

Multivariate analysis

Yann Coadou

CPPM Marseille



European School of Instrumentation
in Particle & Astroparticle Physics

Archamps, 6 February 2017

Thanks to Harrison Prosper, Balázs Kégl, Jérôme Schwindling, Jan Therhaag





esipap...
European School of Instrumentation
in Particle & Astroparticle Physics

**PHYSICS OF PARTICLE AND
ASTROPARTICLE DETECTORS**
— 23 January to 17 February 2017
4 WEEK COURSE

TECHNOLOGIES & APPLICATIONS
— 20 February to 17 March 2017
4 SEPARATE MODULES :

Module 1	Module 2	Module 3	Module 4
Detector Technologies & Electronics	Real Time Computing & Data Handling	Mechanics & Medical Applications	Offline Computing
20 to 24 February	20 to 24 February	27 February to 5 March	15 to 17 March

www.esi-archamps.eu

APPLY NOW

ESIPAP - octobre 2016
Credits photo : ESIPAP - Cam
www.indelectable-creation.com



- 1 Introduction
- 2 Optimal discrimination
 - Bayes limit
 - Multivariate discriminant
- 3 Machine learning
 - Supervised and unsupervised learning
- 4 Multivariate discriminants
 - Random grid search
 - Genetic algorithms
 - Quadratic and linear discriminants
 - Support vector machines
 - Kernel density estimation
 - Neural networks
 - Bayesian neural networks
 - Deep networks
 - Decision trees
- 5 Summary

Typical problems in HEP

- Classification of objects
 - separate real and fake leptons/jets/etc.
- Signal enhancement relative to background
- Regression: best estimation of a parameter
 - lepton energy, \cancel{E}_T value, invariant mass, 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, ...)

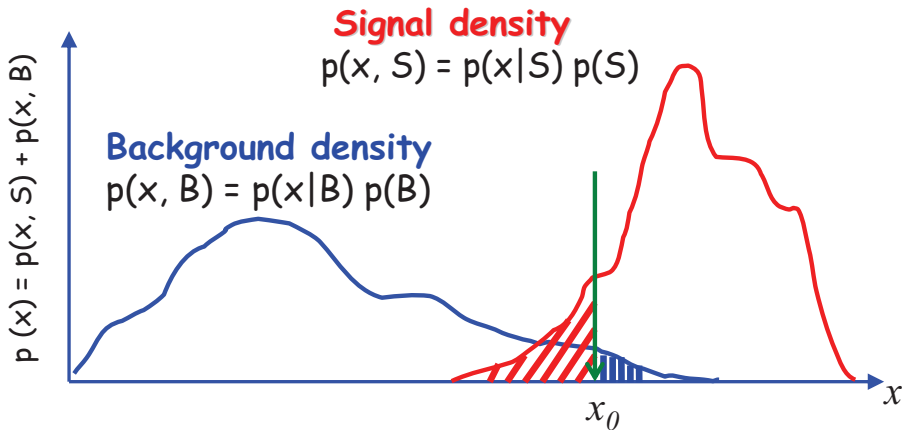
Input information from various sources

- Kinematic variables (masses, momenta, decay angles, ...)
- Event properties (jet multiplicity, sum of charges, brightness ...)
- Event shape (sphericity, aplanarity, ...)
- Detector response (silicon hits, dE/dx , Cherenkov angle, shower profiles, muon hits, ...)

Most data are (highly) multidimensional

- Use dependencies between $x = \{x_1, \dots, x_n\}$ 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 $f(x) \leq q$

- Where to place a cut x_0 on variable x ?

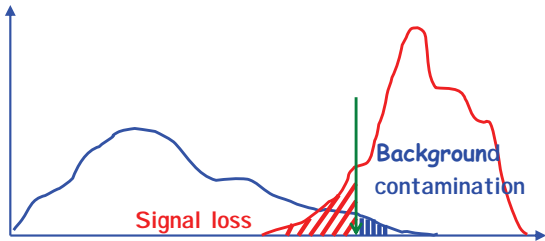


- Optimal choice: minimum misclassification cost at decision boundary
 $x = x_0$

$$C(x_0) = \underbrace{C_S \int H(x_0 - x)p(x, S)dx}_{\text{signal loss}} + \underbrace{C_B \int H(x - x_0)p(x, B)dx}_{\text{background contamination}}$$

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

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**:

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

Probability relationships

- $p(A, B) = p(A|B)p(B) = p(B|A)p(A)$
- Bayes theorem: $p(A|B)p(B) = p(B|A)p(A)$
- $p(S|x) + p(B|x) = 1$

Generalising to multidimensional problem

- The same holds when x is an n -dimensional variable:

$$BD = B \frac{p(S)}{p(B)} \quad \text{where} \quad B = \frac{p(x|S)}{p(x|B)}$$

- B is the **Bayes factor**, identical to the likelihood ratio when class densities $p(x|S)$ and $p(x|B)$ are independent of unknown parameters

Bayes limit

- $p(S|x) = BD/(1 + BD)$ is what should be achieved to minimise cost, achieving classification with the 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**

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:

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

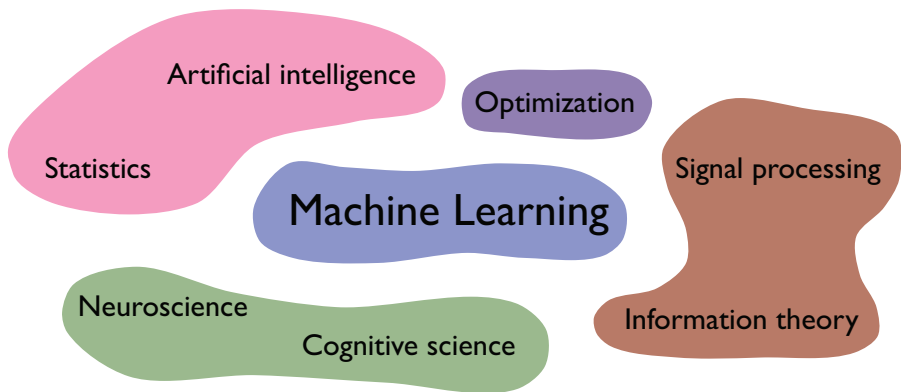
- Cutting on $D(x)$ is equivalent to cutting on $p(S|x)$, implying a corresponding (unknown) cut on $p(S|x)$

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 (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
⇒ generalisation ability



©Balázs Kégl

Machine learning: (un)supervised learning

Supervised learning

- Training events are labelled: N examples $(x, y)_1, (x, y)_2, \dots, (x, y)_N$ of (discriminating) **feature variables** x and **class labels** y
- The learner uses example classes to know how good it is doing

Reinforcement learning

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

Unsupervised learning

- e.g. clustering: find similarities in training sample, without having predefined categories (how Amazon is recommending you books. . .)
- Discover good internal representation of the input
- Not biased by pre-determined classes \Rightarrow may discover unexpected features!

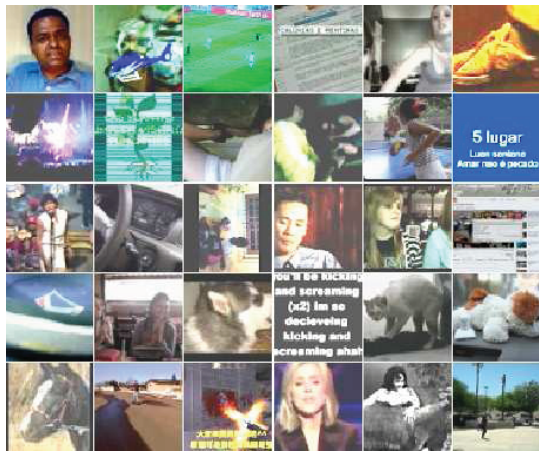
A “giant” neural network

- At Google they trained a 9-layered NN with 1 billion connections
 - trained on 10 million 200×200 pixel images from YouTube videos
 - on 1000 machines (16000 cores) for 3 days, unsupervised learning
- Sounds big? The human brain has 100 billion (10^{11}) neurons and 100 trillion (10^{14}) connections...

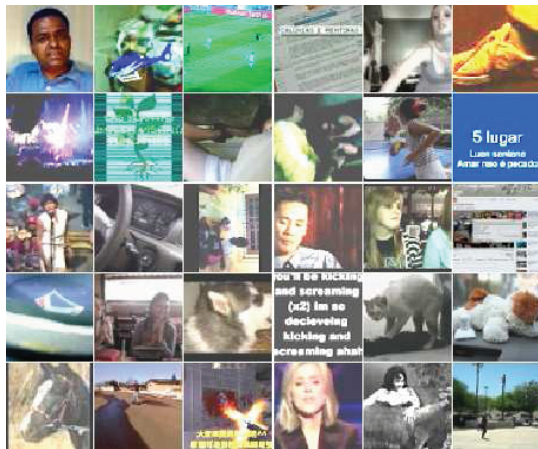
What it did

- It learned to recognise faces, one of the original goals
- ... but also cat faces (among the most popular things in YouTube videos) and body shapes





- Features extracted from such images
- Results shown to be robust to
 - colour
 - translation
 - scaling
 - out-of-plane rotation



- Features extracted from such images
- Results shown to be robust to
 - colour
 - translation
 - scaling
 - out-of-plane rotation

Deep networks

- More details towards the [end of the lecture](#)

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

- Given our N examples $(x, y)_1, \dots, (x, y)_N$ we need
 - a function class $\mathbb{F} = \{f(x, w)\}$ (w : parameters to be found)
 - a constraint $Q(w)$ on \mathbb{F}
 - 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**:

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

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

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

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

Loss function in regression

- Goal: set $f(x, w)$ as close as possible to y
- Therefore, loss increases with difference between $f(x, w)$ and y
- Most widely used loss function is **quadratic loss**:

$$L(y, f) = (f(x, w) - y)^2$$

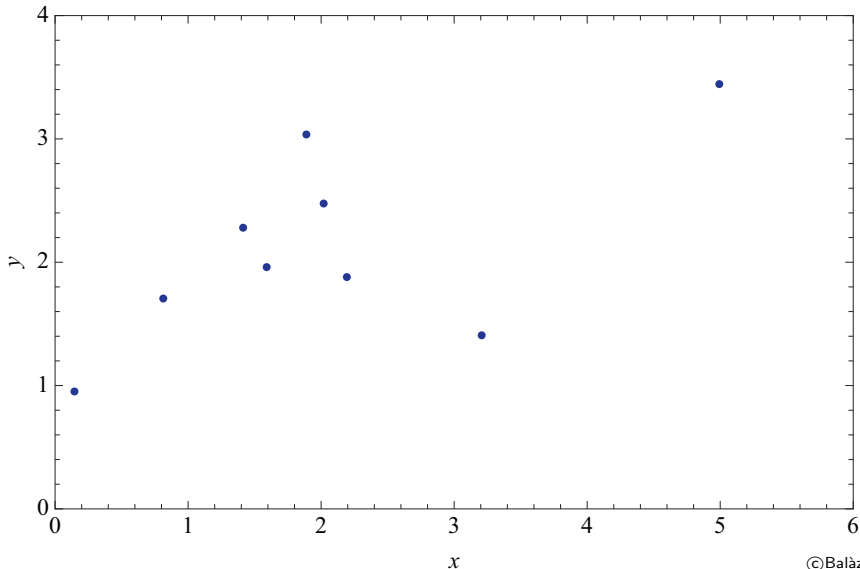
Loss function in classification

- There is no “distance” between classes
- Goal: $f(x, w)$ predicts properly class y
- Usual loss function is **one-loss** or **zero-one loss**:

$$L(y, f) = \mathbb{I}(f(x, w) \neq y)$$

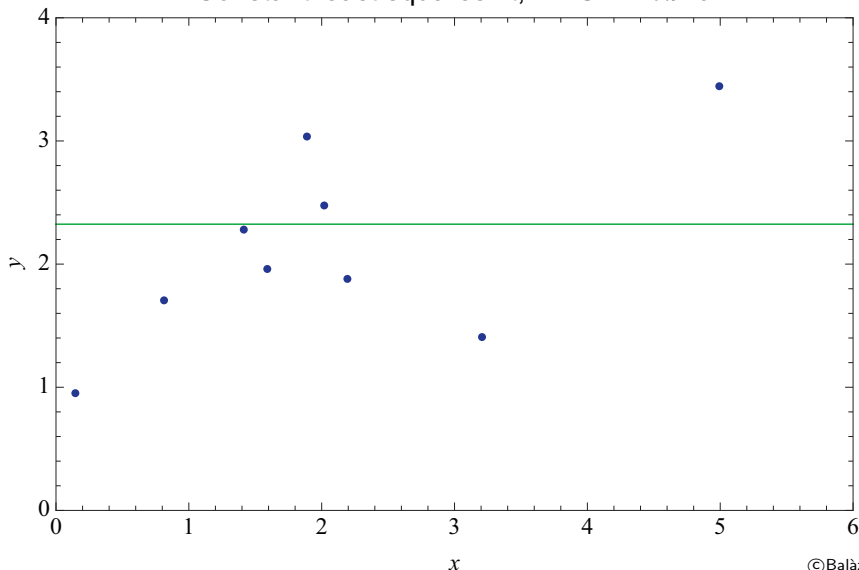
where indicator function $\mathbb{I}(X) = 1$ if X is true, 0 otherwise

Data generated from an unknown function with unknown noise



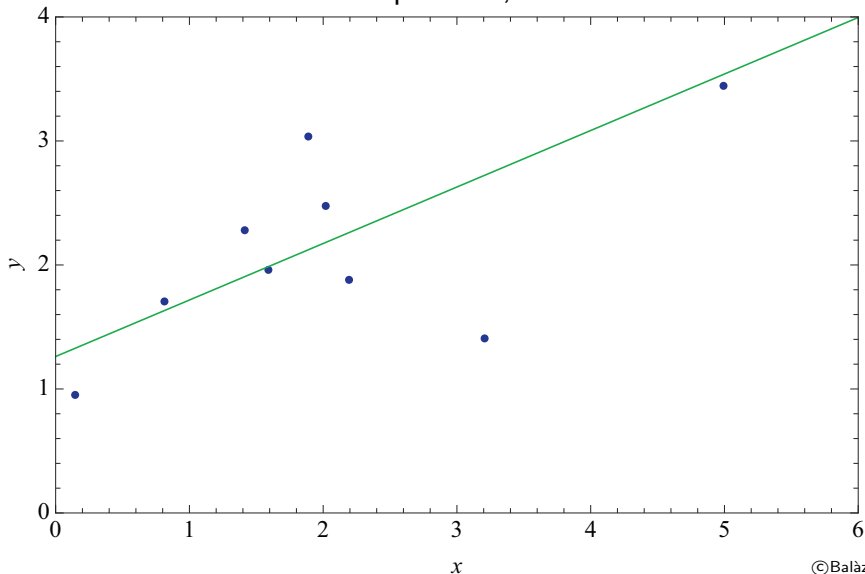
© Balázs Kégl

Constant least squares fit, RMSE = 0.915



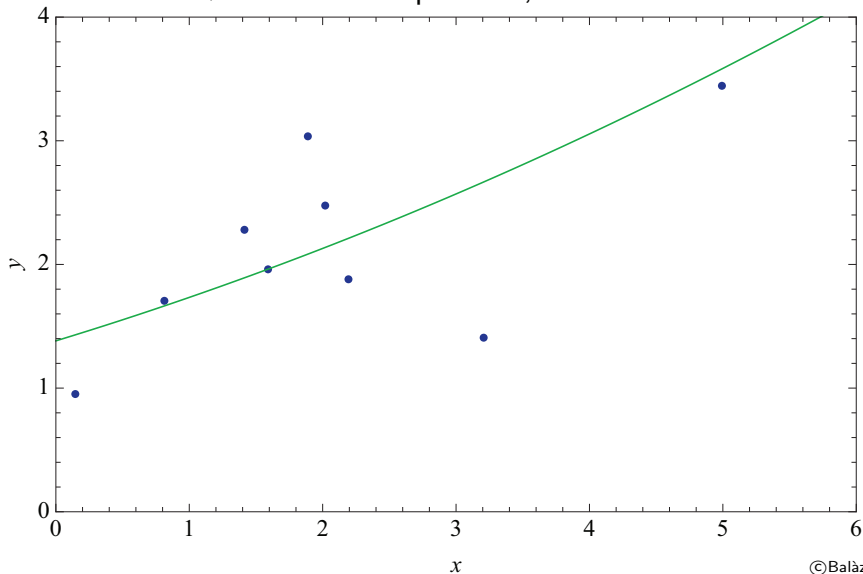
©Balázs Kégl

Linear least squares fit, RMSE = 0.581



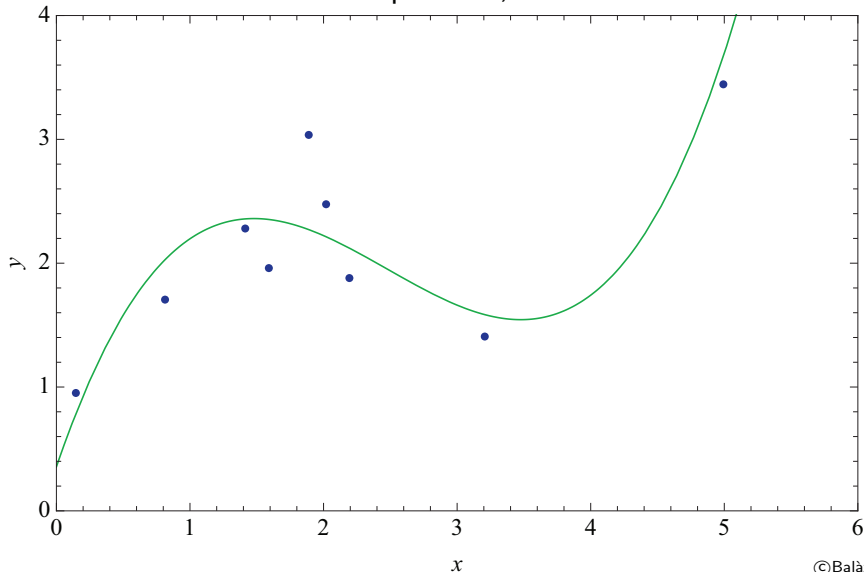
©Balázs Kégl

Quadratic least squares fit, RMSE = 0.579



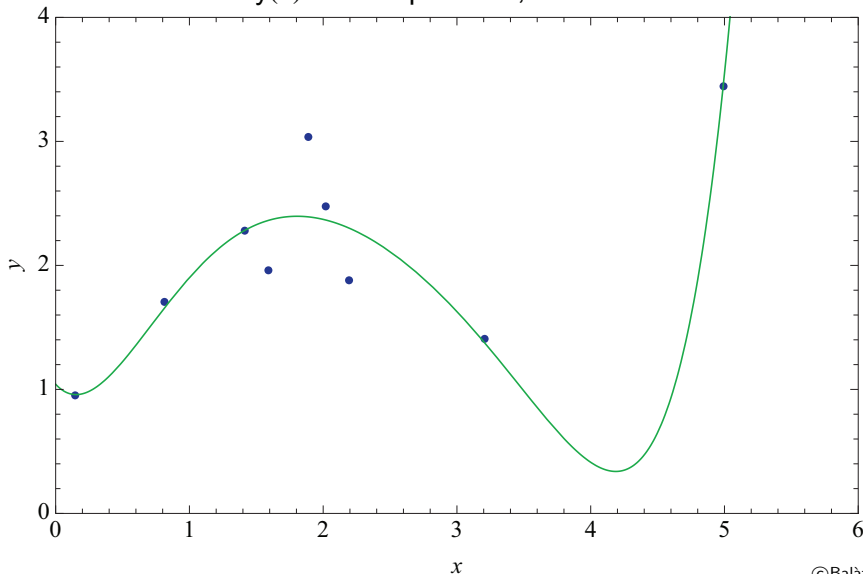
© Balázs Kégl

Cubic least squares fit, RMSE = 0.339



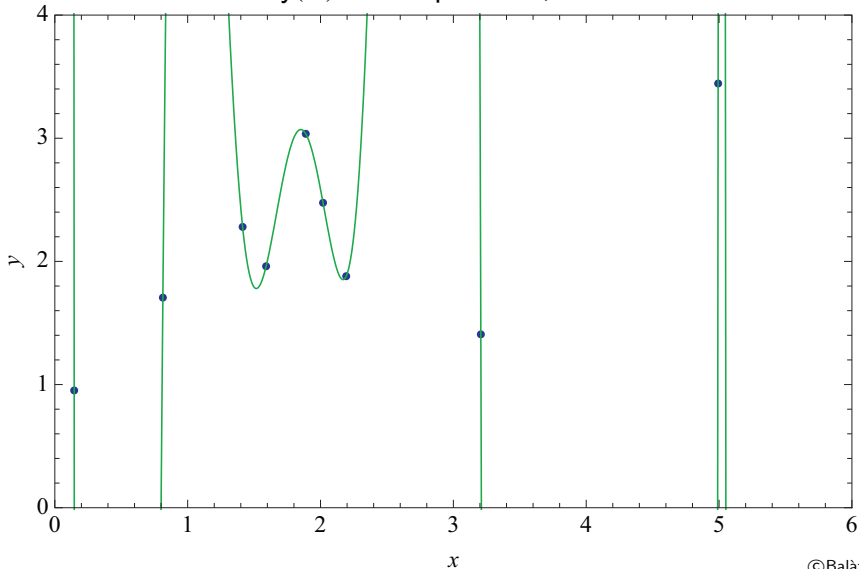
©Balázs Kégl

Poly(6) least squares fit, RMSE = 0.278



© Balázs Kégl

Poly(9) least squares fit, RMSE = 0



©Balázs Kégl

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree \Rightarrow can match to more features
- If degree = $\#$ points, polynomial passes through each point: perfect match!

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree \Rightarrow can match to more features
- If degree = $\#$ points, polynomial passes through each point: perfect match!

Is it meaningful?

- It could be:
 - if there is no noise or uncertainty in the measurement
 - if the true distribution is indeed perfectly described by such a polynomial
- ... not impossible, but not very common...

Quality of fit

- Increasing degree of polynomial increases flexibility of function
- Higher degree \Rightarrow can match to more features
- If degree = $\#$ points, polynomial passes through each point: perfect match!

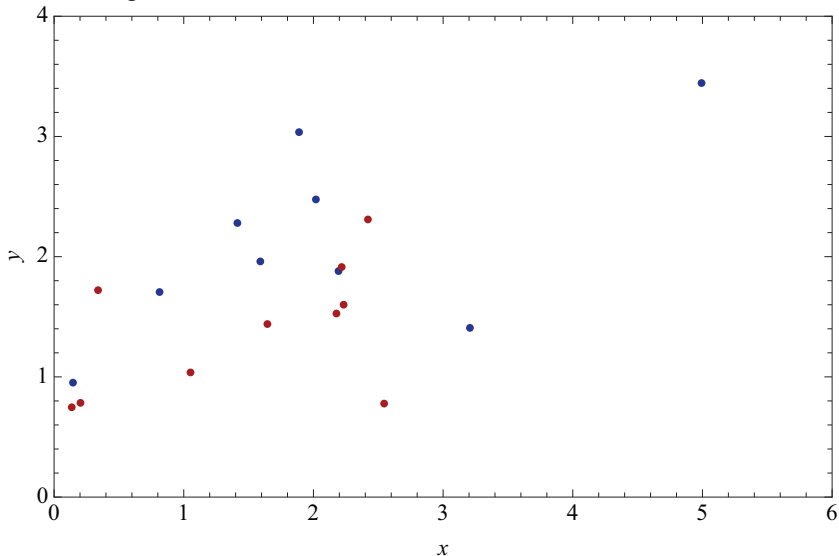
Is it meaningful?

- It could be:
 - if there is no noise or uncertainty in the measurement
 - if the true distribution is indeed perfectly described by such a polynomial
- ... not impossible, but not very common...

Solution: testing sample

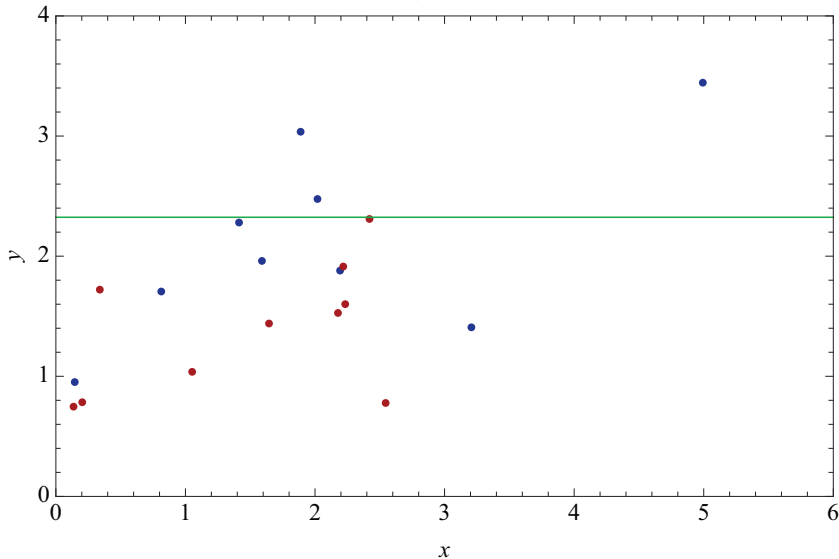
- Use independent sample to validate the result
- Expected: performance will also increase, go through a maximum and decrease again, while it keeps increasing on the training sample

Data generated from an unknown function with unknown noise



Choice of function class: testing

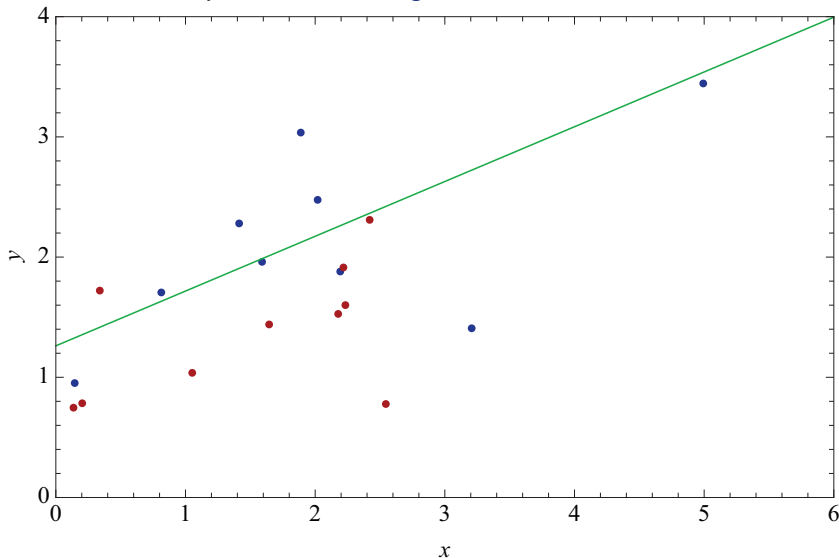
Const. least squares fit, **training** RMSE = 0.915, **test** RMSE = 1.067



©Balázs Kégl

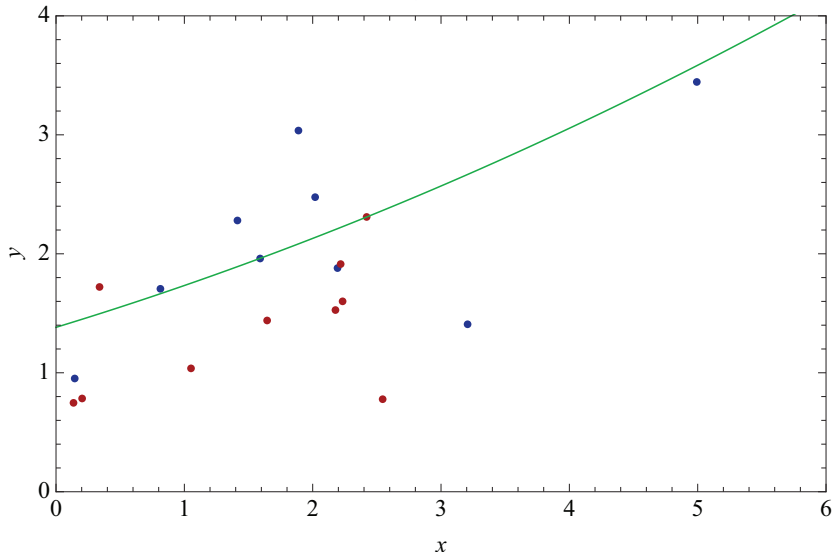
Choice of function class: testing

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



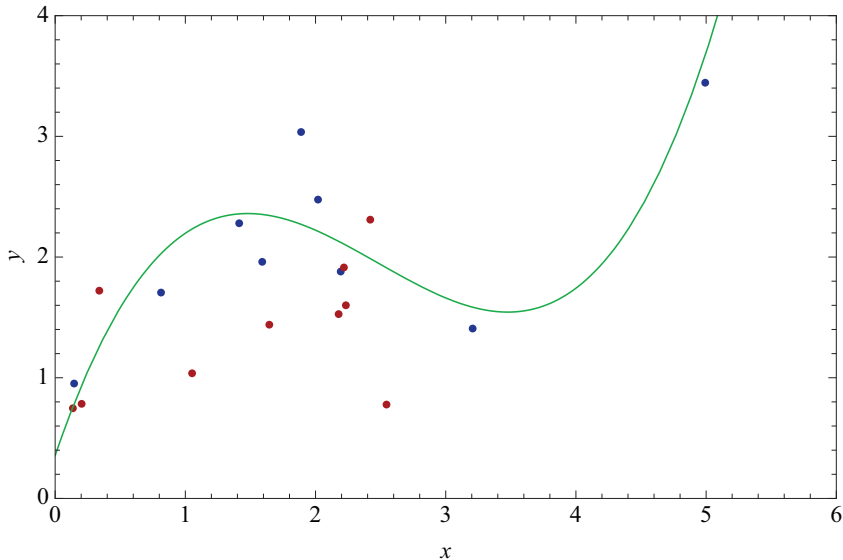
©Balázs Kégl

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

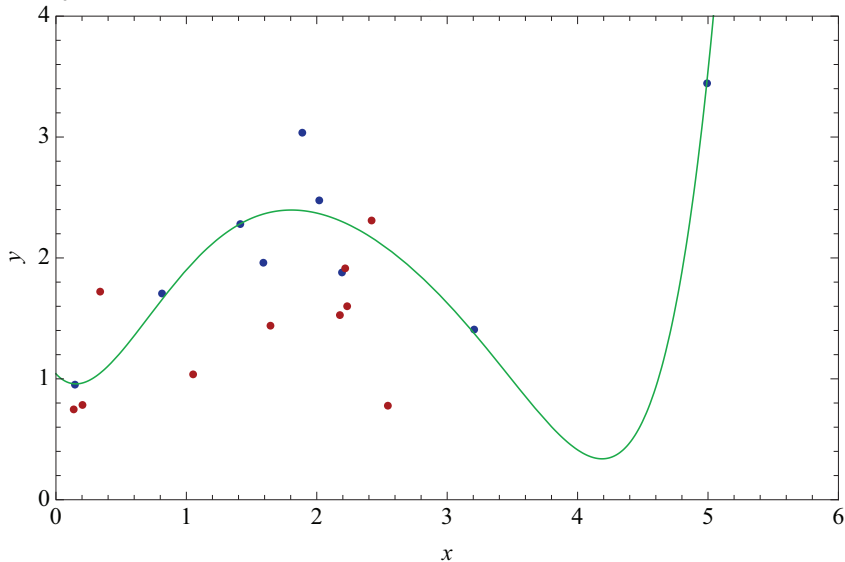


© Balázs Kégl

Cubic least squares fit, **training** RMSE = 0.339, **test** RMSE = 0.672

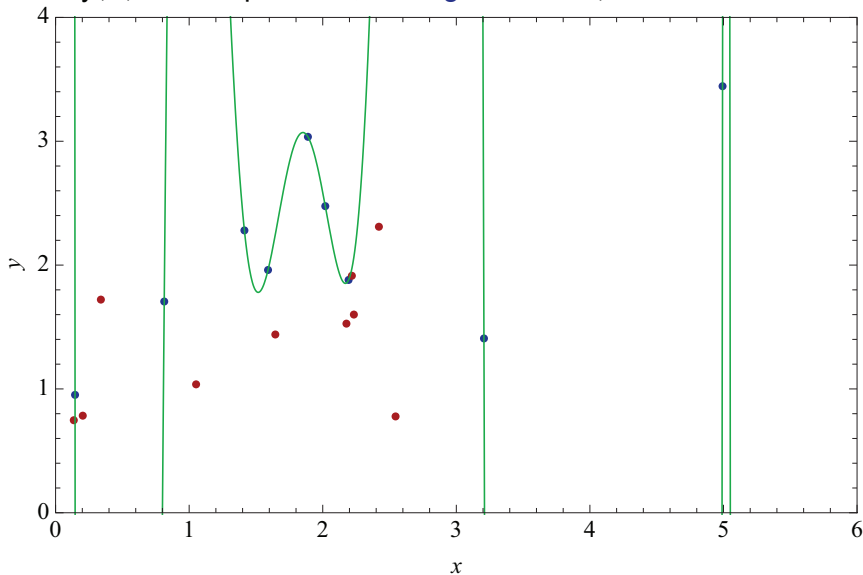


Poly(6) least squares fit, **training** RMSE = 0.278, **test** RMSE = 0.72

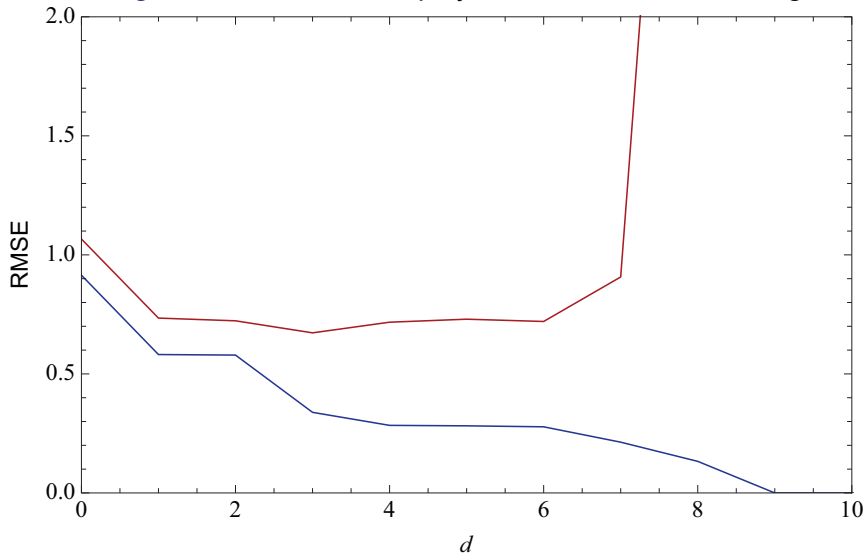


Choice of function class: testing

Poly(9) least squares fit, training RMSE = 0, test RMSE = 46.424



Training and test RMSE's for polynomial fits of different degrees



©Balázs Kégl

Non-parametric fit

- Minimising the training cost (here, RMSE) does not work if the function class is not fixed in advance (e.g. fix the polynomial degree): complete loss of generalisation capability!
- But if you do not know the correct function class, you should not fix it! Dilemma...

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

1 Introduction

2 Optimal discrimination

- Bayes limit
- Multivariate discriminant

3 Machine learning

- Supervised and unsupervised learning

4 Multivariate discriminants

- Random grid search
- Genetic algorithms
- Quadratic and linear discriminants
- Support vector machines
- Kernel density estimation
- Neural networks
- Bayesian neural networks
- Deep networks
- Decision trees

5 Summary

Reminder

- To solve binary classification problem with the fewest number of mistakes, sufficient to compute the multivariate discriminant:

$$D(x) = \frac{s(x)}{s(x) + b(x)}$$

where:

- $s(x) = p(x|S)$ signal density
- $b(x) = p(x|B)$ background density
- Cutting on $D(x)$ is equivalent to cutting on probability $p(S|x)$ that event with x values is of class S

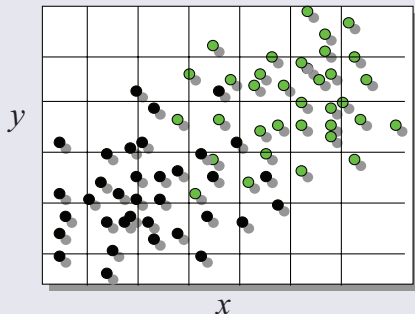
Which approximation to choose?

- Best possible choice: cannot beat Bayes limit (but usually impossible to define)
- No single method can be proven to surpass all others in particular case
- Advisable to try several and use the best one

Cut-based analysis

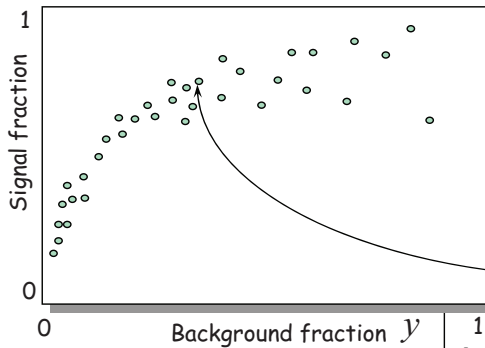
- Simple approach: cut on each discriminating variable
- Difficulty: how to optimise the cuts?

Grid search



©Harrison Prosper

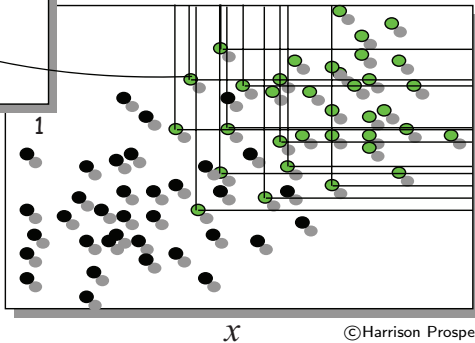
- Split each variable in K values
- Apply cuts at each grid point:
 $x > x_i, y > y_i$
- Number of points scales with K^n : **curse of dimensionality**



- Use each point in signal sample as grid point:

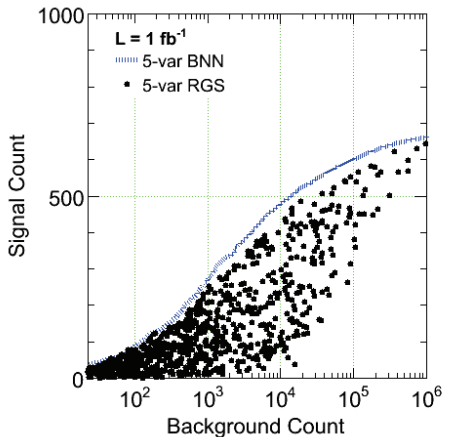
$$x > x_i$$

$$y > y_i$$



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

©Harrison Prosper



©Harrison Prosper

Comparison to BNN

- Blue: 5-dim Bayesian neural network discriminant (see later)
- Points: each cut point from a 5-dim RGS calculation
- Conclusions:
 - RGS can find very good criteria with high discrimination
 - but it usually cannot compete with a full-blown multivariate discriminant
 - and never outsmarts it

- 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

- Better solutions more likely to be selected for mating and mutations, carrying their genetic code (cuts) from generation to generation
- Algorithm:
 - ➊ Create initial random population (cut ensemble)
 - ➋ Select fittest individuals
 - ➌ Create offsprings through crossover (mix best cuts)
 - ➍ Mutate randomly (change some cuts of some individuals)
 - ➎ 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

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

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

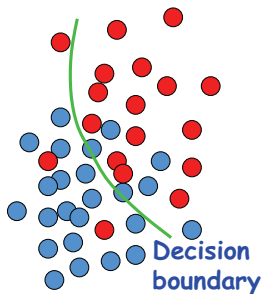
with vector of means μ and covariance matrix Σ

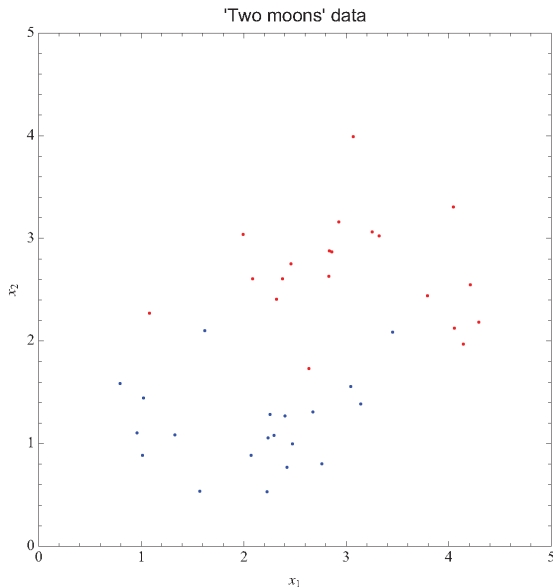
- Then Bayes factor $B(x) = s(x)/b(x)$ (or its logarithm) can be expressed explicitly:

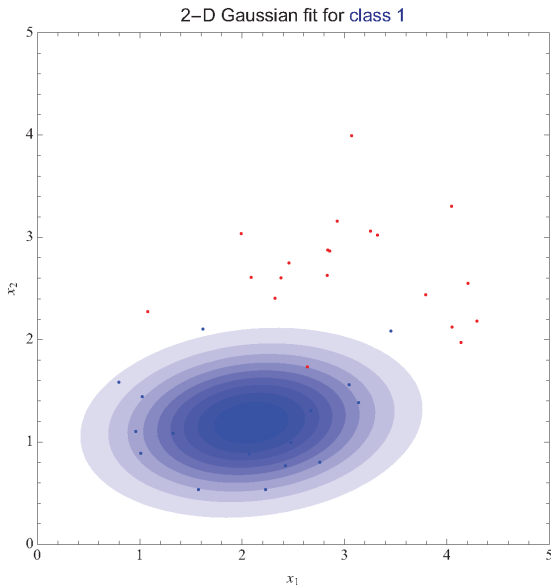
$$\ln B(x) = \lambda(x) \equiv \chi^2(\mu_B, \Sigma_B) - \chi^2(\mu_S, \Sigma_S)$$

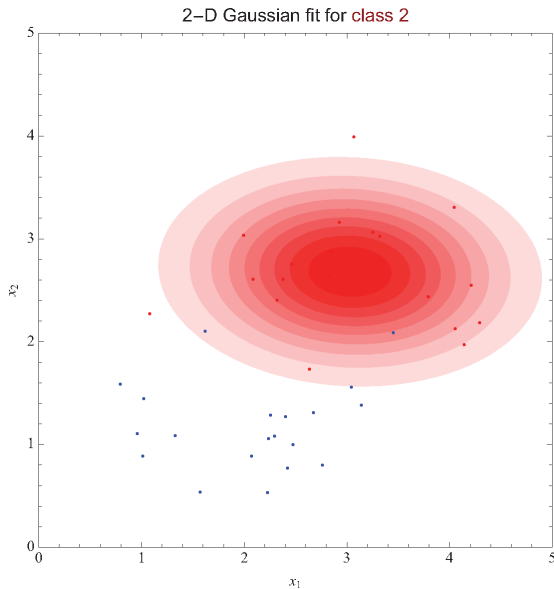
with $\chi^2(\mu, \Sigma) = (x - \mu)^T \Sigma^{-1}(x - \mu)$

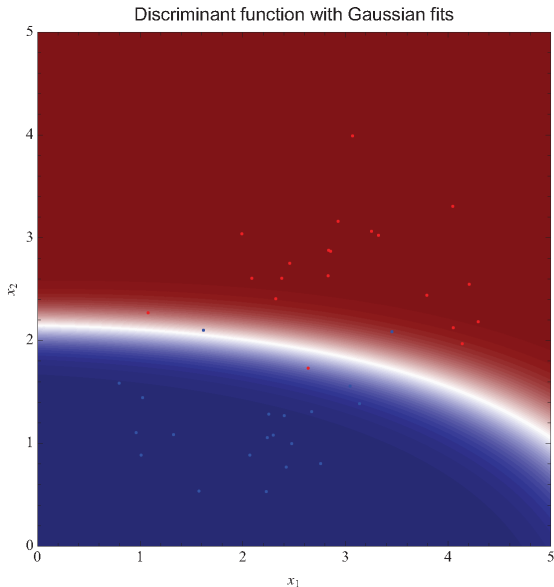
- Fixed value of $\lambda(x)$ defines a quadratic hypersurface partitioning the n -dimensional space into signal-rich and background-rich regions
- Optimal separation if $s(x)$ and $b(x)$ are indeed multivariate Gaussians







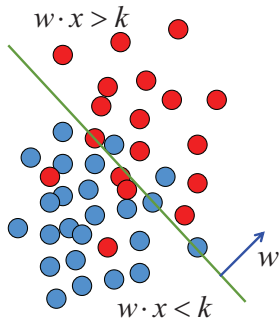




© Balázs Kégl

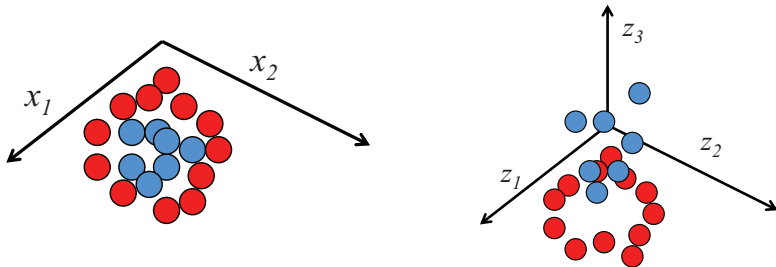
- If in $\lambda(x)$ the same covariance matrix is used for each class (e.g. $\Sigma = \Sigma_S + \Sigma_B$) one gets **Fisher's discriminant**:

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

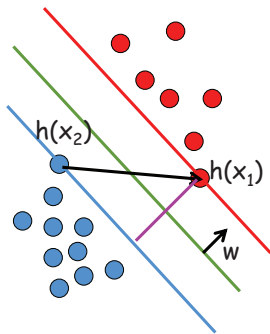


- Optimal linear separation
- Works only if signal and background have different means!
- Optimal classifier (reaches the Bayes limit) for linearly correlated Gaussian-distributed variables

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



- Consider separable data in \mathbb{R}^H , and three parallel hyper-planes:
 $w \cdot h(x) + b = 0$ (separating hyper-plane between red and blue)
 $w \cdot h(x_1) + b = +1$ (contains $h(x_1)$)
 $w \cdot h(x_2) + b = -1$ (contains $h(x_2)$)



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

- 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:
 $w \cdot h(x) + b > 1$ for red dots
 $w \cdot h(x) + b < -1$ for blue dots

all correctly classified points will satisfy constraints:

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

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

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

Minimisation

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

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

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

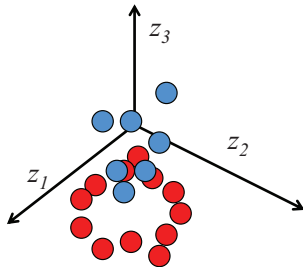
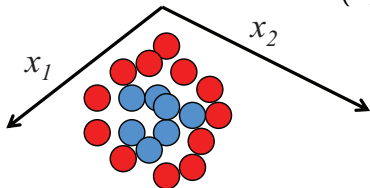
Kernel functions

- 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:

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

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

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

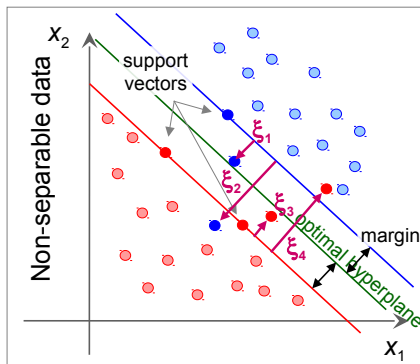


- In reality: do not know a priori the right kernel
- \Rightarrow have to test different standard kernels and use the best one

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

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

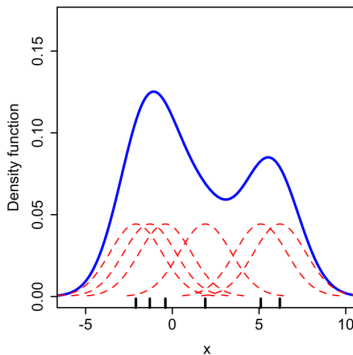
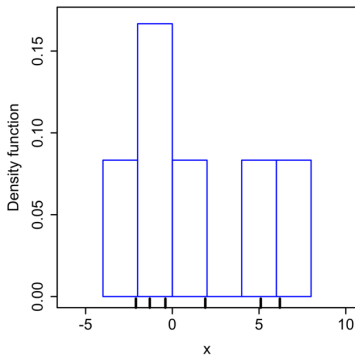
with **slack variables** $\xi_i > 0$



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

- 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:

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



Choice of kernel

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

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

each with **bandwidth** (width) h_i

Optimal bandwidth

- Too narrow: noisy approximation
- Too wide: loose fine structure
- In principle found by minimising risk function

$$R(\hat{p}, p) = \int (\hat{p}(x) - p(x))^2 dx$$

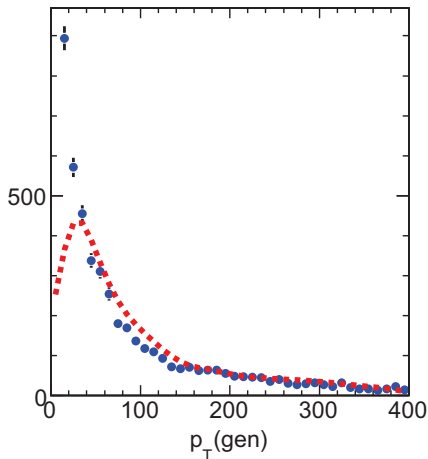
- For Gaussian densities:

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

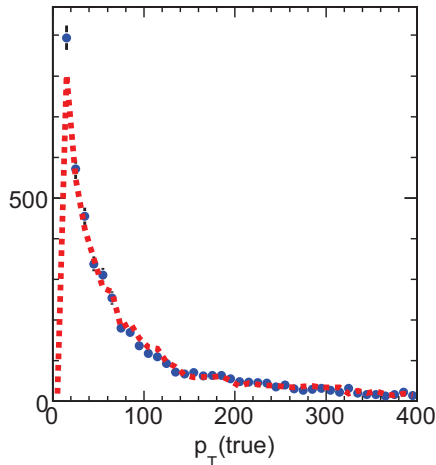
- Far from optimal for non-Gaussian densities

Kernel density estimation (KDE): example

with Gaussian optimal bandwidth



with optimised bandwidth



Why does it work?

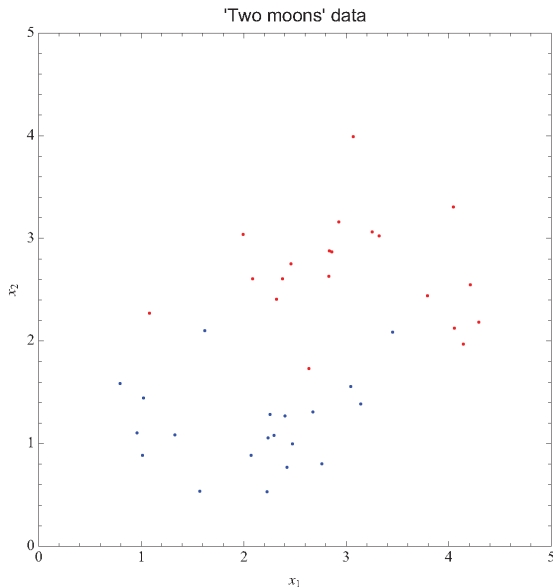
- When $N \rightarrow \infty$:

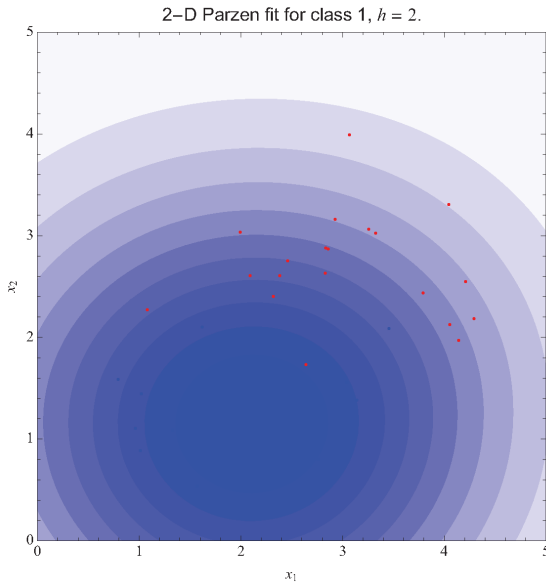
$$\hat{p}(x) = \int K(x, \mu) p(\mu) d\mu$$

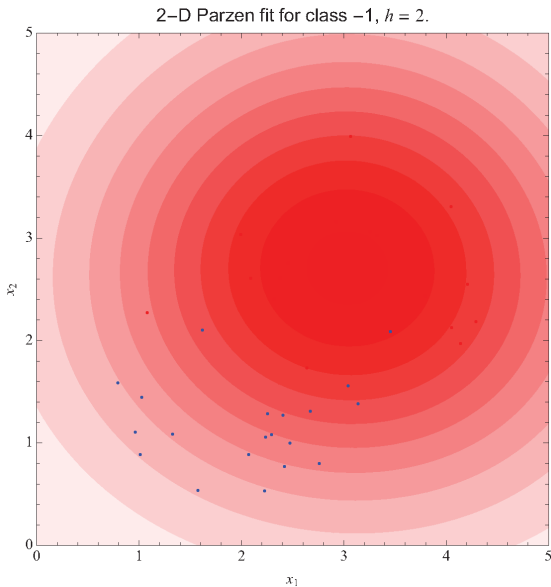
- $p(\mu)$: true density of x
- Kernel bandwidth getting smaller with N , so when $N \rightarrow \infty$, $K(x, \mu) \rightarrow \delta^n(x - \mu)$ and $\hat{p}(x) = p(x)$
- KDE gives consistent estimate of probability density $p(x)$

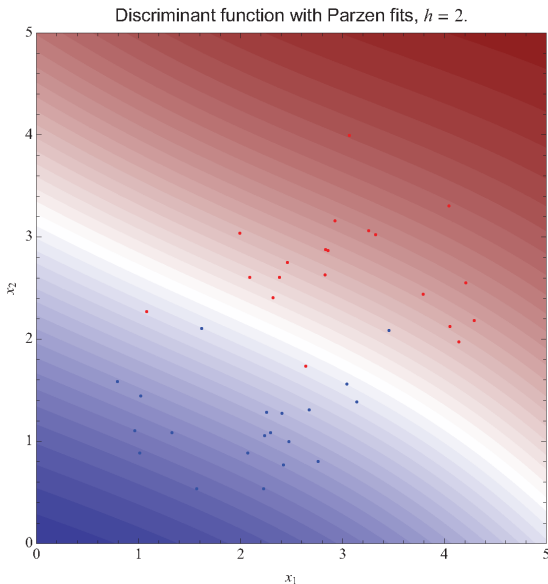
Limitations

- 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

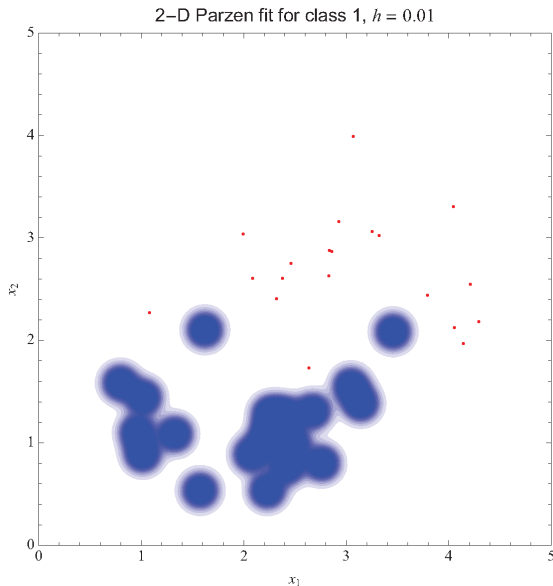


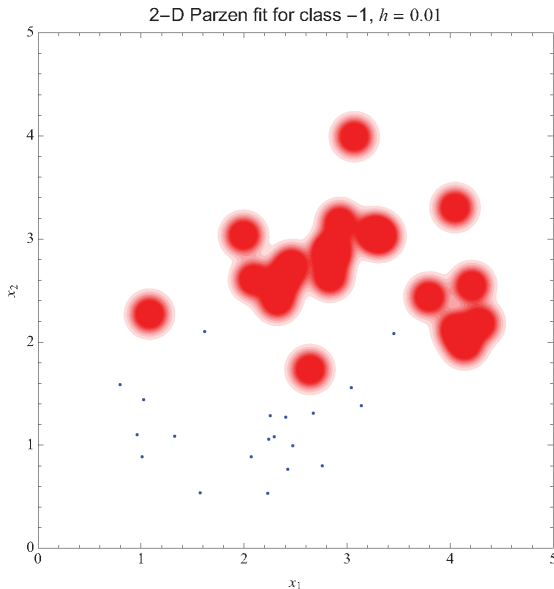


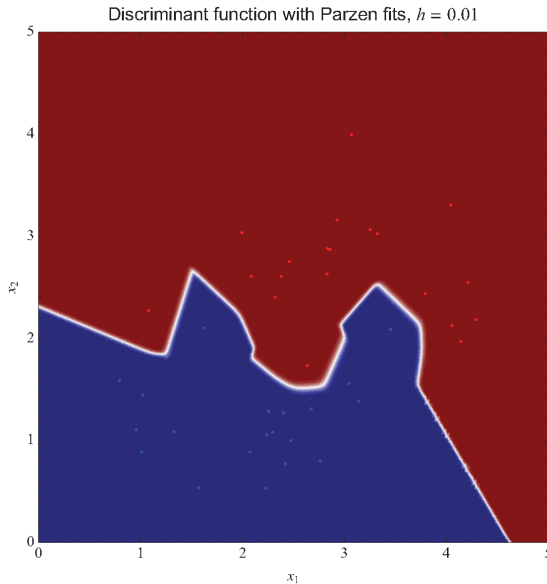


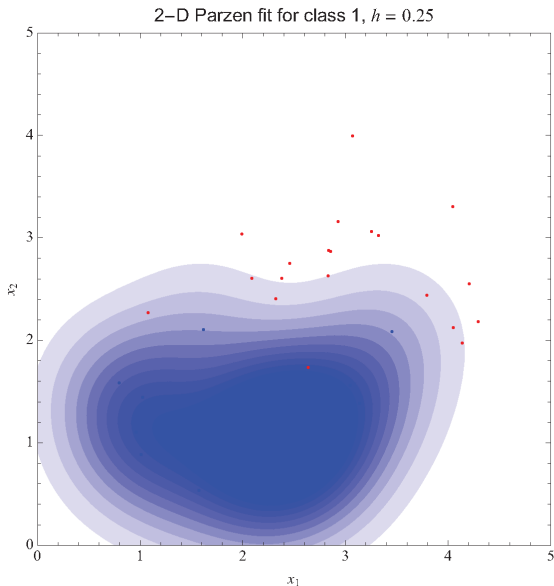


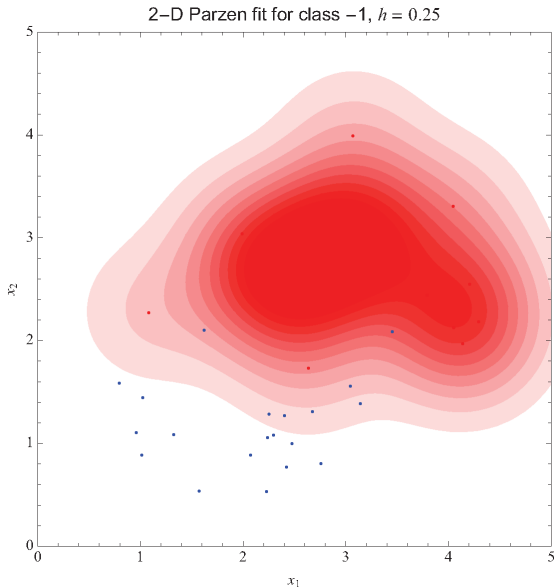
© Balàzs Kégl

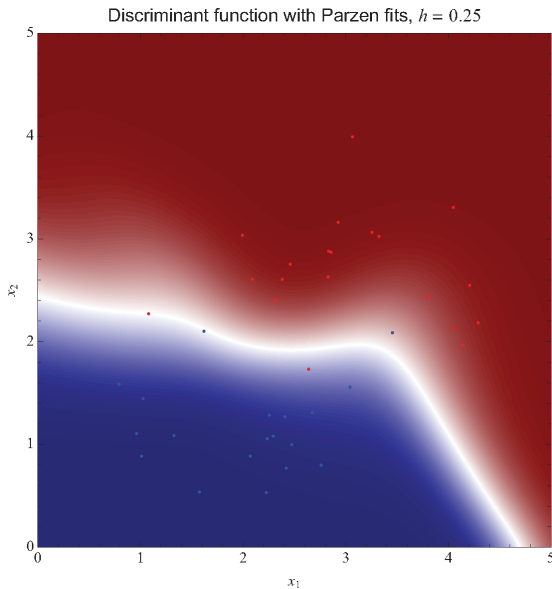






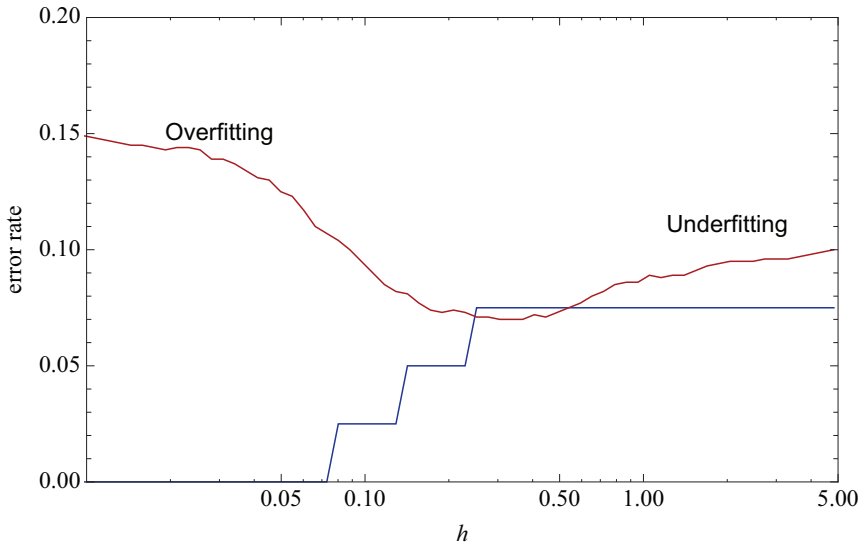






© Balàzs Kégl

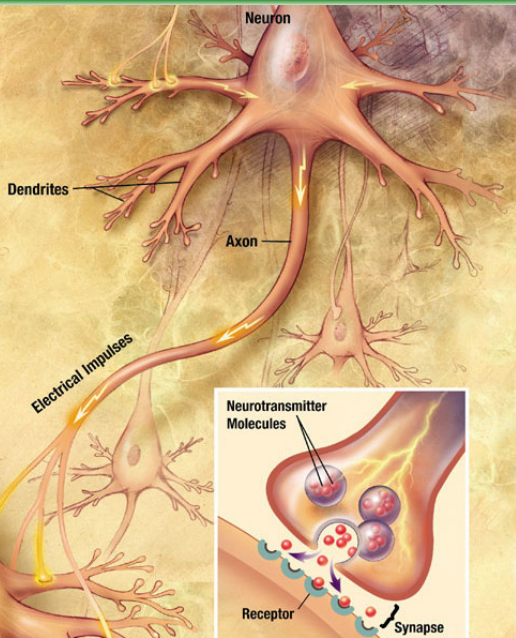
Training and test error rates



© Balázs Kégl

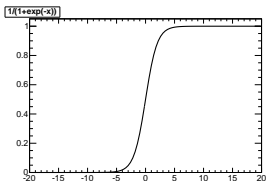
Human brain

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

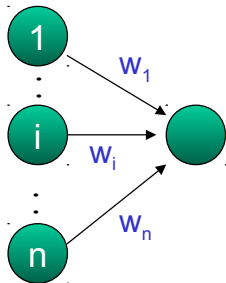


- 1943: W. McCulloch and W. Pitts explore capabilities of networks of simple neurons
- 1958: F. Rosenblatt introduces perceptron (single neuron with adjustable weights and threshold activation function)
- 1969: M. Minsky and S. Papert prove limitations of perceptron (linear separation only) and (wrongly)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

- 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):



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



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

Theorem

Let $\sigma(\cdot)$ 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 $i = 1, \dots, n$ and $j = 1, \dots, M$ 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 $f(\cdot)$, that is $|y(x) - f(x)| < \varepsilon$

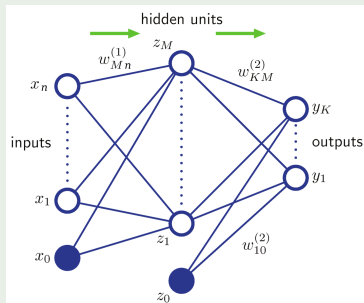
Interpretation

- 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

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



A neural network can fit any function: examples

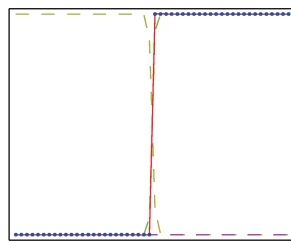
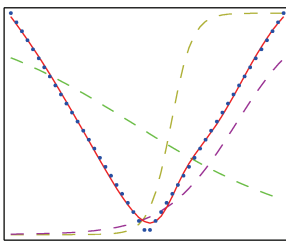
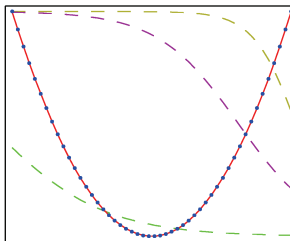
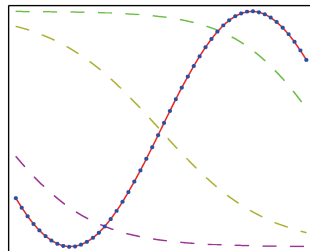
- 1 input (training data), 1 output
- 3 hidden neurons on one hidden layer

©Jan Therhaag

--- z_1
--- z_2
--- z_3

— *output*

... *training data*



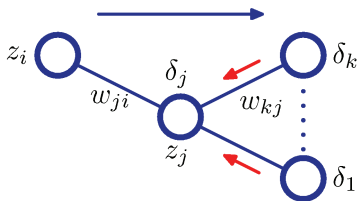
- Training means minimising error function $E(w)$
- For single neuron: $\frac{dE}{dw_k} = (y - t)x_k$
- One can show that for a network:

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

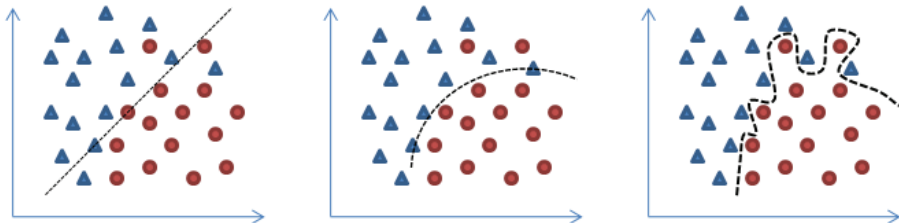
$$\delta_k = (y_k - t_k) \text{ for output neurons}$$

$$\delta_j \propto \sum_k w_{kj} \delta_k \text{ otherwise}$$

- Hence errors are propagated backwards



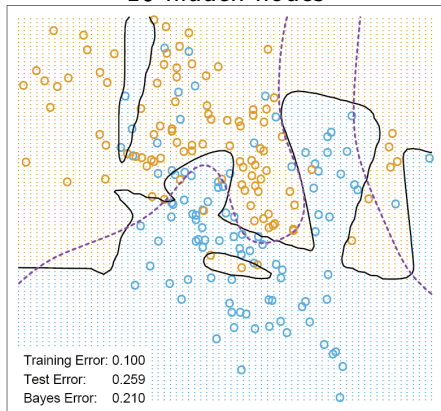
- Minimise error function $E(w)$
- Gradient descent: $w^{(k+1)} = w^{(k)} - \eta \frac{dE^{(k)}}{dw}$
- $\frac{\partial E}{\partial w_j} = \sum_{n=1}^N -(t^{(n)} - y^{(n)})x_j^{(n)}$ with target $t^{(n)}$ (0 or 1), so $t^{(n)} - y^{(n)}$ is the error on event n
- All events at once (batch learning):
 - weights updated all at once after processing the entire training sample
 - finds the actual steepest descent
 - takes more time
- or one-by-one (online learning):
 - speeds up learning
 - useful in HEP because of redundant datasets (large Monte Carlo samples with many similar events)
 - may avoid local minima with stochastic component in minimisation
 - careful: depends on the order of training events
- One epoch: going through the training data once



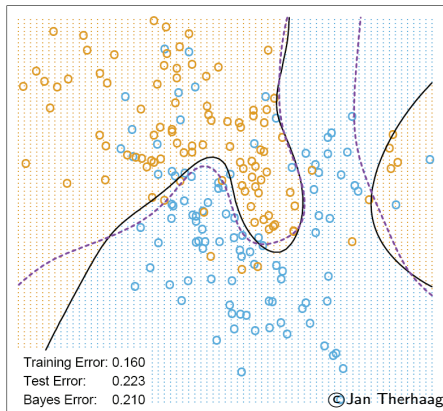
- Diverging weights can cause overfitting
- Mitigate by:
 - early stopping (after a fixed number of epochs)
 - monitoring error on test sample
 - regularisation, introducing a “weight decay” term to penalise large weights, preventing overfitting:

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

10 hidden nodes

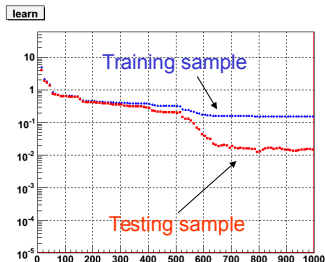


10 hidden nodes and $\alpha = 0.04$

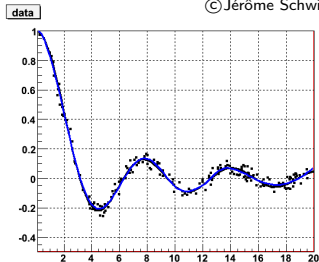


- Much less overfitting, better generalisation properties

- Train on noisy data centred on true value
- Test on no-noise data
- Testing error becomes better: during training, the NN learned the true distribution (average of noisy inputs)
- \Rightarrow testing converges
- Example:
 $\sin(x)/x + \text{rand}(-0.05, +0.05)$
- Of course doesn't work as well if noise is not symmetric

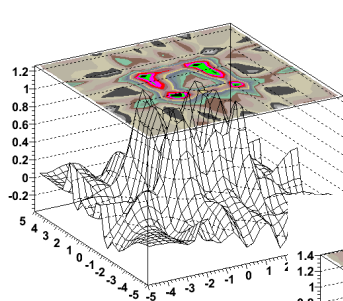


© Jérôme Schwindling

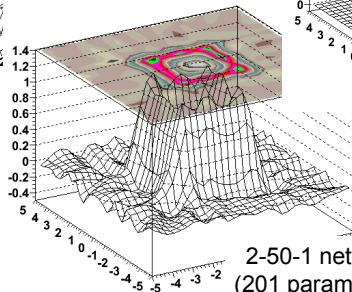
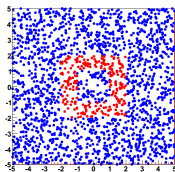


- Preprocess data:
 - if relevant, provide e.g. x/y instead of x and y
 - subtract the mean because the sigmoid derivative becomes negligible very fast (so, input mean close to 0)
 - normalise variances (close to 1)
 - shuffle training sample (order matters in online training)
- Initial random weights should be small to avoid saturation
- Batch/online training: depends on the problem
- Regularise weights to minimise overtraining. May also help select good variables via Automatic Relevance Determination (ARD)
- 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

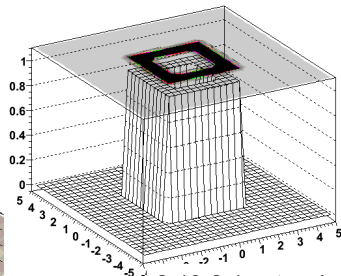
Adding a hidden layer



2-20-1 network
(81 parameters)



2-50-1 network
(201 parameters)



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

- As name says: Bayesian approach, try to *infer* functions $f(x)$
- Training sample T of N examples $(x, y)_1, (x, y)_2, \dots, (x, y)_N$ of discriminating variables x and class labels y
- Each point w corresponds to a function $f(x, w)$
- Assign probability density $p(w|T)$ to it
- If $p(w_1|T) > p(w_2|T)$, then associated function $f(x, w_1)$ more compatible with training data T than function $f(x, w_2)$
- Posterior density $p(w|T)$ is final result of Bayesian inference
- BNN is the predictive distribution

$$p(y|x, T) = \int p(y|x, w)p(w|T)dw$$

where the function class is class of feedforward neural networks with a fixed structure (inputs, layers, hidden nodes, outputs)

- Take the mean of the predictive distribution:

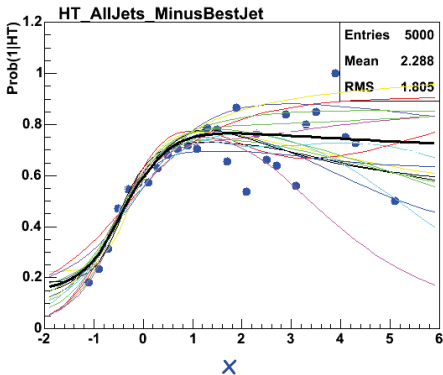
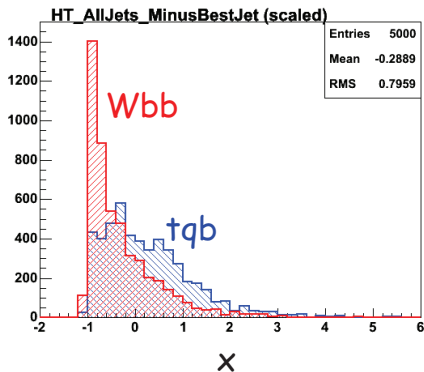
$$\begin{aligned}y(x) &= \int zp(z|x, T)dz \\ &= \int f(x, w)p(w|T)dw\end{aligned}$$

- Why? For classification $p(y|x, w) = f(x, w)^y(1 - f(x, w))^{1-y}$
 - for $y = 1$: $p(y|x, w) = f(x, w)$
 - for $y = 0$: $p(y|x, w) = 1 - f(x, w)$
 - so only $f(x, w)$ contributes to the mean

- Example usage:

$$\begin{aligned}f(x, w) &= \frac{1}{1 + e^{-g(x, w)}} \\ g(x, w) &= b + \sum_{j=1}^H v_j \tanh \left(a_j + \sum_{i=1}^n u_{ij}x_i \right) \\ &\text{with } H \text{ hidden nodes}\end{aligned}$$

- Scanning NN parameter space can be daunting
- Can approximate integral in $y(x)$ using Markov chain Monte Carlo method (MCMC)
- Will generate M sample weights w_1, \dots, w_M from posterior density $p(w|T)$
- $y(x) \approx \frac{1}{M} \sum_{m=1}^M f(x, w_m)$
- Use sparse subset of MCMC points to avoid correlations
- Start with “reasonable” guesses for parameters (e.g. zero-centred Gaussians)



- points: bin by bin histogram ratio
- thin curves: each $f(x, w_k)$
- thick curve: average, which approximates $D(x)$

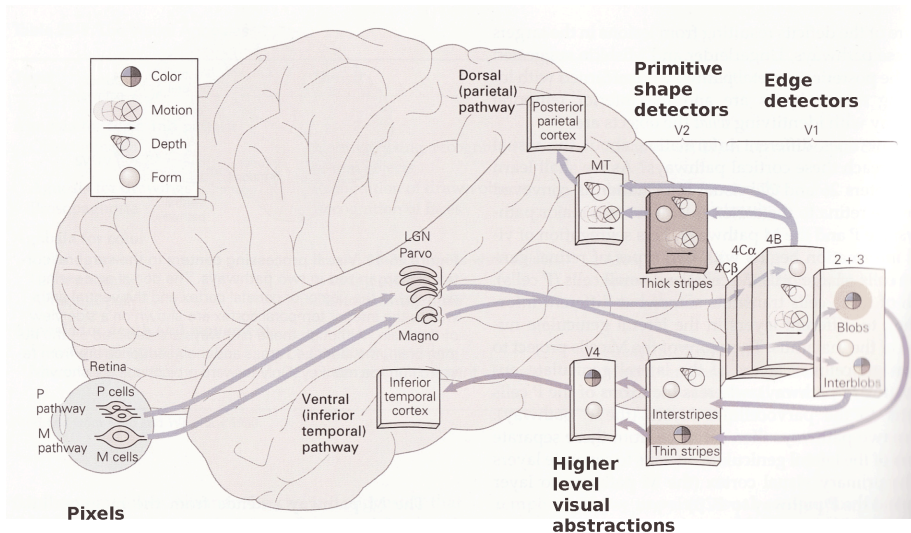
What is learning?

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

Motivation for deep learning

- Just like in the brain!
- Humans organise ideas hierarchically, through composition of simpler ideas
- Heavily unsupervised training, learning simpler tasks first, then combined into more abstract ones
- Learn first order features from raw inputs, then patterns in first order features, then etc.

Deep architecture in the brain



Mimicking the brain

- About 1% of neurons active simultaneously in the brain:
distributed representation
 - activation of small subset of features, not mutually exclusive
 - more efficient than local representation
 - distributed representations necessary to achieve non-local generalization, exponentially more efficient than 1-of- N enumeration
 - example: integers in $1..N$
 - local representation: vector of N bits with single 1 and $N-1$ zeros
 - distributed representation: vector of $\log_2 N$ bits (binary notation), exponentially more compact
- Meaning: information not localised in particular neuron but distributed across them

Deep architecture

- Insufficient depth can hurt
- Learn basic features first, then higher level ones
- Learn good intermediate representations, shared across tasks

Deep networks were unattractive

- One layer is 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
 - “diffusion of gradients”: gradients getting very small further away from output \Rightarrow early layers do not learn much, can even penalise overall performance

Deep networks were unattractive

- One layer is 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
 - “diffusion of gradients”: gradients getting very small further away from output \Rightarrow early layers do not learn much, can even penalise overall performance

Breakthrough in 2006 (Bengio, Hinton, LeCun)

- Try to model structure of input, $p(x)$ instead of $p(y|x)$
- Can use unlabelled data (a lot of it), with unsupervised training
- Most importantly: train each layer independently

Algorithm

- Take input information
- Train feature extractor
- Use output as input to training another feature extractor
- Keep adding layers, train each layer separately
- Finalise with a supervised classifier, taking last feature extractor output as input
- All steps above: pre-training
- Fine-tune the whole thing with supervised training (backpropagation)
 - initial weights are those from pre-training

Feature extractors

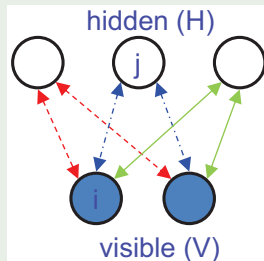
- Restricted Boltzmann machine (RBM), auto-encoder, sparse auto-encoder, denoising auto-encoder, etc.
- Note: important to not use linear activation functions in hidden layers. Combination of linear functions still linear, so equivalent to single hidden layer

Energy minimisation

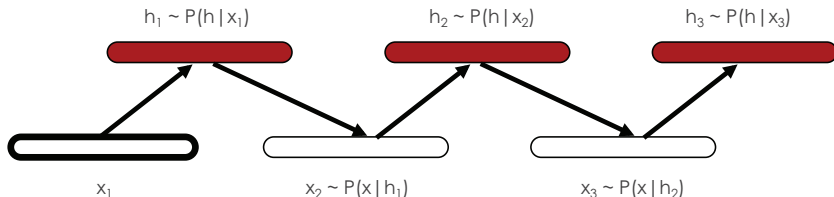
- RBM: visible and hidden variables, without hidden-hidden or visible-visible connections
- Energy of joint combination of visible and hidden units:

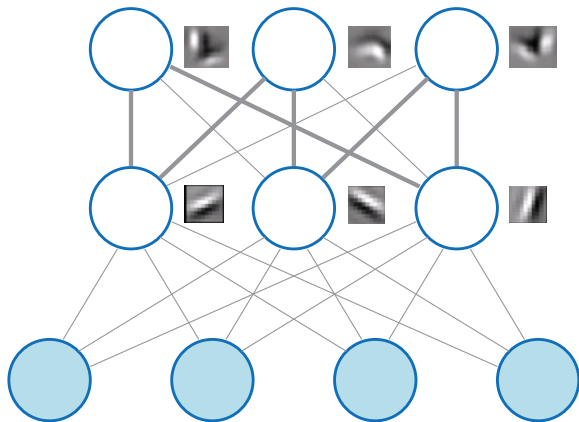
$$E(v, h) = -b^T v - c^T h - h^T W v$$

- b/c : biases of visible/hidden nodes
- W : weights connecting visible and hidden units
- Probability: $p(v, h) \propto e^{-E(v, h)}$



RBM training: Gibbs sampling





**Higher layer: DBNs
(Combinations
of edges)**

**First layer: RBMs
(edges)**

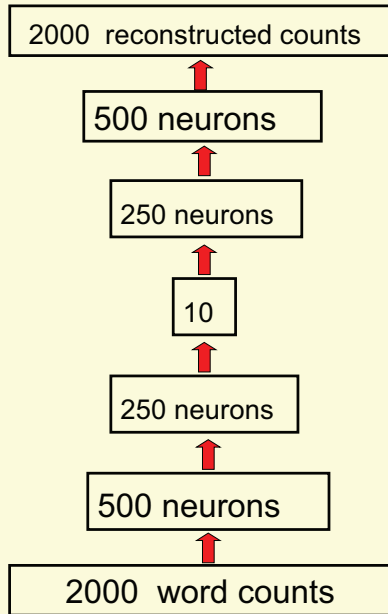
**Input image patch
(pixels)**

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

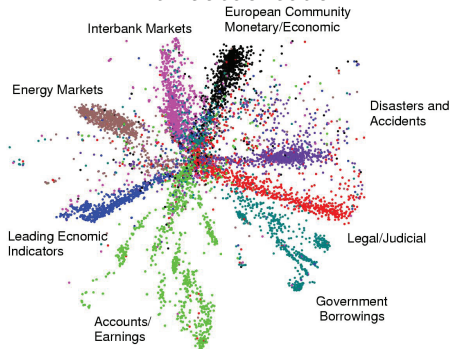
- Get count of 2000 most common words per book
- “Compress” to 10 numbers



With PCA



With autoencoder

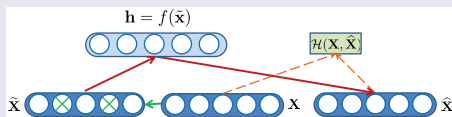


Sparse auto-encoder

- 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

Denosing auto-encoder

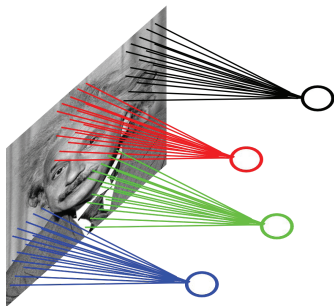
- Stochastically corrupt inputs
- Train to reconstruct uncorrupted input



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

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



- 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

1 $\times 1$	1 $\times 0$	1 $\times 1$	0	0
0 $\times 0$	1 $\times 1$	1 $\times 0$	1	0
0 $\times 1$	0 $\times 0$	1 $\times 1$	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

- 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

1	1 _{x1}	1 _{x0}	0 _{x1}	0
0	1 _{x0}	1 _{x1}	1 _{x0}	0
0	0 _{x1}	1 _{x0}	1 _{x1}	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	

Convolved
Feature

- 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

1	1	1 _{x1}	0 _{x0}	0 _{x1}
0	1	1 _{x0}	1 _{x1}	0 _{x0}
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Convolved
Feature

- 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

1	1	1	0	0
0	1 _{x1}	1 _{x0}	1	0
0	0 _{x0}	0 _{x1}	1 _{x0}	1
0	0 _{x1}	0 _{x0}	1 _{x1}	0
0	1	1	0	0

Image

4	3	4
2		

Convolved
Feature

- 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

1	1	1	0	0
0	1 _{x1}	1 _{x0}	1 _{x1}	0
0	0 _{x0}	1 _{x1}	1 _{x0}	1
0	0 _{x1}	1 _{x0}	1 _{x1}	0
0	1	1	0	0

Image

4	3	4
2	4	

Convolved
Feature

- 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

1	1	1	0	0
0	1	1 _{x1}	1 _{x0}	0 _{x1}
0	0	1 _{x0}	1 _{x1}	1 _{x0}
0	0	1 _{x1}	1 _{x0}	0 _{x1}
0	1	1	0	0

Image

4	3	4
2	4	3

Convolved
Feature

- 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

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	3
2		

Convolved
Feature

- 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

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4
2	4	3
2	3	

Convolved
Feature

- 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

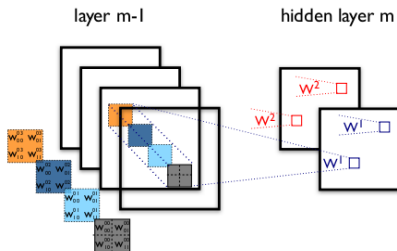
1	1	1	0	0
0	1	1	1	0
0	0	1 _{x1}	1 _{x0}	1 _{x1}
0	0	1 _{x0}	1 _{x1}	0 _{x0}
0	1	1 _{x1}	0 _{x0}	0 _{x1}

Image

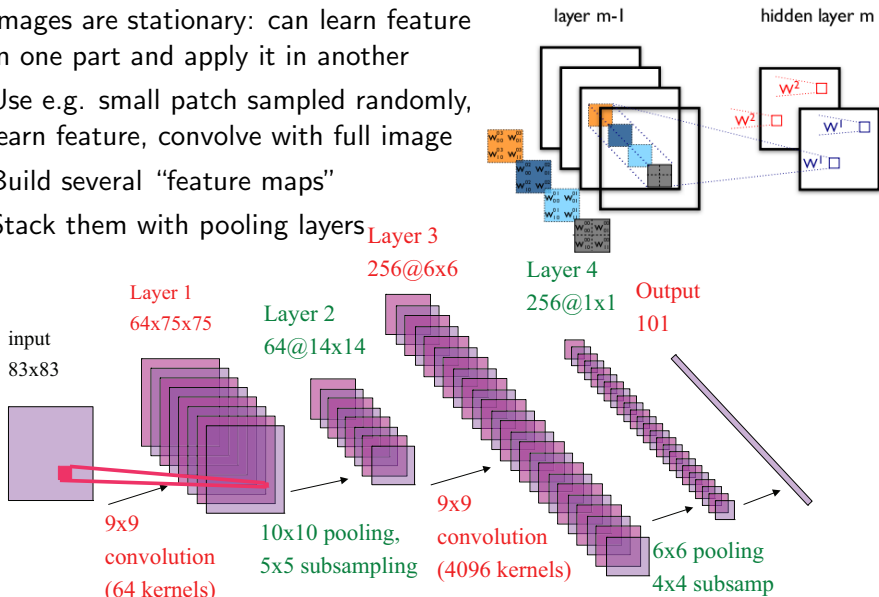
4	3	4
2	4	3
2	3	4

Convolved
Feature

- 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”



- 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



Why does unsupervised training work?

Optimisation hypothesis

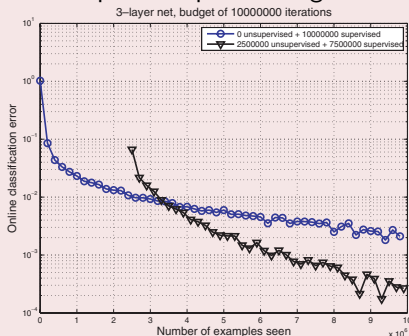
- Training one layer at a time scales well
- Backpropagation from sensible features
- Better local minimum than random initialisation, local search around it

Overfitting/regularisation hypothesis

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

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

- 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
- Influence learning dynamics can have big impact:
 - order and selection of examples matters
 - choose which examples to present first, to guide training and possibly increase learning speed (called shaping in animal training)
- Combination of deep learning and *reinforcement learning*
 - still in its infancy, but already impressive results
- *Adversarial training*
 - train in parallel network that produces difficult examples

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



bird



cat



dog

ILSVRC



flamingo



cock



ruffed grouse



quail



partridge

...



Egyptian cat



Persian cat



Siamese cat



tabby



lynx

...



dalmatian



keeshond



miniature schnauzer



giant schnauzer

...



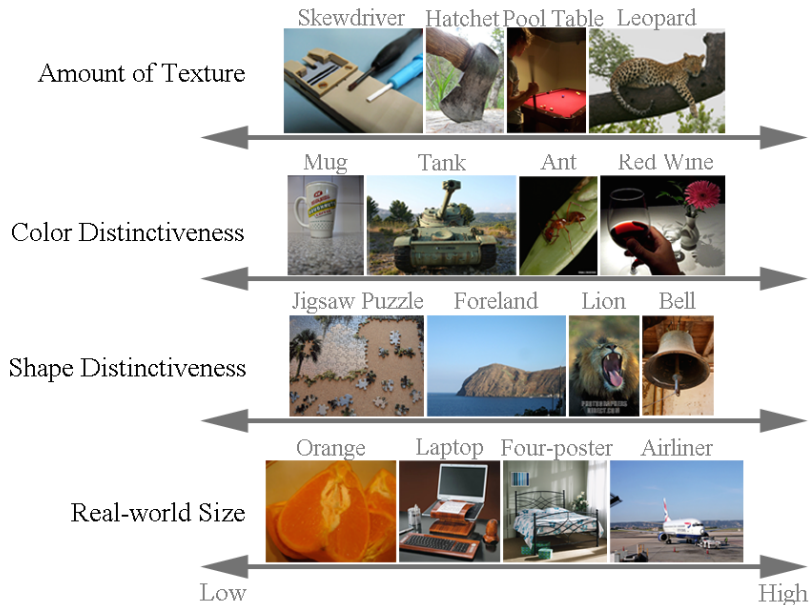


Image classification

Steel drum



Ground truth

Steel drum
Folding chair
Loudspeaker

Accuracy: 1

Scale
T-shirt
Steel drum
Drumstick
Mud turtle

Accuracy: 1

Scale
T-shirt
Giant panda
Drumstick
Mud turtle

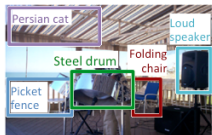
Accuracy: 0

Single-object localization

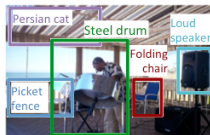
Steel drum



Ground truth



Accuracy: 1

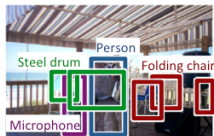


Accuracy: 0

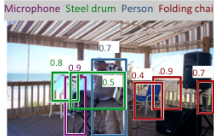


Accuracy: 0

Object detection



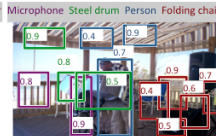
Ground truth



AP: 1.0 1.0 1.0 1.0



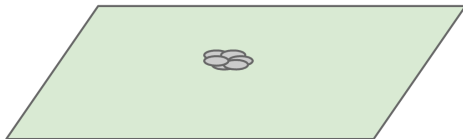
AP: 0.0 0.5 1.0 0.3



AP: 1.0 0.7 0.5 0.9

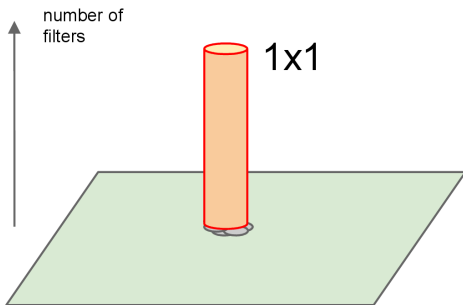
- Google of course! (first time)
- GoogLeNet:

In images, correlations tend to be local



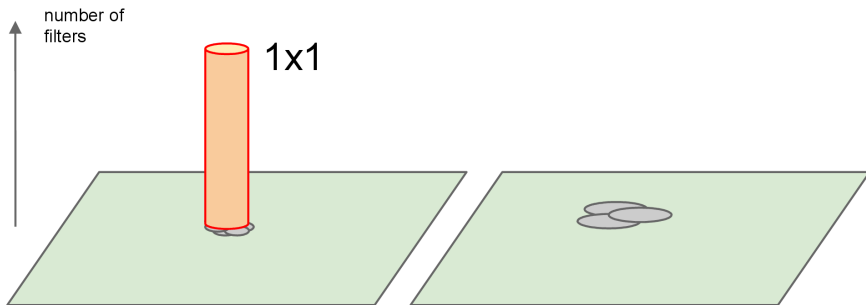
- Google of course! (first time)
- GoogLeNet:

Cover very local clusters by 1x1 convolutions



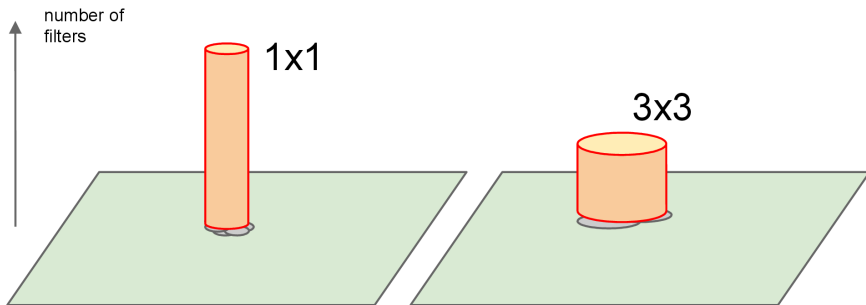
- Google of course! (first time)
- GoogLeNet:

Less spread out correlations



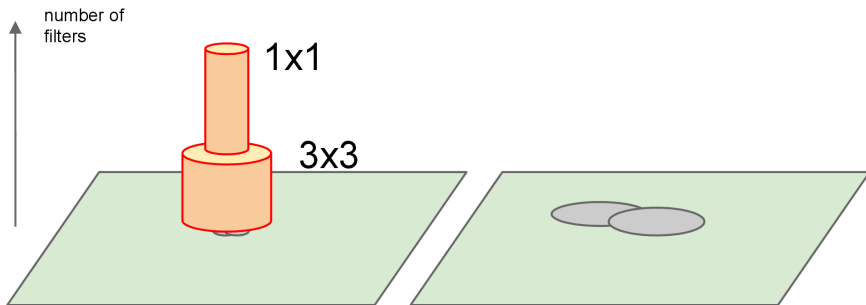
- Google of course! (first time)
- GoogLeNet:

Cover more spread out clusters by 3x3 convolutions



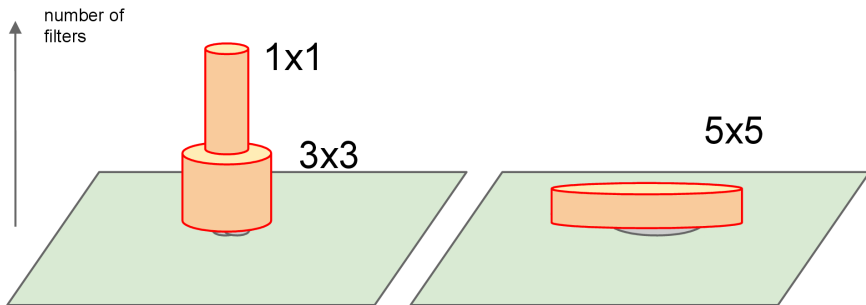
- Google of course! (first time)
- GoogLeNet:

Cover more spread out clusters by 5x5 convolutions



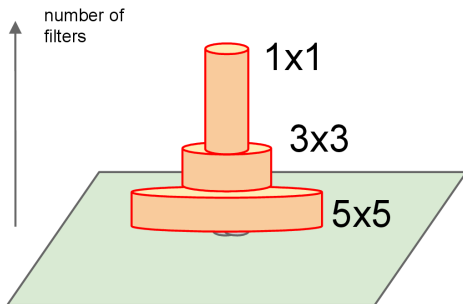
- Google of course! (first time)
- GoogLeNet:

Cover more spread out clusters by 5x5 convolutions



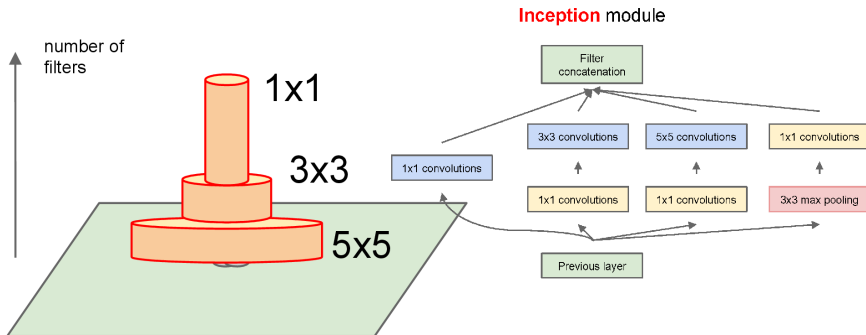
- Google of course! (first time)
- GoogLeNet:

A heterogeneous set of convolutions



- Google of course! (first time)
- GoogLeNet:

Schematic view



- Google of course! (first time)
- GoogLeNet:



9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other

Classification failure cases



Groundtruth: **Police car**

GoogLeNet:

- laptop
- hair drier
- binocular
- **ATM machine**
- seat belt

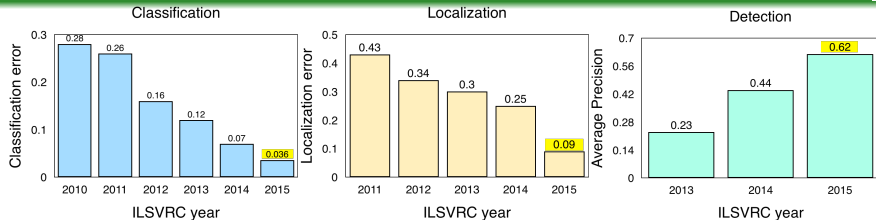
Classification failure cases



Groundtruth: **hay**

GoogLeNet:

- sorrel (horse)
- hartebeest
- Arabian camel
- warthog
- gazelle



2010–14: 4.2x reduction

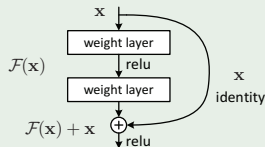
1.7x reduction

1.9x increase

ILSVRC 2015 (same dataset as 2014)

► arXiv:1512.03385

- Winner: MSRA (Microsoft Research in Beijing)
- Deep residual networks with > 150 layers
- Classification error: 6.7% \rightarrow 3.6% (1.9x)
- Localisation error: 26.7% \rightarrow 9.0% (2.8x)
- Object detection: 43.9% \rightarrow 62.1% (1.4x)

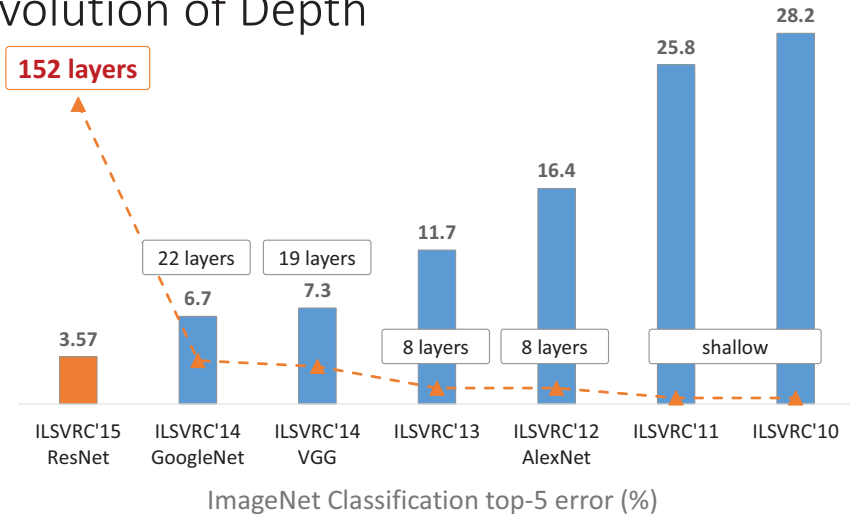


ILSVRC 2016

► <http://image-net.org/challenges/LSVRC/2016>

- Mostly ResNets. Classification: 0.030; localisation: 0.08; detection: 0.66

Revolution of Depth



- 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 $Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = s, a_t = a, \pi]$
 - 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 $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

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

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

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

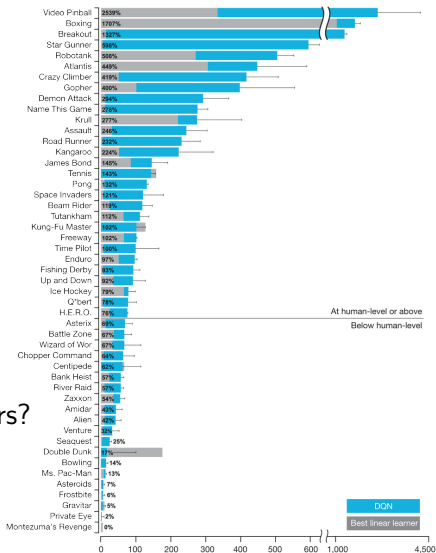
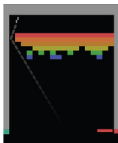
Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

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

End For

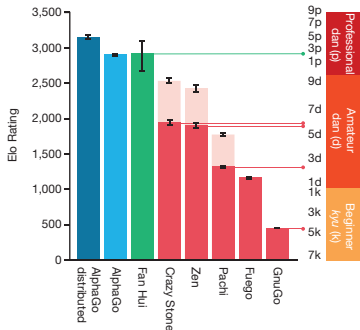
End For

• What about Breakout or Space invaders?



- Game of Go considered very challenging for AI
- 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.

- 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 beat human world champion) ⇒ better position selection (policy network) and better evaluation (value network)
- Then played Lee Sedol (top Go player 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’ ”

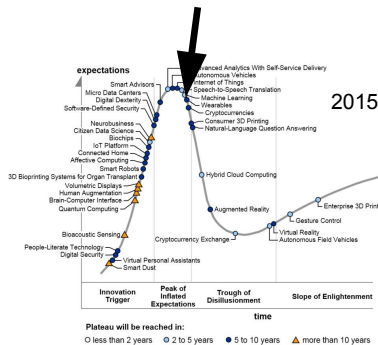


- Playing poker
 - Libratus (AI developed by Carnegie Mellon University) defeated four of the world's best professional poker players last month (Jan 2017)
 - After 120,000 hands of Heads-up, No-Limit Texas Hold'em, led the pros by a collective \$1,766,250 in chips
 - Learnt to bluff, and win with incomplete information and opponents' misinformation
 - Details next Thursday at [Association for the Advancement of Artificial Intelligence](#) meeting in San Francisco
- Lip reading [arXiv:1611.05358 \[cs.CV\]](#)
 - human professional: deciphers less than 25% of spoken words
 - CNN+LSTM trained on television news programs: 50%
- Limitations [arXiv:1312.6199 \[cs.CV\]](#)



- left: correctly classified image
- middle: difference between left image and adversarial image ($\times 10$)
- right: adversarial image, classified as ostrich

Hype cycle



Source: Gartner (July 2016)

Data Science @ LHC 2015

Bridging High-Energy Physics and Machine Learning communities

9 - 13 November 2015, CERN

Local Organising Committee

- Xavier Cid (CERN)
- Gilles Louppe (CERN)
- Michelangelo Mangano (CERN)
- Maurizio Pierini (CERN)
- Jean-Roch Vlimant (CERN)

Program Committee

- Kyle Cranmer (New York U)
- Céline Guerin (CERN)
- Vladimir Vazirani (CERN)
- Gilles Louppe (CERN)
- Andrew Lowe (Wigner RCP)
- Maurizio Pierini (CERN)
- David Rousseau (LAL Orsay)
- Maria Spiropulu (CERN)
- Jean-Roch Vlimant (CERN)
- Daniel Whiteson (UC Irvine)

International Advisory Committee

- Roger Barlow (Middlesbrough U)
- Tommaso Dorigo (INFN Padova)
- Ian Fisk (Simons Foundation)
- Maria Gionis (CERN)
- Eilam Gross (Weizmann)
- Balázs Kégl (LAL Orsay)
- Constantin Lelouch (BNL)
- Stuart Russell (UC Berkeley)
- Victoria Stodden (UI Urbana-Champaign)
- Max Welling (Amsterdam U)

sponsored by

LHC Physics Center at CERN: <http://lpc.web.cern.ch>

Fermilab National Laboratory: <http://fnal.gov>

Moore-Sloan Data Science Environment: <http://cds.nyu.edu/mooreloan>

<http://cern.ch/DataScienceLHC2015>

Data Science @ LHC 2015

Bridging High-Energy Physics and Machine Learning communities

Exploring the potential for Machine Learning on ATLAS

ATLAS Machine Learning Workshop

29th-31st March 2016, CERN

Organising Committee:

Matthew Beckingham (Warwick)
Michael Kagan (SLAC)
David Rousseau (LAL-Orsay)

<http://cern.ch/AtlasML2016>

Data Science @ LHC 2015

Bridging High-Energy Physics and Machine Learning communities

Exploring the potential for Machine Learning on ATLAS

ATLAS Machine Learning Workshop

MLHEP

20-26 June 2016
Lund, Sweden

Second Machine Learning School for High Energy Physics

<http://cern.ch/AtlasML2016>

Data Science @ LHC 2015

Bridging High-Energy Physics and Machine Learning communities

Exploring the

ATLAS Works


M

Second M

Higgs challenge the HiggsML challenge

May to September 2014

When High Energy Physics meets Machine Learning



info to participate and compete : <https://www.kaggle.com/c/higgs-boson>

Logos: ATLAS EXPERIMENT, CMS, LHCb, ALICE, ALCATRA, kaggle, Google

Organization committee: Fabrice Agot - Agot@AL, David Rousseau - Atlas@AL, Isabelle Guyon - Chokran

Advisory committee: Thorsten Waigler - Atlas@AL, Jeong Seobin - Atlas@AL

g on ATLAS

Learning

20-26 June 2016

Energy Physics

L2016

Data Science @ LHC 2015

Bridging High-Energy Physics and Machine Learning communities

Exploring the

ATLAS
Workshop

Second Meeting

Higgs
challenge



the HiggsML challenge

May to September 2014

When High Energy Physics meets Machine Learning

Workshop on ATLAS

Learning

NIPS 2016

Monday December 05 -- Saturday December 10, 2016

Centre Convencions Internacional Barcelona, Barcelona SPAIN

2016 Pricing »

Registration 2016 »

Dates

Calls

Student
Support

Program
Books

Schedule

Barcelona

View Earlier Meetings »

2015 Workshop Videos »

Invited Speakers

Yann LeCun (Facebook), Susan Holmes
(Stanford), Kyle Cranmer (NYU), Jaket
Navlakha (Salesforce), Drew Purves
(Deep Mind), Marc Raibert (Boston
Dynamics), Irina Rish (IBM)

Tutorials

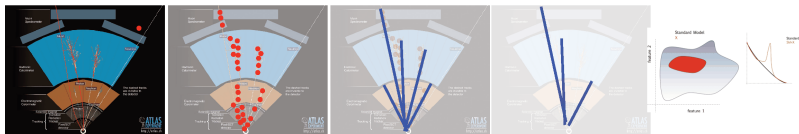
The tutorial times and rooms have not
been set yet. View the list of tutorials
using the button below.

View Tutorials »

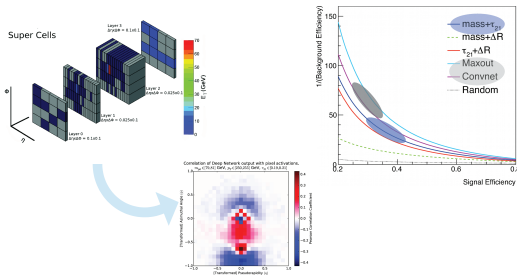
http://

- Going to lower level features [arXiv:1410.3469](#)

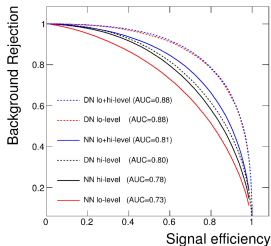
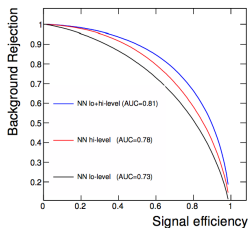
Raw	Sparsified	Reco	Select	Physics	Ana
1e7	1e4	100-ish*	50	10	1



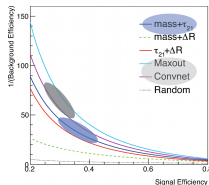
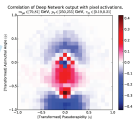
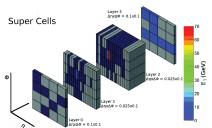
- Transforming inputs into images [arXiv:1511.05190](#)



- Going to lower level features ▶ arXiv:1410.3469



- Transforming inputs into images ▶ arXiv:1511.05190



Next lecture

Summary of MVA techniques

Criteria		Classifiers								
		Cuts	Likelihood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Performance	no / linear correlations	😊	😊	😊	😊	😊	😊	😊	😊	😊
	nonlinear correlations	😊	😞	😊	😞	😞	😊	😊	😊	😊
Speed	Training	😞	😊	😊	😊	😊	😊	😞	😊	😞
	Response	😊	😊	😞/😊	😊	😊	😊	😊	😊	😊
Robustness	Overtraining	😊	😊	😊	😊	😊	😞	😞	😊	😊
	Weak input variables	😊	😊	😞	😊	😊	😊	😊	😊	😊
Curse of dimensionality		😞	😊	😞	😊	😊	😊	😊	😊	😊
Transparency		😊	😊	😊	😊	😊	😞	😞	😞	😞

(according to TMVA authors)

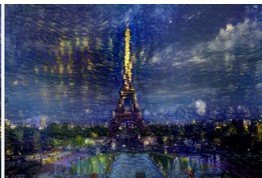
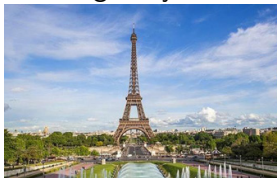
©Andreas Hoecker

- When trying to achieve optimal discrimination one can try to approximate

$$D(x) = \frac{s(x)}{s(x) + b(x)}$$






- Many techniques and tools exist to achieve this
- (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
- Multivariate techniques are at work in your everyday life without your knowing and can easily outsmart you for many tasks
- Try this to convince yourself http://www.phi-t.de/mousegame/index_eng.html








- Learning a style [▶ arXiv:1508.06576 \[cs.CV\]](#) [▶ Neural-style](#)



- Computer dreams [▶ Google original](#) [▶ deepdream](#)



-  V. Vapnik, *The Nature of Statistical Learning Theory*, Springer, New York, 2nd Edition, 2000
-  T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer-Verlag, New York, 2nd Edition, 2009
-  R.M. Neal, *Bayesian Learning of Neural Networks*, Springer-Verlag, New York, 1996
-  C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2007
-  M. Minsky and S. Papert, "Perceptrons", M.I.T. Press, Cambridge, Mass., 1969

-  H.B. Prosper, "The Random Grid Search: A Simple Way to Find Optimal Cuts", Computing in High Energy Physics (CHEP 95) conference, Rio de Janeiro, Brazil, 1995
-  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
-  P.C. Bhat and H.B. Prosper, "Bayesian neural networks", in *Statistical Problems in Particles, Astrophysics and Cosmology*, Imperial College Press, Editors L. Lyons and M. Ünel, 2005



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>



G.E. Hinton, S. Osindero and Y. Teh, “A fast learning algorithm for deep belief nets”, *Neural Computation* 18:1527-1554, 2006



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)
▶ <http://www.iro.umontreal.ca/lisa/publications2/index.php/publications/show/239>



A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. von Toerne, and H. Voss, “TMVA: Toolkit for Multivariate Data Analysis”, PoS A CAT 040 (2007) [physics/0703039] ▶ <http://tmva.sourceforge.net>