Multivariate analysis

Yann Coadou

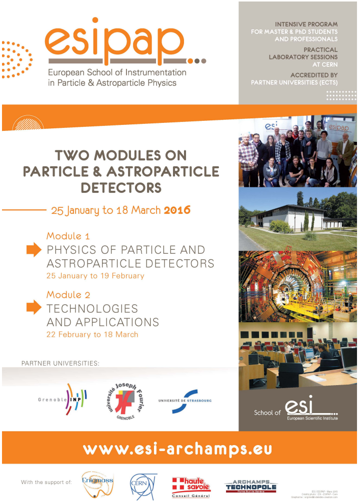
CPPM Marseille

esipap
European School of Instrumentation in Particle & Astroparticle Physics

Archamps, 8 February 2016

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

Outline



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**

Introduction

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, $\not{E}_{\mathrm{T}}$ 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, ...)

Introduction

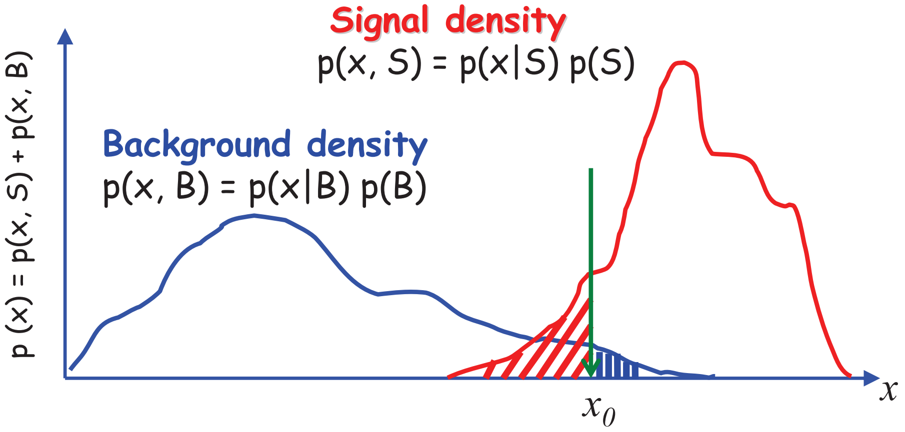
Input information from various sources

- 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 $x = \{x_1, \cdots, x_n\}$ 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 $f(x) \leq q$

- Where to place a cut x_0 on variable x?

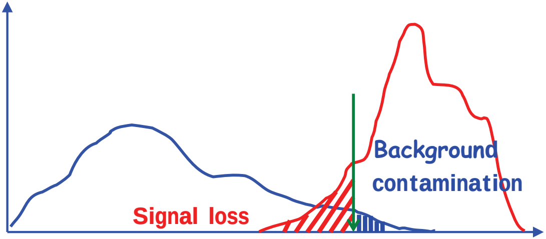


- Optimal choice: minimum misclassification cost at decision boundary $x = x_0$

Optimal discrimination: cost of misclassification

$$\begin{aligned} C(x_0) &= C_S \int H(x_0 - x) p(x,S) dx \qquad \text{signal loss} \\ &+ C_B \int H(x - x_0) p(x,B) dx \qquad \text{background contamination} \end{aligned}$$

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:

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

Probability relationships

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

Optimal discrimination: Bayes limit

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

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:

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

- 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

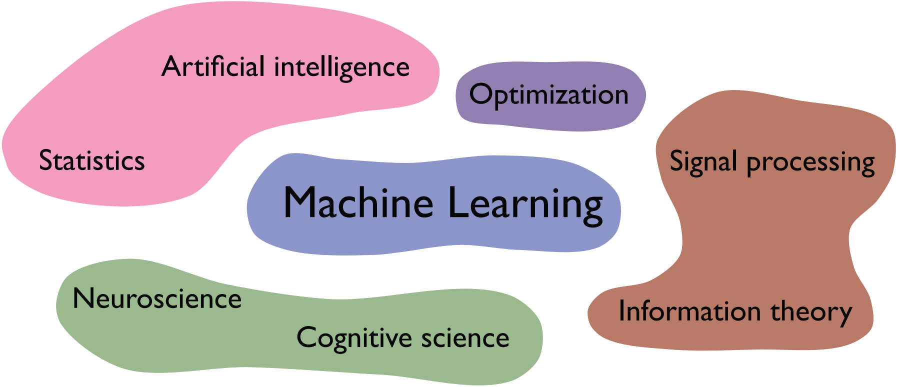
Several types of problems

- Classification/decision:
  - signal or background
  - type Ia supernova or not
  - will pay his/her credit back on time or not
- Regression (mostly ignored in these lectures)
- Clustering (cluster analysis):
  - in exploratory data mining, finding features

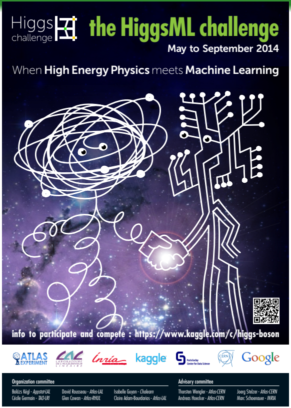
Our goal

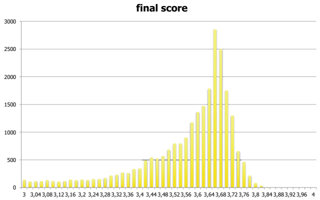
- Teach a machine to learn the discriminant $f(x)$ using examples from a training dataset
- Be careful to not learn too much the properties of the training sample
  - no need to memorise the training sample
  - instead, interested in getting the right answer for new events
    $\Rightarrow$ generalisation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

Machine learning and connected fields



©Balàzs Kégl





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		

Machine learning: (un)supervised learning

Supervised learning

- Training events are labelled: N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ 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!

An example from Google research team

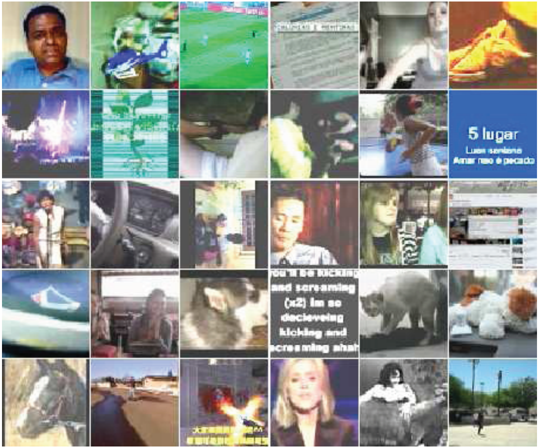
A "giant" neural network

- 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

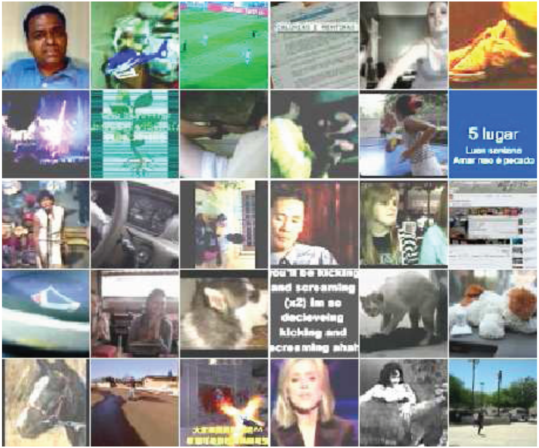
- It learned to recognise faces, one of the original goals
- ... but also cat faces (among the most popular things in YouTube videos) and body shapes

Google's research on building high-level features



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

Google's research on building high-level features



- 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 $(x, y)_1, \ldots, (x, y)_N$ 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:

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

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

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

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

Machine learning: choice of loss function

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:

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

Loss function in classification

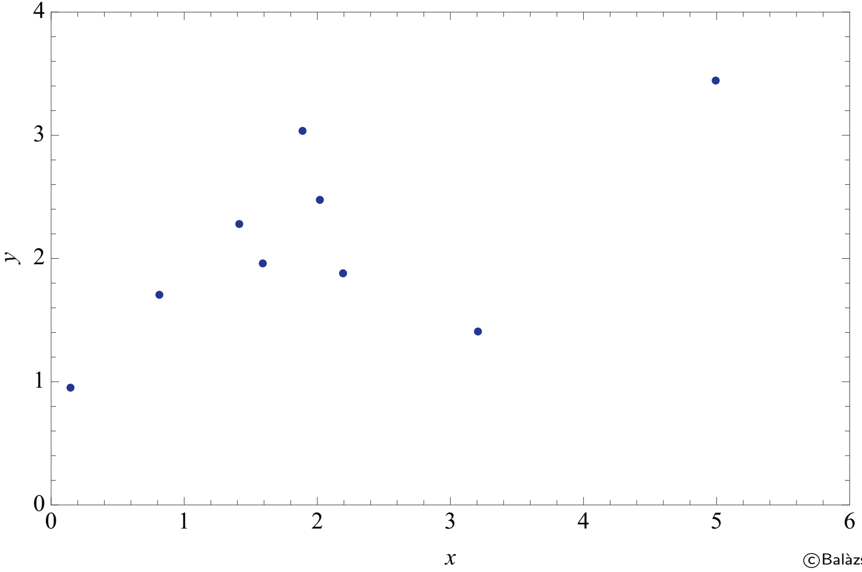
- There is no "distance" between classes
- Goal: $f(x, w)$ predicts properly class y
- Usual loss function is one-loss or zero-one loss:

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

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

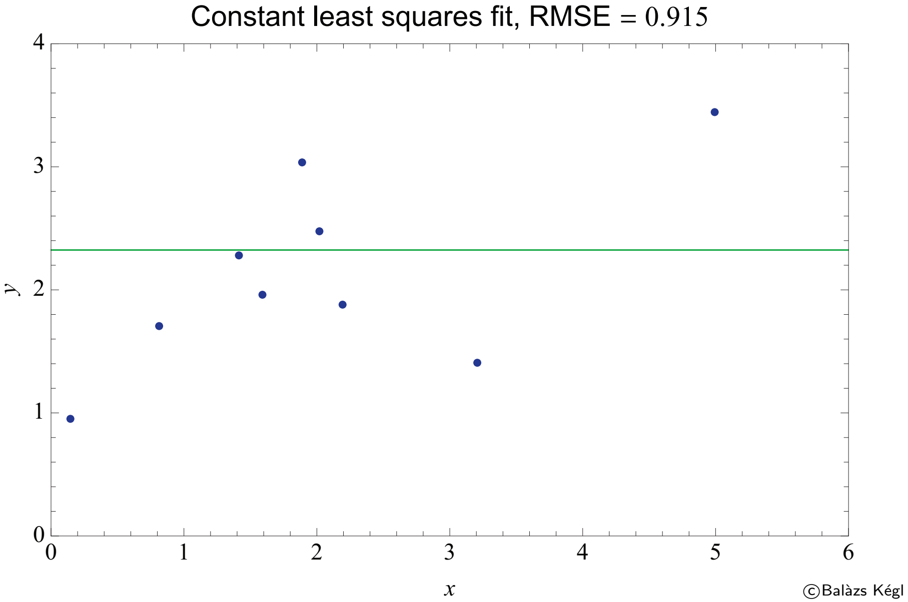
Choice of function class: training

Data generated from an unknown function with unknown noise

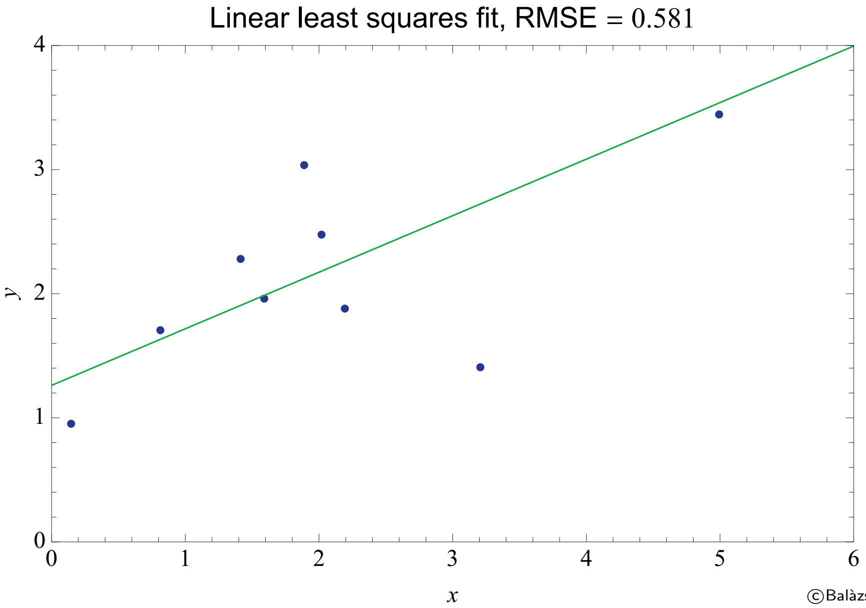


©Balàzs Kégl

Choice of function class: training

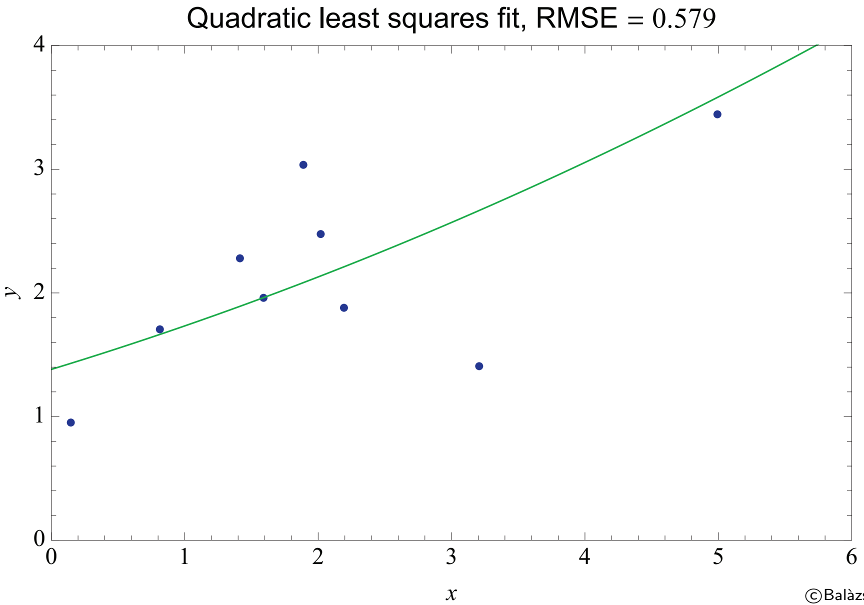


Choice of function class: training



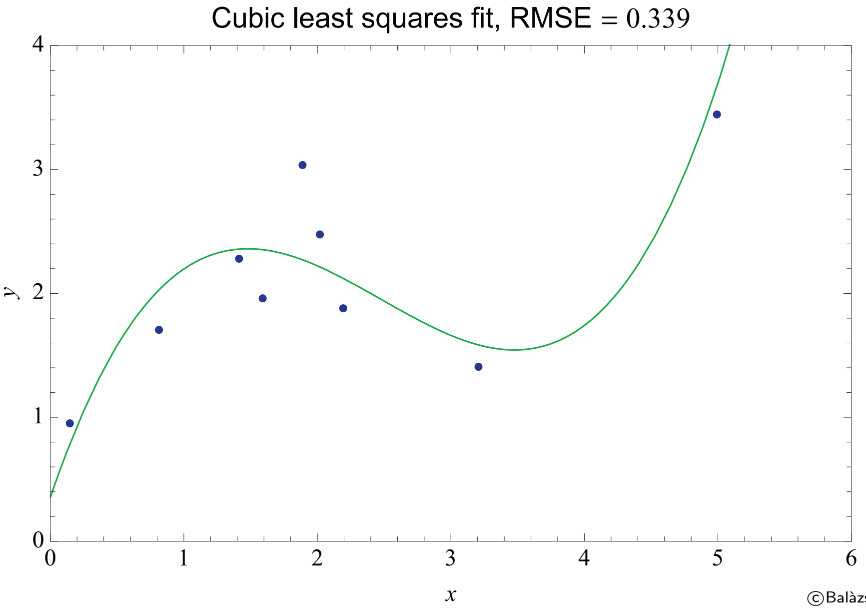
©Balàzs Kégl

Choice of function class: training



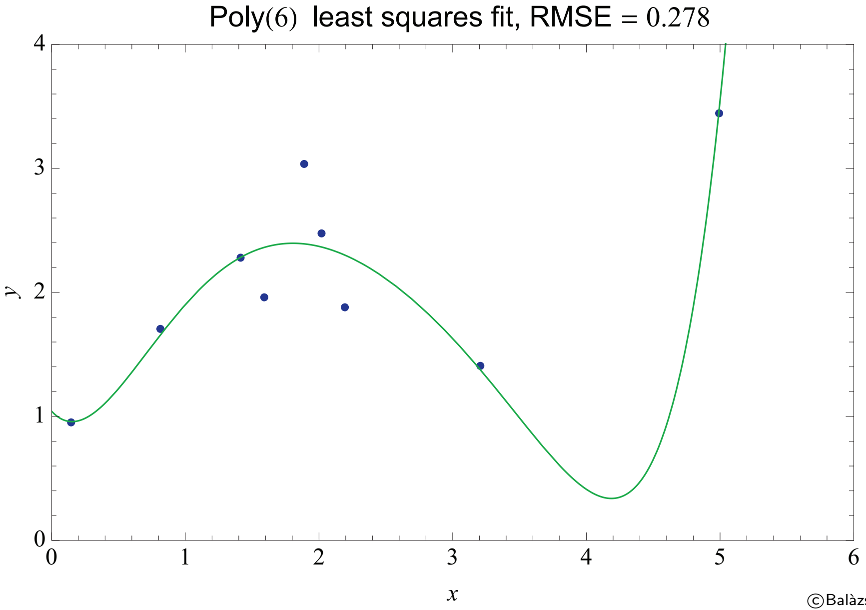
©Balàzs Kégl

Choice of function class: training

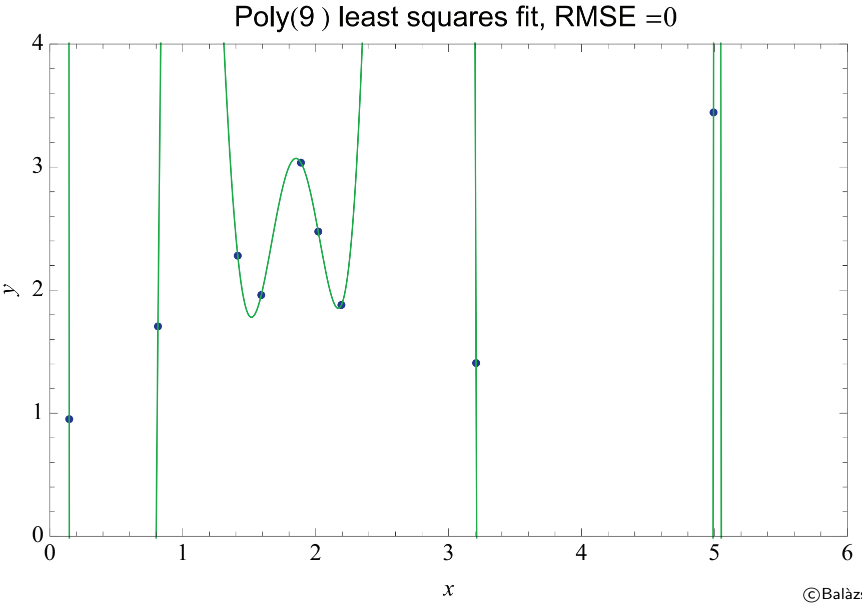


©Balàzs Kégl

Choice of function class: training



Choice of function class: training



©Balàzs Kégl

Choice of function class

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!

Choice of function class

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...

Choice of function class

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!

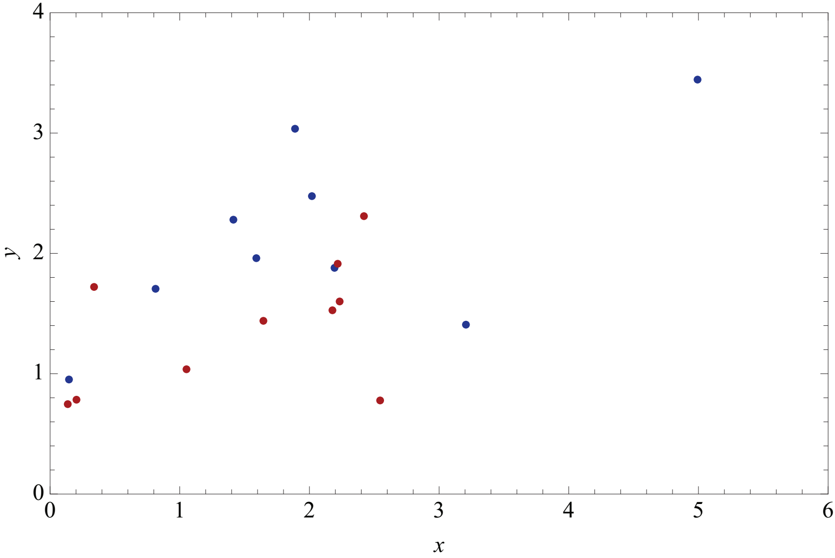
Is it meaningful?

- 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

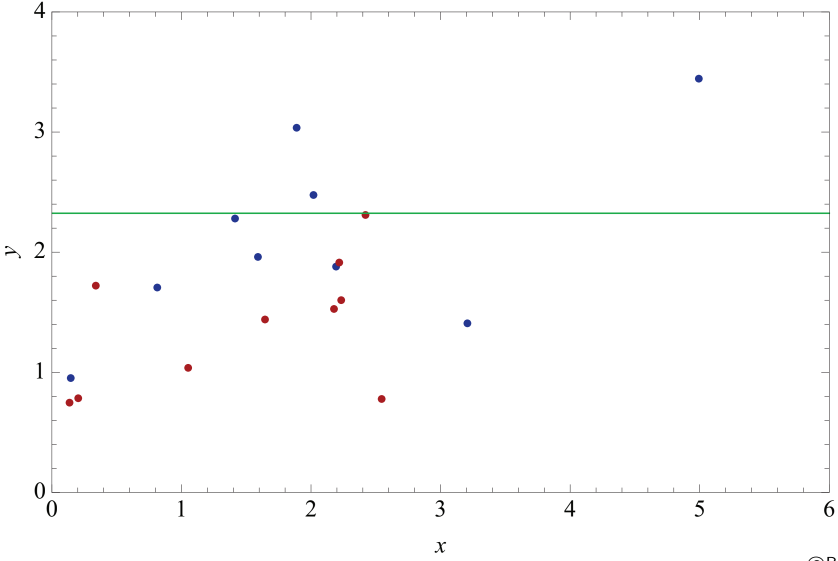
Data generated from an unknown function with unknown noise



©Balàzs Kégl

Choice of function class: testing

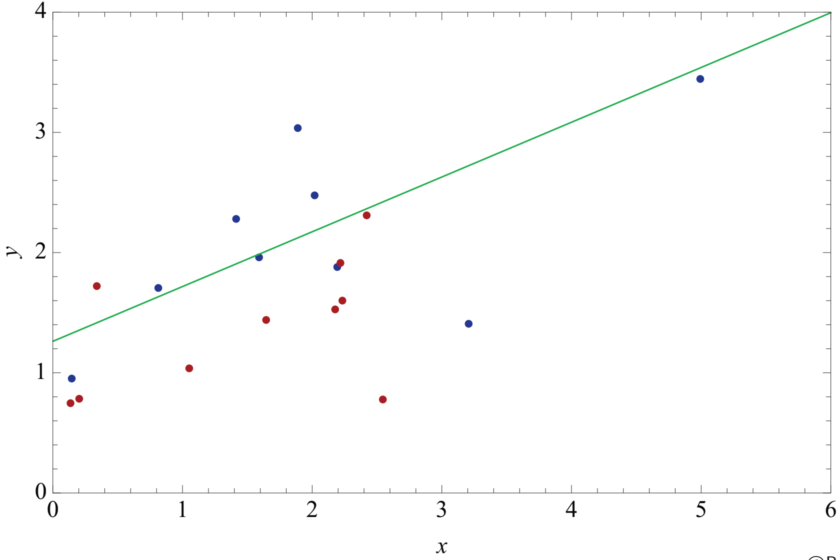
Const. least squares fit, training RMSE = 0.915, test RMSE = 1.067



©Balàzs Kégl

Choice of function class: testing

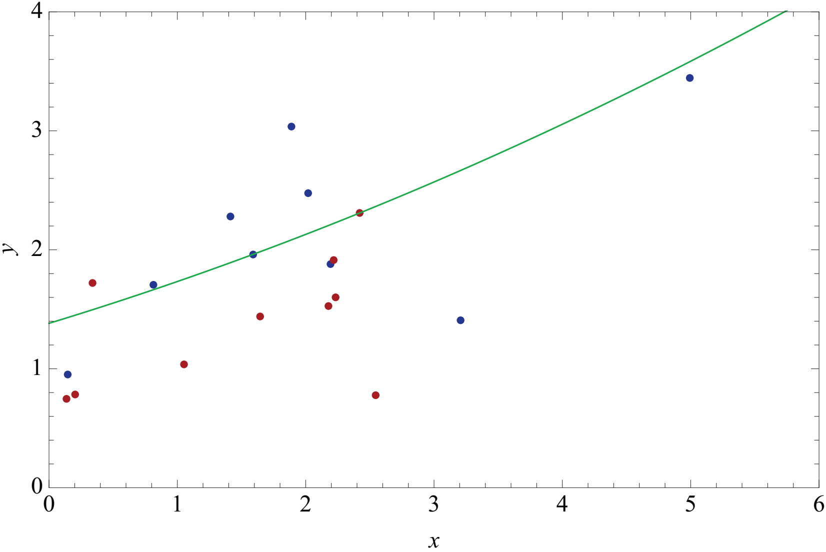
Linear least squares fit, training RMSE = 0.581, test RMSE = 0.734



©Balàzs Kégl

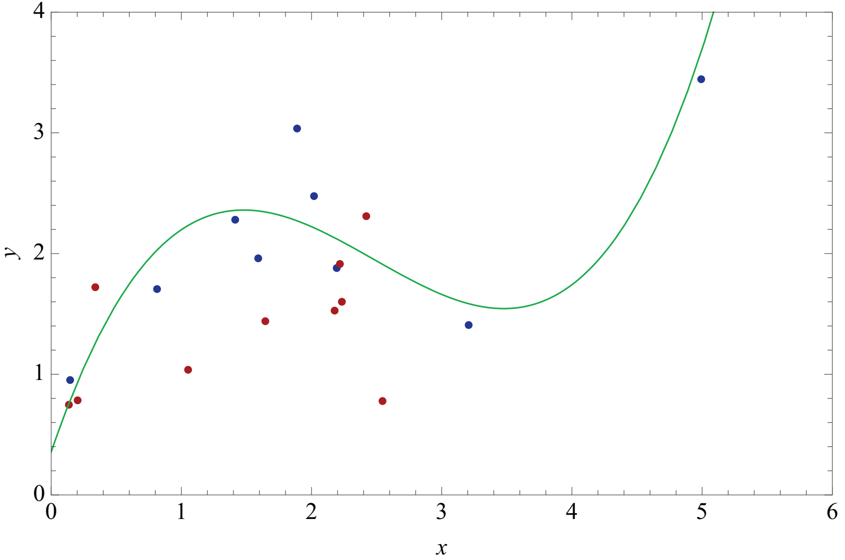
Choice of function class: testing

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



©Balàzs Kégl

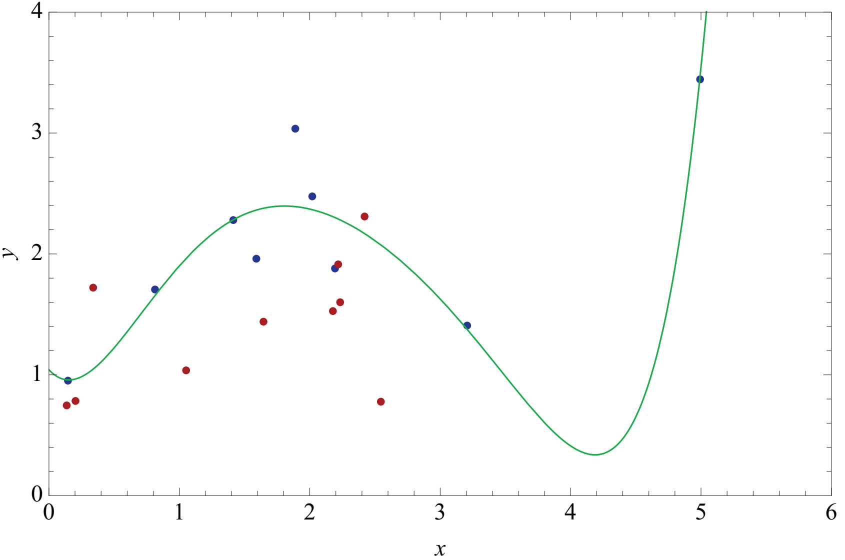
Cubic least squares fit, training RMSE $= 0.339$, test RMSE $= 0.672$



©Balàzs Kégl

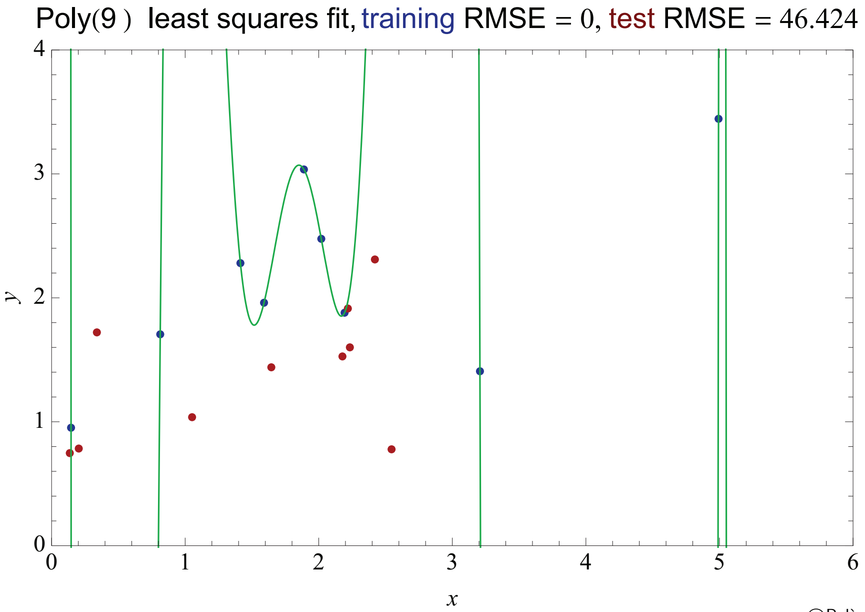
Choice of function class: testing

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



©Balàzs Kégl

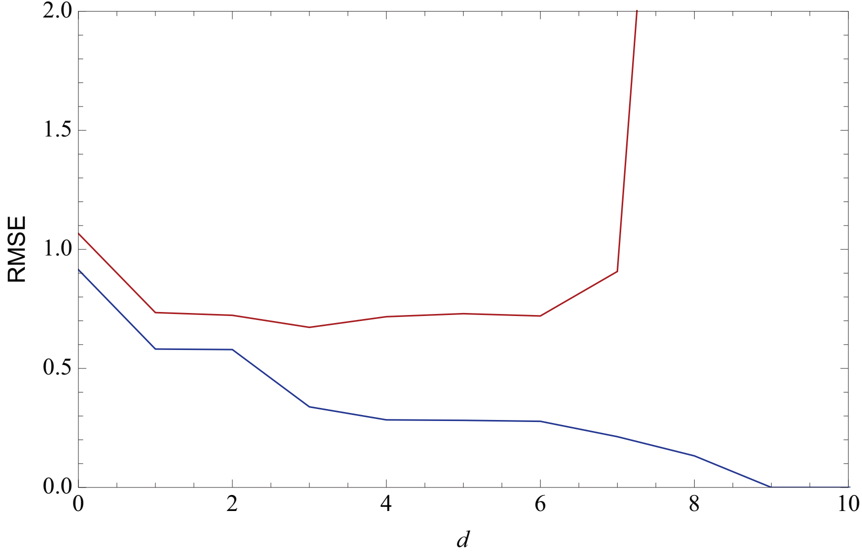
Choice of function class: testing



©Balàzs Kégl

Choice of function class

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



©Balàzs Kégl

Choice of function class

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

Multivariate discriminants

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?

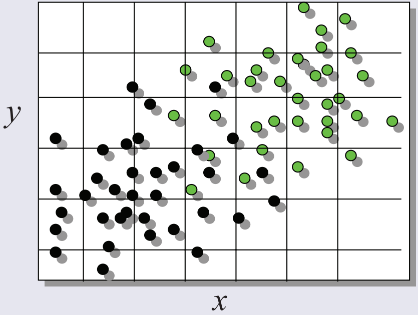
- Best possible choice: cannot beat Bayes limit (but usually impossible to define)
- No single method can be proven to surpass all others in particular case
- Advisable to try several and use the best one

Cut-based analysis and grid search

Cut-based analysis

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

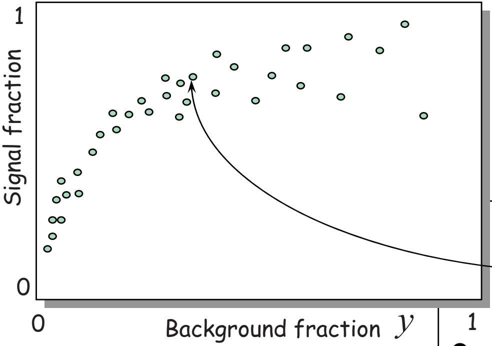
Grid search



©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

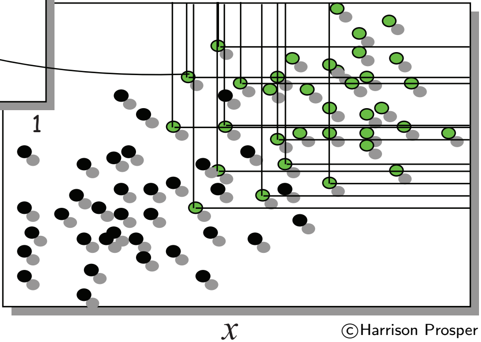
Random grid search



- Use each point in signal sample as grid point:

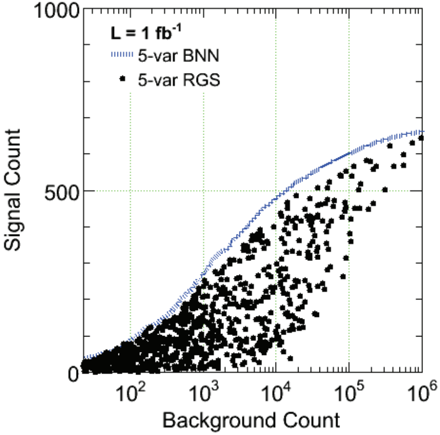
$$x > x_i$$

$$y > y_i$$



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

Random grid search example



©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

Genetic algorithms: survival of the fittest

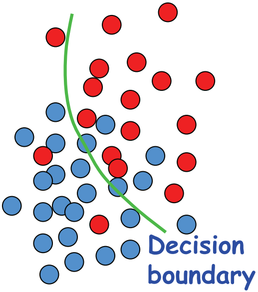
- Inspired by biological evolution
- Model: group (population) of abstract representations (genome/discriminating variables) of possible solutions (individuals/list of cuts)
- Typical processes at work in evolutionary processes:
  - inheritance
  - mutation
  - sexual recombination (a.k.a. crossover)
- 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

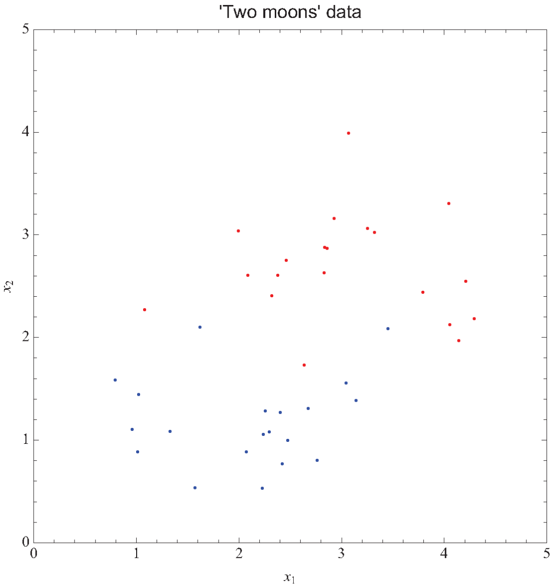
- Suppose densities $s(x)$ and $b(x)$ are multivariate Gaussians:
$$\text{Gaussian}(x|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$
with vector of means μ and covariance matrix Σ
- Then Bayes factor $B(x) = s(x)/b(x)$ (or its logarithm) can be expressed explicitly:
$$\ln B(x) = \lambda(x) \equiv \chi^2(\mu_B,\Sigma_B) - \chi^2(\mu_S,\Sigma_S)$$



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

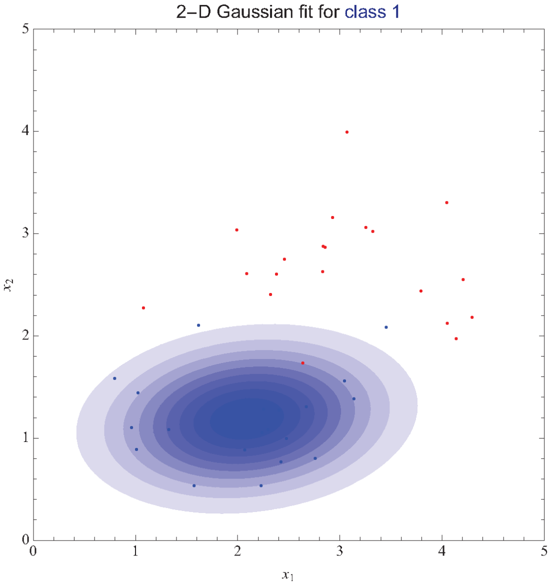
- Fixed value of $\lambda(x)$ defines a quadratic hypersurface partitioning the n-dimensional space into signal-rich and background-rich regions
- Optimal separation if $s(x)$ and $b(x)$ are indeed multivariate Gaussians

Quadratic discriminant



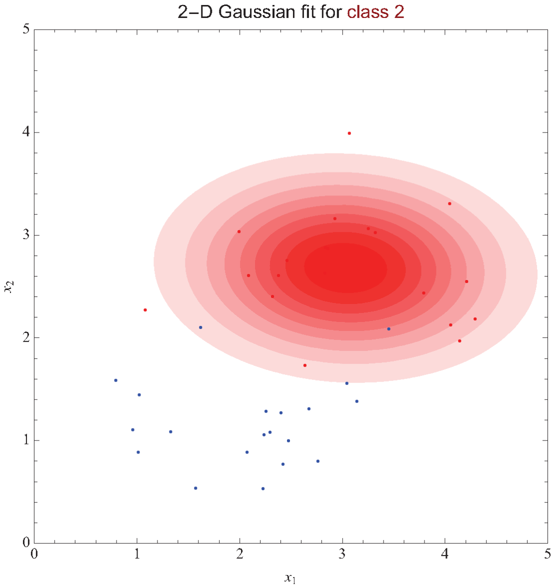
©Balàzs Kégl

Quadratic discriminant



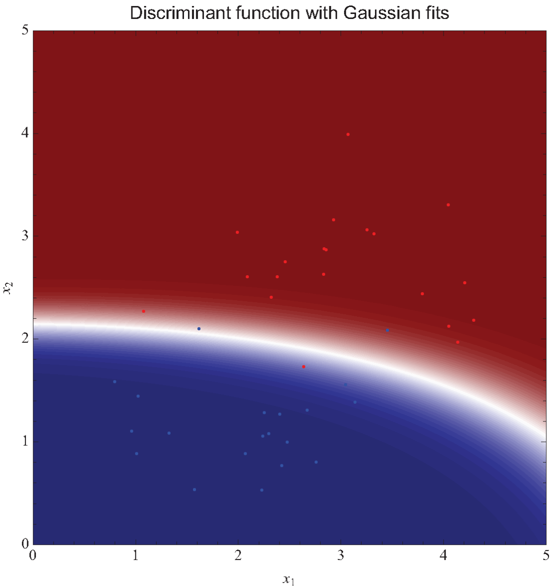
©Balàzs Kégl

Quadratic discriminant



©Balàzs Kégl

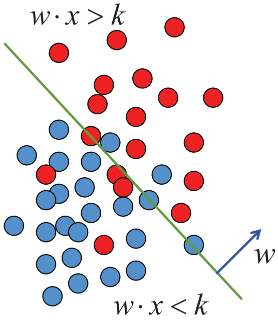
Quadratic discriminant



©Balàzs 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:

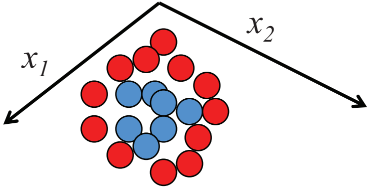
$$\lambda(x) = w \cdot x \quad \text{with} \quad w \propto \Sigma^{-1}(\mu_S - \mu_B)$$

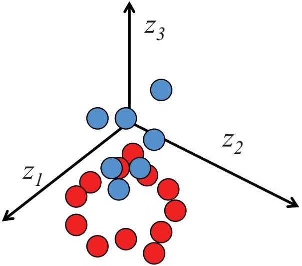


- Optimal linear separation
- Works only if signal and background have different means!
- Optimal classifier (reaches the Bayes limit) for linearly correlated Gaussian-distributed variables

Support vector machines

- 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 $\mathbb{R}^H$: $h : x \in \mathbb{R}^n \to z \in \mathbb{R}^H$
- Use hyper-planes to partition higher dim space: $f(x) = w \cdot h(x) + b$
- Example:$h : (x_1, y_2) \to (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

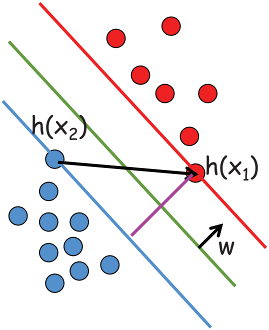




Support vector machines: separable data

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

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



- Subtract blue from red: $w \cdot \left(h(x_1) - h(x_2)\right) = 2$
- With unit vector $\hat{w} = w/\|w\|$: $\hat{w} \cdot \left(h(x_1) - h(x_2)\right) = 2/\|w\| = m$
- 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:

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

  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:

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

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

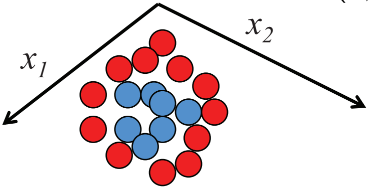
Kernel functions

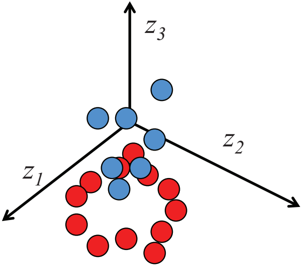
- Issues:
  - need to find h mappings (potentially of infinite dimension)
  - need to compute scalar products $h(x_i) \cdot h(x_j)$
- Fortunately $h(x_i) \cdot h(x_j)$ are equivalent to some kernel function $K(x_i, x_j)$ that does the mapping and the scalar product:

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

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

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



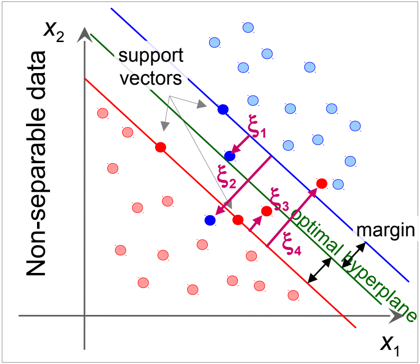


- In reality: do not know a priori the right kernel
- $\Rightarrow$ have to test different standard kernels and use the best one

Support vector machines: non-separable data

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

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

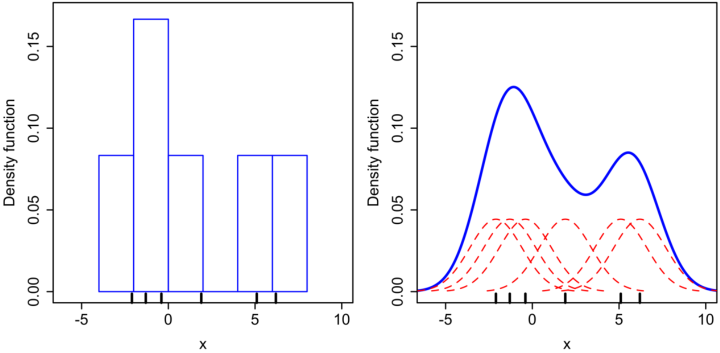


with slack variables $\xi_i > 0$

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

- 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

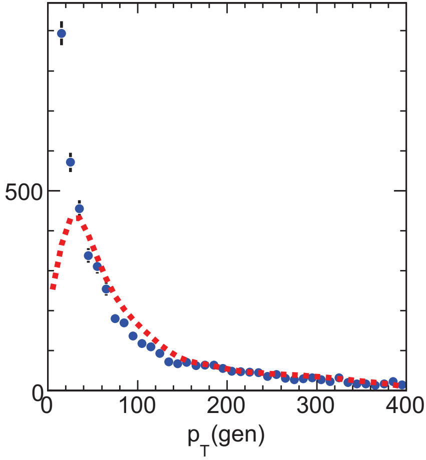
- Too narrow: noisy approximation
- Too wide: loose fine structure
- In principle found by minimising risk function $R(\hat{p}, p) = \int \left(\hat{p}(x) - p(x)\right)^2 dx$
- For Gaussian densities:

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

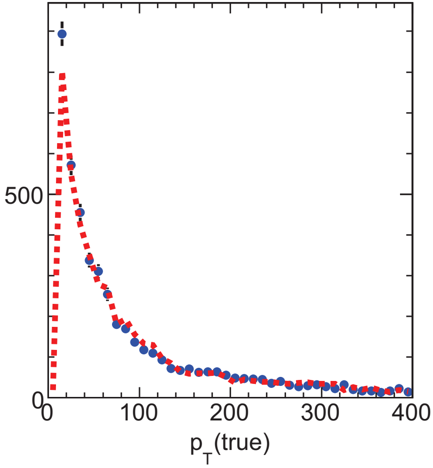
- Far from optimal for non-Gaussian densities

Kernel density estimation (KDE): example

with Gaussian optimal bandwidth



with optimised bandwidth



Kernel density estimation (KDE)

Why does it work?

- When $N \to \infty$:

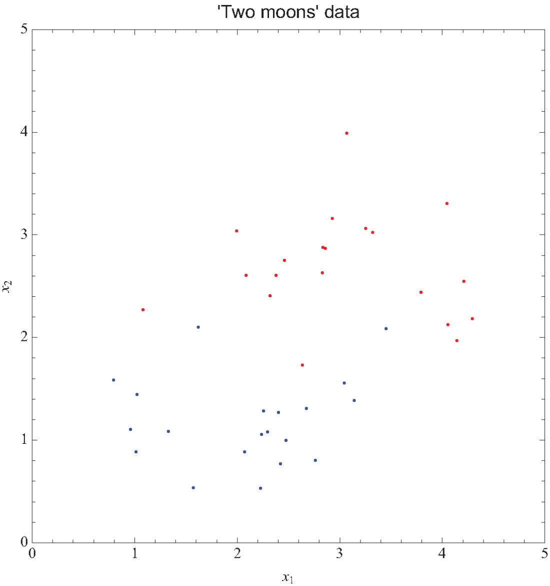
$$\hat{p}(x) = \int K(x, \mu) p(\mu) d\mu$$

- $p(\mu)$: 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

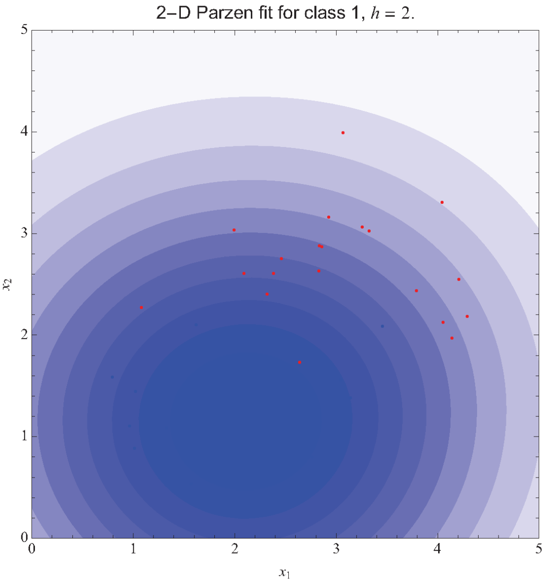
- Choice of bandwidth non-trivial
- Difficult to model sharp structures (e.g. boundaries)
- Kernels too far apart in regions of low point density
- (both can be mitigated with adaptive bandwidth choice)
- Requires evaluation of N n-dimensional kernels

Kernel density estimation (KDE)



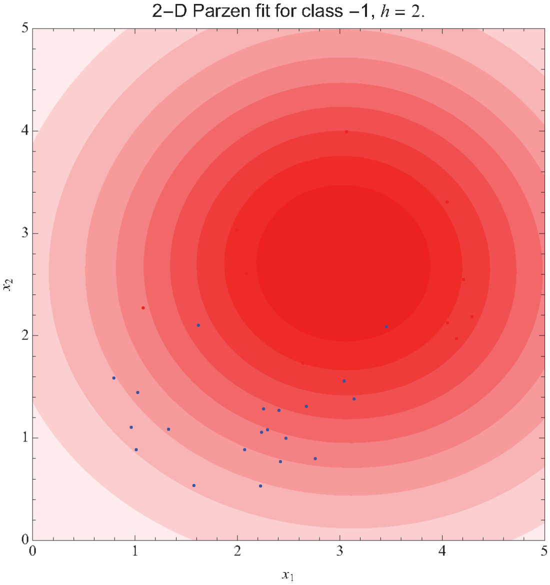
©Balàzs Kégl

Kernel density estimation (KDE)



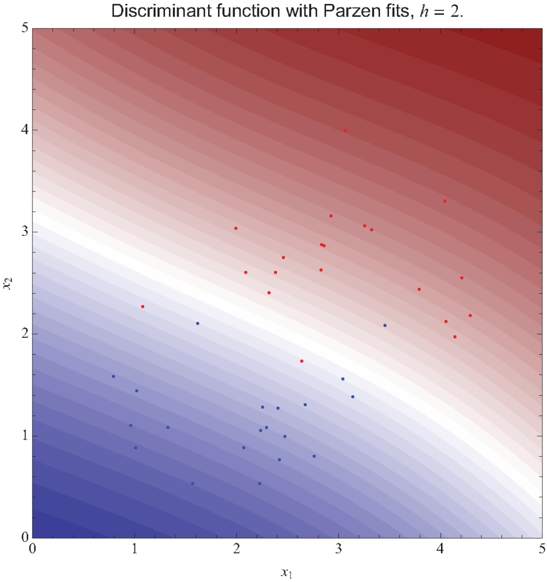
©Balàzs Kégl

Kernel density estimation (KDE)

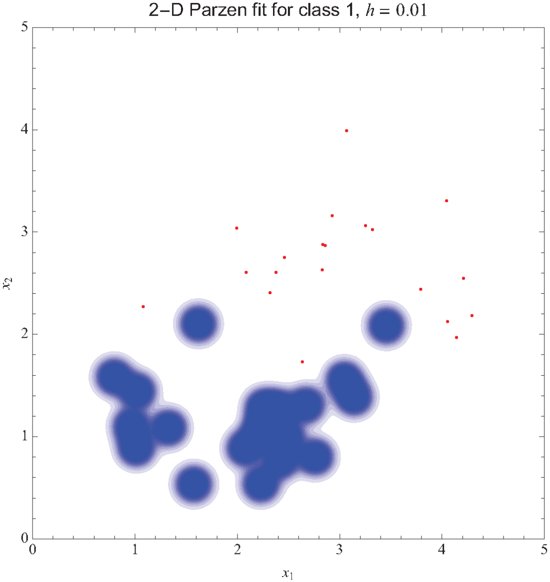


©Balàzs Kégl

Kernel density estimation (KDE)

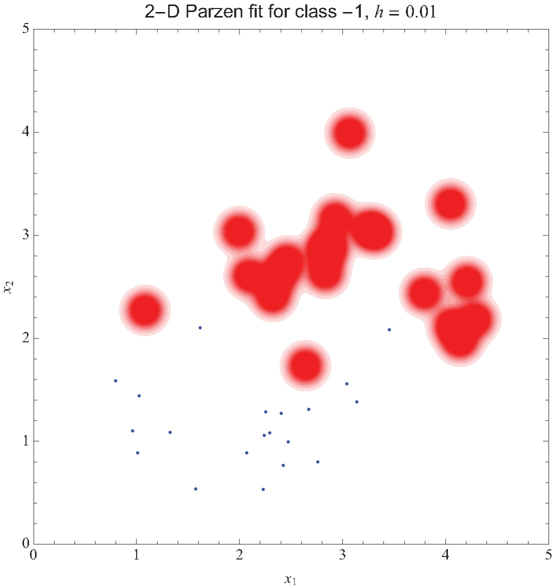


Kernel density estimation (KDE)



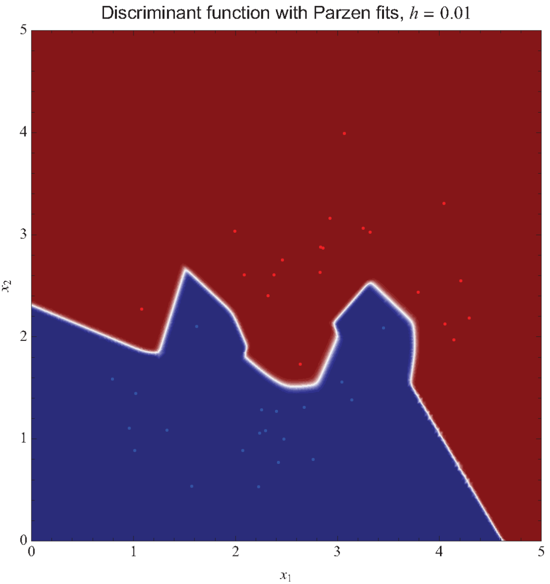
©Balàzs Kégl

Kernel density estimation (KDE)



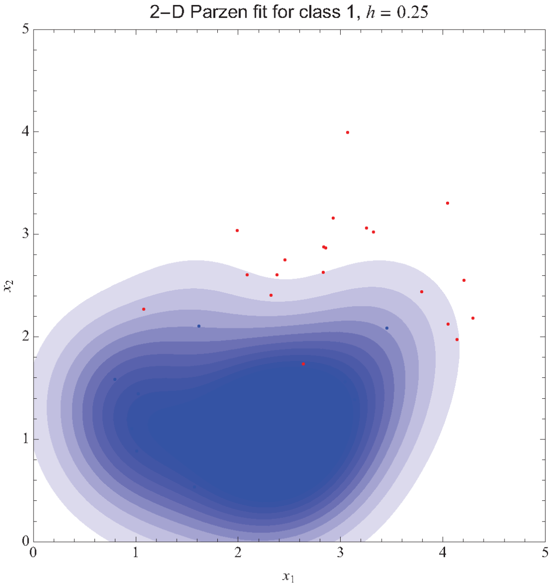
©Balàzs Kégl

Kernel density estimation (KDE)



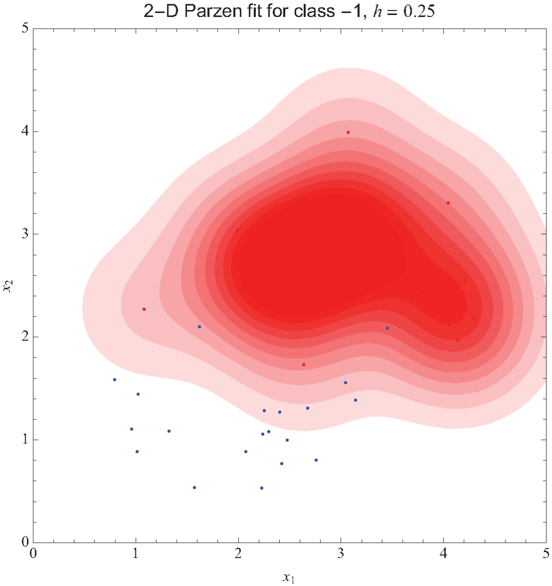
©Balàzs Kégl

Kernel density estimation (KDE)



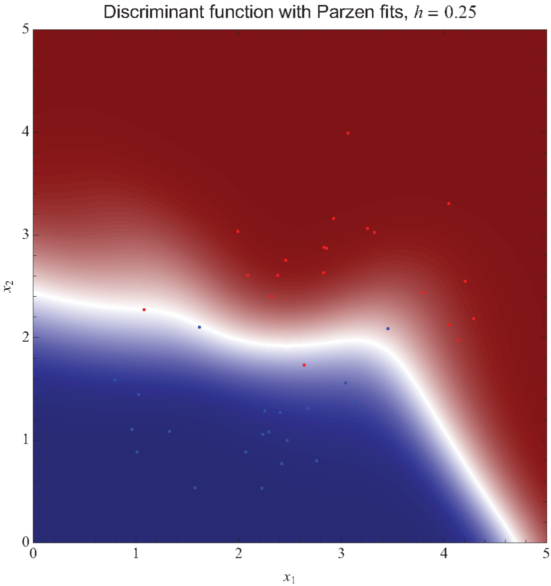
©Balàzs Kégl

Kernel density estimation (KDE)



©Balàzs Kégl

Kernel density estimation (KDE)



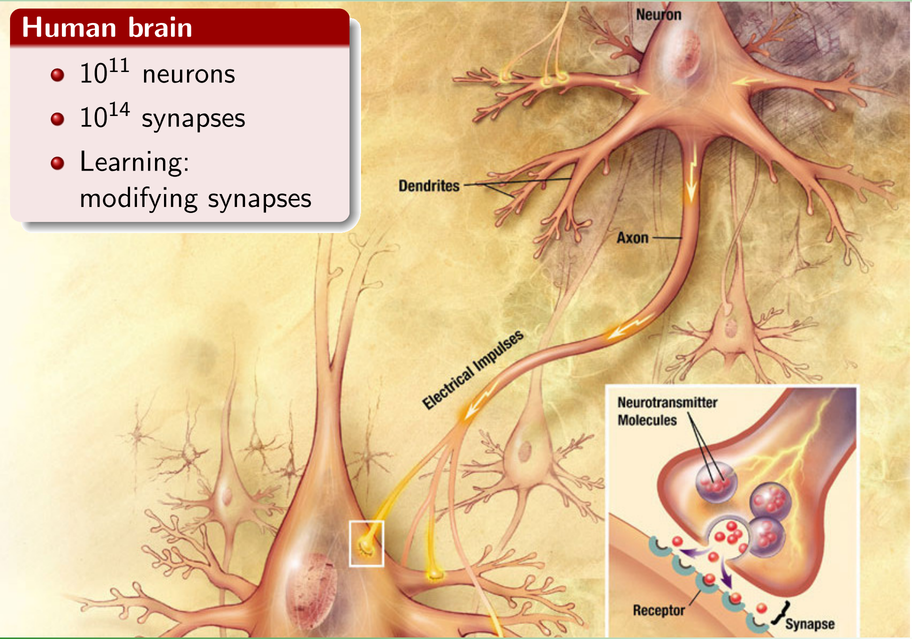
©Balàzs Kégl

KDE: choice of bandwidth

©Balàzs Kégl

Human brain

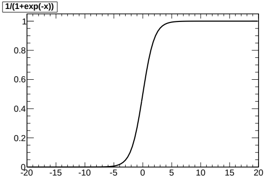
- 10^{11} neurons
- 10^{14} synapses
- Learning:
  modifying synapses



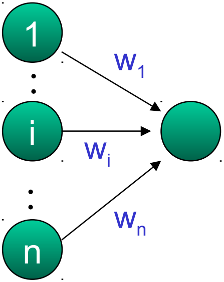
Brief history of artificial neural networks

- 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
- 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):



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



- $\sigma(\lambda)$ is neuron activity, λ is activation
- 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

Neural networks

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 $|y(x) - f(x)| < \varepsilon$

Neural networks

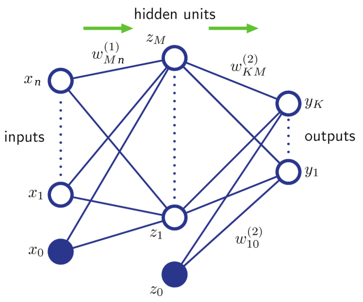
Interpretation

- 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

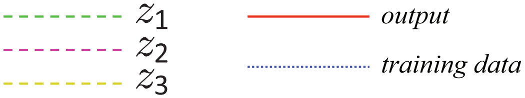
- Neurons organised in layers
- Output of one layer becomes input to next layer

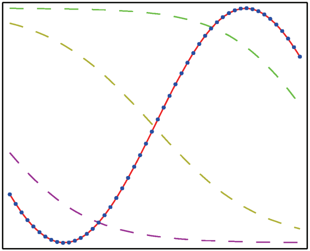
$$y_k(x, w) = \sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{\sigma\left(\sum_{i=0}^{n} w_{ji}^{(1)} x_i\right)}_{z_j}$$

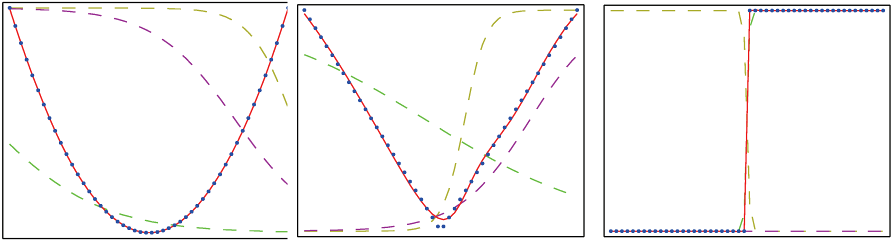


- 1 input (training data), 1 output
- 3 hidden neurons on one hidden layer

©Jan Therhaag







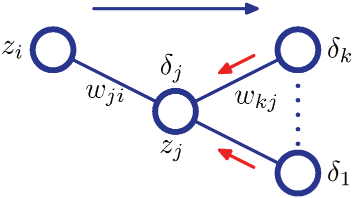
Backpropagation

- 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:

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

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

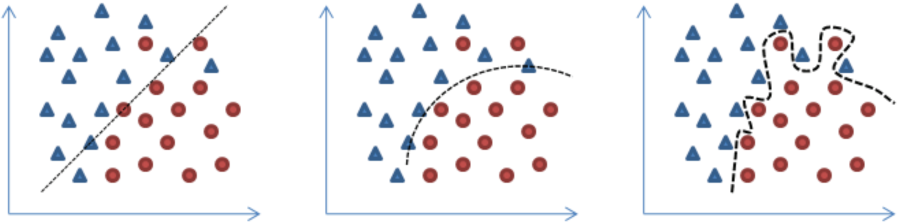
- Hence errors are propagated backwards



Neural network training

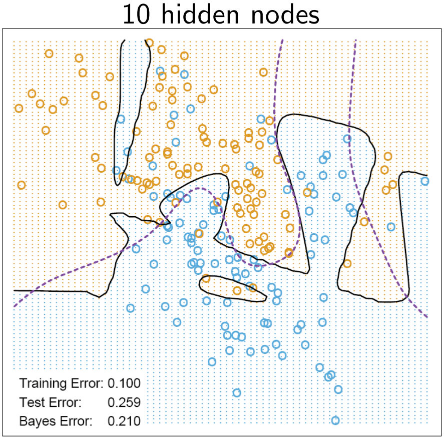
- Minimise error function $E(w)$
- Gradient descent: $w^{(k+1)} = w^{(k)} - \eta \frac{dE^{(k)}}{dw}$
- $\frac{\partial E}{\partial w_j} = \sum_{n=1}^{N} -(t^{(n)} - y^{(n)})x_j^{(n)}$ with target $t^{(n)}$ (0 or 1), so $t^{(n)} - y^{(n)}$ is the error on event n
- All events at once (batch learning):
    - 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

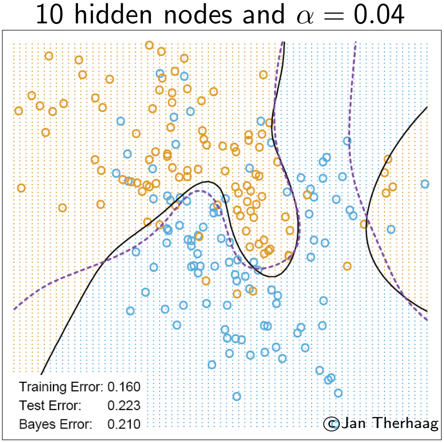
Neural network overtraining



- 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:

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

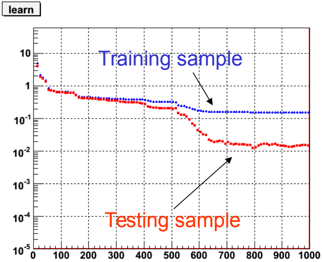




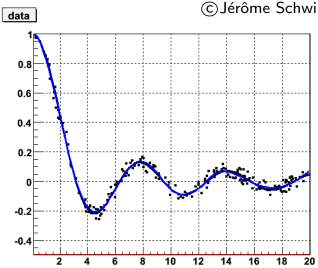
- Much less overfitting, better generalisation properties

Getting confused: testing better than training?

- 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: $\sin(x)/x + rand(-0.05, +0.05)$
- Of course doesn't work as well if noise is not symmetric



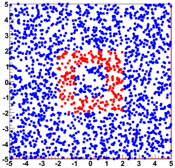
©Jérôme Schwindling

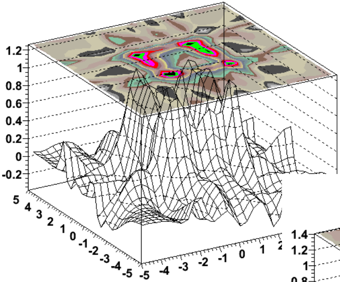


Neural networks: Tricks of the trade

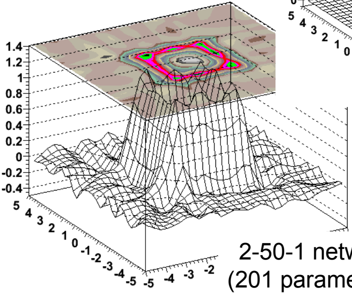
- Preprocess data:
  - if relevant, provide e.g. x/y instead of x and y
  - subtract the mean because the sigmoid derivative becomes negligible very fast (so, input mean close to 0)
  - normalise variances (close to 1)
  - shuffle training sample (order matters in online training)
- Initial random weights 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

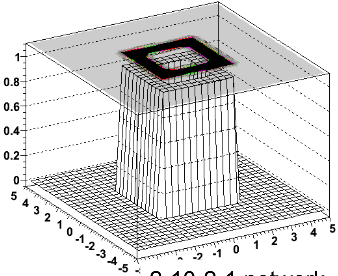




2-20-1 network
(81 parameters)



2-50-1 network
(201 parameters)



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

Bayesian neural networks

- As name says: Bayesian approach, try to *infer* functions $f(x)$
- Training sample T of N examples $(x, y)_1, (x, y)_2, \ldots, (x, y)_N$ 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)

Bayesian neural networks

- Take the mean of the predictive distribution:

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

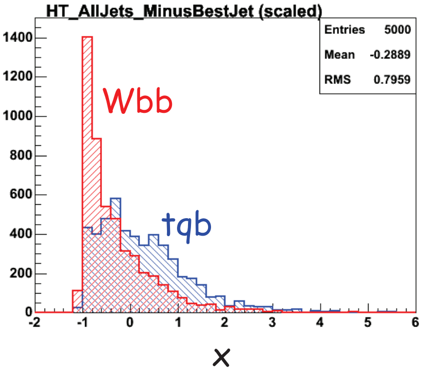
- Why? For classification $p(y|x,w) = f(x,w)^y \left(1 - f(x,w)\right)^{1-y}$
  - for $y = 1$: $p(y|x,w) = f(x,w)$
  - for $y = 0$: $p(y|x,w) = 1 - f(x,w)$
  - so only $f(x,w)$ contributes to the mean
- Example usage:

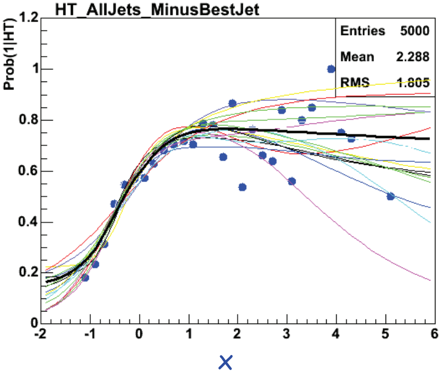
$$\begin{aligned} f(x,w) &= \frac{1}{1+e^{-g(x,w)}} \\ g(x,w) &= b + \sum_{j=1}^{H} v_j \tanh(a_j + \sum_{i=1}^{n} u_{ij} x_i) \end{aligned}$$

with H hidden nodes

Bayesian neural networks

- 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 $w_1, \ldots, w_M$ from posterior density $p(w|T)$
- $y(x) \approx \frac{1}{M}\sum_{m=1}^{M} f(x, w_m)$
- Use spare 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 $D(x)$

Deep learning

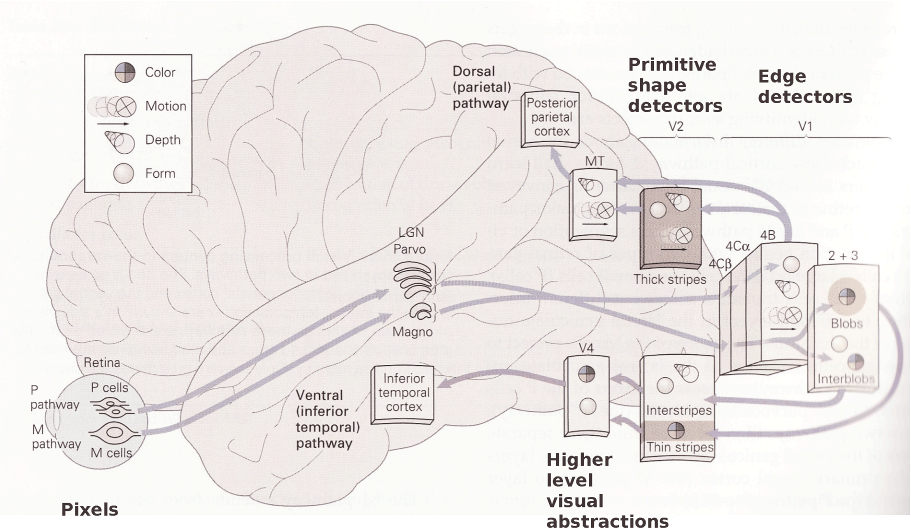
What is learning?

- 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

- 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



Deep learning in artificial intelligence

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 learning revolution

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 learning revolution

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

Greedy layer-wise pre-training

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

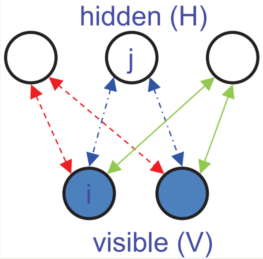
Feature extractors

- 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

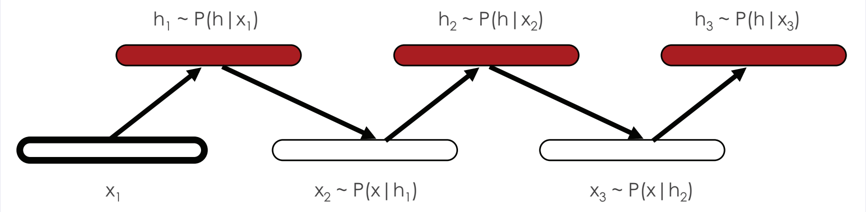
Restricted Boltzmann machine

Energy minimisation

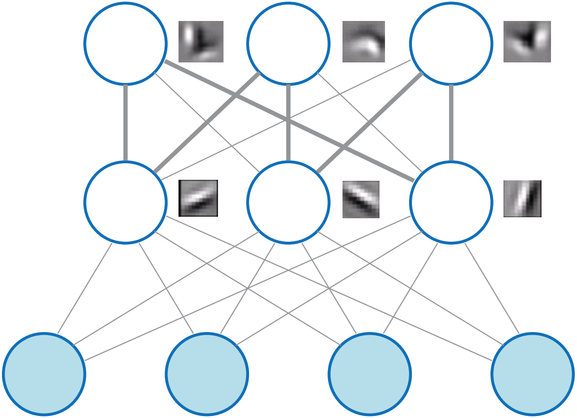
- RBM: visible and hidden variables, without hidden-hidden or visible-visible connections
- Energy of joint combination of visible and hidden units:
  $E(v, h) = -b^T v - c^T h - h^T W v$
- b/c: biases of visible/hidden nodes
- W: weights connecting visible and hidden units
- Probability: $p(v, h) \propto e^{-E(v,h)}$



RBM training: Gibbs sampling



Learning feature hierarchy



Higher layer: DBNs (Combinations of edges)

First layer: RBMs (edges)

Input image patch (pixels)

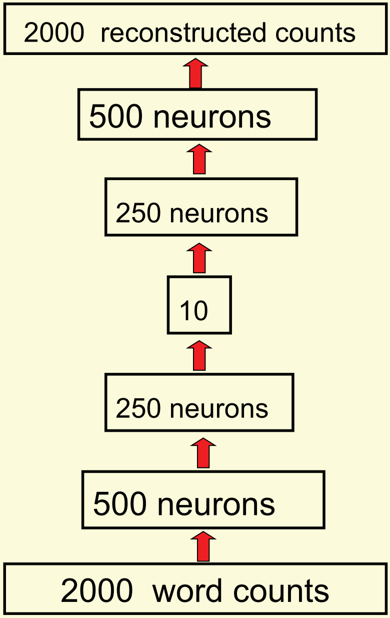
Auto-encoders

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

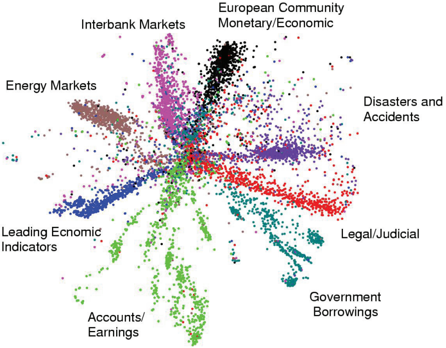
Finding similar books

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



With PCA

With autoencoder



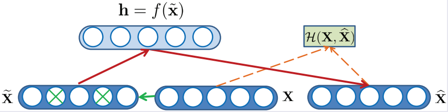
Other auto-encoders

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

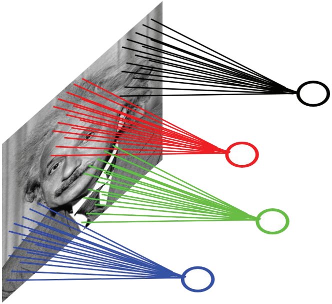


Locally connected auto-encoder

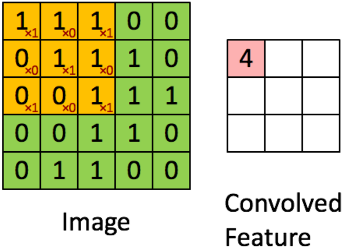
- Allow hidden units to connect only to small subset of input units
- Useful with increasing number of input features (e.g., bigger image)
- Inspired by biology: visual system has localised receptive fields

Convolutional networks

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

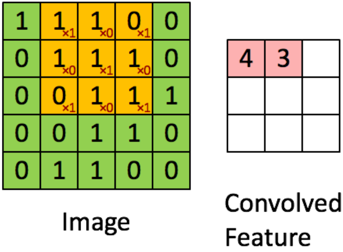


- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



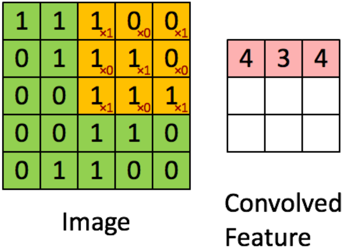
Convolutional networks

- 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

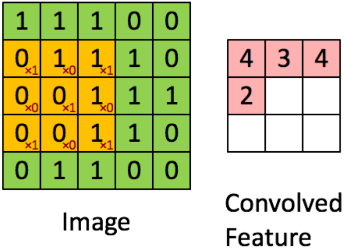


Convolutional networks

- 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

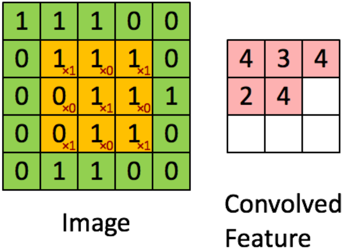


- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image



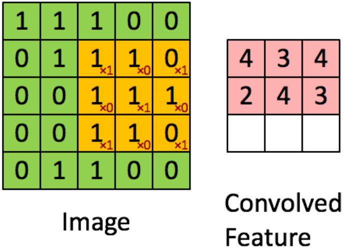
Convolutional networks

- 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



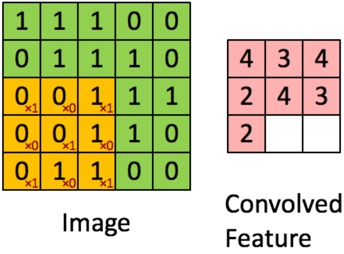
Convolutional networks

- 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



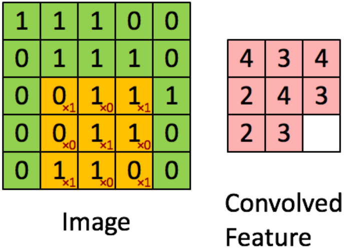
Convolutional networks

- 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



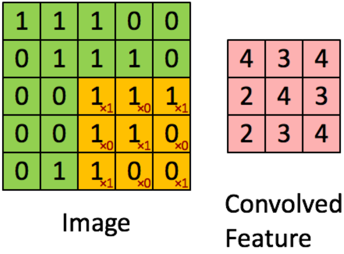
Convolutional networks

- 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



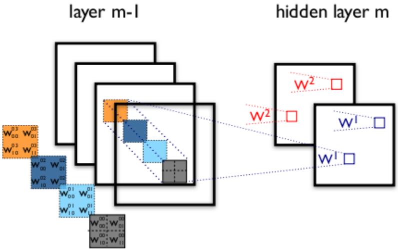
Convolutional networks

- 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



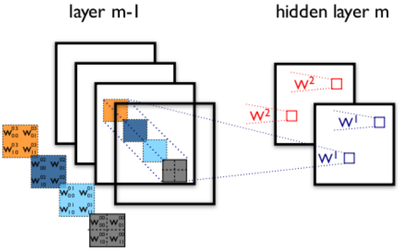
Convolutional networks

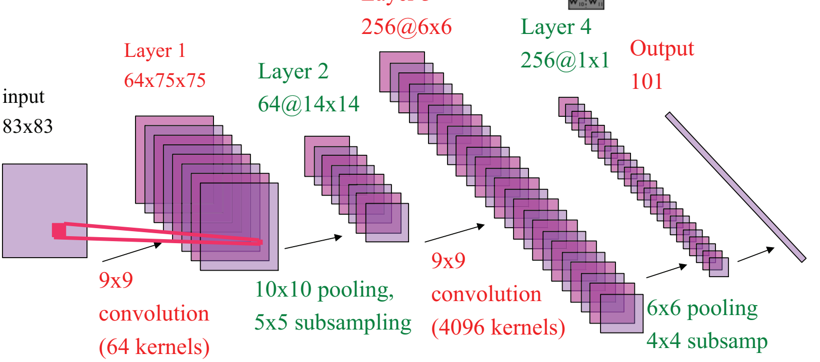
- Images are stationary: can learn feature in one part and apply it in another
- Use e.g. small patch sampled randomly, learn feature, convolve with full image
- Build several "feature maps"



Convolutional networks

- 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

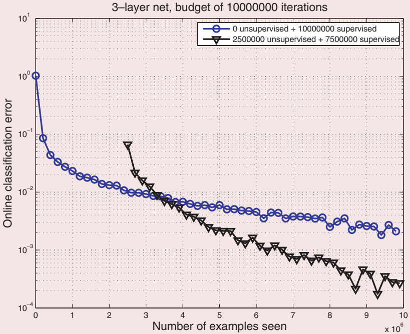
- Training one layer at a time scales well
- Backpropagation from sensible features
- Better local minimum than random initialisation, local search around it

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

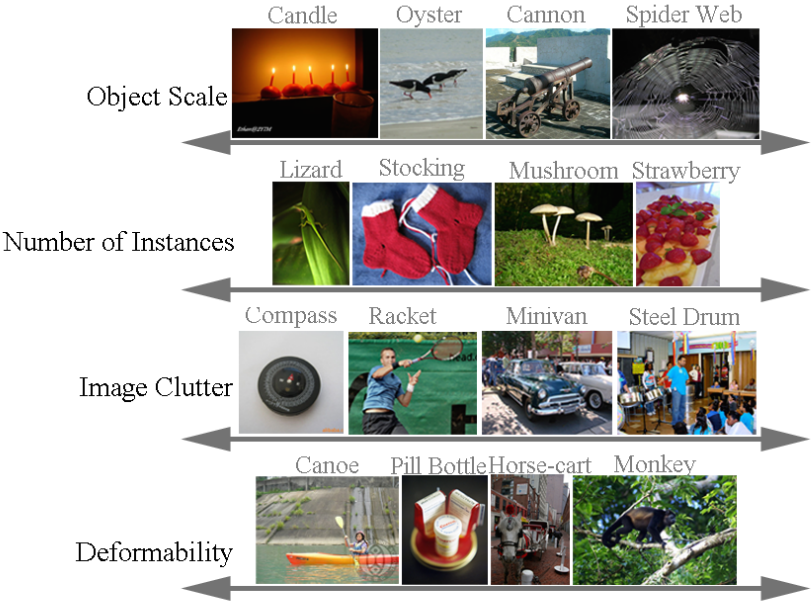
Deep learning: looking forward

- Very active field of research in machine learning and artificial intelligence
  - not just at universities (Google, Facebook, Microsoft, NVIDIA, etc...)
- Possible path forward: continuation methods
  - solve easier or smoothed version first, and gradually consider less smoothing
- Training with curriculum:
  - what humans do over 20 years
  - learn different concepts at different times
  - 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

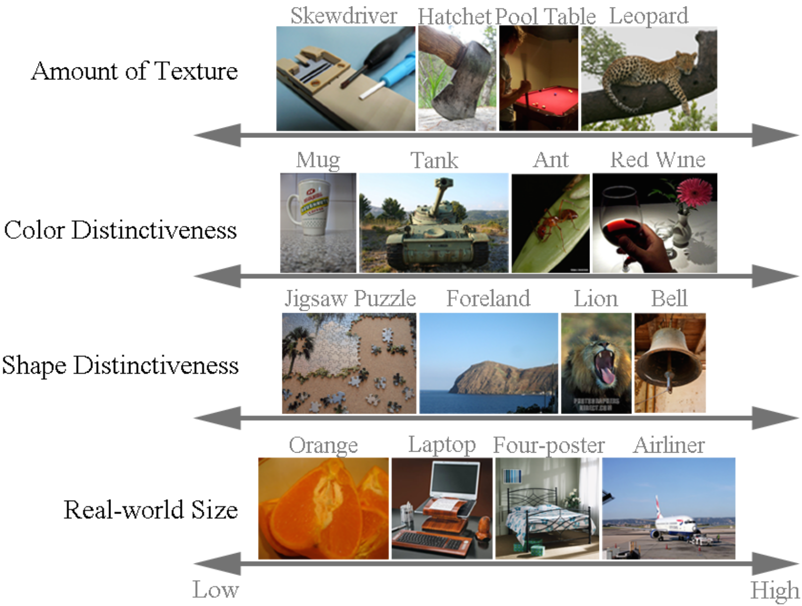
ImageNet Large Scale Visual Recognition Challenge

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

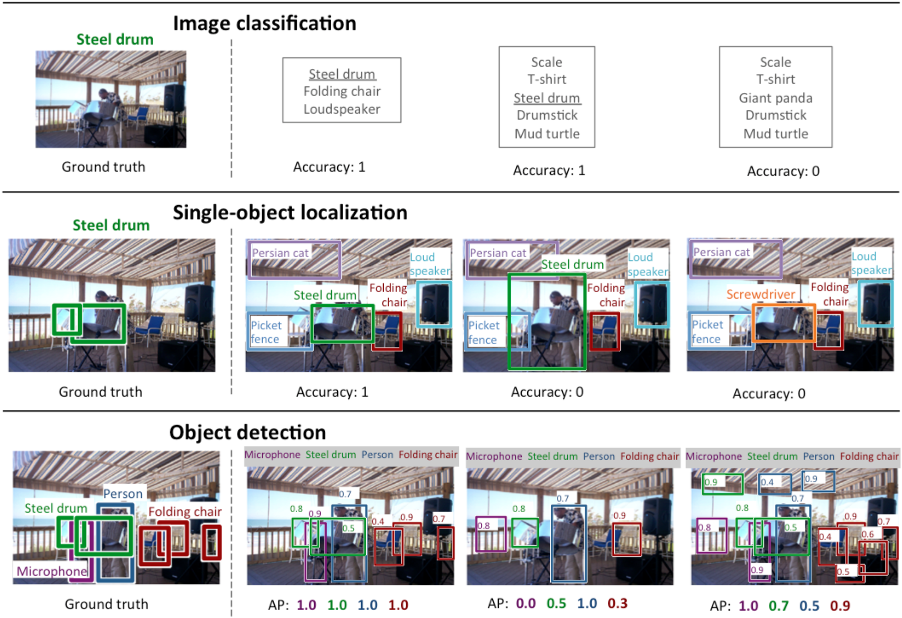
ILSVRC 2014 images



ILSVRC 2014 images

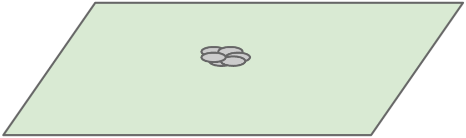


ILSVRC 2014 tasks



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

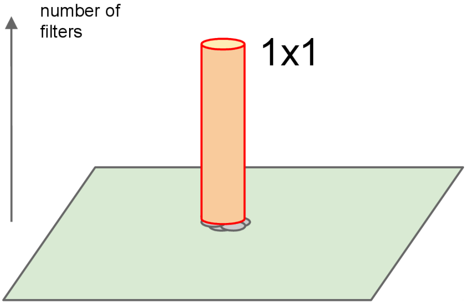
In images, correlations tend to be local



ILSVRC 2014 And the winner is. . .

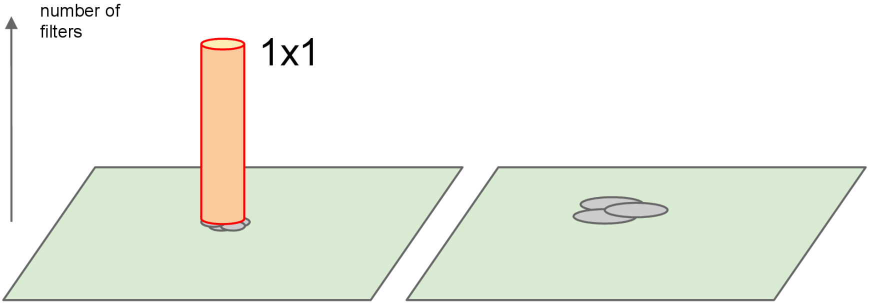
- Google of course! (first time)
- GoogLeNet:

Cover very local clusters by 1x1 convolutions



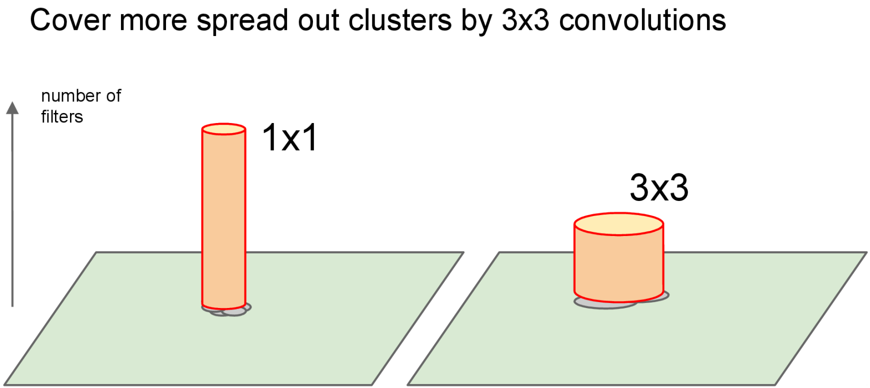
- Google of course! (first time)
- GoogLeNet:

Less spread out correlations



ILSVRC 2014 And the winner is. . .

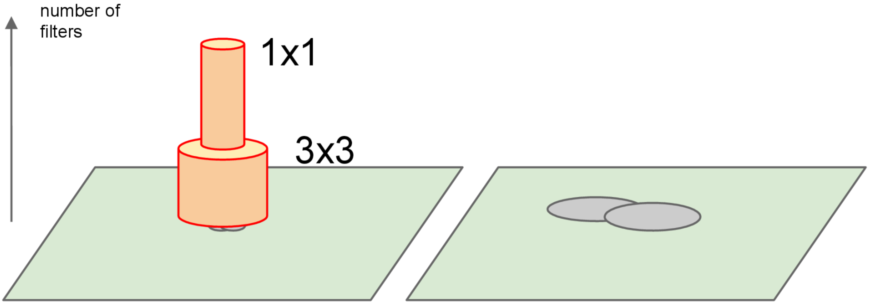
- Google of course! (first time)
- GoogLeNet:



ILSVRC 2014 And the winner is. . .

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

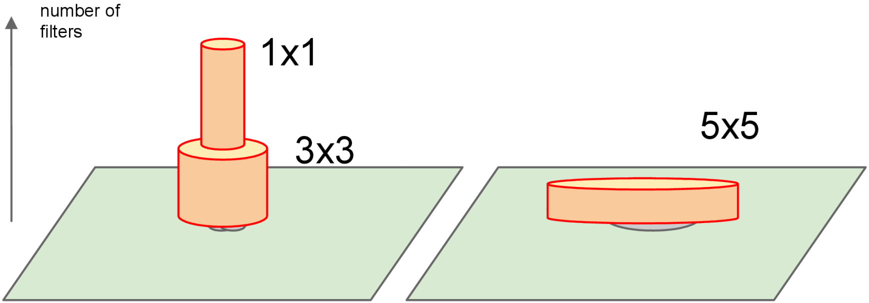
Cover more spread out clusters by 5x5 convolutions



ILSVRC 2014 And the winner is. . .

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

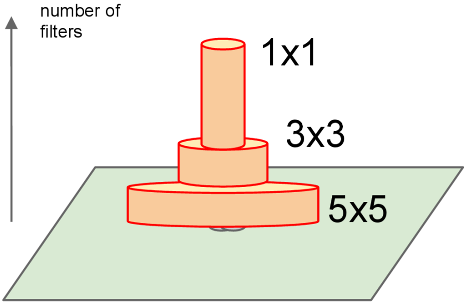
Cover more spread out clusters by 5x5 convolutions



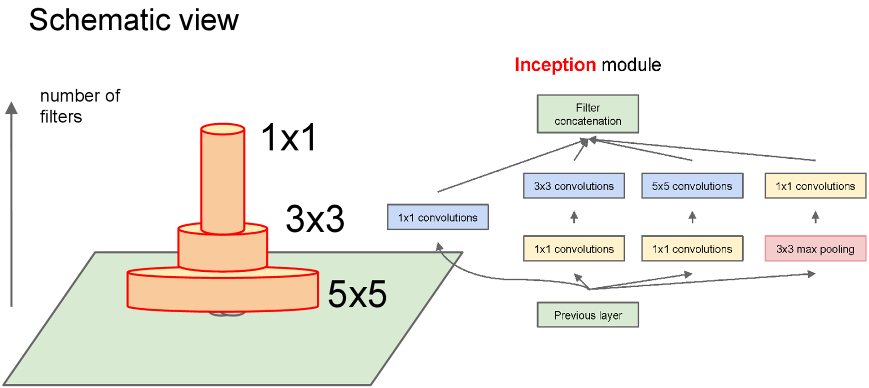
ILSVRC 2014 And the winner is. . .

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

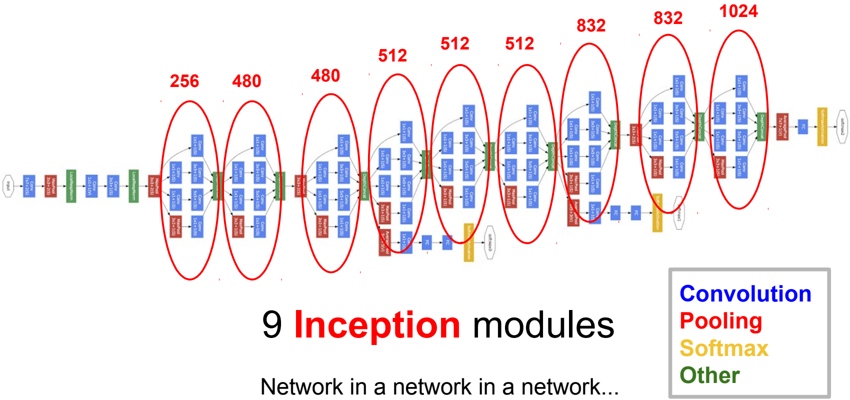
A heterogeneous set of convolutions



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



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



9 **Inception** modules

Network in a network in a network...

Classification failure cases

Groundtruth: **Police car**

GoogLeNet:

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

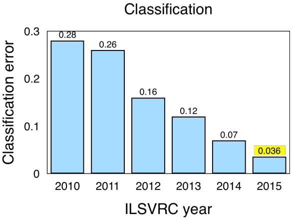
Classification failure cases

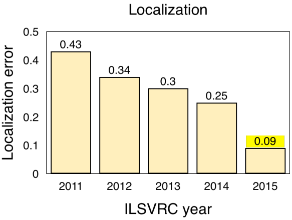
Groundtruth: **hay**

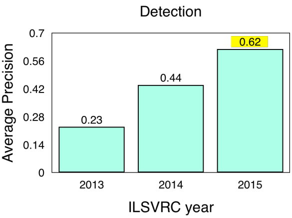
GoogLeNet:

- **sorrel (horse)**
- **hartebeest**
- **Arabian camel**
- **warthog**
- **gaselle**

ILSVRC 2010–2015





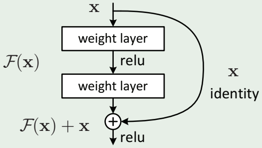


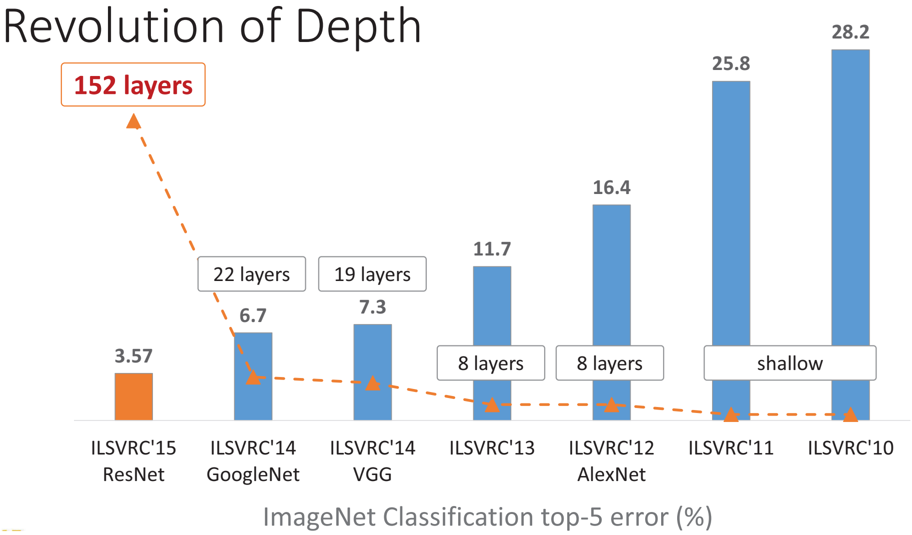
2010–14: 4.2x reduction   1.7x reduction   1.9x increase

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)





Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)

Google DeepMind: arcade games

- 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 $Q^*(s,a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s, a_t = a, \pi]$
  - 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

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

    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

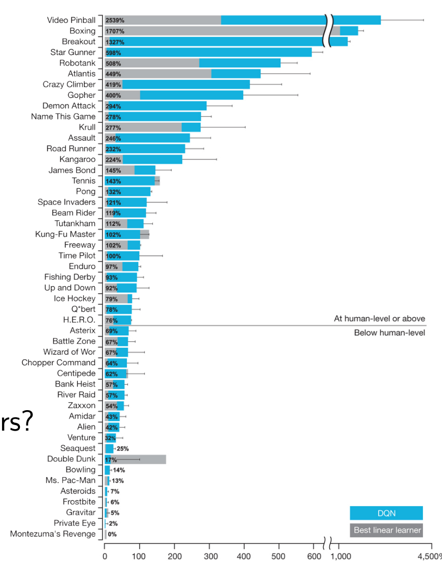
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    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?

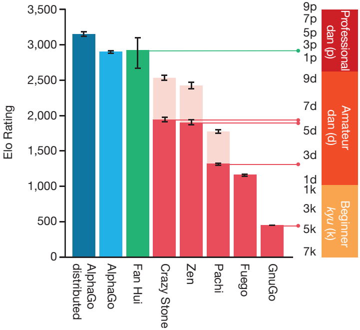
Google DeepMind: mastering Go

▸ Nature 529, 484 (2016)

- 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 bit human world champion) ⇒ better position selection (policy network) and better evaluation (value network)
- Will play Lee Sedol (top Go play in the world over last decade) in March 2016



Next lecture

Summary of MVA techniques

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

(according to TMVA authors)

©Andreas Hoecker

Conclusion

- 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 knowning and can easily outsmart you for many tasks
- Try this to convince yourself http://www.phi-t.de/mousegame/index_eng.html

References I

- 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

References II

- 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
- 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