a constraint $Q(w)$ on $\mathbb{F}$ (regularisation term)
  - a loss or error function $L(y, f)$, encoding what is lost if f is poorly chosen in $\mathbb{F}$ (i.e., $f(x, w)$ far from the desired $y = f(x)$)
- Cannot minimise L directly (would depend on the dataset used), but rather its average over a training sample, the empirical risk:

$$R(w) = \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(x_i, w)\big)$$

subject to constraint $Q(w)$, so we minimise the cost function:

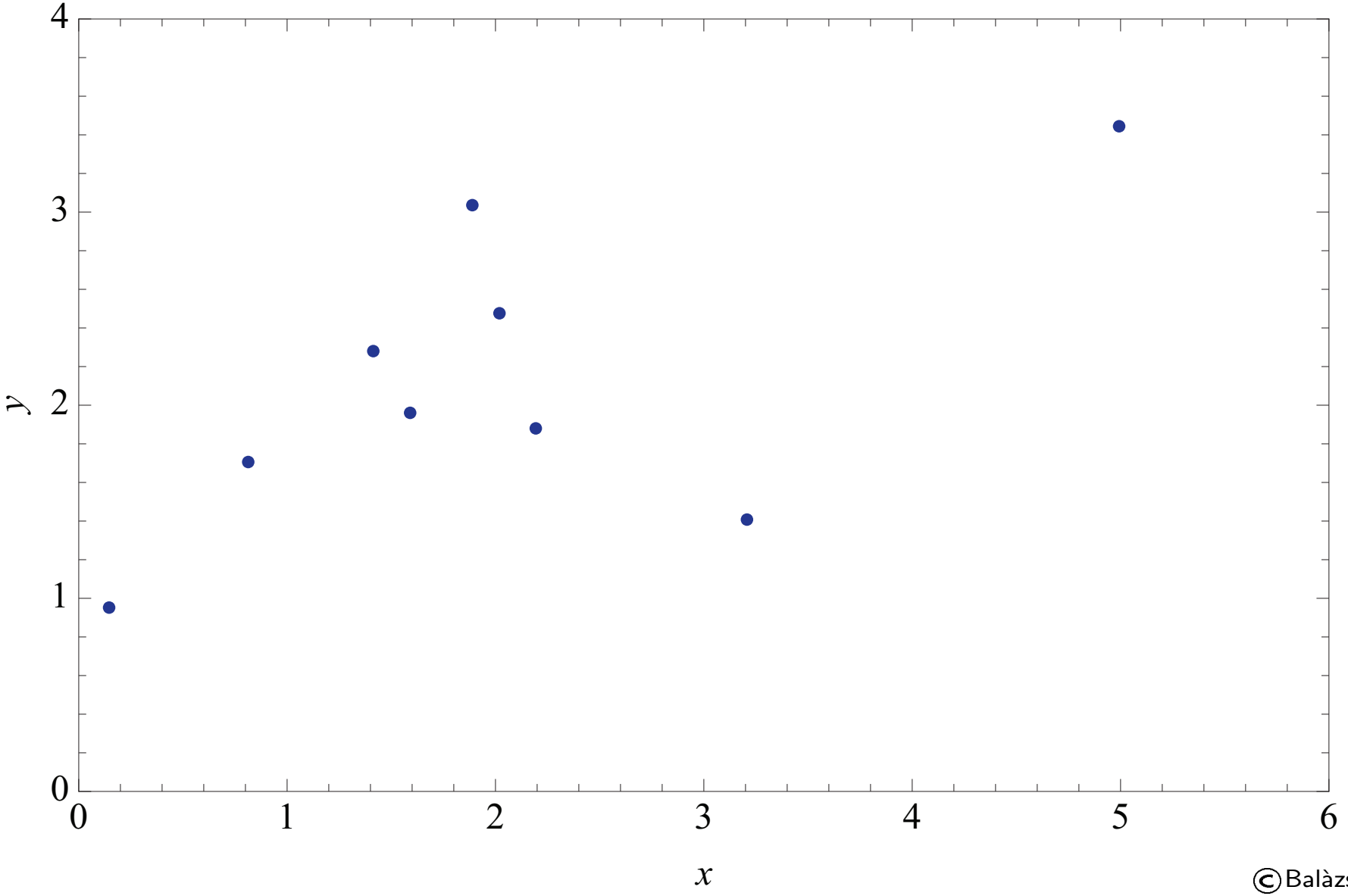
$$C(w) = R(w) + \lambda Q(w)$$

where λ controls the strength of regularisation

- At the minimum of $C(w)$ we select $f(x, w_*)$, our estimate of $y = f(x)$

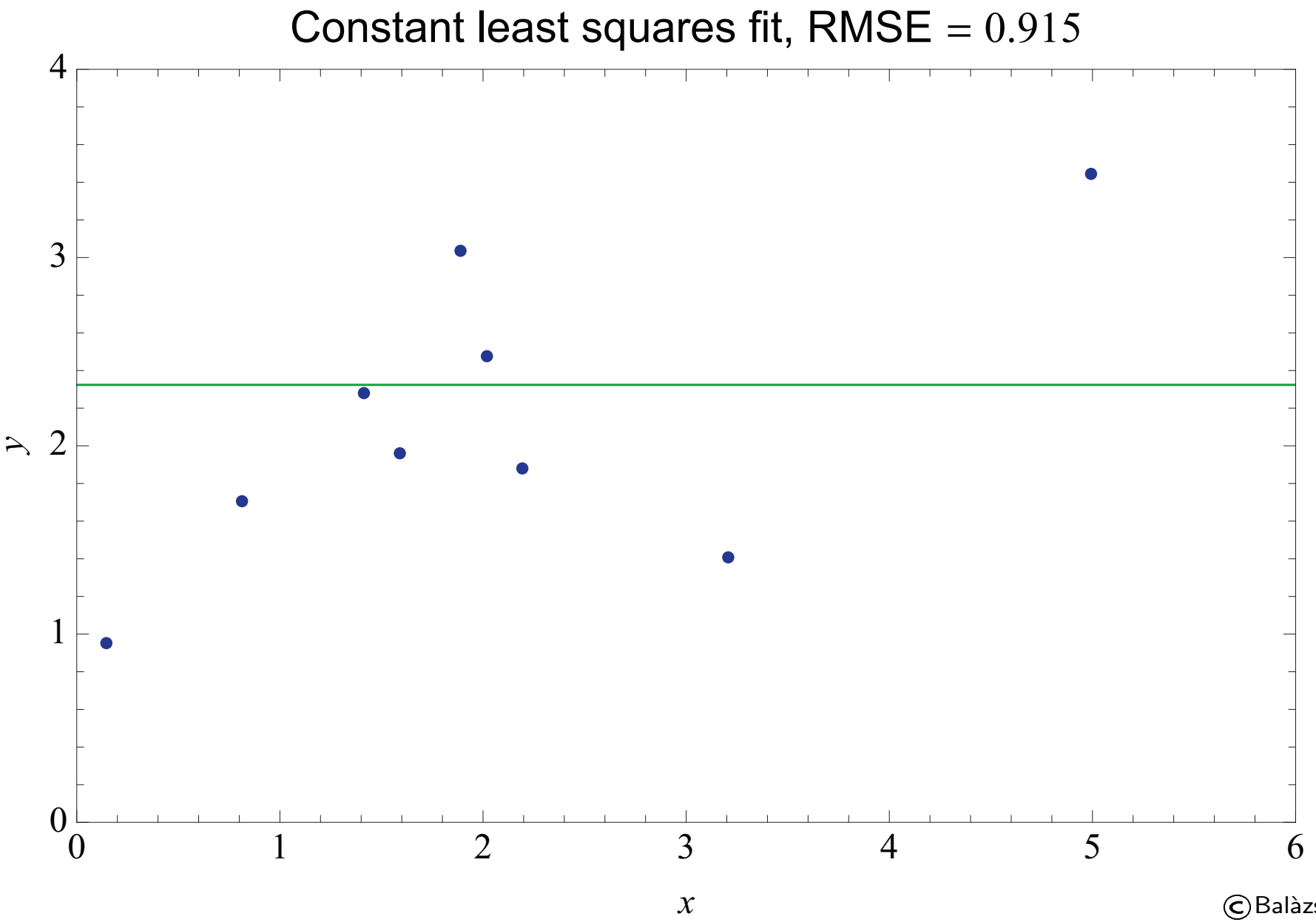
Choice of function class: training

Data generated from an unknown function with unknown noise



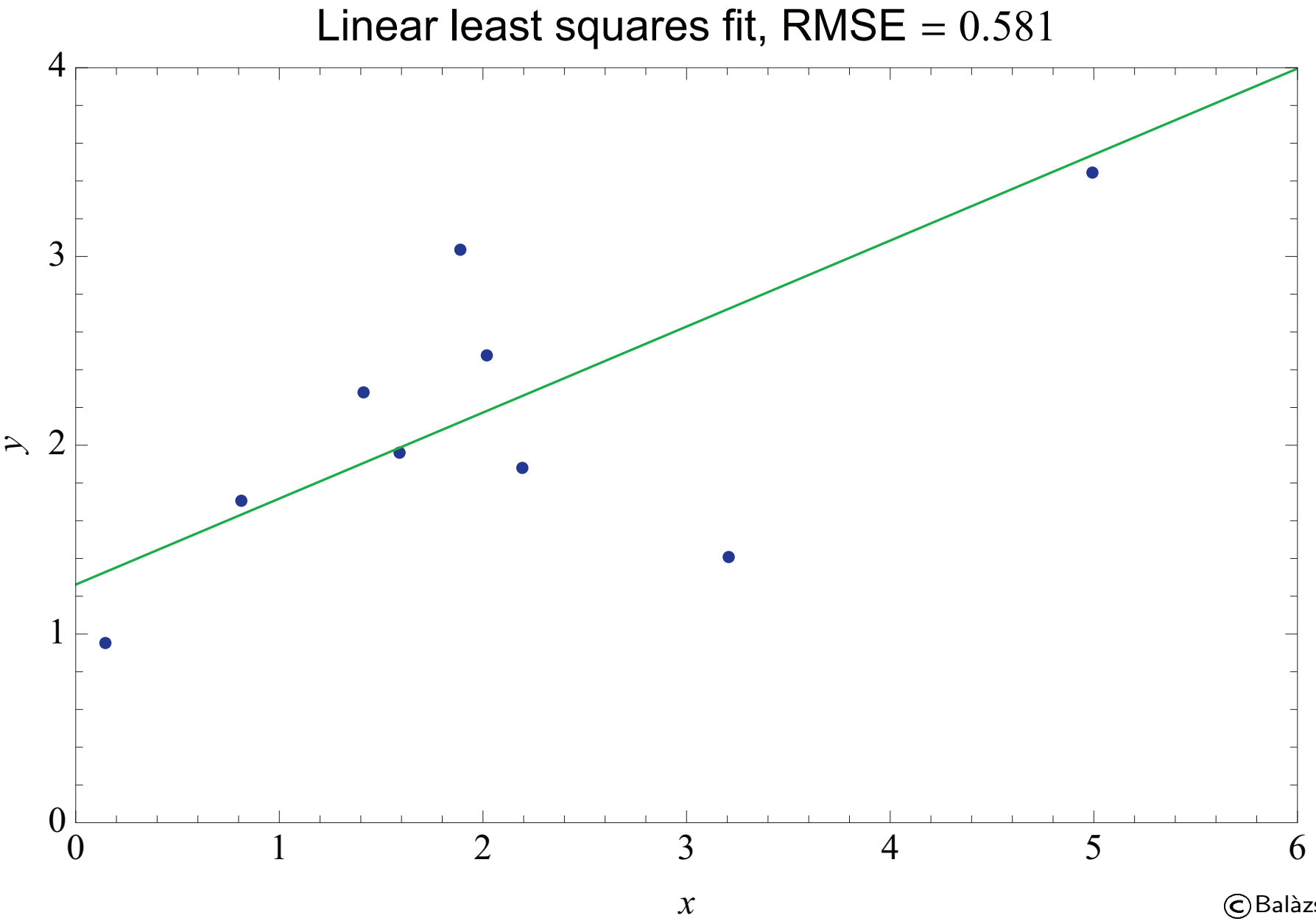
©Balàzs Kégl

Choice of function class: training

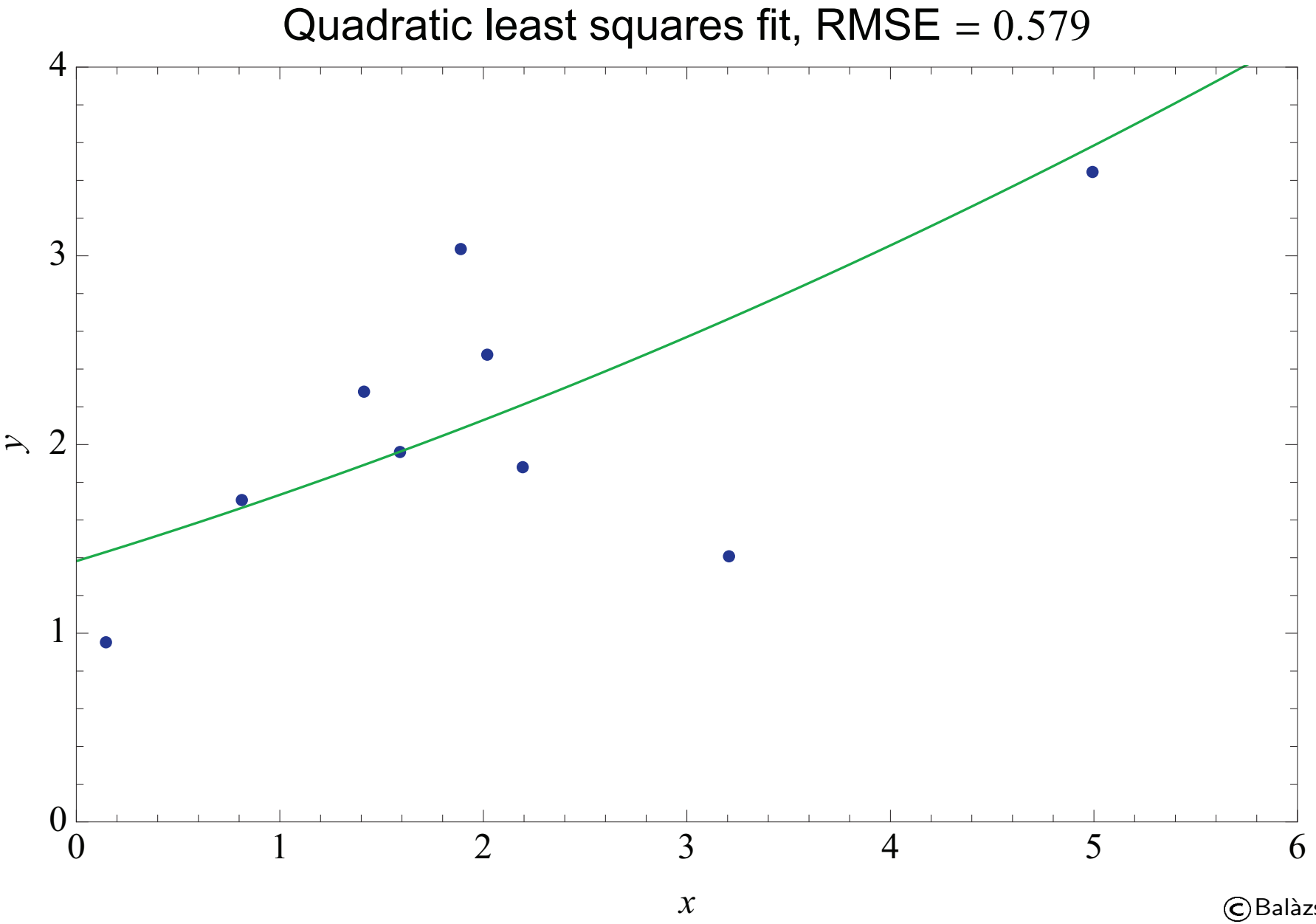


©Balàzs Kégl

Choice of function class: training

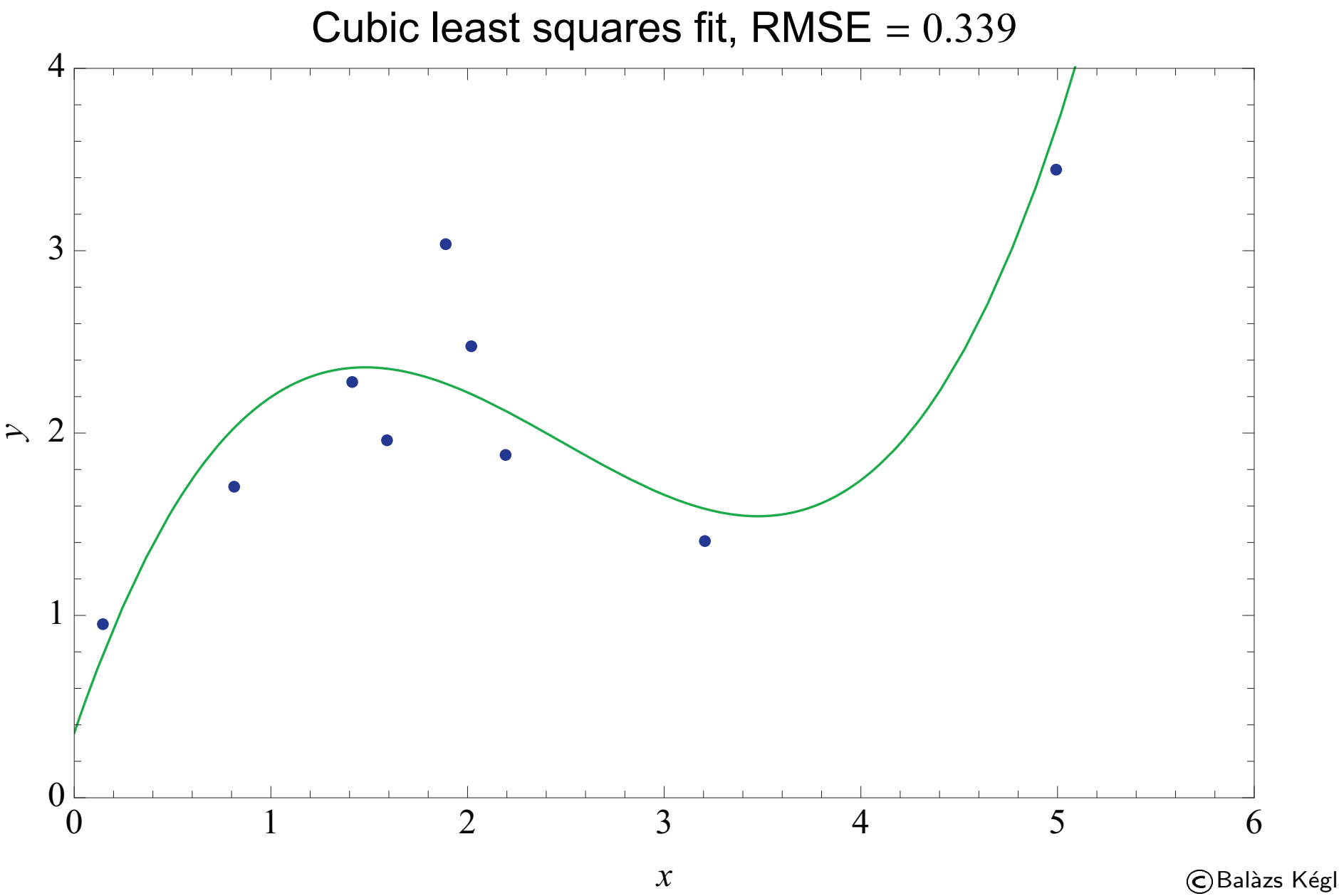


Choice of function class: training

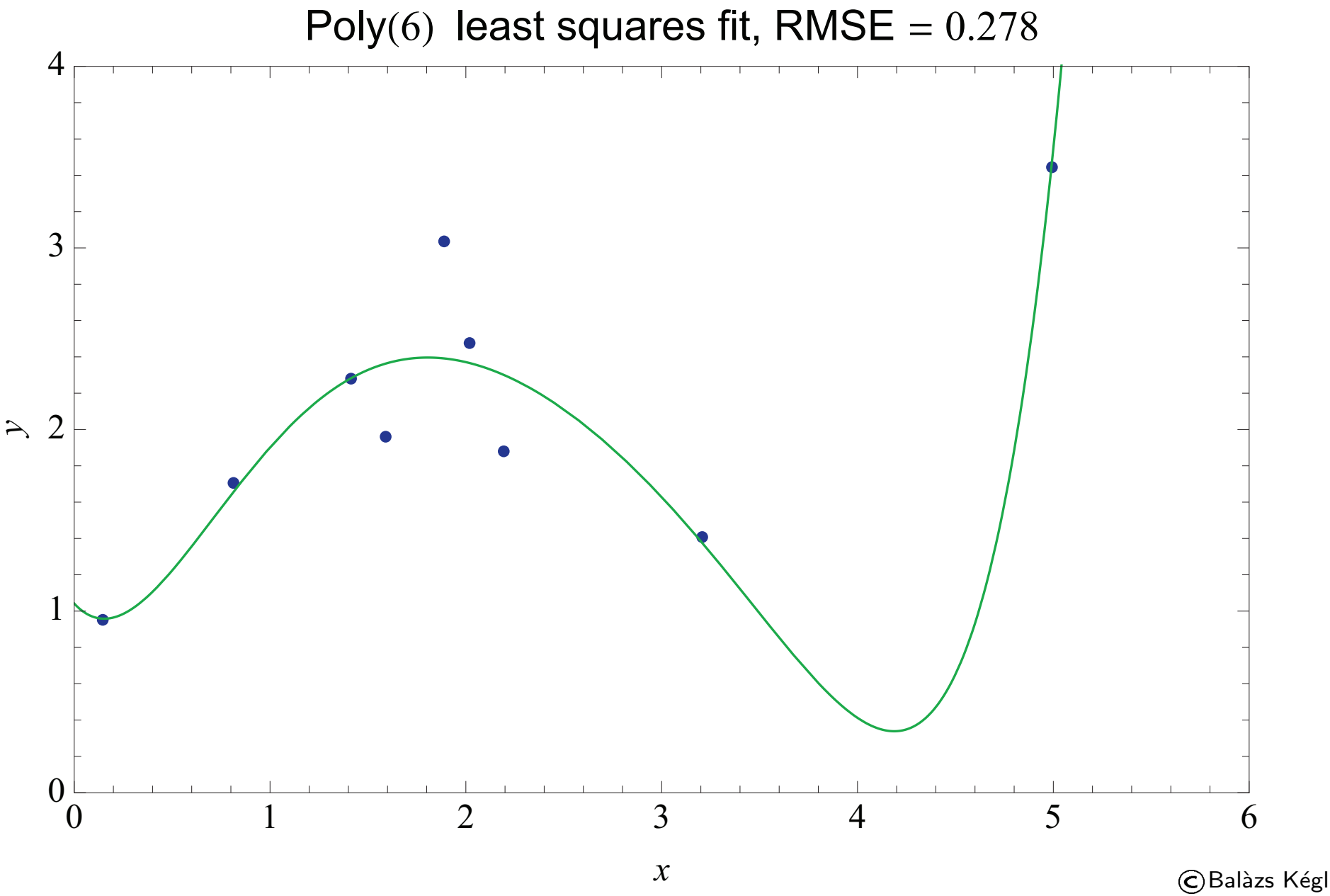


©Balàzs Kégl

Choice of function class: training

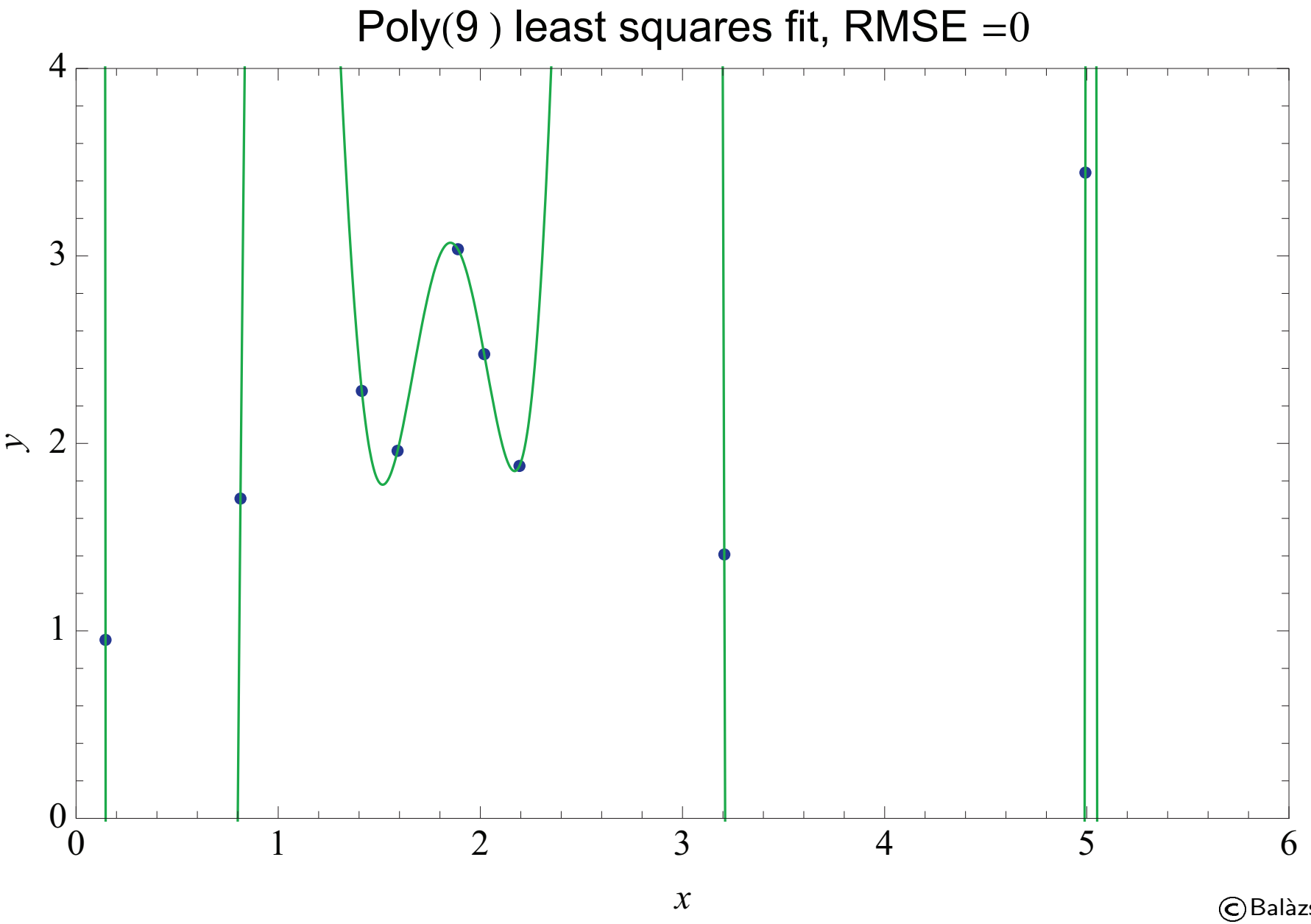


Choice of function class: training



©Balàzs Kégl

Choice of function class: training



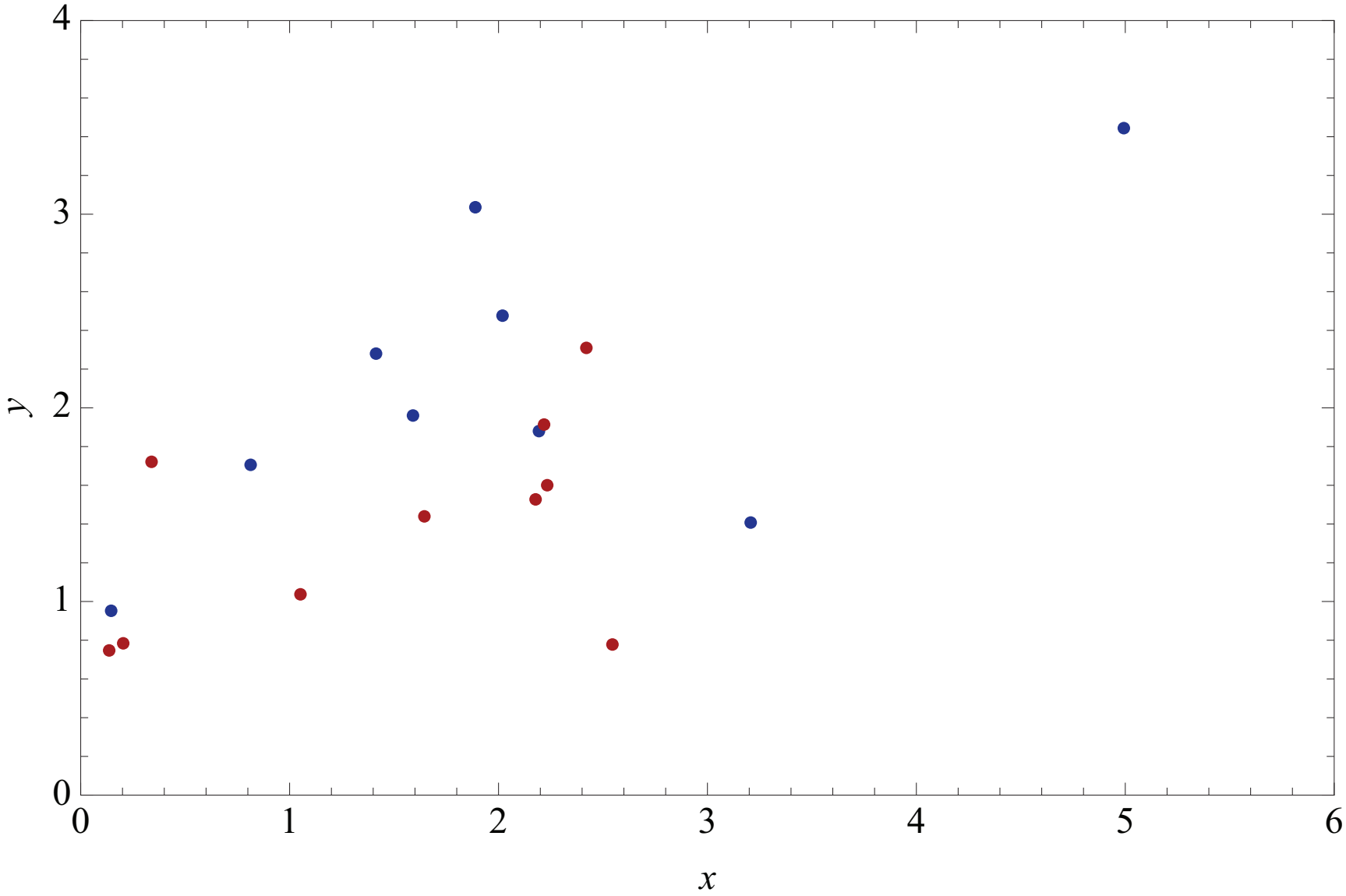
©Balàzs Kégl

Choice of function class

What happens if running on new data points?

Choice of function class: testing

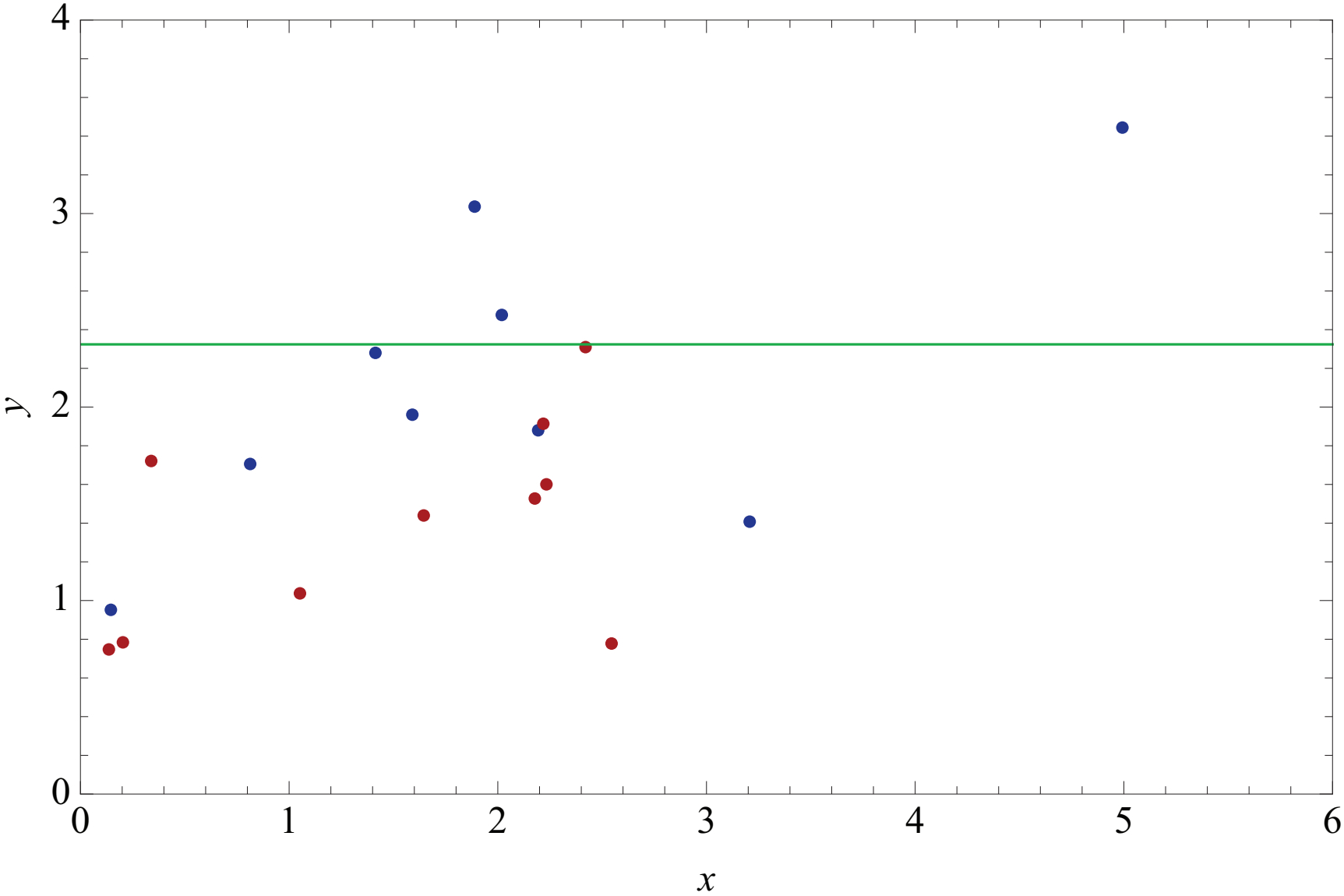
Data generated from an unknown function with unknown noise



©Balàzs Kégl

Choice of function class: testing

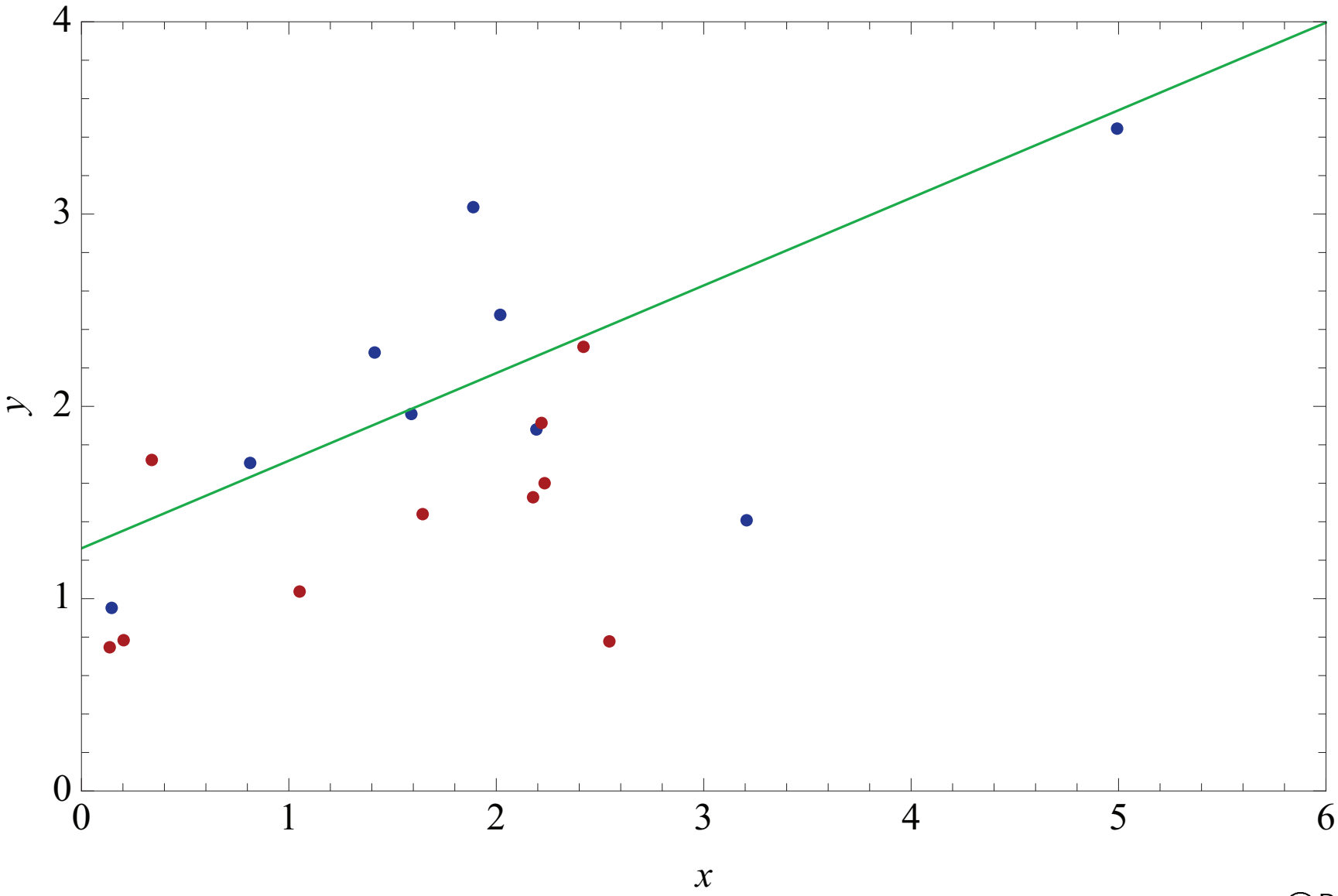
Const. least squares fit, training RMSE = 0.915, test RMSE = 1.067



©Balàzs Kégl

Choice of function class: testing

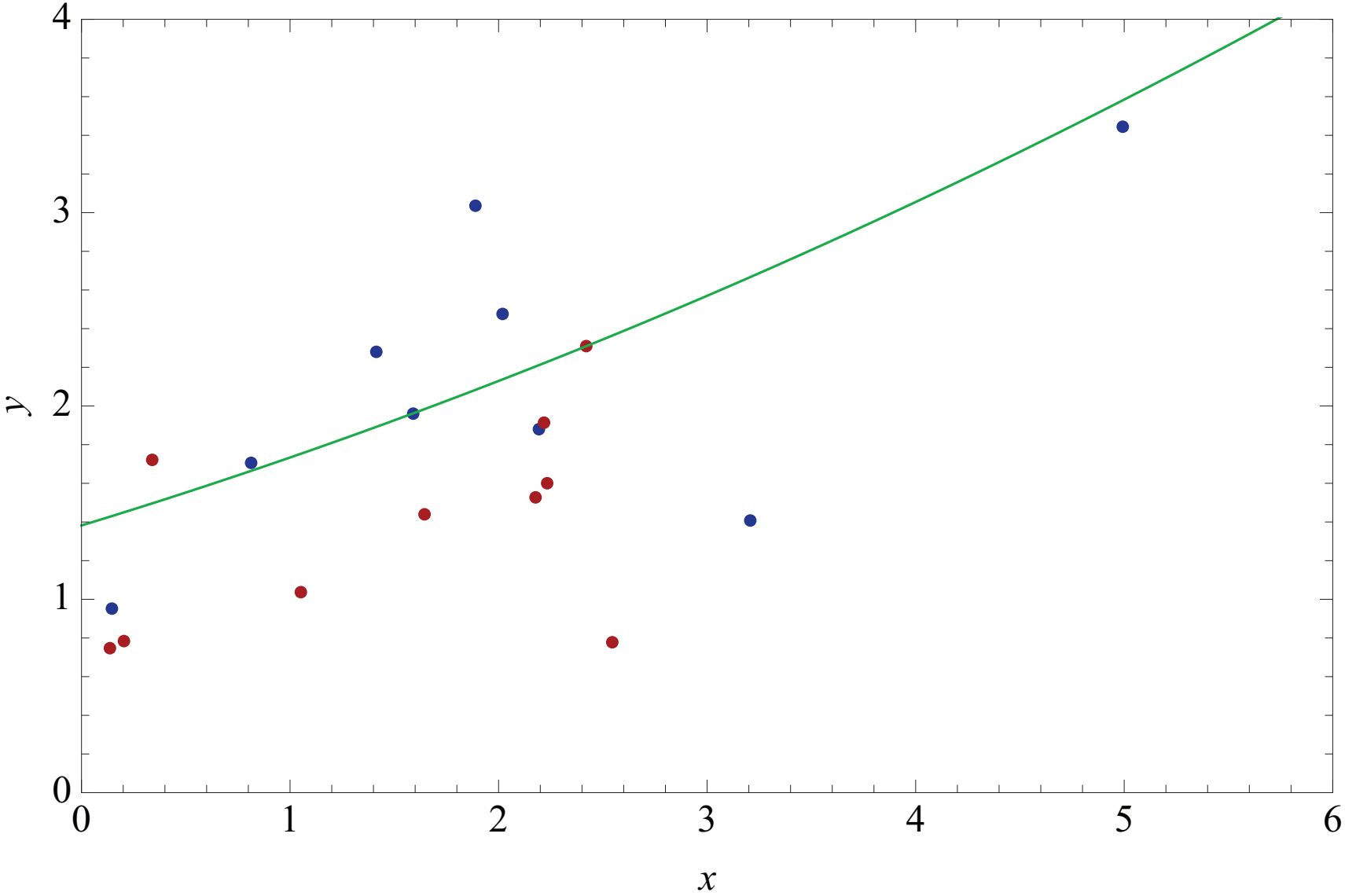
Linear least squares fit, training RMSE = 0.581, test RMSE = 0.734



©Balàzs Kégl

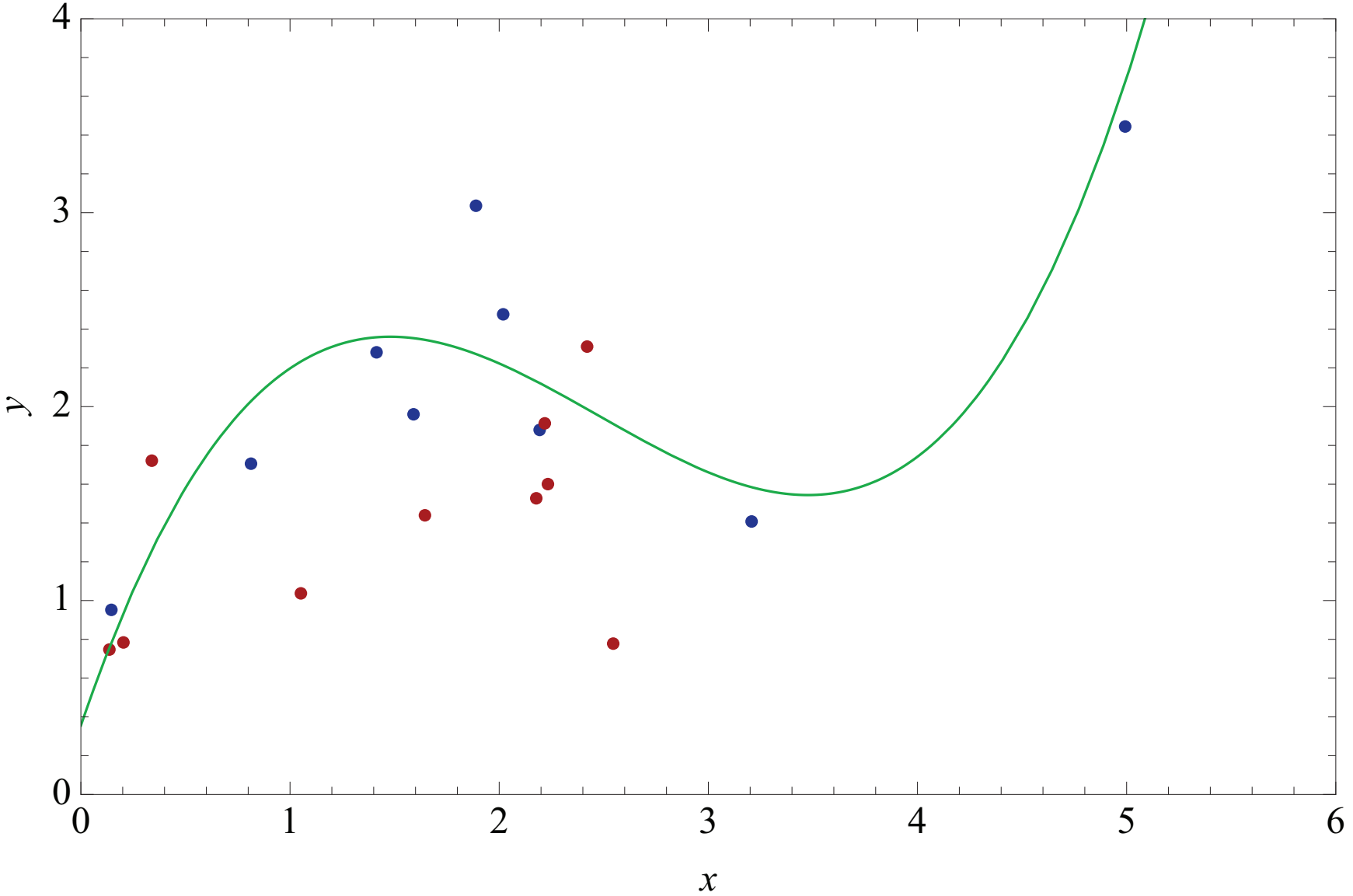
Choice of function class: testing

Quadr. least squares fit, training RMSE = 0.579, test RMSE = 0.723



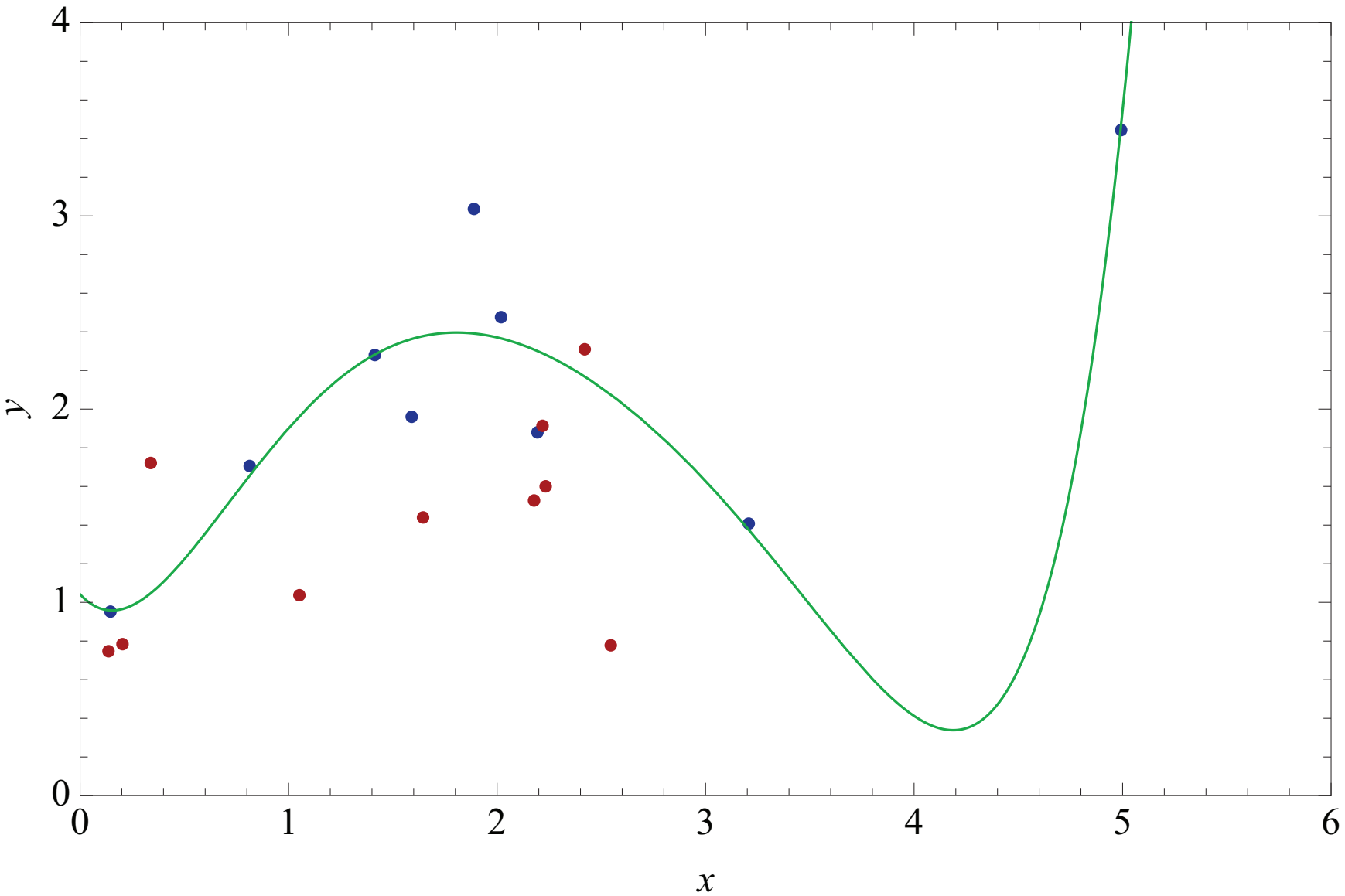
©Balàzs Kégl

Choice of function class: testing



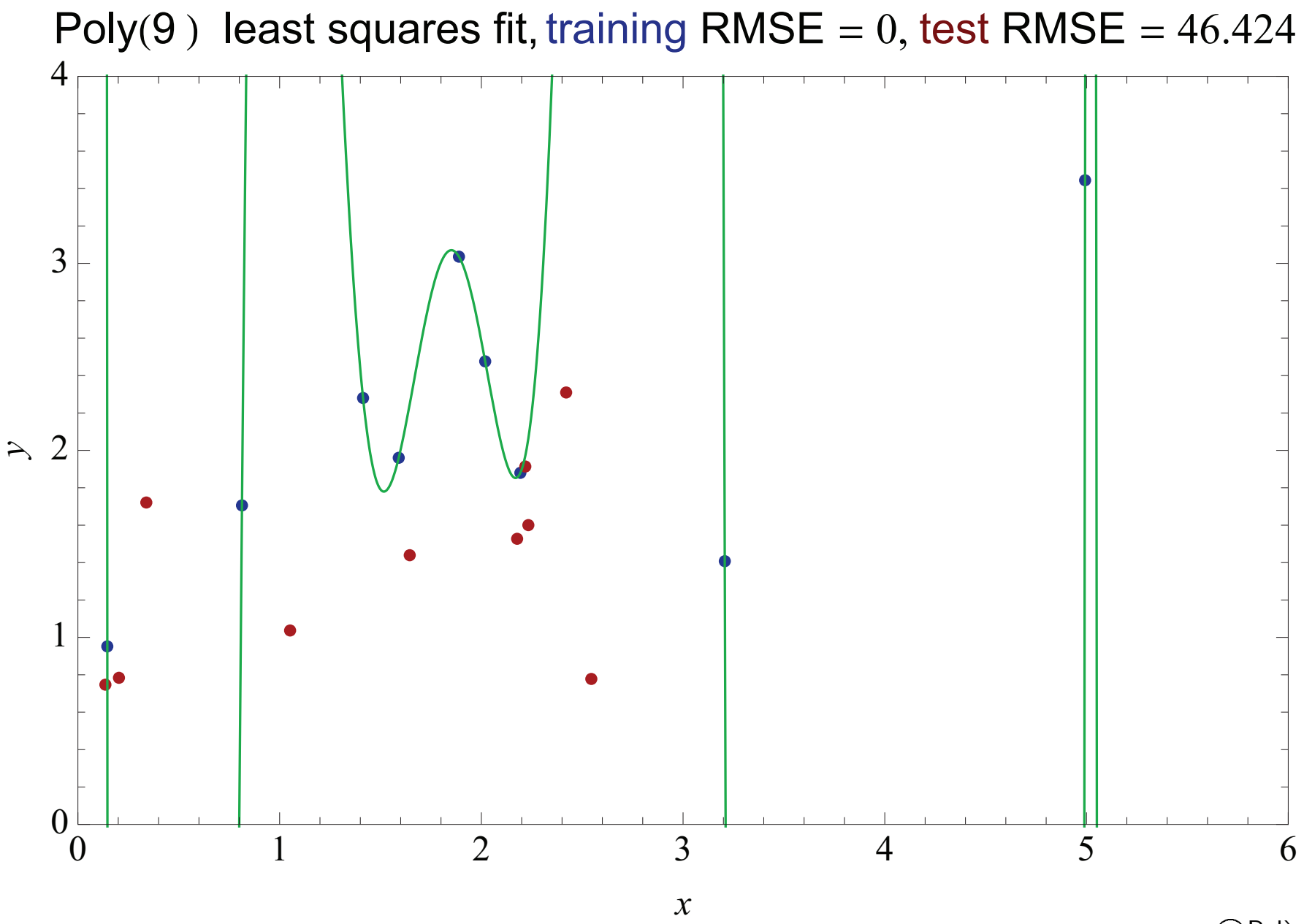
©Balàzs Kégl

Choice of function class: testing



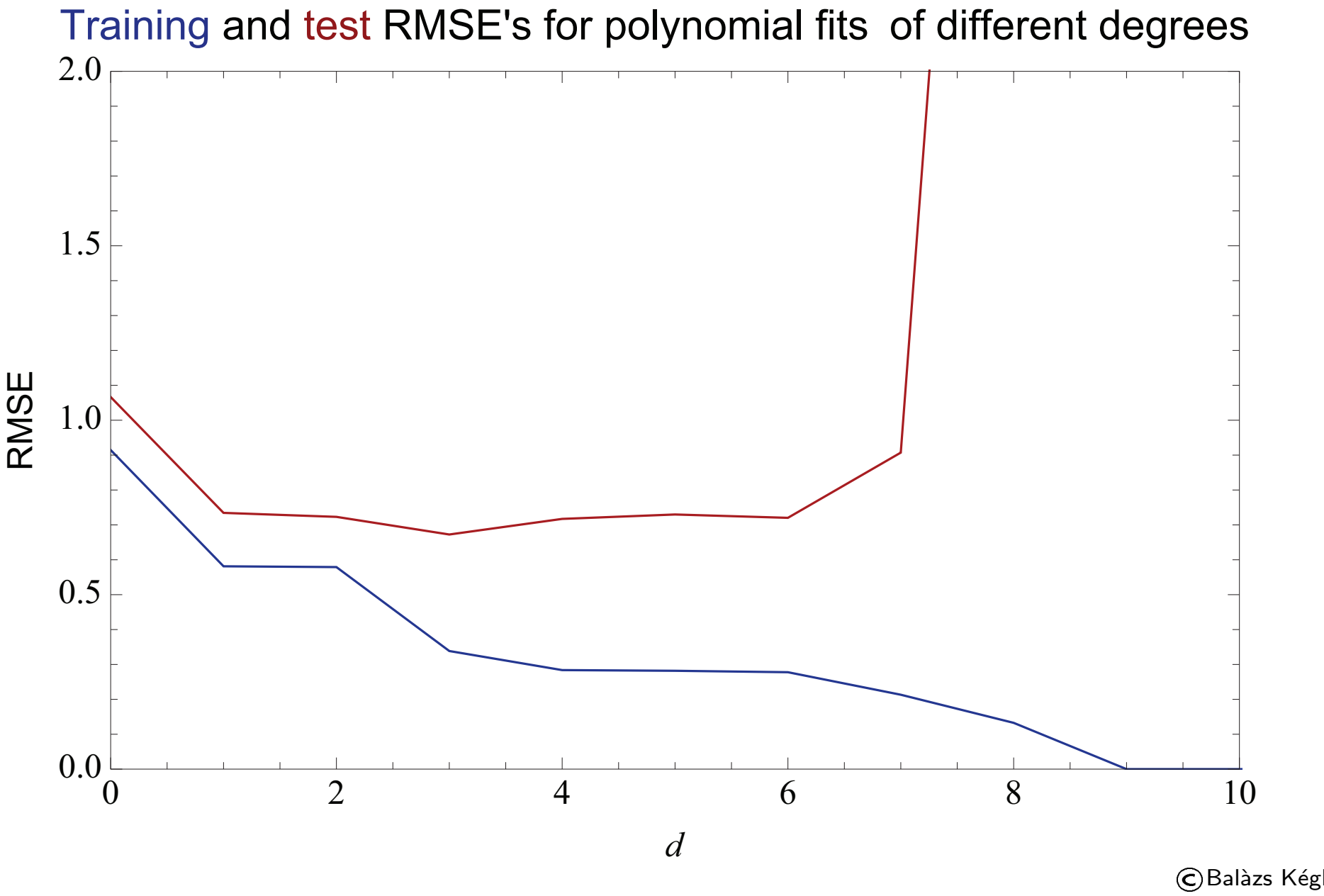
©Balàzs Kégl

Choice of function class: testing



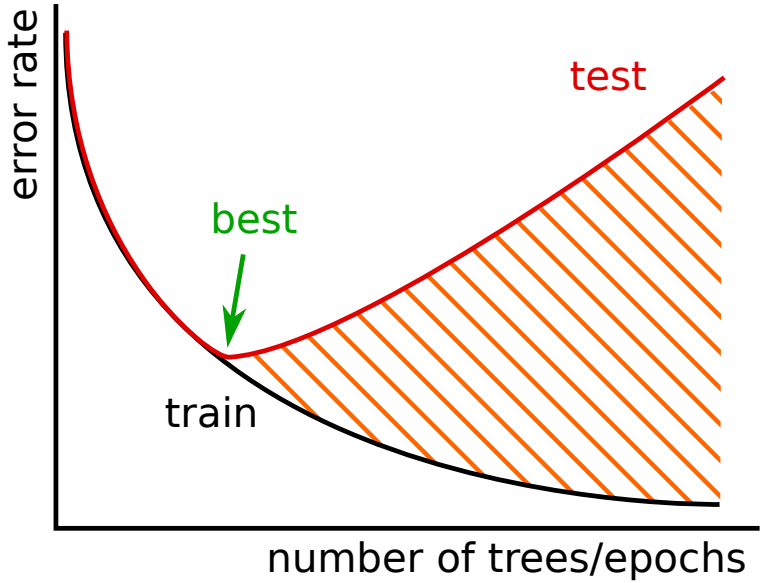
©Balàzs Kégl

Choice of function class



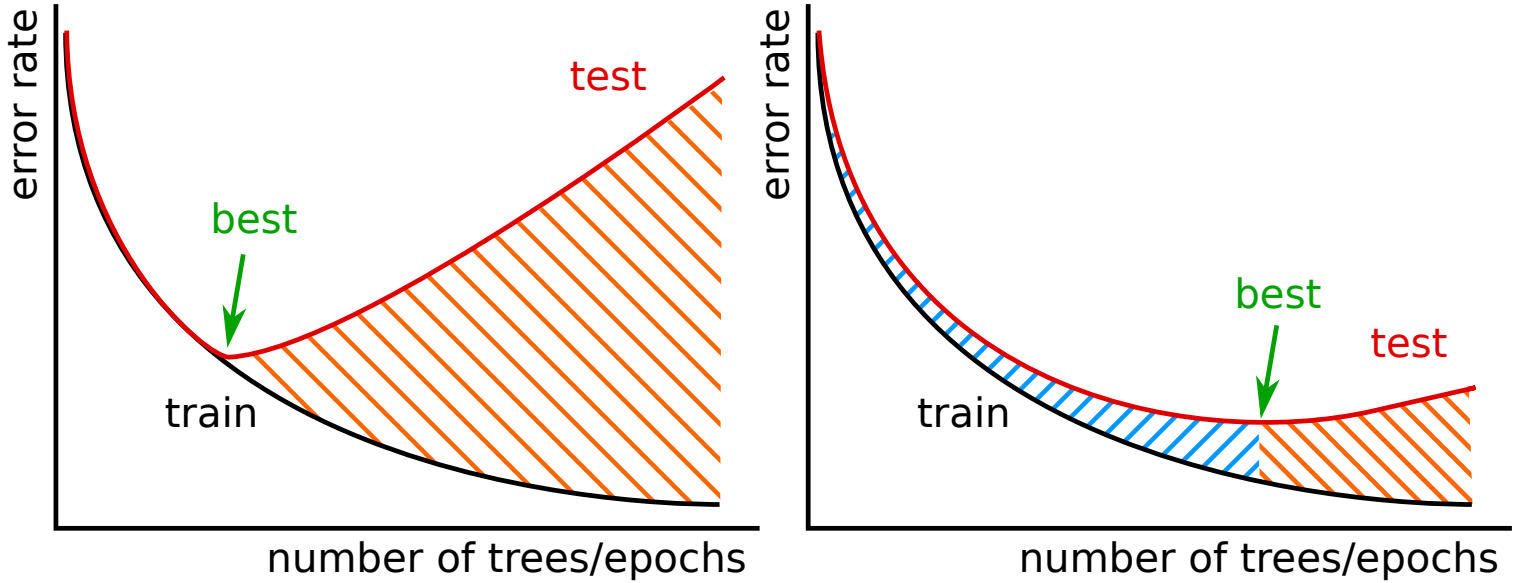
Overtraining estimation: good or bad?

"bad" overtraining (overfitting) / "good" overtraining (still underfitting)



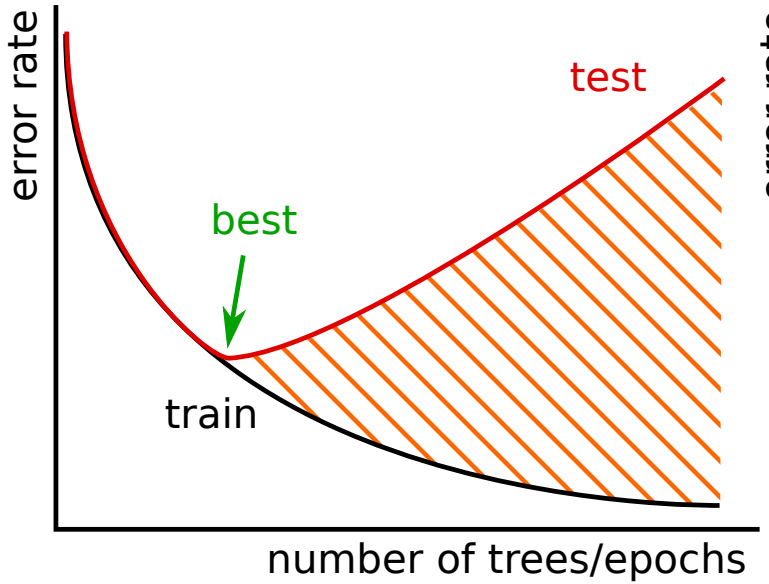
Overtraining estimation: good or bad?

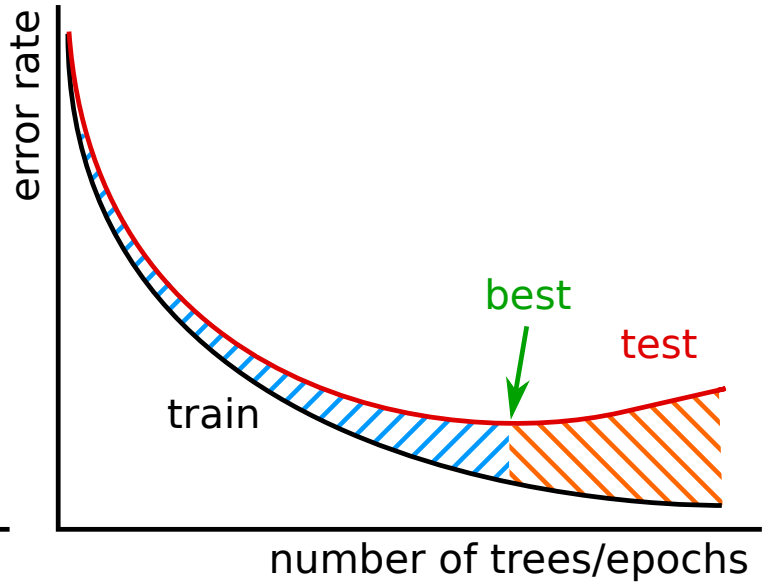
"bad" overtraining (overfitting) / "good" overtraining (still underfitting)

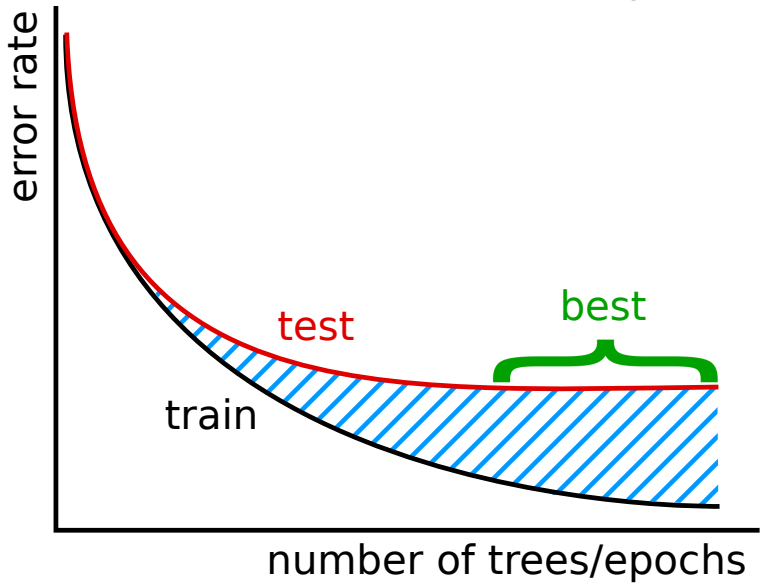


Overtraining estimation: good or bad?

"bad" overtraining (overfitting) / "good" overtraining (still underfitting)

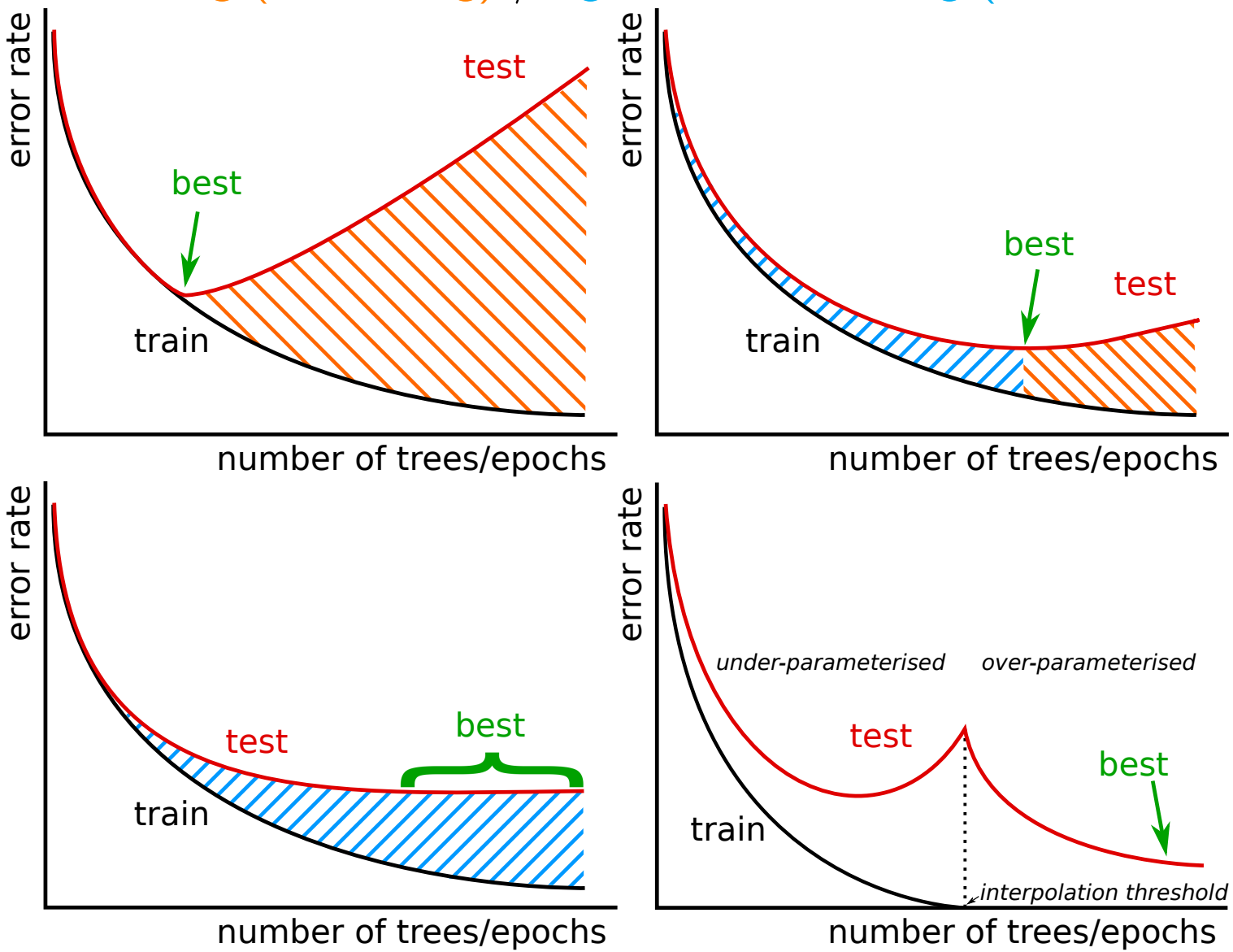






Overtraining estimation: good or bad?

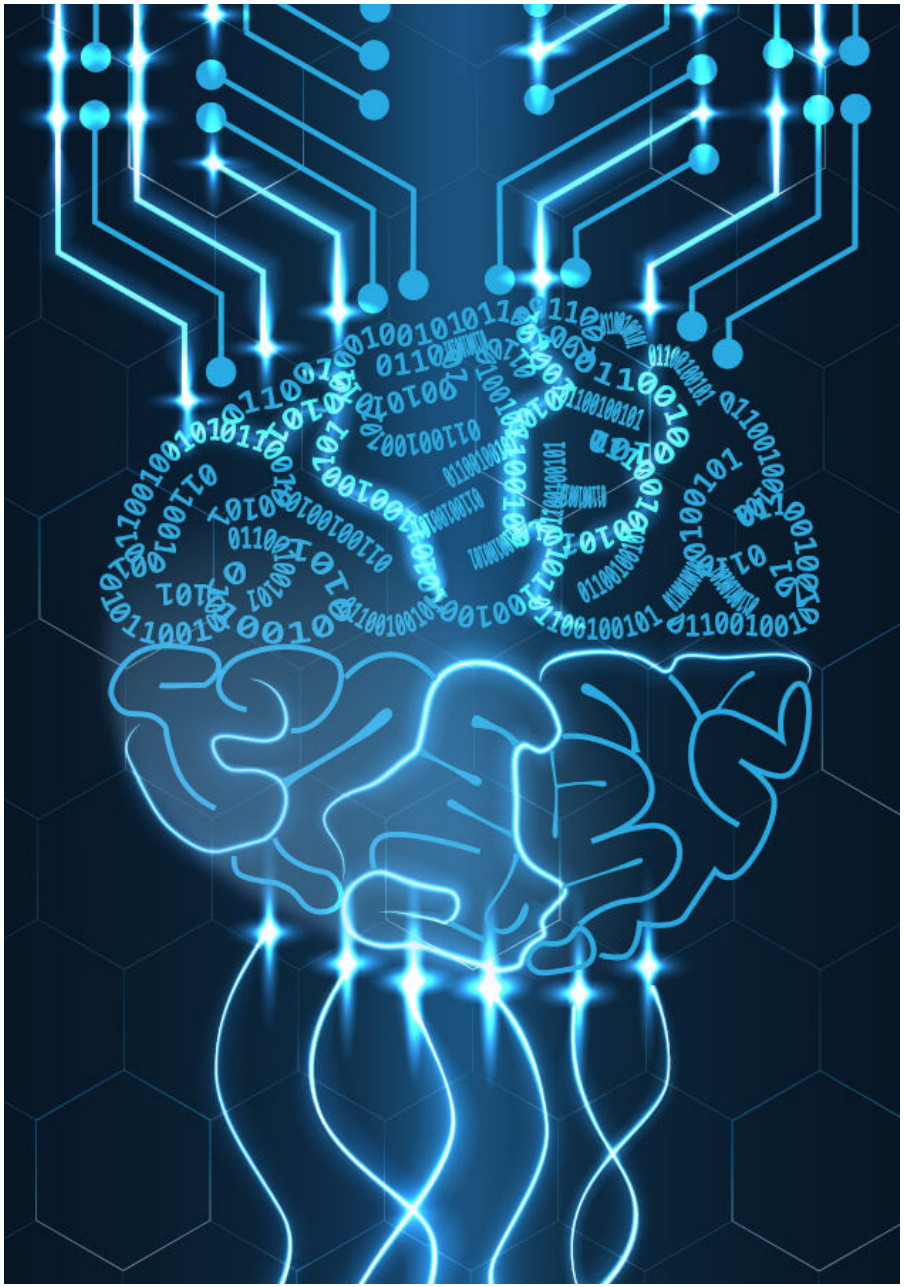
"bad" overtraining (overfitting) / "good" overtraining (still underfitting)



Capacity control and regularisation

- Trade-off between approximation error and estimation error
- Take into account sample size
- Measure (and penalise) complexity
- Use independent test sample
- In practice, no need to correctly guess the function class, but need enough flexibility in your model, balanced with complexity cost

Machine learning algorithms



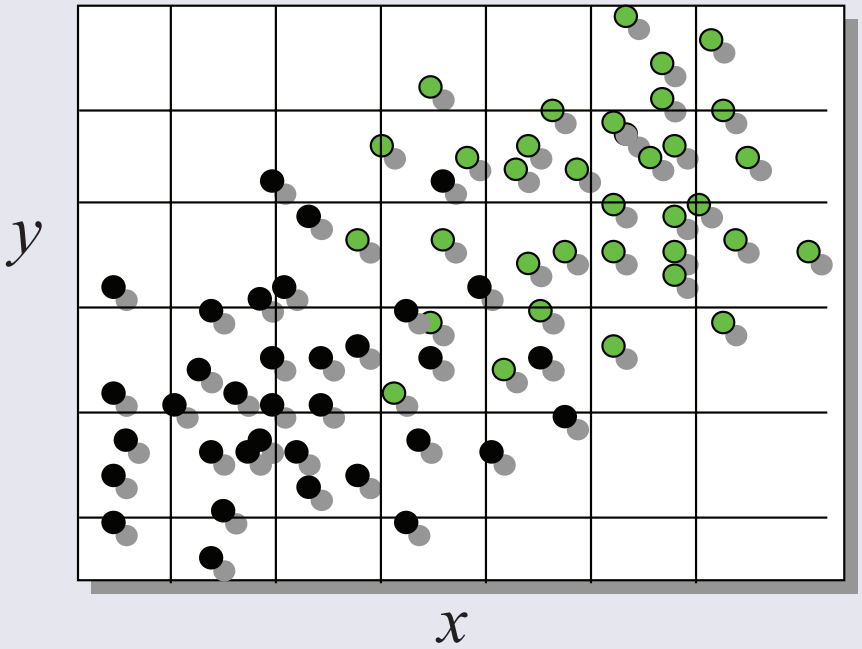
- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Kernel density estimation
- Decision trees
- Boosted decision trees
- Neural networks
- Deep neural networks

Cut-based analysis and grid search

Cut-based analysis

- Simple approach: cut on each discriminating variable
- Difficulty: how to optimise the cuts?

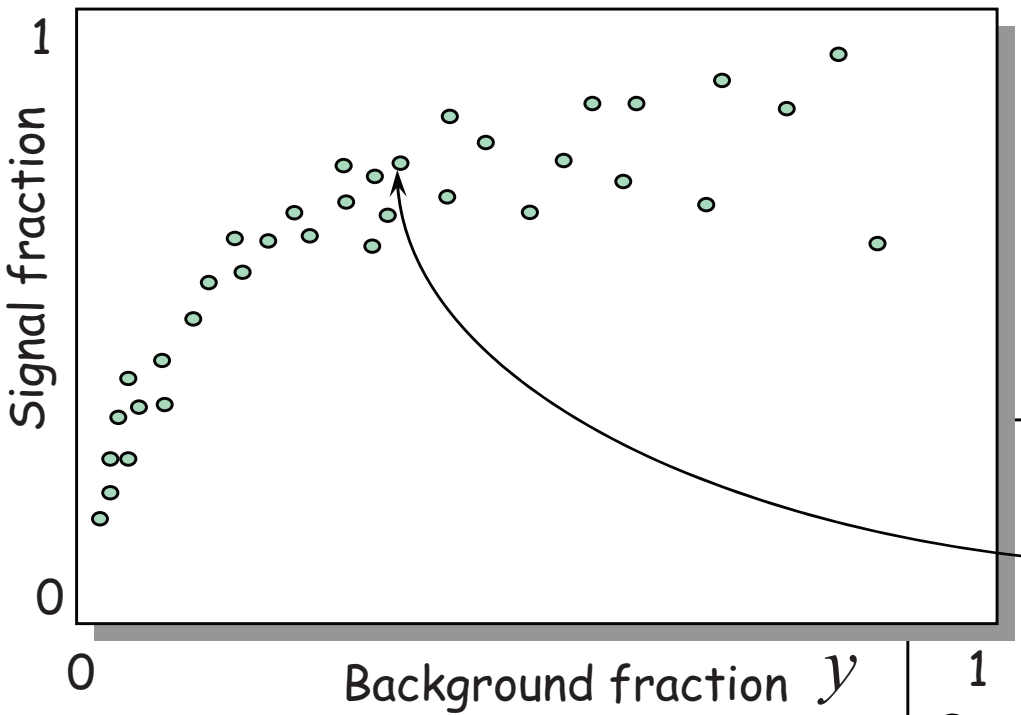
Grid search



©Harrison Prosper

- Split each variable in K values
- Apply cuts at each grid point: $x > x_i$, $y > y_i$
- Number of points scales with K^n: curse of dimensionality

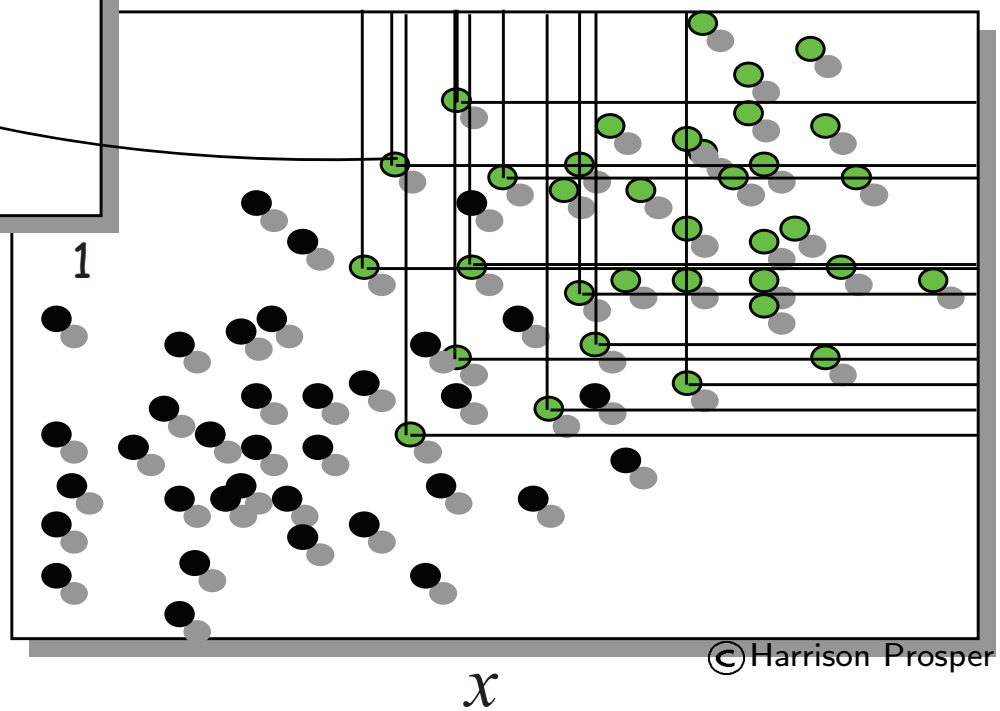
Random grid search



- Use each point in signal sample as grid point:

$$x > x_i$$

$$y > y_i$$



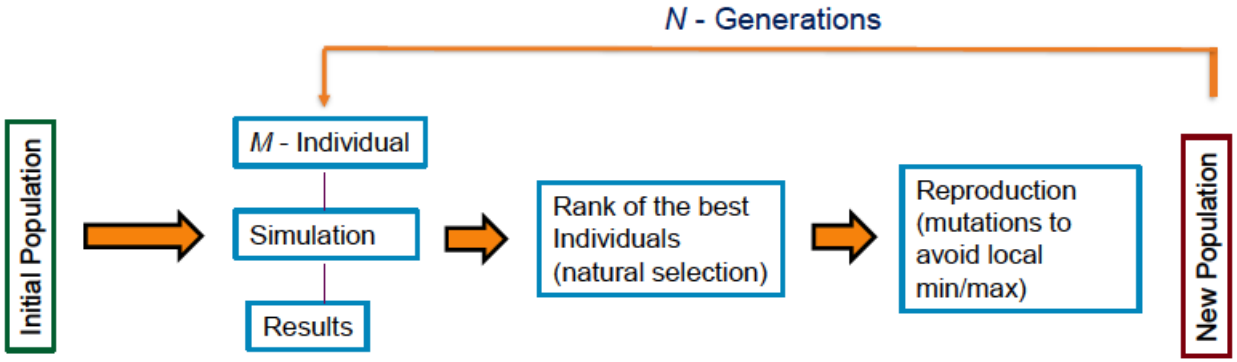
- Number of cut points independent of dimensionality
- Sampled points density follows signal density

Genetic algorithms: survival of the fittest

- Inspired by biological evolution
- Model: group (population) of abstract representations (genome/discriminating variables) of possible solutions (individuals/list of cuts)
- Typical processes at work in evolutionary processes:
  - inheritance
  - mutation
  - sexual recombination (a.k.a. crossover)
- Fitness function: value representing the individual's goodness, or comparison of two individuals
- For cut optimisation:
  - good background rejection and high signal efficiency
  - compare individuals in each signal efficiency bin and keep those with higher background rejection

Genetic algorithms

- Better solutions more likely to be selected for mating and mutations, carrying their genetic code (cuts) from generation to generation
- Algorithm:
  1. Create initial random population (cut ensemble)
  2. Select fittest individuals
  3. Create offsprings through crossover (mix best cuts)
  4. Mutate randomly (change some cuts of some individuals)
  5. Repeat from 2 until convergence (or fixed number of generations)



- Good fitness at one generation $\Rightarrow$ average fitness in the next
- Algorithm focuses on region with higher potential improvement

Quadratic discriminants: Gaussian problem

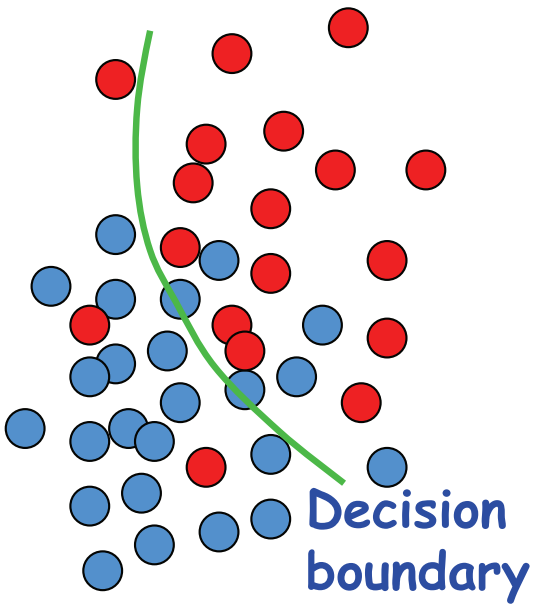
- Suppose densities $s(x)$ and $b(x)$ are multivariate Gaussians:

$$\text{Gaussian}(x|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$

  with vector of means μ and covariance matrix Σ

- Then $B(x) = s(x)/b(x)$ (or its logarithm) can be expressed explicitly:

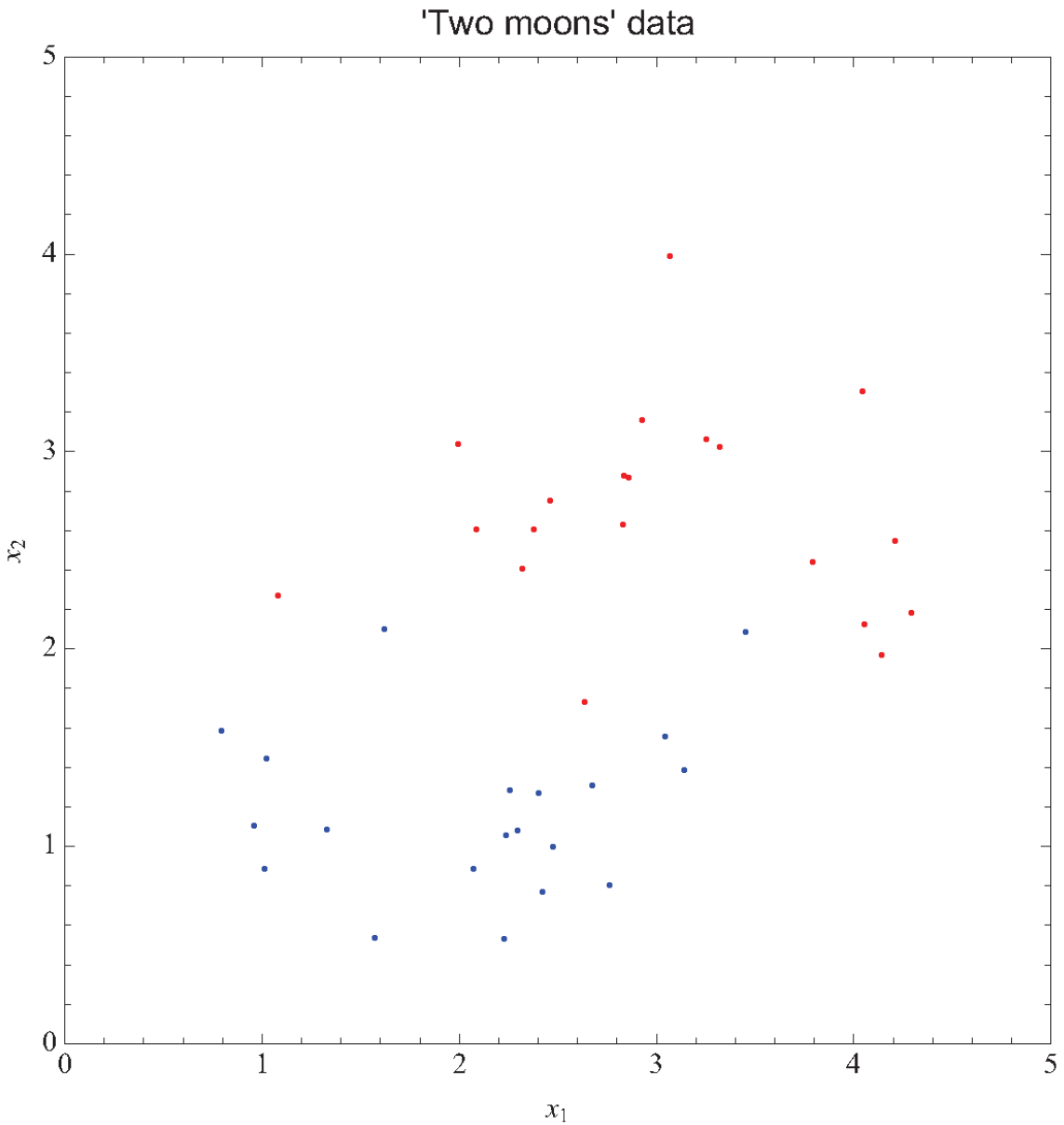
$$\ln B(x) = \lambda(x) \equiv \chi^2(\mu_B,\Sigma_B) - \chi^2(\mu_S,\Sigma_S)$$



with $\chi^2(\mu,\Sigma) = (x-\mu)^T\Sigma^{-1}(x-\mu)$

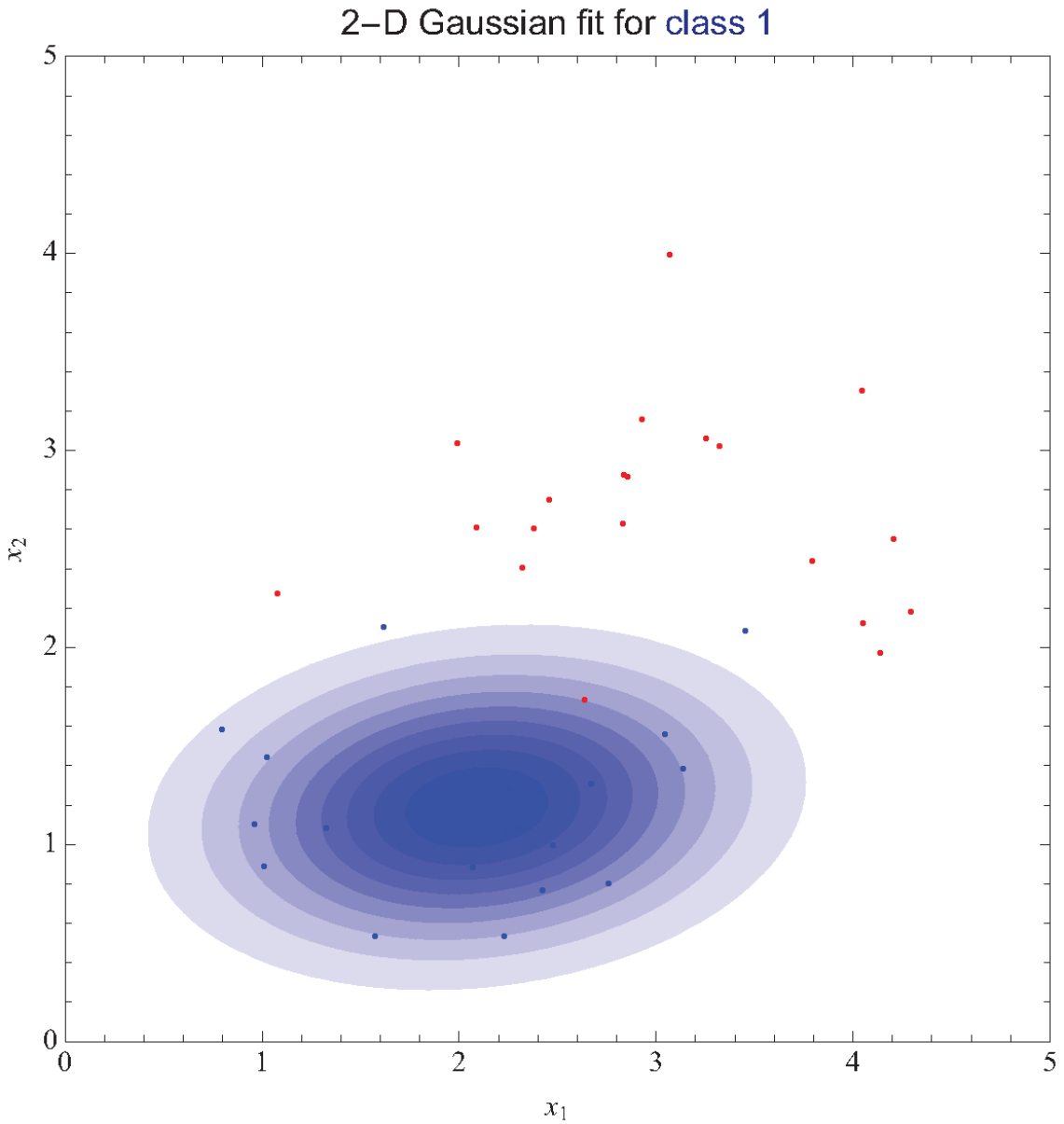
- Fixed value of $\lambda(x)$ defines quadratic hypersurface partitioning n-dimensional space into signal-rich and background-rich regions
- Optimal separation if $s(x)$ and $b(x)$ are indeed multivariate Gaussians

Quadratic discriminant



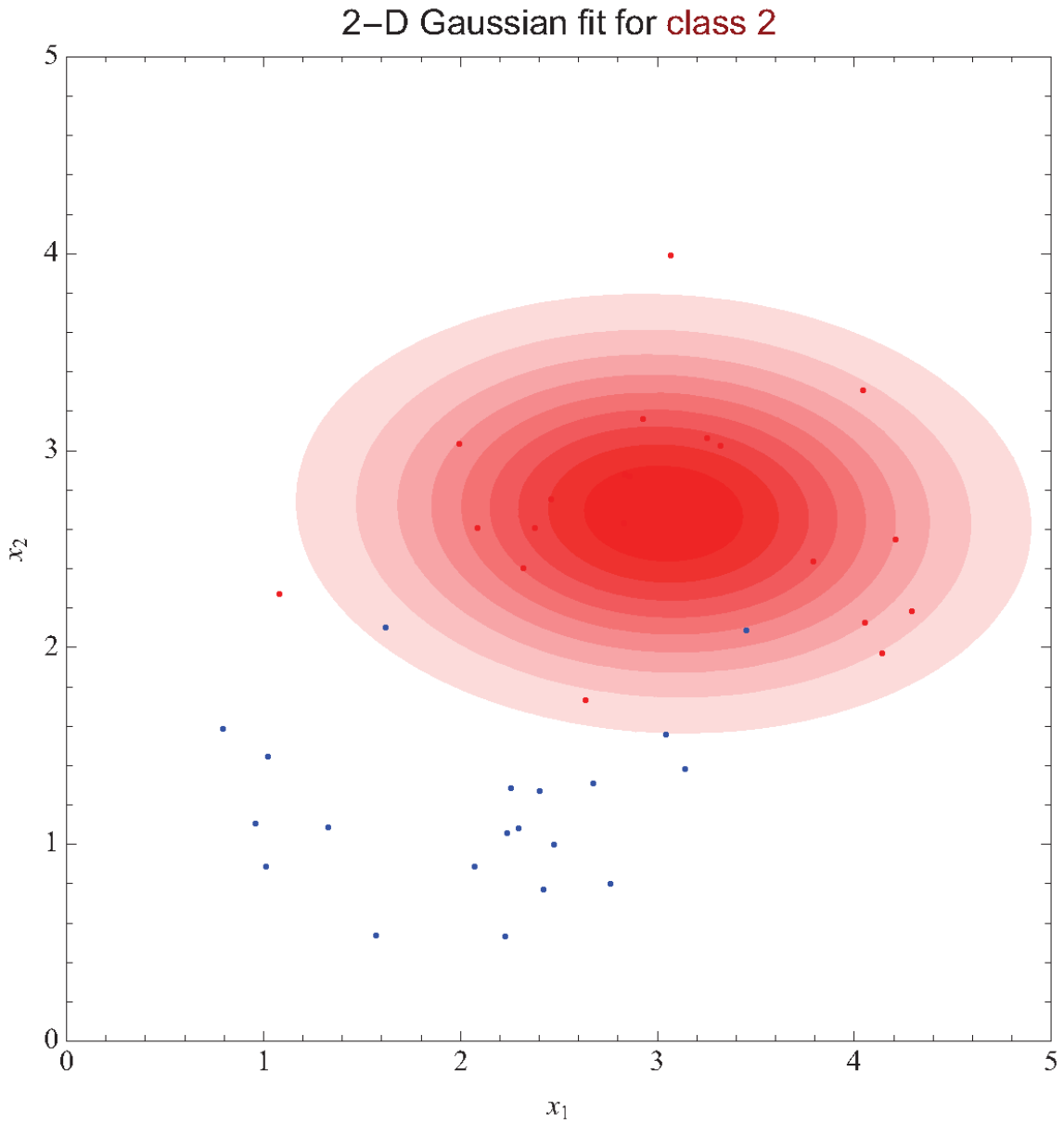
©Balàzs Kégl

Quadratic discriminant



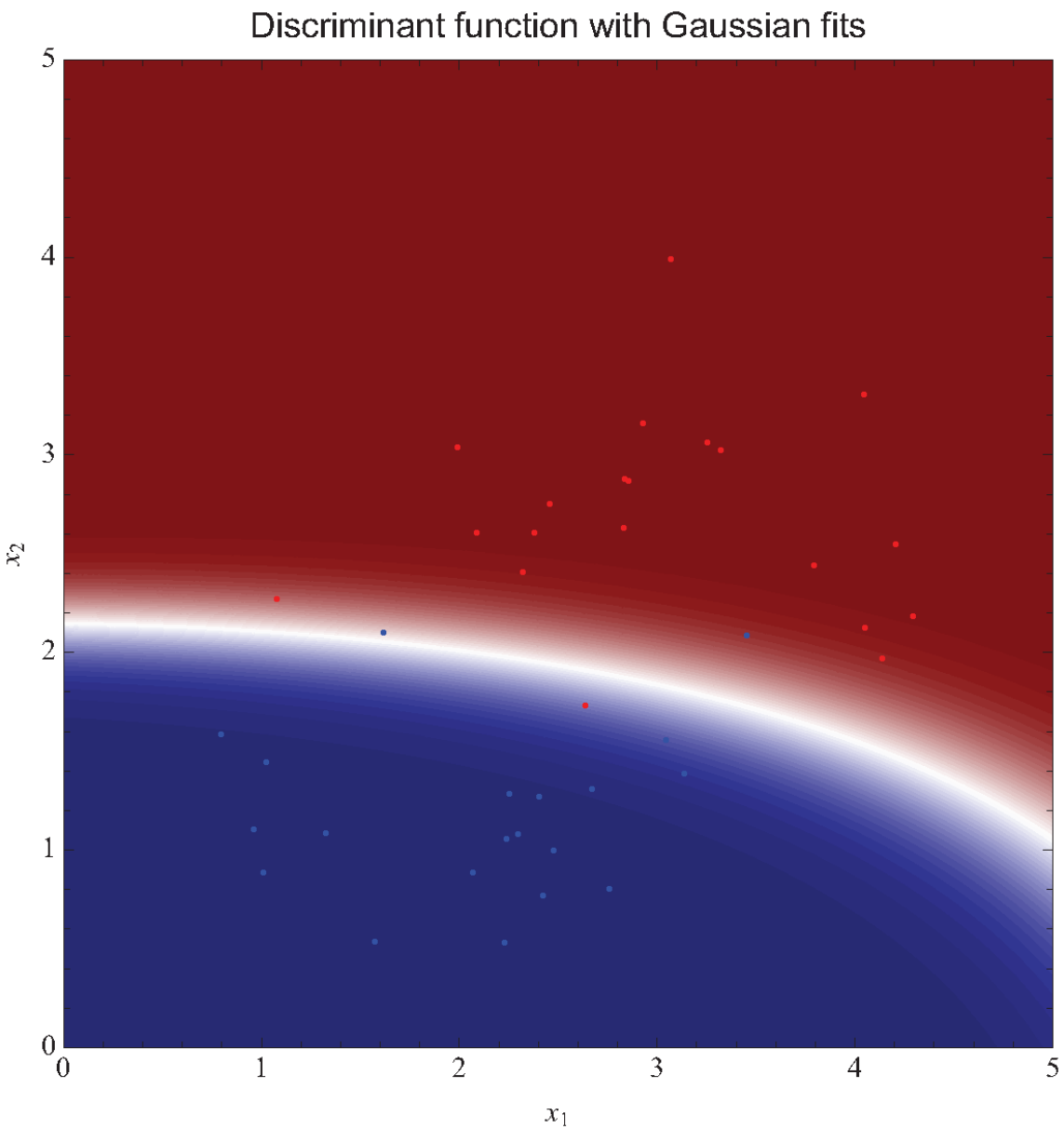
©Balàzs Kégl

Quadratic discriminant



©Balàzs Kégl

Quadratic discriminant

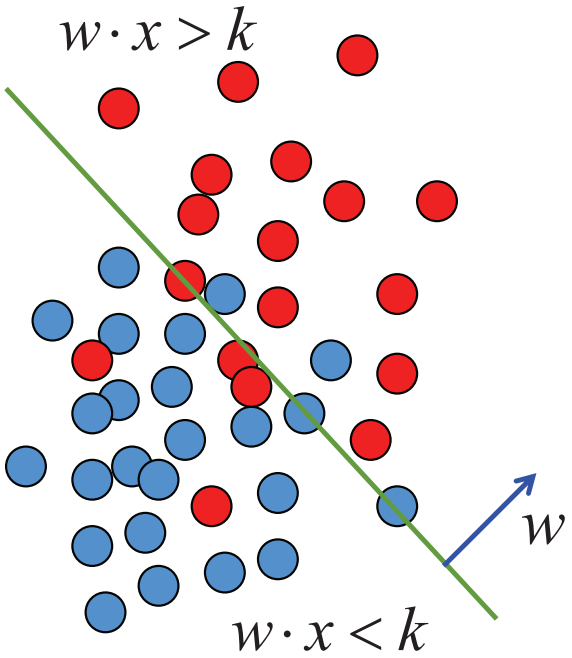


©Balàzs Kégl

Linear discriminant: Fisher's discriminant

- If in $\lambda(x)$ the same covariance matrix is used for each class (e.g. $\Sigma = \Sigma_S + \Sigma_B$) one gets Fisher's discriminant:

$$\lambda(x) = w \cdot x \quad \text{with} \quad w \propto \Sigma^{-1}(\mu_S - \mu_B)$$

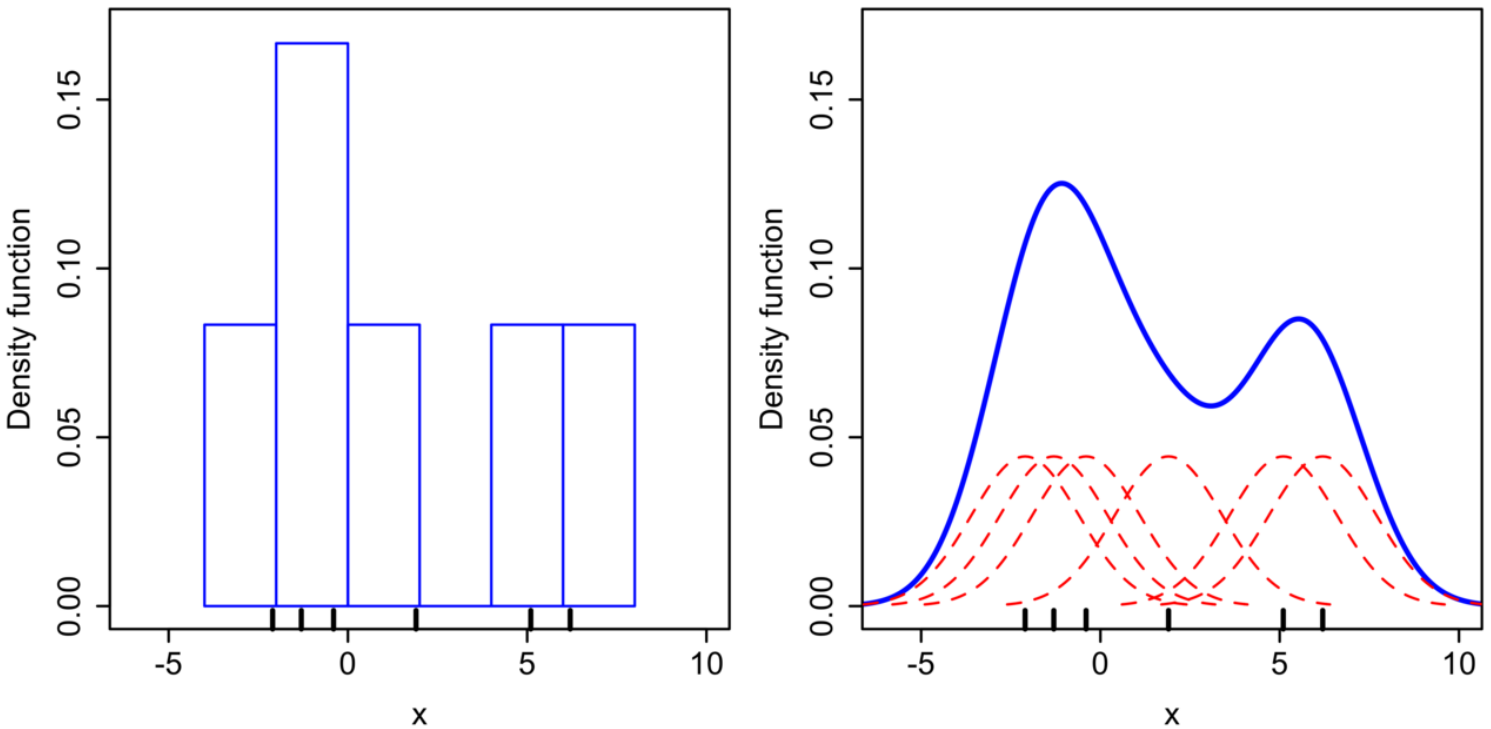


- Optimal linear separation
- Works only if signal and background have different means!
- Optimal classifier (reaches the Bayes limit) for linearly correlated Gaussian-distributed variables
- Extension to non-linear problems: support vector machines (see ▸ backup)

Kernel density estimation (KDE)

- Introduced by E. Parzen in the 1960s
- Place a kernel $K(x, \mu)$ at each training point μ
- Density $p(x)$ at point x approximated by:

$$p(x) \approx \hat{p}(x) = \frac{1}{N} \sum_{j=1}^{N} K(x, \mu_j)$$



Kernel density estimation (KDE)

Choice of kernel

- Any kernel can be used
- In practice, often product of Gaussians:

$$K(x, \mu) = \prod_{i=1}^{n} Gaussian(x_i|\mu, h_i)$$

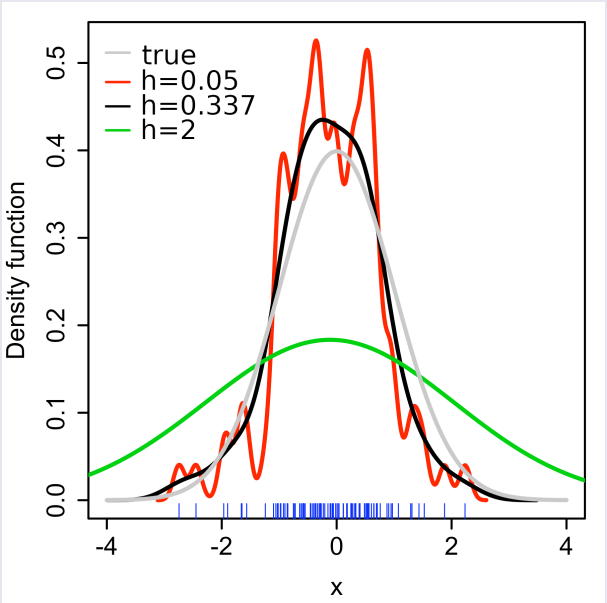
each with bandwidth (width) h_i

Optimal bandwidth

- Too narrow: noisy approximation
- Too wide: loose fine structure
- In principle found by minimising risk function $R(\hat{p}, p) = \int \left(\hat{p}(x) - p(x)\right)^2 dx$
- For Gaussian densities:

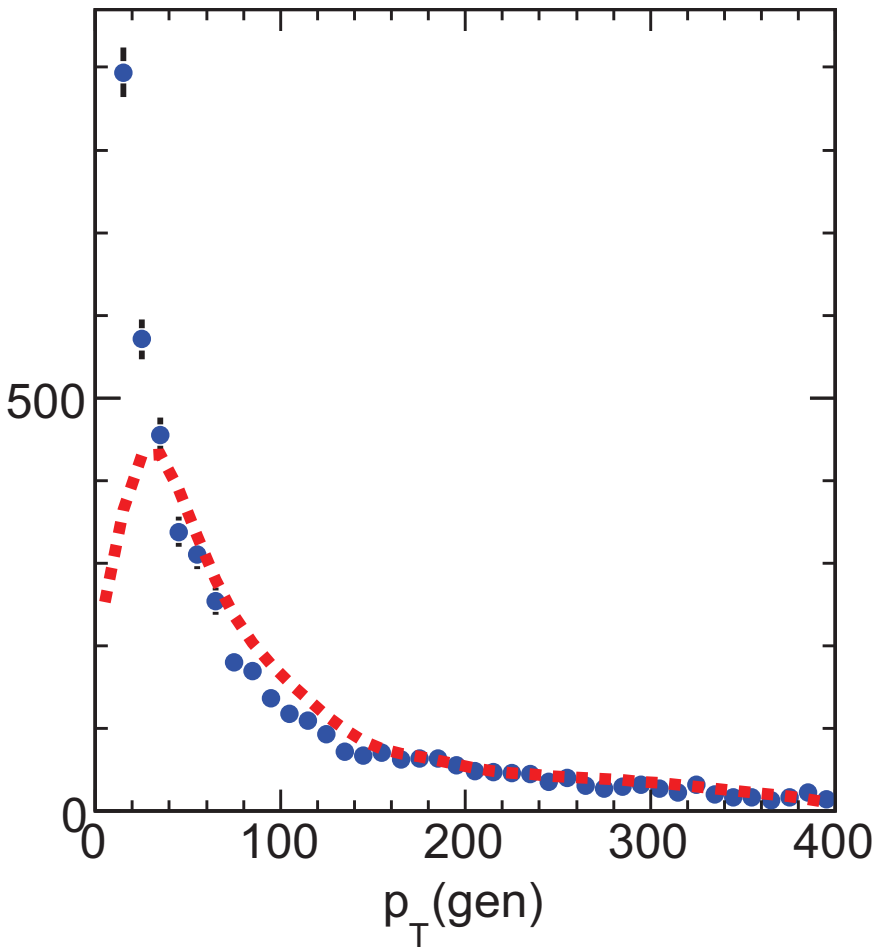
$$h = \sigma \left(\frac{4}{(n+2)N}\right)^{1/(n+4)}$$

- Far from optimal for non-Gaussian densities

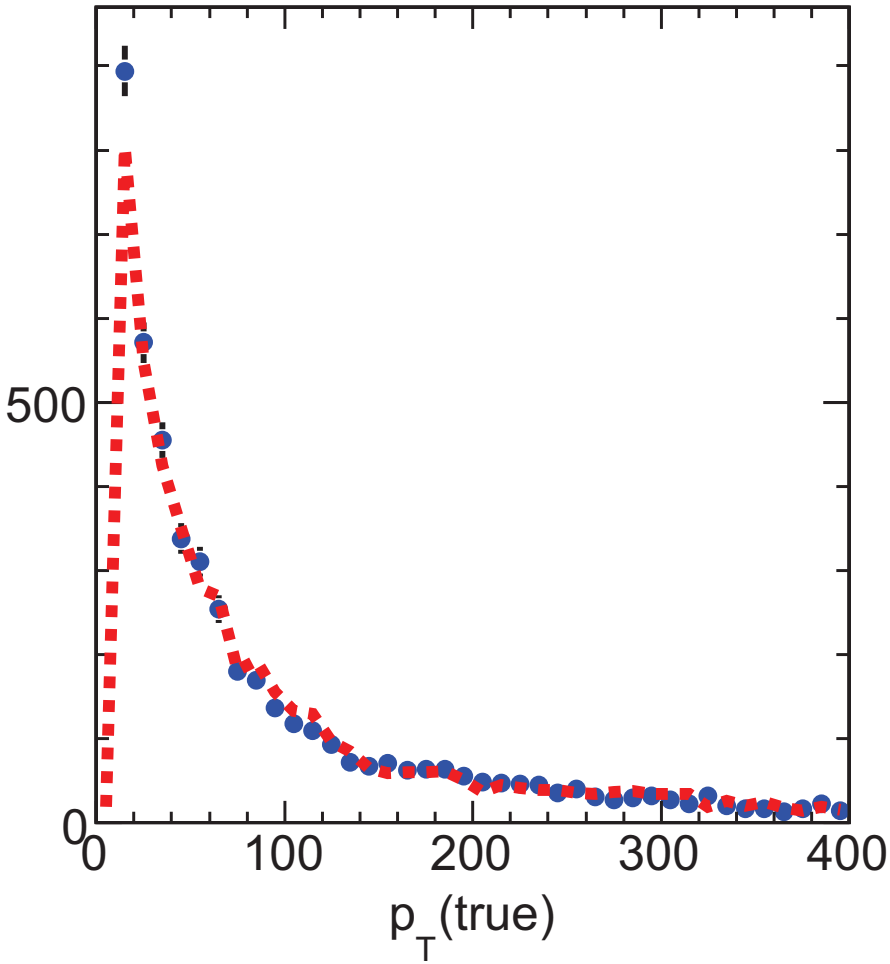


Kernel density estimation (KDE): example

with Gaussian optimal bandwidth



with optimised bandwidth



Kernel density estimation (KDE)

Why does it work?

- When $N \to \infty$:

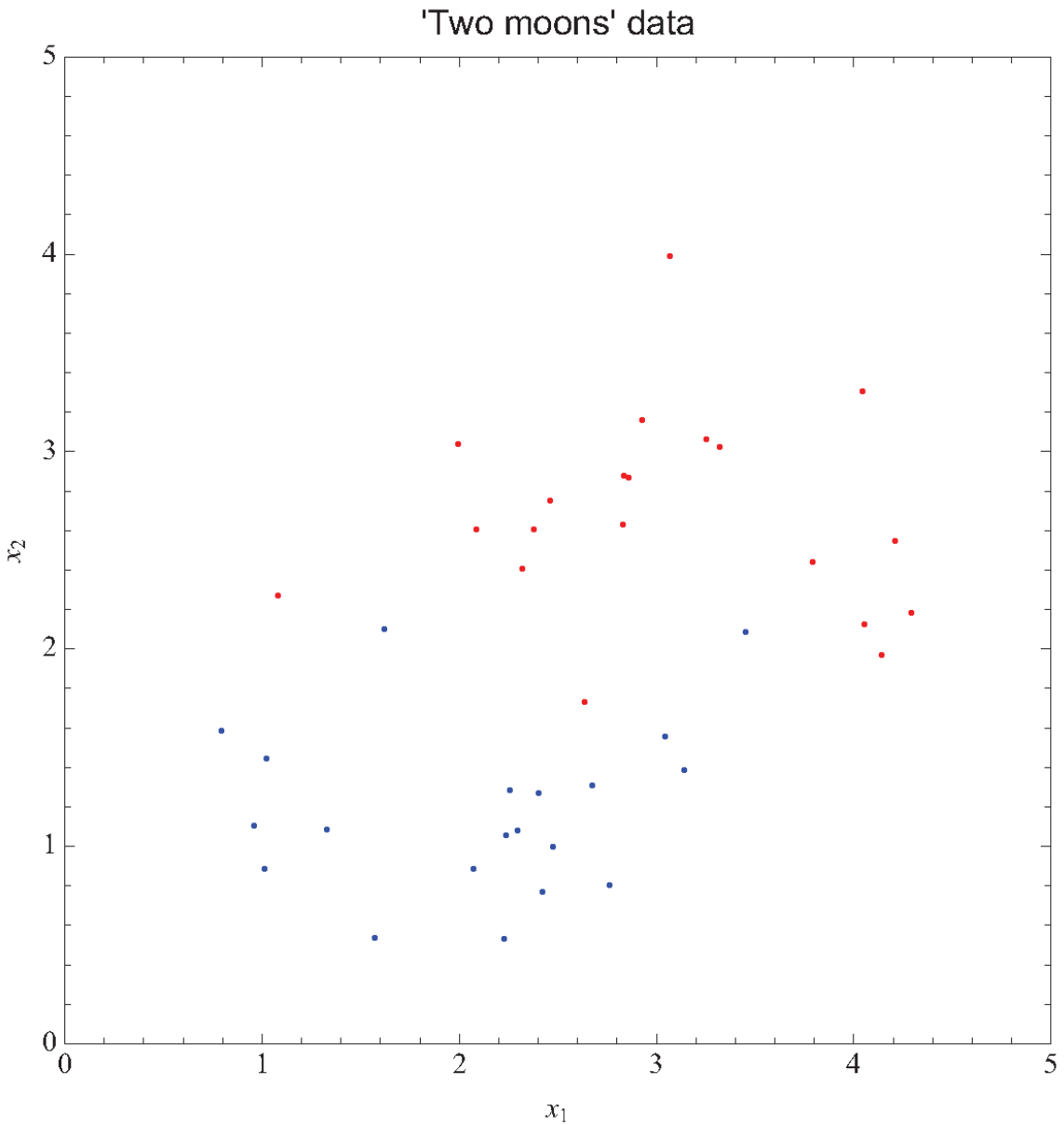
$$\hat{p}(x) = \int K(x, \mu) p(\mu) d\mu$$

- $p(\mu)$: true density of x
- Kernel bandwidth getting smaller with N, so when $N \to \infty$, $K(x, \mu) \to \delta^n(x - \mu)$ and $\hat{p}(x) = p(x)$
- KDE gives consistent estimate of probability density $p(x)$

Limitations

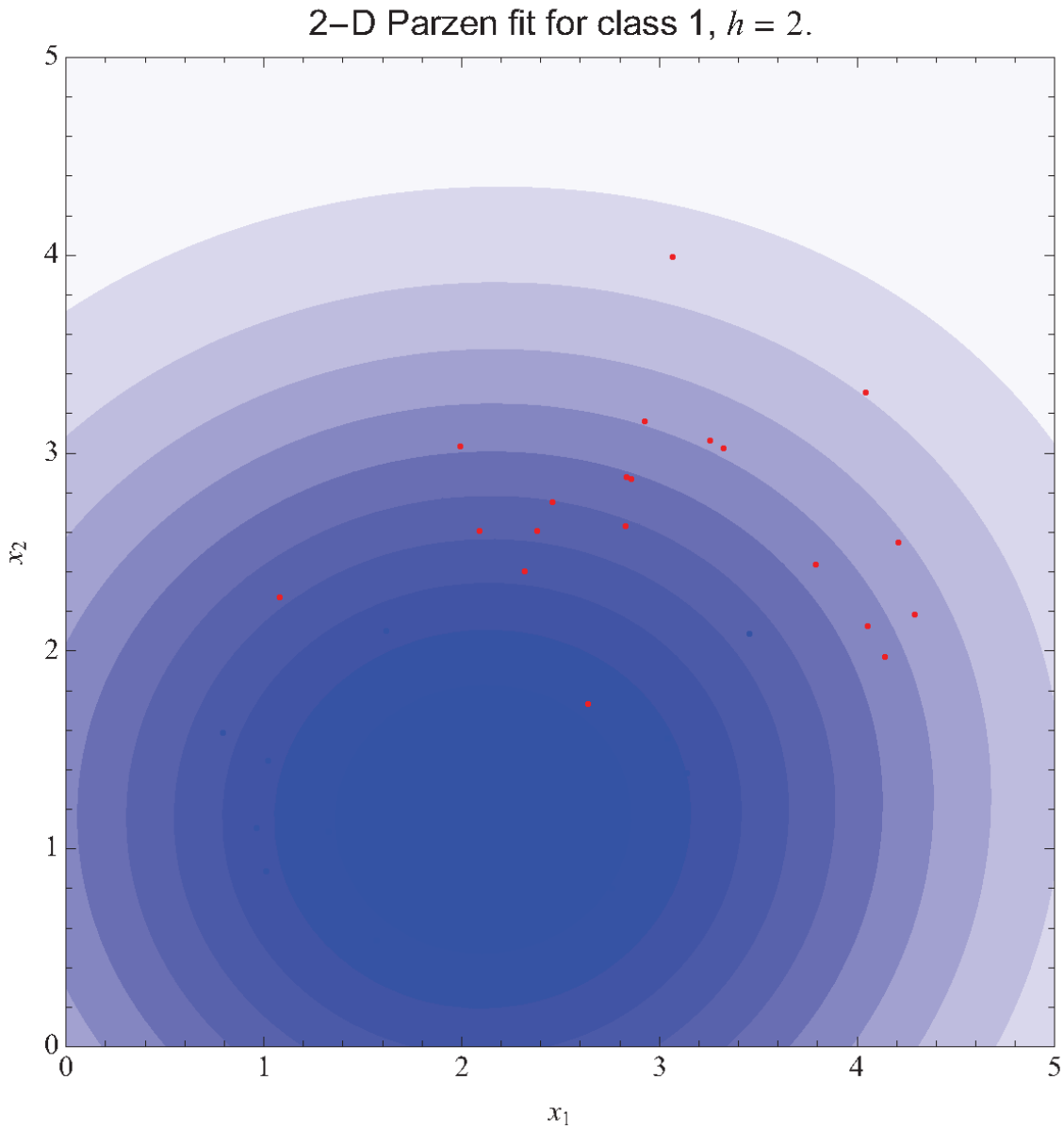
- Choice of bandwidth non-trivial
- Difficult to model sharp structures (e.g. boundaries)
- Kernels too far apart in regions of low point density
- (both can be mitigated with adaptive bandwidth choice)
- Requires evaluation of N n-dimensional kernels

Kernel density estimation (KDE)



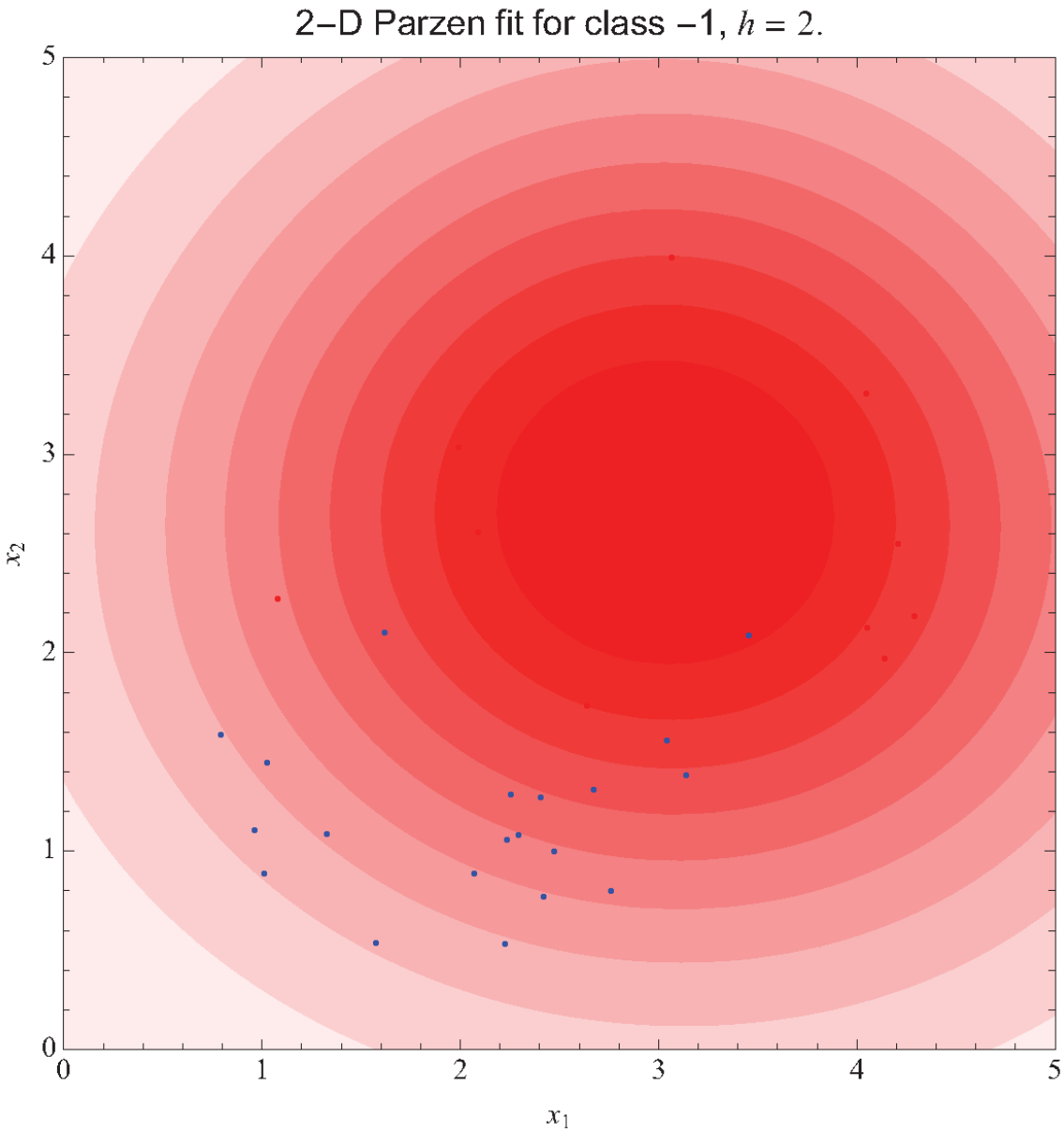
©Balàzs Kégl

Kernel density estimation (KDE)



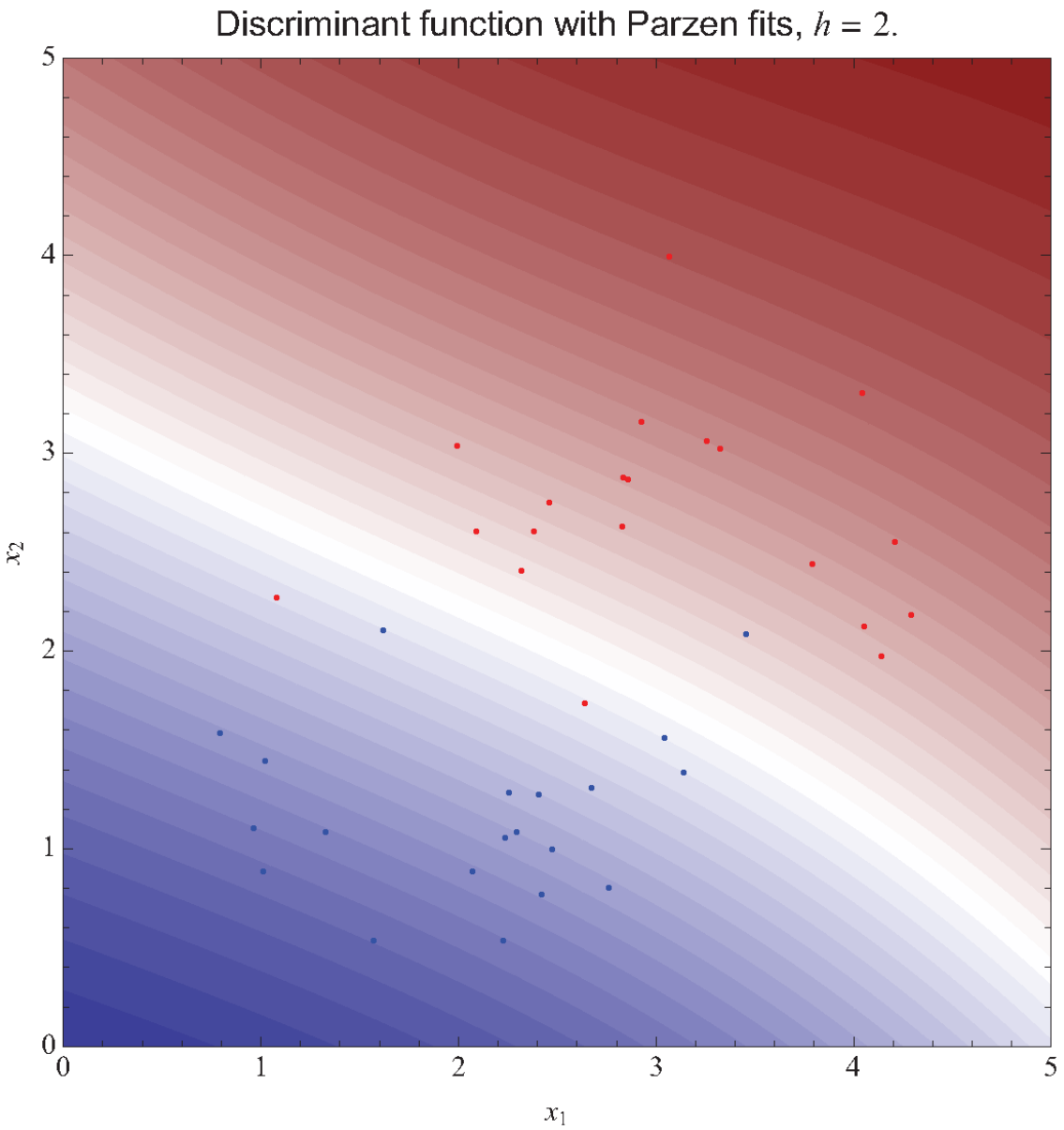
©Balàzs Kégl

Kernel density estimation (KDE)



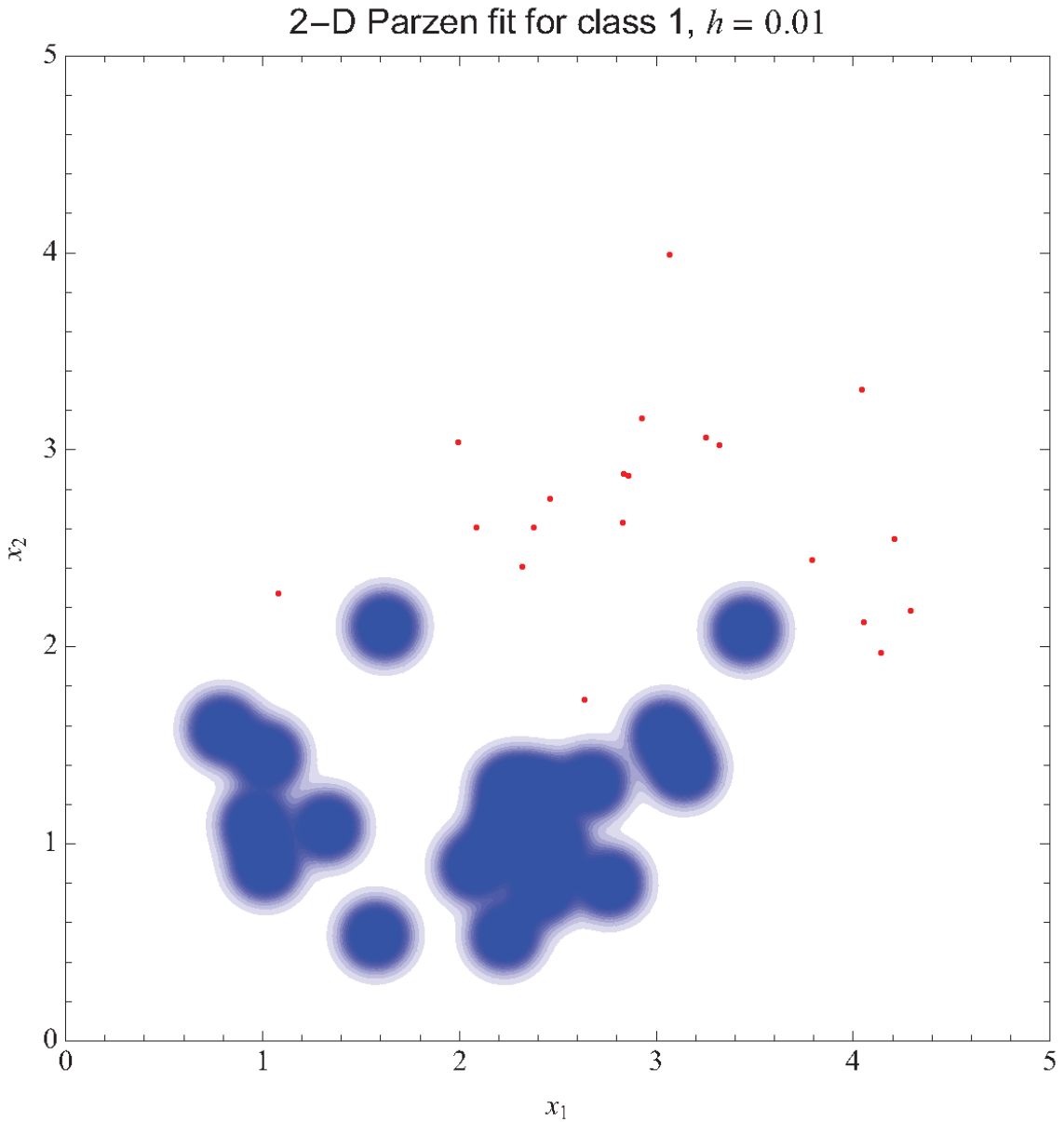
©Balàzs Kégl

Kernel density estimation (KDE)



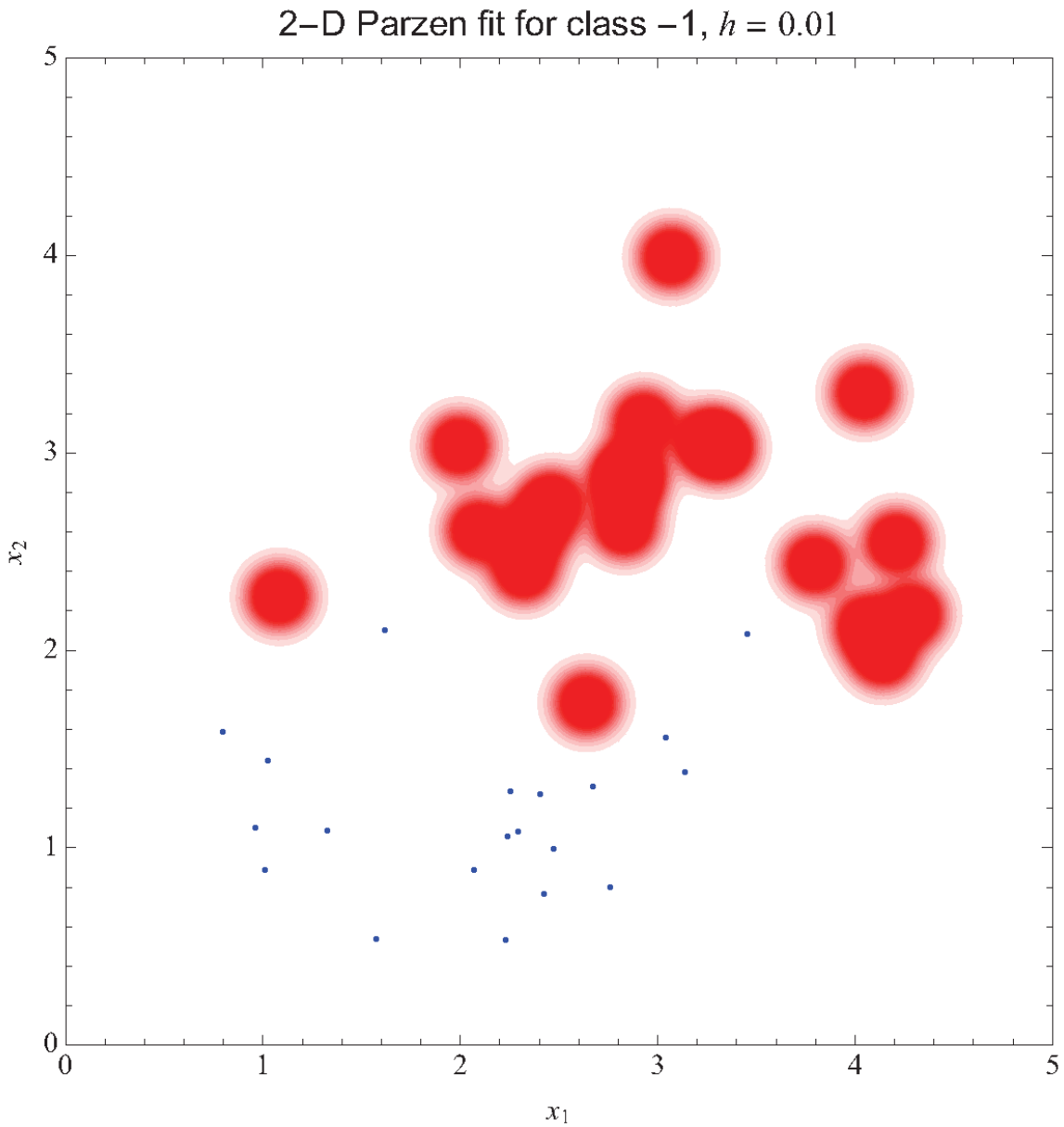
©Balàzs Kégl

Kernel density estimation (KDE)



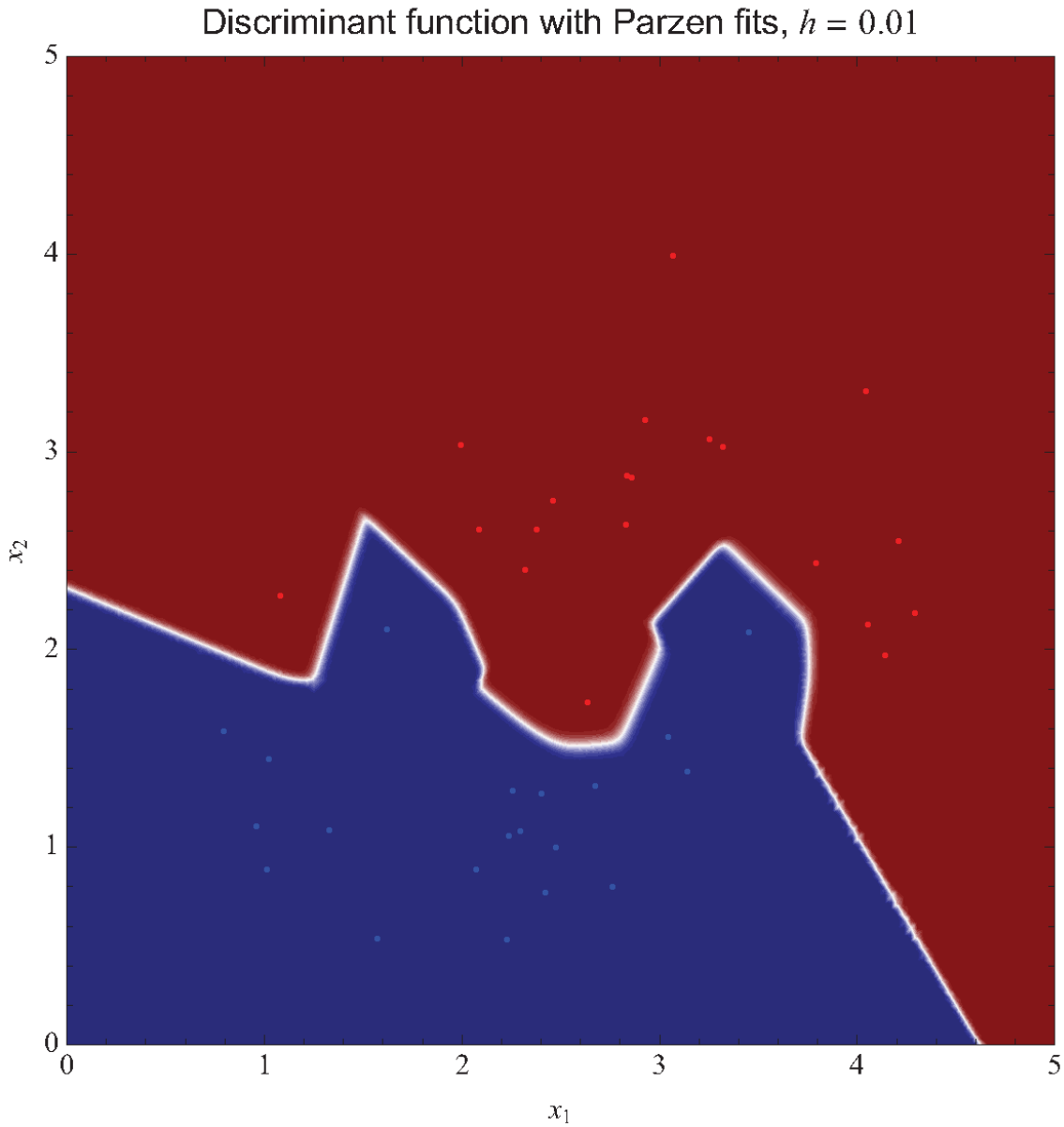
©Balàzs Kégl

Kernel density estimation (KDE)



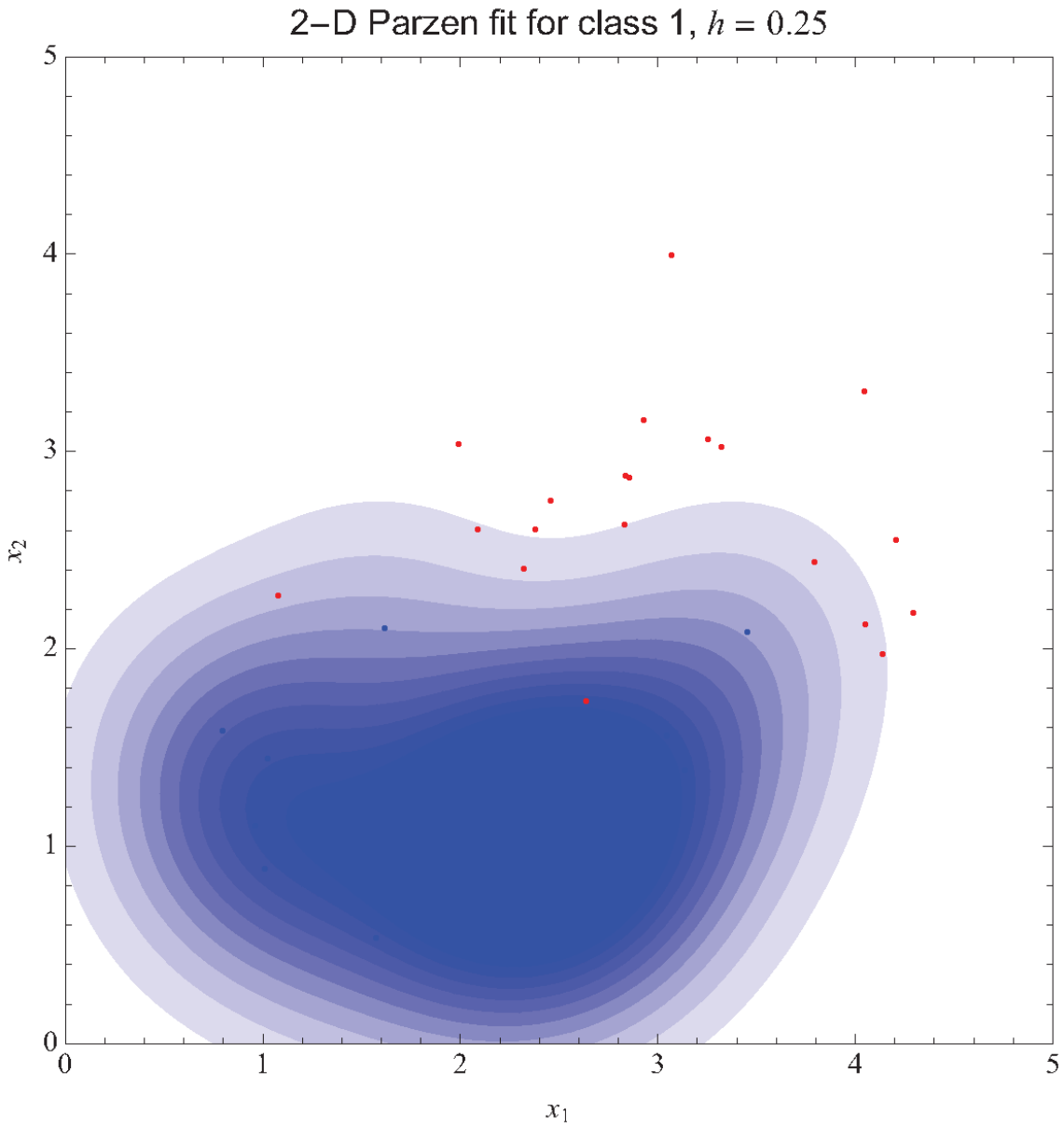
©Balàzs Kégl

Kernel density estimation (KDE)



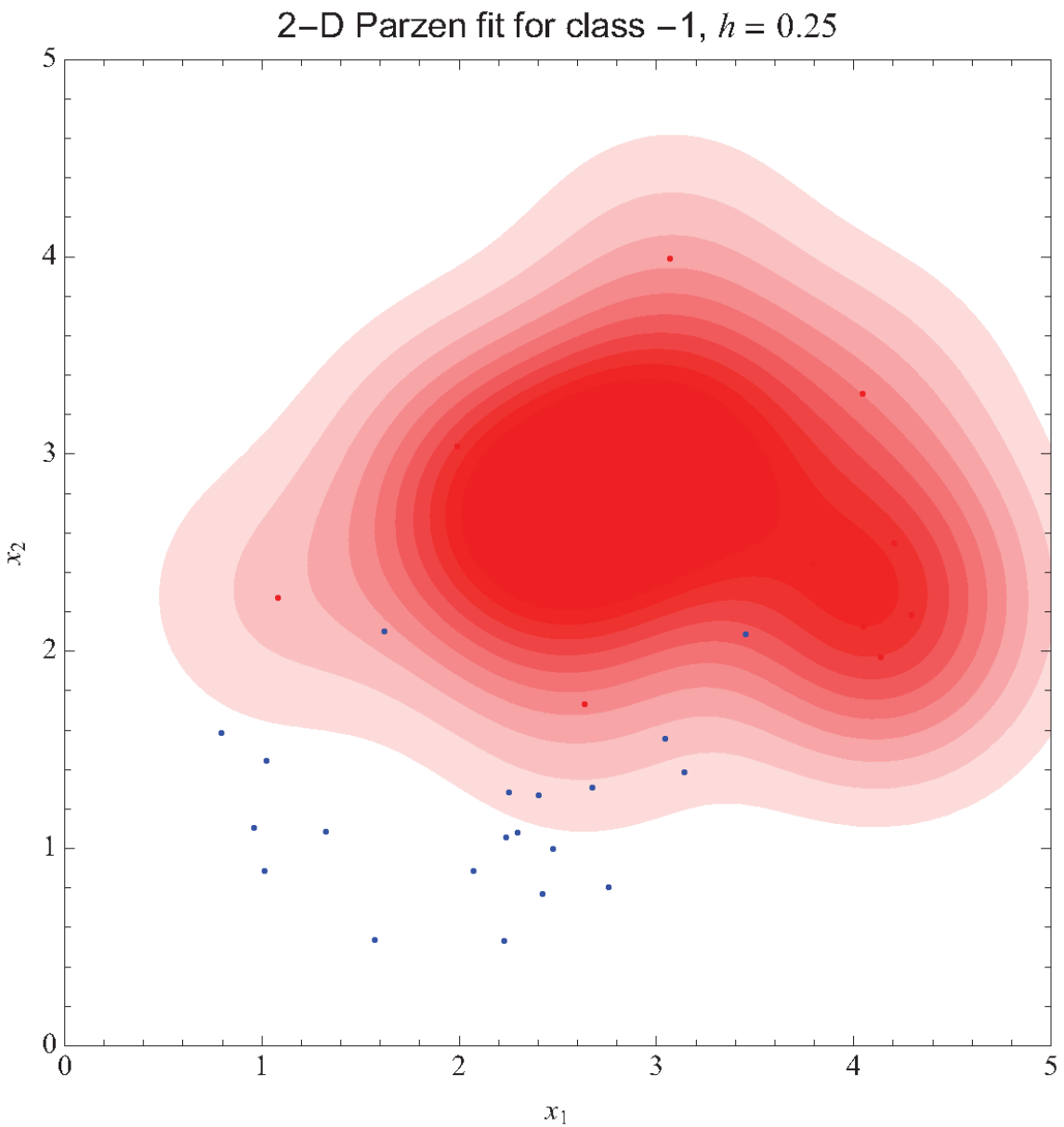
©Balàzs Kégl

Kernel density estimation (KDE)



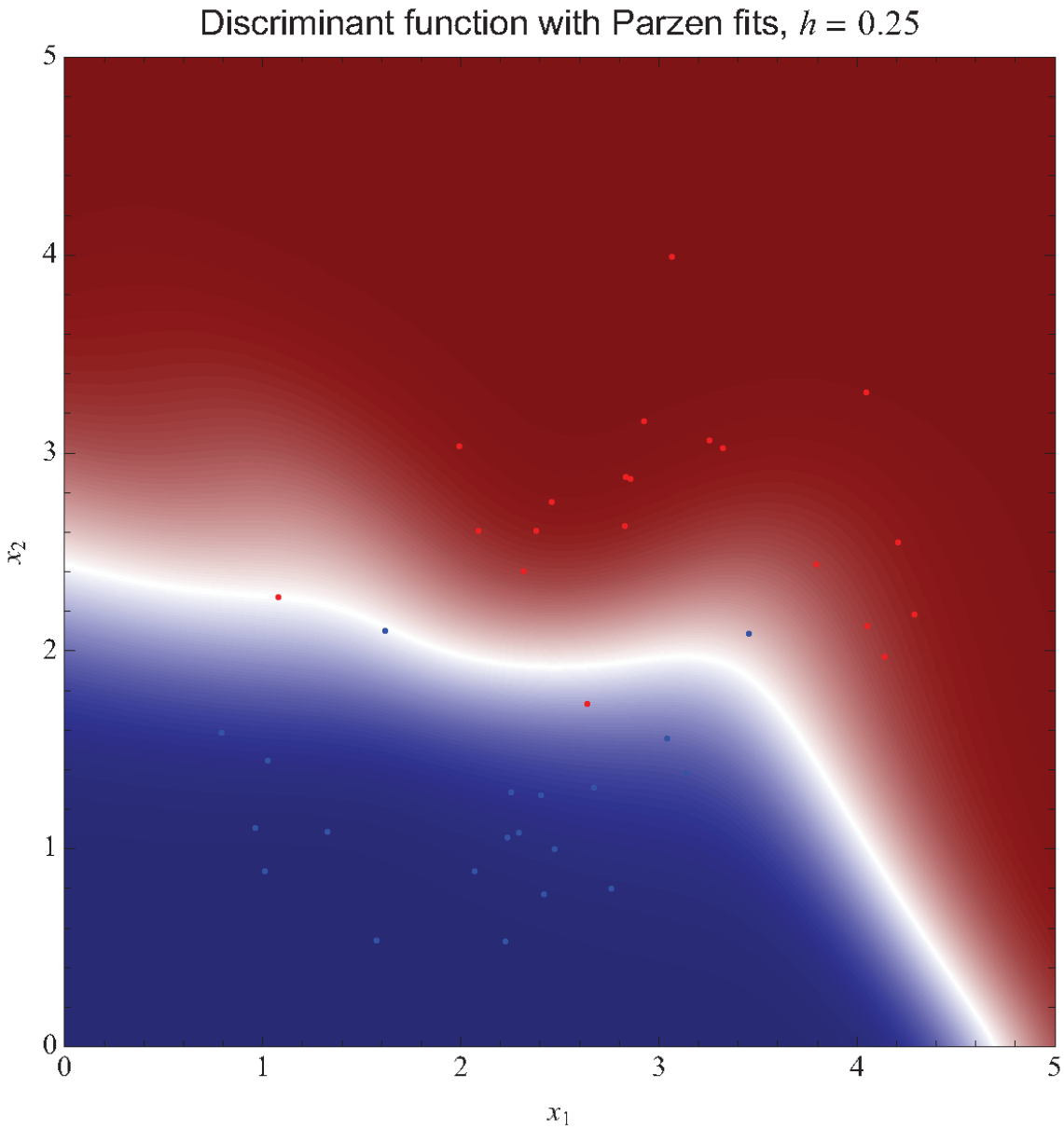
©Balàzs Kégl

Kernel density estimation (KDE)



©Balàzs Kégl

Kernel density estimation (KDE)

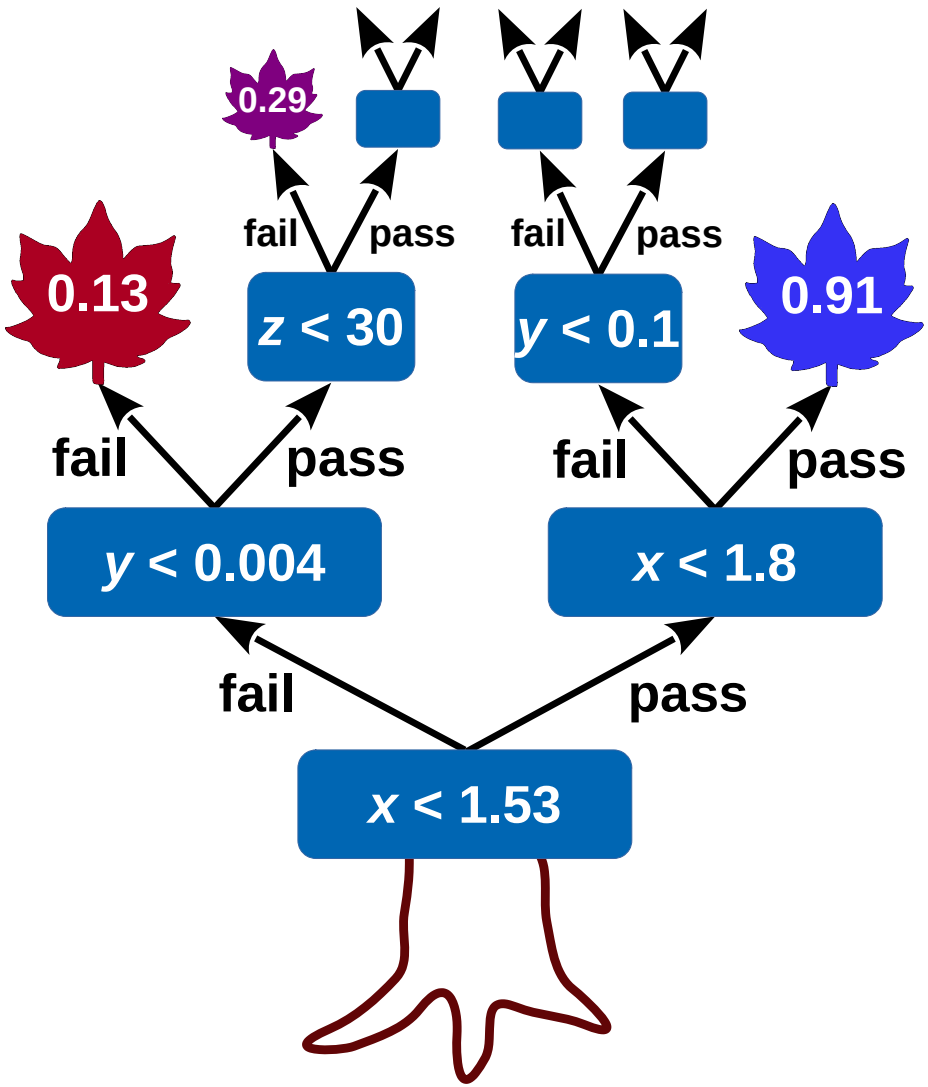


©Balàzs Kégl

KDE: choice of bandwidth

©Balàzs Kégl

Decision trees



- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Kernel density estimation
- Decision trees
  - Algorithm
  - Splitting a node
  - Variable selection
  - Limitations
- Boosted decision trees
- Neural networks
- Deep neural networks

Introduction

Decision tree origin

- Machine-learning technique, widely used in social sciences.
  Originally data mining/pattern recognition, then medical diagnosis, insurance/loan screening, etc.

L. Breiman *et al.*, "Classification and Regression Trees" (1984)

Basic principle

- Extend cut-based selection
  - many (most?) events do not have *all* characteristics of signal or background
  - try not to rule out events failing a particular criterion
- Keep events rejected by one criterion and see whether other criteria could help classify them properly

Binary trees

- Trees can be built with branches splitting into many sub-branches
- In this lecture: mostly binary trees

Tree building algorithm

Start with all events (signal and background) = first (root) node

- sort all events by each variable
- for each variable, find splitting value with best separation between two children
  - mostly signal in one child
  - mostly background in the other
- select variable and splitting value with best separation, produce two branches (nodes)
  - events failing criterion on one side
  - events passing it on the other

Keep splitting

- Now have two new nodes. Repeat algorithm recursively on each node
- Can reuse the same variable
- Iterate until stopping criterion is reached (min leaf size, max tree depth, insufficient improvement, perfect classification, etc.)
- Splitting stops: terminal node = leaf

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_{T} of leading jet, top mass M_t and scalar sum of p_{T}'s of all objects in the event H_{T}

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_{T} of leading jet, top mass M_t and scalar sum of p_{T}'s of all objects in the event H_{T}
  - sort all events by each variable:
    - $p_{\mathrm{T}}^{s_1} \leq p_{\mathrm{T}}^{b_{34}} \leq \cdots \leq p_{\mathrm{T}}^{b_2} \leq p_{\mathrm{T}}^{s_{12}}$
    - $H_{\mathrm{T}}^{b_5} \leq H_{\mathrm{T}}^{b_3} \leq \cdots \leq H_{\mathrm{T}}^{s_{67}} \leq H_{\mathrm{T}}^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$

Algorithm example

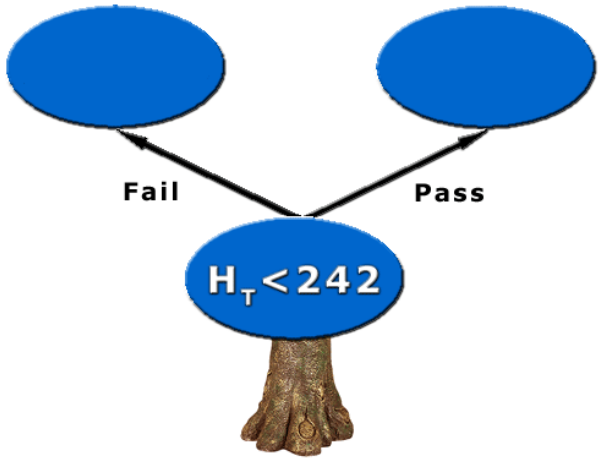
- Consider signal (s_i) and background (b_j) events described by 3 variables: p_{T} of leading jet, top mass M_t and scalar sum of p_{T}'s of all objects in the event H_{T}
  - sort all events by each variable:
    - $p_{\mathrm{T}}^{s_1} \leq p_{\mathrm{T}}^{b_{34}} \leq \cdots \leq p_{\mathrm{T}}^{b_2} \leq p_{\mathrm{T}}^{s_{12}}$
    - $H_{\mathrm{T}}^{b_5} \leq H_{\mathrm{T}}^{b_3} \leq \cdots \leq H_{\mathrm{T}}^{s_{67}} \leq H_{\mathrm{T}}^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_{\mathrm{T}} < 56$ GeV, separation $= 3$
    - $H_{\mathrm{T}} < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$

Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_{T} of leading jet, top mass M_t and scalar sum of p_{T}'s of all objects in the event H_{T}
  - sort all events by each variable:
    - $p_{\mathrm{T}}^{s_1} \leq p_{\mathrm{T}}^{b_{34}} \leq \cdots \leq p_{\mathrm{T}}^{b_2} \leq p_{\mathrm{T}}^{s_{12}}$
    - $H_{\mathrm{T}}^{b_5} \leq H_{\mathrm{T}}^{b_3} \leq \cdots \leq H_{\mathrm{T}}^{s_{67}} \leq H_{\mathrm{T}}^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_{\mathrm{T}} < 56$ GeV, separation $= 3$
    - $H_{\mathrm{T}} < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$

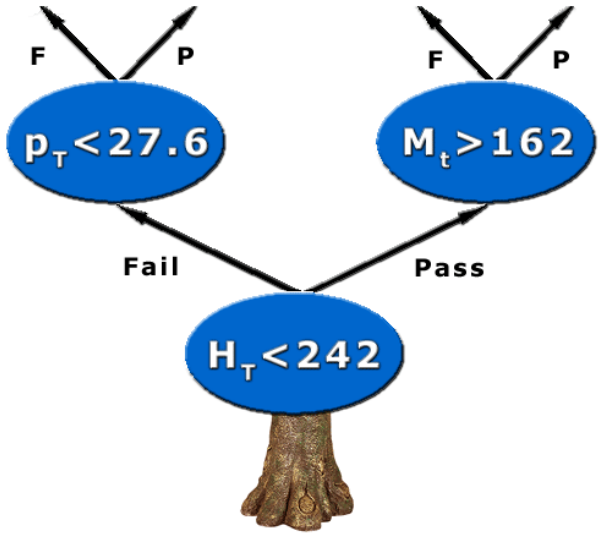
Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_{T} of leading jet, top mass M_t and scalar sum of p_{T}'s of all objects in the event H_{T}
  - sort all events by each variable:
    - $p_{\mathrm{T}}^{s_1} \leq p_{\mathrm{T}}^{b_{34}} \leq \cdots \leq p_{\mathrm{T}}^{b_2} \leq p_{\mathrm{T}}^{s_{12}}$
    - $H_{\mathrm{T}}^{b_5} \leq H_{\mathrm{T}}^{b_3} \leq \cdots \leq H_{\mathrm{T}}^{s_{67}} \leq H_{\mathrm{T}}^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_{\mathrm{T}} < 56$ GeV, separation $= 3$
    - $H_{\mathrm{T}} < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$
  - split events in two branches: pass or fail $H_{\mathrm{T}} < 242$ GeV



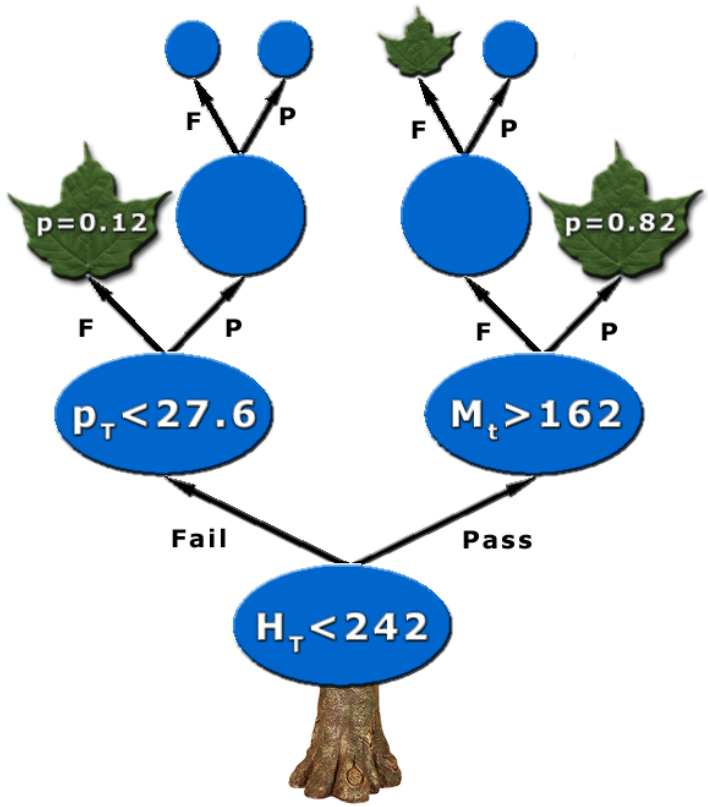
Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$
  - split events in two branches: pass or fail $H_T < 242$ GeV
- Repeat recursively on each node



Algorithm example

- Consider signal (s_i) and background (b_j) events described by 3 variables: p_T of leading jet, top mass M_t and scalar sum of p_T's of all objects in the event H_T
  - sort all events by each variable:
    - $p_T^{s_1} \leq p_T^{b_{34}} \leq \cdots \leq p_T^{b_2} \leq p_T^{s_{12}}$
    - $H_T^{b_5} \leq H_T^{b_3} \leq \cdots \leq H_T^{s_{67}} \leq H_T^{s_{43}}$
    - $M_t^{b_6} \leq M_t^{s_8} \leq \cdots \leq M_t^{s_{12}} \leq M_t^{b_9}$
  - best split (arbitrary unit):
    - $p_T < 56$ GeV, separation $= 3$
    - $H_T < 242$ GeV, separation $= 5$
    - $M_t < 105$ GeV, separation $= 0.7$
  - split events in two branches: pass or fail $H_T < 242$ GeV

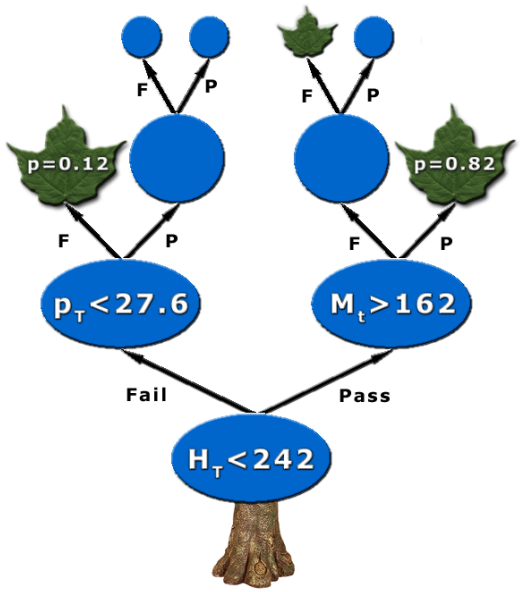


- Repeat recursively on each node
- Splitting stops: e.g. events with $H_T < 242$ GeV and $M_t > 162$ GeV are signal like ($p = 0.82$)

Decision tree output

Run event through tree

- Start from root node
- Apply first best cut
- Go to left or right child node
- Apply best cut for this node
- ...Keep going until...
- Event ends up in leaf



DT Output

- Purity $\left(\frac{s}{s+b}\right.$, with weighted events$\left.\right)$ of leaf, close to 1 for signal and 0 for background
- or binary answer (discriminant function $+1$ for signal, -1 or 0 for background) based on purity above/below specified value (e.g. $\frac{1}{2}$) in leaf
- E.g. events with $H_T < 242$ GeV and $M_t > 162$ GeV have a DT output of 0.82 or $+1$

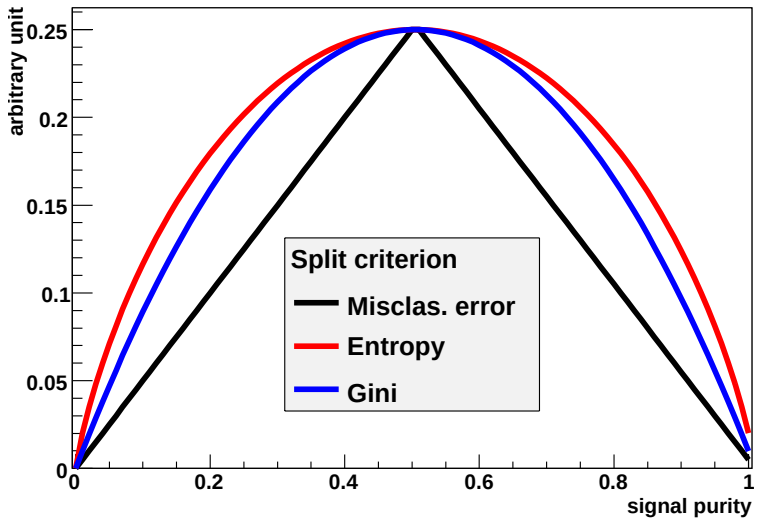
Splitting a node

Optimal split: figure of merit

- Decrease of impurity for split s of node t into children t_P and t_F (goodness of split): $\Delta i(s,t) = i(t) - p_P \cdot i(t_P) - p_F \cdot i(t_F)$
- Aim: find split s^* such that $\Delta i(s^*,t) = \max_{s \in \{\text{splits}\}} \Delta i(s,t)$
- Maximising $\Delta i(s,t) \equiv$ minimising overall tree impurity

Common impurity functions

- misclassification error $= 1 - max(p, 1-p)$
- (cross) entropy $= -\sum_{i=s,b} p_i \log p_i$
- Gini index (details in ▸ backup)



- Also cross section $(-\frac{s^2}{s+b})$ and excess significance $(-\frac{s^2}{b})$

Variable selection I

Reminder

- Need model giving good description of data

Variable selection I

Reminder

- Need model giving good description of data

Playing with variables

- Number of variables:
  - not affected too much by "curse of dimensionality"
  - CPU consumption scales as $nN \log N$ with n variables and N training events
- Variable order does not matter: all variables treated equal
- Order of training events is irrelevant (batch training)
- Irrelevant variables:
  - no discriminative power $\Rightarrow$ not used
  - only costs a little CPU time, no added noise
- Can use continuous and discrete variables, simultaneously

Variable selection II

Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them (same order $\Rightarrow$ same DT):
  - convert MeV $\rightarrow$ GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)

$\Rightarrow$ Some immunity against outliers

Variable selection II

Transforming input variables

- Completely insensitive to replacement of any subset of input variables by (possibly different) arbitrary strictly monotone functions of them (same order $\Rightarrow$ same DT):
  - convert MeV $\rightarrow$ GeV
  - no need to make all variables fit in the same range
  - no need to regularise variables (e.g. taking the log)

$\Rightarrow$ Some immunity against outliers

Note about actual implementation

- The above is strictly true only if testing all possible cut values
- If there is some computational optimisation (e.g., check only 20 possible cuts on each variable), it may not work anymore

Variable selection III

Variable ranking (mean decrease impurity MDI)

- Ranking of x_i: add up decrease of impurity each time x_i is used
- Largest decrease of impurity = best variable

Shortcoming: masking of variables

- x_j may be just a little worse than x_i but will never be picked
- x_j is ranked as irrelevant
- But remove x_i and x_j becomes very relevant
  $\Rightarrow$ careful with interpreting ranking (specific to training)

Permutation importance (mean decrease accuracy MDA)

- Applicable to any already trained classifier
- Randomly shuffle each variable in turn and measure decrease of performance
- Important variable $\Rightarrow$ big loss of performance
- Can also be performed on validation sample
- Beware of correlations [Breiman 2001]

Variable selection IV

Choosing variables

- Usually try to have as few variables as possible
- But difficult: correlations, possibly large number to consider, large phase space with different properties in different regions
- *Brute force:* with n variables train all n, $n-1$, etc. combinations, pick best
- *Backward elimination:* train with n variables, then train all $n-1$ variables trees and pick best one; now train all $n-2$ variables trees starting from the $n-1$ variable list; etc. Pick optimal cost-complexity tree.
- *Forward greedy selection:* start with $k=1$ variable, then train all $k+1$ variables trees and pick the best; move to $k+2$ variables; etc.

Tree instability: training sample composition

- Small changes in sample can lead to very different tree structures (high variance)
- Performance on testing events may be as good, or not
- Not optimal to understand data from DT rules
- Does not give confidence in result:
  - DT output distribution discrete by nature
  - granularity related to tree complexity
  - tendency to have spikes at certain purity values (or just two delta functions at ± 1 if not using purity)

Pruning a tree

Why prune a tree?

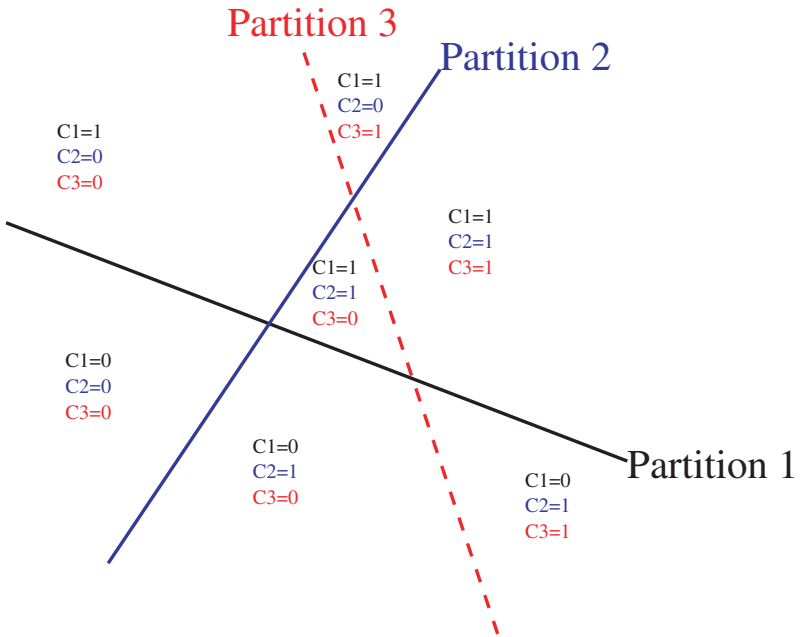
- Possible to get a perfect classifier on training events
- Mathematically misclassification error can be made as little as wanted
- E.g. tree with one class only per leaf (down to 1 event per leaf if necessary)
- Training error is zero
- But run new independent events through tree (testing or validation sample): misclassification is probably > 0, overtraining
- Pruning: eliminate subtrees (branches) that seem too specific to training sample:
  - a node and all its descendants turn into a leaf

Pruning algorithms (details in ▸ backup)

- Pre-pruning (early stopping condition like min leaf size, max depth)
- Expected error pruning (based on statistical error estimate)
- Cost-complexity pruning (penalise "complex" trees with many nodes/leaves)

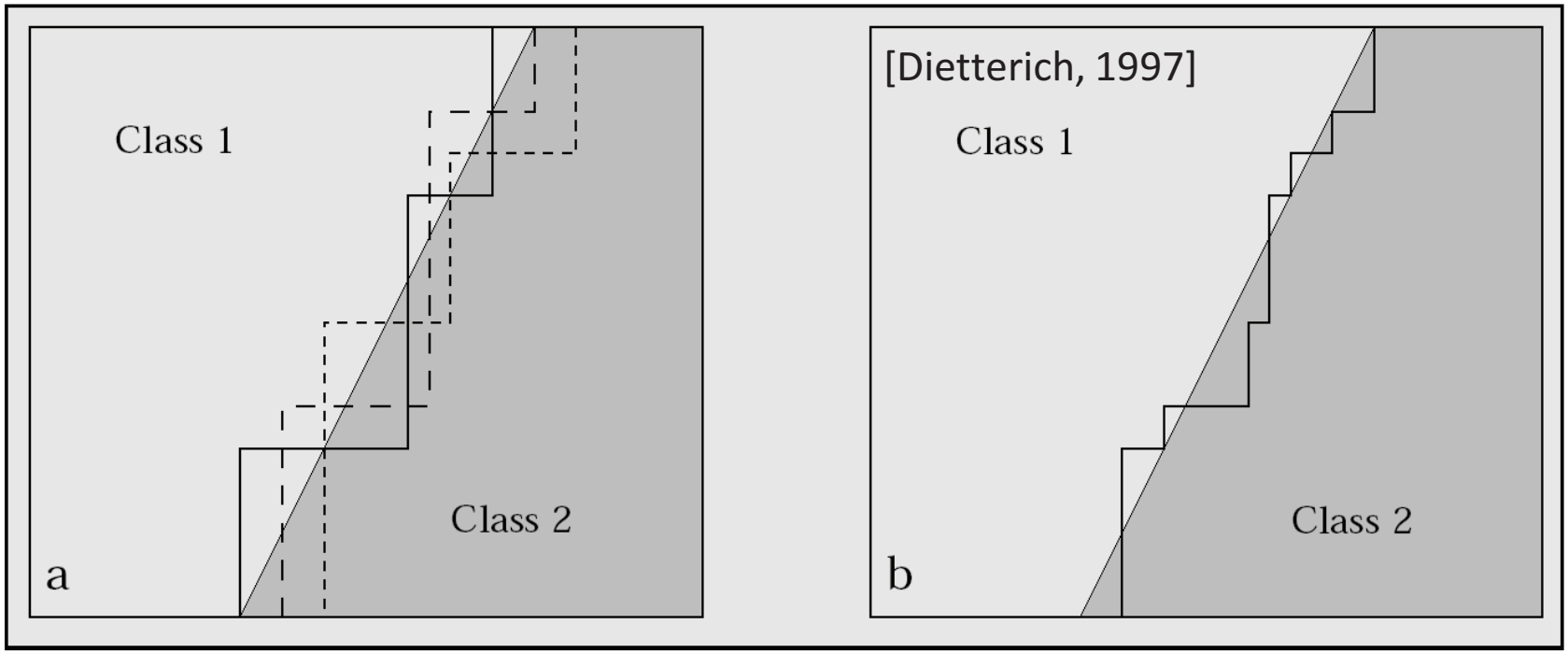
Tree (in)stability: distributed representation

- One tree:
  - one information about event (one leaf)
  - cannot really generalise to variations not covered in training set (at most as many leaves as input size)
- Many trees:
  - distributed representation: number of intersections of leaves exponential in number of trees
  - many leaves contain the event ⇒ richer description of input pattern



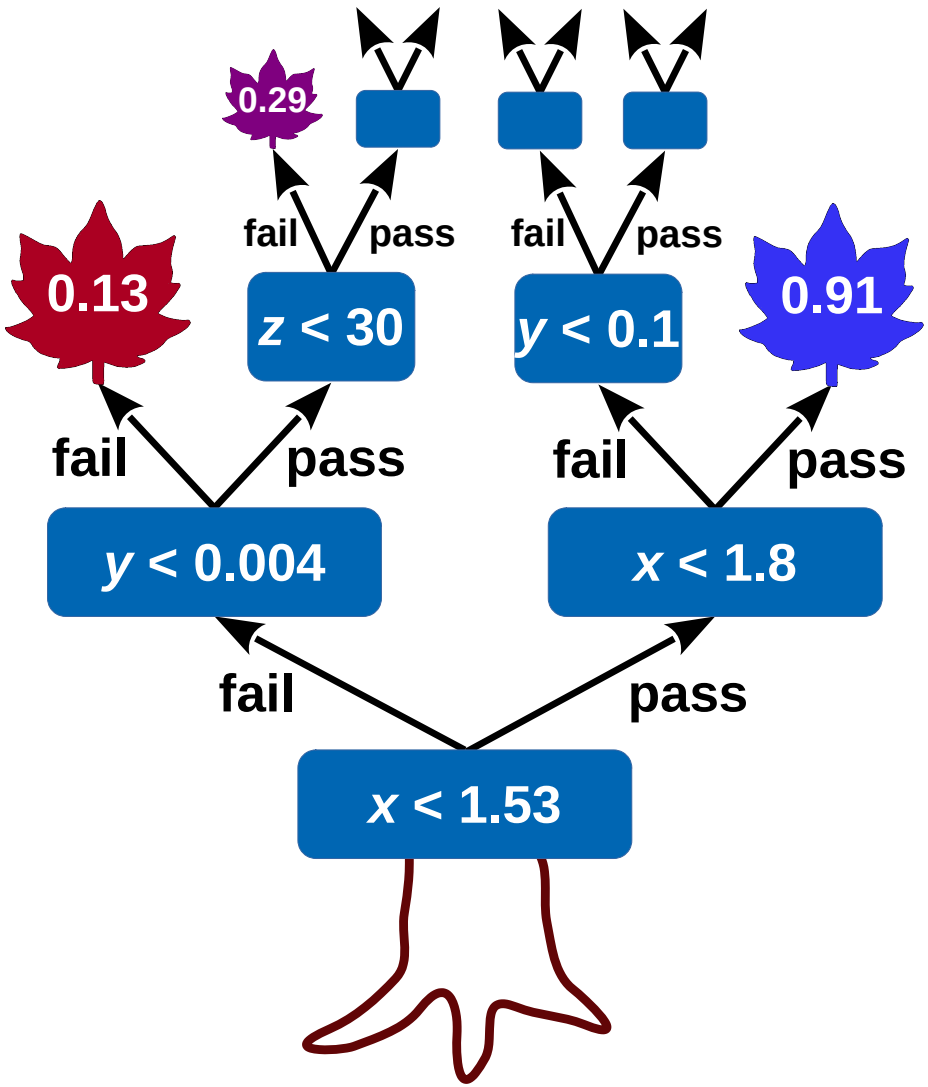
Tree (in)stability solution: averaging

- Build several trees and average the output



- K-fold cross-validation (good for small samples)
  - divide training sample $\mathcal{L}$ in K subsets of equal size: $\mathcal{L} = \bigcup_{k=1..K} \mathcal{L}_k$
  - Train tree T_k on $\mathcal{L} - \mathcal{L}_k$, test on $\mathcal{L}_k$
  - DT output $= \frac{1}{K} \sum_{k=1..K} T_k$
- Bagging, boosting, random forests: ensemble learning

Boosted decision trees



- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Kernel density estimation
- Decision trees
- Boosted decision trees
  - Introduction
  - AdaBoost
  - Clues to boosting performance
  - Gradient boosting
  - Performance examples
  - BDTs in real physics cases
- Neural networks
- Deep neural networks

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: MajorityVote(T_1, T_2, T_3)

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: MajorityVote(T_1, T_2, T_3)

Then

- Variation [Freund 1995]: boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1st functional model AdaBoost (1996)

Boosting: a brief history

First provable algorithm [Schapire 1990]

- Train classifier T_1 on N events
- Train T_2 on new N-sample, half of which misclassified by T_1
- Build T_3 on events where T_1 and T_2 disagree
- Boosted classifier: MajorityVote(T_1, T_2, T_3)

Then

- Variation [Freund 1995]: boost by majority (combining many learners with fixed error rate)
- Freund&Schapire joined forces: 1st functional model AdaBoost (1996)

When it really picked up in HEP

- MiniBooNe compared performance of different boosting algorithms and neural networks for particle ID [MiniBooNe 2005]
- D0 claimed first evidence for single top quark production [D0 2006]
- CDF copied 😉 (2008). Both used BDT for single top observation

What is boosting?

- General method, not limited to decision trees
- Hard to make a very good learner, but easy to make simple, error-prone ones (but still better than random guessing)
- Goal: combine such weak classifiers into a new more stable one, with smaller error

AdaBoost

- Introduced by Freund&Schapire in 1996
- Stands for *adaptive boosting*
- Learning procedure adjusts to training data to classify it better
- Many variations on the same theme for actual implementation
- Usually leads to better results than without boosting

AdaBoost algorithm

- Check which events of training sample $\mathbb{T}_k$ are misclassified by T_k:
  - $\mathbb{I}(X) = 1$ if X is true, 0 otherwise
  - for DT output in $\{\pm 1\}$: $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times T_k(x_i) \leq 0\big)$
  - or $\text{isMisclassified}_k(i) = \mathbb{I}\big(y_i \times (T_k(x_i) - 0.5) \leq 0\big)$ in purity convention
  - misclassification rate:

$$R(T_k) = \varepsilon_k = \frac{\sum_{i=1}^{N} w_i^k \times \text{isMisclassified}_k(i)}{\sum_{i=1}^{N} w_i^k}$$

- Derive tree weight $\alpha_k = \beta \times \ln((1 - \varepsilon_k)/\varepsilon_k)$
- Increase weight of misclassified events in $\mathbb{T}_k$ to create $\mathbb{T}_{k+1}$:

$$w_i^k \to w_i^{k+1} = w_i^k \times e^{\alpha_k}$$

- Train T_{k+1} on $\mathbb{T}_{k+1}$
- Boosted result of event i:

$$T(i) = \frac{1}{\sum_{k=1}^{N_{\text{tree}}} \alpha_k} \sum_{k=1}^{N_{\text{tree}}} \alpha_k T_k(i)$$

AdaBoost error rate

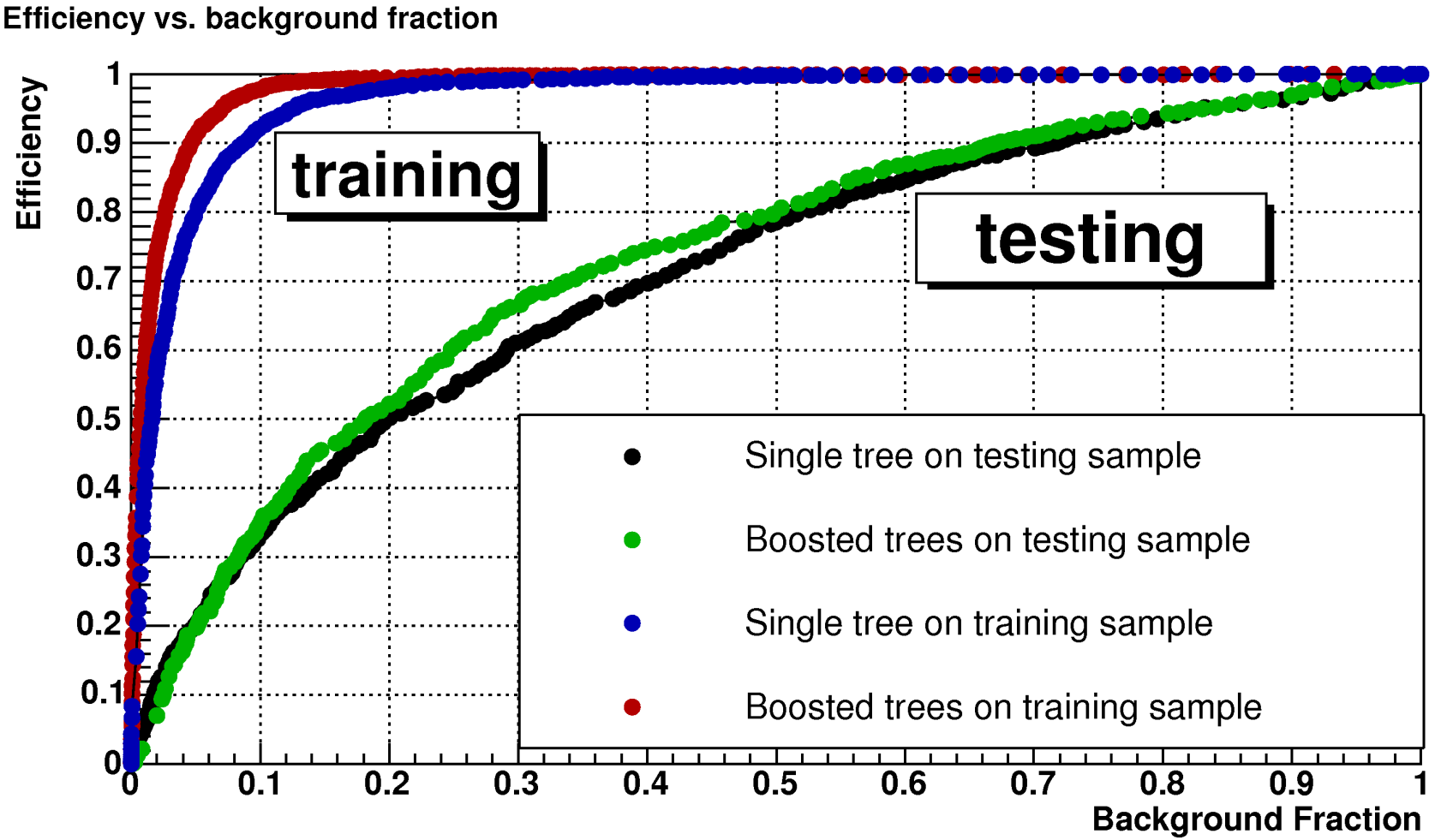
Misclassification rate ε on training sample

- Can be shown to be bound:
$$\varepsilon \leq \prod_{k=1}^{N_{tree}} 2\sqrt{\varepsilon_k(1-\varepsilon_k)}$$
- If each tree has $\varepsilon_k \neq 0.5$ (i.e. better than random guessing):
the error rate falls to zero for sufficiently large N_{tree}
- Corollary: training data is overfitted

Overtraining?

- Error rate on test sample may reach a minimum and then potentially rise. Stop boosting at the minimum.
- In principle AdaBoost *must* overfit training sample
- In many cases in literature, no loss of performance due to overtraining
  - may have to do with fact that successive trees get in general smaller and smaller weights
  - trees that lead to overtraining contribute very little to final DT output on validation sample

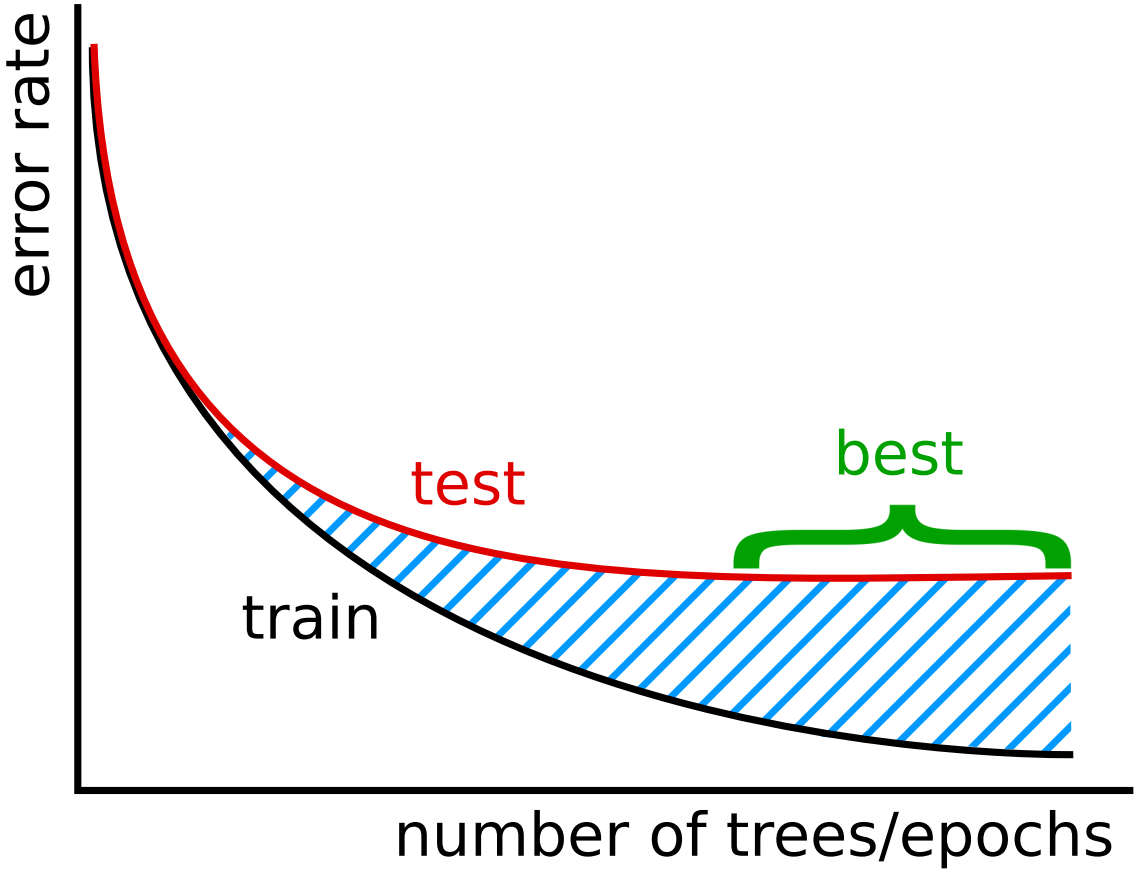
Training and generalisation error



- Clear overtraining, but still better performance after boosting

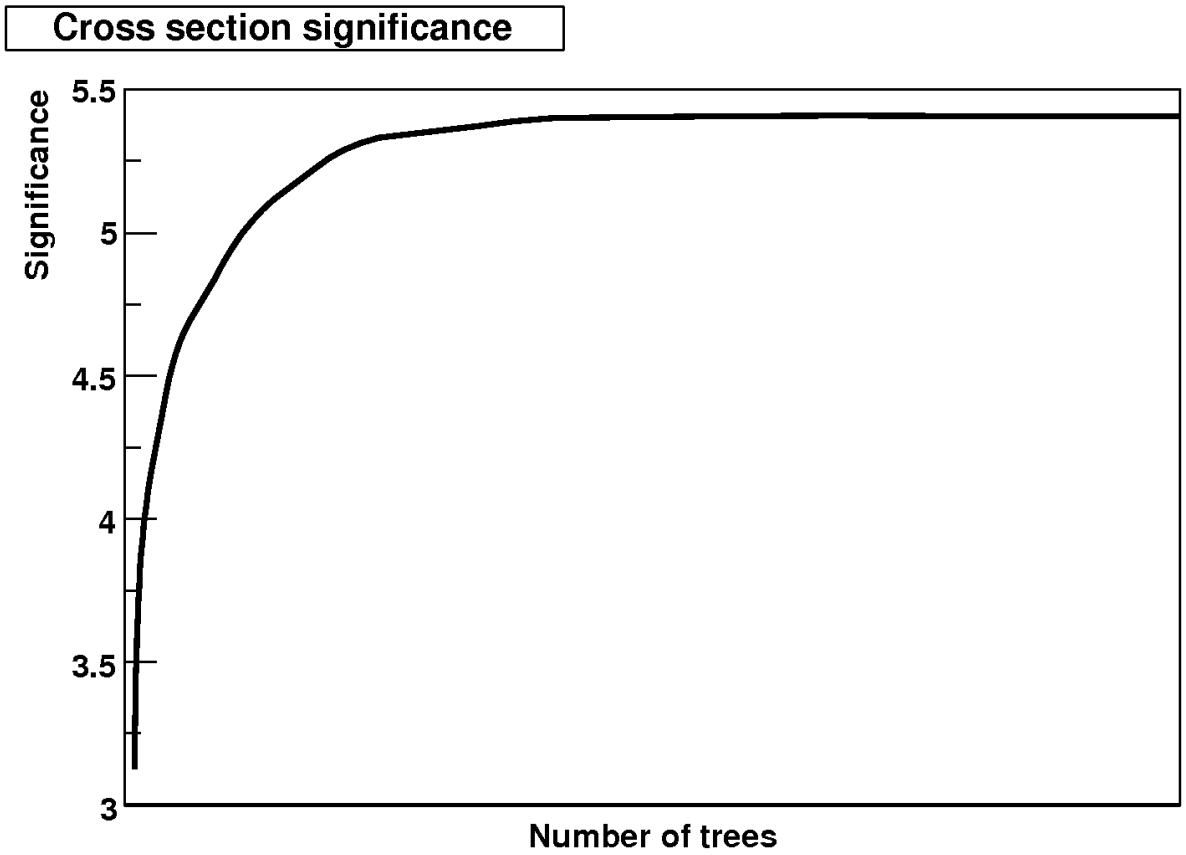
Overtraining estimation: good or bad?

- Typical situation for boosted decision trees



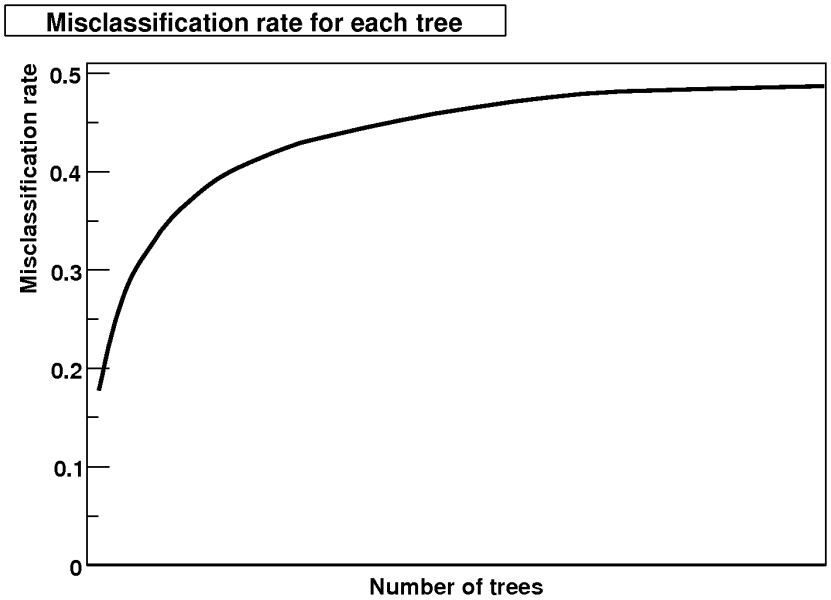
"bad" overtraining (overfitting) / "good" overtraining (still underfitting)

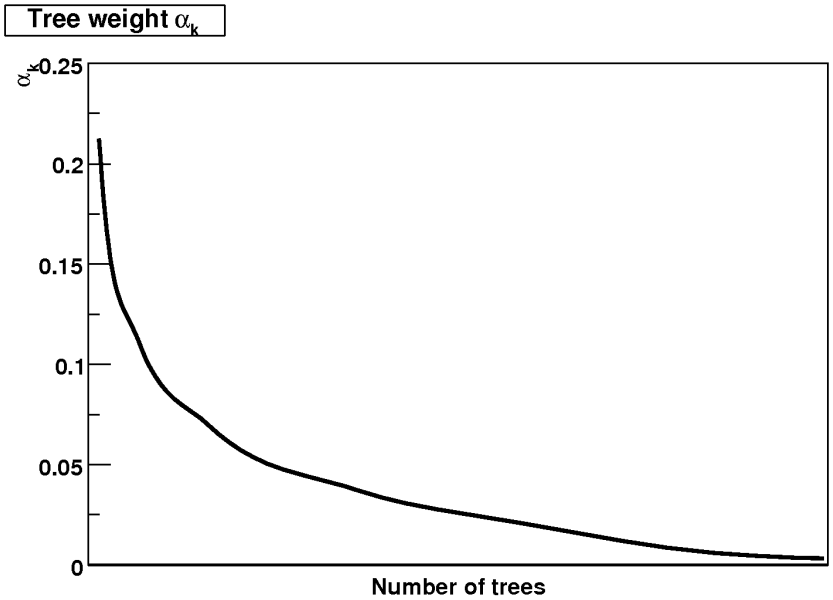
Cross section significance $(s/\sqrt{s+b})$



- More relevant than testing error
- Reaches plateau
- Afterwards, boosting does not hurt (just wasted CPU)
- Applicable to any other figure of merit of interest for your use case

Clues to boosting performance





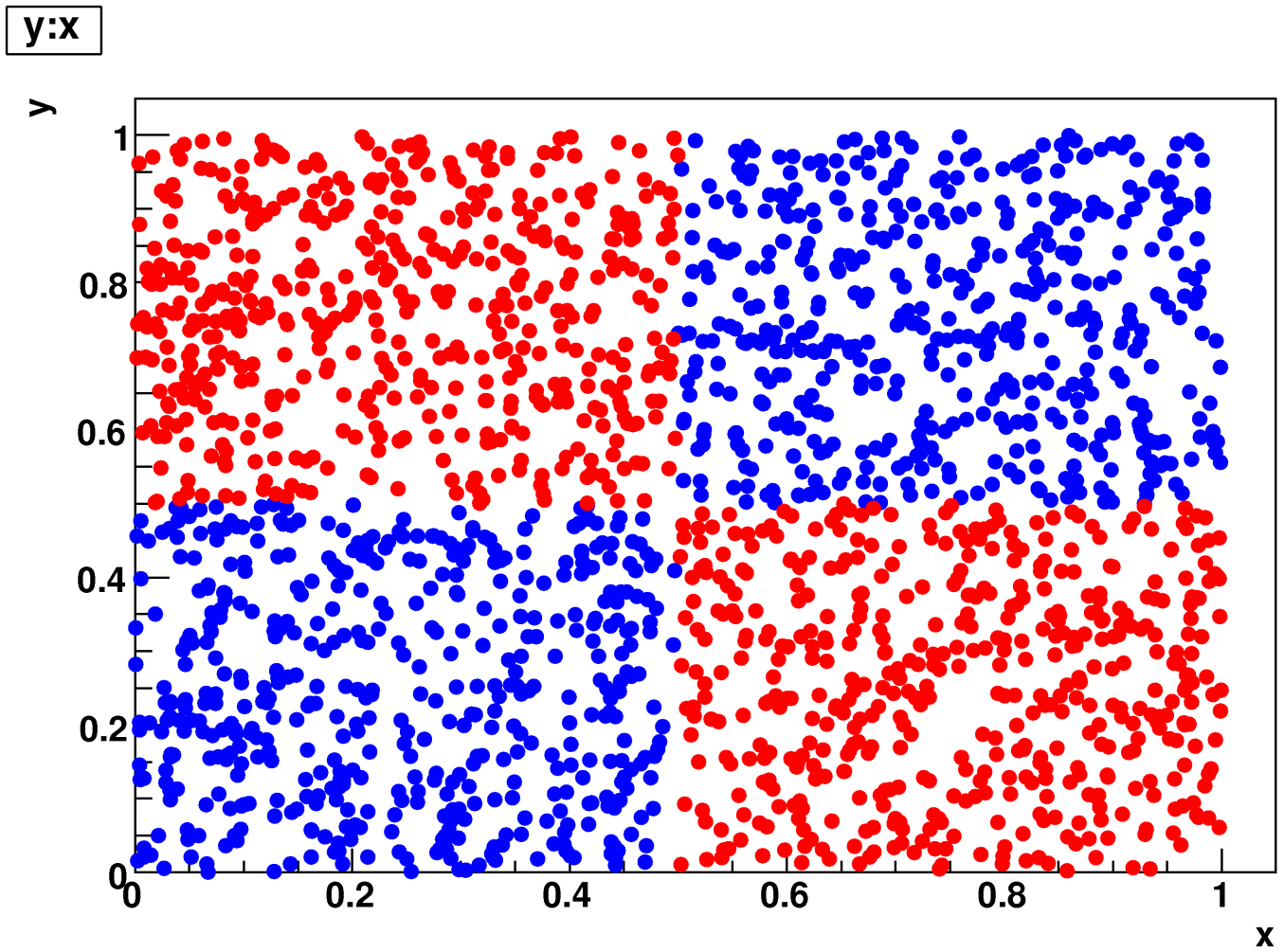
- First tree is best, others are minor corrections
- Specialised trees do not perform well on most events $\Rightarrow$ decreasing tree weight and increasing misclassification rate
- Last tree is not better evolution of first tree, but rather a pretty bad DT that only does a good job on few cases that the other trees could not get right
- But adding trees may increase reliability of prediction: margins explanation [Shapire&Freund 2012]
- Double descent risk curve and interpolation regime [Belkin 2019]

Gradient boosting [Friedman 2001]

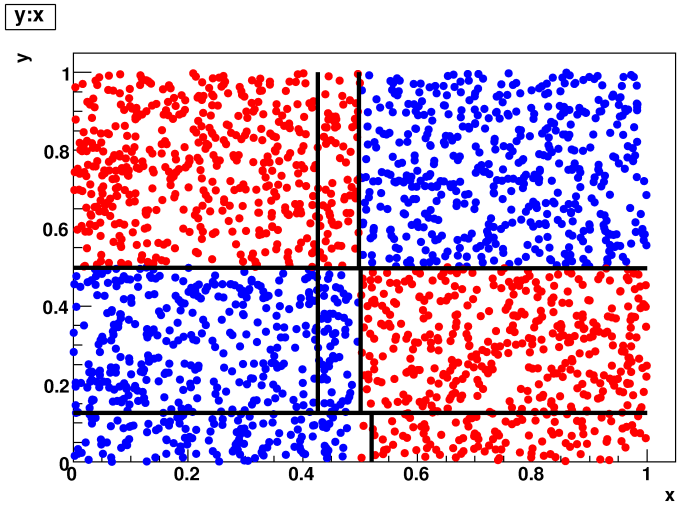
- AdaBoost recast in a statistical framework: corresponds to minimising an exponential loss
- Generalisation: formulate boosting as numerical optimisation problem, minimise loss function by adding trees using gradient descent procedure
- Procedure:
  - Build imperfect model F_k at step k (sometimes $F_k(x) \neq y$)
  - Improve model: $F_{k+1}(x) = F_k(x) + h_k(x) = y$, or residual $h_k(x) = y - F_k(x)$
  - Train new classifier on residual
- Example: mean squared error loss function $L_{\text{MSE}}(x, y) = \frac{1}{2}\left(y - F_k(x)\right)^2$
  - minimising loss $J = \sum_i L_{\text{MSE}}(x_i, y_i)$ leads to $\frac{\partial J}{\partial F_k(x_i)} = F_k(x_i) - y_i$

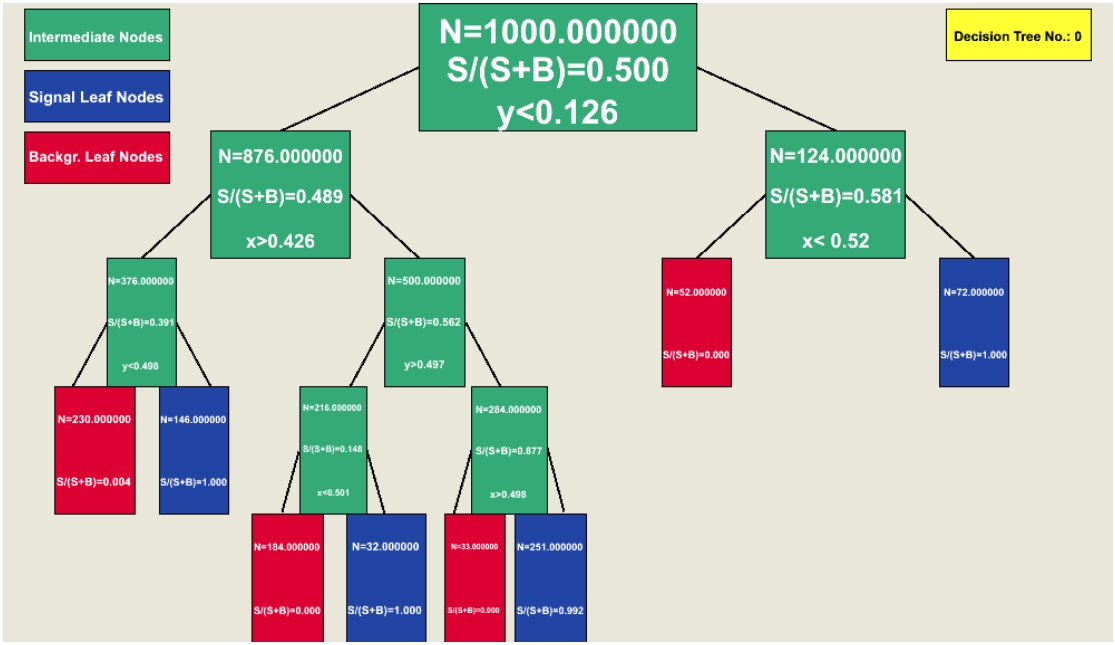
    $\Rightarrow$ residual as negative gradient: $h_k(x_i) = y_i - F_k(x_i) = -\frac{\partial J}{\partial F_k(x_i)}$
- Generalised to any differentiable loss function

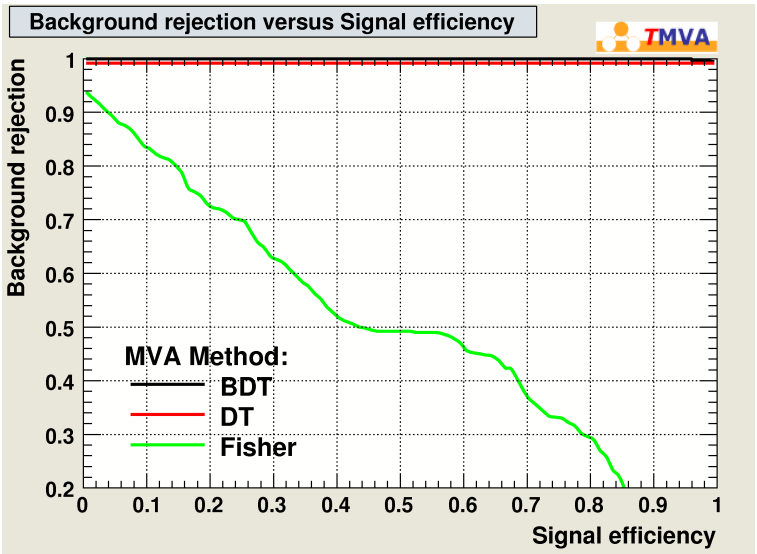
Example: XOR problem



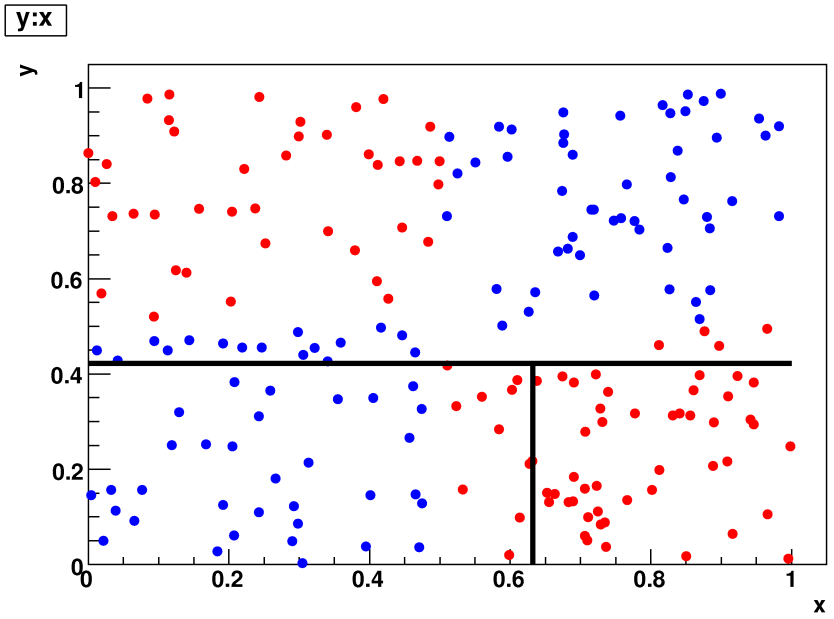
Example: XOR problem





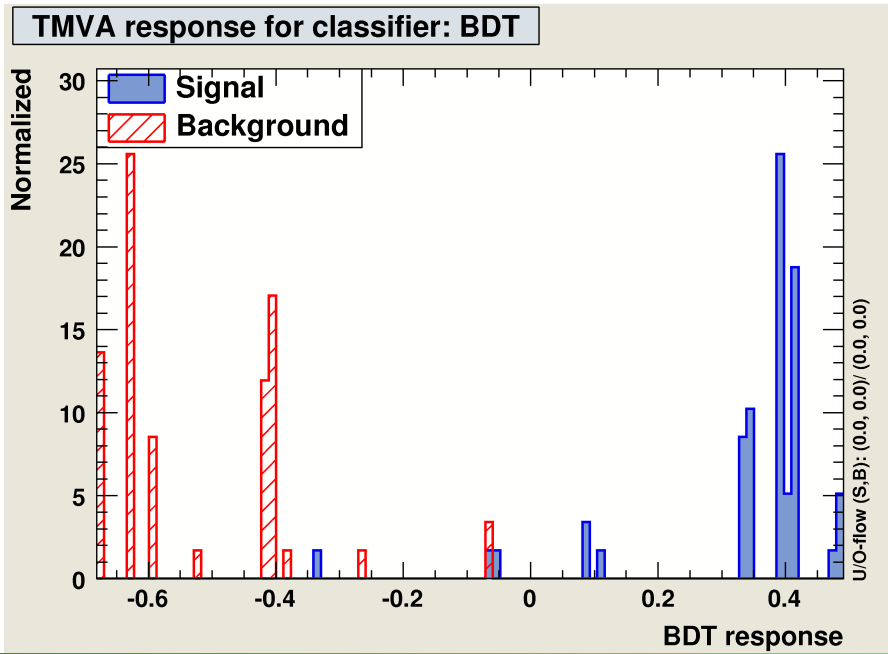


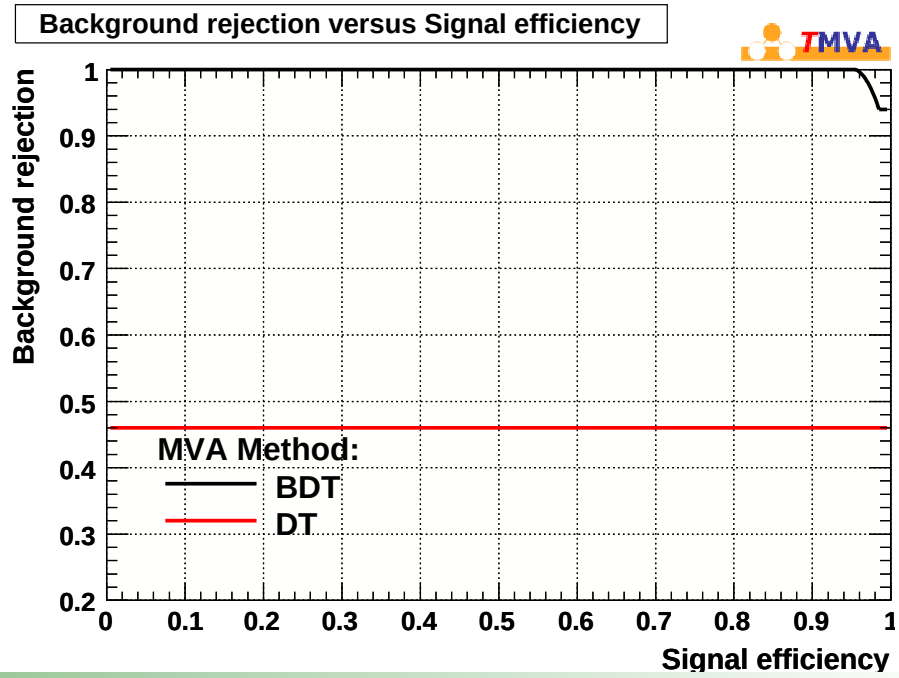
Example: XOR with 100 events



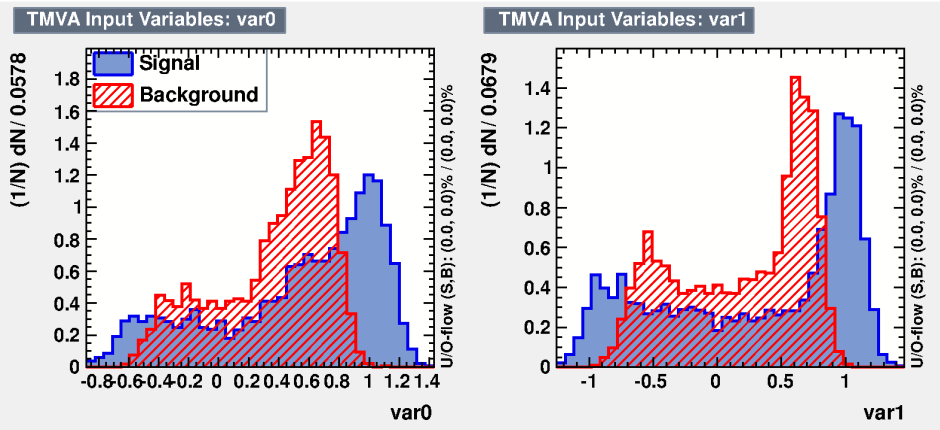
Small statistics

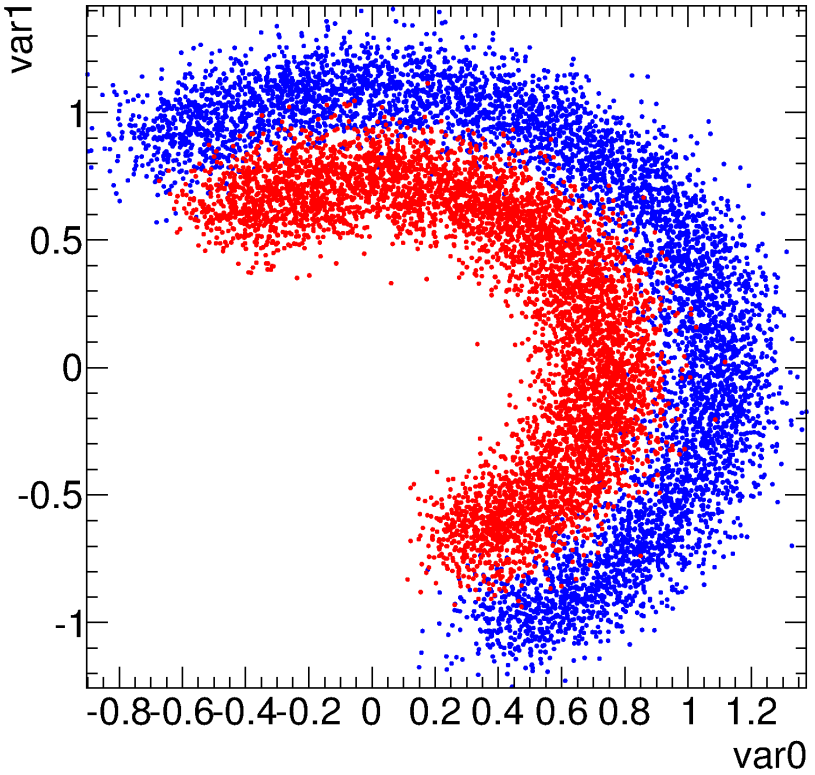
- Single tree not so good
- BDT very good: high performance discriminant from combination of weak classifiers





Circular correlation

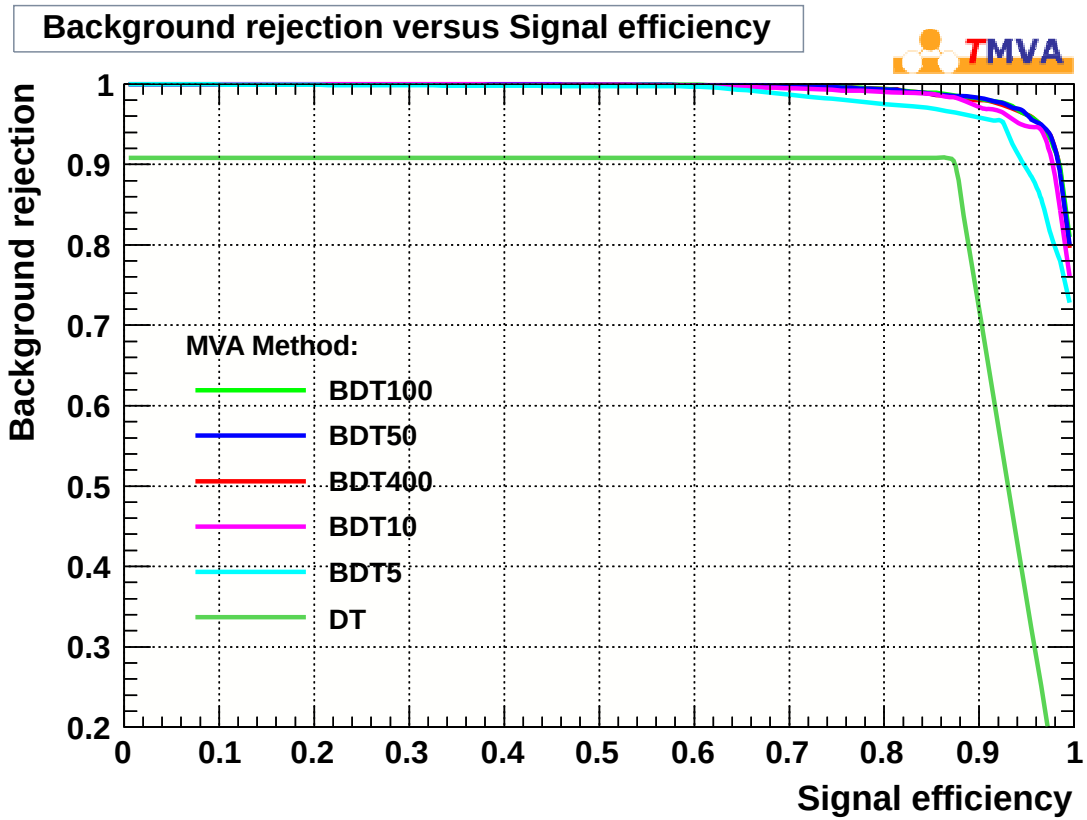




Circular correlation

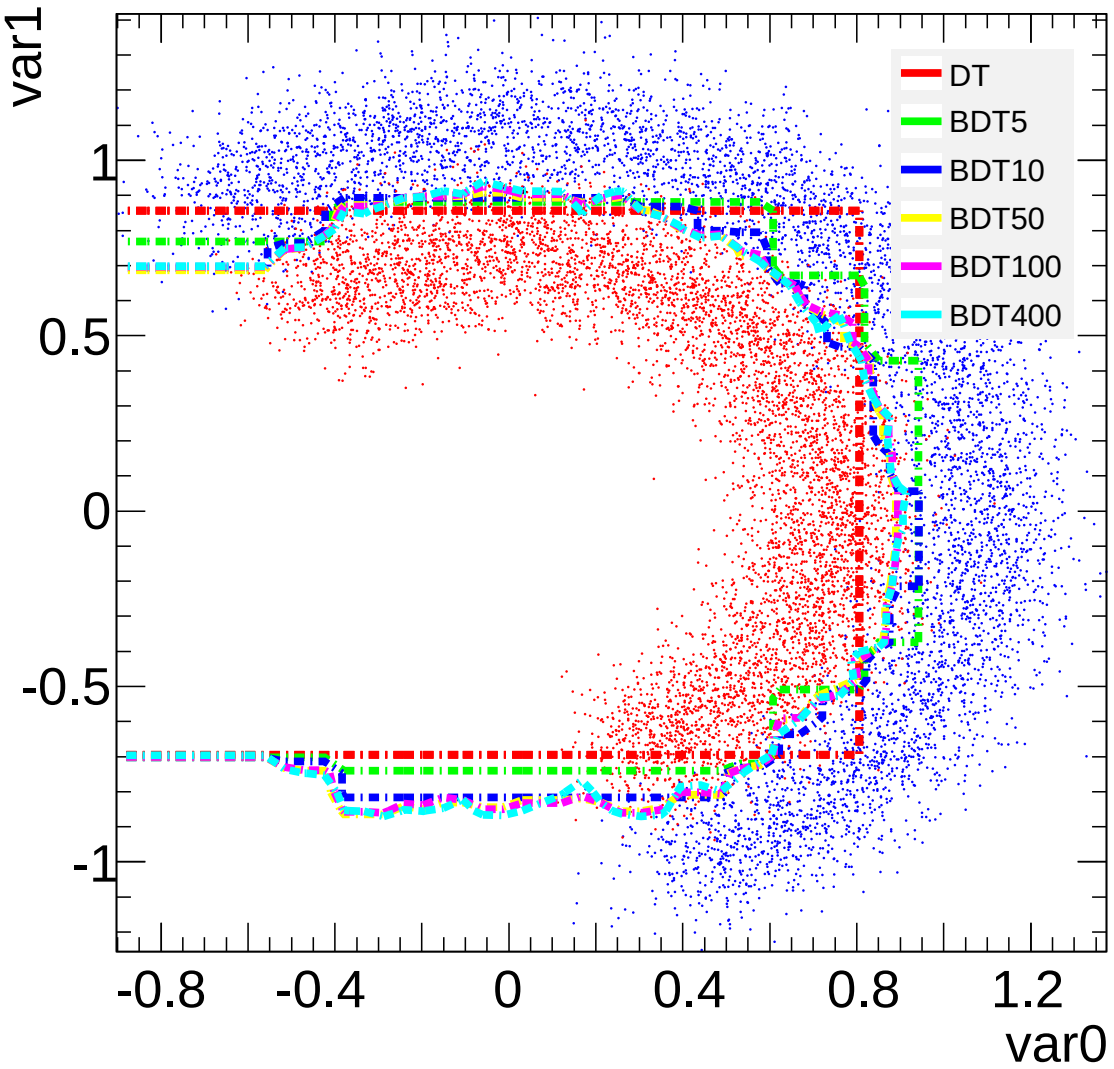
Boosting longer

- Compare performance of single DT and BDT with more and more trees (5 to 400)
- All other parameters unchanged



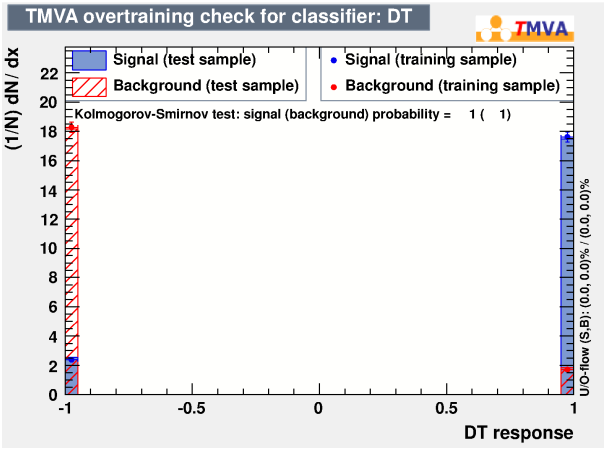
- Single (small) DT: not so good
- More trees $\Rightarrow$ improve performance until saturation

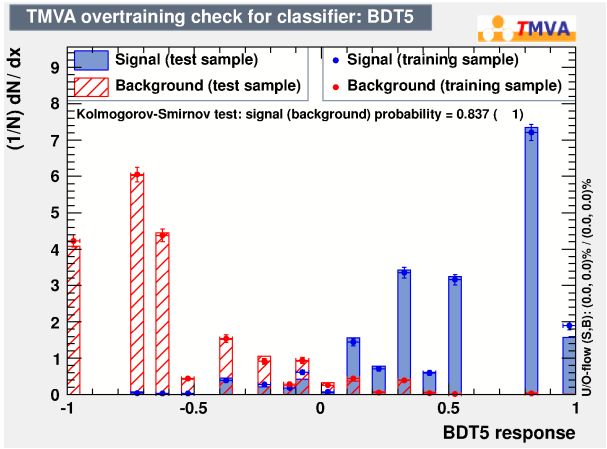
Decision contours

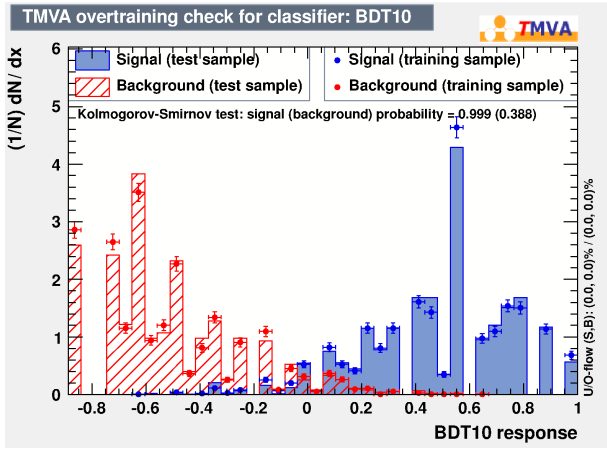


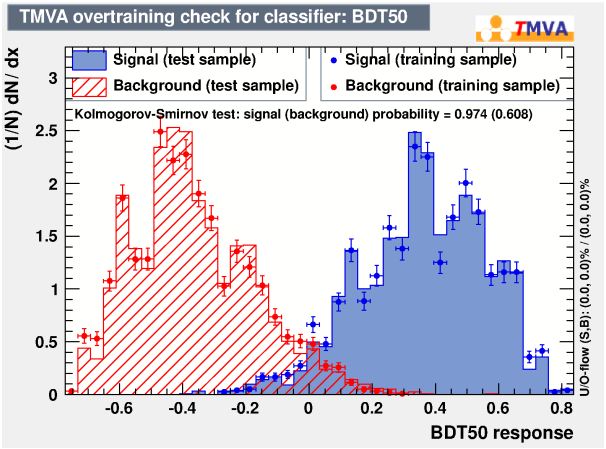
- Note: max tree depth = 3
- Single (small) DT: not so good. Note: a larger tree would solve this problem
- More trees ⇒ improve performance (less step-like, closer to optimal separation) until saturation
- Largest BDTs: wiggle a little around the contour ⇒ picked up features of training sample, that is, overtraining

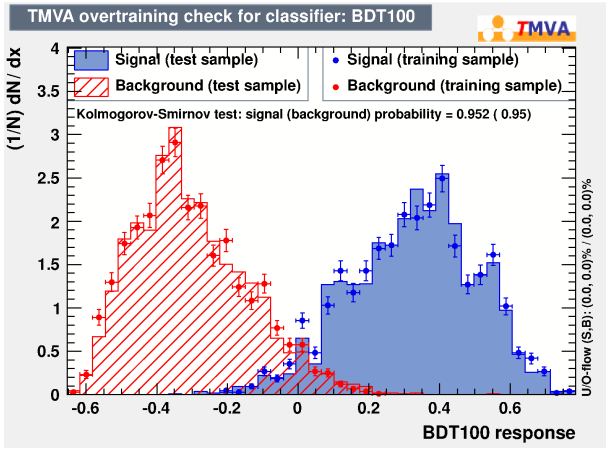
Training/testing output

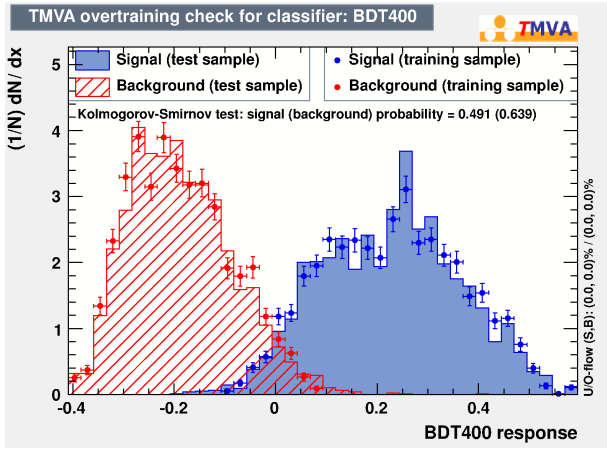






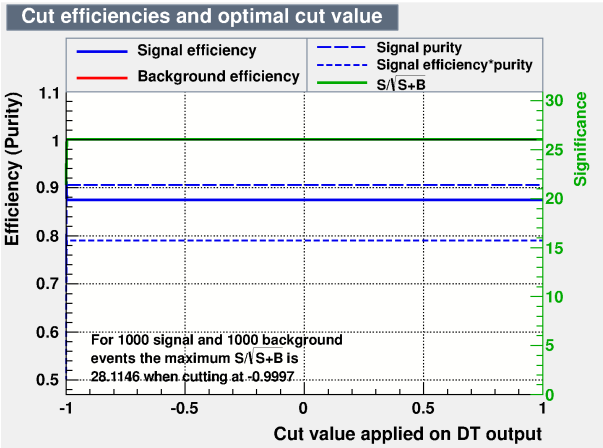


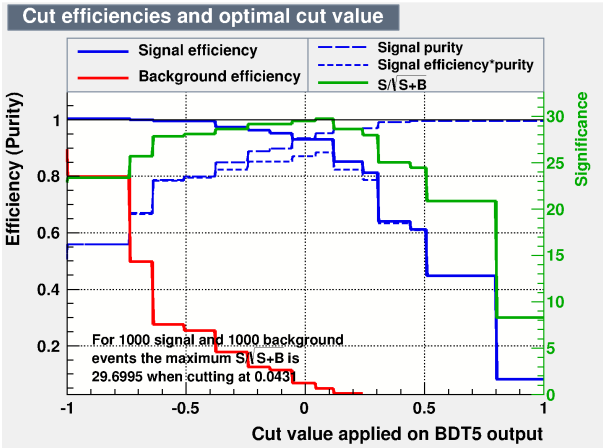


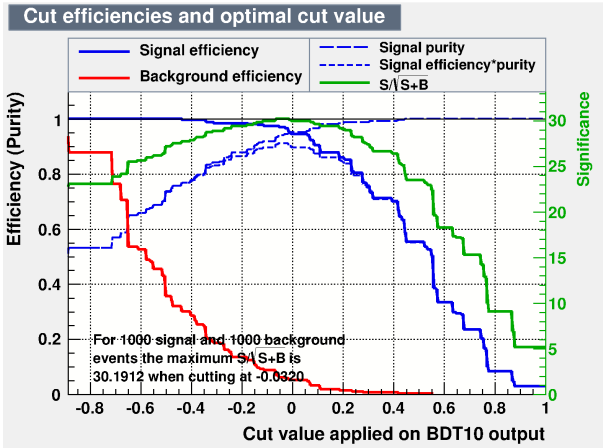


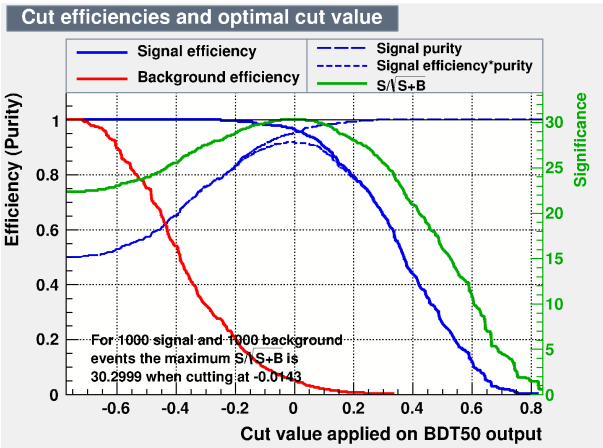
- Better shape with more trees: quasi-continuous
- Overtraining because of disagreement between training and testing? Let's see...

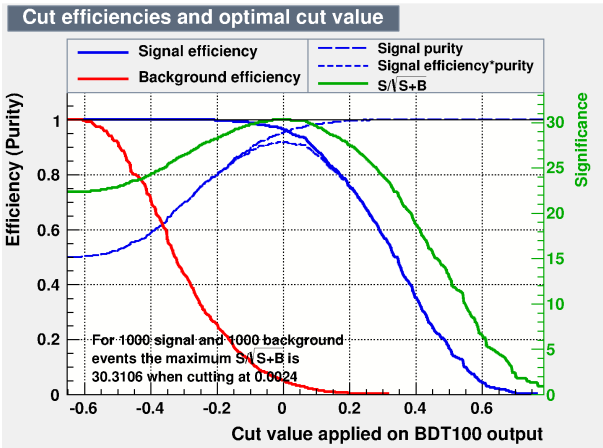
Performance in optimal significance

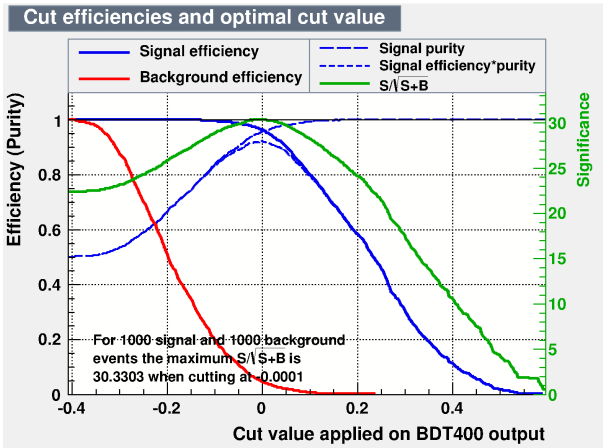






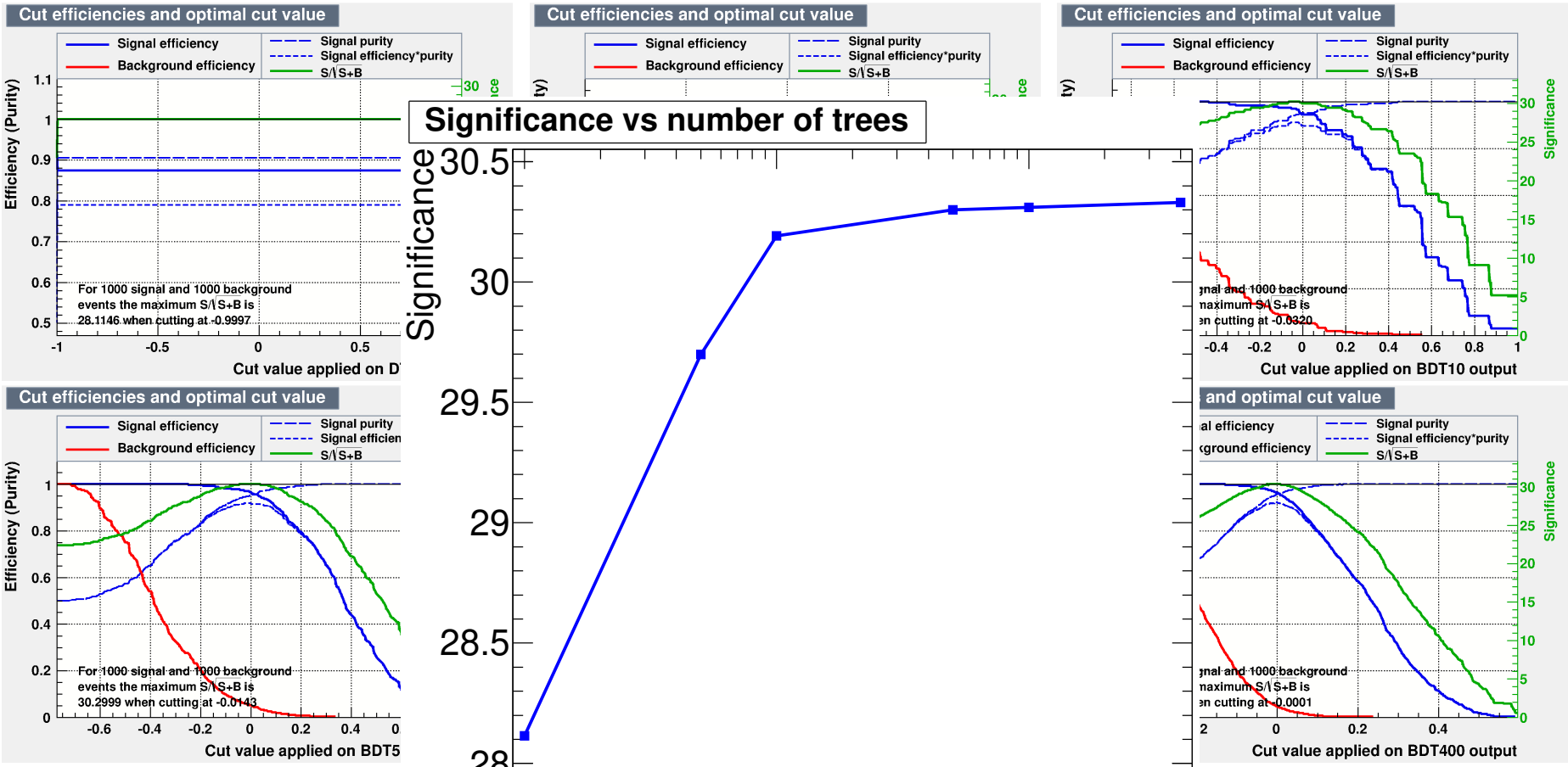






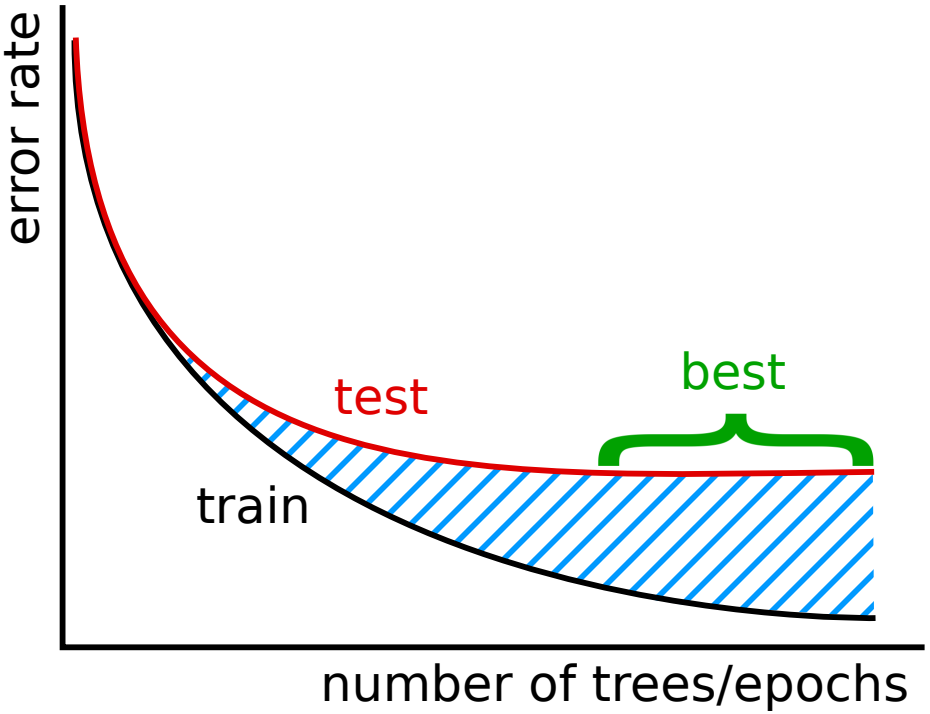
- Best significance actually obtained with last BDT, 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less "stepped" output desirable? $\Rightarrow$ maybe 50 is reasonable

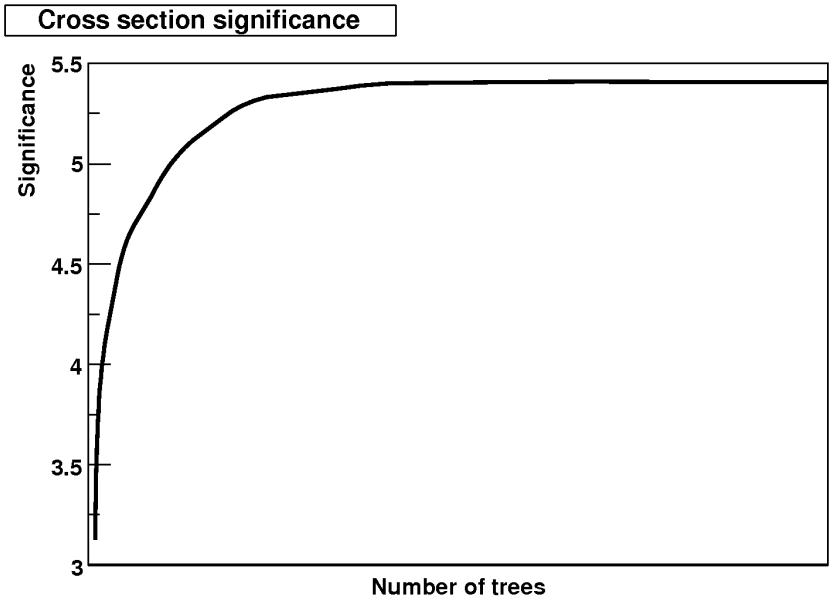
Performance in optimal significance



- Best significar 400 trees!
- But to be fair, equivalent performance with 10 trees already
- Less "stepped" output desirable? $\Rightarrow$ maybe 50 is reasonable

Getting best performance





Other averaging techniques

Bagging (Bootstrap aggregating) [Breiman 1996]

- Before building tree T_k take random sample of N events from training sample with replacement
- Train T_k on it
- Events not picked form "out of bag" validation sample
- Applicable to other techniques than DT
  - tends to produce more stable and better classifier

Other averaging techniques

Bagging (Bootstrap aggregating) [Breiman 1996]

- Before building tree T_k take random sample of N events from training sample with replacement
- Train T_k on it
- Events not picked form "out of bag" validation sample
- Applicable to other techniques than DT
  - tends to produce more stable and better classifier

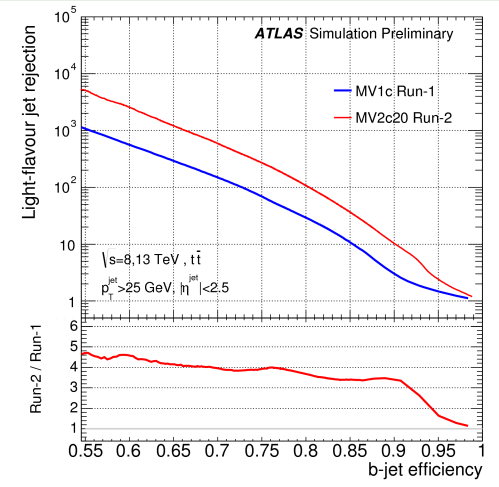
Random forests [Breiman 2001]

- Same as bagging
- In addition, pick random subset of variables to consider for each node split
- Two levels of randomisation, much more stable output
- Often as good as boosting

ATLAS b-tagging in Run 2

▸ Eur. Phys. J. C 79 (2019) 970

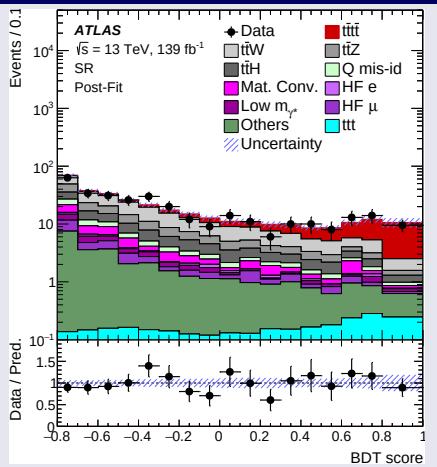
- Run 1 MV1c: NN trained from output of other taggers
- Run 2 MV2c20: BDT using feature variables of underlying algorithms and p_T, η of jets
- Run 2: introduced IBL (new innermost pixel layer)
  $\Rightarrow$ explains part of the performance gain, but not all



ATLAS $t\bar{t}t\bar{t}$ production evidence

▸ Eur. Phys. J. C **80** (2020) 1085 ▸ arXiv:2007.14858 [hep-ex]

- BDT output used in final fit to measure cross section
- Constraints on systematic uncertainties from profiling

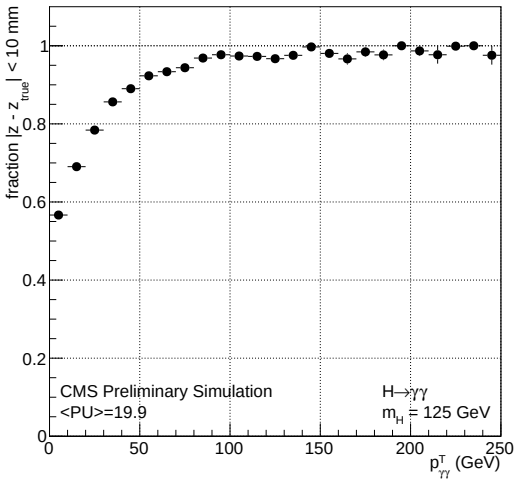


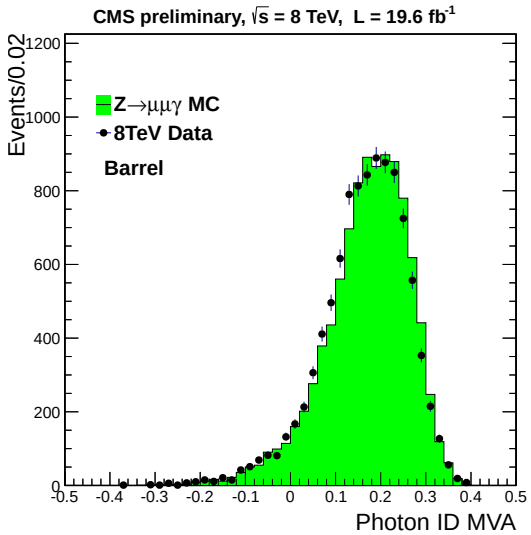
BDT in HEP: CMS $H \to \gamma\gamma$ result

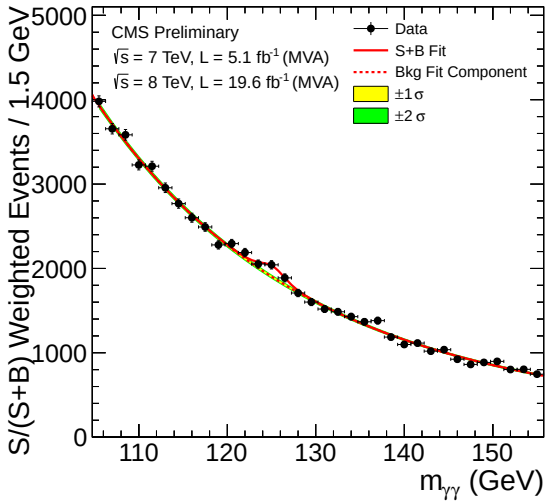
▸ CMS-PAS-HIG-13-001

Hard to use more BDT in an analysis:

- vertex selected with BDT
- 2nd vertex BDT to estimate probability to be within 1cm of interaction point
- photon ID with BDT
- photon energy corrected with BDT regression
- event-by-event energy uncertainty from another BDT
- several BDT to extract signal in different categories





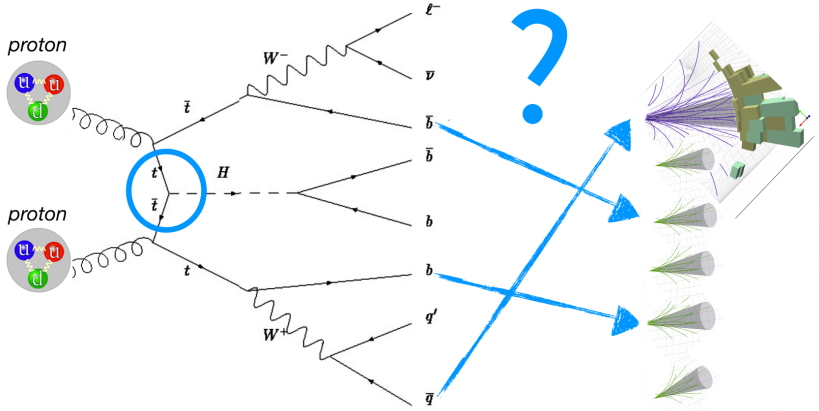


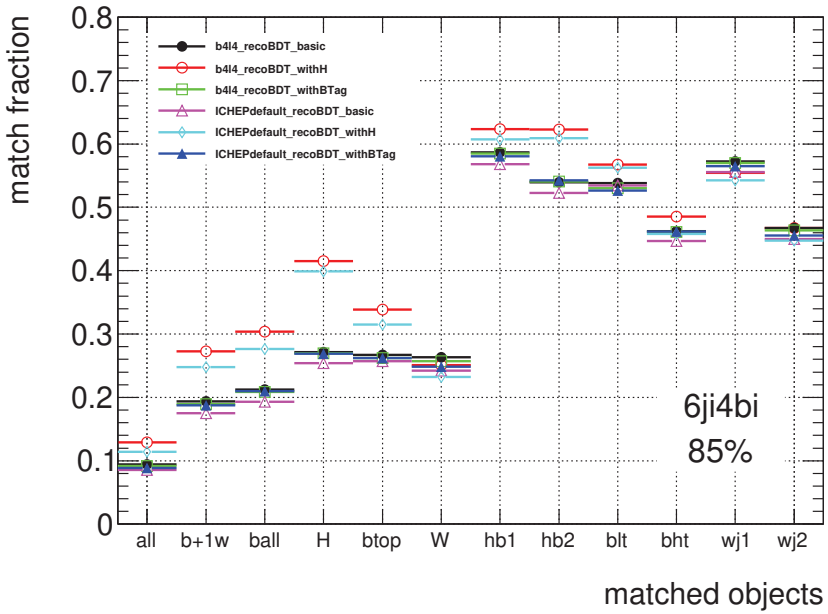
BDT in HEP: reducing combinatorics

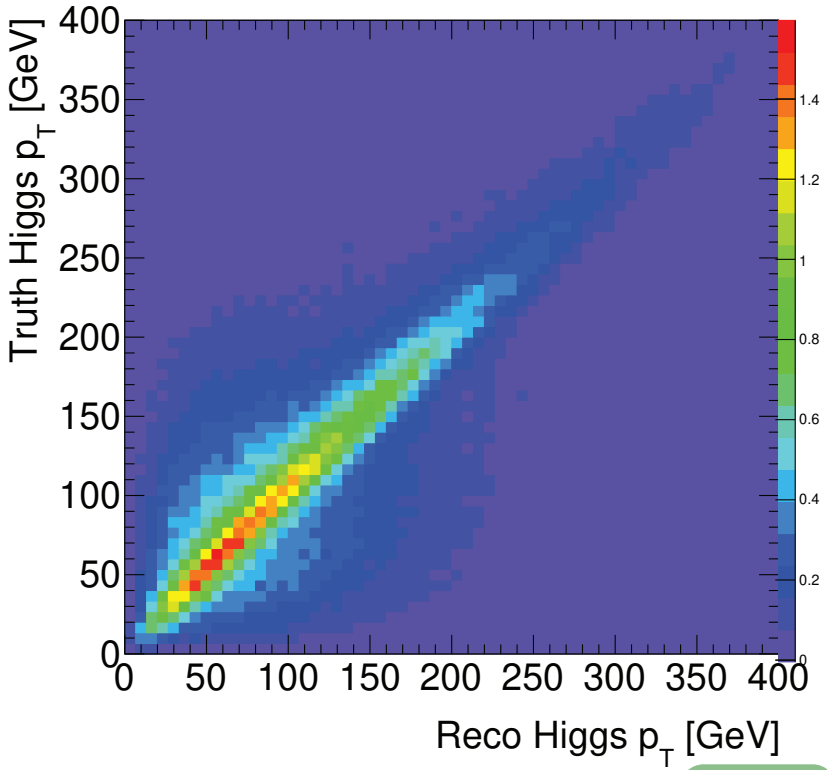
$t\bar{t}H(b\bar{b})$ reconstruction

- Match jets and partons in high-multiplicity final state
- BDT trained on all combinations
- New inputs to classification BDT
- Access to Higgs p_T, origin of b-jets

▸ Phys. Rev. D 97, 072016 (2018)







▸ thesis

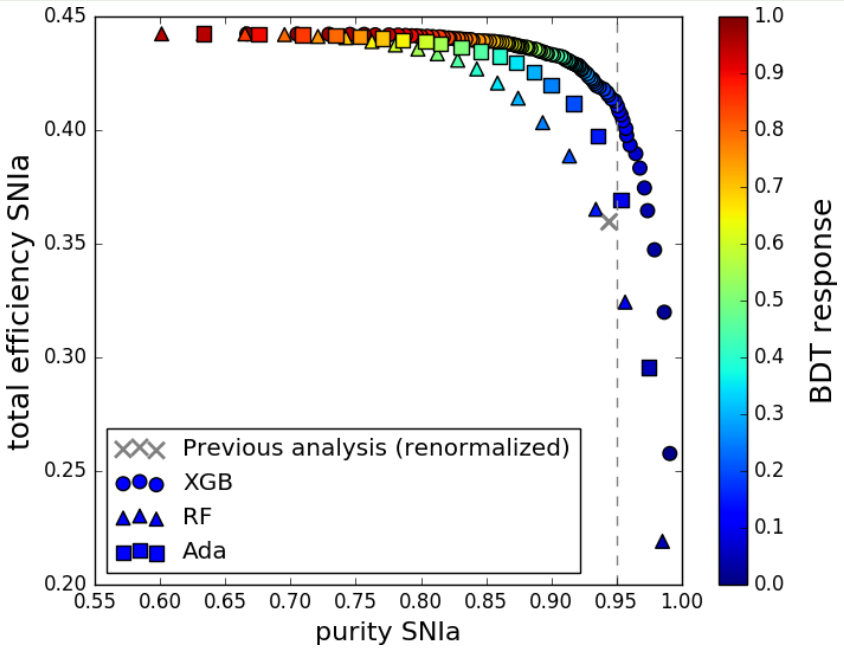
▸ thesis

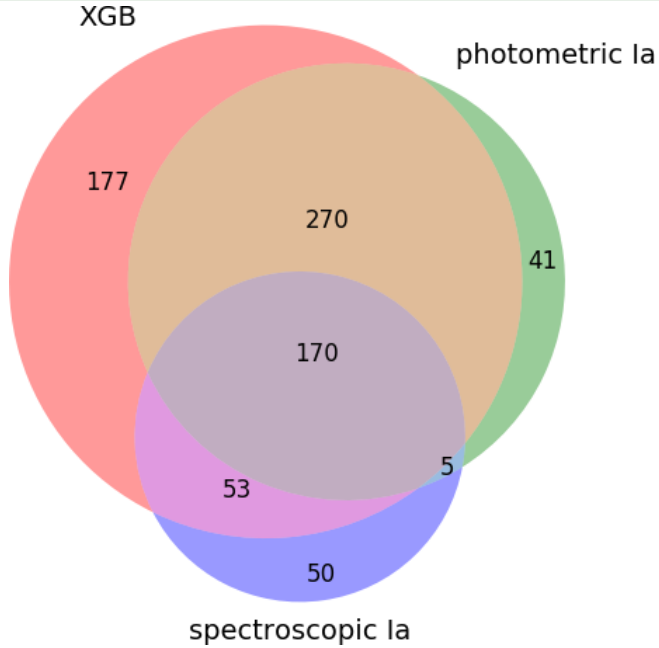
BDT in cosmology: photometric classification

- Type Ia supernovae photometric classification
- Deriving redshifts from SN light curves on real data
- Tried random forest, AdaBoost and XGBoost

	AdaBoost	Random Forest	XGBoost
% SNIa ($CI = 4$ or 5)	74 ± 3	87 ± 3	96 ± 2
% SNIa* ($CI = 3$)	58 ± 6	73 ± 6	88 ± 4

▸ JCAP 12 (2016) 008

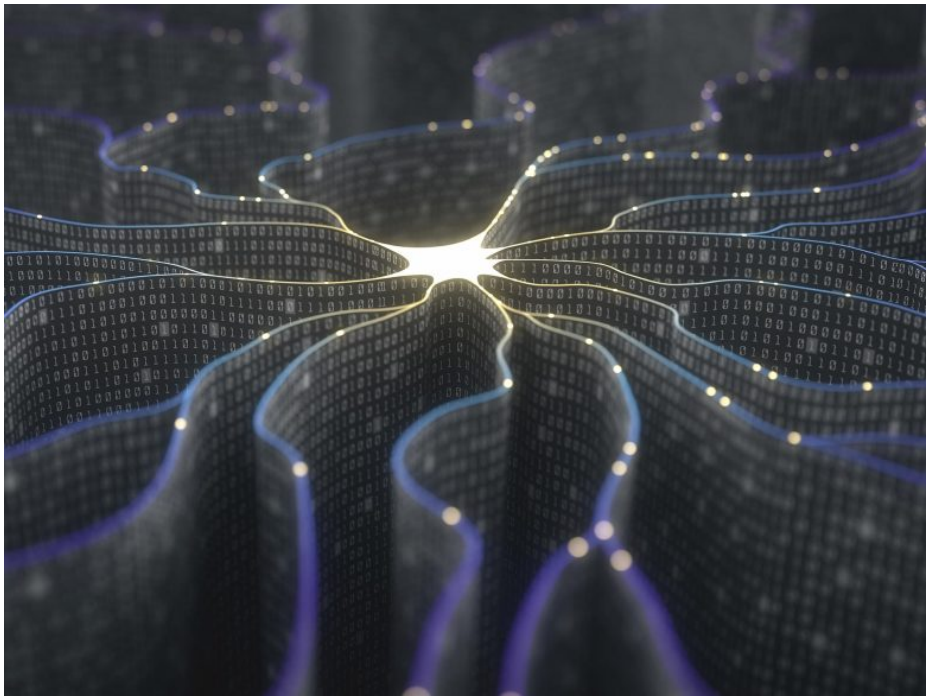




Decision trees are not dead! e.g. NeurIPS2019

- PIDForest: Anomaly Detection via Partial Identification ▸ NeurIPS
- A Debiased MDI (Mean Decrease of Impurity) Feature Importance Measure for Random Forests ▸ NeurIPS
- MonoForest framework for tree ensemble analysis ▸ NeurIPS
- Faster Boosting with Smaller Memory (Yoav S Freund) ▸ NeurIPS
- Minimal Variance Sampling in Stochastic Gradient Boosting ▸ NeurIPS
- Regularized Gradient Boosting ▸ NeurIPS
- Partitioning Structure Learning for Segmented Linear Regression Trees ▸ NeurIPS
- Random Tessellation Forests ▸ NeurIPS
- Optimal Sparse Decision Trees ▸ NeurIPS
- Provably robust boosted decision stumps and trees against adversarial attacks ▸ NeurIPS
- Robustness Verification of Tree-based Models ▸ NeurIPS

Neural networks

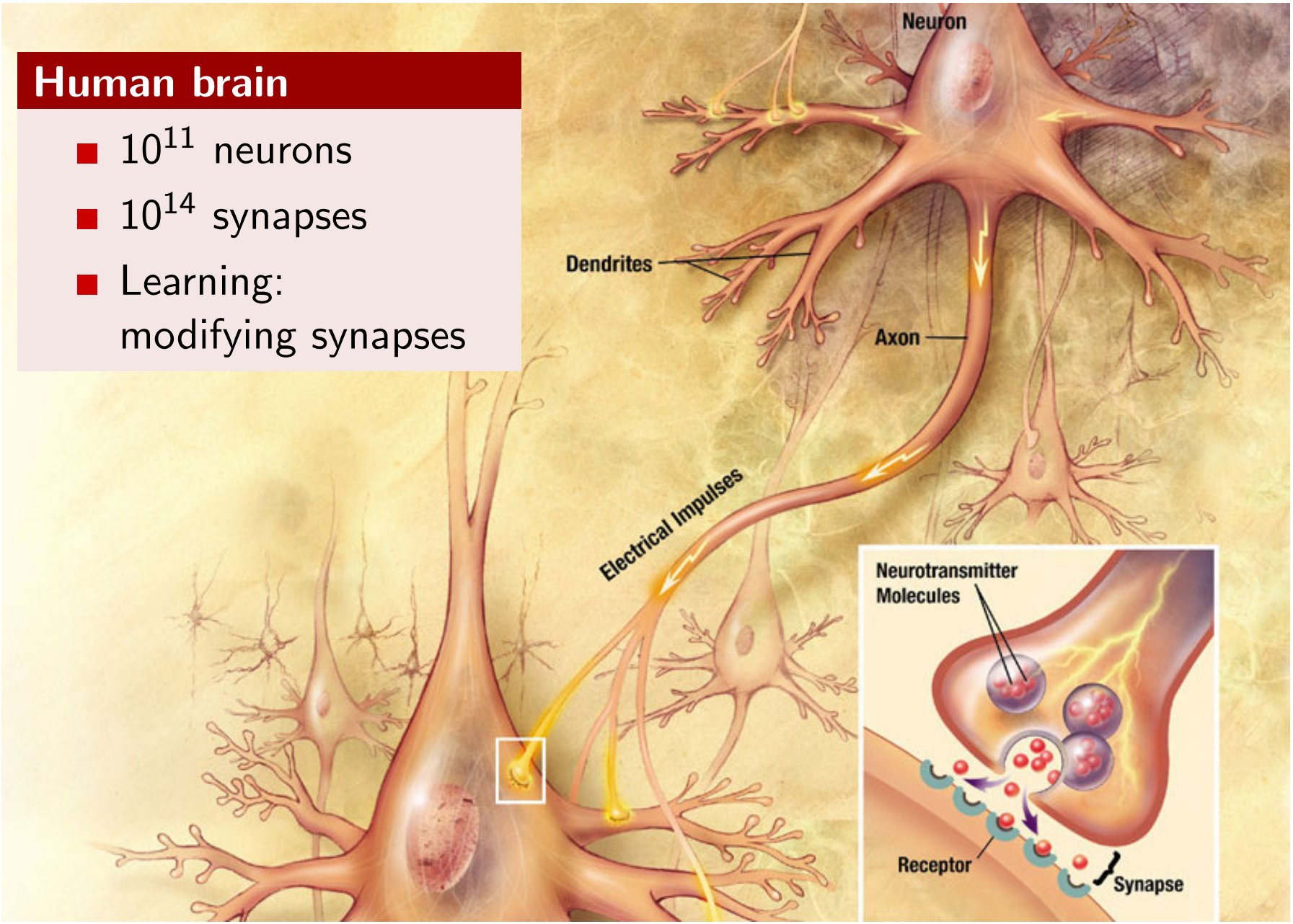


- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Kernel density estimation
- Decision trees
- Boosted decision trees
- Neural networks
  - History
  - Single neuron
  - Neural network training
- Deep neural networks

Neural networks

Human brain

- 10^{11} neurons
- 10^{14} synapses
- Learning: modifying synapses

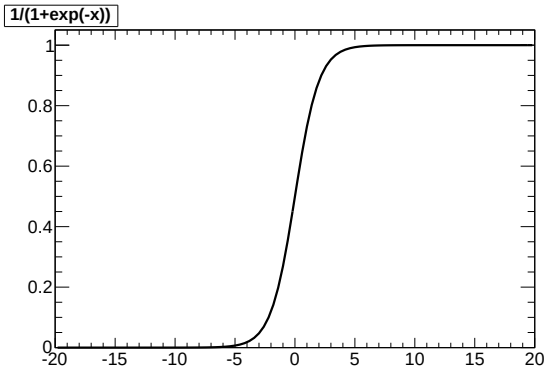


Brief history of artificial neural networks

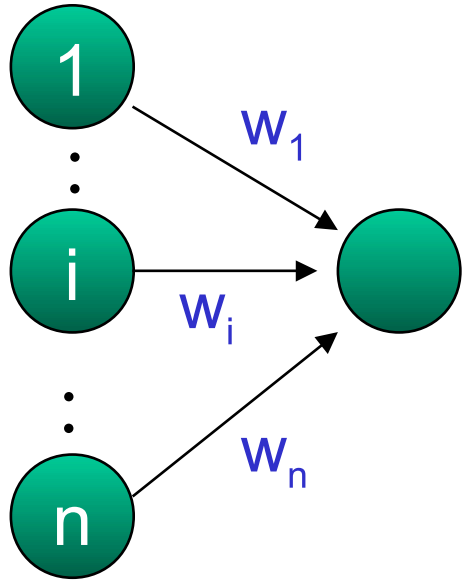
- 1943: W. McCulloch and W. Pitts explore capabilities of networks of simple neurons
- 1958: F. Rosenblatt introduces perceptron (single neuron with adjustable weights and threshold activation function)
- 1969: M. Minsky and S. Papert prove limitations of perceptron (linear separation only) and (wrongly) conjecture that multi-layered perceptrons have same limitations
  $\Rightarrow$ ANN research almost abandoned in 1970s!!!
- 1986: Rumelhart, Hinton and Williams introduce "backward propagation of errors": solves (partially) multi-layered learning

Single neuron

- Remember linear separation (Fisher discriminant):
  $\lambda(x) = w \cdot x = \sum_{i=1}^{n} w_i x_i + w_0$
- Boundary at $\lambda(x) = 0$
- Replace threshold boundary by sigmoid (or tanh):



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



- σ: activation function (neuron activity)
- Neuron behaviour completely controlled by weights $w = \{w_0, \ldots, w_n\}$
- Training: minimisation of error/loss function (quadratic deviations, entropy [maximum likelihood]), via gradient descent or stochastic approximation

Universal approximation theorem

Let $\sigma(.)$ be a non-constant, bounded, and monotone-increasing continuous function. Let $\mathcal{C}(I_n)$ denote the space of continuous functions on the n-dimensional hypercube. Then, for any given function $f \in \mathcal{C}(I_n)$ and $\varepsilon > 0$ there exists an integer M and sets of real constants w_j, w_{ij} where $i = 1, \ldots, n$ and $j = 1, \ldots, M$ such that

$$y(x, w) = \sum_{j=1}^{M} w_j \sigma \left(\sum_{i=1}^{n} w_{ij} x_i + w_{0j} \right)$$

is an approximation of $f(.)$, that is $|y(x) - f(x)| < \varepsilon$.

Artificial neural network

Interpretation

- You can approximate any continuous function to arbitrary precision with a linear combination of sigmoids
- Corollary 1: can approximate any continuous function with neurons!
- Corollary 2: a single hidden layer is enough
- Corollary 3: a linear output neuron is enough

Artificial neural network

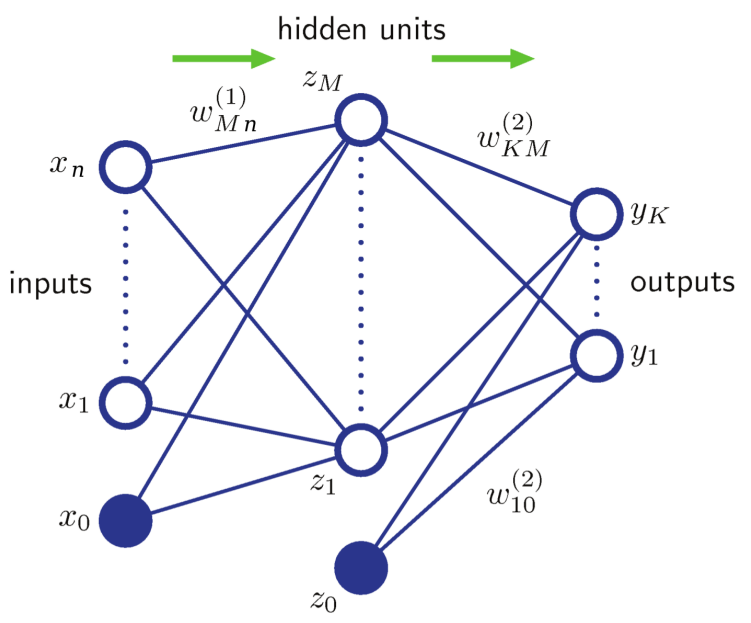
Interpretation

- You can approximate any continuous function to arbitrary precision with a linear combination of sigmoids
- Corollary 1: can approximate any continuous function with neurons!
- Corollary 2: a single hidden layer is enough
- Corollary 3: a linear output neuron is enough

Multilayer perceptron: feedforward network

- Neurons organised in layers
- Output of one layer becomes input to next layer

$$y_k(x, w) = \sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{\sigma\left(\sum_{i=0}^{n} w_{ji}^{(1)} x_i\right)}_{z_j}$$



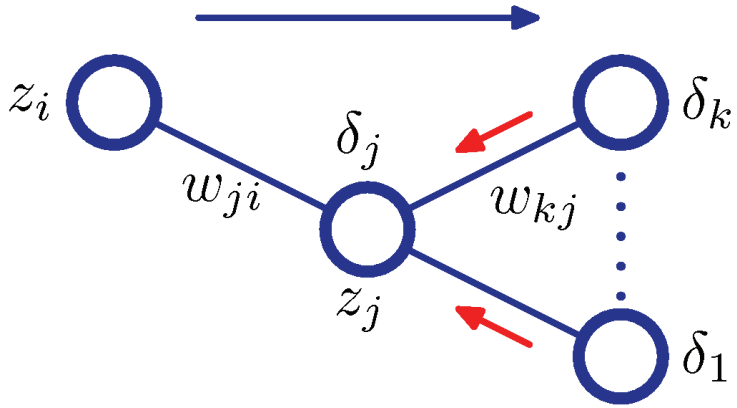
Neural network training

- Training means minimising error function $E(w)$
- $\frac{\partial E}{\partial w_j} = \sum_{n=1}^{N} -(t^{(n)} - y^{(n)}) x_j^{(n)}$ with target $t^{(n)}$ (0 or 1), so $t^{(n)} - y^{(n)}$ is the error on event n
- All events at once (batch learning):
  - weights updated all at once after processing the entire training sample
  - finds the actual steepest descent
  - takes more time
  - usually: mini-batches (send events by batches)
  - new training events: need to restart training from scratch
- or one-by-one (online learning):
  - incremental learning: new training events included as they come
  - speeds up learning
  - may avoid local minima with stochastic component in minimisation
  - careful: depends on the order of training events
- One epoch: going through the entire training data once

Backpropagation

- Training means minimising error function $E(w)$
- For single neuron: $\frac{dE}{dw_k} = (y - t)x_k$
- One can show that for a network:

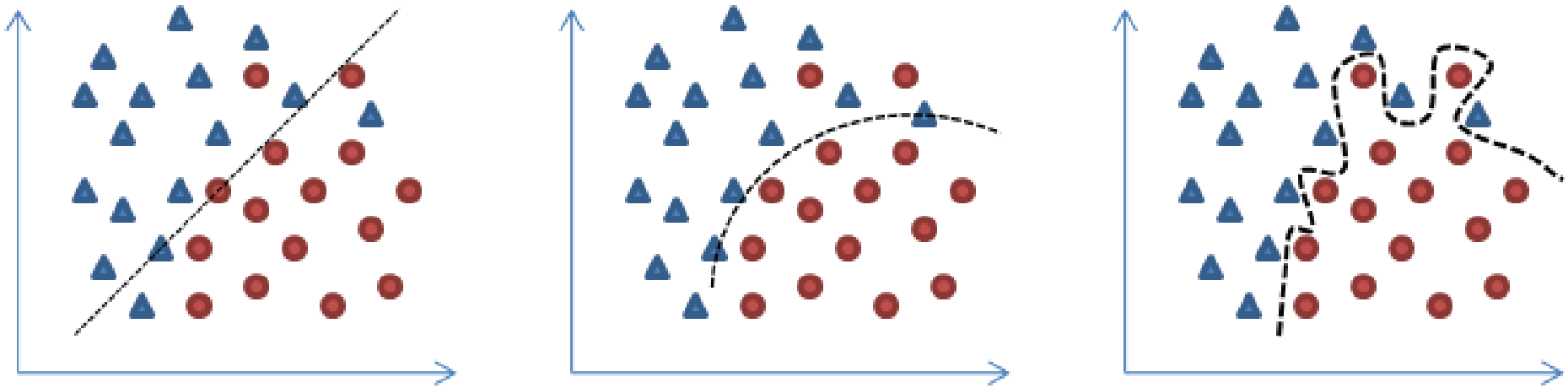
$$\frac{dE}{dw_{ji}} = \delta_j z_i, \text{ where}$$



$$\begin{aligned} \delta_k &= (y_k - t_k) \text{ for output neurons} \\ \delta_j &\propto \sum_k w_{kj}\delta_k \text{ otherwise} \end{aligned}$$

- Hence errors are propagated backwards

Neural network overtraining

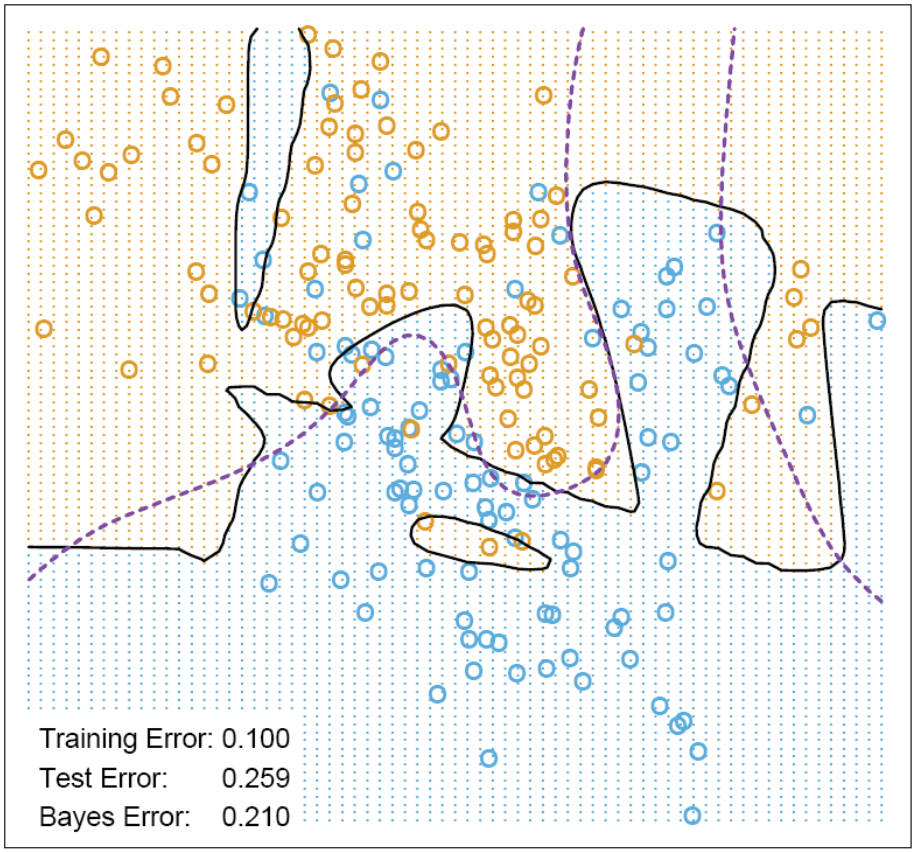


- Diverging weights can cause overfitting
- Mitigate by:
  - early stopping (after a fixed number of epochs)
  - monitoring error on test sample
  - regularisation, introducing a "weight decay" term to penalise large weights, preventing overfitting. For instance L2-regularisation:

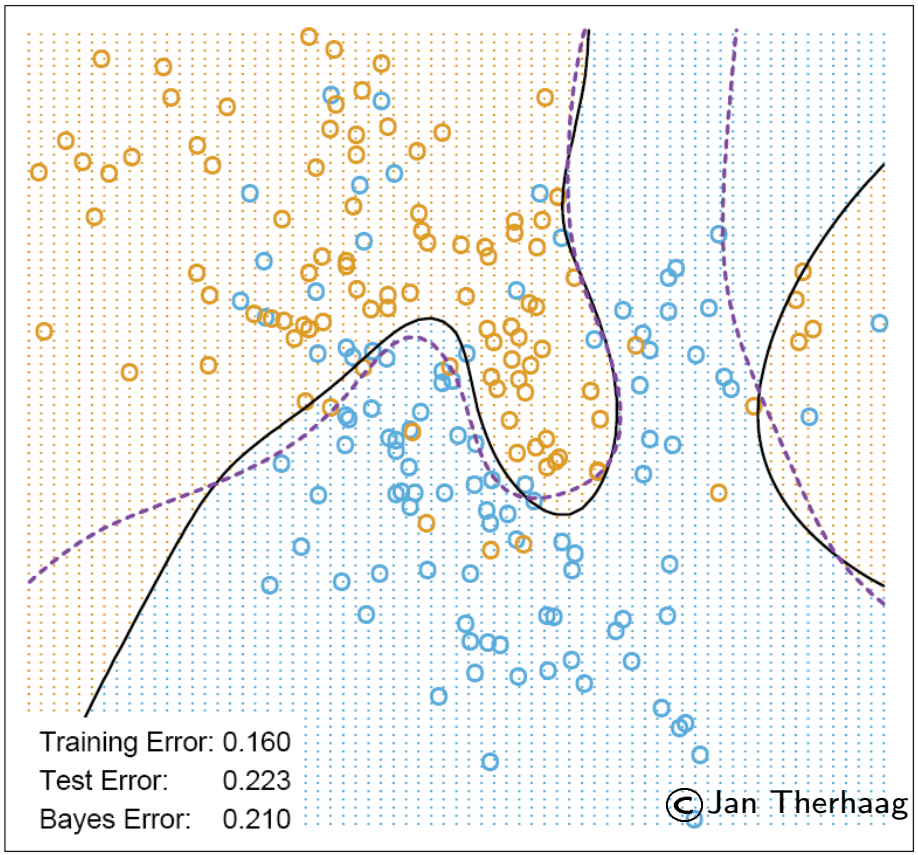
$$\tilde{E}(w) = E(w) + \frac{\alpha}{2} \sum_i w_i^2$$

Regularisation

10 hidden nodes, $\alpha = 0$



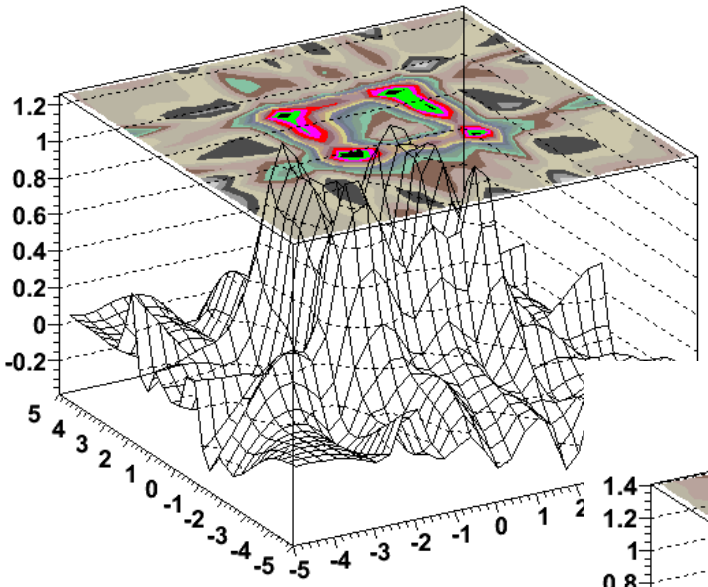
10 hidden nodes, $\alpha = 0.04$



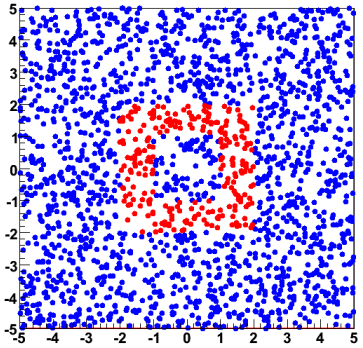
- Much less overfitting, better generalisation properties

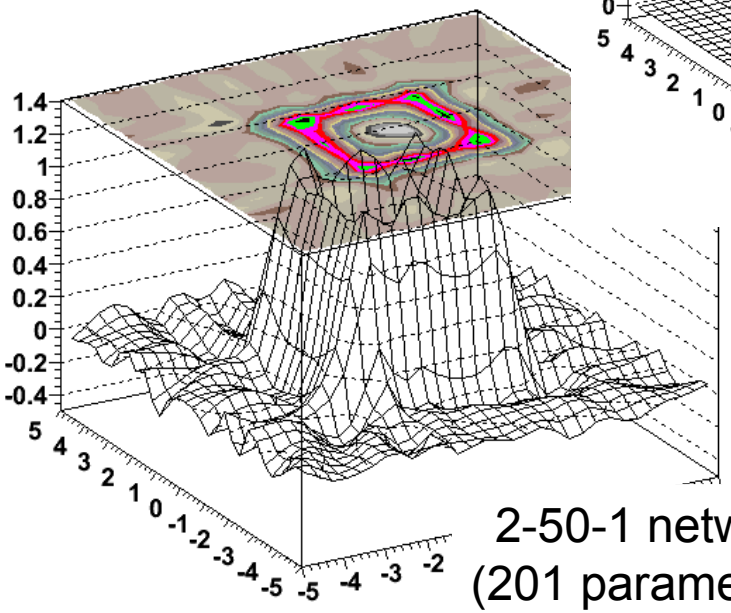
- Preprocess data:
  - if relevant, provide e.g. x/y instead of x and y
  - subtract the mean because the sigmoid derivative becomes negligible very fast (so, input mean close to 0)
  - normalise variances (close to 1)
  - shuffle training sample (order matters in online training)
- Initial random weights should be small to avoid saturation
- Regularise weights to minimise overtraining
- Make sure the training sample covers the full parameter space
- No rule (not even guestimates) about the number of hidden nodes (unless using constructive algorithm, adding resources as needed)
- A single hidden layer is enough for all purposes, but multiple hidden layers may allow for a solution with fewer parameters

Adding a hidden layer

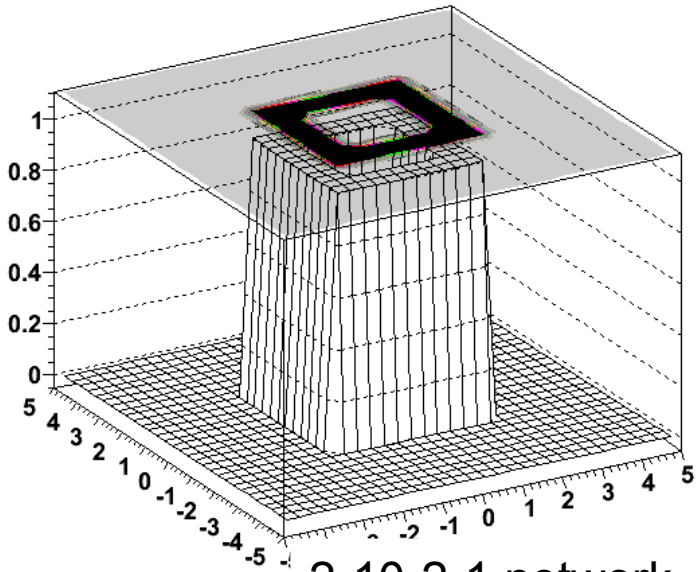


2-20-1 network
(81 parameters)





2-50-1 network
(201 parameters)



2-10-2-1 network
(55 parameters)

Deep neural networks

- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Kernel density estimation
- Decision trees
- Boosted decision trees
- Neural networks
- Deep neural networks
  - ANN interesting again!
  - New activation functions
  - Convolutional networks
  - Recurrent neural networks
  - Auto-encoders
  - Domain adaptation
  - Examples

Deep learning

What is learning?

- Ability to learn underlying and previously unknown structure from examples
  $\Rightarrow$ capture variations
- Deep learning: have several hidden layers (> 2) in a neural network

Motivation for deep learning

- Inspired by the brain (esp. visual cortex)
- Humans organise ideas hierarchically, through composition of simpler ideas
- Heavily unsupervised training, learning simpler tasks first, then combining into more abstract ones
- Learn first order features from raw inputs, then patterns in first order features, then etc.

Deep learning revolution

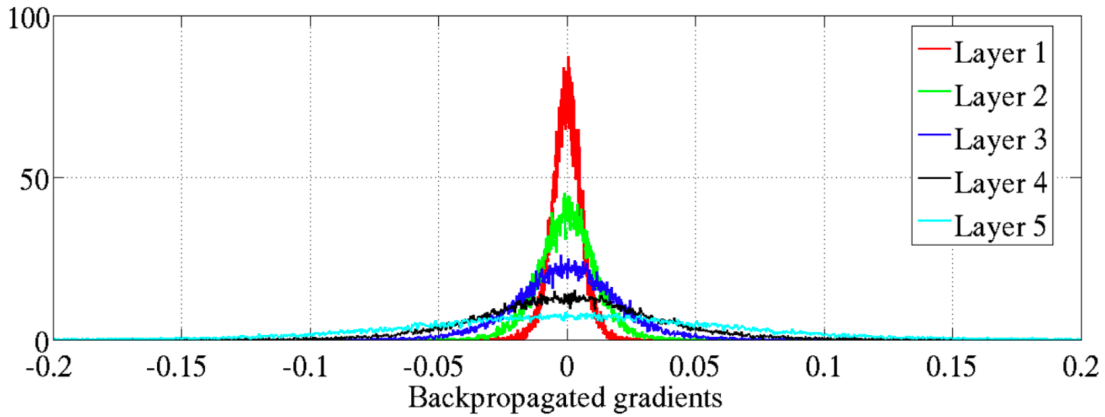
Deep networks were unattractive

- One layer theoretically enough for everything
- Used to perform worse than shallow networks with 1 or 2 hidden layers
- Apparently difficult/impossible to train (using random initial weights and supervised learning with backpropagation)
- Backpropagation issues:
  - requires labelled data (usually scarce and expensive)
  - does not scale well, getting stuck in local minima
  - "vanishing gradient": gradients getting very small further away from output $\Rightarrow$ early layers do not learn much, can even penalise overall performance

Deep learning revolution

Deep networks were unattractive

- One layer theoretically enough for everything
- Used to perform worse than shallow networks with 1 or 2 hidden layers
- Apparently difficult/impossible to train (using random initial weights and supervised learning with backpropagation)
- Backpropagation issues:
  - requires labelled data (usually scarce and expensive)
  - does not scale well, getting stuck in local minima
  - "vanishing gradient": gradients getting very small further away from output $\Rightarrow$ early layers do not learn much, can even penalise overall performance



Deep learning revolution

Deep networks were unattractive

- One layer theoretically enough for everything
- Used to perform worse than shallow networks with 1 or 2 hidden layers
- Apparently difficult/impossible to train (using random initial weights and supervised learning with backpropagation)
- Backpropagation issues:
  - requires labelled data (usually scarce and expensive)
  - does not scale well, getting stuck in local minima
  - "vanishing gradient": gradients getting very small further away from output $\Rightarrow$ early layers do not learn much, can even penalise overall performance

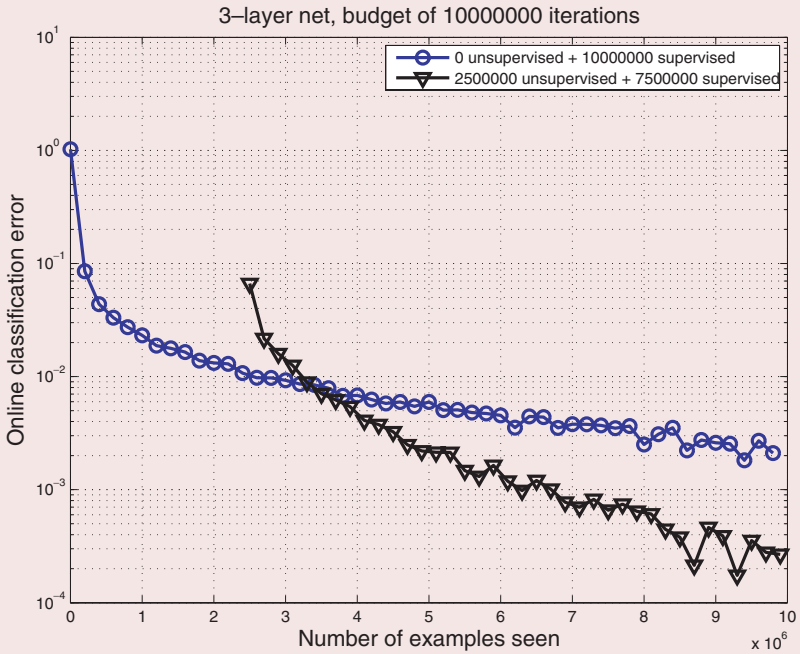
Breakthroughs around 2006 (Bengio, Hinton, LeCun)

- Train each layer independently (first practical approach)
- Can use unlabelled data (a lot of it), with unsupervised training
- New activation functions
- Possible thanks to algorithmic innovations, computing resources, data!

Why does unsupervised training work?

Example

- Stacked denoising auto-encoders
- 10 million handwritten digits
- First 2.5 million used for unsupervised pre-training



- Worse with supervision: eliminates projections of data not useful for local cost but helpful for deep model cost

Optimisation hypothesis

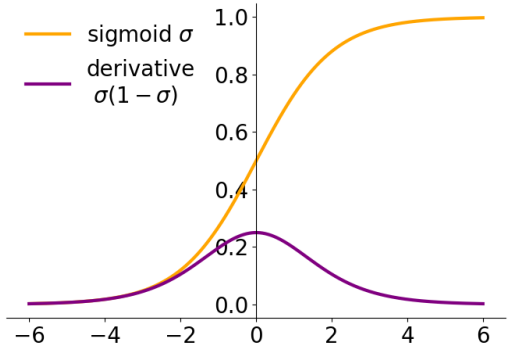
- Training one layer at a time scales well
- Backpropagation from sensible features
- Better local minimum than random initialisation, local search around it

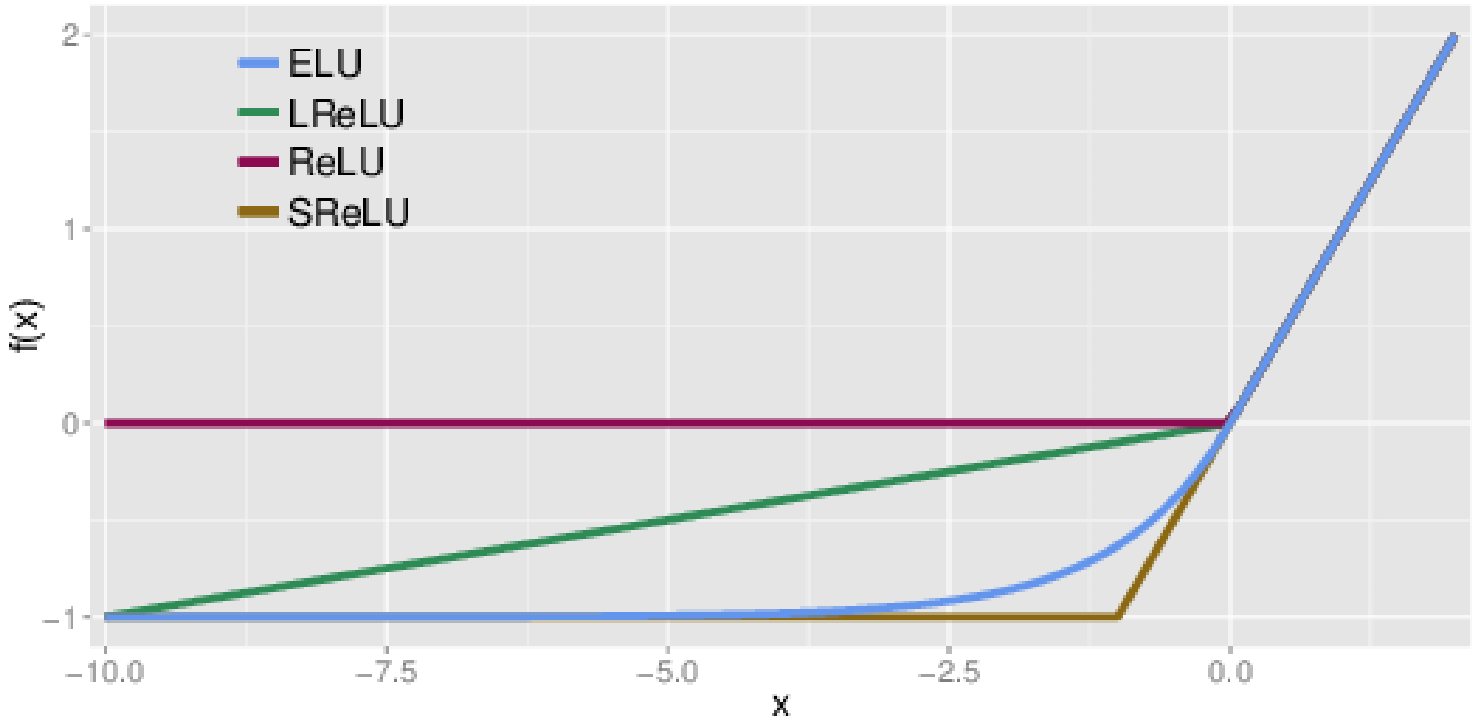
Overfitting/regularisation hypothesis

- More info in inputs than labels
- No need for final discriminant to discover features
- Fine-tuning only at category boundaries

Rectified linear unit (ReLU) activation

- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

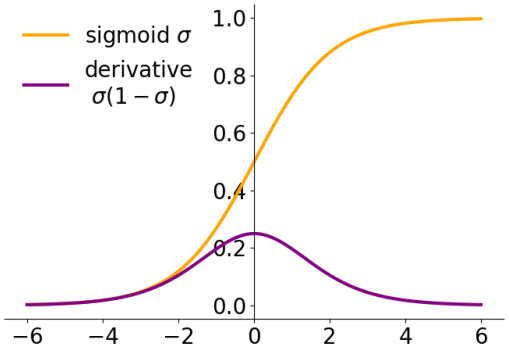


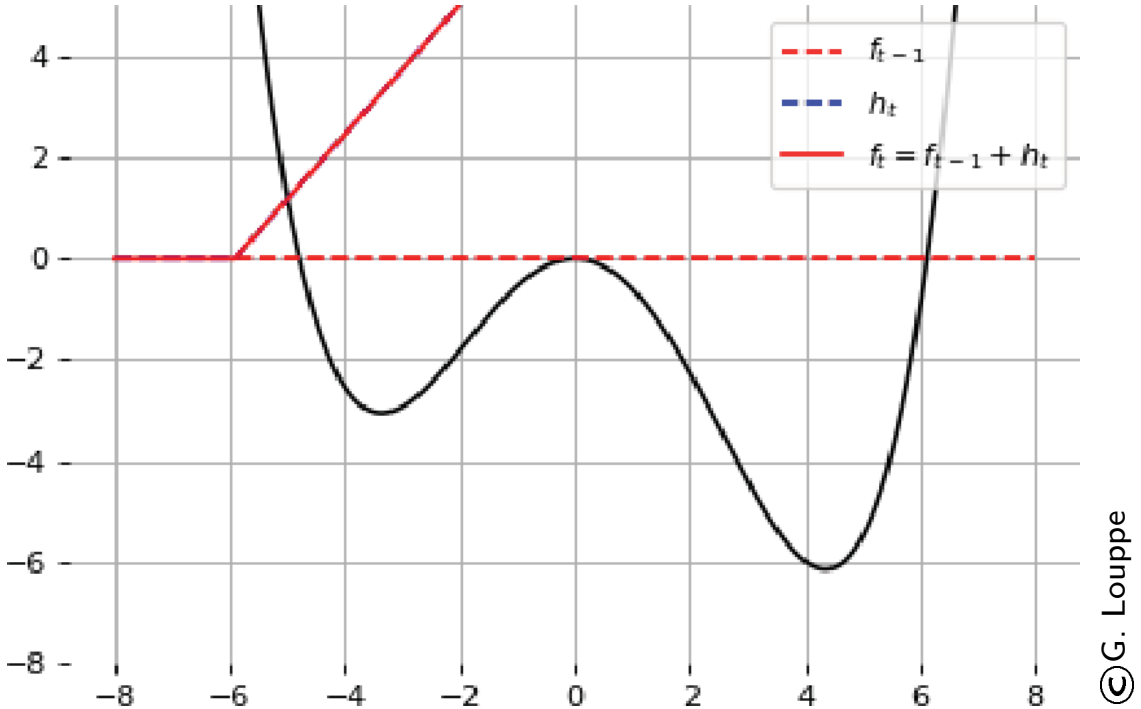


▸ arXiv: 1511.07289

Rectified linear unit (ReLU) activation

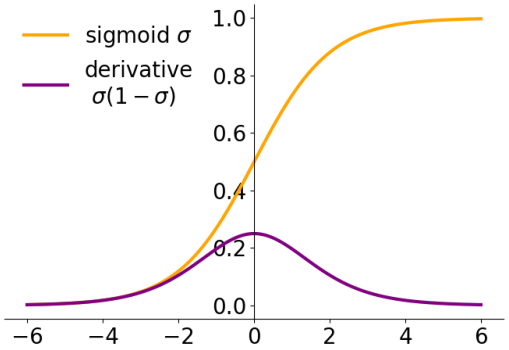
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

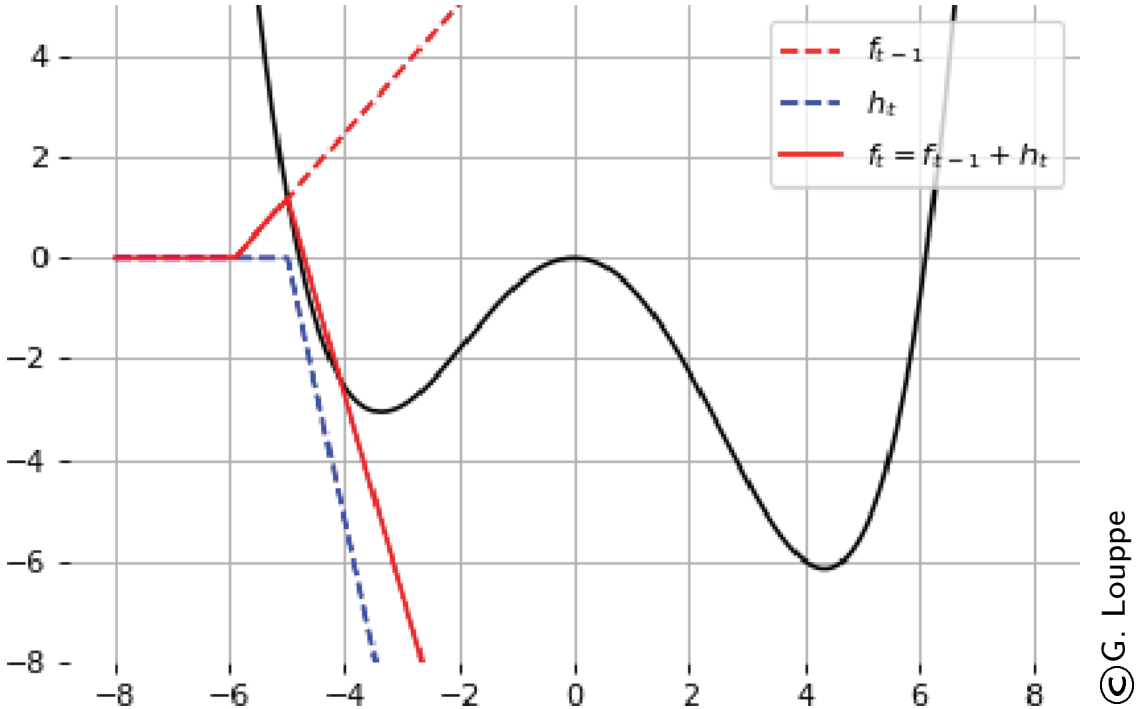




Rectified linear unit (ReLU) activation

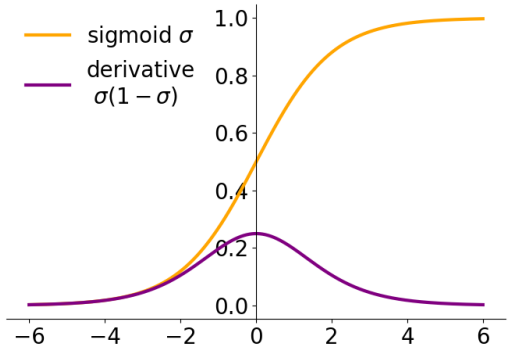
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

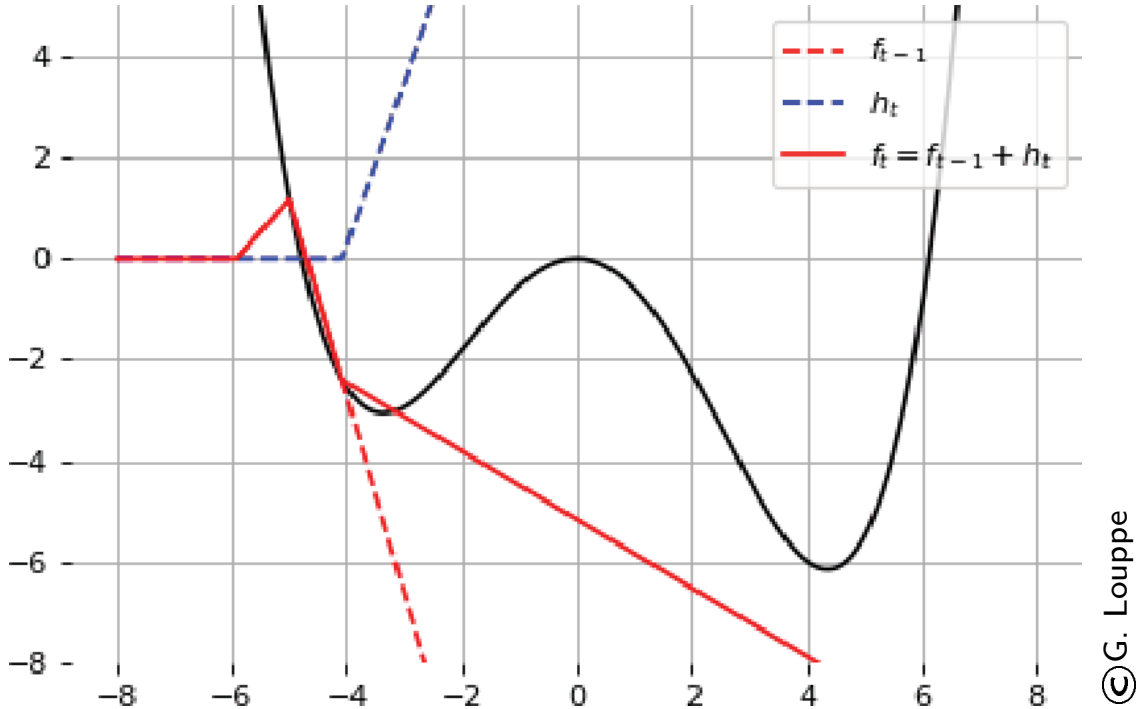




Rectified linear unit (ReLU) activation

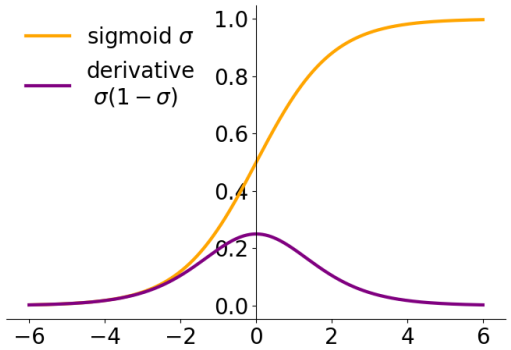
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

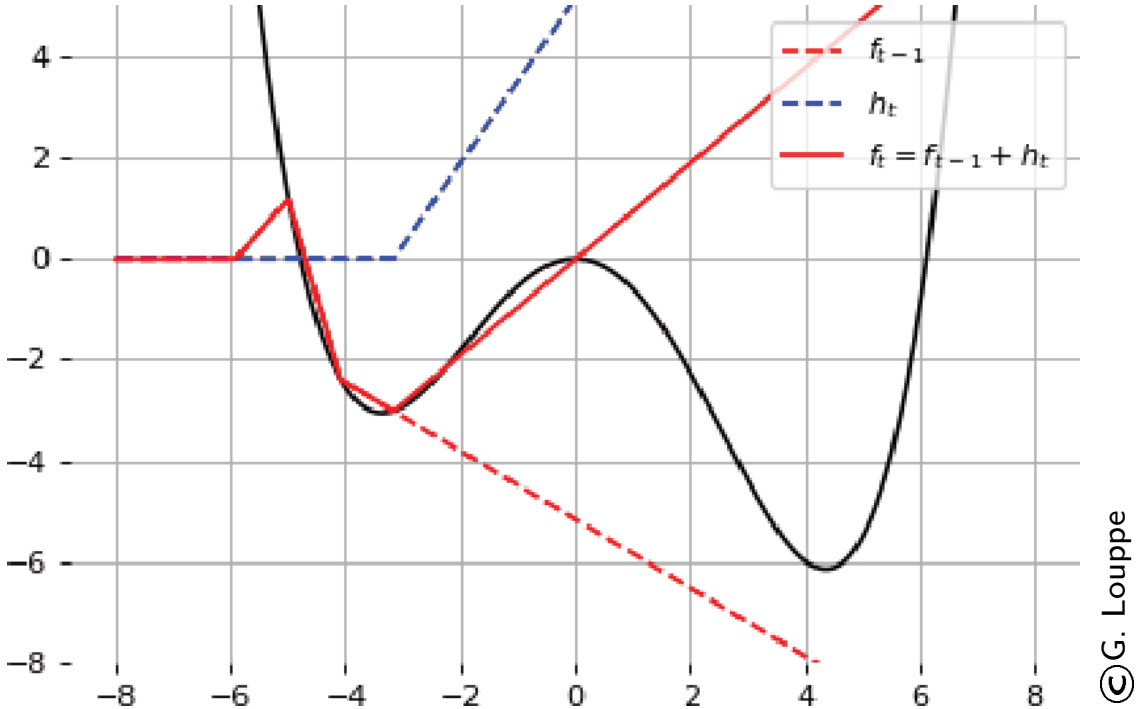




Rectified linear unit (ReLU) activation

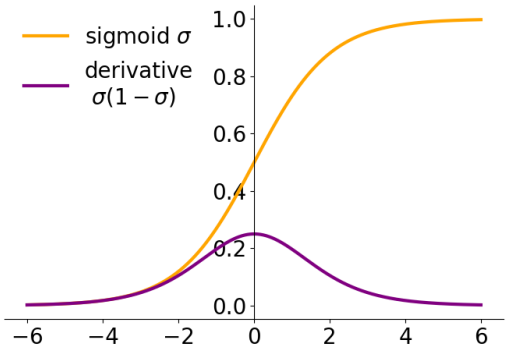
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

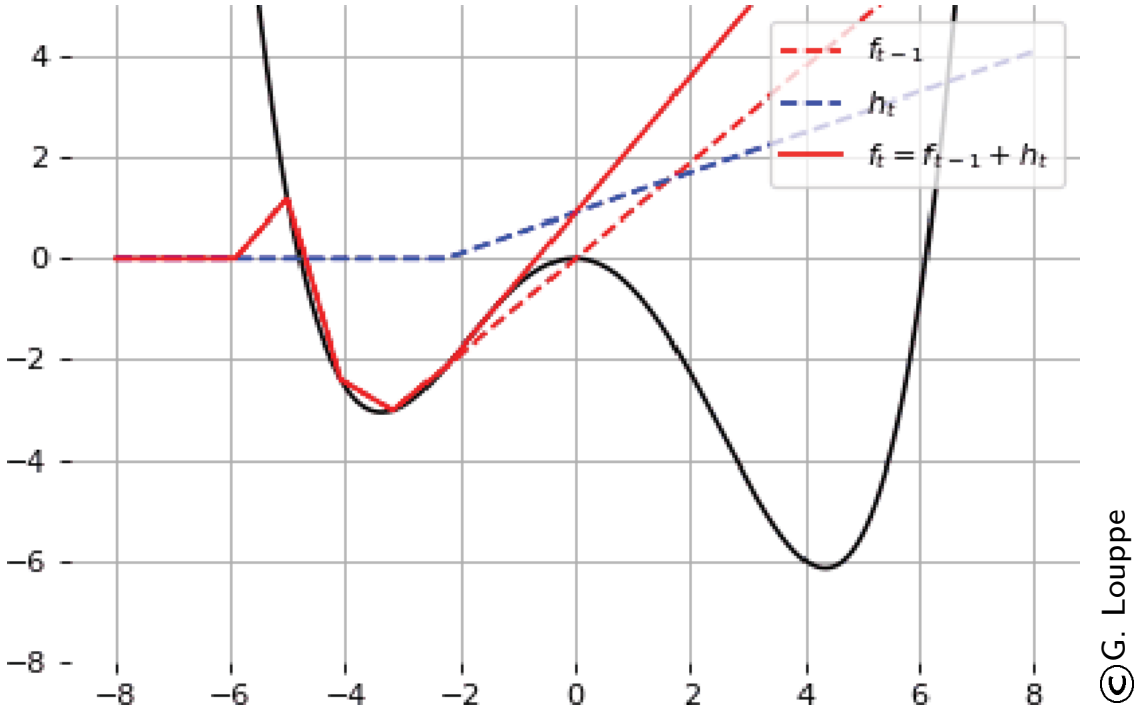




Rectified linear unit (ReLU) activation

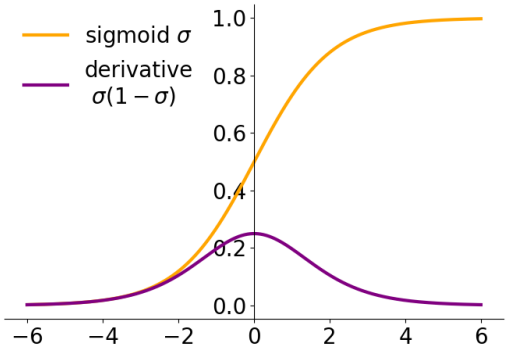
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

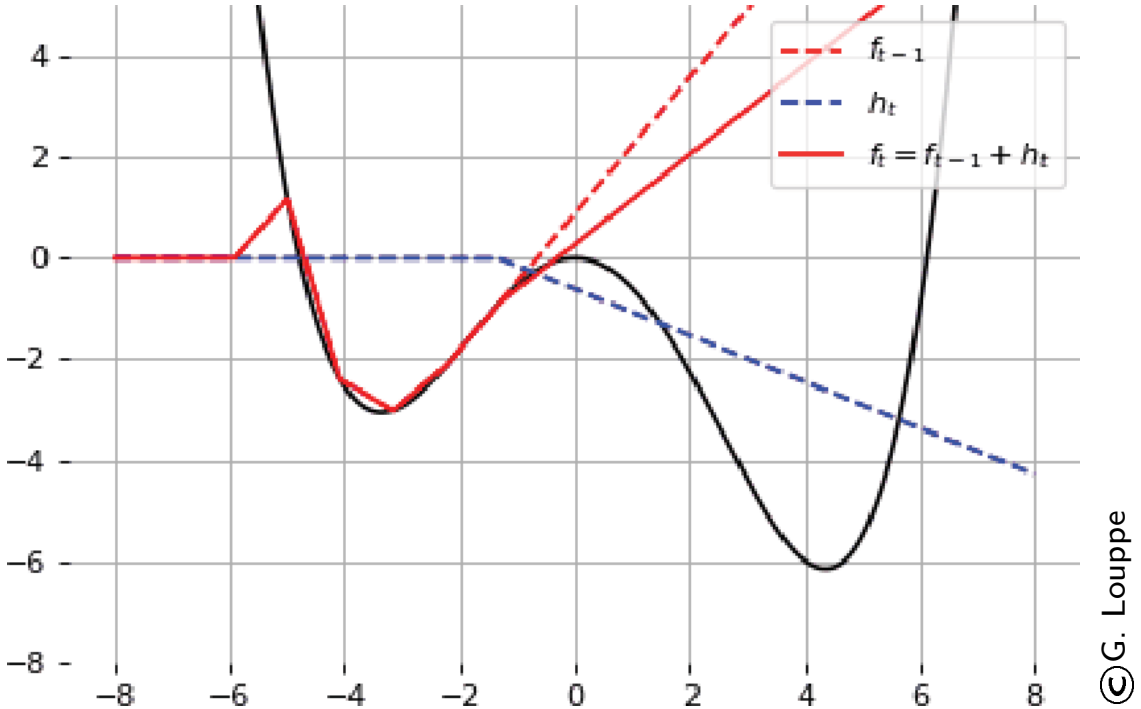




Rectified linear unit (ReLU) activation

- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

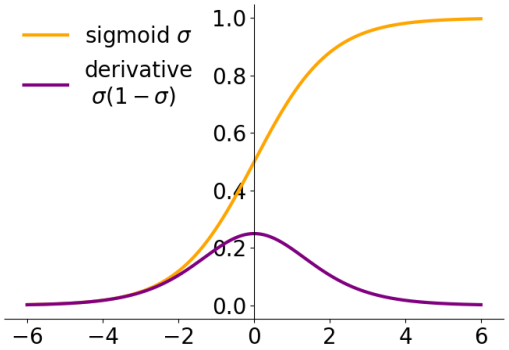


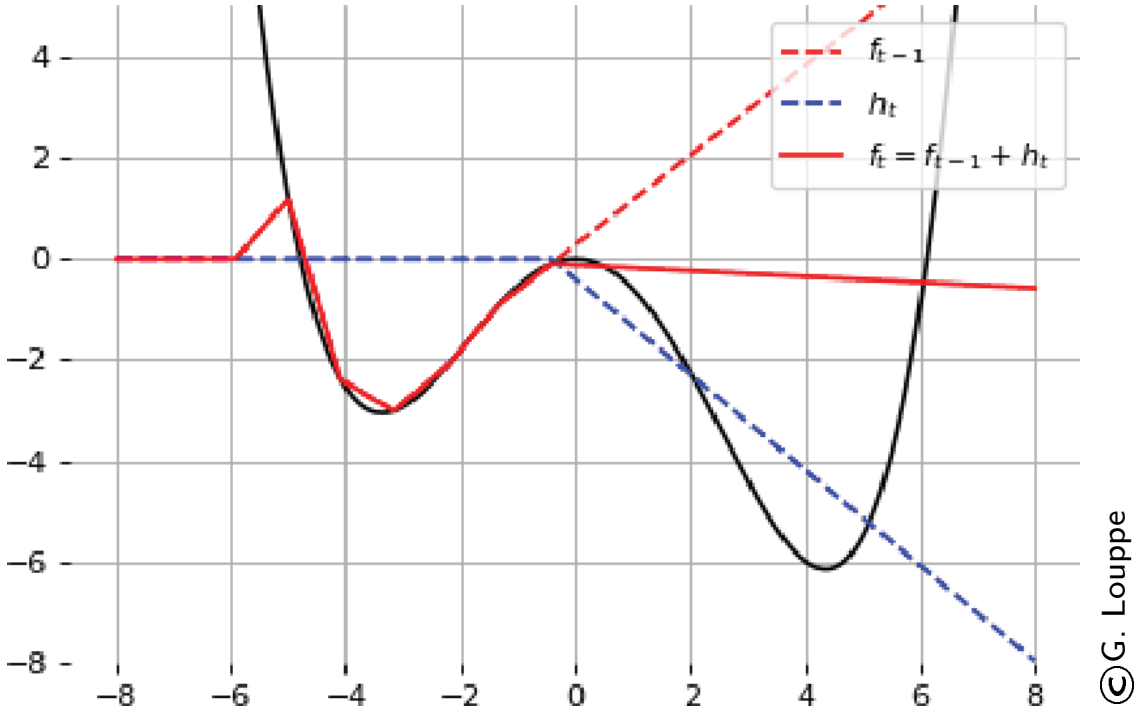


©G. Louppe

Rectified linear unit (ReLU) activation

- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

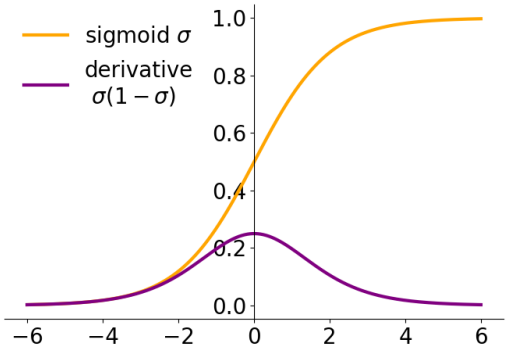


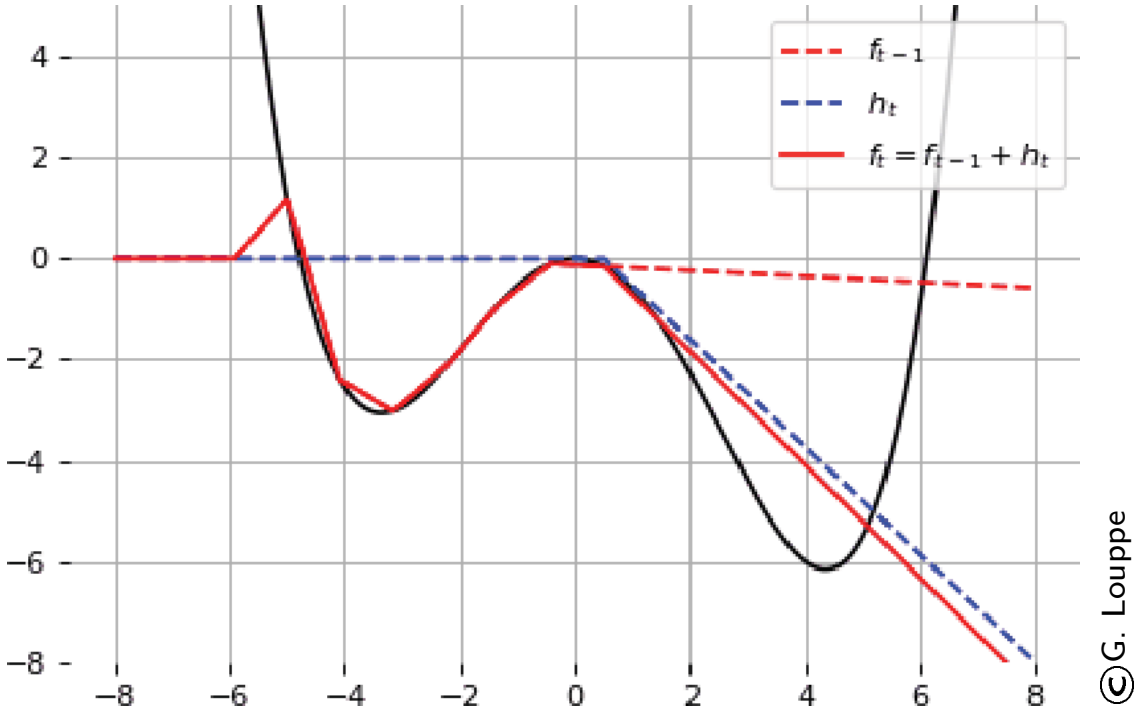


©G. Louppe

Rectified linear unit (ReLU) activation

- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

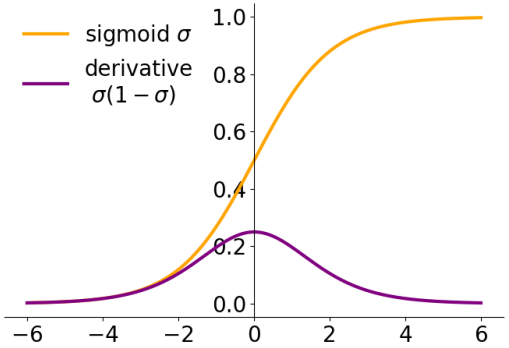


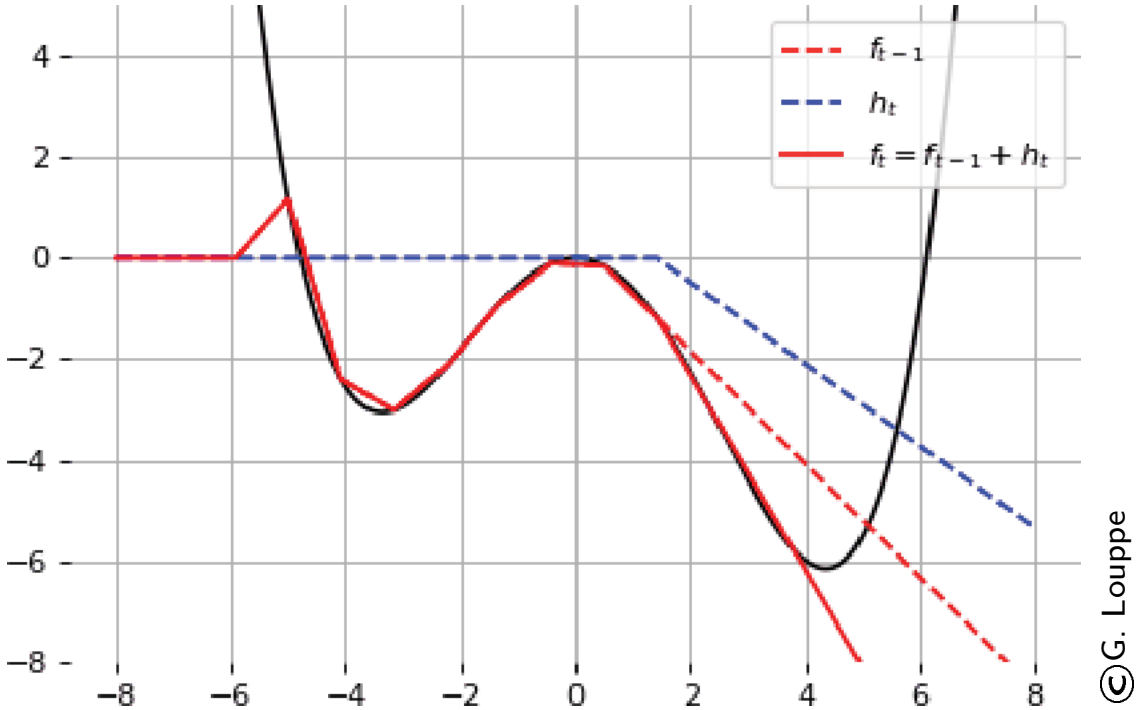


©G. Louppe

Rectified linear unit (ReLU) activation

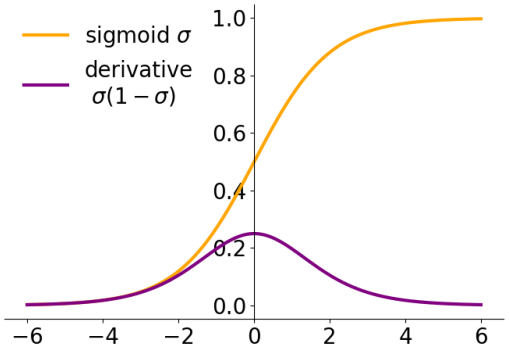
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

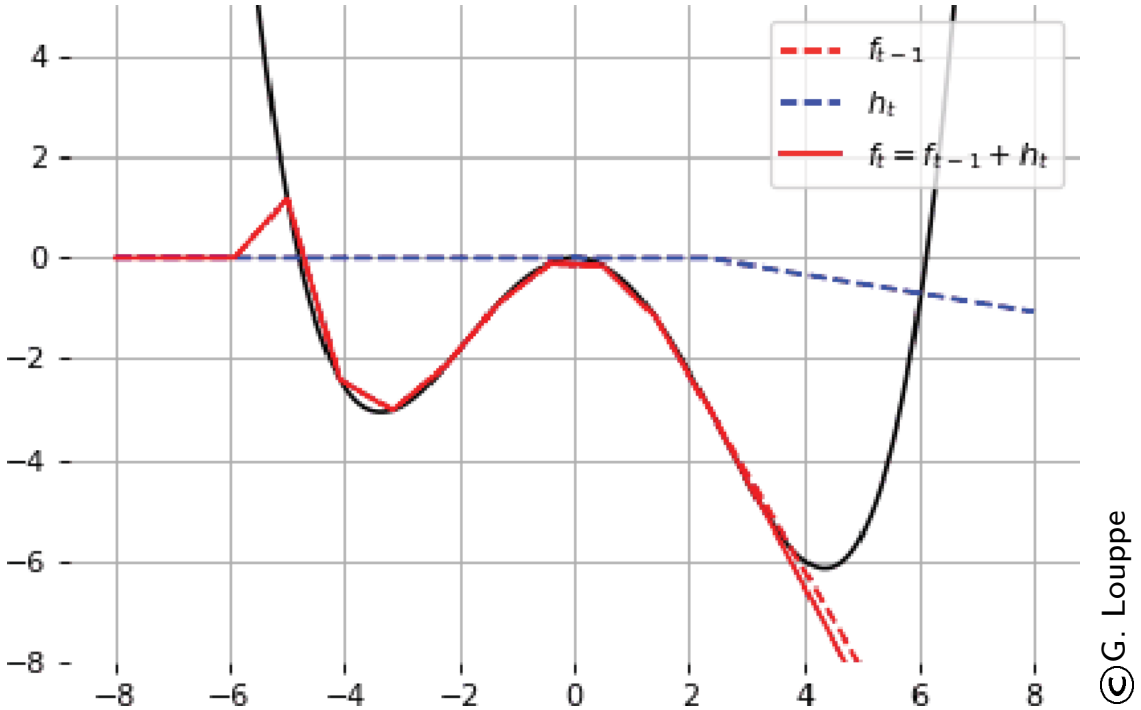




Rectified linear unit (ReLU) activation

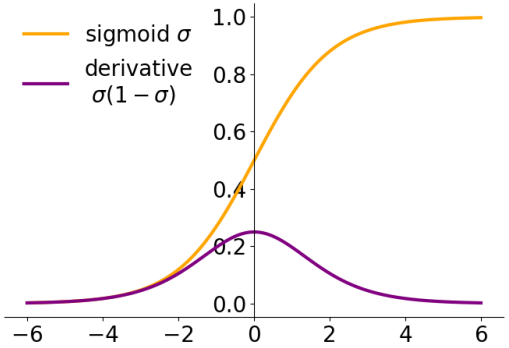
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

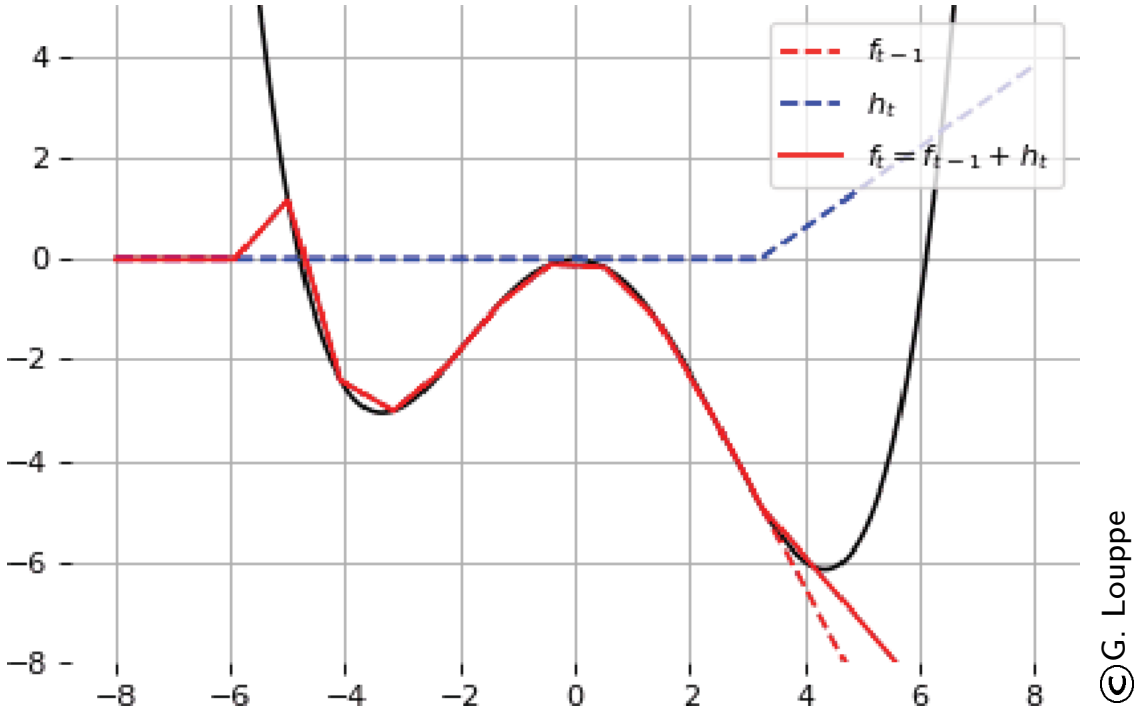




Rectified linear unit (ReLU) activation

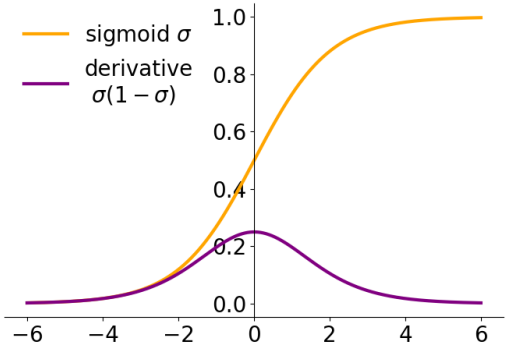
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

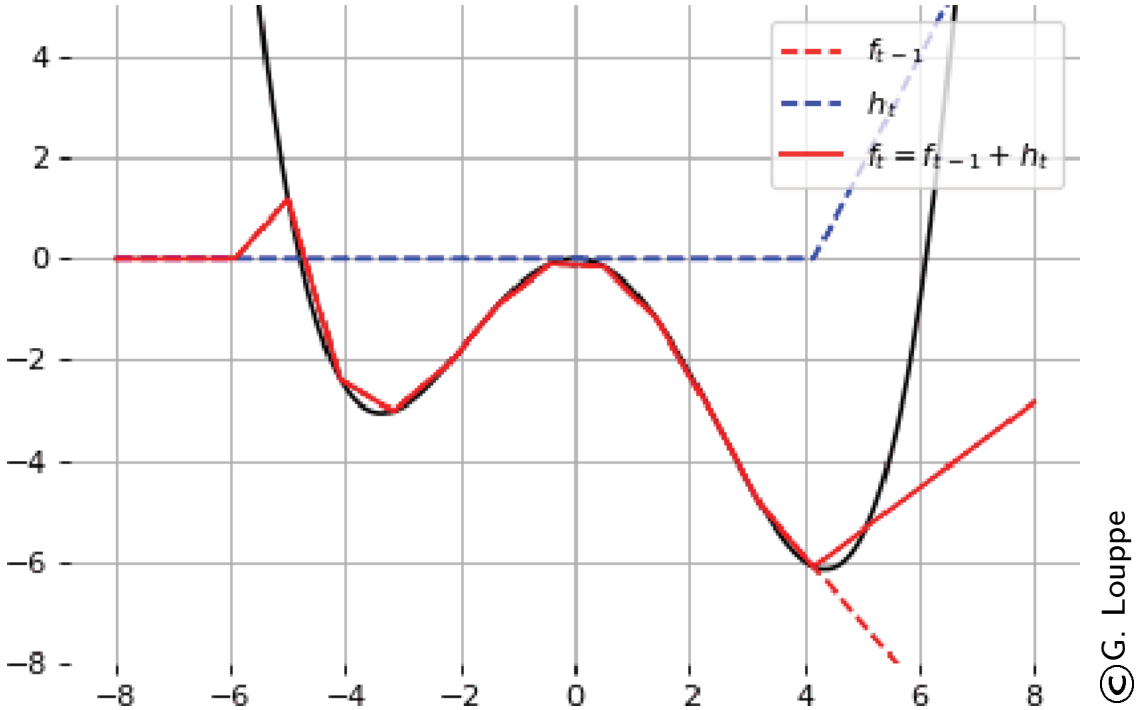




Rectified linear unit (ReLU) activation

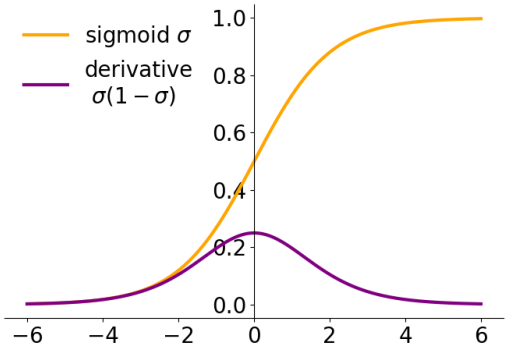
- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.

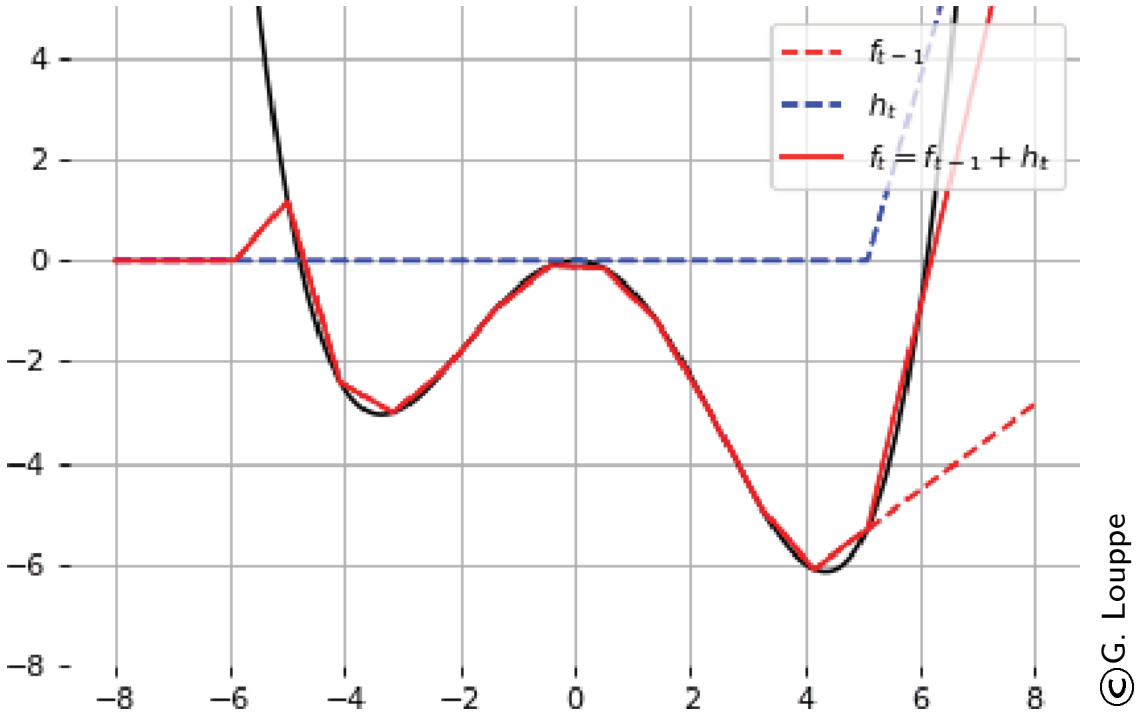




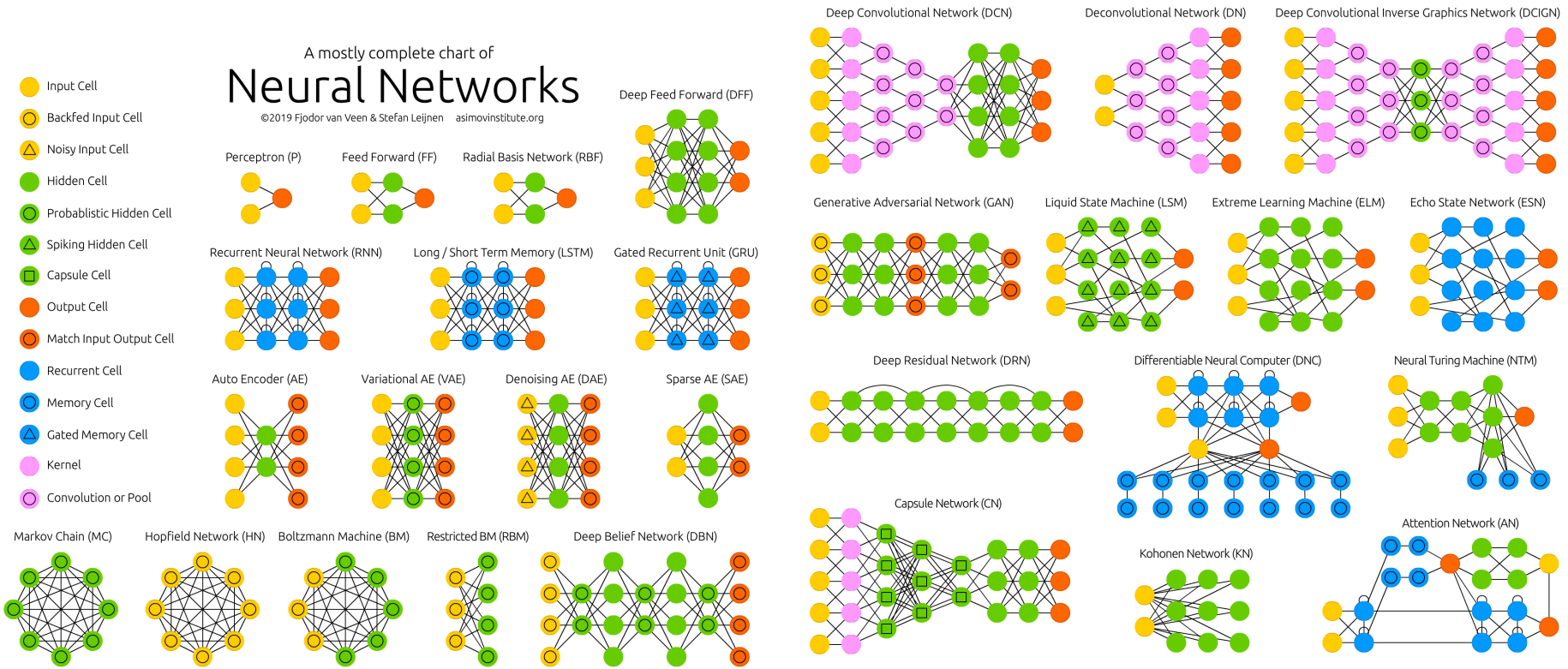
Rectified linear unit (ReLU) activation

- One of reasons for vanishing gradient: sigmoid activation
  - tiny non-varying derivative away from zero
- Solution: non-saturating function
- Simplest case: rectified linear unit ReLU
- Other variants: leaky ReLU, shifted ReLU (SReLU), exponential linear unit (ELU), etc.





Neural network zoo

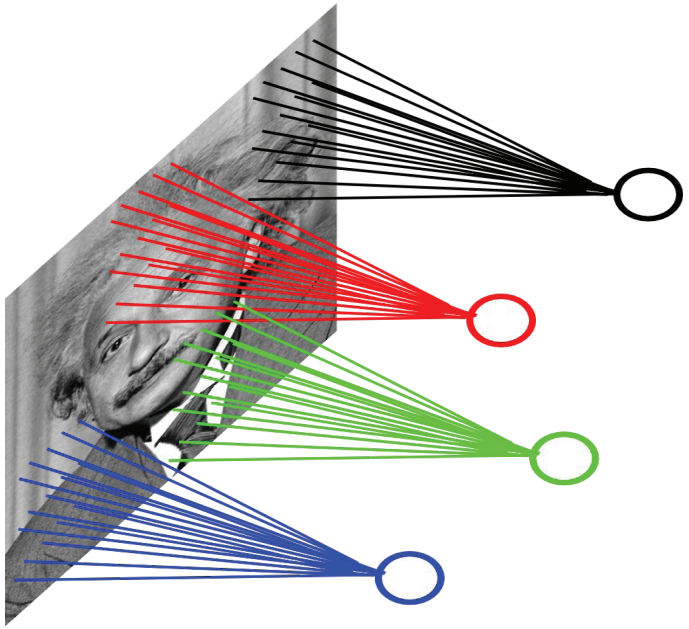


▸ https://www.asimovinstitute.org/

- Many possible network structures
- Moving away from feature engineering to model design

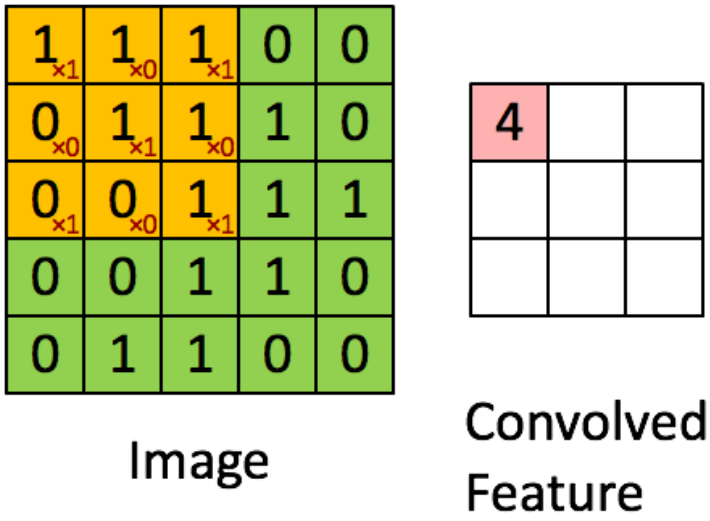
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another



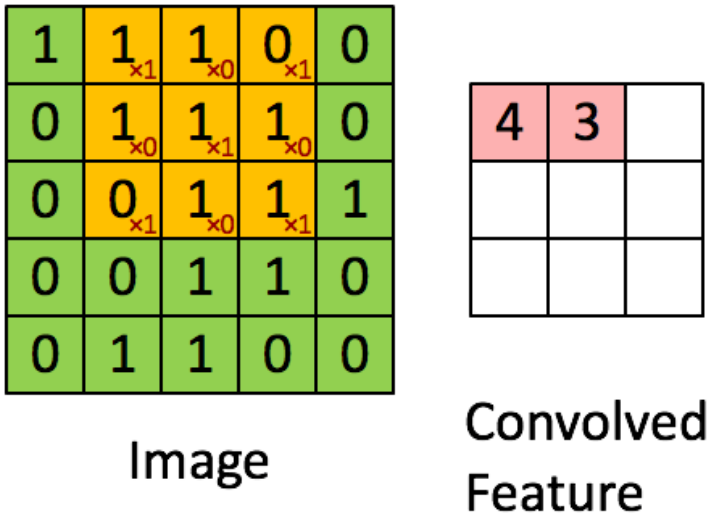
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



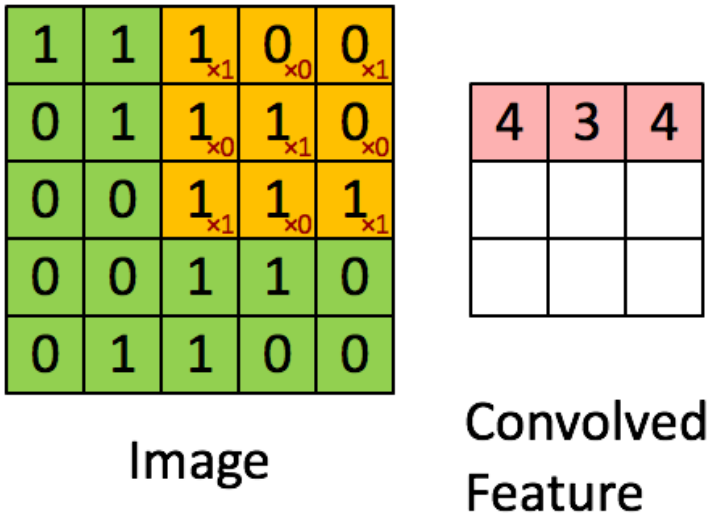
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



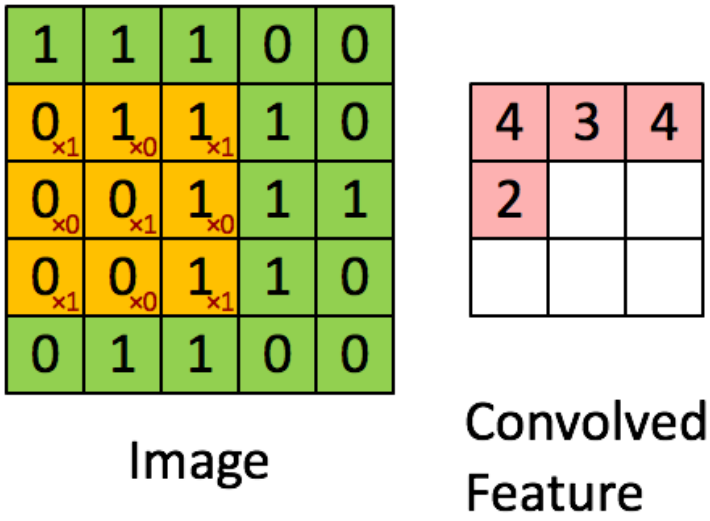
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



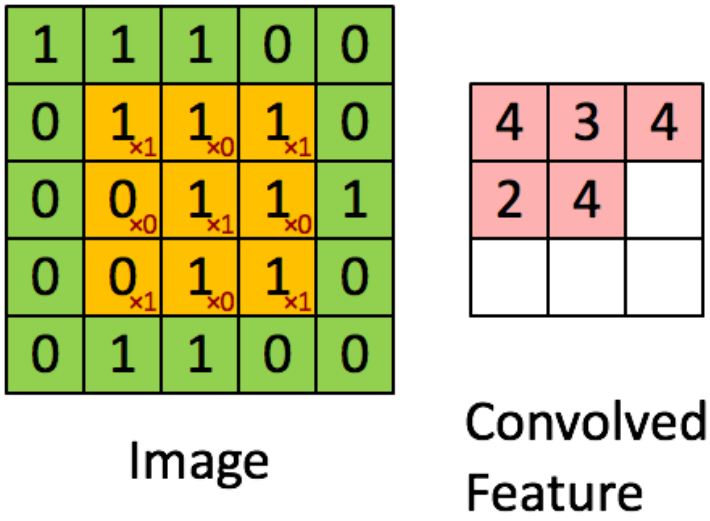
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



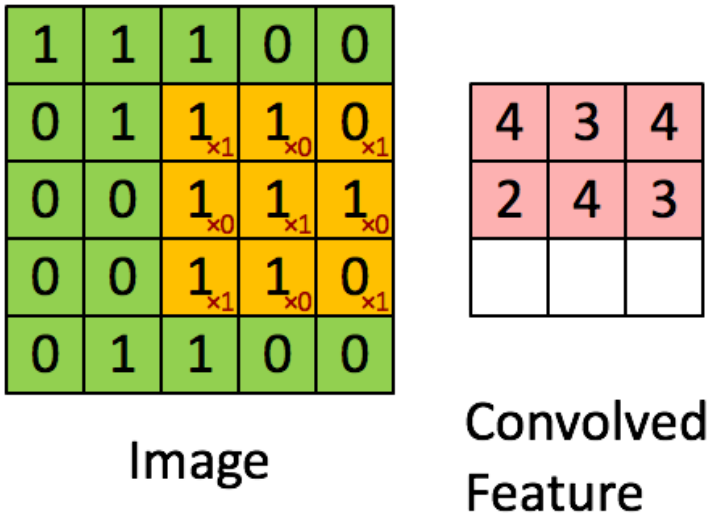
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



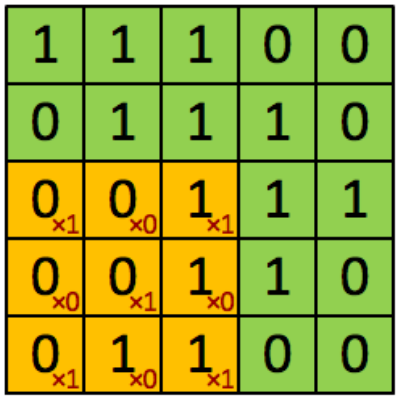
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image

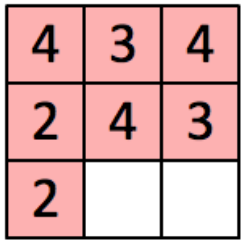


Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



Image



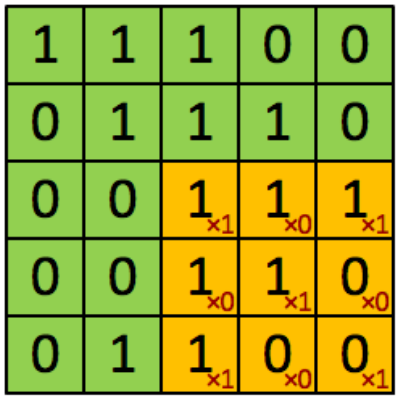
Convolved Feature

Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image

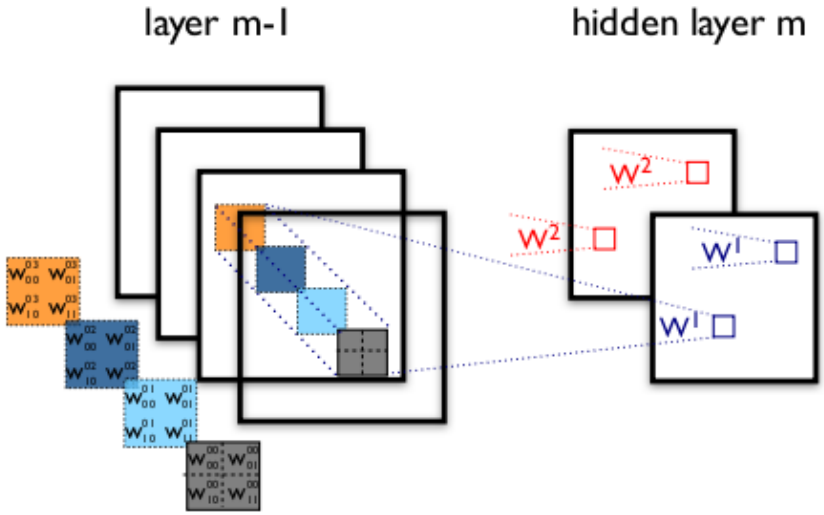
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



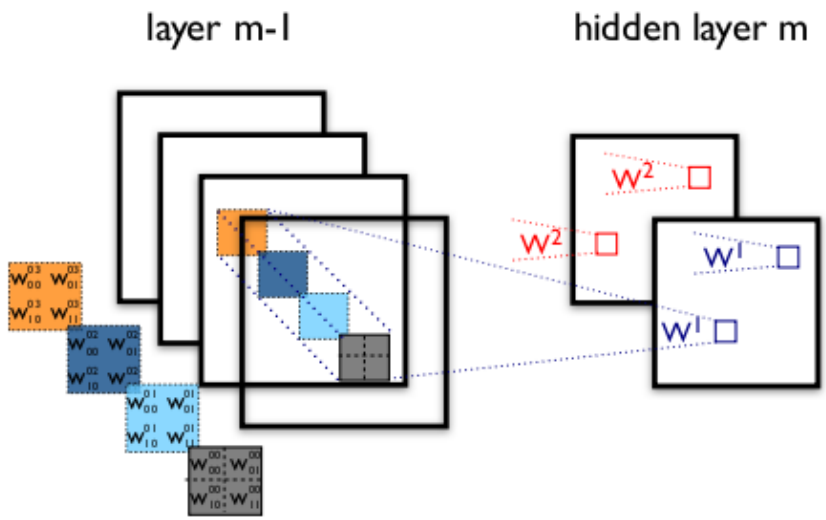
Convolutional networks (CNN)

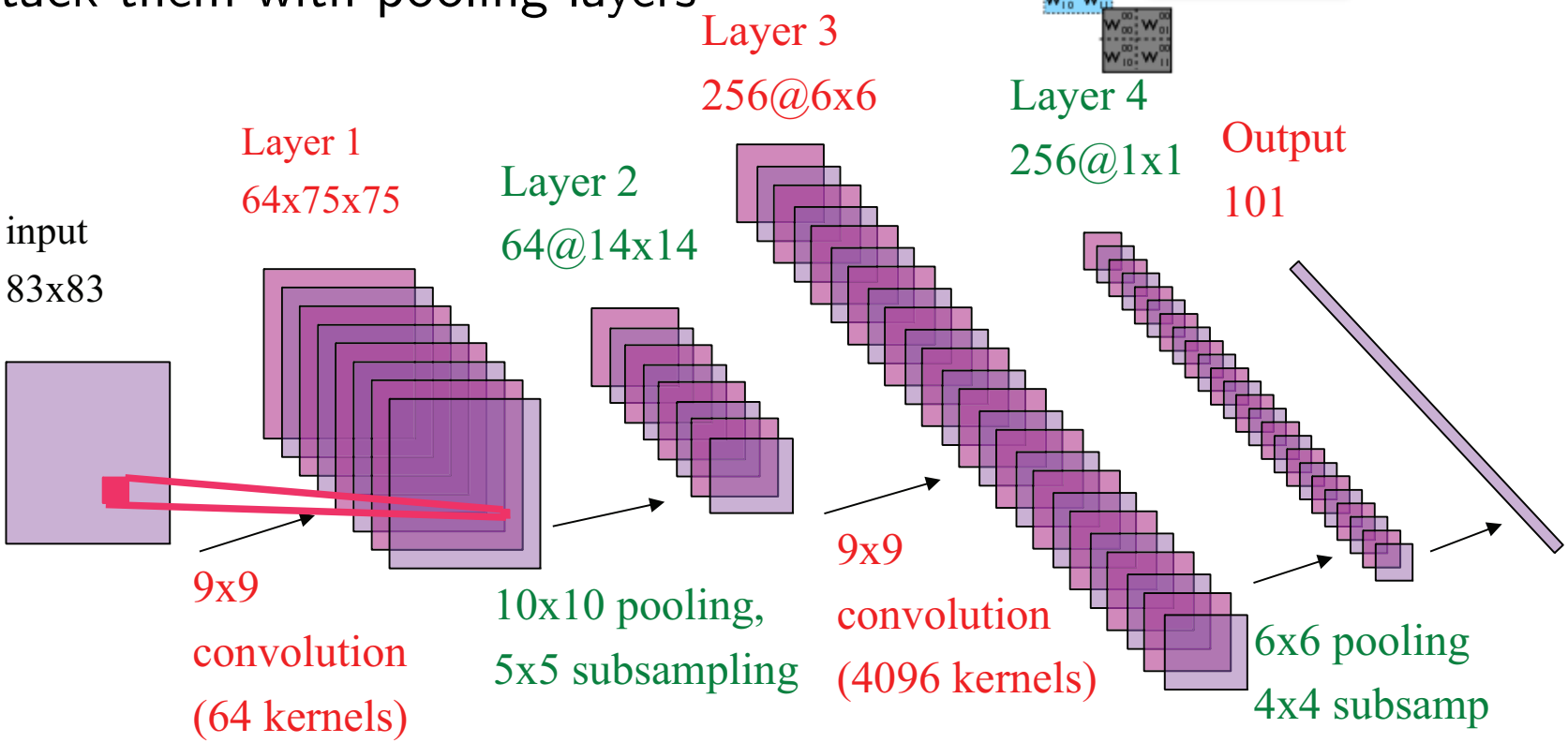
- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image
- Build several "feature maps"



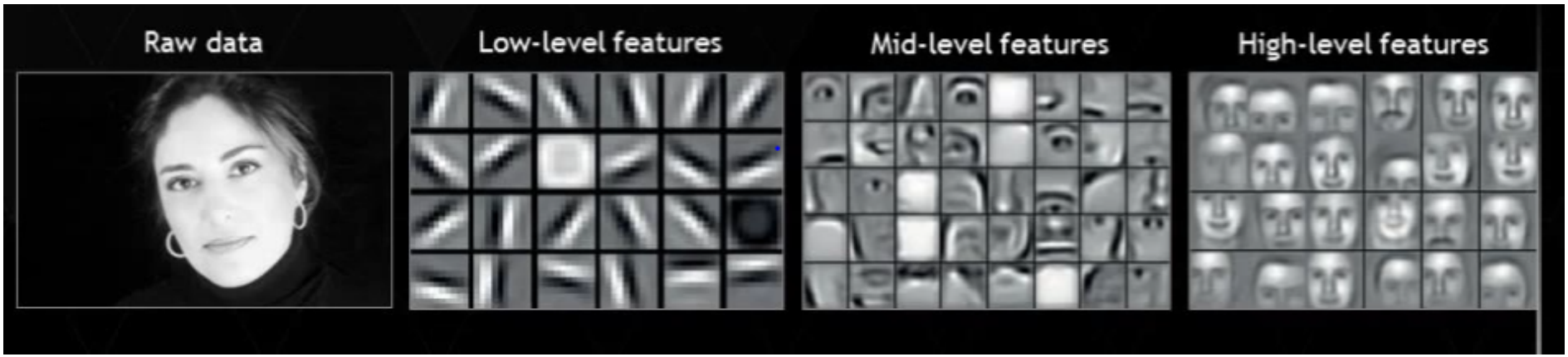
Convolutional networks (CNN)

- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image
- Build several "feature maps"
- Stack them with pooling layers





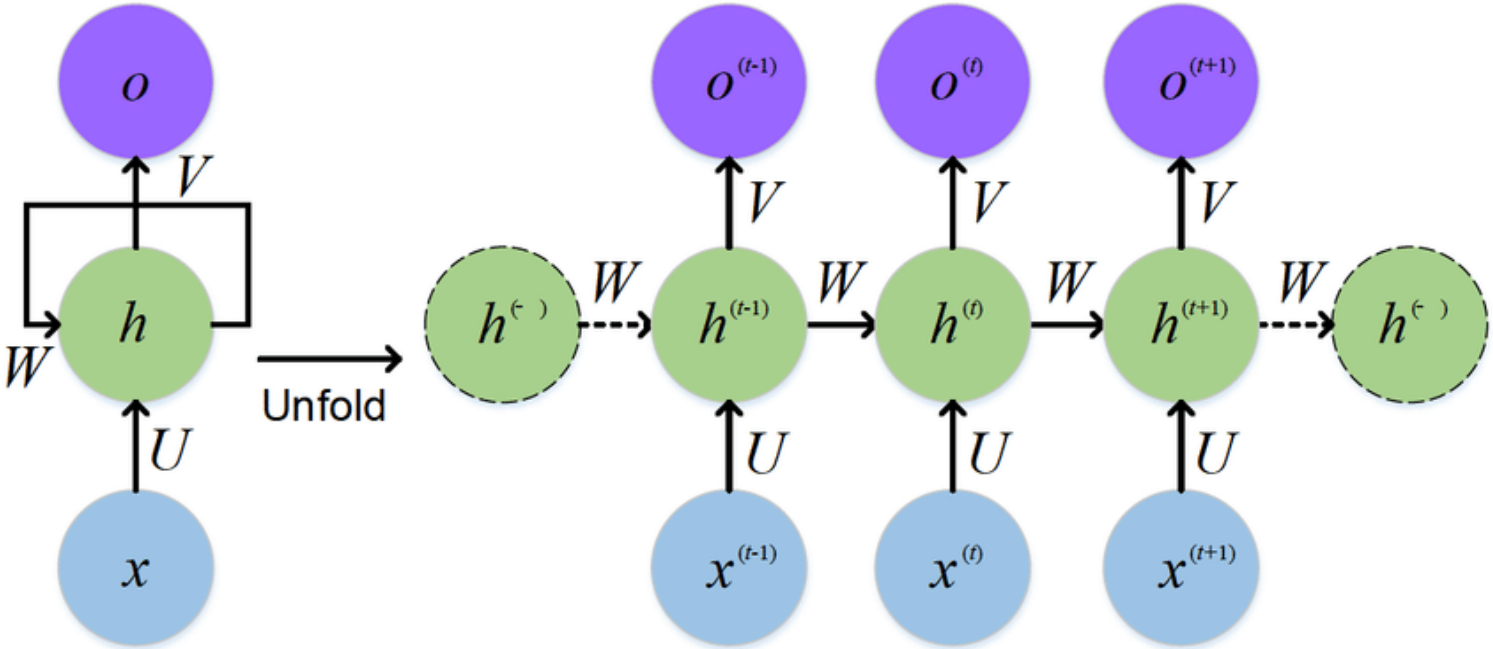
Learning feature hierarchy



Recurrent neural networks (RNN)

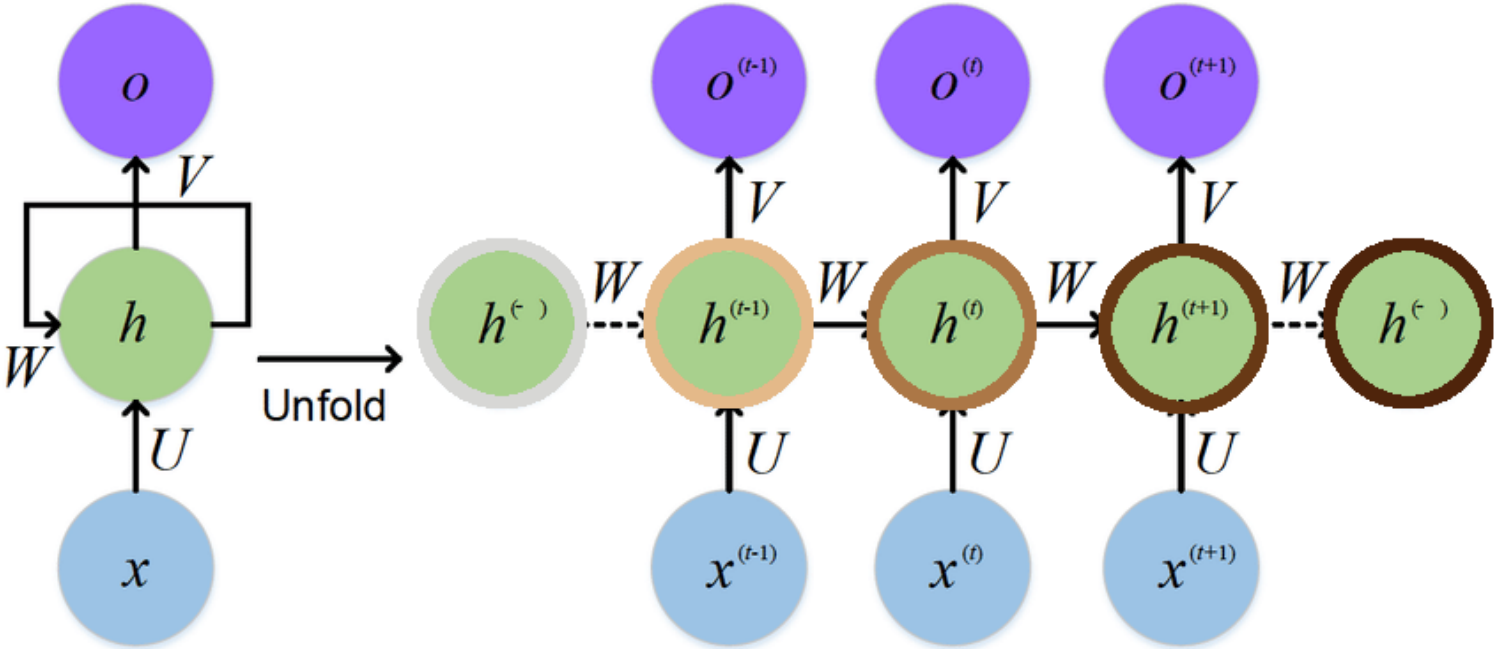
- Many problems require processing a sequence
  - sequence classification
    - text analysis ("sentiment analysis")
    - DNA sequencing
    - action selection
  - sequence synthesis
    - text synthesis
    - music/video
  - sequence translation
    - speech recognition
    - translation
- Usually variable length sequences (number of words/ notes/ frames/ etc.)
- Use a recurrent model, maintaining a recurrent state updated after each step

Recurrent neural networks



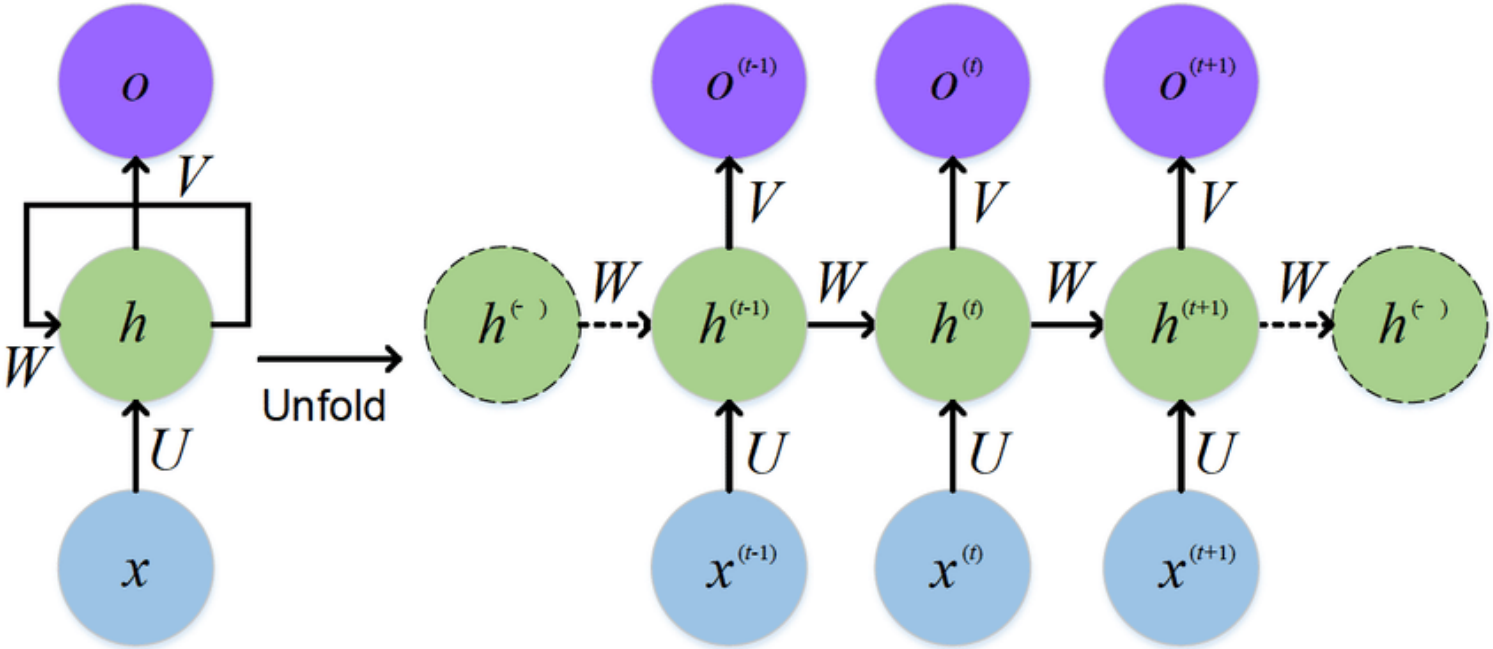
- Keeps information from earlier frames while processing (variable-size) sequence
- Could also be bi-directional, consuming sequence in both directions

Recurrent neural networks



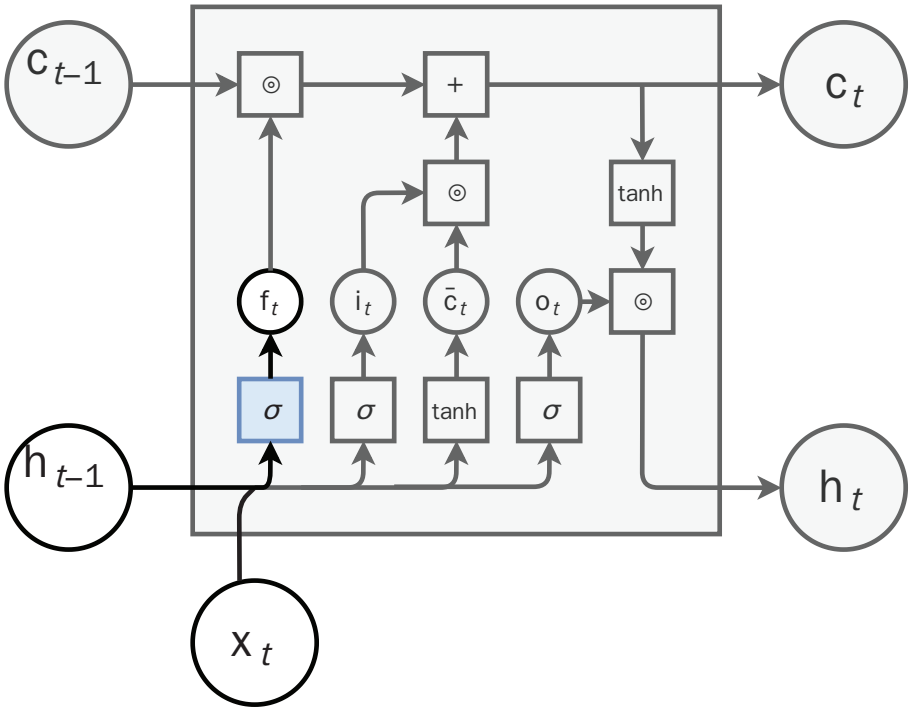
- Keeps information from earlier frames while processing (variable-size) sequence
- Could also be bi-directional, consuming sequence in both directions
- Issue: early frames diluted over sequence $\Rightarrow$ memory loss

Recurrent neural networks



- Keeps information from earlier frames while processing (variable-size) sequence
- Could also be bi-directional, consuming sequence in both directions
- Issue: early frames diluted over sequence $\Rightarrow$ memory loss
- Introducing long short-term memory (LSTM) networks
  - using forget gate to regulate information flow
  - also possible with gated recurrent units (GRU)

Long short-term memory (LSTM)

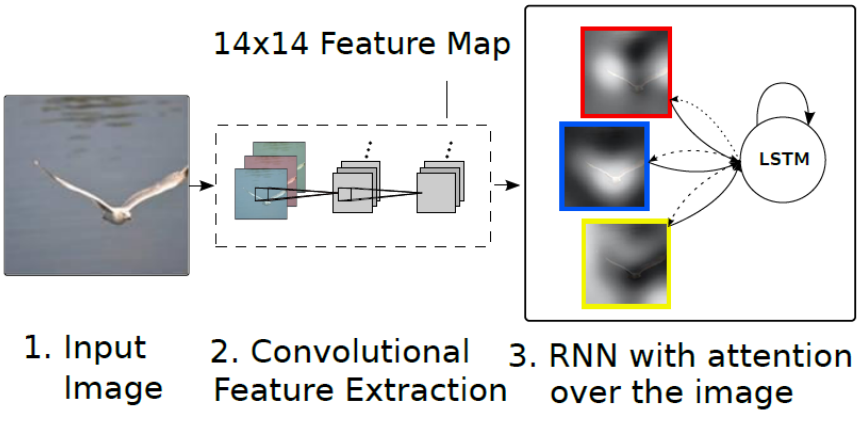


- Recurrent state split in two parts
  - cell state c_t
  - output state h_t
- Forget gate f_t to erase cell state info
- Input gate i_t to update cell state info
- Output gate o_t to select output state

Recurrent neural networks examples

Labelling images

▸ arXiv:1502.03044

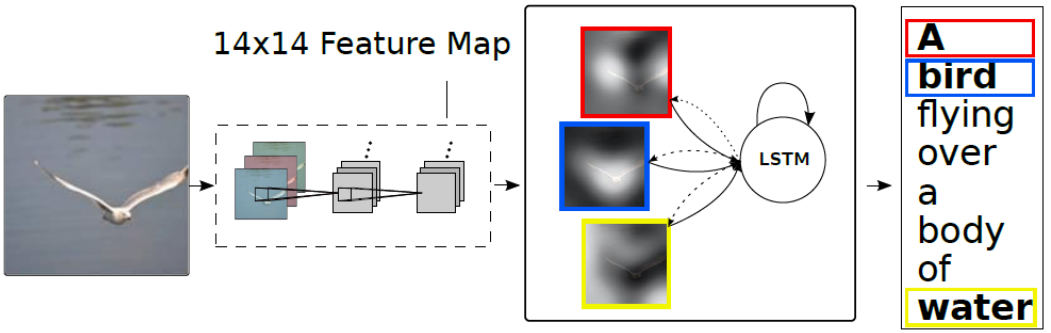


A woman is throwing a frisbee in a park.

Recurrent neural networks examples

Labelling images

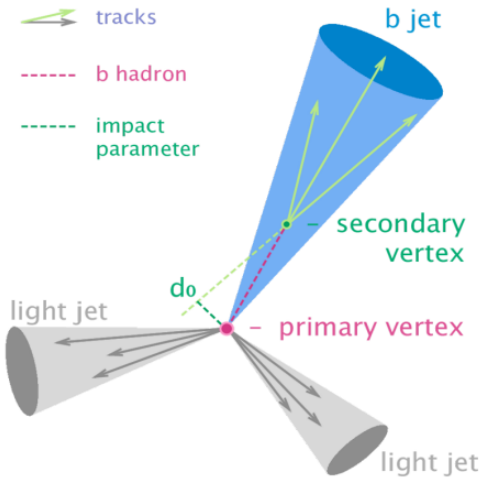
arXiv:1502.03044



A woman is throwing a frisbee in a park.

b-jet tagging in ATLAS experiment

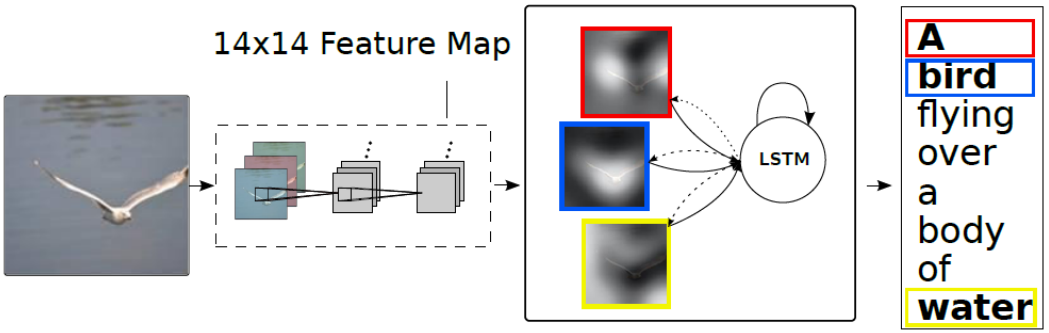
ATL-PHYS-PUB-2017-003



Recurrent neural networks examples

Labelling images

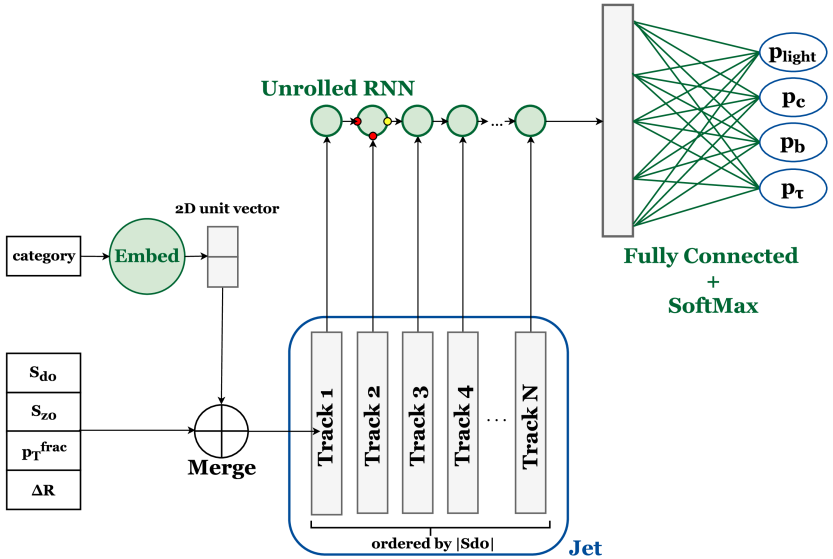
▸ arXiv:1502.03044

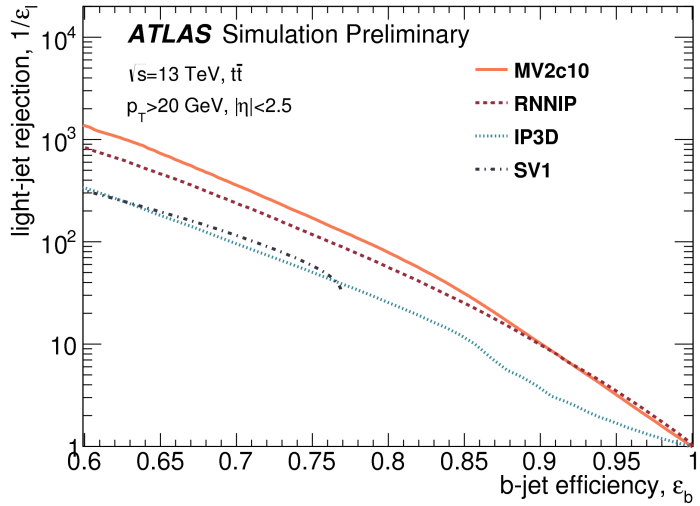


A woman is throwing a frisbee in a park.

b-jet tagging in ATLAS experiment

▸ ATL-PHYS-PUB-2017-003





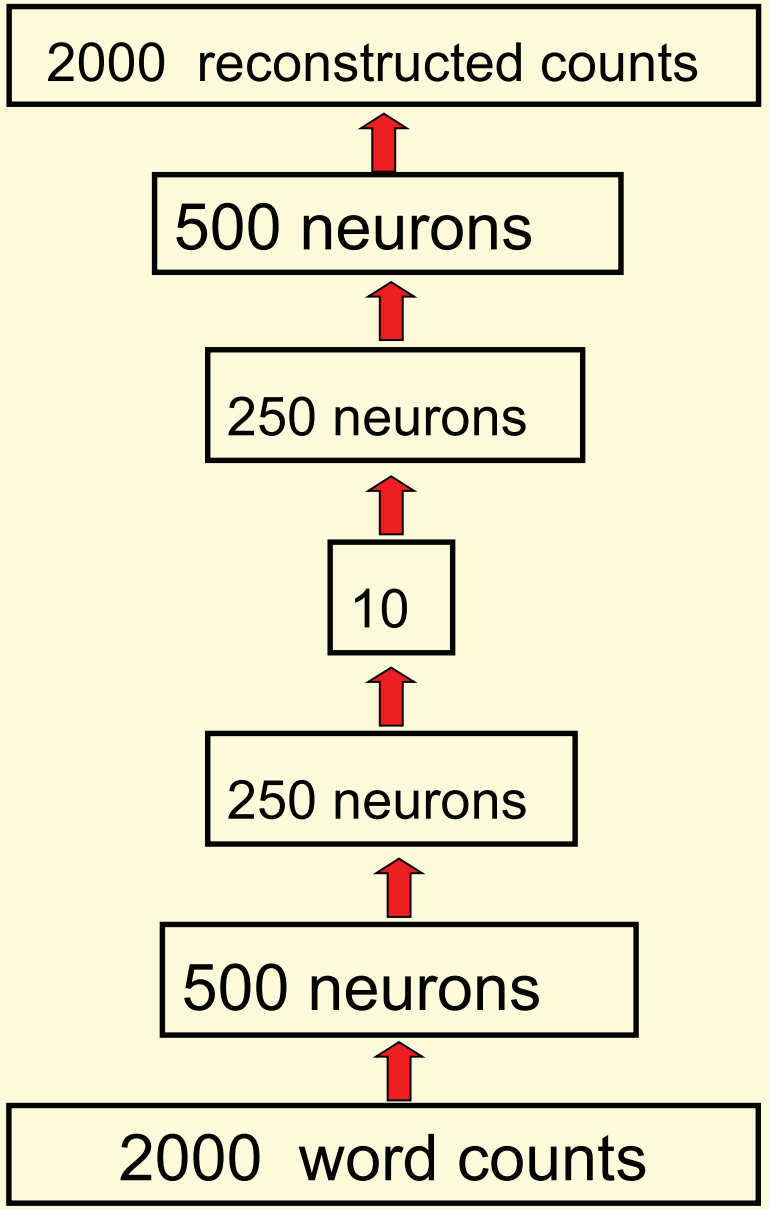
Auto-encoders

Approximate the identity function

- Build a network whose output is similar to its input
- Sounds trivial? Except if imposing constraints on network (e.g., # of neurons, locally connected network) to discover interesting structures
- Can be viewed as lossy compression of input

Finding similar books

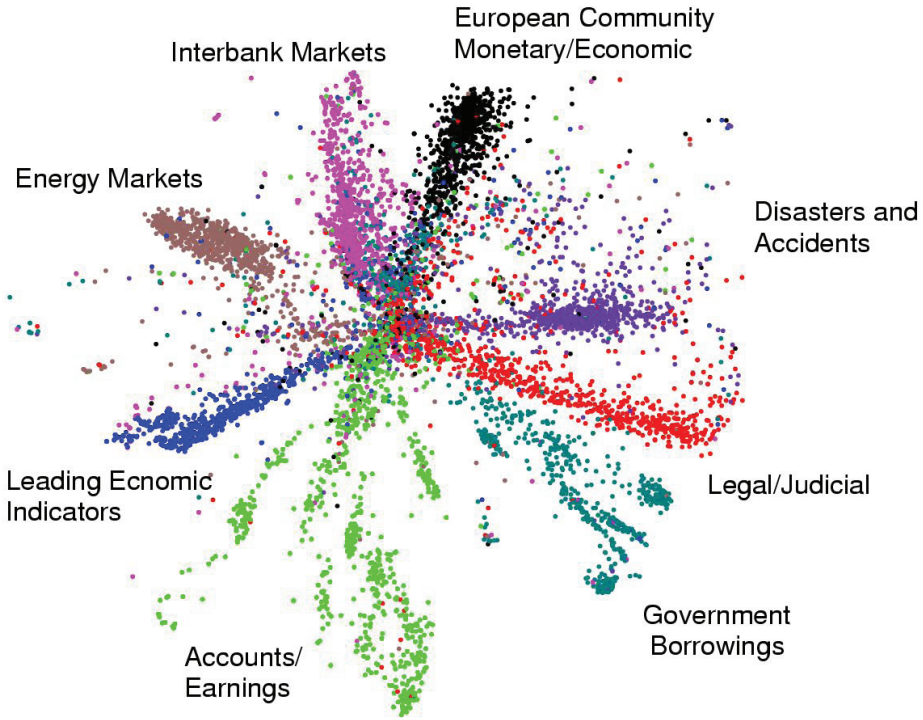
- Get count of 2000 most common words per book
- "Compress" to 10 numbers



Auto-encoders

With principle component analysis (PCA)

With autoencoder



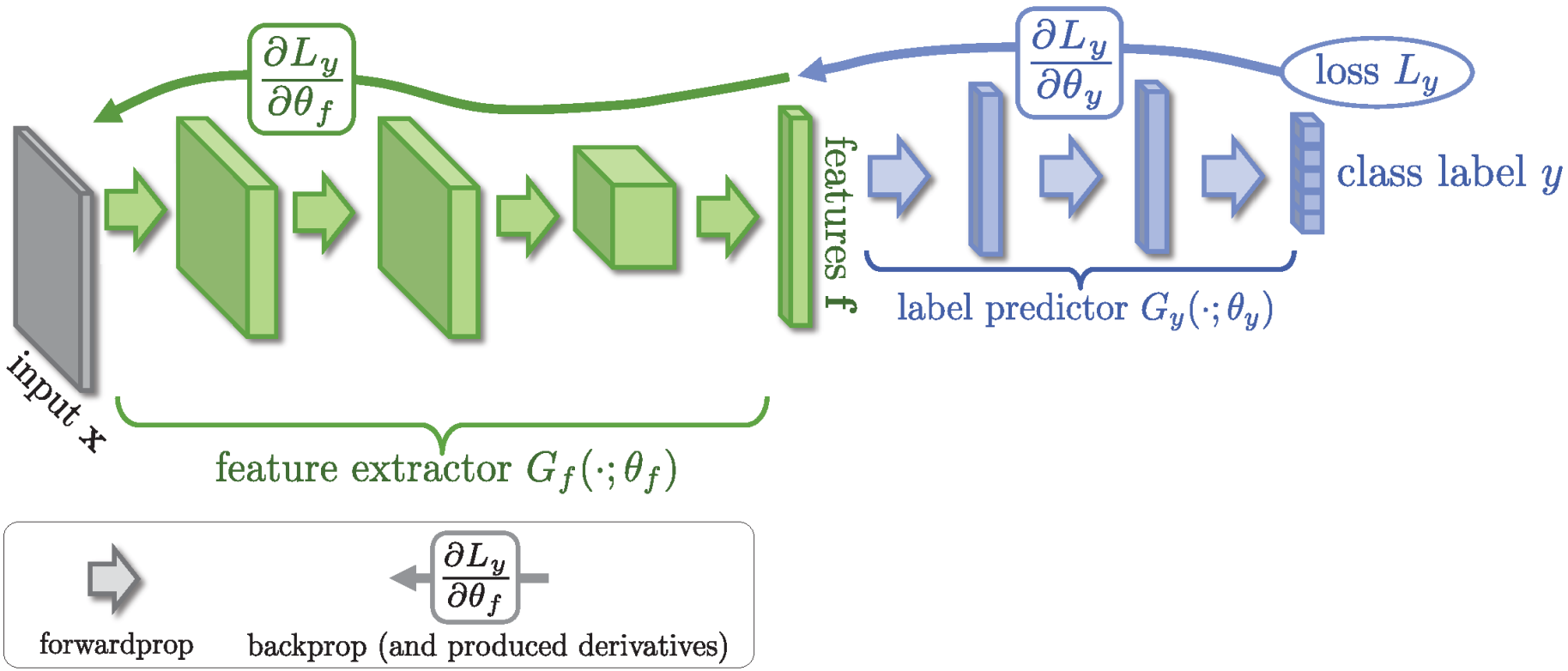
▸ more in backup

Domain adaptation and adversarial training

- Typical training
  - signal and background from simulation
  - results compared to real data to make measurement
- Requires good data–simulation agreement

arXiv:1409.7495

arXiv:1505.07818

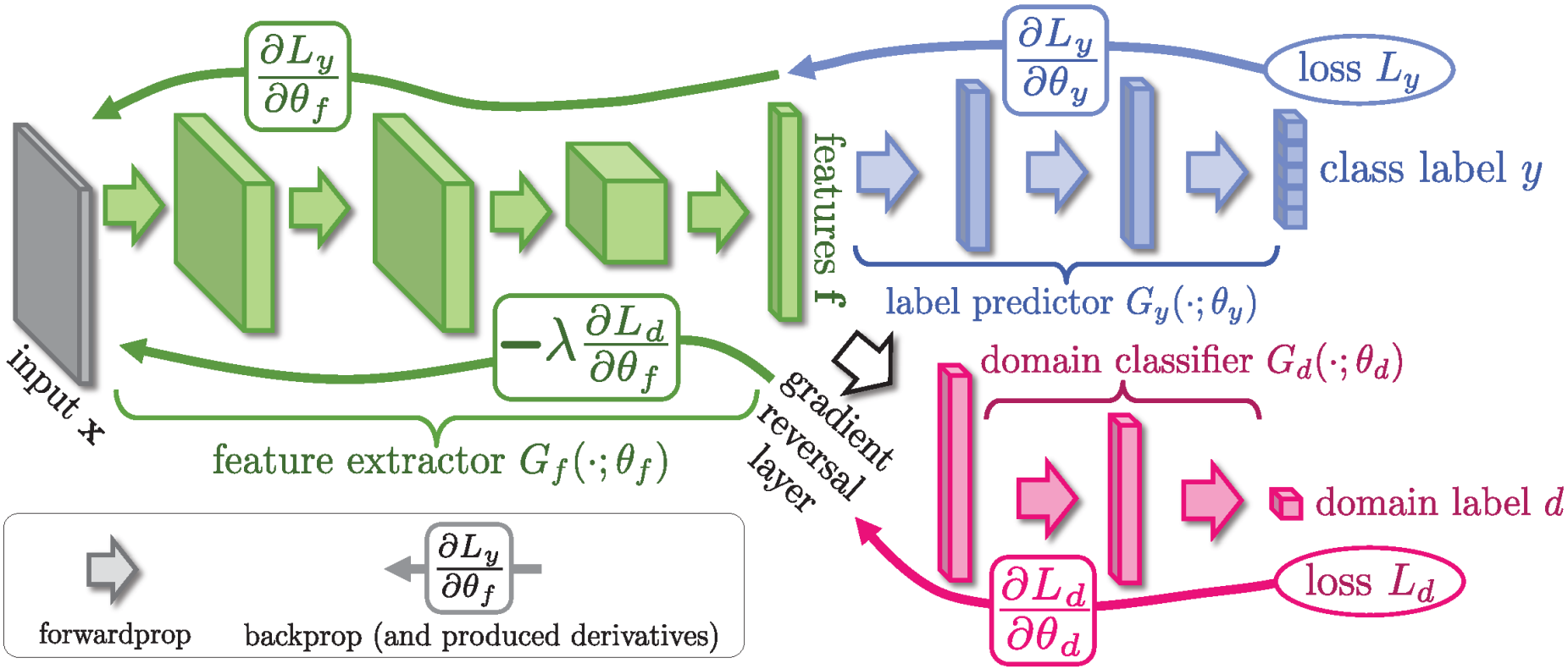


Domain adaptation and adversarial training

- Typical training
  - signal and background from simulation
  - results compared to real data to make measurement
- Requires good data–simulation agreement
- Possibility to use adversarial training and domain adaptation to account for discrepancies/systematic uncertainties

arXiv:1409.7495

arXiv:1505.07818

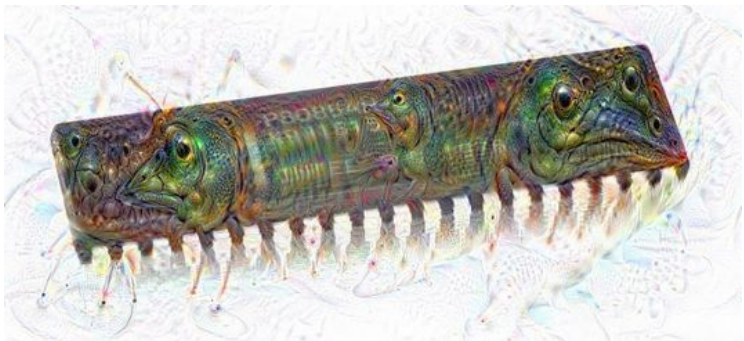


Deep networks and art

- Learning a style ▸ arXiv:1508.06576 [cs.CV] ▸ Neural-style

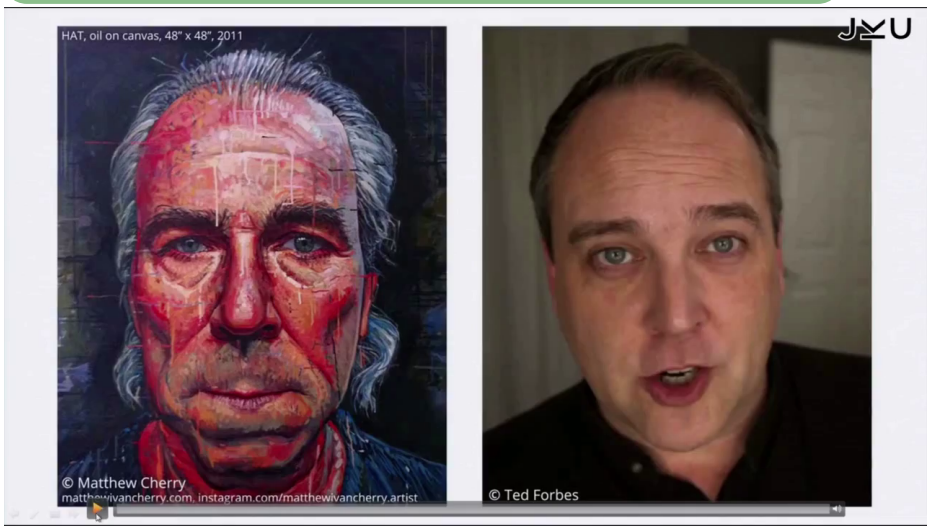
- Computer dreams ▸ Google original

▸ deepdream



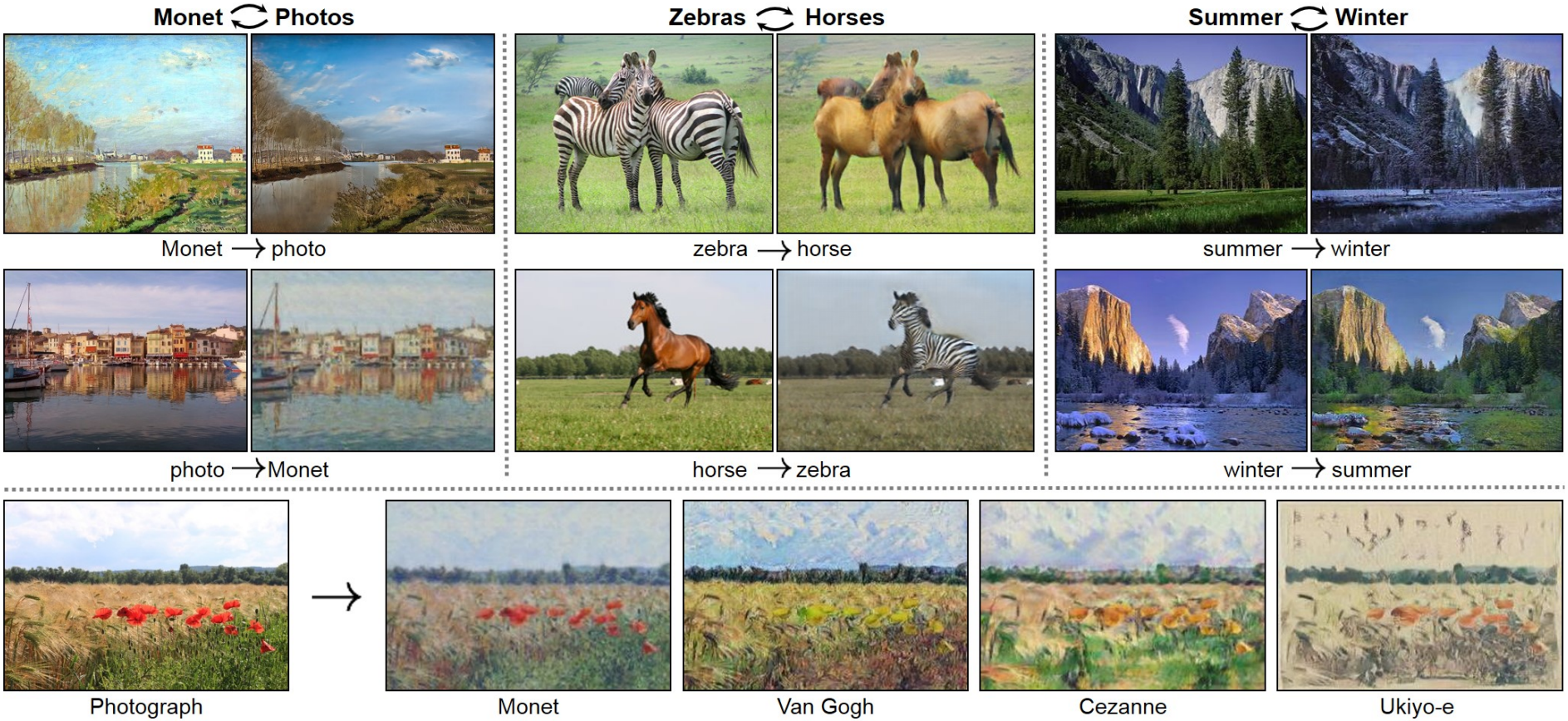
- Face Style ▸ http://facestyle.org

▸ http://dcgi.fel.cvut.cz/home/sykorad/facestyle.html



- GAN = generative adversarial network

Summary

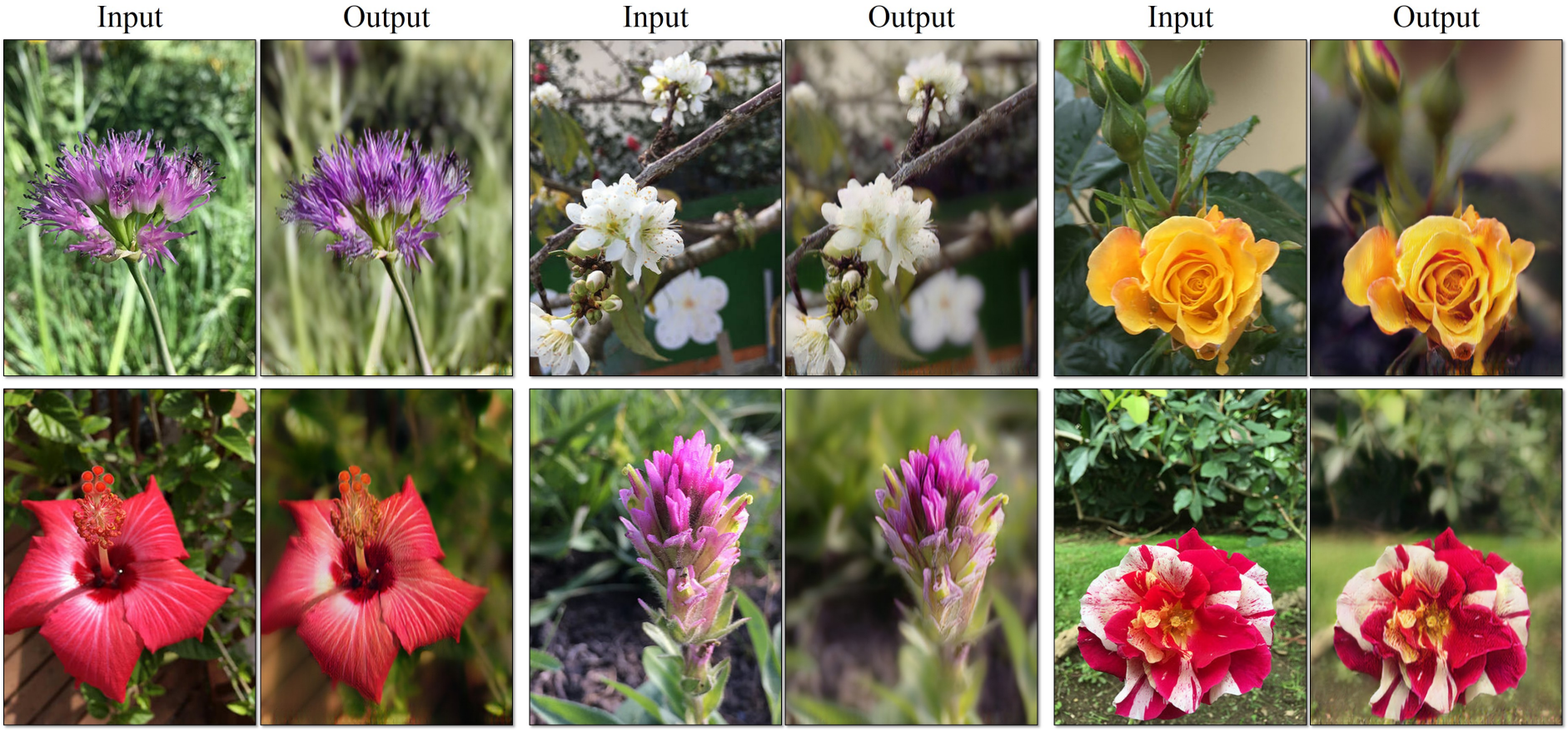


CycleGAN

arXiv:1703.10593 | github

- GAN = generative adversarial network

Photo enhancement



CycleGAN

- GAN = generative adversarial network

Failure 

ImageNet Large Scale Visual Recognition Challenge

- ImageNet: database with 14 million images and 20k categories
- Used 1000 categories and about 1.3 million manually annotated images

PASCAL | ILSVRC


bird


flamingo cock ruffed grouse quail partridge . . .

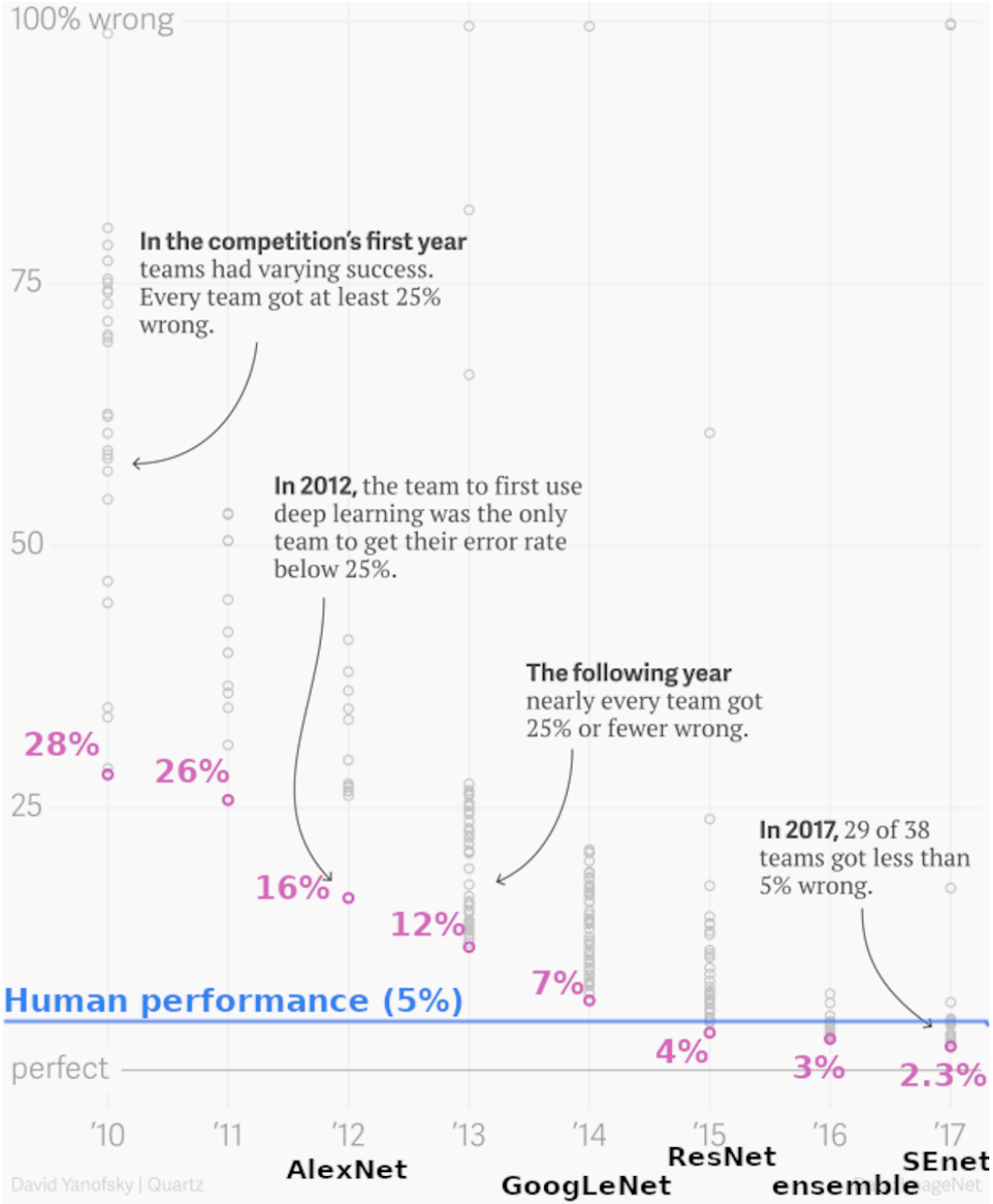

cat


Egyptian cat Persian cat Siamese cat tabby lynx . . .

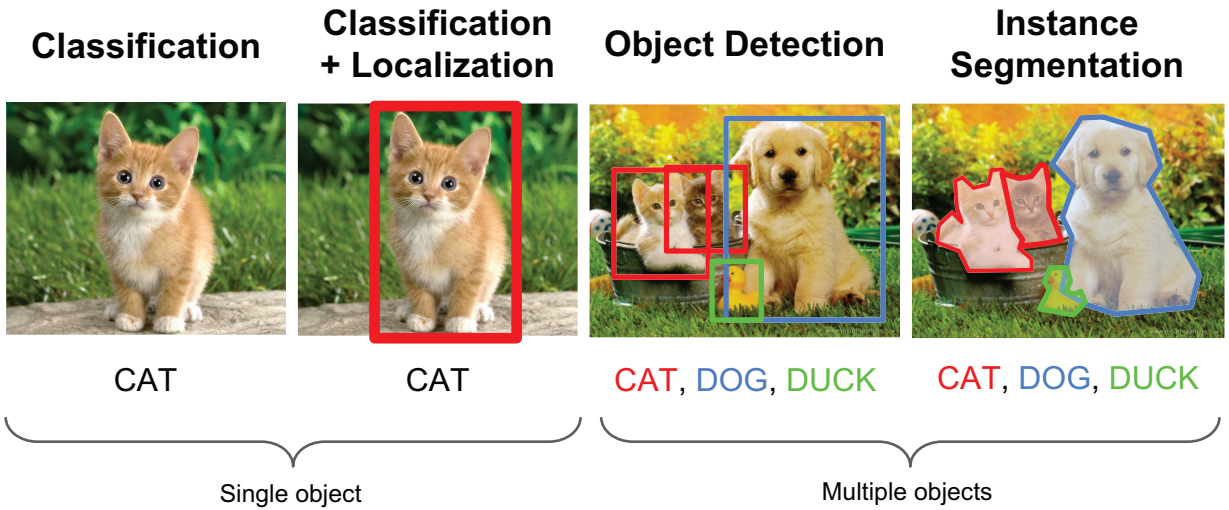

dog


dalmatian keeshond miniature schnauzer standard schnauzer giant schnauzer . . .

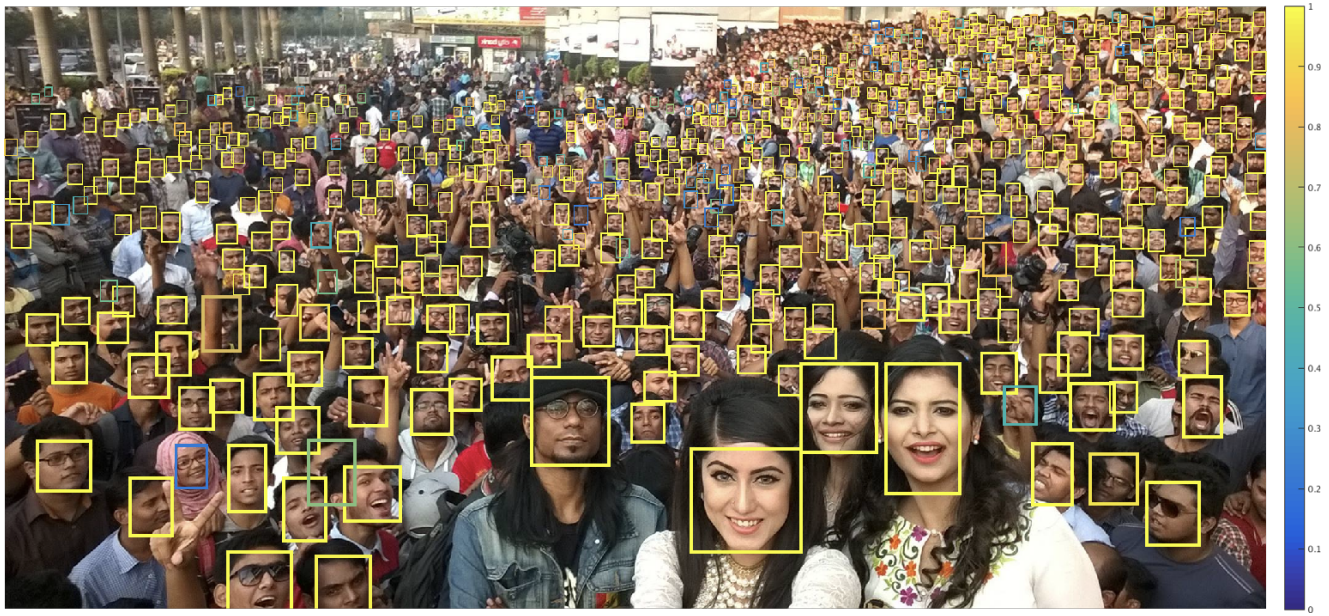
ILSVRC classification performance



Increasing refinement



- More and more refinement (segmentation)
- More objects, in real time on video1/video2/video3



Google DeepMind: arcade games

Nature 518, 529 (2015)

- Learning to play 49 different Atari 2600 games
- No knowledge of the goals/rules, just 84x84 pixel frames
- 60 frames per second, 50 million frames (38 days of game experience)
- Deep convolutional network with reinforcement: DQN (deep Q-network)
  - action-value function $Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\, a_t = a,\, \pi\right]$
  - maximum sum of rewards r_t discounted by γ at each timestep t, achievable by a behaviour policy $\pi = P(a|s)$, after making observation s and taking action a
- Tricks for scalability and performance:
  - experience replay (use past frames)
  - separate network to generate learning targets (iterative update of Q)
- Outperforms all previous algorithms, and professional human player on most games

Google DeepMind: training&performance

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

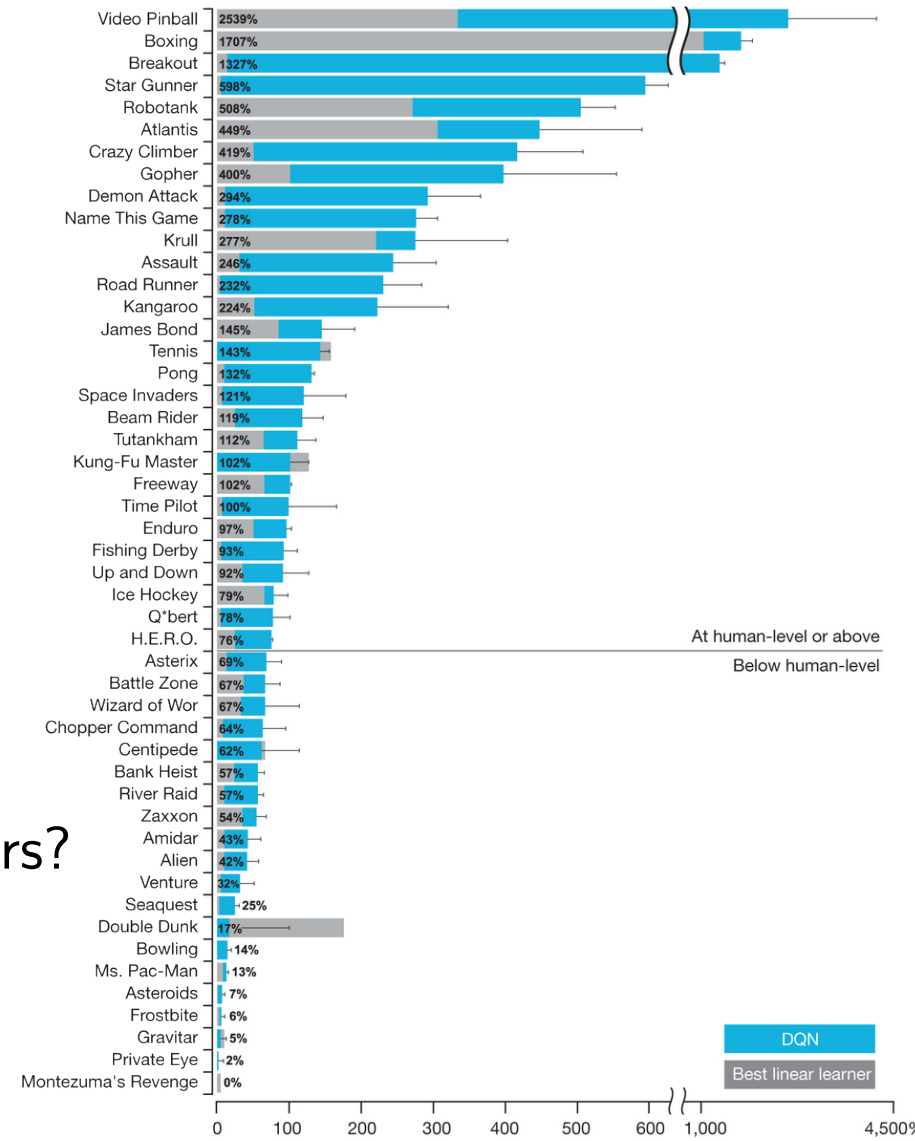
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

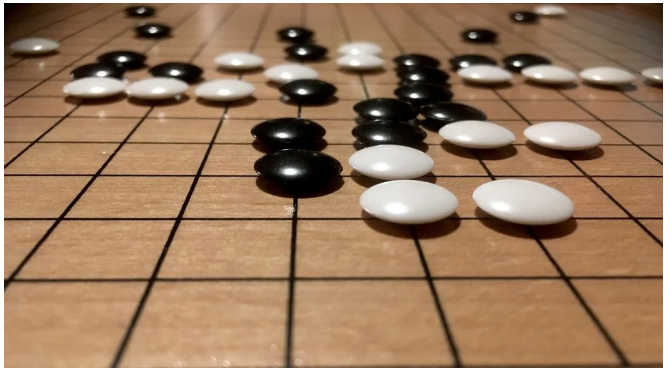
End For



- What about Breakout or Space invaders?

Google DeepMind: mastering Go

Nature 529, 484 (2016)

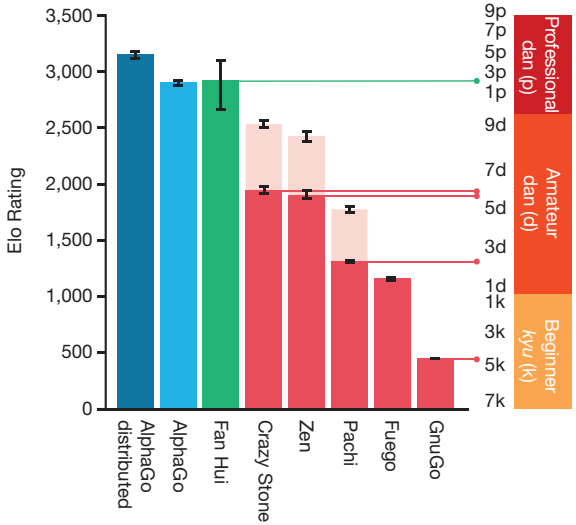


- Board games: can be solved with search tree of b^d possible sequences of moves
  - b = breadth [number of legal moves]
  - d = depth [length of game]
- Chess: $b \approx 35$, $d \approx 80$ $\rightarrow$ go: $b \approx 250$, $d \approx 150$
- Reduction:
  - of depth by position evaluation (replace subtree by approximation that predicts outcome)
  - of breadth by sampling actions from probability distribution (policy $p(a|s)$) over possible moves a in position s
- 19×19 image, represented by CNN
- Supervised learning policy network from expert human moves, reinforcement learning policy network on self-play (adjusts policy towards winning the game), value network that predicts winner of games in self-play.

Google DeepMind: AlphaGo

▸ Nature 529, 484 (2016)

- AlphaGo: 40 search threads, simulations on 48 CPUs, policy and value networks on 8 GPUs. Distributed AlphaGo: 1020 CPUs, 176 GPUs
- AlphaGo won 494/495 games against other programs (and still 77% against Crazy Stone with four handicap stones)
  - Fan Hui: 2013/14/15 European champion
  - Distributed AlphaGo won 5–0
  - AlphaGo evaluated thousands of times fewer positions than Deep Blue (first chess computer to bit human world champion) ⇒ better position selection (policy network) and better evaluation (value network)

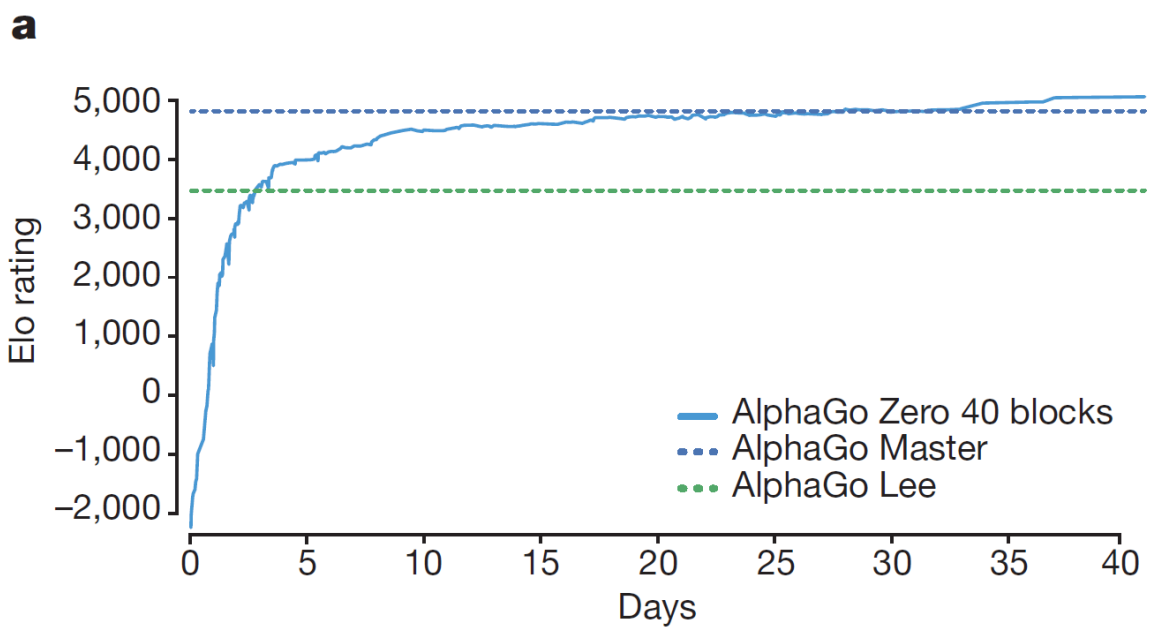


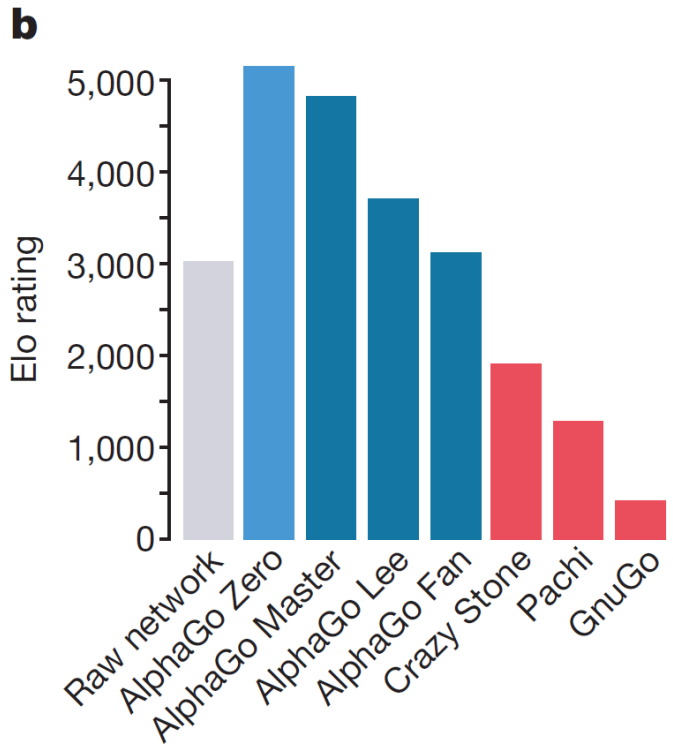
- Then played Lee Sedol (top Go play in the world over last decade) in March 2016 ⇒ won 4–1. AlphaGo given honorary professional ninth dan, considered to have "reach a level 'close to the territory of divinity' "
- Ke Jie (Chinese world #1): "Bring it on!". May 2017: 3–0 win for AlphaGo. New comment: "I feel like his game is more and more like the 'Go god'. Really, it is brilliant"

DeepMind AlphaGo Zero

Nature 550, 354 (2017)

- Learn from scratch, just from the rules and random moves
- Reinforcement learning from self-play, no human data/guidance
- Combined policy and value networks
- 4.9 million self-play games
- Beats AlphaGo Lee (several months of training) after just 36 hours
- Single machine with four TPU

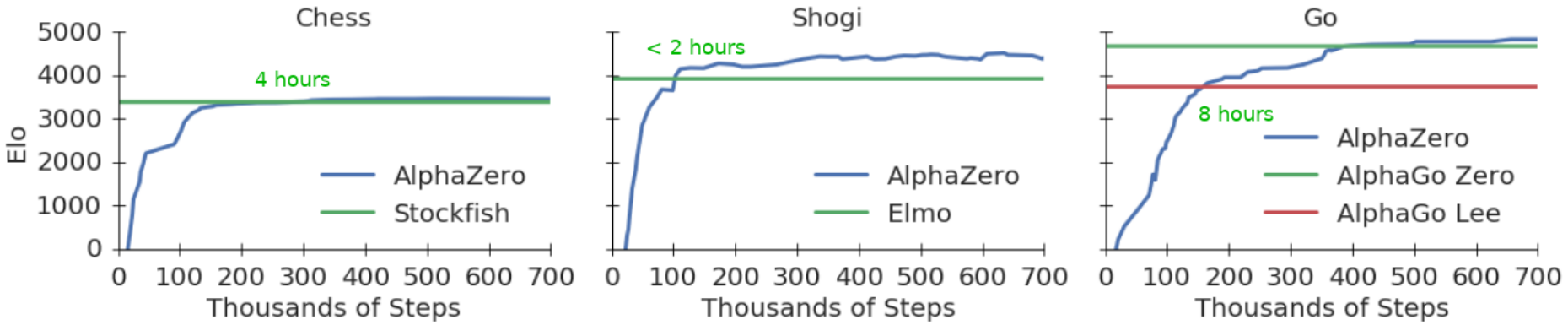




DeepMind AlphaZero

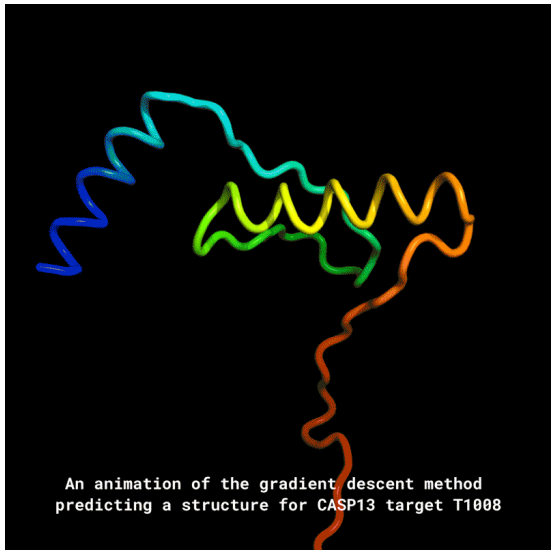
arXiv:1712.01815 [cs.AI]

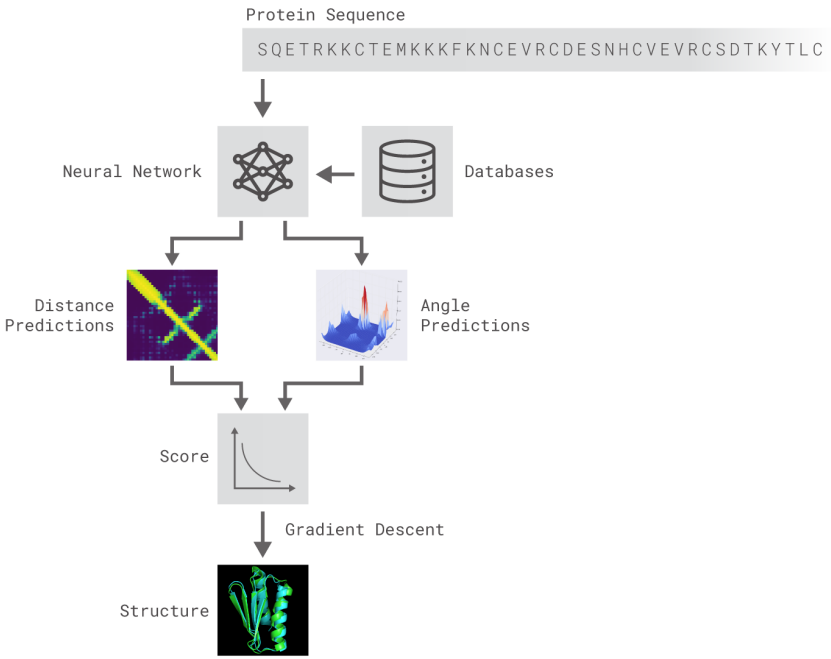
- Same philosophy as AlphaGo Zero, applied to chess, shogi and go
- Changes:
  - not just win/loss, but also draw or other outcomes
  - no additional training data from game symmetries
  - using always the latest network to generate self-play games rather than best one
  - tree search: 80k/70M for chess AlphaZero/Stockfish, 40k/35M for shogi AlphaZero/Elmo

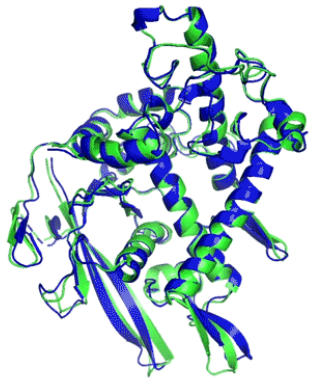


DeepMind AlphaFold

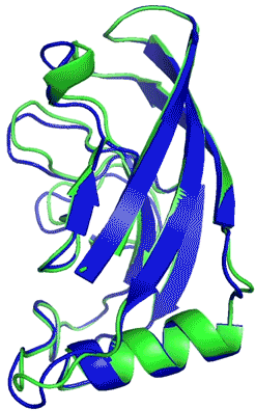
- Trying to tackle scientific problem
- Goal: predict 3D structure of protein based solely on genetic sequence
- Using DNN to predict
  - distances between pairs of amino acids
  - angles between chemical bonds
- Search DB to find matching existing substructures
- Also train a generative NN to invent new fragments
- Achieved best prediction ever







T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



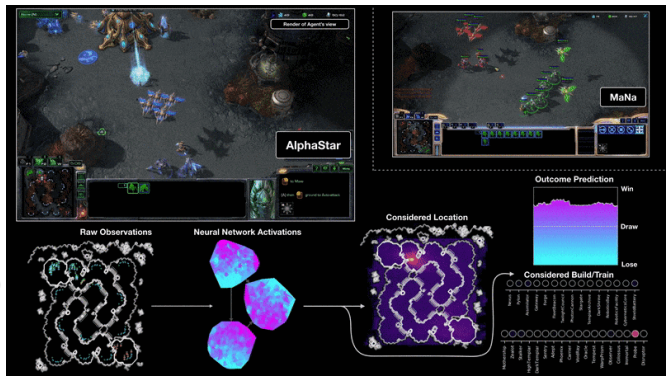
T1049 / 6y4f
93.3 GDT
(adhesin tip)

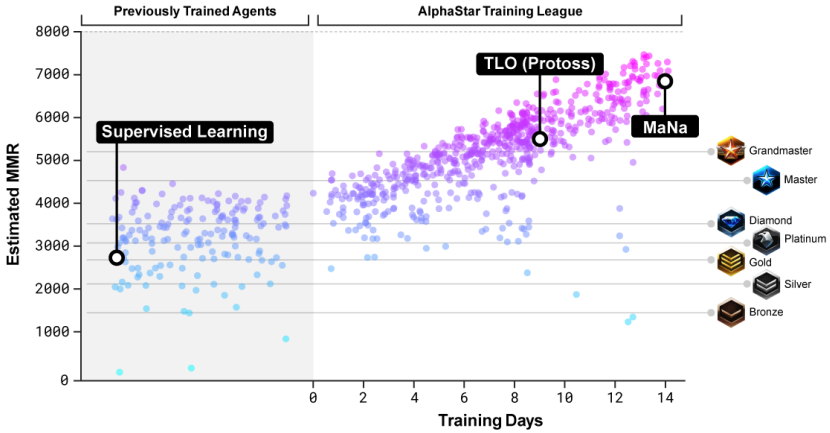
Experimental result
Computational prediction

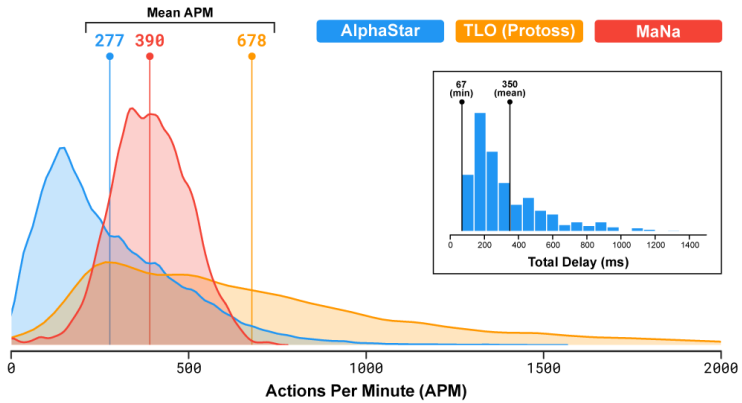
DeepMind AlphaStar

▸ Blog Jan 2019 ▸ Nature 575, 350–354 (2019)

- Mastering real-time strategy game StarCraft II
- Challenges in game theory (no single best strategy), imperfect information (hidden parts of game), long term planning, real time (continuous flow of actions), large action space (many units/buildings)
- Using DNN trained
  - directly on raw data games
  - supervised learning on human games
  - reinforcement learning (continuous league)
- DNN output: list of actions
- Trained for 14 days; each agent: up to 200 years of real-time play
- Runs on single desktop GPU
- Defeated 5–0 one of best pro-players

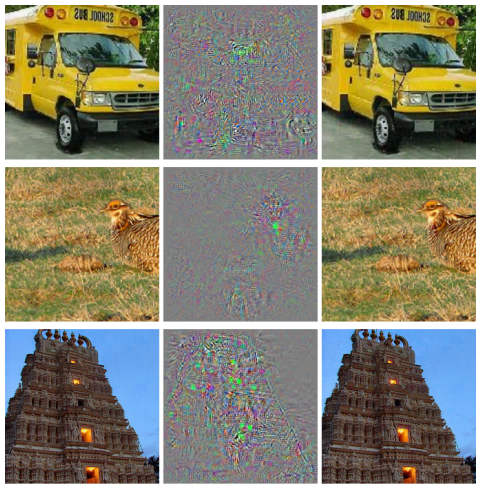






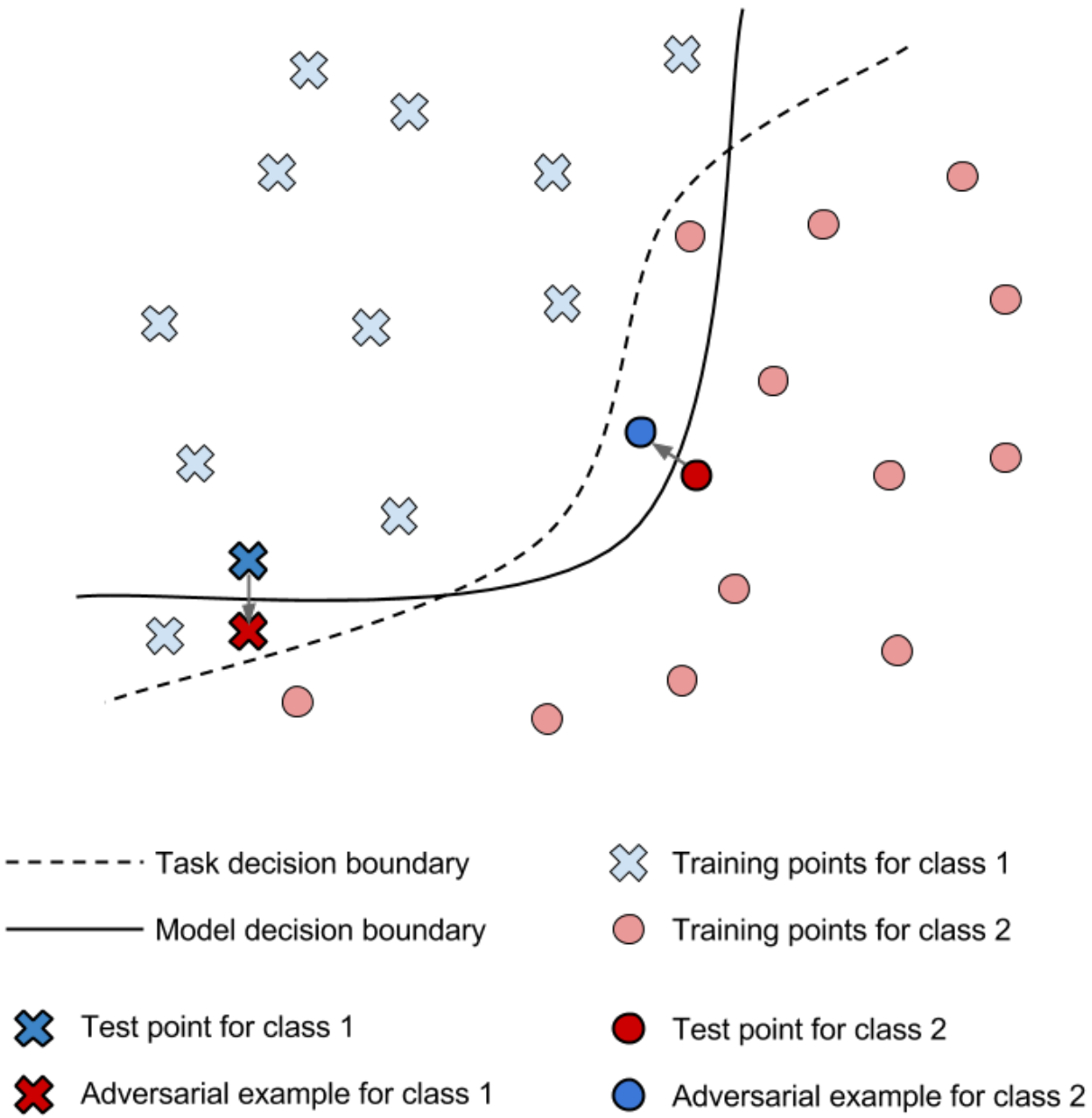
Deep networks at work

- Playing poker
  - Libratus (AI developed by Carnegie Mellon University) defeated four of the world's best professional poker players (Jan 2017 ▸ arXiv:1705.02955)
  - After 120,000 hands of Heads-up, No-Limit Texas Hold'em, led the pros by a collective $1,766,250 in chips
  - Learned to bluff, and win with incomplete information and opponents' misinformation
- Lip reading ▸ arXiv:1611.05358 [cs.CV]
  - human professional: deciphers less than 25% of spoken words
  - CNN+LSTM trained on television news programs: 50%
- **Limitation: adversarial attacks** ▸ arXiv:1312.6199 [cs.CV]



- left: correctly classified image
- middle: difference between left image and adversarial image (x10)
- right: adversarial image, classified as ostrich

Adversarial attack: what is happening?



▸ Breaking things easy, Papernot and Goodfellow, 2016

Adversarial attacks

▸ Cisse et al arXiv:1707.05373

original semantic segmentation framework

Adversarial attacks

▸ Cisse et al arXiv:1707.05373


original semantic segmentation framework

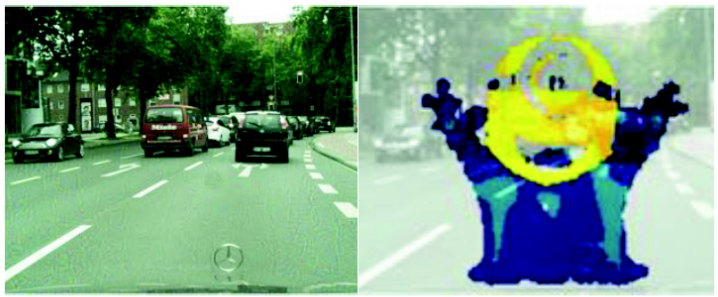
adversarial attack

Adversarial attacks

▸ Cisse et al arXiv:1707.05373

original semantic segmentation framework

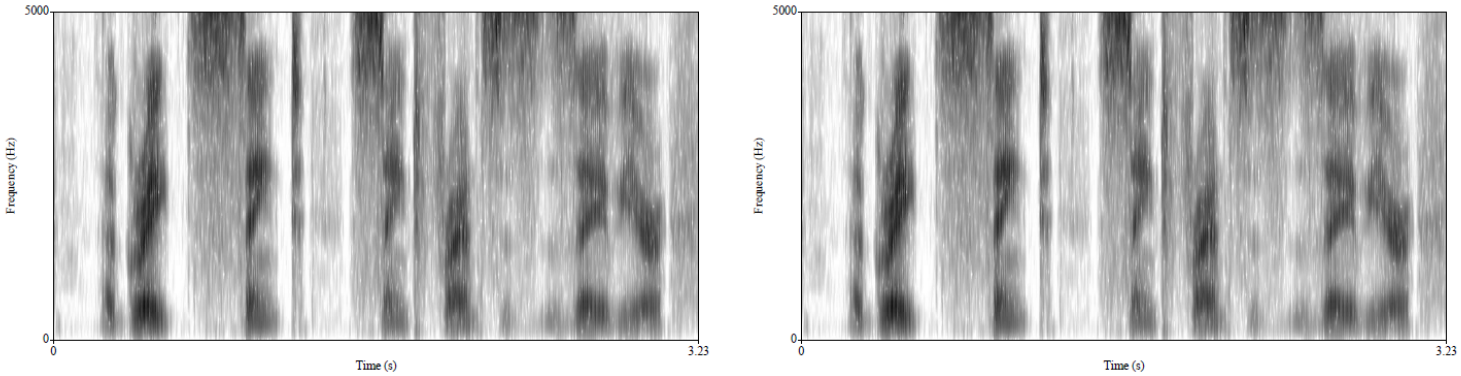
adversarial attack



compromised semantic segmentation framework

Adversarial attacks

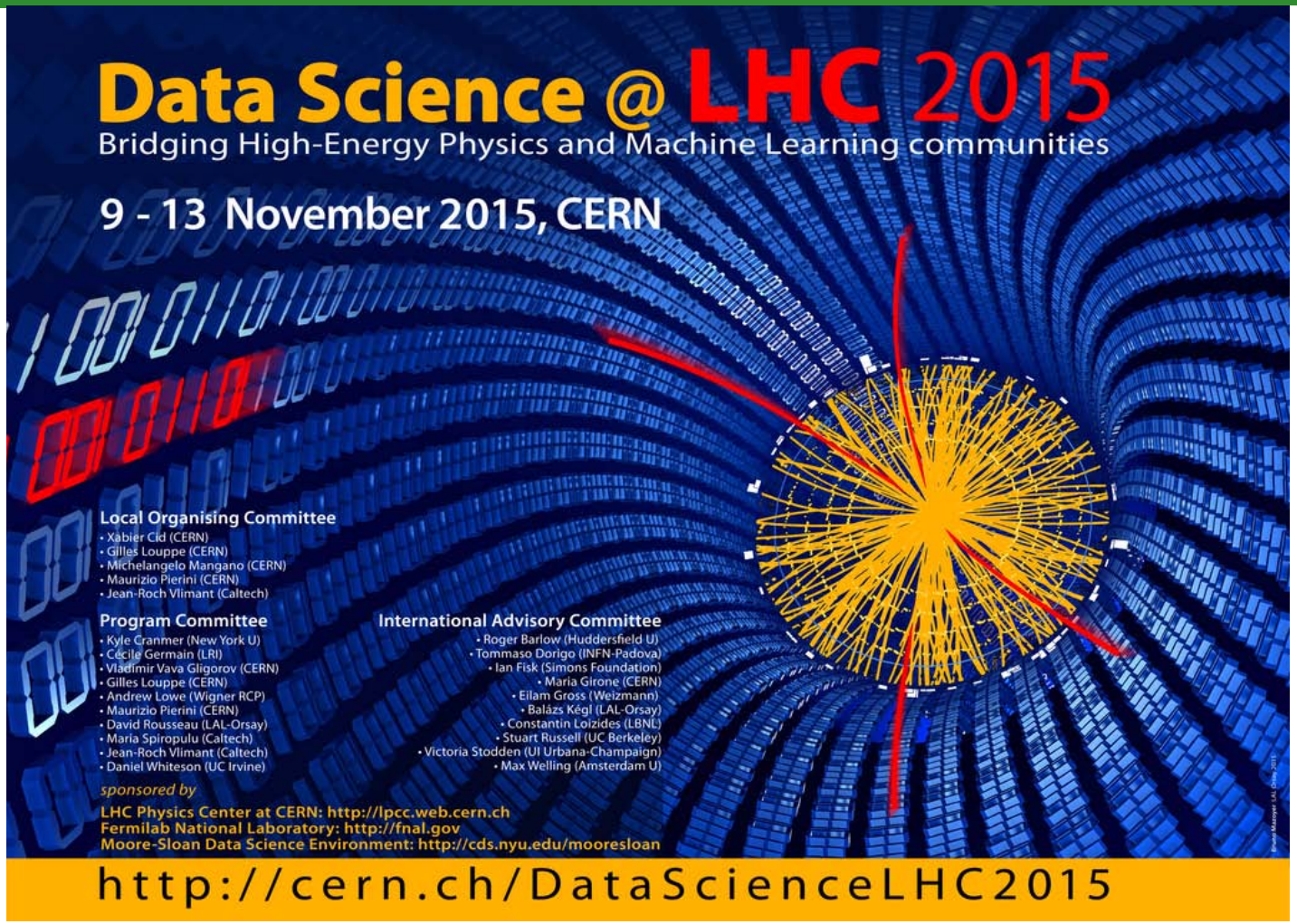
▸ Cisse et al arXiv:1707.05373



(a) a great saint saint francis zaviour (b) i great sinkt shink t frimsuss avir

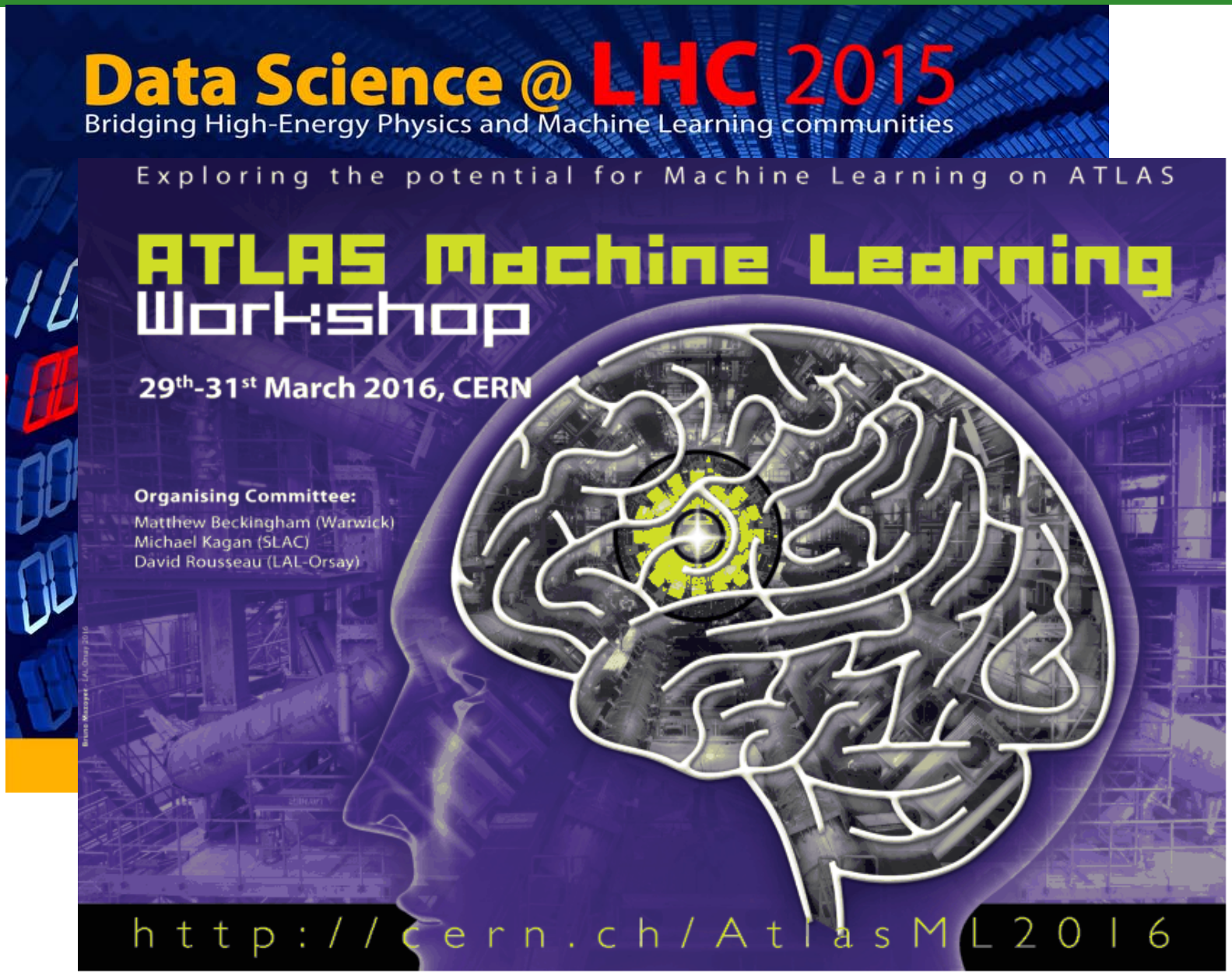
Figure 7: The model models' output for each of the spectrograms is located at the bottom of each spectrogram. The target transcription is: `A Great Saint Saint Francis Xavier.`

Machine learning and particle physics



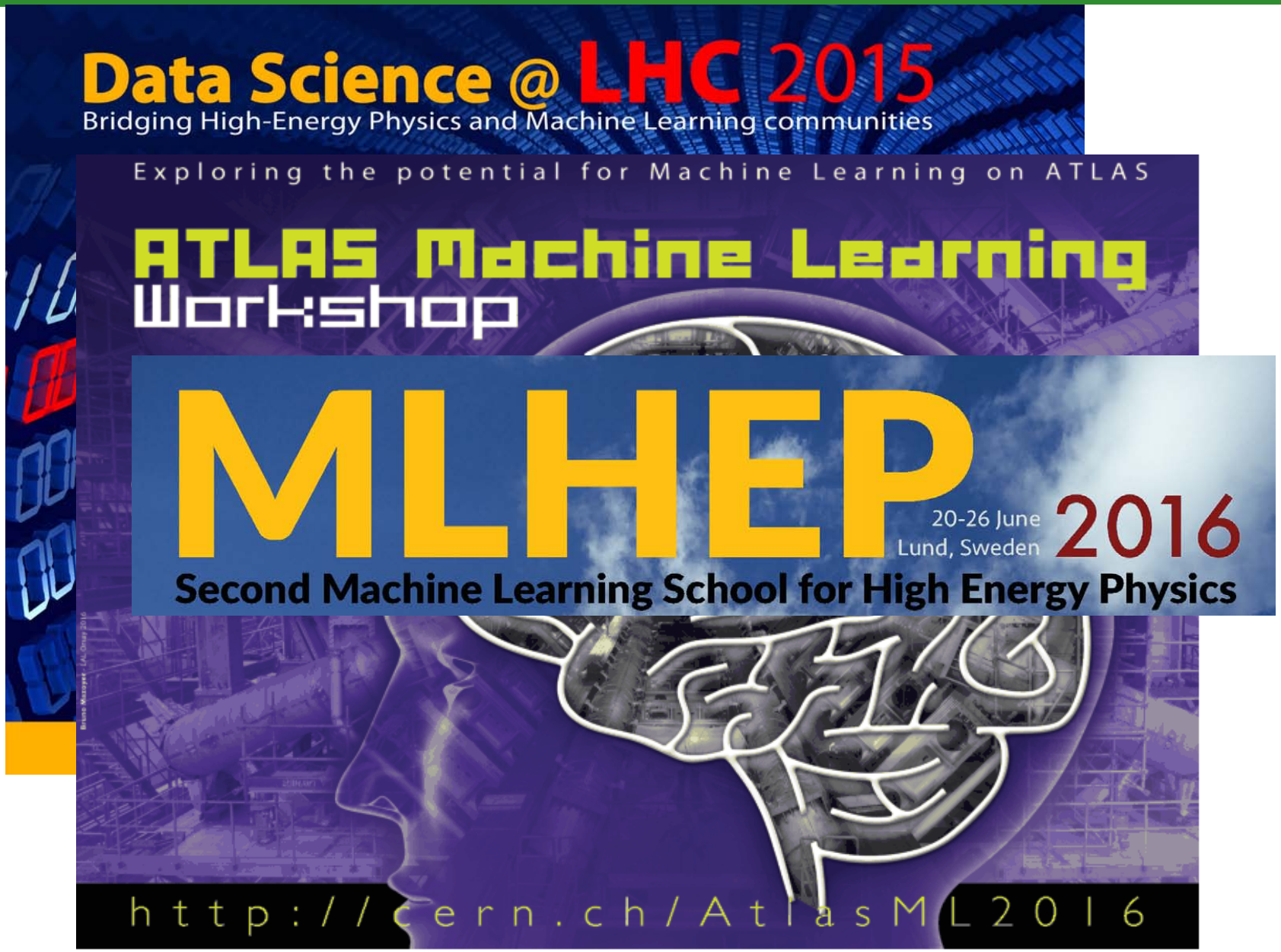
http://opendata.cern.ch

Machine learning and particle physics



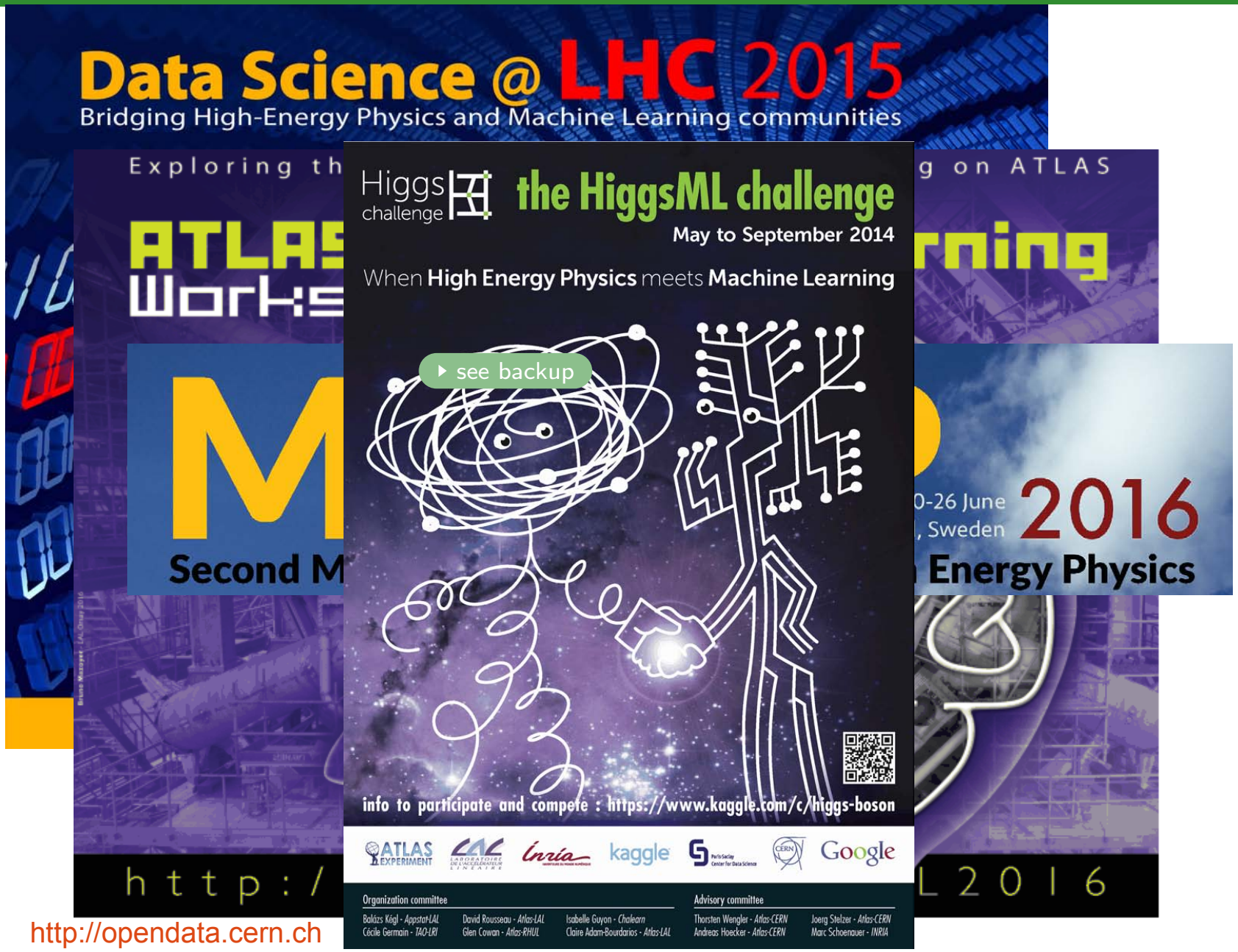
http://opendata.cern.ch

Machine learning and particle physics



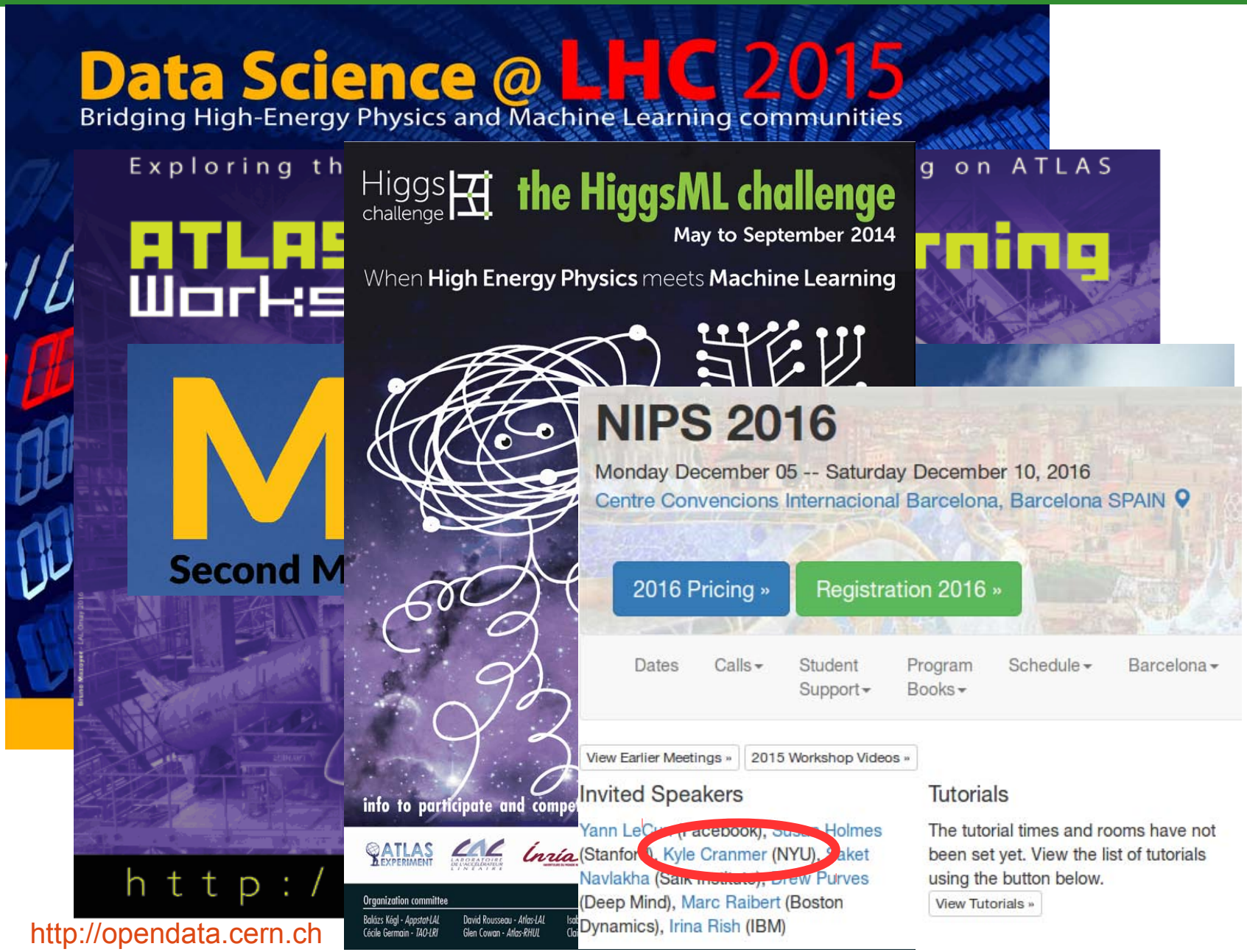
http://opendata.cern.ch

Machine learning and particle physics



http://opendata.cern.ch

Machine learning and particle physics

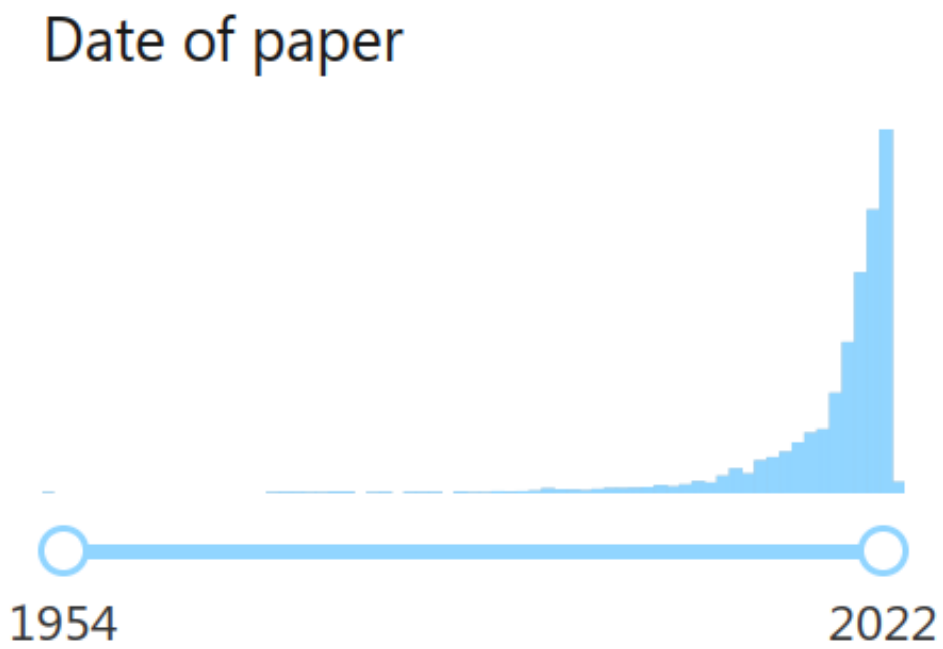


http://opendata.cern.ch

Machine learning and particle physics

http://opendata.cern.ch

Machine learning and (particle) physics



- https://inspirehep.net/literature?q=machine learning or deep learning or multivariate

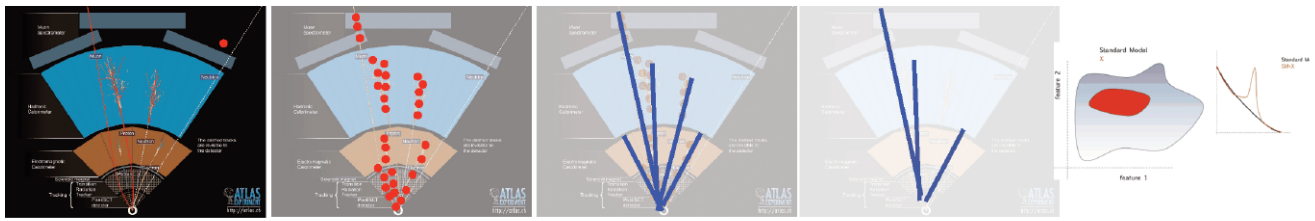
Up-to-date comprehensive review of papers

- https://github.com/iml-wg/HEPML-LivingReview

Machine learning and (particle) physics

- Reduce data dimensionality to allow analysis

Raw	Sparsified	Reco	Select	Physics	Ana
1e7	1e4	100-ish*	50	10	1

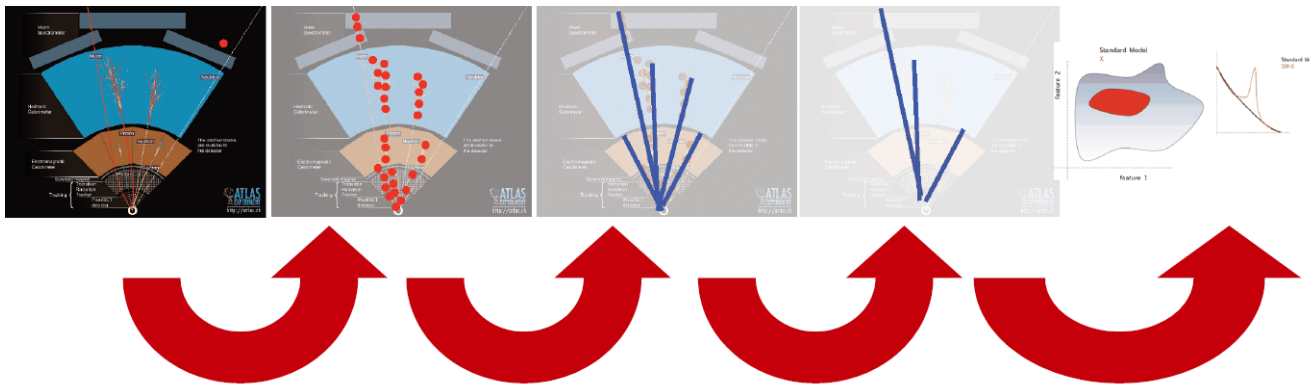


- Losing information at each simplification step

Machine learning and (particle) physics

- Reduce data dimensionality to allow analysis

Raw	Sparsified	Reco	Select	Physics	Ana
1e7	1e4	100-ish*	50	10	1

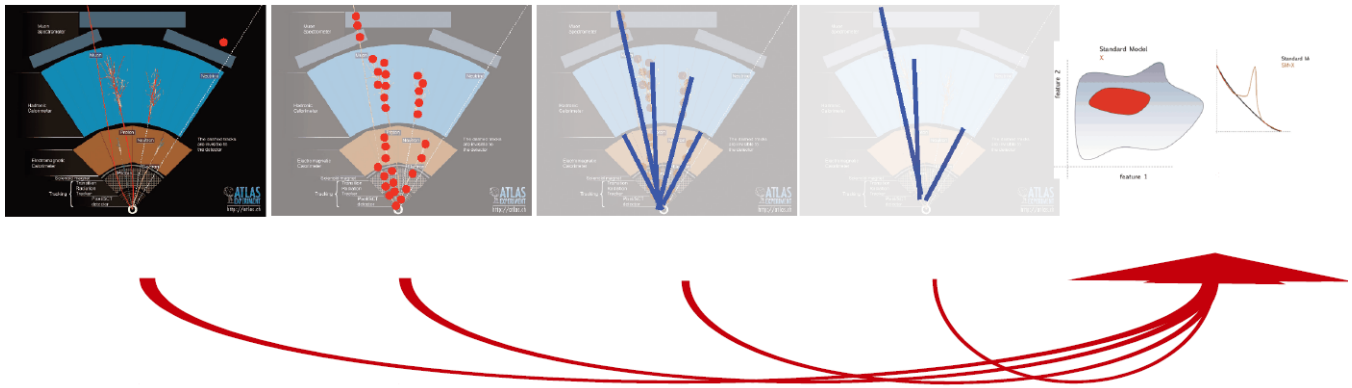


- Losing information at each simplification step
- Improve each step with ML?

Machine learning and (particle) physics

- Reduce data dimensionality to allow analysis

Raw	Sparsified	Reco	Select	Physics	Ana
1e7	1e4	100-ish*	50	10	1

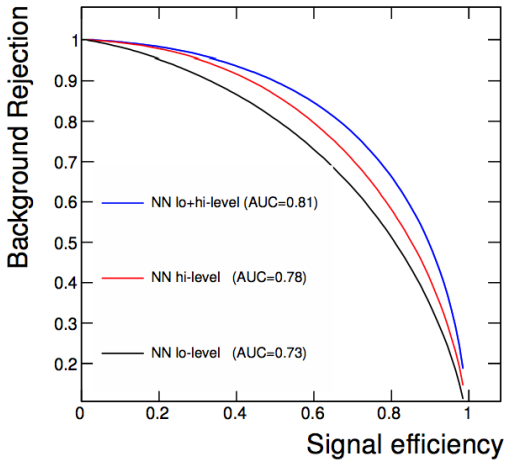


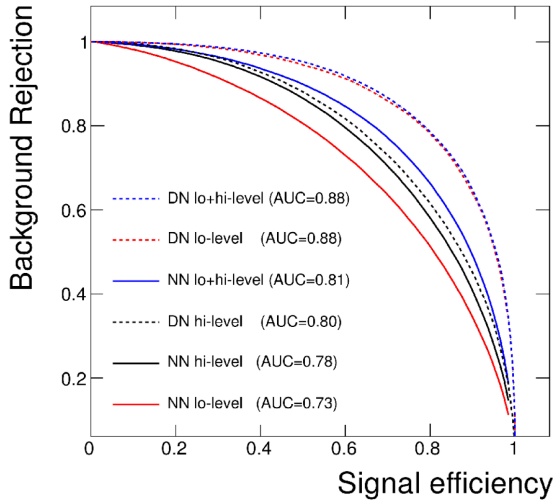
- Losing information at each simplification step
- Improve each step with ML?
- Skip one or more steps with ML?

Machine learning and particle physics

- Going to lower level features

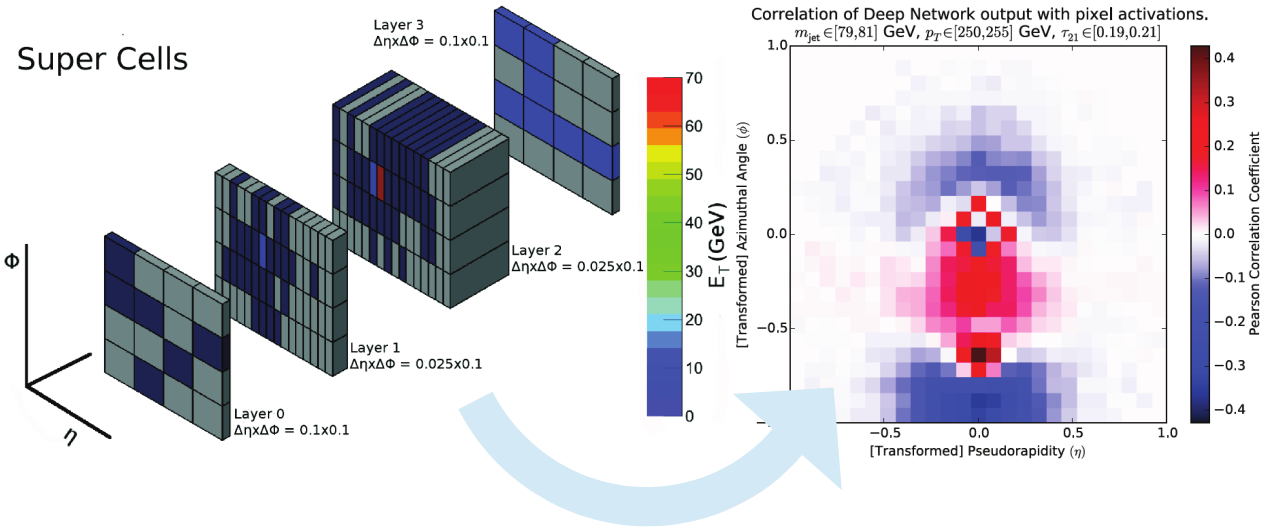
arXiv:1402.4735

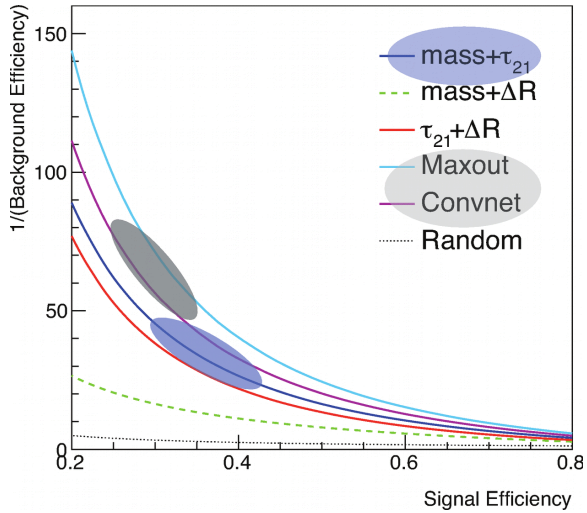




- Transforming inputs into images

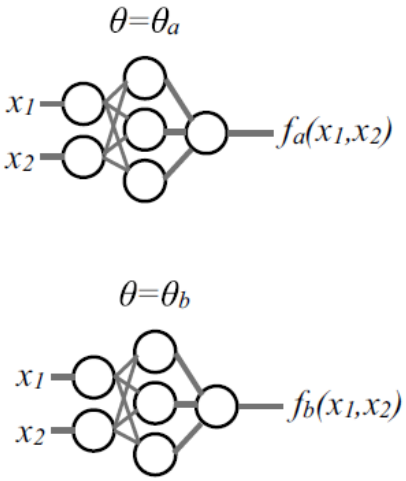
arXiv:1511.05190

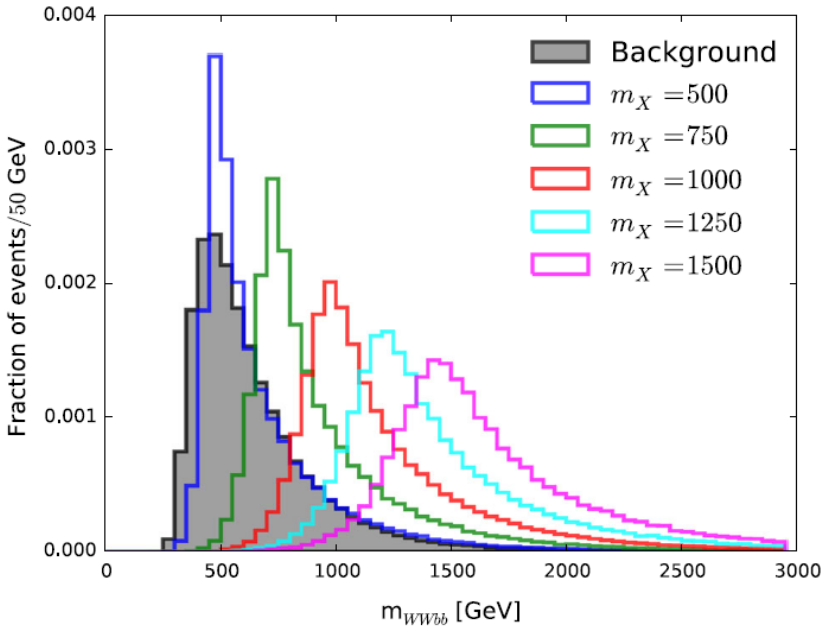




Machine learning and particle physics

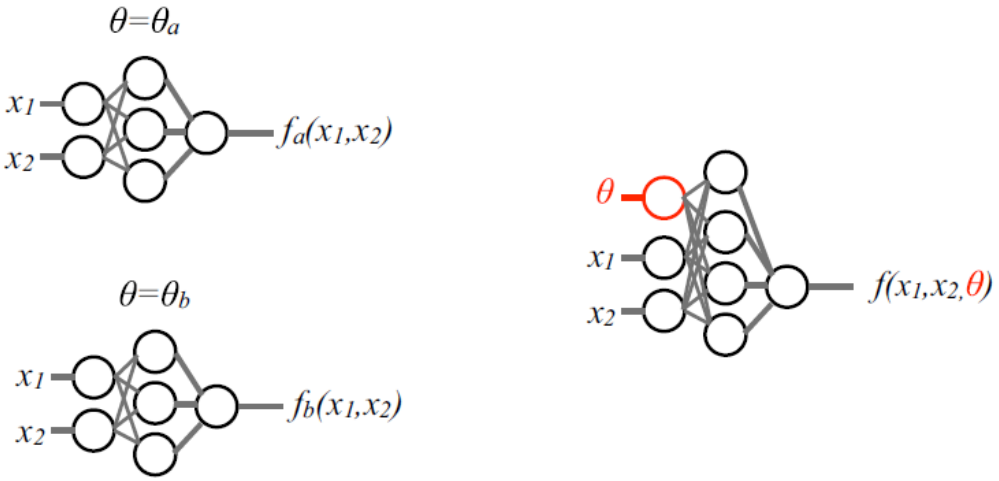
- Looking for new physics scenario with unknown mass
  $\Rightarrow$ one NN for each mass point

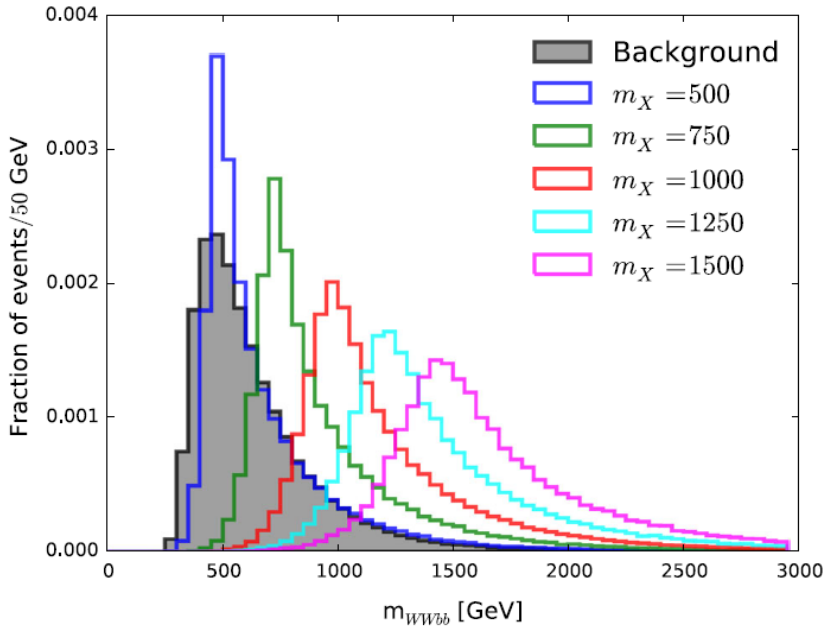




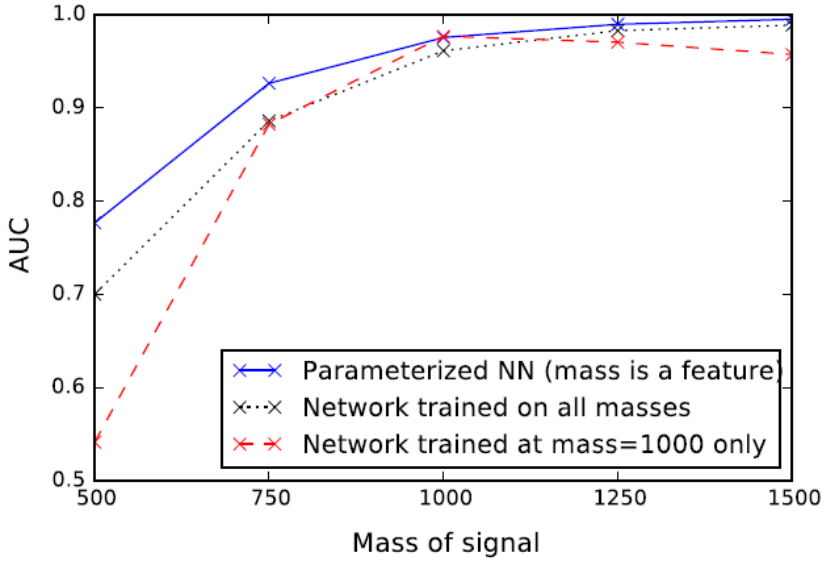
Machine learning and particle physics

- Looking for new physics scenario with unknown mass
  $\Rightarrow$ one NN for each mass point





- Parameterised NN
  ▸ EPJC (2016) 76:235
  ▸ arXiv:1601.07913
  - mass as training parameter
  - as good as dedicated training
  - generalises better

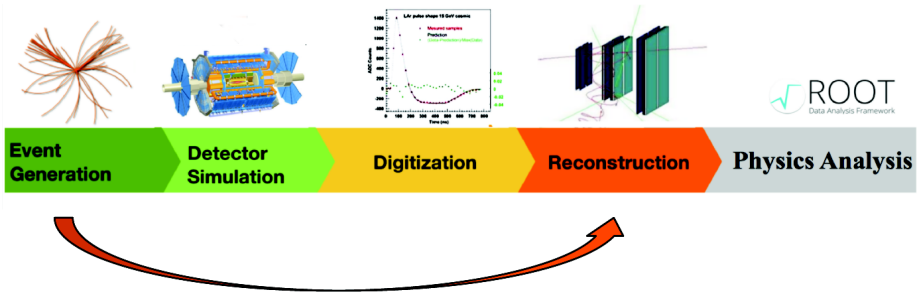


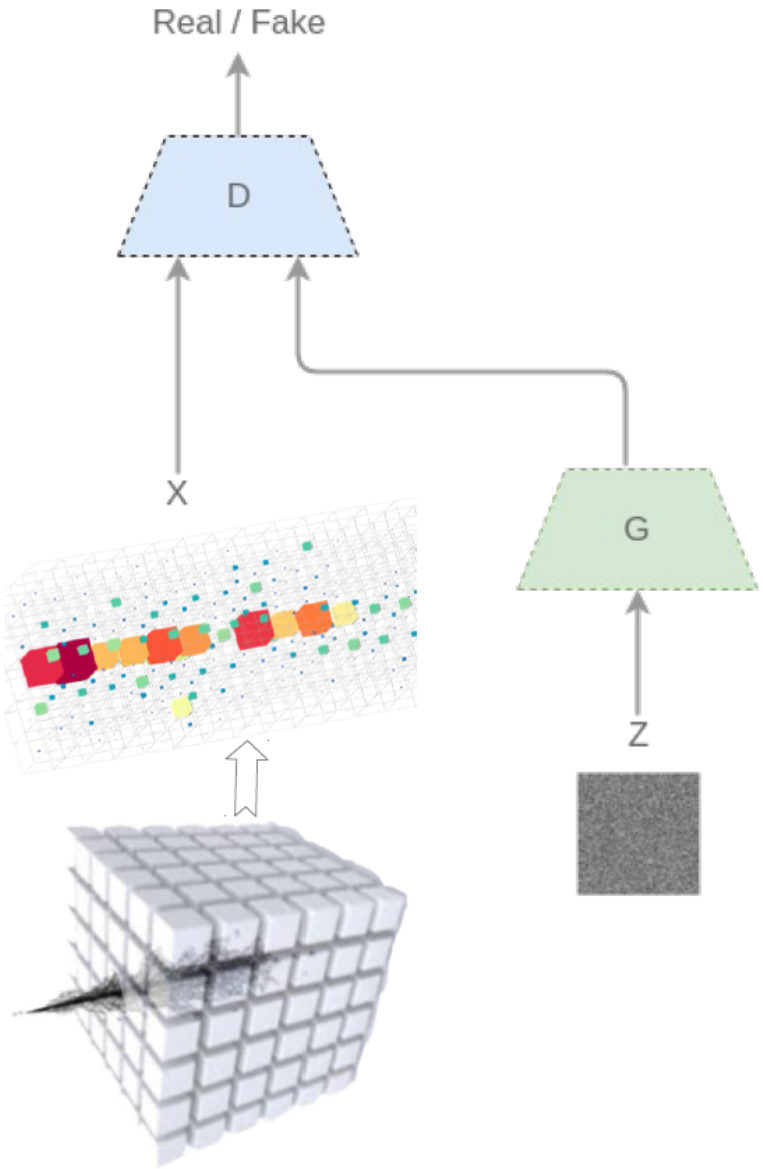
Machine learning and particle physics

Fast simulation with generative models

- Heavy CPU cost of simulation ($> 50\%$ of grid resources)
  - MC stats becoming limiting factor in analyses
- Replace "full simulation" with approximation
  - already routinely done, using parameterisation of showers or library of pre-simulated objects
  - use GAN to simulate medium-range hadrons in ATLAS

  ▸ arXiv:2109.02551





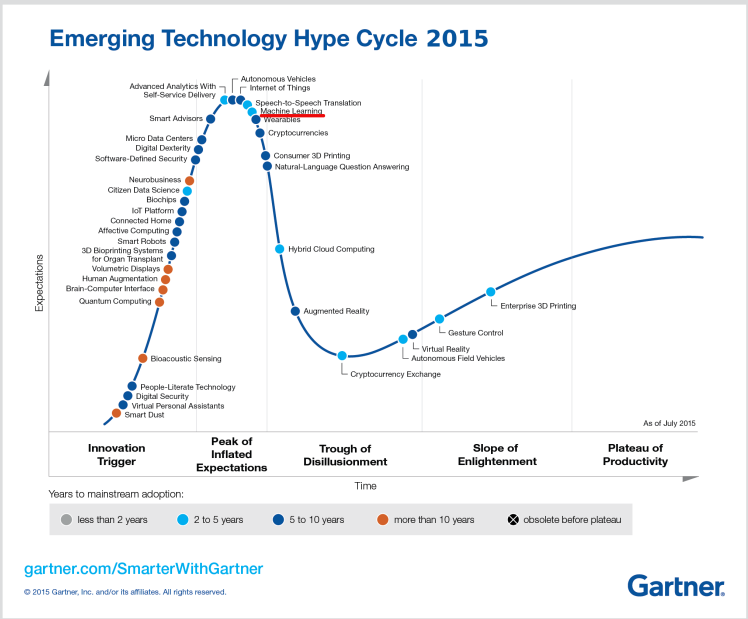
Machine learning and systematics

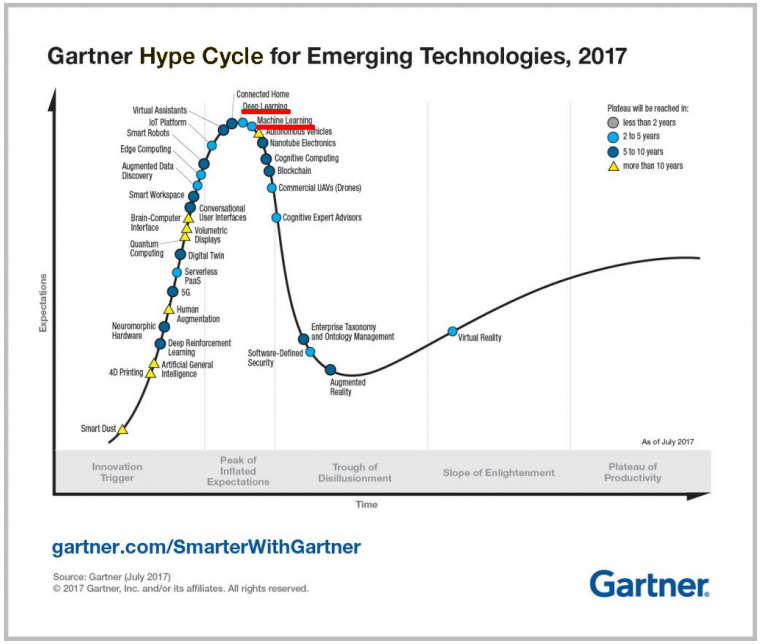
- No particular rule
- ML algorithm output can be considered as any other cut variable (just more powerful). Evaluate systematics by:
  - varying cut value
  - retraining
  - calibrating, etc.
- Most common (and appropriate): propagate other uncertainties (detector, theory, etc.) up to ML algorithm ouput and check how much the analysis is affected
- More and more common: profiling
  Watch out:
  - ML algorithm output powerful
  - signal region (high ML algorithm output) probably low statistics
    $\Rightarrow$ potential recipe for disaster if modelling is not good
- May require extra systematics, not so much on technique itself, but because it probes specific corners of phase space and/or wider parameter space (usually loosening pre-ML selection cuts)

Deep learning: looking forward

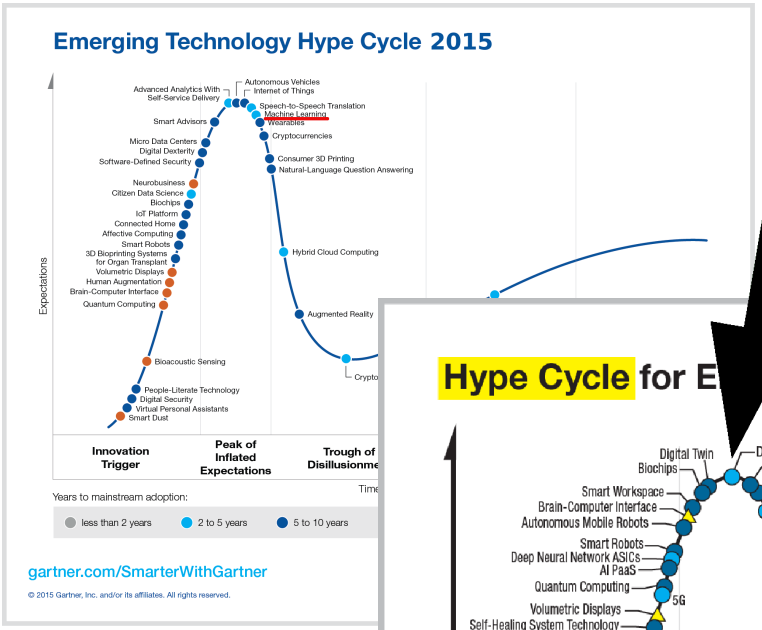
- Very active field of research in machine learning and artificial intelligence
  - not just at universities (Google, Facebook, Microsoft, NVIDIA, etc. . .)
- Training with curriculum:
  - what humans do over 20 years, or even a lifetime
  - learn different concepts at different times
  - solve easier or smoothed version first, and gradually consider less smoothing
  - exploit previously learned concepts to ease learning of new abstractions
- Combination of deep learning and reinforcement learning
- Domain adaptation and adversarial training
  - e.g. train in parallel network that produces difficult examples
  - learn discrimination (s vs. b) and difference between training and application samples (e.g. Monte Carlo simulation and real data)
- Getting popular: graph networks

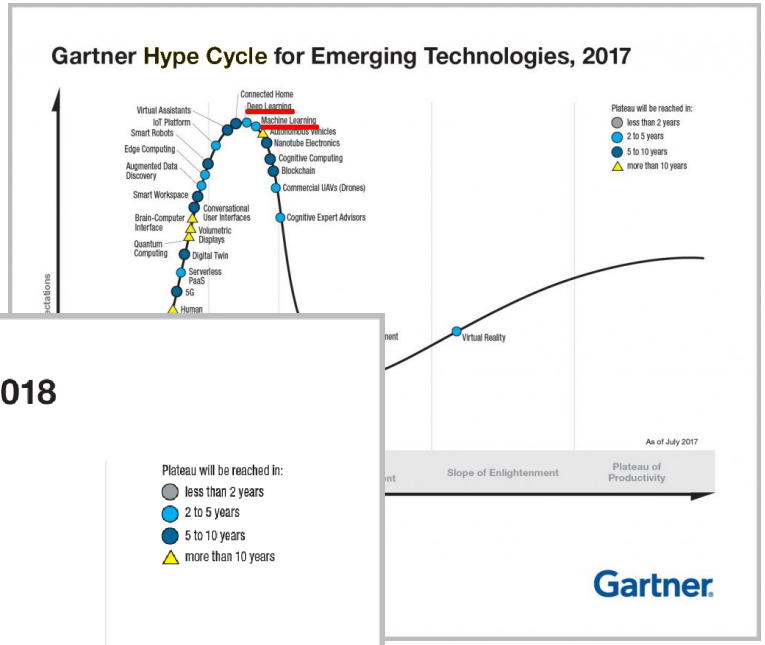
Hype cycle

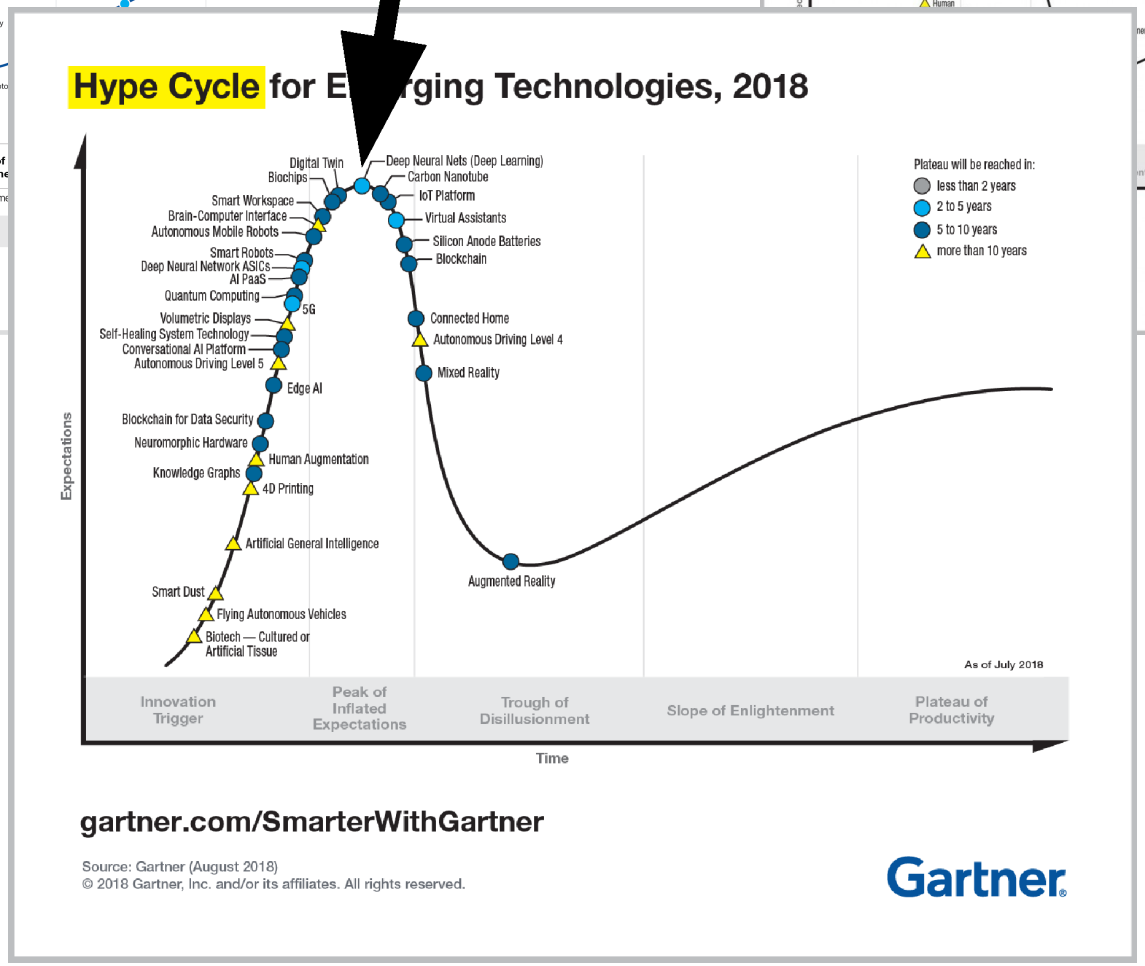

Emerging Technology Hype Cycle 2015
Expectations
Advanced Analytics With Self-Service Delivery
Autonomous Vehicles
Internet of Things
Speech-to-Speech Translation
Machine Learning
Wearables
Smart Advisors
Cryptocurrencies
Micro Data Centers
Digital Dexterity
Software-Defined Security
Consumer 3D Printing
Natural-Language Question Answering
Neurobusiness
Citizen Data Science
Biochips
IoT Platform
Connected Home
Affective Computing
Smart Robots
3D Bioprinting Systems for Organ Transplant
Volumetric Displays
Human Augmentation
Brain-Computer Interface
Quantum Computing
Hybrid Cloud Computing
Augmented Reality
Enterprise 3D Printing
Gesture Control
Virtual Reality
Autonomous Field Vehicles
Cryptocurrency Exchange
Bioacoustic Sensing
People-Literate Technology
Digital Security
Virtual Personal Assistants
Smart Dust
As of July 2015
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
Time
Years to mainstream adoption:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
obsolete before plateau
gartner.com/SmarterWithGartner
© 2015 Gartner, Inc. and/or its affiliates. All rights reserved.
Gartner


Gartner Hype Cycle for Emerging Technologies, 2017
Expectations
Connected Home
Deep Learning
Machine Learning
Virtual Assistants
IoT Platform
Smart Robots
Autonomous Vehicles
Nanotube Electronics
Edge Computing
Cognitive Computing
Augmented Data Discovery
Blockchain
Commercial UAVs (Drones)
Smart Workspace
Conversational User Interfaces
Cognitive Expert Advisors
Brain-Computer Interface
Volumetric Displays
Quantum Computing
Digital Twin
Serverless PaaS
5G
Human Augmentation
Neuromorphic Hardware
Deep Reinforcement Learning
Enterprise Taxonomy and Ontology Management
Virtual Reality
Software-Defined Security
Artificial General Intelligence
4D Printing
Augmented Reality
Smart Dust
Plateau will be reached in:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
As of July 2017
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
Time
gartner.com/SmarterWithGartner
Source: Gartner (July 2017)
© 2017 Gartner, Inc. and/or its affiliates. All rights reserved.
Gartner

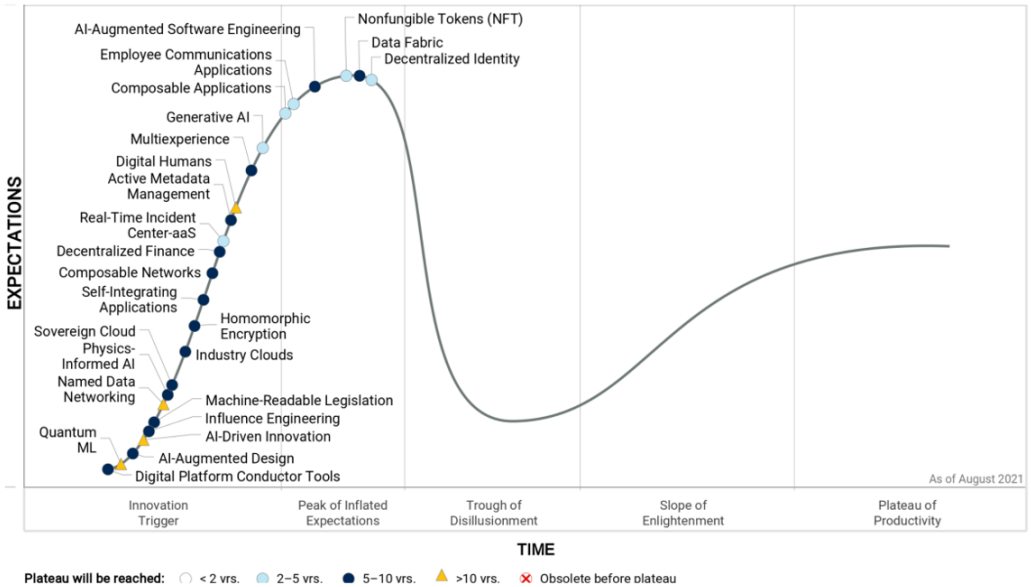
Hype cycle


Emerging Technology Hype Cycle 2015
Expectations
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionme
Time
Years to mainstream adoption:
less than 2 years
2 to 5 years
5 to 10 years
gartner.com/SmarterWithGartner
© 2015 Gartner, Inc. and/or its affiliates. All rights reserved.


Gartner Hype Cycle for Emerging Technologies, 2017
Plateau will be reached in:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
Virtual Reality
As of July 2017
Slope of Enlightenment
Plateau of Productivity
Gartner


Hype Cycle for Emerging Technologies, 2018
Digital Twin
Biochips
Smart Workspace
Brain-Computer Interface
Autonomous Mobile Robots
Smart Robots
Deep Neural Network ASICs
AI PaaS
Quantum Computing
5G
Volumetric Displays
Self-Healing System Technology
Conversational AI Platform
Autonomous Driving Level 5
Edge AI
Blockchain for Data Security
Neuromorphic Hardware
Human Augmentation
Knowledge Graphs
4D Printing
Artificial General Intelligence
Smart Dust
Flying Autonomous Vehicles
Biotech — Cultured or Artificial Tissue
Deep Neural Nets (Deep Learning)
Carbon Nanotube
IoT Platform
Virtual Assistants
Silicon Anode Batteries
Blockchain
Connected Home
Autonomous Driving Level 4
Mixed Reality
Augmented Reality
Plateau will be reached in:
less than 2 years
2 to 5 years
5 to 10 years
more than 10 years
Expectations
Innovation Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity
Time
As of July 2018
gartner.com/SmarterWithGartner
Source: Gartner (August 2018)
© 2018 Gartner, Inc. and/or its affiliates. All rights reserved.
Gartner

Hype cycle 2021

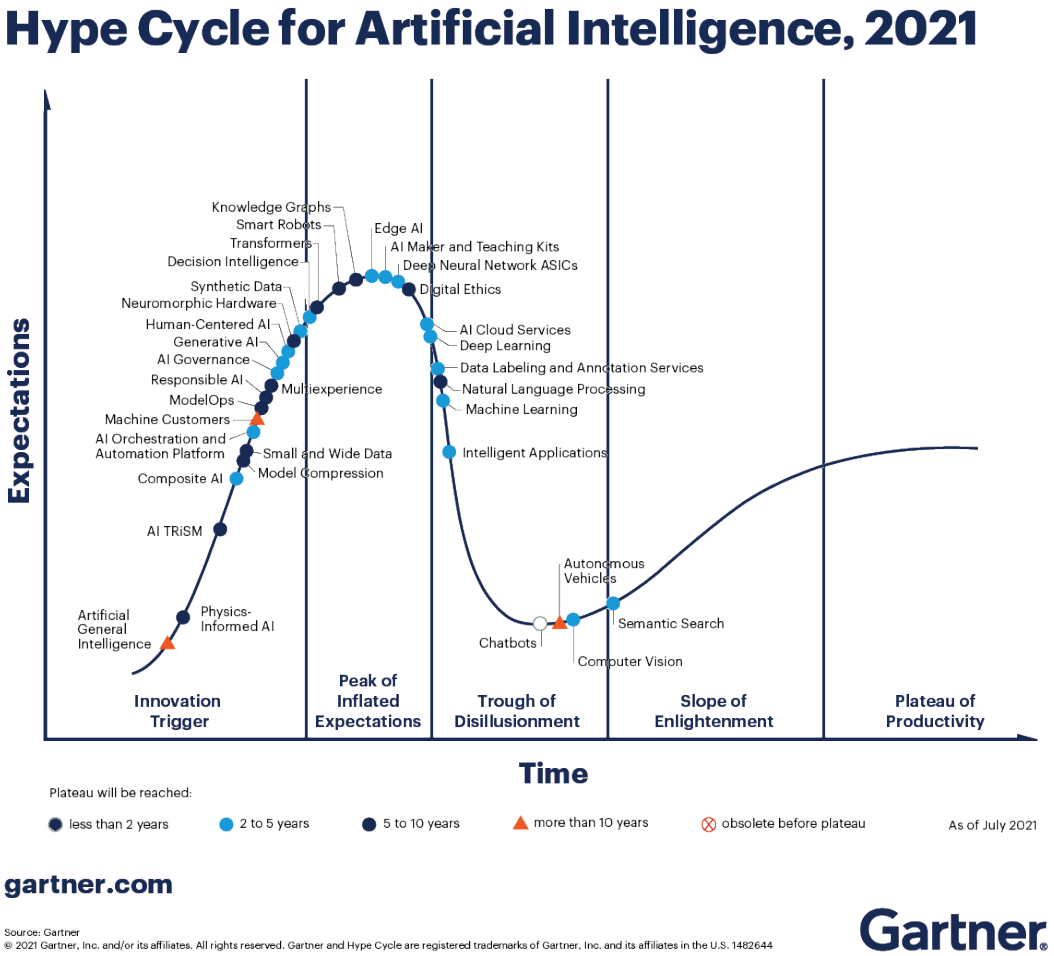


Source: Gartner (August 2021)

747576

- No deep learning/ML anymore (since 2019)
- Instead, all sorts of "AI" and ML-driven systems
  - data fabric
  - AI-augmented software engineering
  - generative AI
  - physics-informed AI
  - AI-driven innovation
  - AI-augmented design
  - quantum ML

Hype cycle 2021



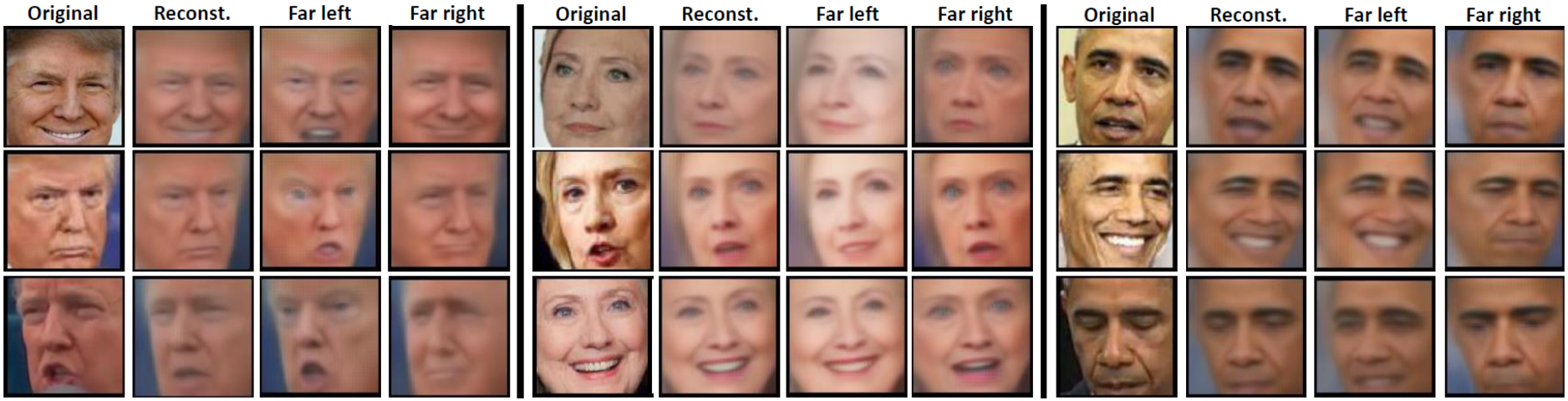
- No deep learning/ML anymore (since 2019)
- Instead, all sorts of "AI" and ML-driven systems
  - data fabric
  - AI-augmented software engineering
  - generative AI
  - physics-informed AI
  - AI-driven innovation
  - AI-augmented design
  - quantum ML

AI and society

- ML achieves super-human performance for well-designed problems, or games with score $\Rightarrow$ where one can define a proper loss function or reward
- Scale to "real" problems?
  - explainability
  - causality
  - "moral" stand
  - culture, art
- Many advances in medical imaging, modelling of various phenomena, supernova analysis or LHC physics, but issues with:
  - out-of-distribution generalisation
  - scalability of computing resources, carbon footprint
  - reliability
  - decision bias (gender, race, etc.)
- Workshops/socials on Fairness & ethics, AI for Good, Tackling Climate Change with ML, AI for Humanitarian Assistance and Disaster Response, Safety and Robustness in Decision-making, ...
- Importance of personal decisions

From NeurIPS2019: Hidden information

▸ Predicting the Politics of an Image Using Webly Supervised Data



▸ Computational Mirrors: Blind Inverse Light Transport by Deep Matrix Factorization (edited video)

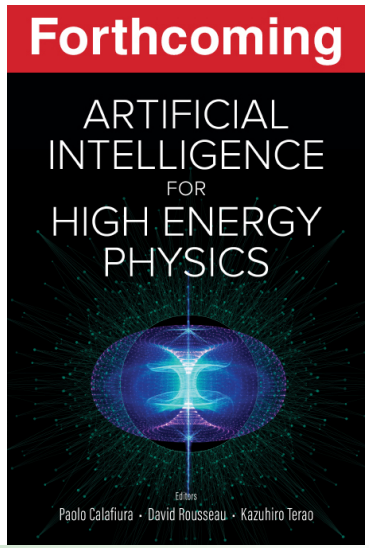
Conclusion

- Many techniques and tools exist to achieve optimal discrimination
- (Un)fortunately, no one method can be shown to outperform the others in all cases
- One should try several and pick the best one for any given problem
- Latest machine learning algorithms (e.g. deep networks) require enormous hyperparameter space optimisation...
- Machine learning and multivariate techniques are at work in your everyday life without your knowning and can easily outsmart you for many tasks
- Try this to convince yourself ▸ http://www.phi-t.de/mousegame/mousegame_en.html

Upcoming reference book (out next month)

Artificial Intelligence for High Energy Physics

▸ https://doi.org/10.1142/12200



Machine learning software

- Go for a fully integrated solution
  - use different multivariate techniques easily
  - spend your time on understanding your data and model
- Examples:
  - TMVA (Toolkit for MultiVariate Analysis)
    Integrated in ROOT, complete manual ▸ https://root.cern/tmva
    - Example code in ▸ backup. See tutorial next week
  - scikit-learn (python) ▸ https://scikit-learn.org
- Dedicated to BDT but transparently integrated with e.g. scikit-learn:
  - XGBoost (popular in HEP) ▸ arXiv:1603.02754 ▸ https://github.com/dmlc/xgboost
    (note: cannot handle negative weights)
  - LightGBM [Microsoft] ▸ https://lightgbm.readthedocs.io
  - CatBoost [Yandex] ▸ https://catboost.ai/
- Specifically for Deep Learning:
  - TensorFlow (+Keras) [Google] ▸ https://www.tensorflow.org/
  - PyTorch [Facebook] ▸ https://pytorch.org/
  - Torch (in lua) ▸ http://www.torch.ch/

References I: boosted decision trees


L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees*, Wadsworth, Stamford, 1984


R.E. Schapire, "The strength of weak learnability" ▸ Machine Learning **5** (1990) 197


Y. Freund, "Boosting a weak learning algorithm by majority" ▸ Information and computation **121** (1995) 256


Y. Freund and R.E. Schapire, "Experiments with a New Boosting Algorithm" in *Machine Learning: Proceedings of the Thirteenth International Conference*, edited by L. Saitta (Morgan Kaufmann, San Fransisco, 1996) p. 148


Y. Freund and R.E. Schapire, "A short introduction to boosting" ▸ Journal of Japanese Society for Artificial Intelligence 14 (1999) 771


R. E. Schapire and Y. Freund, "Boosting: Foundations and Algorithms", MIT Press, 2012.


Y. Freund and R.E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting" ▸ Journal of Computer and System Sciences **55** (1997) 119


J.H. Friedman, T. Hastie and R. Tibshirani, "Additive logistic regression: a statistical view of boosting" ▸ Annals of Statistics **28** (2000) 377

References II: decision trees and more

- J. H. Friedman, "Greedy function approximation: A gradient boosting machine" ▸ Annals of Statistics **29** (2001) 1189
- T. Hastie, R. Tibshirani, and J. Friedman, "The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd edition)" ▸ Springer Series in Statistics, 2009
- S. Shalev-Shwartz and S. Ben-David, "Understanding Machine Learning: From Theory to Algorithms" ▸ Cambridge University Press, 2014
- M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off" ▸ PNAS **116** (2019) 15849, ▸ arXiv:1812.11118 [stat.ML]
- L. Breiman, "Bagging Predictors" ▸ Machine Learning **24** (1996) 123
- L. Breiman, "Random forests" ▸ Machine Learning **45** (2001) 5
- B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor ▸ Nucl. Instr. Meth. A **543** (2005) 577; H.-J. Yang, B.P. Roe, and J. Zhu ▸ Nucl. Instr. Meth. A **555** (2005) 370
- V. M. Abazov *et al.* [D0 Collaboration], "Evidence for production of single top quarks" ▸ Phys. Rev. D **78** (2008) 012005
- V. Vapnik, "The Nature of Statistical Learning Theory", ▸ Springer, New York, 2nd Edition, 2000

References III: neural networks


C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2007


M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, 1969


W.S. McCulloch & W. Pitts, "A logical calculus of the ideas immanent in nervous activity", Bulletin of Mathematical Biophysics, 5, 115-137, 1943


F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage & Organization in the Brain", Psychological Review, 65, pp. 386-408, 1958


D.E.Rumelhart et al., "Learning representations by back-propagating errors", Nature vol. 323, p. 533, 1986


K.Hornik et al., "Multilayer Feedforward Networks are Universal Approximators", Neural Networks, Vol. 2, pp 359-366, 1989


Y. LeCun, L. Bottou, G. Orr and K. Muller, "Efficient BackProp", in *Neural Networks: Tricks of the trade*, Orr, G. and Muller K. (Eds), Springer, 1998

References IV: deep neural networks

Q.V. Le et al., "Building High-level Features Using Large Scale Unsupervised Learning", in *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012 ▸ http://research.google.com/pubs/pub38115.html

Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, "Greedy Layer-Wise Training of Deep Networks", in *Advances in Neural Information Processing Systems 19* (NIPS'06), pages 153–160, MIT Press 2007

M.A. Ranzato, C. Poultney, S. Chopra and Y. LeCun, in J. Platt et al., "Efficient Learning of Sparse Representations with an Energy-Based Model", in *Advances in Neural Information Processing Systems 19* (NIPS'06), pages 1137–1144, MIT Press, 2007

Y. Bengio, "Learning deep architectures for AI", *Foundations and Trends in Machine Learning*, Vol. 2, No. 1 (2009) 1–127. Also book at ▸ Now Publishers ▸ http://www.iro.umontreal.ca/ lisa/publications2/index.php/publications/show/239

I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016 ▸ http://www.deeplearningbook.org

G. Carleo et al., "Machine learning and the physical sciences" ▸ Rev. Mod. Phys. 91, 045002 (2019) ▸ arXiv:1903.10563

Backup

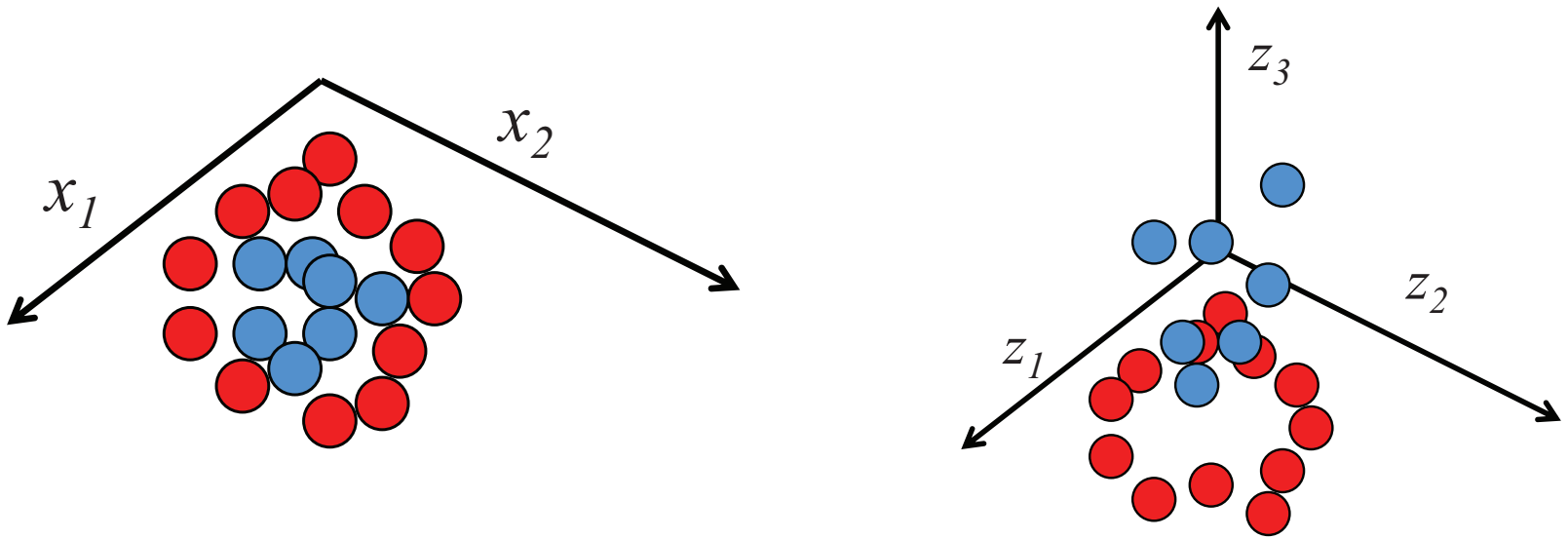
- Example exam questions
- Support vector machines
- Gini index of diversity
- Pruning a decision tree
- Single top production evidence at D0 (2006)
- Other auto-encoders
- HiggsML Kaggle challenge
- ML training with TMVA

Example exam questions

- Explain in your own words the three main approaches to train machine learning algorithms
- Comment on differences between boosted decision trees and neural networks: input variables, resilience to overtraining, hyperparameter tuning
- Explain why neural networks used to be shallow (mathematical motivation, training)
- Describe briefly the main features and use cases of CNN, RNN and auto-encoders
- Explain briefly the behaviour on each of the four plots on page 18, showing the training and testing curves as a function of the number of trees or epochs (difference between train/test, overfitting/underfitting, optimal classifier, etc.)

Support vector machines [Vapnik 2000]

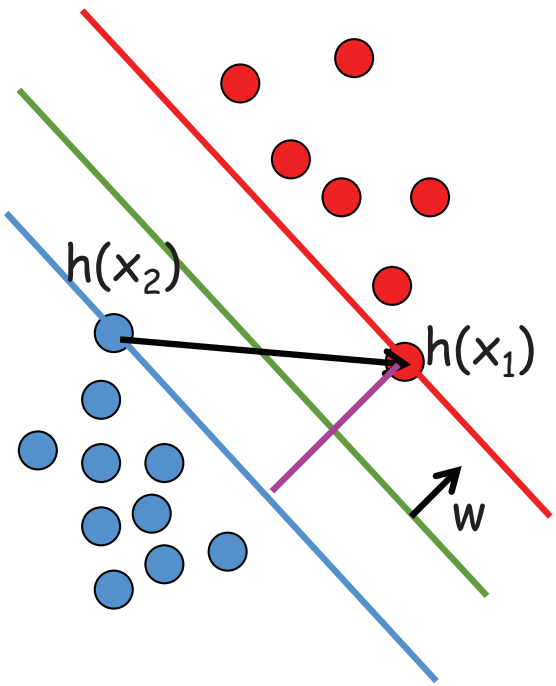
- Fisher discriminant: may fail completely for highly non-Gaussian densities
- But linearity is good feature $\Rightarrow$ try to keep it
- Generalising Fisher discriminant: data non-separable in n-dim space $\mathbb{R}^n$, but better separated if mapped to higher dimension space $\mathbb{R}^H$: $h : x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^H$
- Use hyper-planes to partition higher dim space: $f(x) = w \cdot h(x) + b$
- Example: $h : (x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$



Support vector machines: separable data

- Consider separable data in $\mathbb{R}^H$, and three parallel hyper-planes:

$$w \cdot h(x) + b = 0 \text{ (separating hyper-plane between red and blue)}$$
$$w \cdot h(x_1) + b = +1 \text{ (contains } h(x_1))$$
$$w \cdot h(x_2) + b = -1 \text{ (contains } h(x_2))$$



- Subtract blue from red:
  $w \cdot \big(h(x_1) - h(x_2)\big) = 2$
- With unit vector $\hat{w} = w/\|w\|$:
  $\hat{w} \cdot \big(h(x_1) - h(x_2)\big) = 2/\|w\| = m$
- Margin m is distance between red and blue planes
- Best separation: maximise margin
- $\Rightarrow$ empirical risk margin to minimise:
  $R(w) \propto \|w\|^2$

Support vector machines: constraints

- When minimising $R(w)$, need to keep signal and background separated
- Label red dots $y = +1$ ("above" red plane) and blue dots $y = -1$ ("below" blue plane)
- Since:

$$w \cdot h(x) + b > \quad 1 \text{ for red dots}$$
$$w \cdot h(x) + b < -1 \text{ for blue dots}$$

  all correctly classified points will satisfy constraints:

$$y_i\big(w \cdot h(x_i) + b\big) \geq 1, \ \forall i = 1, \ldots, N$$

- Using Lagrange multipliers $\alpha_i > 0$, cost function can be written:

$$C(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \big[y_i\big(w \cdot h(x_i) + b\big) - 1\big]$$

Support vector machines

Minimisation

- Minimise cost function $C(w, b, \alpha)$ with respect to w and b:

$$C(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \big(h(x_i) \cdot h(x_j)\big)$$

- At minimum of $C(\alpha)$, only non-zero α_i correspond to points on red and blue planes: support vectors

Kernel functions

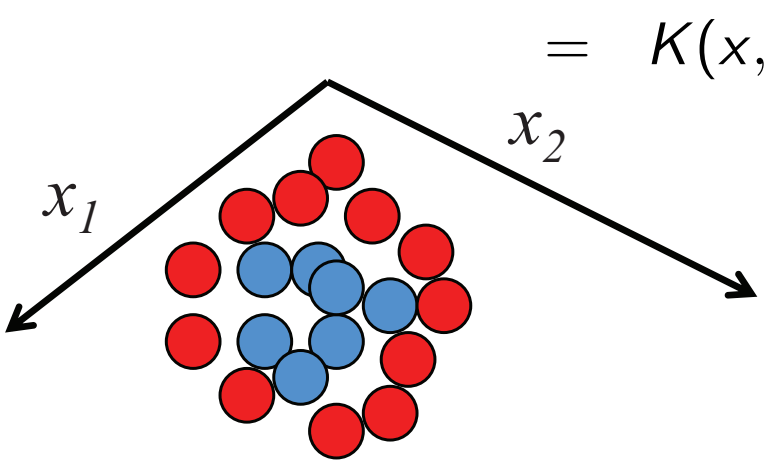
- Issues:
  - need to find h mappings (potentially of infinite dimension)
  - need to compute scalar products $h(x_i) \cdot h(x_j)$
- Fortunately $h(x_i) \cdot h(x_j)$ are equivalent to some kernel function $K(x_i, x_j)$ that does the mapping and the scalar product:

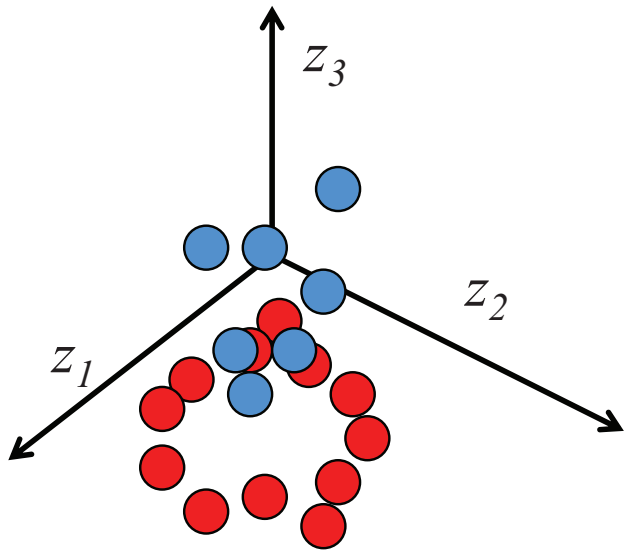
$$C(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Support vector machines: example

- $h : (x_1, x_2) \rightarrow (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

$$\begin{aligned} h(x) \cdot h(y) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2) \\ &= (x \cdot y)^2 \\ &= K(x, y) \end{aligned}$$



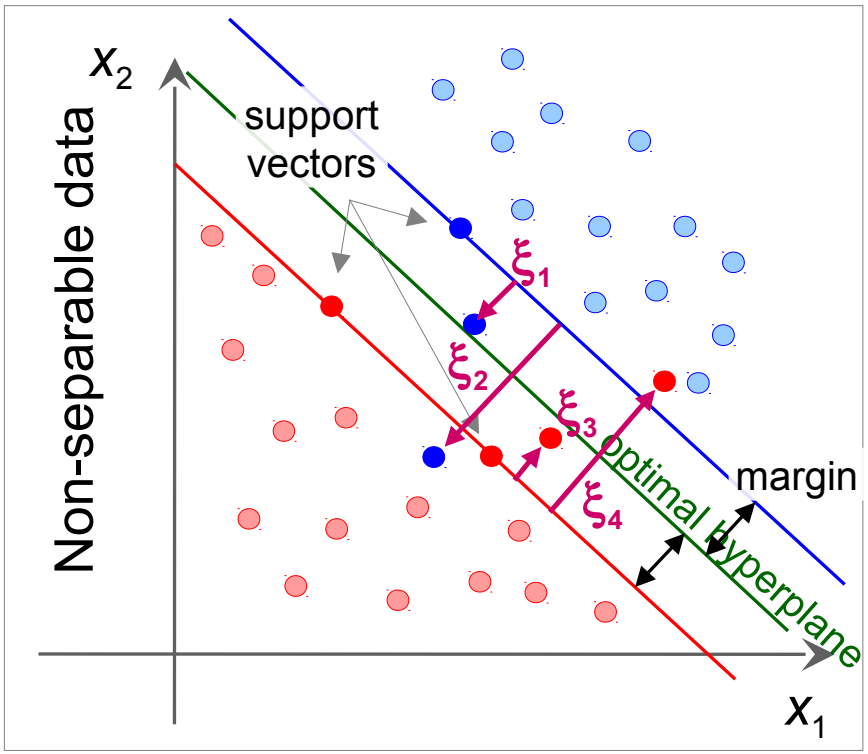


- In reality: do not know a priori the right kernel
- $\Rightarrow$ have to test different standard kernels and use the best one

Support vector machines: non-separable data

- Even in infinite dimension space, data are often non-separable
- Need to relax constraints:

$$y_i\left(w \cdot h(x_i) + b\right) \geq 1 - \xi_i$$



with slack variables $\xi_i > 0$

- $C(w, b, \alpha, \xi)$ depends on ξ, modified $C(\alpha, \xi)$ as well
- Values determined during minimisation

Splitting a node: Gini index of diversity

Defined for many classes

- Gini $= \sum_{i,j\in\{\text{classes}\}}^{i\neq j} p_i p_j$

Statistical interpretation

- Assign random object to class i with probability p_i.
- Probability that it is actually in class j is p_j
- $\Rightarrow$ Gini = probability of misclassification

For two classes (signal and background)

- $i = s, b$ and $p_s = p = 1 - p_b$
- $\Rightarrow$ Gini $= 1 - \sum_{i=s,b} p_i^2 = 2p(1-p) = \frac{2sb}{(s+b)^2}$
- Most popular in DT implementations
- Usually similar performance to e.g. entropy

Pruning a tree I

Pre-pruning

- Stop tree growth during building phase
- Already seen: minimum leaf size, minimum separation improvement, maximum depth, etc.
- Careful: early stopping condition may prevent from discovering further useful splitting

Expected error pruning

- Grow full tree
- When result from children not significantly different from result of parent, prune children
- Can measure statistical error estimate with binomial error $\sqrt{p(1-p)/N}$ for node with purity p and N training events
- No need for testing sample
- Known to be "too aggressive"

Pruning a tree II: cost-complexity pruning

- Idea: penalise "complex" trees (many nodes/leaves) and find compromise between good fit to training data (larger tree) and good generalisation properties (smaller tree)
- With misclassification rate $R(T)$ of subtree T (with N_T nodes) of fully grown tree T_{max}:

  $$\text{cost complexity } R_\alpha(T) = R(T) + \alpha N_T$$

  $\alpha =$ complexity parameter
- Minimise $R_\alpha(T)$:
  - small α: pick T_{max}
  - large α: keep root node only, T_{max} fully pruned
- First-pass pruning, for terminal nodes t_L, t_R from split of t:
  - by construction $R(t) \geq R(t_L) + R(t_R)$
  - if $R(t) = R(t_L) + R(t_R)$ prune off t_L and t_R

Pruning a tree III: cost-complexity pruning

- For node t and subtree T_t:
  - if t non-terminal, $R(t) > R(T_t)$ by construction
  - $R_\alpha(\{t\}) = R_\alpha(t) = R(t) + \alpha$ ($N_T = 1$)
  - if $R_\alpha(T_t) < R_\alpha(t)$ then branch has smaller cost-complexity than single node and should be kept
  - at critical $\alpha = \rho_t$, node is preferable
  - to find ρ_t, solve $R_{\rho_t}(T_t) = R_{\rho_t}(t)$, or: $\rho_t = \dfrac{R(t) - R(T_t)}{N_T - 1}$
  - node with smallest ρ_t is *weakest link* and gets pruned
  - apply recursively till you get to the root node
- This generates sequence of decreasing cost-complexity subtrees
- Compute their true misclassification rate on validation sample:
  - will first decrease with cost-complexity
  - then goes through a minimum and increases again
  - pick this tree at the minimum as the best pruned tree

- Note: best pruned tree may not be optimal in a forest

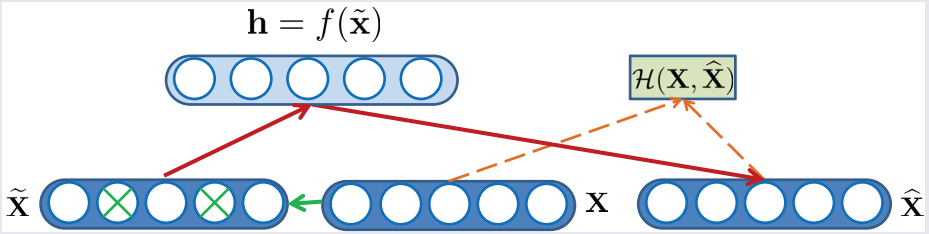
Other auto-encoders

Sparse auto-encoder

- Sparsity: try to have low activation of neurons (like in the brain)
- Compute average activation of each hidden unit over training set
- Add constraint to cost function to make average lower than some value close to 0

Denoising auto-encoder

- Stochastically corrupt inputs
- Train to reconstruct uncorrupted input



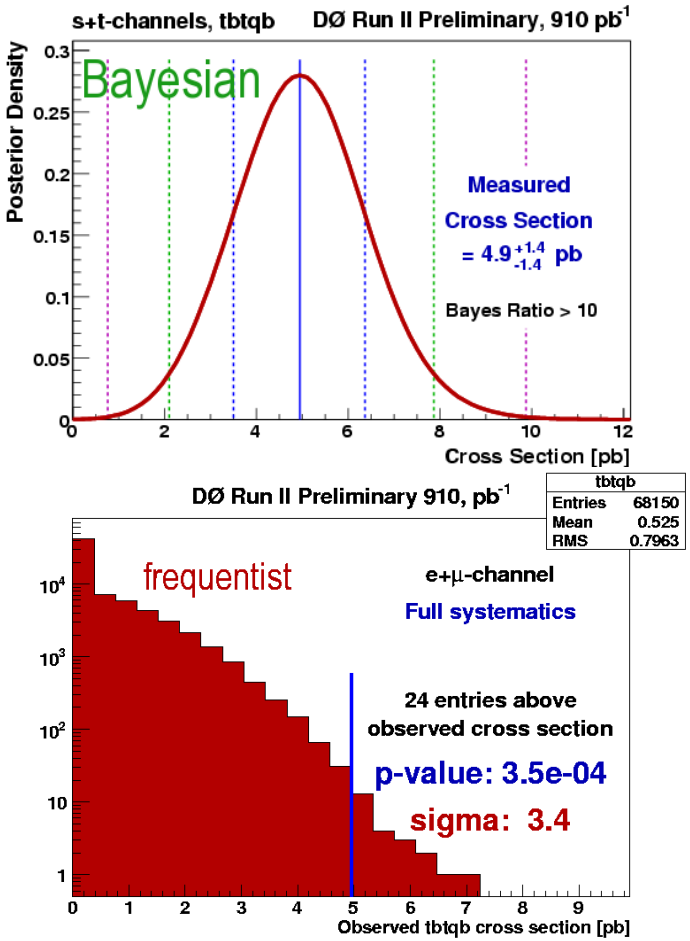
Locally connected auto-encoder

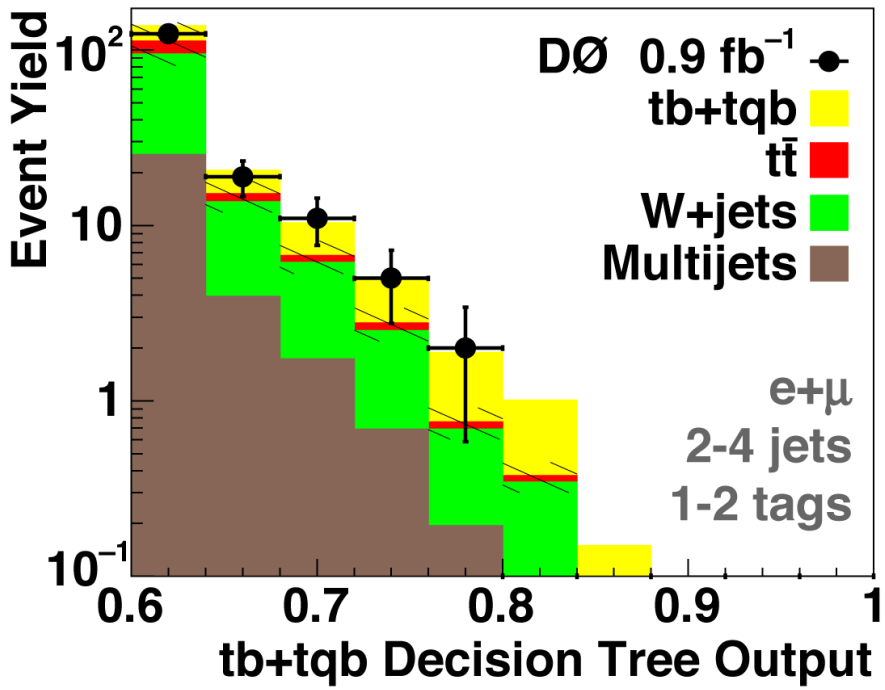
- Allow hidden units to connect only to small subset of input units
- Useful with increasing number of input features (e.g., bigger image)
- Inspired by biology: visual system has localised receptive fields

Single top production evidence at D0 (2006)

- Three multivariate techniques: BDT, Matrix Elements, BNN
- Most sensitive: BDT

$\sigma_{s+t} = 4.9 \pm 1.4$ pb
p-value $= 0.035\%$ (3.4σ)
SM compatibility: 11% (1.3σ)





$\sigma_s = 1.0 \pm 0.9$ pb
$\sigma_t = 4.2^{+1.8}_{-1.4}$ pb

▸ Phys. Rev. D**78**, 012005 (2008)

Decision trees — 49 input variables

Object Kinematics

- p_T(jet1)
- p_T(jet2)
- p_T(jet3)
- p_T(jet4)
- p_T(best1)
- p_T(notbest1)
- p_T(notbest2)
- p_T(tag1)
- p_T(untag1)
- p_T(untag2)

Angular Correlations

- ΔR(jet1,jet2)
- cos(best1,lepton)$_{\text{besttop}}$
- cos(best1,notbest1)$_{\text{besttop}}$
- cos(tag1,alljets)$_{\text{alljets}}$
- cos(tag1,lepton)$_{\text{btaggedtop}}$
- cos(jet1,alljets)$_{\text{alljets}}$
- cos(jet1,lepton)$_{\text{btaggedtop}}$
- cos(jet2,alljets)$_{\text{alljets}}$
- cos(jet2,lepton)$_{\text{btaggedtop}}$
- cos(lepton,Q(lepton)$\times z$)$_{\text{besttop}}$
- cos(lepton$_{\text{besttop}}$,besttop$_{\text{CMframe}}$)
- cos(lepton$_{\text{btaggedtop}}$,btaggedtop$_{\text{CMframe}}$)
- cos(notbest,alljets)$_{\text{alljets}}$
- cos(notbest,lepton)$_{\text{besttop}}$
- cos(untag1,alljets)$_{\text{alljets}}$
- cos(untag1,lepton)$_{\text{btaggedtop}}$

Event Kinematics

- Aplanarity(alljets,W)
- M(W,best1) ("best" top mass)
- M(W,tag1) ("b-tagged" top mass)
- H_T(alljets)
- H_T(alljets$-$best1)
- H_T(alljets$-$tag1)
- H_T(alljets,W)
- H_T(jet1,jet2)
- H_T(jet1,jet2,W)
- M(alljets)
- M(alljets$-$best1)
- M(alljets$-$tag1)
- M(jet1,jet2)
- M(jet1,jet2,W)
- M_T(jet1,jet2)
- M_T(W)
- Missing E_T
- p_T(alljets$-$best1)
- p_T(alljets$-$tag1)
- p_T(jet1,jet2)
- Q(lepton)$\times\eta$(untag1)
- $\sqrt{\hat{s}}$
- Sphericity(alljets,W)

- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels

Decision trees — 49 input variables

Object Kinematics

- p_T(jet1)
- p_T(jet2)
- p_T(jet3)
- p_T(jet4)
- p_T(best1)
- p_T(notbest1)
- p_T(notbest2)
- p_T(tag1)
- p_T(untag1)
- p_T(untag2)

Angular Correlations

- ΔR(jet1,jet2)
- cos(best1,lepton)$_{\text{besttop}}$
- cos(best1,notbest1)$_{\text{besttop}}$
- cos(tag1,alljets)$_{\text{alljets}}$
- cos(tag1,lepton)$_{\text{btaggedtop}}$
- cos(jet1,alljets)$_{\text{alljets}}$
- cos(jet1,lepton)$_{\text{btaggedtop}}$
- cos(jet2,alljets)$_{\text{alljets}}$
- cos(jet2,lepton)$_{\text{btaggedtop}}$
- cos(lepton,Q(lepton)$\times z$)$_{\text{besttop}}$
- cos(lepton$_{\text{besttop}}$,besttop$_{\text{CMframe}}$)
- cos(lepton$_{\text{btaggedtop}}$,btaggedtop$_{\text{CMframe}}$)
- cos(notbest,alljets)$_{\text{alljets}}$
- cos(notbest,lepton)$_{\text{besttop}}$
- cos(untag1,alljets)$_{\text{alljets}}$
- cos(untag1,lepton)$_{\text{btaggedtop}}$

Event Kinematics

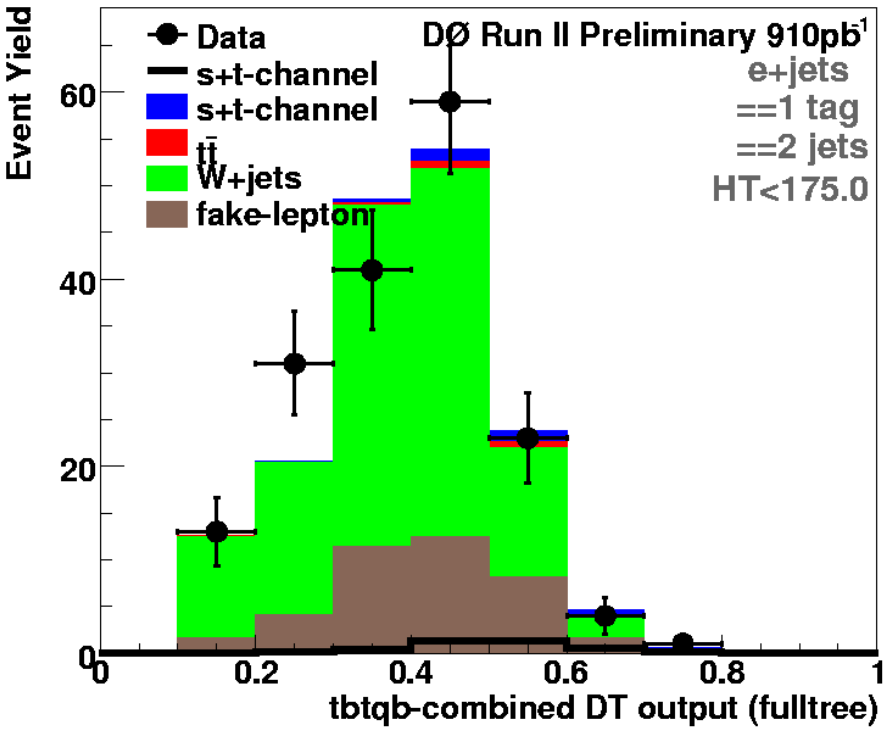
- Aplanarity(alljets,W)
- M(W,best1) ("best" top mass)
- M(W,tag1) ("b-tagged" top mass)
- H_T(alljets)
- H_T(alljets−best1)
- H_T(alljets−tag1)
- H_T(alljets,W)
- H_T(jet1,jet2)
- H_T(jet1,jet2,W)
- M(alljets)
- M(alljets−best1)
- M(alljets−tag1)
- M(jet1,jet2)
- M(jet1,jet2,W)
- M_T(jet1,jet2)
- M_T(W)
- Missing E_T
- p_T(alljets−best1)
- p_T(alljets−tag1)
- p_T(jet1,jet2)
- Q(lepton)$\times\eta$(untag1)
- $\sqrt{\hat{s}}$
- Sphericity(alljets,W)

- Adding variables did not degrade performance
- Tested shorter lists, lost some sensitivity
- Same list used for all channels
- Best theoretical variable: H_T(alljets,W). But detector not perfect ⇒ capture the essence from several variations usually helps "dumb" MVA

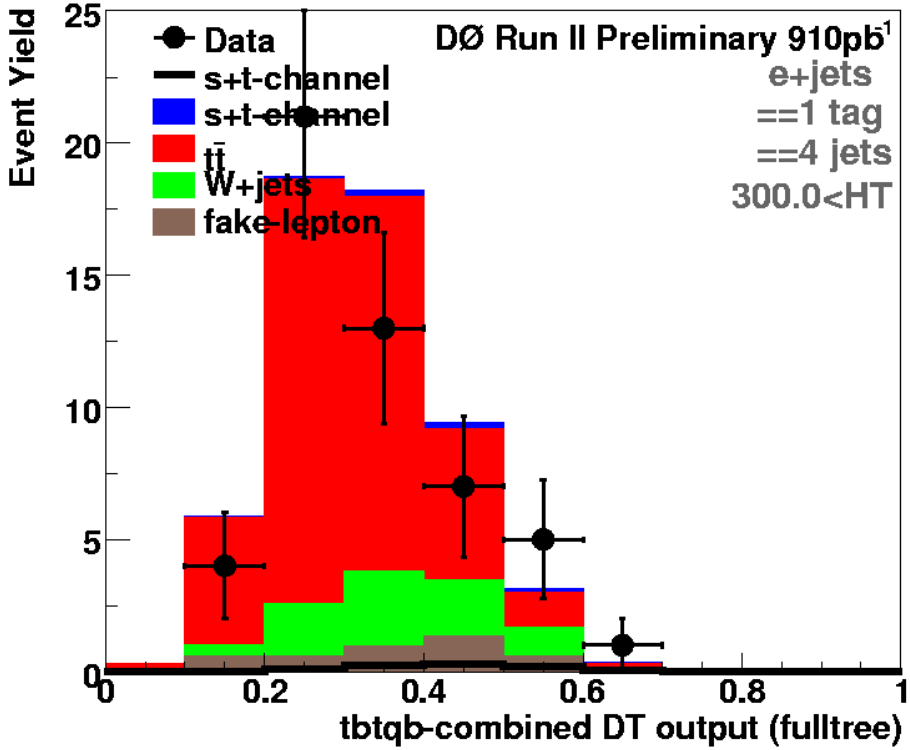
Cross-check samples

- Validate method on data in no-signal region

- **"W+jets"**: = 2 jets, $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) < 175$ GeV



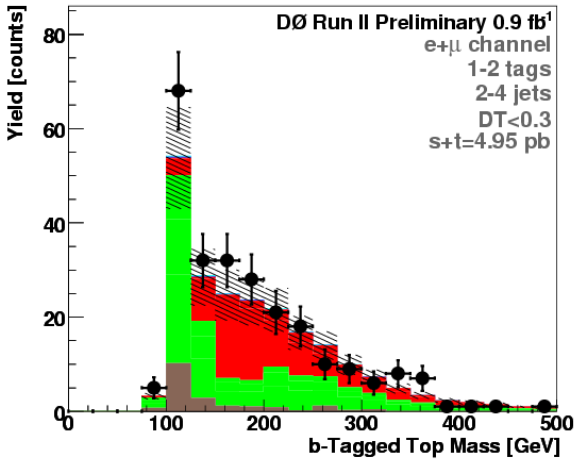
- **"ttbar"**: = 4 jets, $H_T(\text{lepton}, E_T^{\text{miss}}, \text{alljets}) > 300$ GeV

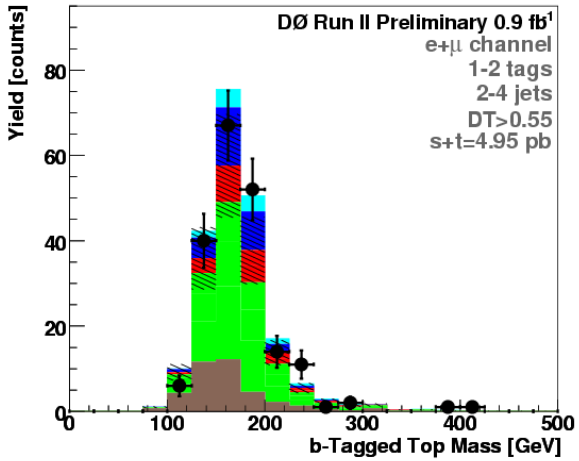


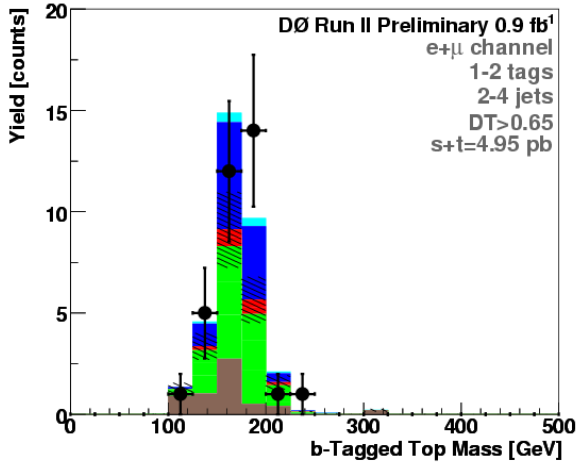
- Good agreement

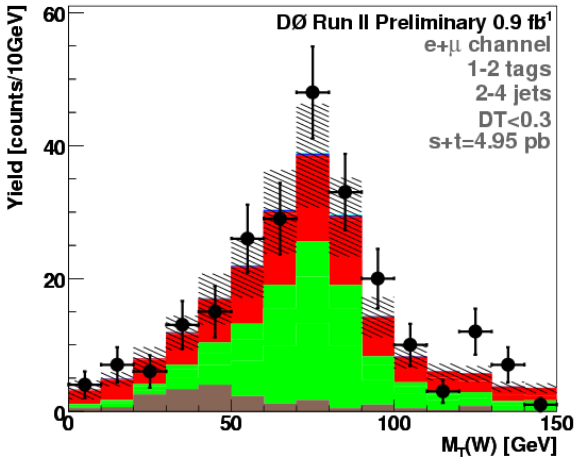
Boosted decision tree event characteristics

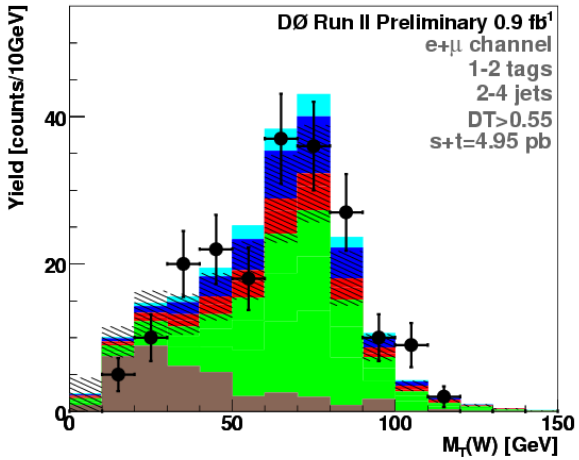
$DT < 0.3$ | $DT > 0.55$ | $DT > 0.65$

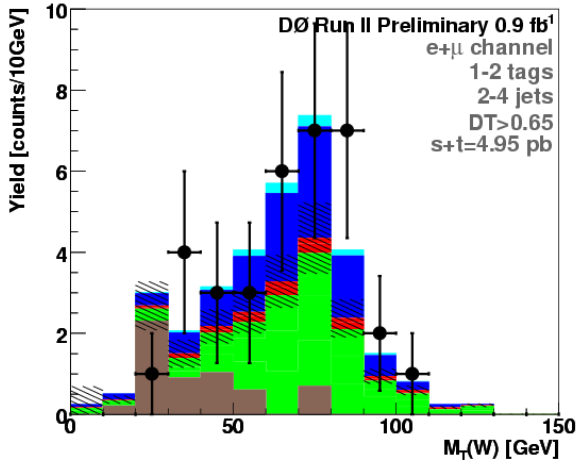








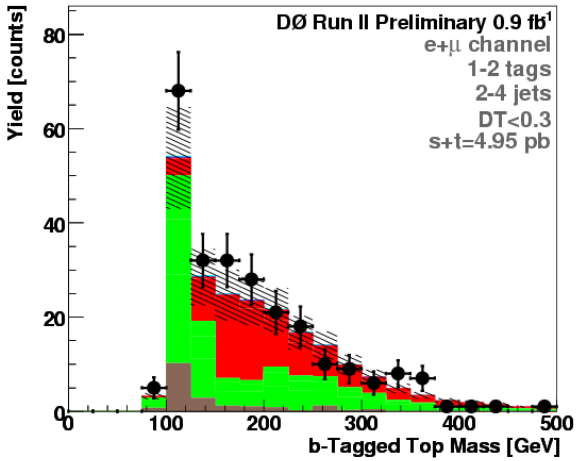


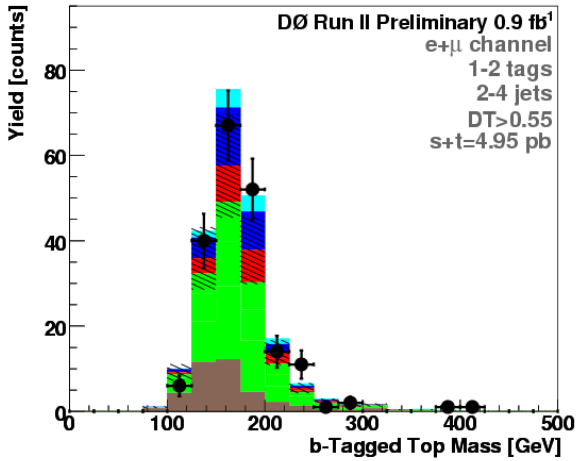


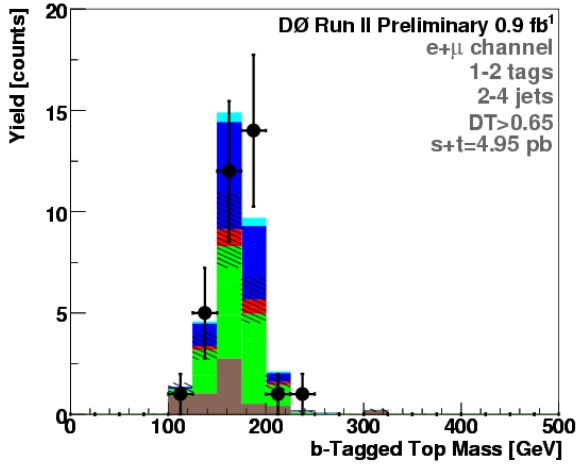
- High BDT region = shows masses of real t and W $\Rightarrow$ expected
- Low BDT region = background-like $\Rightarrow$ expected

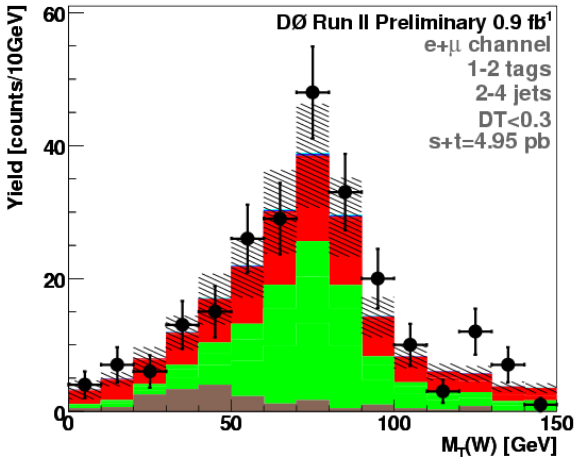
Boosted decision tree event characteristics

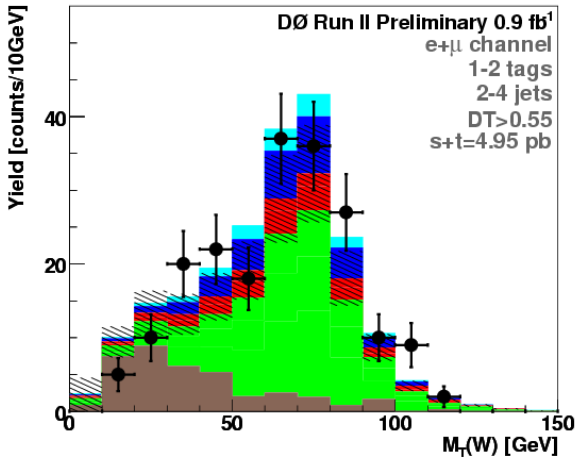
$DT < 0.3$ | $DT > 0.55$ | $DT > 0.65$

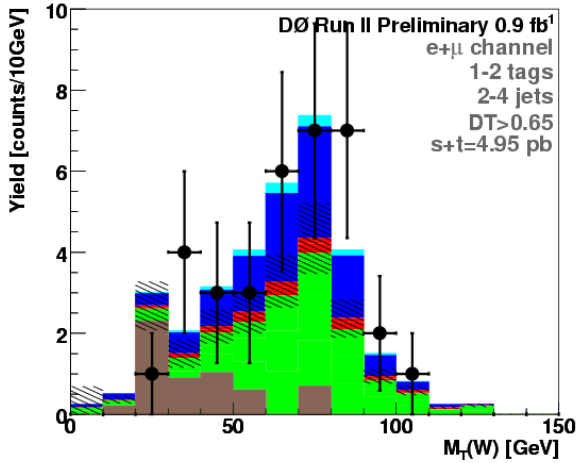






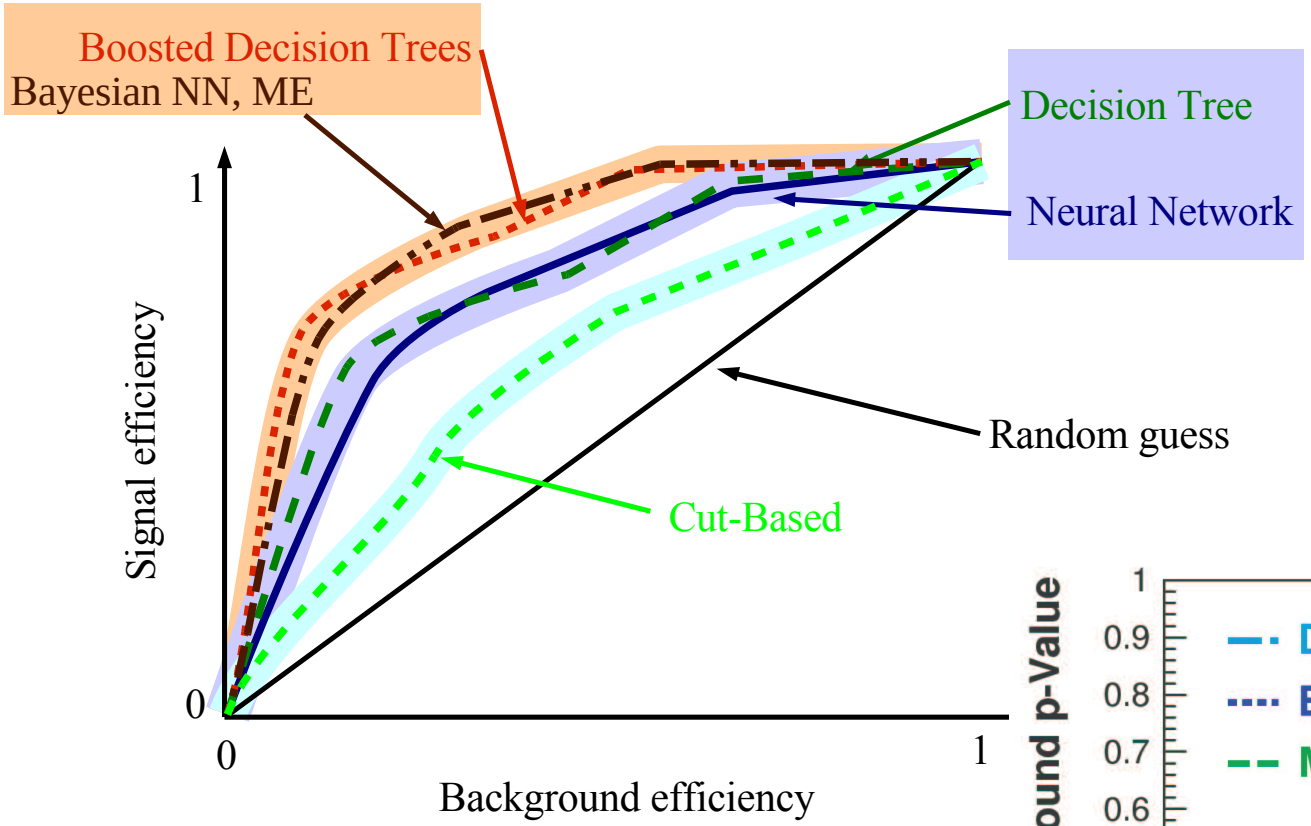




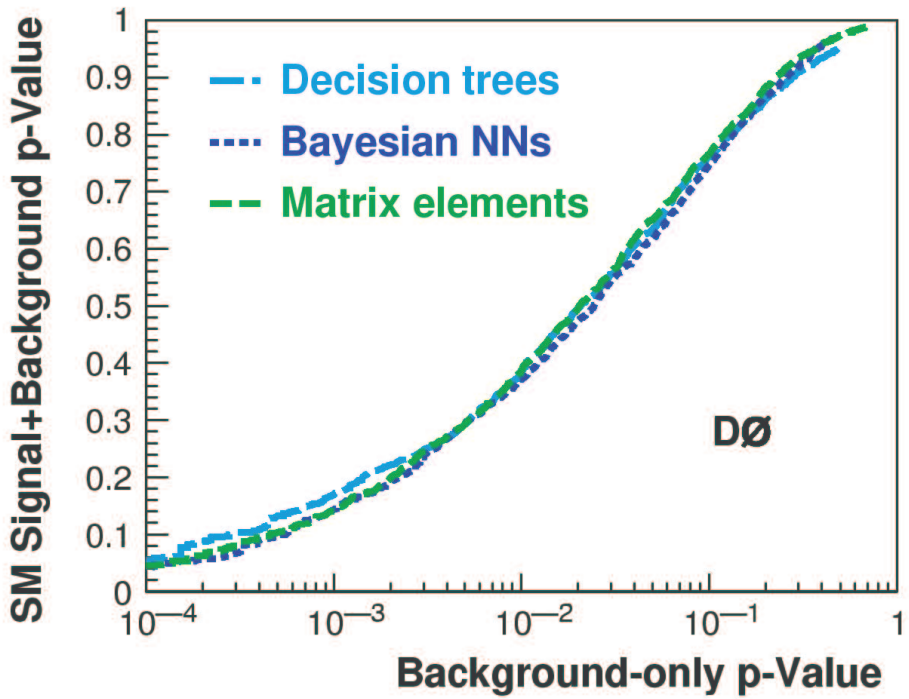


- High BDT region = shows masses of real t and W $\Rightarrow$ expected
- Low BDT region = background-like $\Rightarrow$ expected
- Above does NOT tell analysis is ok, but not seeing this could be a sign of a problem

Comparison for D0 single top evidence

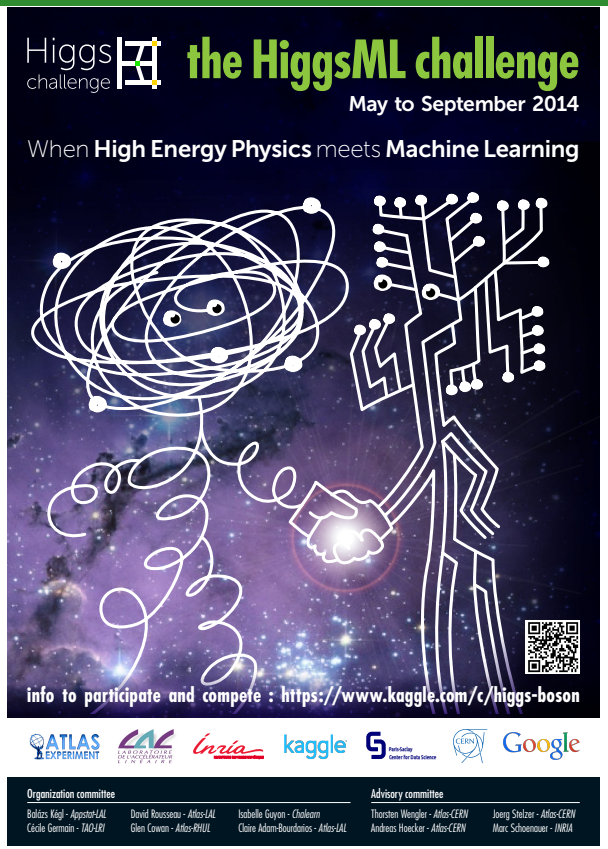


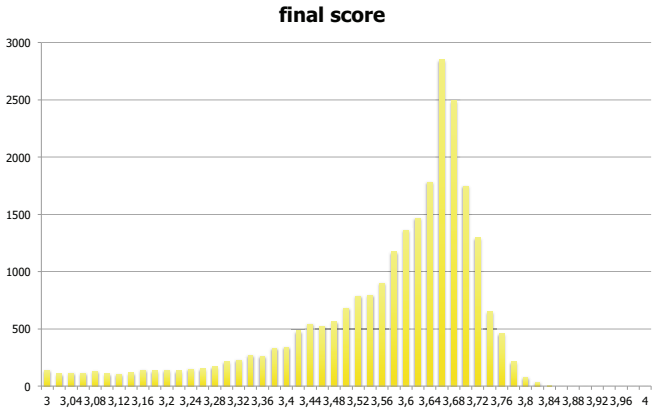
- Cannot know *a priori* which method will work best
- $\Rightarrow$ Need to experiment with different techniques



Power curve

HiggsML Kaggle challenge





HiggsML challenge

- Put ATLAS Monte Carlo samples on the web ($H \to \tau\tau$ analysis)
- Compete for best signal–bkg separation
- 1785 teams (most popular challenge ever)
- 35772 uploaded solutions
- See ▸ Kaggle web site and ▸ more information

#	Δrank	Team Name ‡ model uploaded * in the money		Score	Entries	Last Submission UTC (Best – Last Submission)
1	↑1	Gábor Melis ‡ *	7000$	3.80581	110	Sun, 14 Sep 2014 09:10:04 (-0h)
2	↑1	Tim Salimans ‡ *	4000$	3.78913	57	Mon, 15 Sep 2014 23:49:02 (-40.6d)
3	↑1	nhlx5haze ‡ *	2000$	3.78682	254	Mon, 15 Sep 2014 16:50:01 (-76.3d)
4	↑38	ChoKo Team		3.77526	216	Mon, 15 Sep 2014 15:21:36 (-42.1h)
5	↑35	cheng chen		3.77384	21	Mon, 15 Sep 2014 23:29:29 (-0h)
6	↑16	quantify		3.77086	8	Mon, 15 Sep 2014 16:12:48 (-7.3h)
7	↑1	Stanislav Semenov & Co (HSE Yandex)		3.76211	68	Mon, 15 Sep 2014 20:19:03
8	↓7	Luboš Motl's team		3.76050	589	Mon, 15 Sep 2014 08:38:49 (-1.6h)
9	↑8	Roberto-UCIIIM		3.75864	292	Mon, 15 Sep 2014 23:44:42 (-44d)
10	↑2	Davut & Josef		3.75838	161	Mon, 15 Sep 2014 23:24:32 (-4.5d)
45	↑5	crowwork ‡	HEP meets ML award Free trip to CERN	3.71885	94	Mon, 15 Sep 2014 23:45:00 (-5.1d)
782	↓149	Eckhard	TMVA expert, with TMVA improvements	3.49945	29	Mon, 15 Sep 2014 07:26:13 (-46.1h)
991	↑4	Rem.		3.20423	2	Mon, 16 Jun 2014 21:53:43 (-30.4h)
		simple TMVA boosted trees		3.19956		

Introduction to TMVA (ROOT v6.20.06)

- *TMVA*: Toolkit for MultiVariate Analysis
  - https://root.cern/tmva
  - https://github.com/root-project/root/tree/master/tmva
- Written by physicists
- In C++ (also python API), integrated in ROOT
- Quite complete manual
- Includes many different multivariate/machine learning techniques
- To compile, add appropriate header files in your code (e.g., `#include "TMVA/Factory.h"`) and this to your compiler command line: `` `root-config --cflags --libs` -lTMVA ``
- More complete examples of code: `$ROOTSYS/tutorials/tmva`
  - `createData.C` macro to make example datasets
  - classification and regression macros
  - also includes Keras examples (deep learning)
- Sometimes useful performance measures (more in these headers):

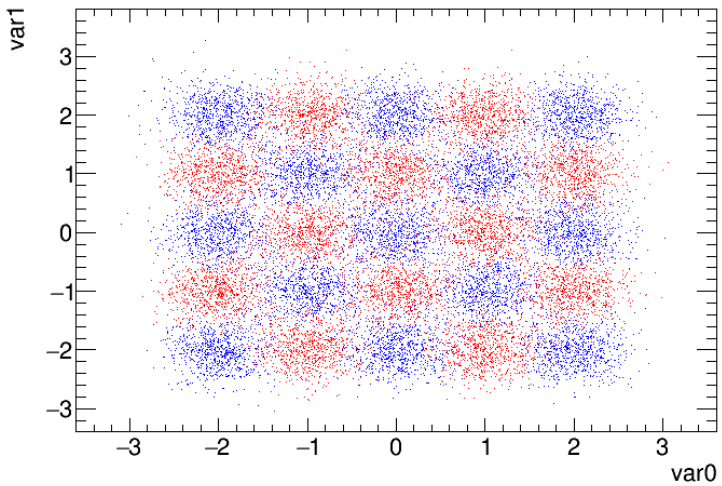
```
#include "TMVA/ROCCalc.h"
TMVA::ROCCalc(TH1* S,TH1* B).GetROCIntegral();
#include "TMVA/Tools.h"
TMVA::gTools().GetSeparation(TH1* S,TH1* B);
```

Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
```

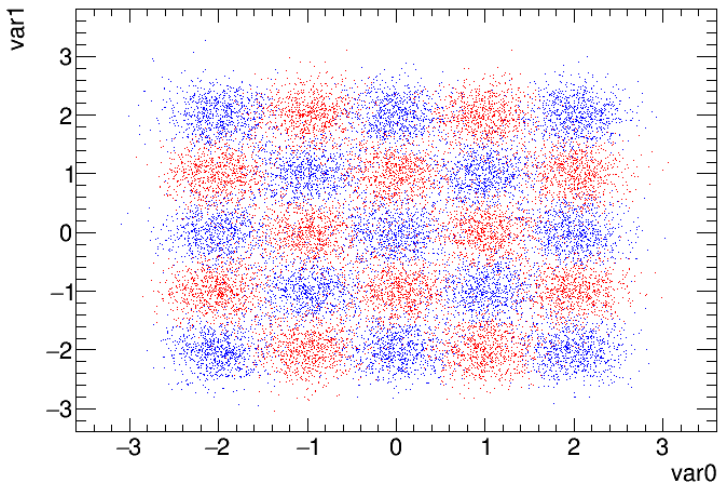
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```



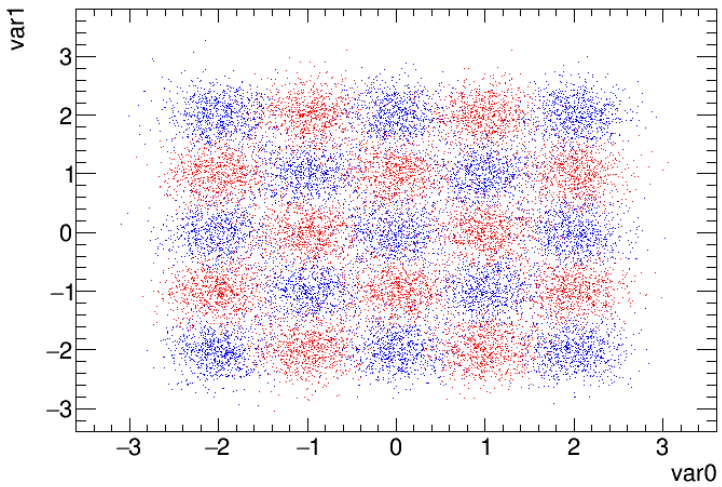
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
```



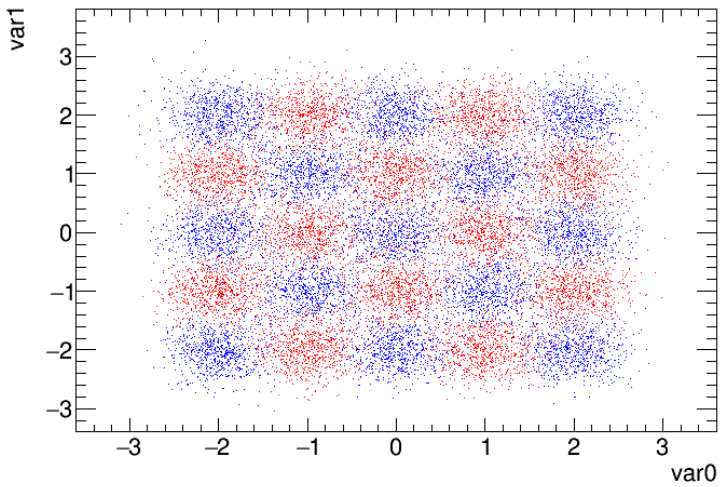
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
```



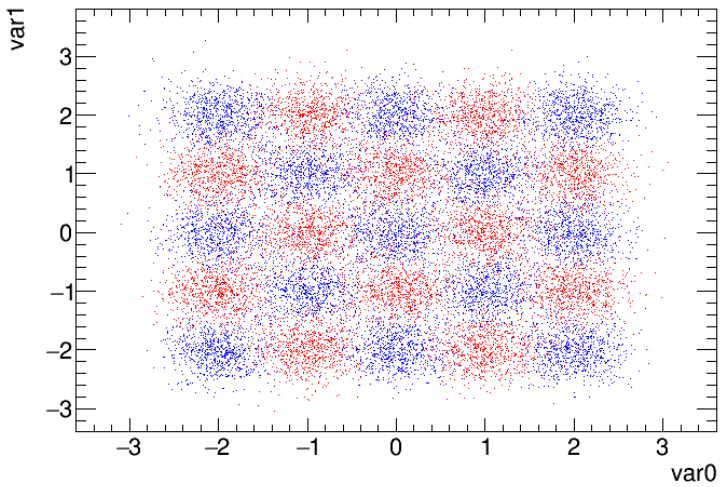
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
```



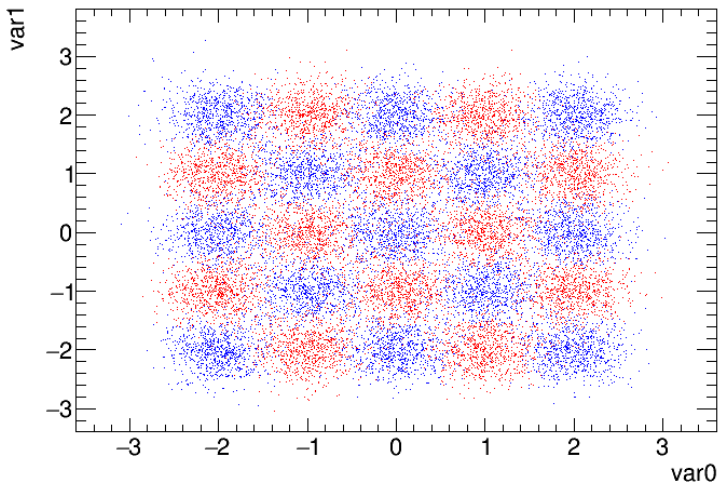
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
```



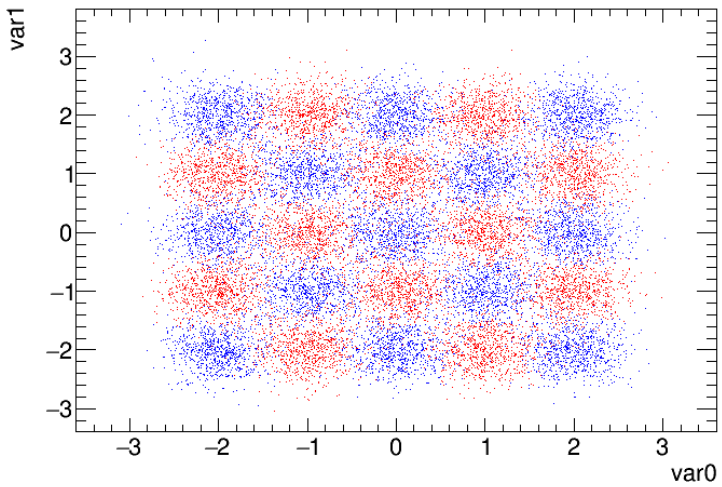
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
```



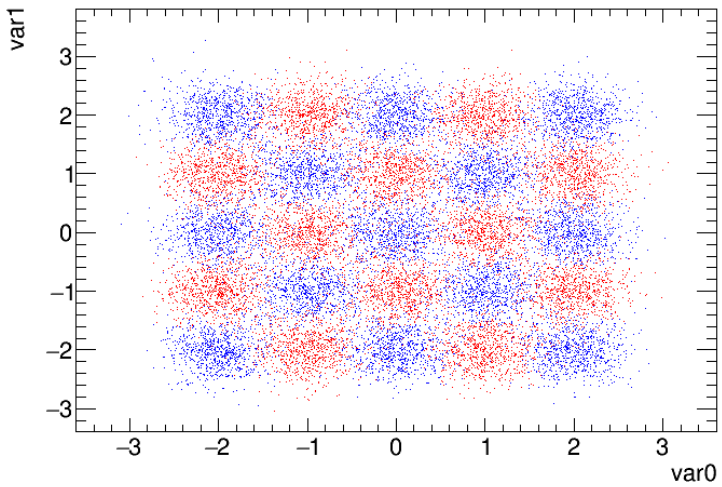
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
```



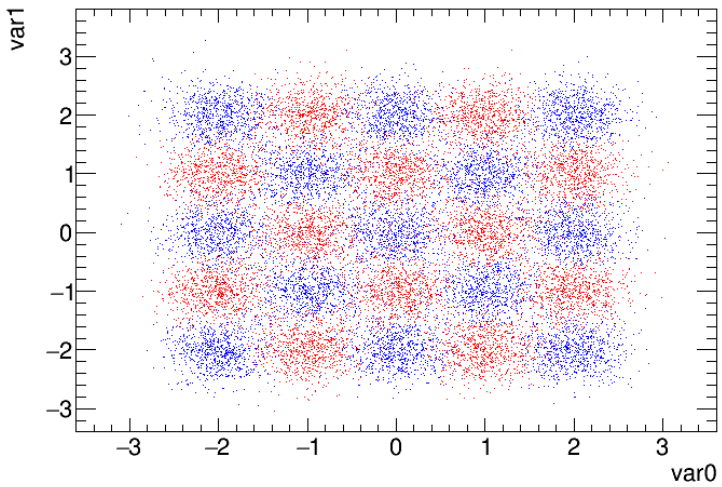
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
```



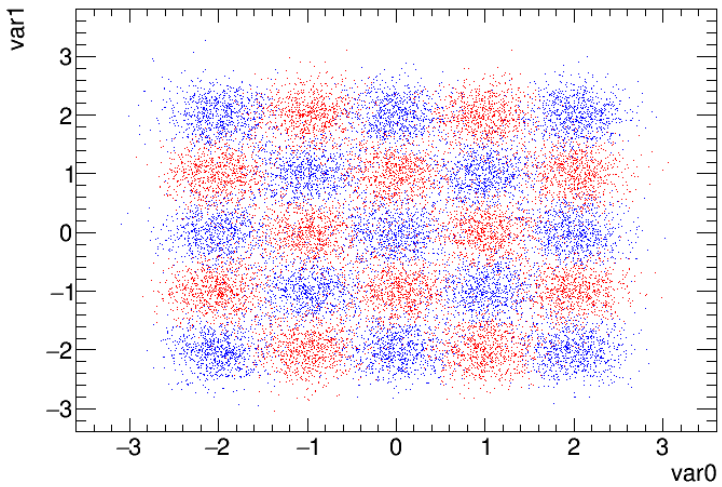
Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;
```



Training with TMVA (Train.C)

```
TFile* outputFile = TFile::Open("output.root","RECREATE");
TMVA::Factory *factory = new TMVA::Factory( "TMVAClassification", outputFile,
    "!V:Color:DrawProgressBar:Transformations=I:AnalysisType=Classification");
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* sig = (TTree*)inputFile->Get("TreeS");
TTree* bkg = (TTree*)inputFile->Get("TreeB");
double sigWeight = 1.0; double bkgWeight = 1.0;
TMVA::DataLoader *dataloader =
    new TMVA::DataLoader("dataset");
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
dataloader->AddVariable("var0", 'F');
dataloader->AddVariable("var1", 'F');
TCut mycut = "";
dataloader->PrepareTrainingAndTestTree(mycut,"SplitMode=Random");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:
    MinNodeSize=4%:MaxDepth=5:BoostType=AdaBoost:AdaBoostBeta=0.15:nCuts=80");
factory->BookMethod(dataloader, TMVA::Types::kFisher, "Fisher", "!H:!V:Fisher");
factory->TrainAllMethods(); // Train MVAs using training events
factory->TestAllMethods(); // Evaluate all MVAs using test events
// ----- Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();
auto c1 = factory->GetROCCurve(dataloader); // Eager to compare performance
outputFile->Close();
delete factory; delete dataloader;          TMVA::TMVAGui("output.root");
```



Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ------- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
  data->GetEntry(ievt);
  double bdt = reader->EvaluateMVA("My BDT");
  double fisher = reader->EvaluateMVA("Fisher discriminant");
  cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();
```

Apply classifier with TMVA (Apply.C)

```
TFile* inputFile = new TFile("dataSchachbrett.root");
TTree* data = (TTree*)inputFile->Get("TreeS");
Float_t var0=-99., var1=-99.;
data->SetBranchAddress("var0", &var0);
data->SetBranchAddress("var1", &var1);
TMVA::Reader *reader = new TMVA::Reader();
reader->AddVariable( "var0", &var0 );
reader->AddVariable( "var1", &var1 );
reader->BookMVA( "My BDT", "dataset/weights/TMVAClassification_BDT.weights.xml");
reader->BookMVA( "Fisher discriminant",
  "dataset/weights/TMVAClassification_Fisher.weights.xml");
// ------- start your event loop
for (Long64_t ievt=0; ievt<10; ++ievt) {
  data->GetEntry(ievt);
  double bdt = reader->EvaluateMVA("My BDT");
  double fisher = reader->EvaluateMVA("Fisher discriminant");
  cout<<"var0="<<var0<<" var1="<<var1<<" BDT="<<bdt<<" Fisher="<<fisher<<endl;
}
delete reader;
inputFile->Close();
```

- More complete tutorials:
  - https://github.com/lmoneta/tmva-tutorial

Compiling TMVA with C++

- To make code compilable (and MUCH faster)
  - Need ROOT and TMVA corresponding header files
  - e.g., for Train.C:

```
#include "TFile.h"
#include "TTree.h"
#include "TMVA/Factory.h"
#include "TMVA/DataLoader.h"
#include "TMVA/TMVAGui.h"
```

  - Need a "main" function

```
int main() {
  Train();
  return 0;
}
```

  - Compilation:

```
g++  Train.C `root-config --cflags --libs` -lTMVA  -lTMVAGui  -o  TMVATrainer
```

  - Train.C: file to compile
  - TMVATrainer: name of executable
  - -lTMVAGui: just because of `TMVA::TMVAGui("output.root");`

TMVA: training refinements

- Common technique: train on even event numbers, test on odd event numbers (and vice versa)
- Can also think of more than two-fold
- Achieve in TMVA by replacing:

```
dataloader->AddSignalTree(sig, sigWeight);
dataloader->AddBackgroundTree(bkg, bkgWeight);
```

- with:

```
TString trainString = "(eventNumber % 2 == 0)";
TString testString = "!"+trainString;
dataloader->AddTree(sig, "Signal", sigWeight, trainString.Data(), "Training");
dataloader->AddTree(sig, "Signal", sigWeight, testString.Data(), "Test");
dataloader->AddTree(bkg, "Background", bkgWeight, trainString.Data(), "Training");
dataloader->AddTree(bkg, "Background", bkgWeight, testString.Data(), "Test");
```

- Use individual event weights:

```
string eventWeight = "TMath::Abs(eventWeight)"; //Compute event weight
dataloader->SetSignalWeightExpression(eventWeight);
dataloader->SetBackgroundWeightExpression(eventWeight); //Can differ
```