Statistical Methods in Particle Physics

8. Multivariate Analysis

Prof. Dr. Klaus Reygers (lectures)
Dr. Sebastian Neubert (tutorials)

Heidelberg University
WS 2017/18

Multi-Variate Classification

Consider events which can be either signal or background events.

Each event is characterized by *n* observables:

$$\vec{x} = (x_1, ..., x_n)$$ "feature vector"

Goal: classify events as signal or background in an optimal way.

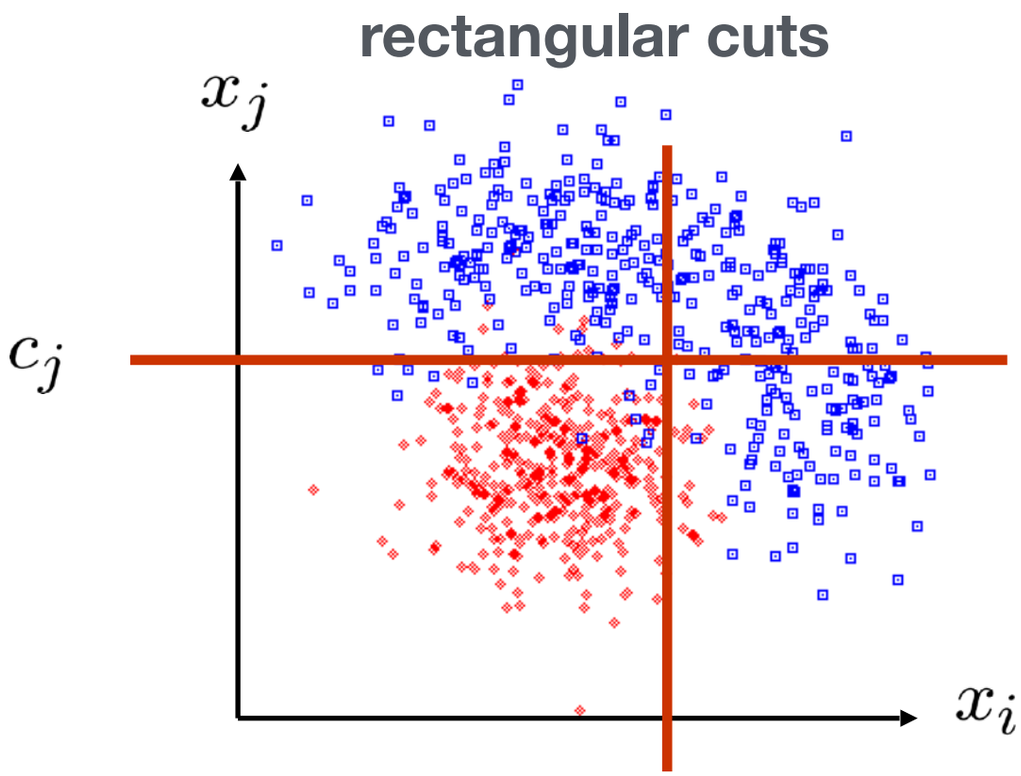
This is usually done by mapping the feature vector to a single variable, i.e., to scalar test statistic:

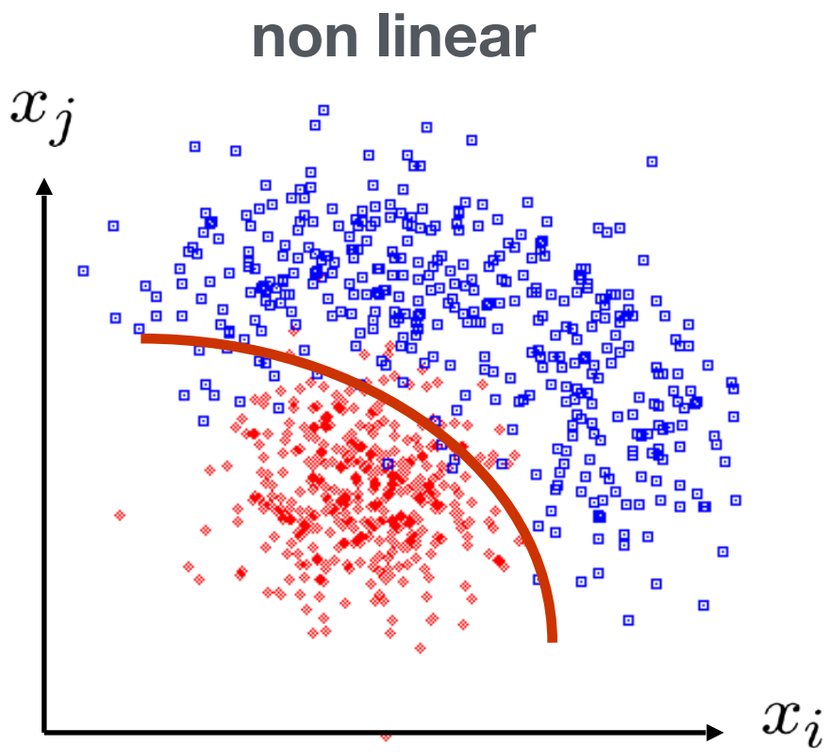
$$\mathbb{R}^n \to \mathbb{R}: \quad y(\vec{x})$$

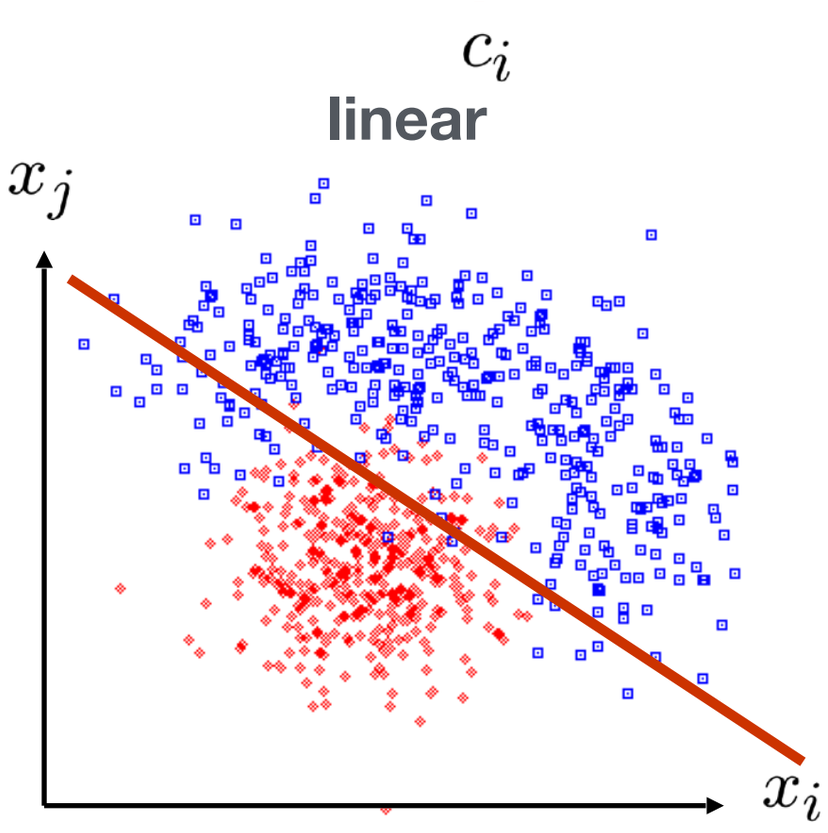
A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the x_i.

Problem closely related to *machine learning* (*pattern recognition*, *data mining*, …)

Classification: Different Approaches







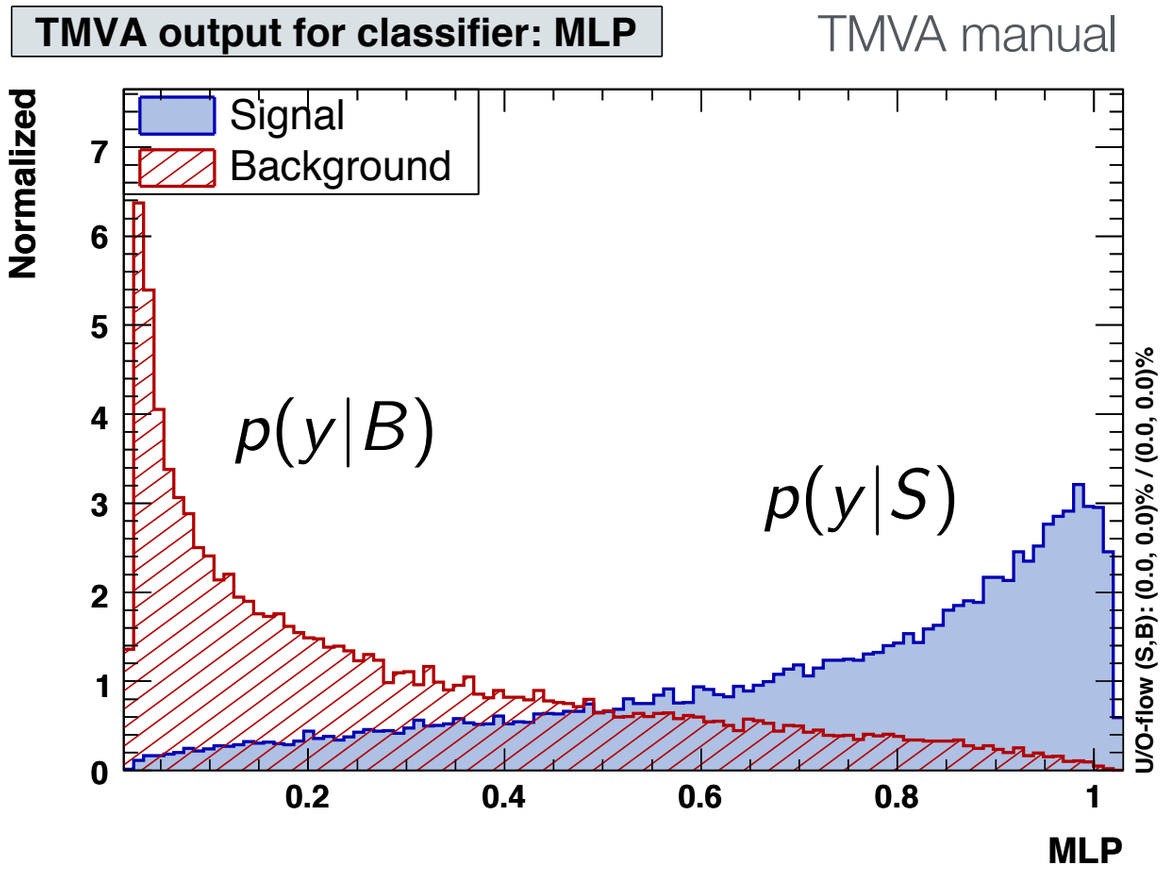
k-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines

…

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Signal Probability Instead of Hard Decisions

Example: test statistic y for signal and background from a Multi-Layer Perceptron (MLP):

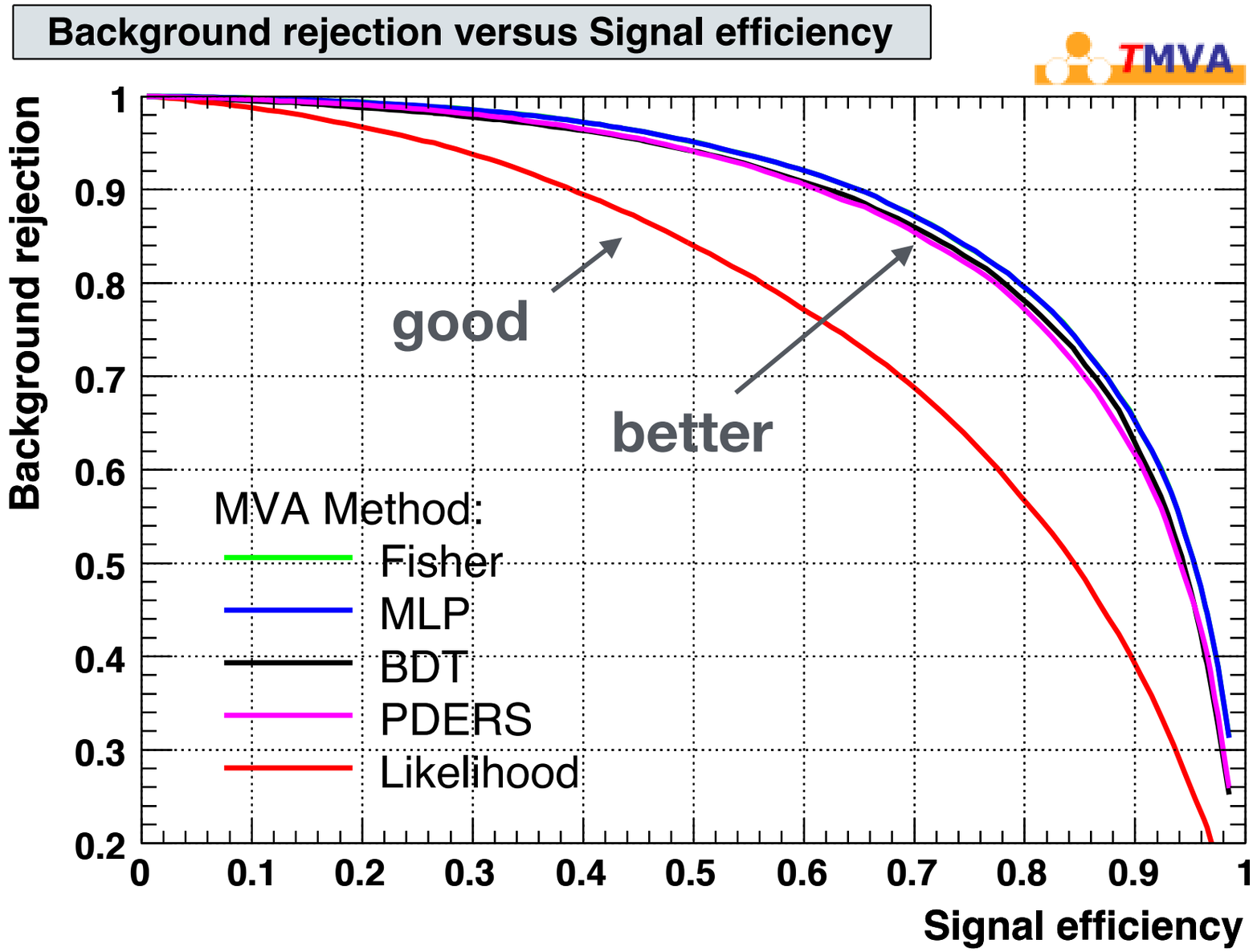


Instead of a hard yes/no decision one can also define the probability of an event to be a signal event:

$$P_s(y) \equiv P(S|y) = \frac{p(y|S) \cdot f_s}{p(y|S) \cdot f_s + p(y|B) \cdot (1 - f_s)}, \qquad f_s = \frac{n_s}{n_s + n_b}$$

ROC Curve

Quality of the classification can be characterized by the *receiver operating characteristic* (ROC curve)



$1 - \varepsilon_B$

ε_B: background efficiency

Different Approaches to Classification

Neyman-Pearson lemma states that likelihood ratio provides an optimal test statistic for classification:

$$y(\vec{x}) = \frac{p(\vec{x}|S)}{p(\vec{x}|B)}$$

Problem: the underlying pdf's are almost never known explicitly.

Two approaches:

1. Estimate signal and background pdf's and construct test statistic based on Neyman-Pearson lemma, e.g. Naïve Bayes classifier (= Likelihood classifier)

2. Decision boundaries determined directly without approximating the pdf's (linear discriminants, decision trees, neural networks, …)

Supervised Machine Learning (I)

Supervised Machine Learning requires *labeled training data*, i.e., a training sample where for each event it is known whether it is a signal or background event

- Decision boundary defined by minimizing a loss function ("training")

Bias-variance tradeoff

- Classifiers with a small number of degrees of freedom are less prone to statistical fluctuations: different training samples would result in a similar classification boundaries ("small variance")
- However, if the data contain features that a model with few degrees of freedom cannot describe, a *bias* is introduced. In this case a classifier with more degrees of freedom would be better.
- User has to find a good balance

Supervised Machine Learning (II)

Training, validation, and test sample

- Decision boundary fixed with *training sample*
- Performance on training sample becomes better with more iterations
- Danger of *overtraining*: Statistical fluctuations of the training sample will be learnt
- *Validation sample* = independent labeled data set not used for training
  → check for overtraining
- Sign of overtraining: performance on validation sample becomes worse
  → Stop training when signs of overtraining are observed ("early stopping")
- Performance: apply classifier to independent *test sample*
- Often: test sample = validation sample (only small bias)

Supervised Machine Learning (III)

Rule of thumb if training data not expensive

- Training sample: 50%
- Validation sample: 25%
- Test sample: 25%

often test sample = validation sample,
i.e., training : validation/test = 50:50

Cross validation (efficient use of scarce training data)

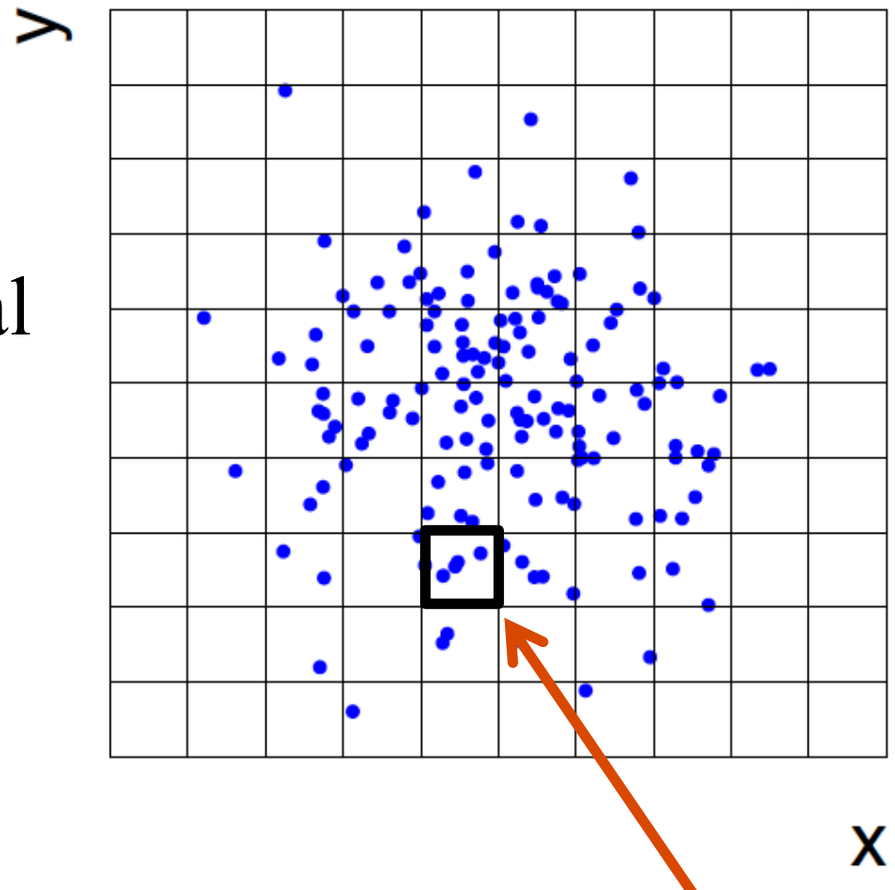
- Split training sample in k independent subset T_k of the full sample T
- Train on $T \setminus T_k$ resulting in k different classifiers
- For each training event there is one classifier that didn't use this event for training
- Validation results are then combined

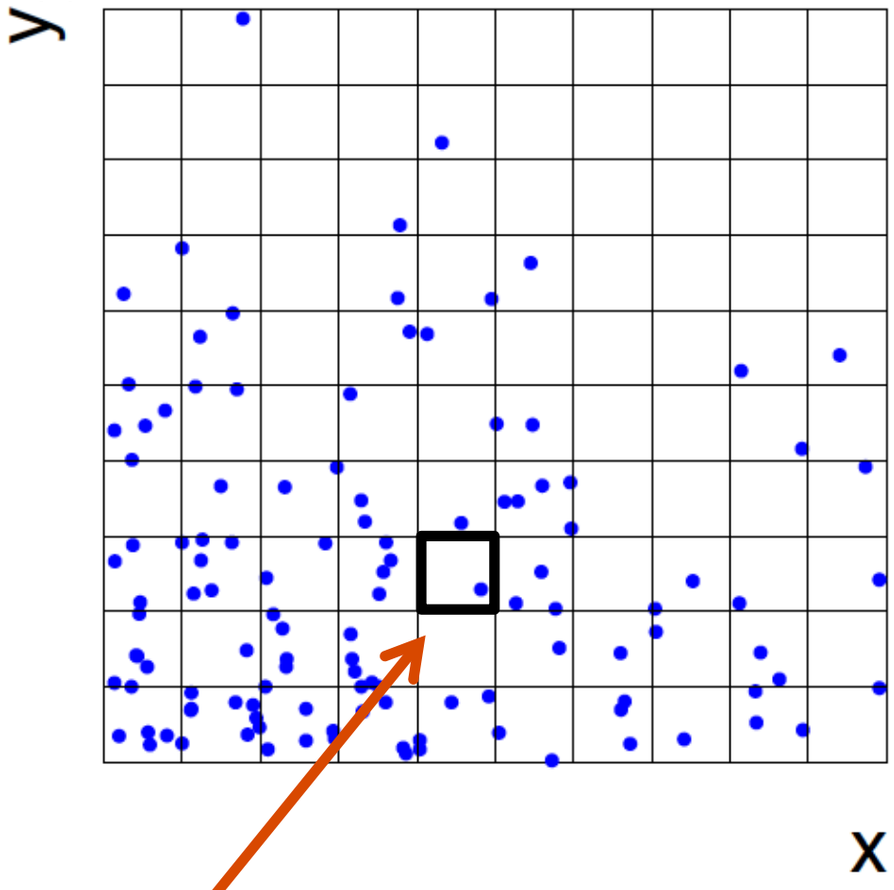
Estimating PDFs from Histograms?

Consider 2d example:

G. Cowan': https://www.pp.rhul.ac.uk/~cowan/stat_course.html

signal





back-
ground

approximate PDF by $N(x, y|S)$ and $N(x, y|B)$

M bins per variable in d dimensions: M^d cells

→ hard to generate enough training data (often not practical for $d > 1$)

In general in machine learning, problems related to a large number of dimensions of the feature space are referred to as the "curse of dimensionality"

k-Nearest Neighbor Method (I)

k-NN classifier

- Estimates probability density around the input vector
- $p(\vec{x}|S)$ and $p(\vec{x}|B)$ are approximated by the number of signal and background events in the training sample that lie in a small volume around the point $\vec{x}$

Algorithms finds k nearest neighbors:

$$k = k_s + k_b$$

Probability for the event to be of signal type:

$$p_s(\vec{x}) = \frac{k_s(\vec{x})}{k_s(\vec{x}) + k_b(\vec{x})}$$

k-Nearest Neighbor Method (II)

Simplest choice for distance measure in feature space is the Euclidean distance:

$$R = |\vec{x} - \vec{y}|$$

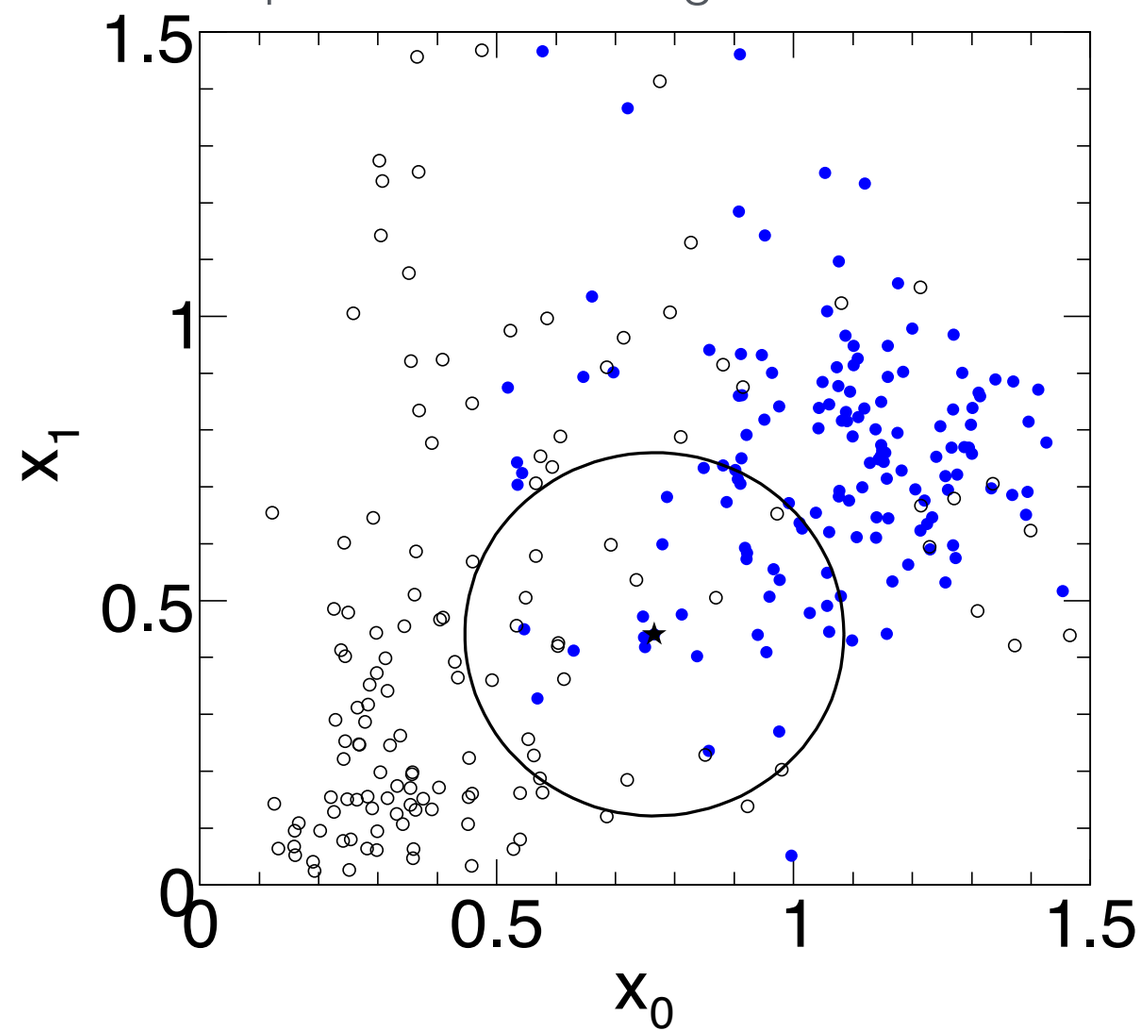
Better: take correlations between variables into account:

$$R = \sqrt{(\vec{x} - \vec{y})^T V^{-1} (\vec{x} - \vec{y})}$$

V = covariance matrix

"Mahalanobis distance"

TMVA manual
https://root.cern.ch/guides/tmva-manual



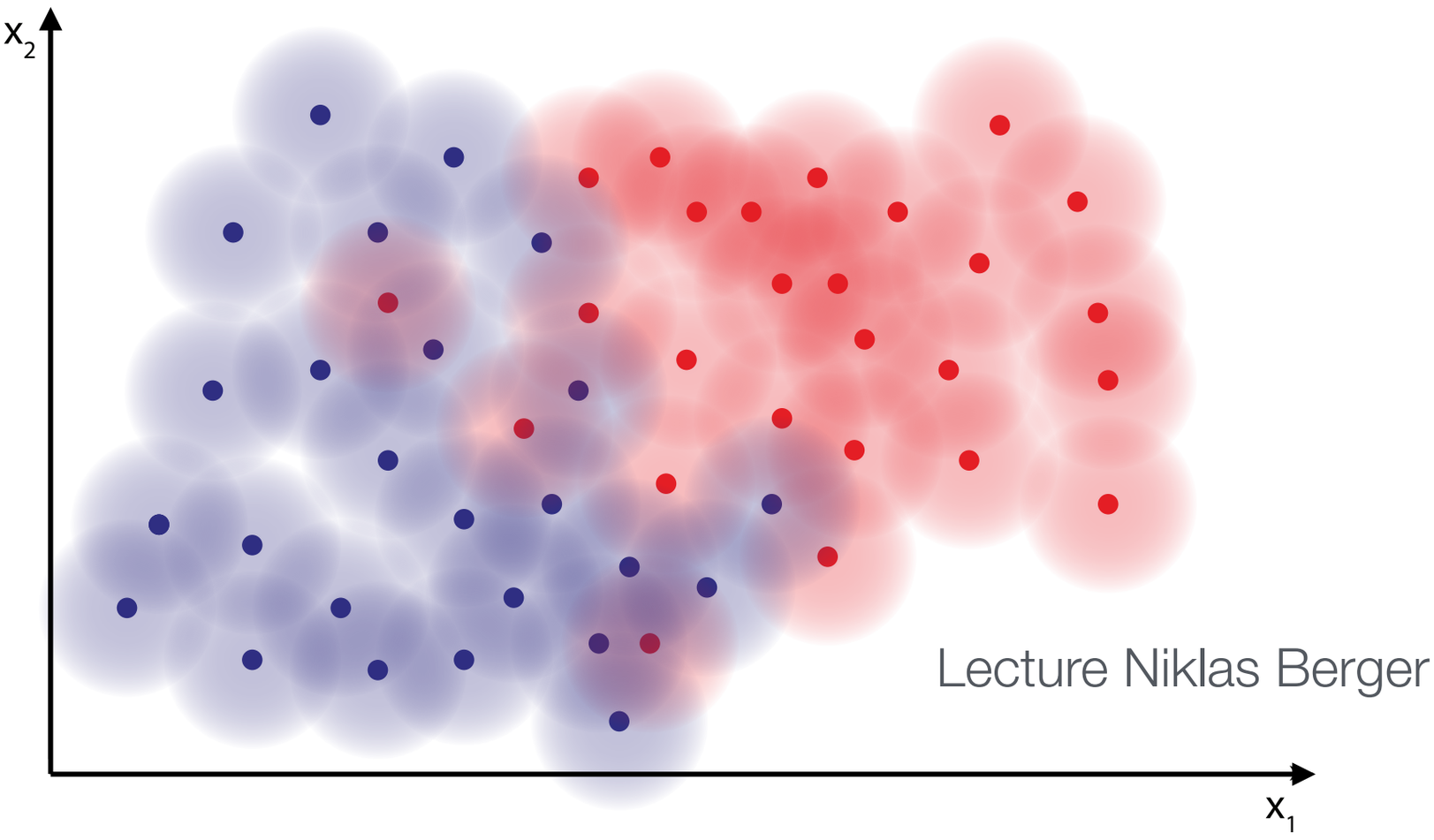
The k-NN classifier has best performance when the boundary that separates signal and background events has irregular features that cannot be easily approximated by parametric learning methods.

Kernel Density Estimator (KDE) [Just mentioned briefly here]

Idea: Smear training data to get an estimate of the PDFs

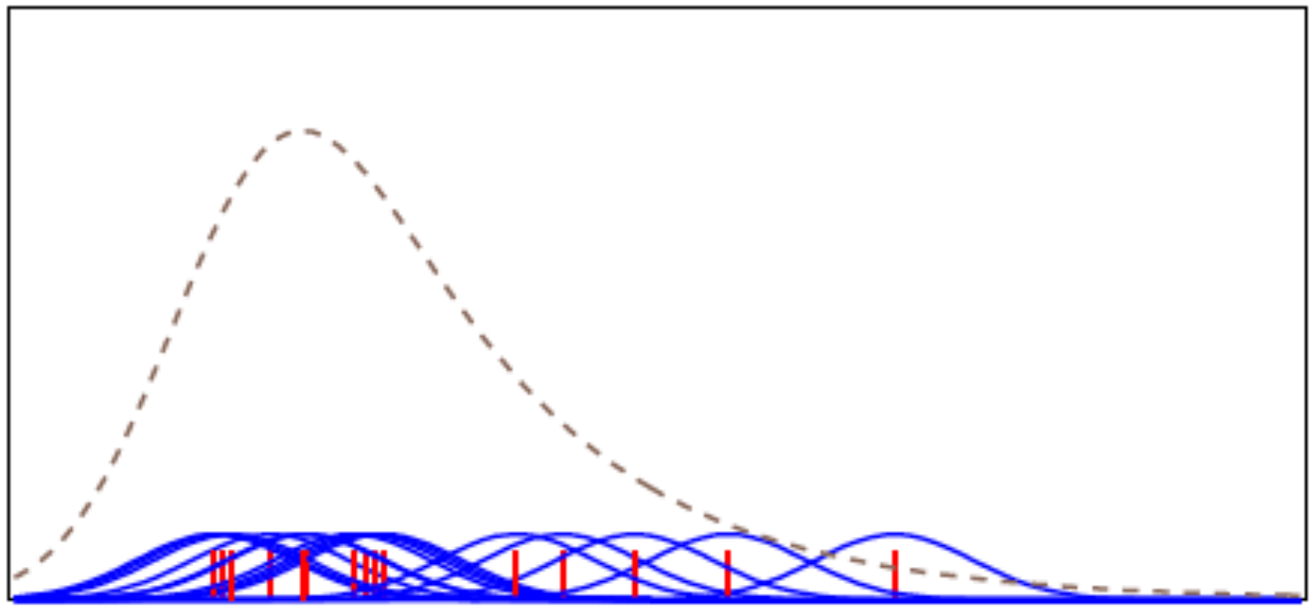
$$\hat{f}_h(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} K_h(\vec{x}-\vec{x}_i) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{\vec{x}-\vec{x}_i}{h}) \qquad K = \text{"Kernel"}$$
$$h = \text{"bandwidth" (smoothing parameter)}$$

Use, e.g., Gaussian kernel: $K(\vec{x}) = \frac{1}{(2\pi)^{d/2}} e^{-|\vec{x}|^2/2}$ d = dimension of feature space



Gaussian KDE in 1 Dimension

G. Cowan': https://www.pp.rhul.ac.uk/~cowan/stat_course.html



Adaptive KDE: adjust width of kernel according to PDF (wide where pdf is low)

Advantage of KDE: no training!

Disadvantage of KDE: need to sum of all training events to evaluate PDF, i.e., method can be slow

Fisher Linear Discriminant

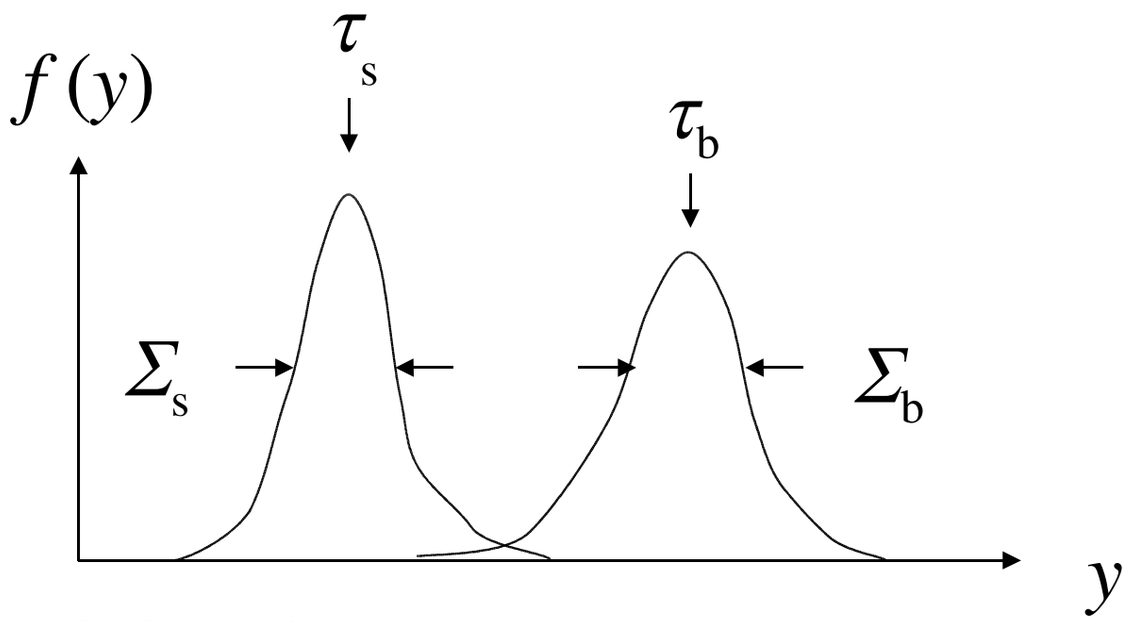
Linear discriminant is simple. Can still be optimal if amount of training data is limited.

Ansatz for test statistic: $$y(\vec{x}) = \sum_{i=1}^{n} w_i x_i = \vec{w}^{\mathsf{T}} \vec{x}$$

Choose parameters w_i so that separation between signal and background distribution is maximum.

Need to define "separation".

Fisher: maximize $$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher Linear Discriminant: Variable Definitions

Mean and covariance for signal and background:

$$\mu_i^{\mathrm{s,b}} = \int x_i \, f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x}$$

$$V_{ij}^{\mathrm{s,b}} = \int (x_i - \mu_i^{\mathrm{s,b}})(x_j - \mu_j^{\mathrm{s,b}}) \, f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x}$$

Mean and covariance of $y(\vec{x})$ for signal and background:

$$\tau_{\mathrm{s,b}} = \int y(\vec{x}) f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x} = \vec{w}^{\mathsf{T}} \vec{\mu}_{\mathrm{s,b}}$$

$$\Sigma_{\mathrm{s,b}}^2 = \int (y(\vec{x}) - \tau_{\mathrm{s,b}})^2 f(\vec{x}|H_{\mathrm{s,b}}) \, \mathrm{d}\vec{x} = \vec{w}^{\mathsf{T}} V_{\mathrm{s,b}} \vec{w}$$

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher Linear Discriminant: Determining the Coefficients w_i

Numerator of $J(\vec{w})$:

$$(\tau_\mathrm{s} - \tau_\mathrm{b})^2 = \left(\sum_{i=1}^{n} w_i (\mu_i^\mathrm{s} - \mu_i^\mathrm{b}) \right)^2 = \sum_{i,j=1}^{n} w_i w_j (\mu_i^\mathrm{s} - \mu_i^\mathrm{b})(\mu_j^\mathrm{s} - \mu_j^\mathrm{b})$$

$$\equiv \sum_{i,j=1}^{n} w_i w_j B_{ij} = \vec{w}^\mathsf{T} B \vec{w}$$

Denominator of $J(\vec{w})$:

$$\Sigma_\mathrm{s}^2 + \Sigma_\mathrm{b}^2 = \sum_{i,j=1}^{n} w_i w_j \, (V^\mathrm{s} + V^\mathrm{b})_{ij} \equiv \vec{w}^\mathsf{T} W \vec{w}$$

Maximize:

$$J(\vec{w}) = \frac{\vec{w}^\mathsf{T} B \vec{w}}{\vec{w}^\mathsf{T} W \vec{w}} = \frac{\text{separation between classes}}{\text{separation within classes}}$$

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Fisher Linear Discriminant: Determining the Coefficients w_i

Setting $\frac{\partial J}{\partial w_i} = 0$ gives:

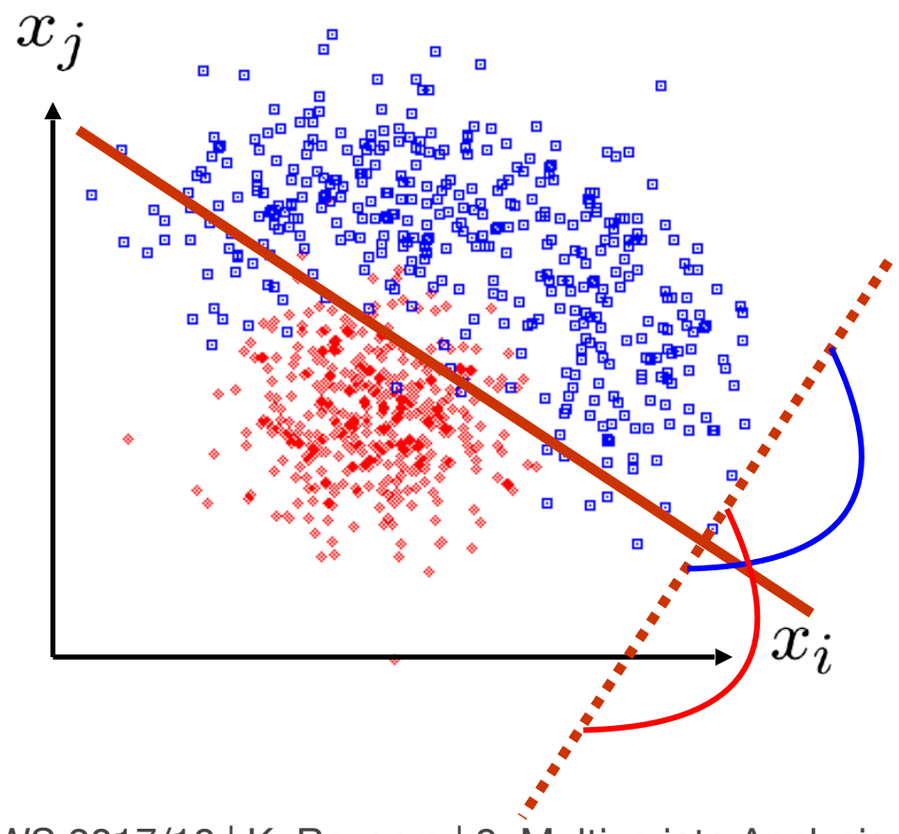
$$y(\vec{x}) = \vec{w}^{\mathsf{T}}\vec{x} \qquad \text{with} \quad \vec{w} \propto W^{-1}(\vec{\mu}_{\mathsf{s}} - \vec{\mu}_{\mathsf{b}})$$

We obtain linear decision boundaries.

Weight vector $\vec{w}$ can be interpreted as a direction in feature space on which the events are projected.

G. Cowan':
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

linear decision boundary



Fisher Linear Discriminant: Remarks

In case the signal and background pdfs $f(\vec{x}|H_s)$ and $f(\vec{x}|H_b)$ are both multivariate Gaussian with the same covariance but different means, the Fisher discriminant is

$$y(\vec{x}) \propto \ln \frac{f(\vec{x}|H_s)}{f(\vec{x}|H_b)}$$

That is, in this case the Fisher discriminant is an optimal classifier according to the Neyman-Pearson lemma (as $y(\vec{x})$ is a monotonic function of the likelihood ratio)

Test statistic can be written as

$$y(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i$$

where events with $y > 0$ are classified as signal. Same functional form as for the **perceptron** (prototype of neural networks).

Perceptron

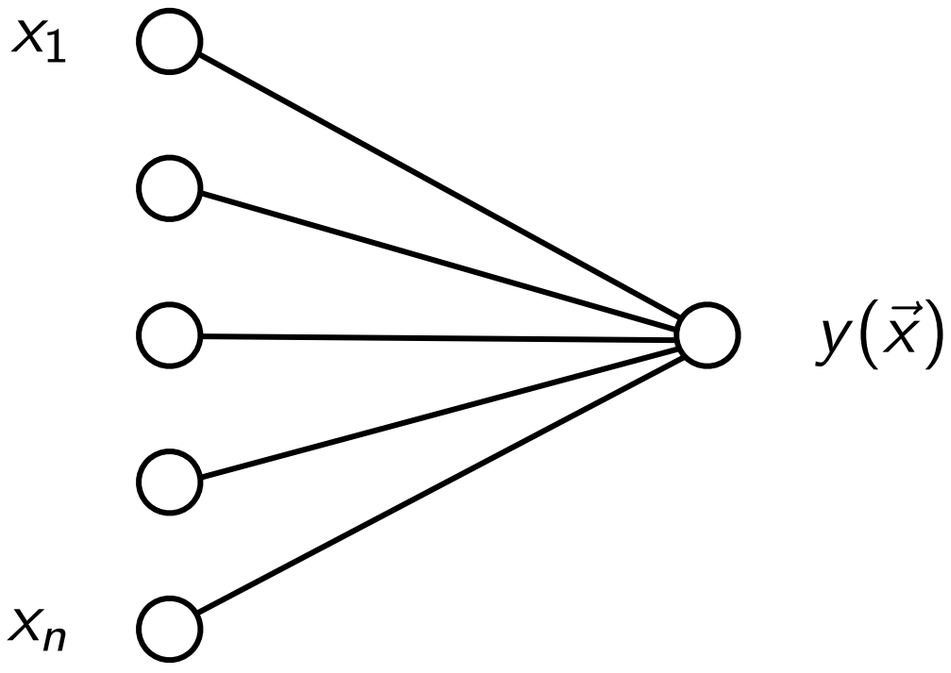
Discriminant:

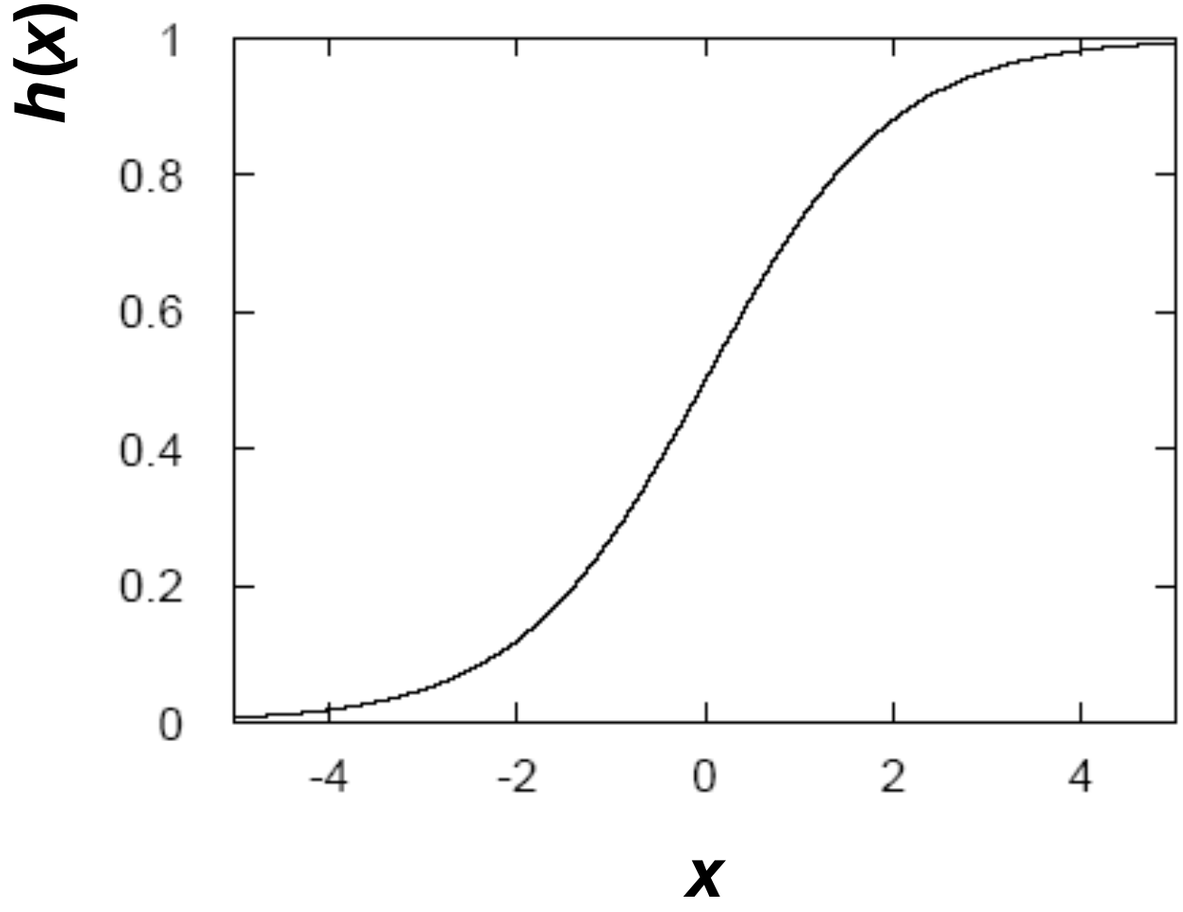
$$y(\vec{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

The nonlinear, monotonic function h is called *activation function*.

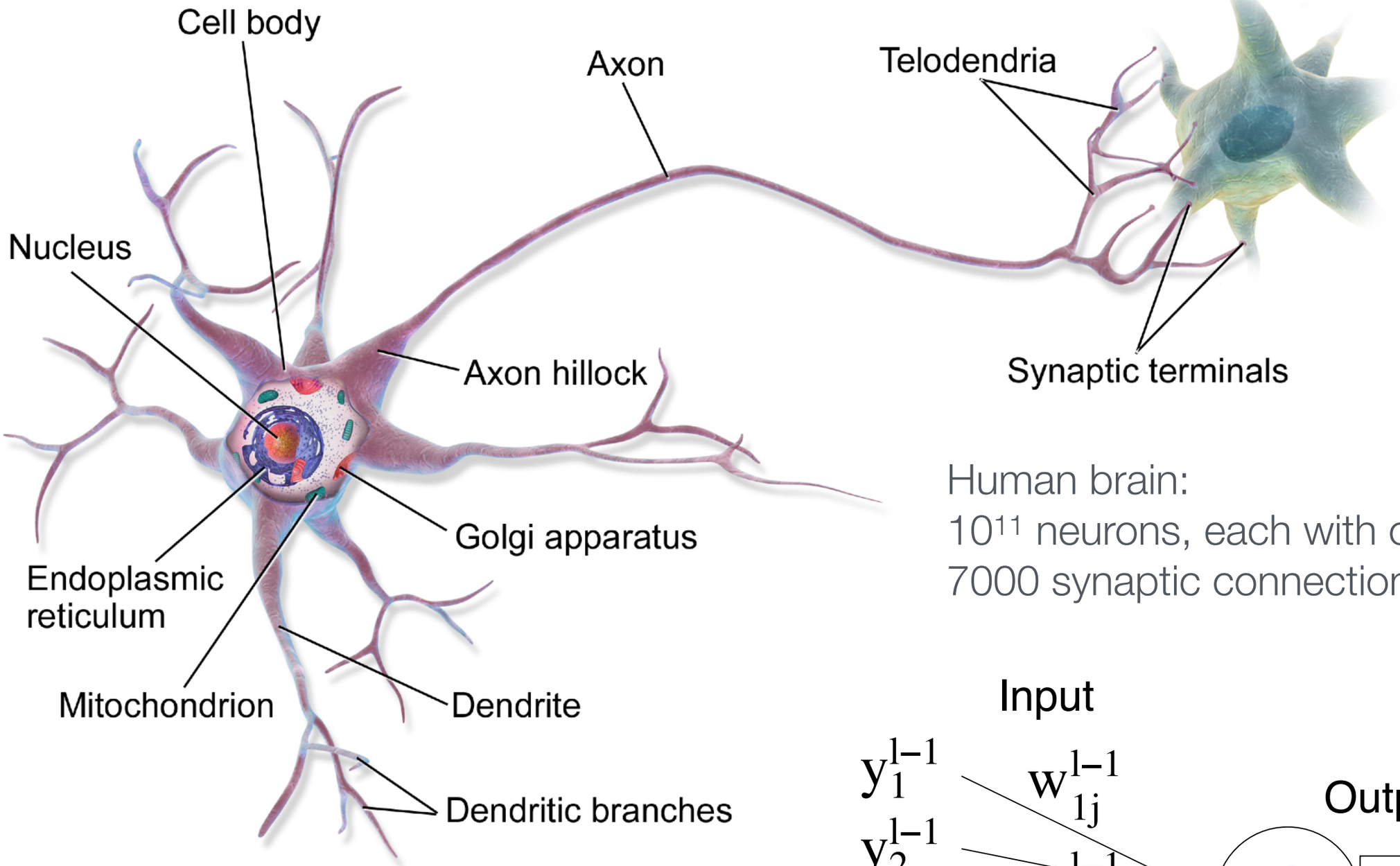
Typical choices for h:

$$\frac{1}{1+e^{-x}} \text{ (”sigmoid”)}, \quad \tanh x$$



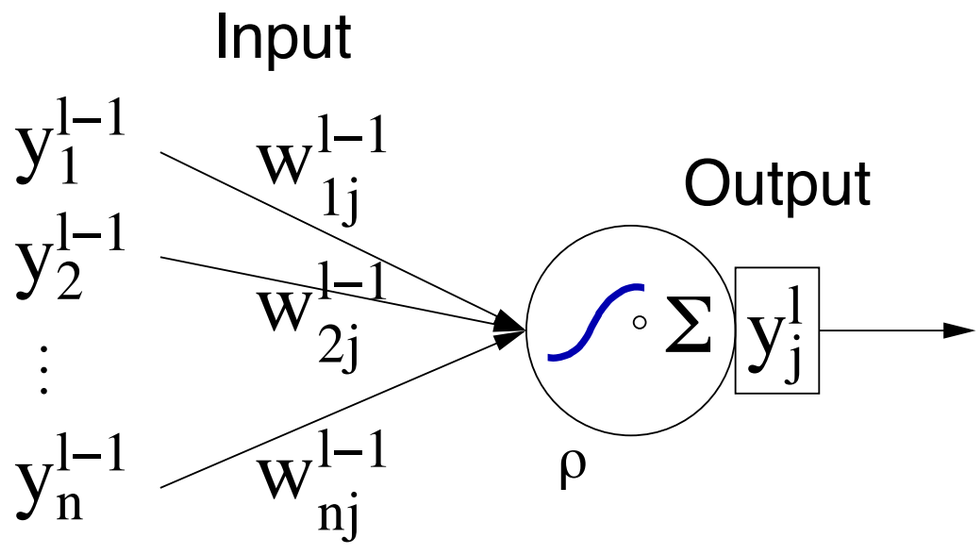


The Biological Inspiration: the Neuron

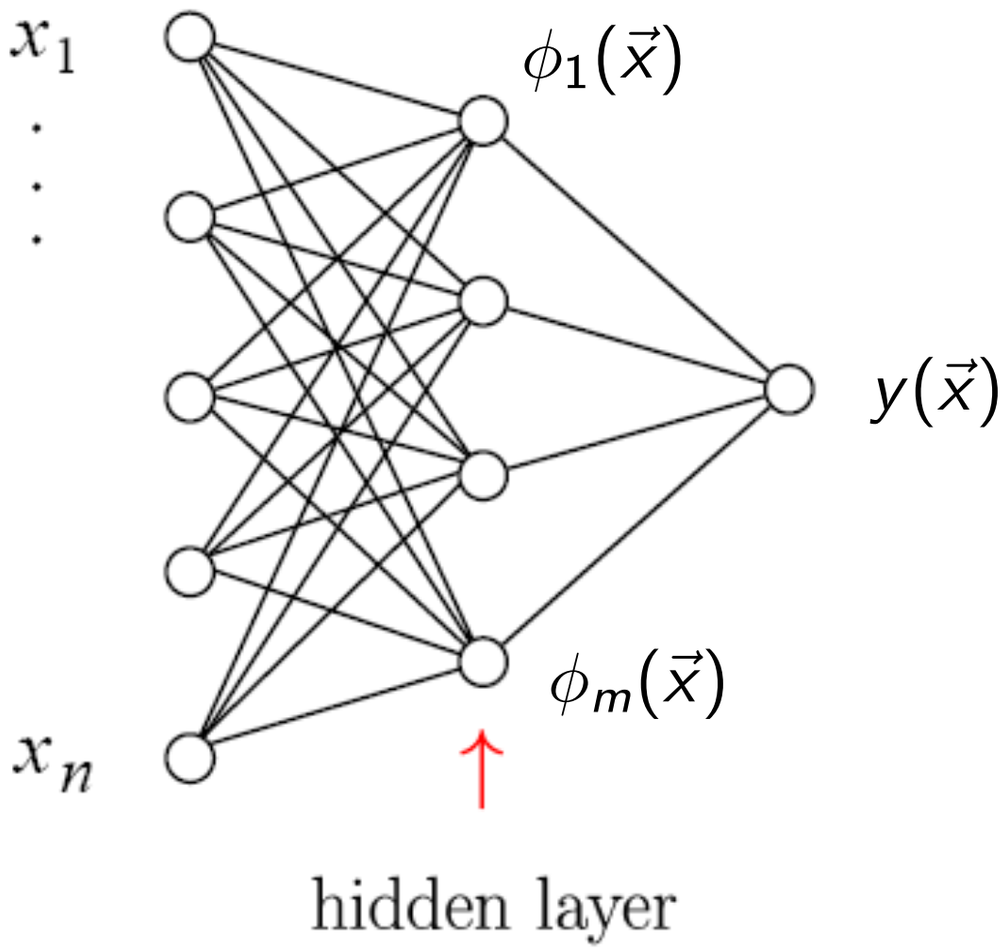


Human brain:
10^{11} neurons, each with on average 7000 synaptic connections

https://en.wikipedia.org/wiki/Neuron



Feedforward Neural Network with One Hidden Layer



superscripts indicates layer number

$$\phi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

$$y(\vec{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{m} w_{1j}^{(2)} \phi_j(\vec{x})\right)$$

Straightforward to generalize to multiple hidden layers

Network Training

$\vec{x}_a$: training event, $a = 1, ..., N$

t_a : correct label for training event a

e.g., t_a = 1, 0 for signal and background, respectively

$\vec{w}$: vector containing all weights

Error function:

$$E(\vec{w}) = \frac{1}{2}\sum_{a=1}^{N}(y(\vec{x}_a, \vec{w}) - t_a)^2 = \sum_{a=1}^{N} E_a(\vec{w})$$

Weights are determined by minimizing the error function.

Backpropagation

Start with an initial guess $\vec{w}^{(0)}$ for the weights an then update weights after each training event:

$$\vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} - \eta \nabla E_a(\vec{w}^{(\tau)})$$

η: learning rate

Let's write network output as follows:

$$y(\vec{x}) = h(u(\vec{x})) \;\text{ with }\; u(\vec{x}) = \sum_{j=0}^{m} w_{1j}^{(2)} \phi_j(\vec{x}),\; \phi_j(\vec{x}) = h\left(\sum_{k=0}^{n} w_{jk}^{(1)} x_k\right) \equiv h\left(v_j(\vec{x})\right)$$

Here we defined $\phi_0 = x_0 = 1$ and the sums start from 0 to include the offsets.

Weights from hidden layer to output:

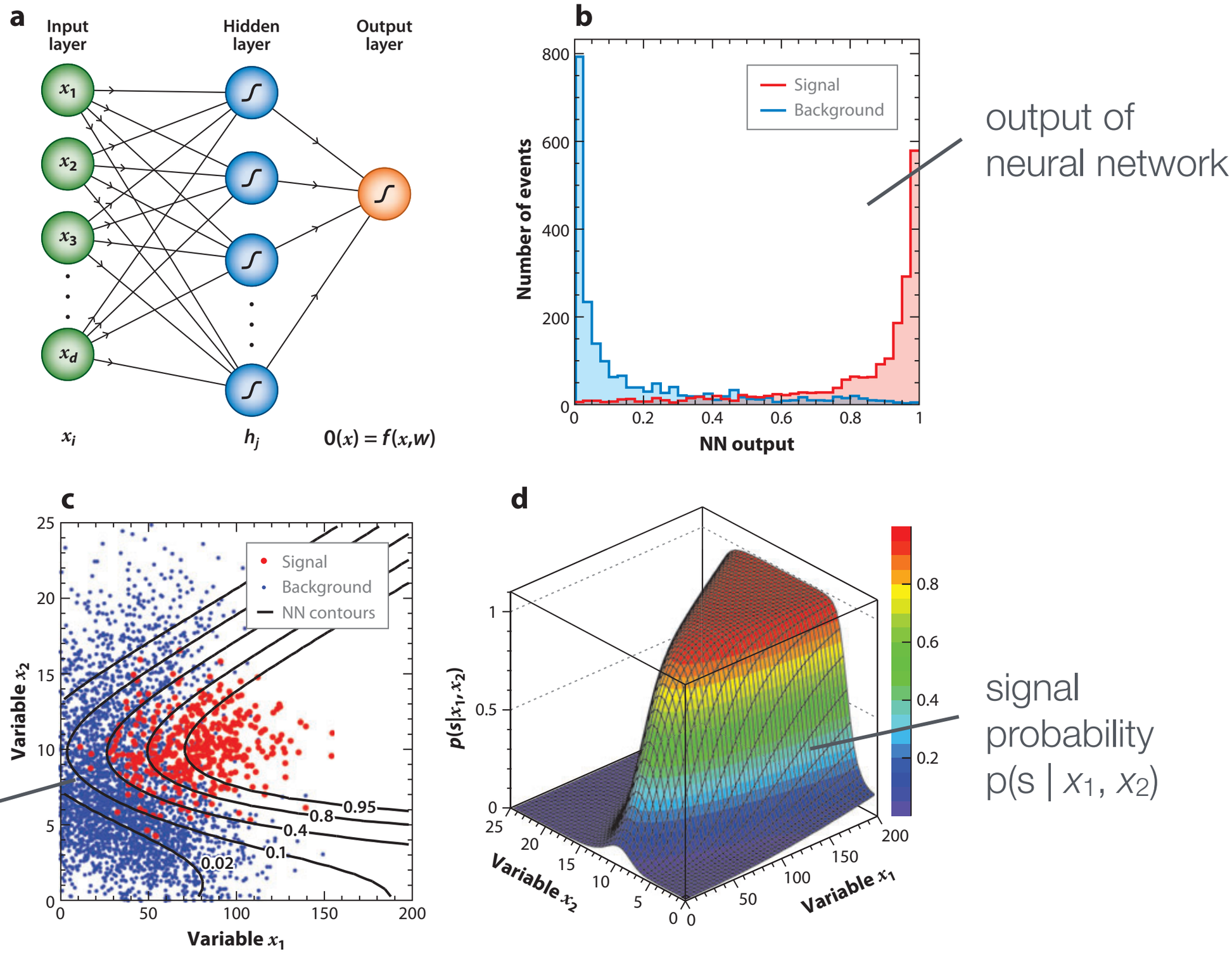
$$E_a = \frac{1}{2}(y_a - t_a)^2 \;\rightarrow\; \frac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a) h'(u(\vec{x}_a)) \frac{\partial u}{\partial w_{1j}^{(2)}} = (y_a - t_a) h'(u(\vec{x}_a)) \phi_j(\vec{x}_a)$$

Weights from input layer to hidden layer (→ further application of chain rule):

$$\frac{\partial E_a}{\partial w_{jk}^{(1)}} = (y_a - t_a) h'(u(\vec{x}_a)) w_{1j}^{(2)} h'\left(v_j(\vec{x}_a)\right) x_{a,k} \qquad \vec{x}_a \equiv (x_{a,1}, \ldots, x_{a,n})$$

Neural Network Output and Decision Boundaries

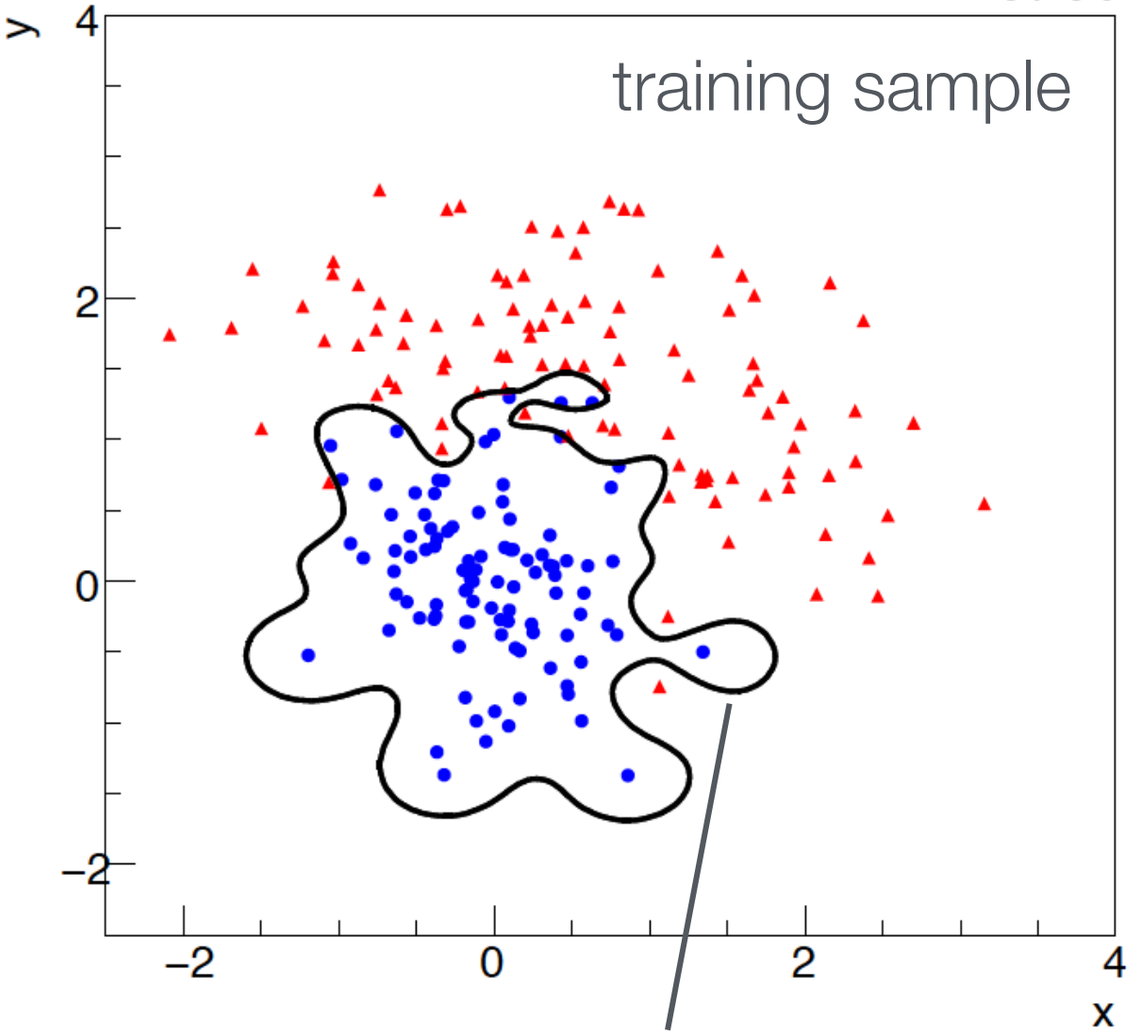
P. Bhat, Multivariate Analysis Methods in Particle Physics, inspirehep.net/record/879273

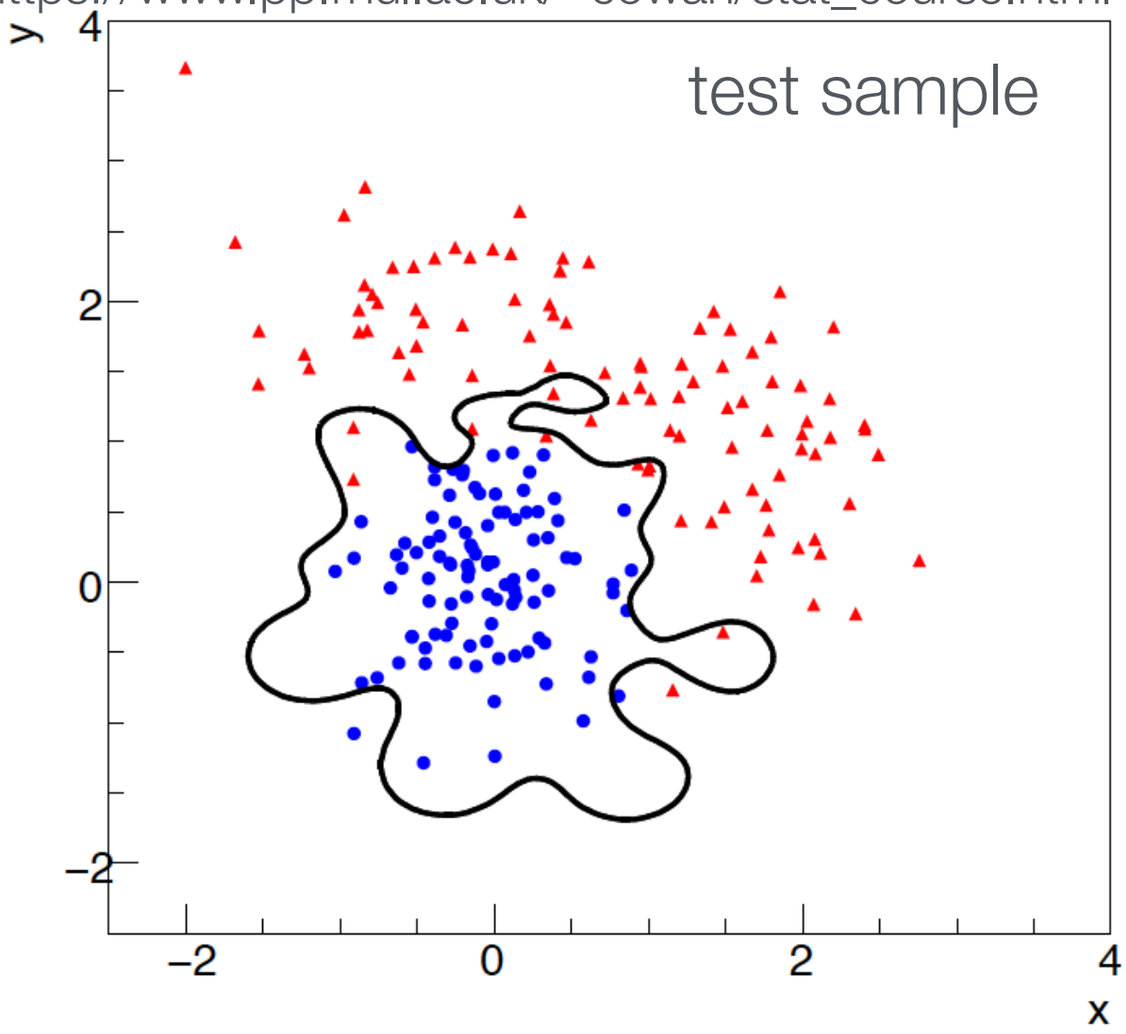


Example of Overtraining

Too many neurons/layers make a neural network too flexible
→ overtraining

G. Cowan: https://www.pp.rhul.ac.uk/~cowan/stat_course.html



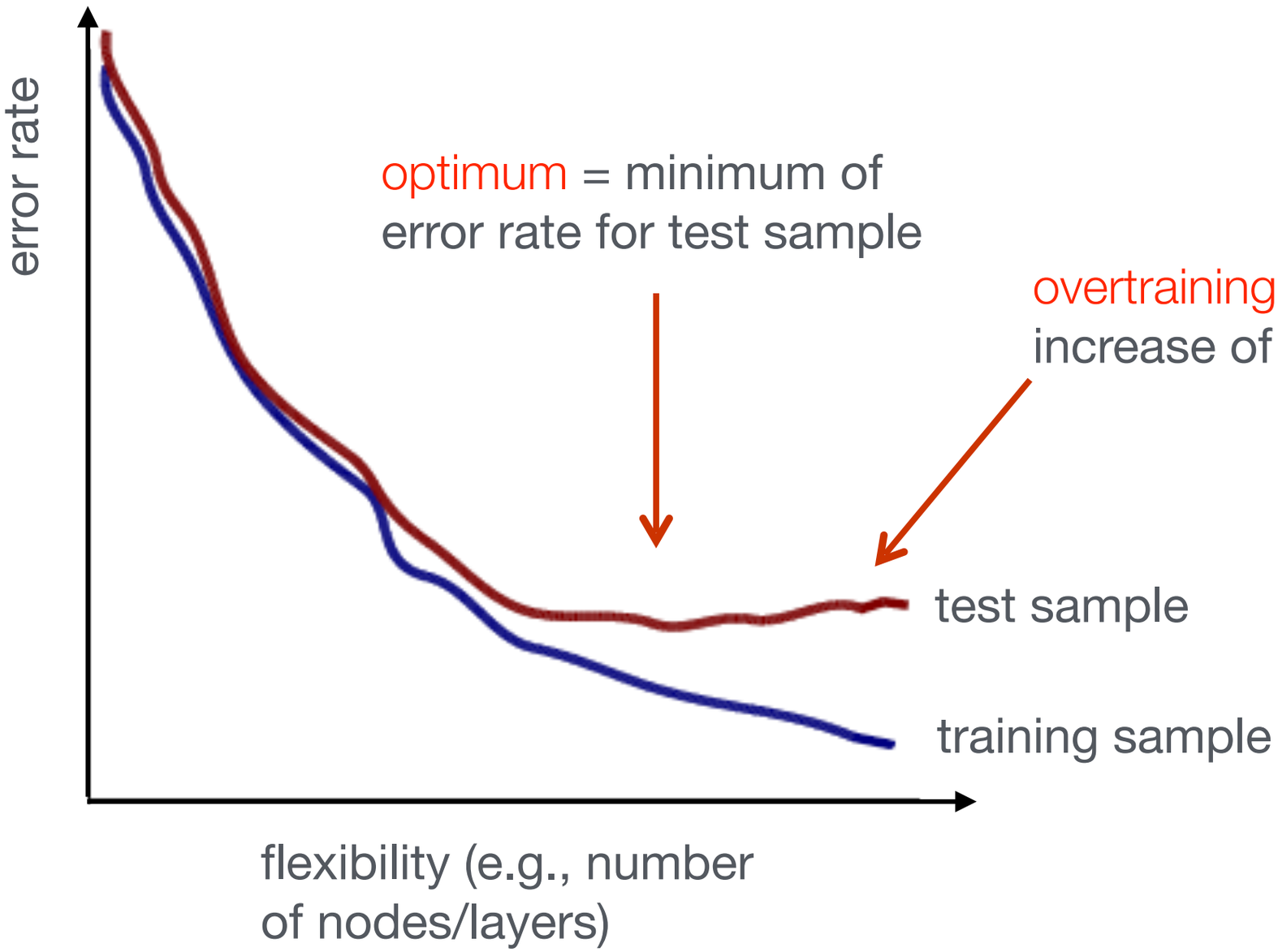


Network "learns" features that are merely statistical fluctuations in the training sample

Monitoring Overtraining

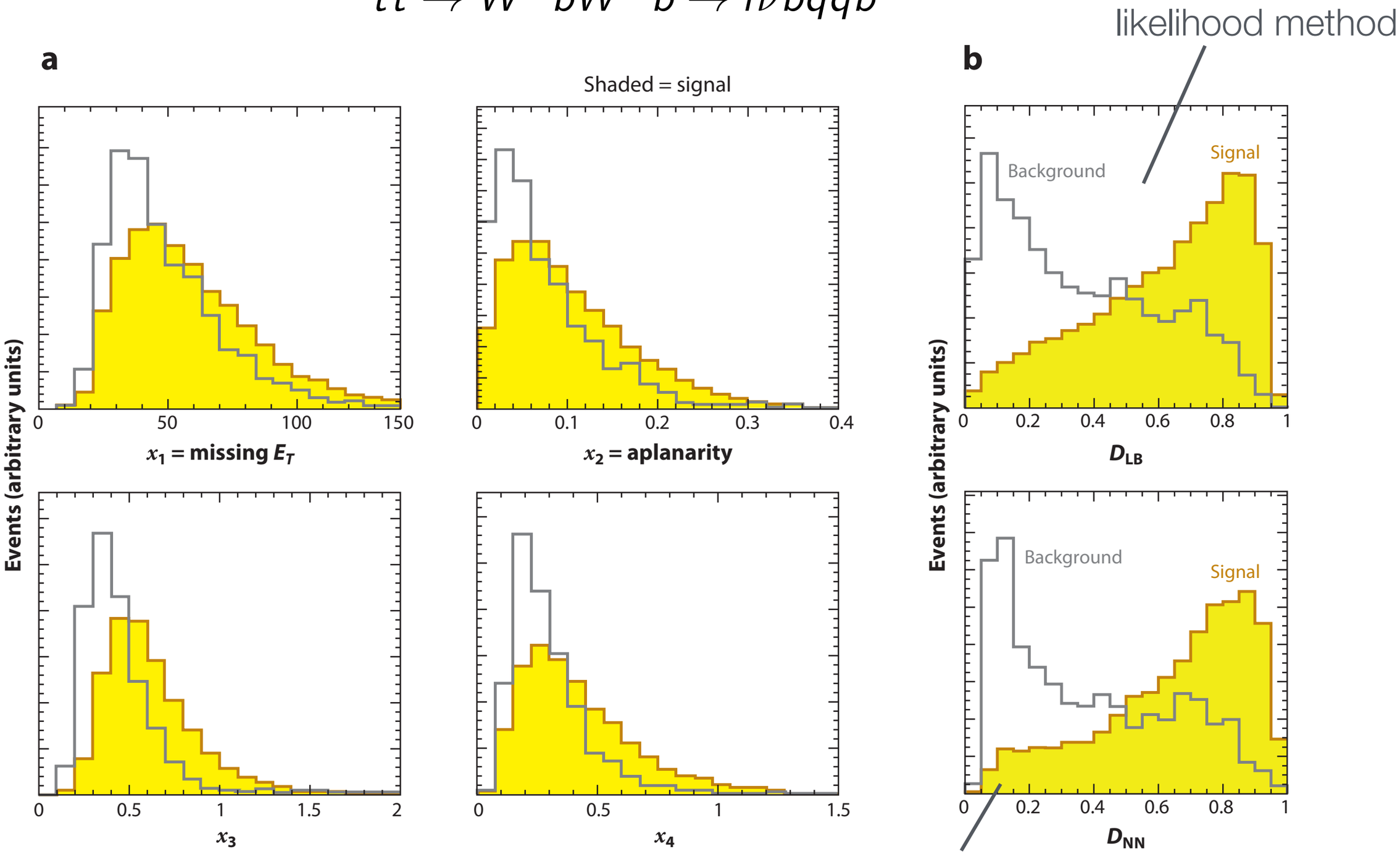
G. Cowan:
https://www.pp.rhul.ac.uk/~cowan/stat_course.html

Monitor fraction of misclassified events (or error function:)



Example: Identification of Events with Top-Quarks

$$t\bar{t} \rightarrow W^+ b W^- \bar{b} \rightarrow l\nu b q\bar{q}\bar{b}$$



D0 experiment,
plot from inspirehep.net/record/879273

Deep Neural Networks

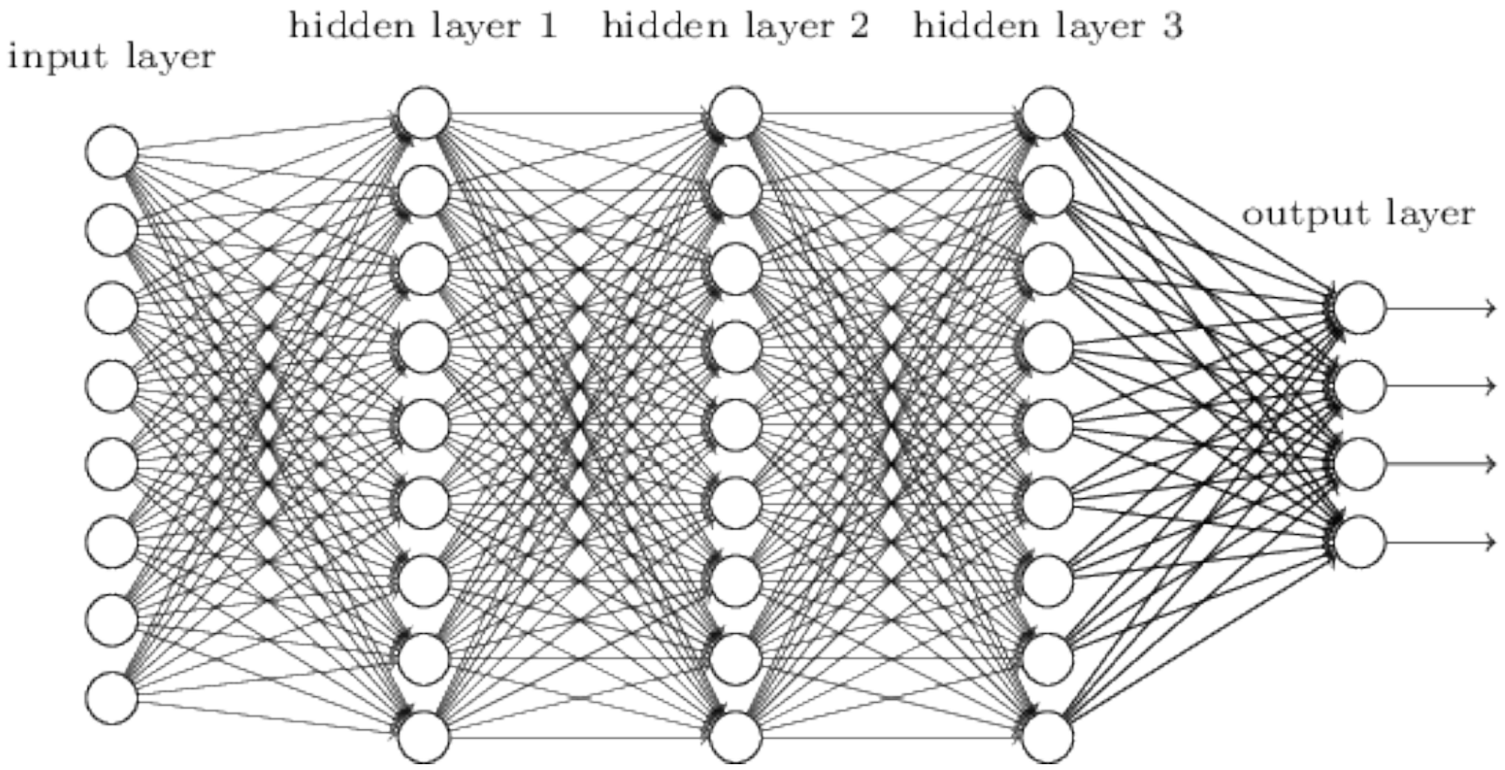
Deep networks: many hidden layers with large number of neurons

Challenges

- Hard too train ("vanishing gradient problem")
- Training slow
- Risk of overtraining

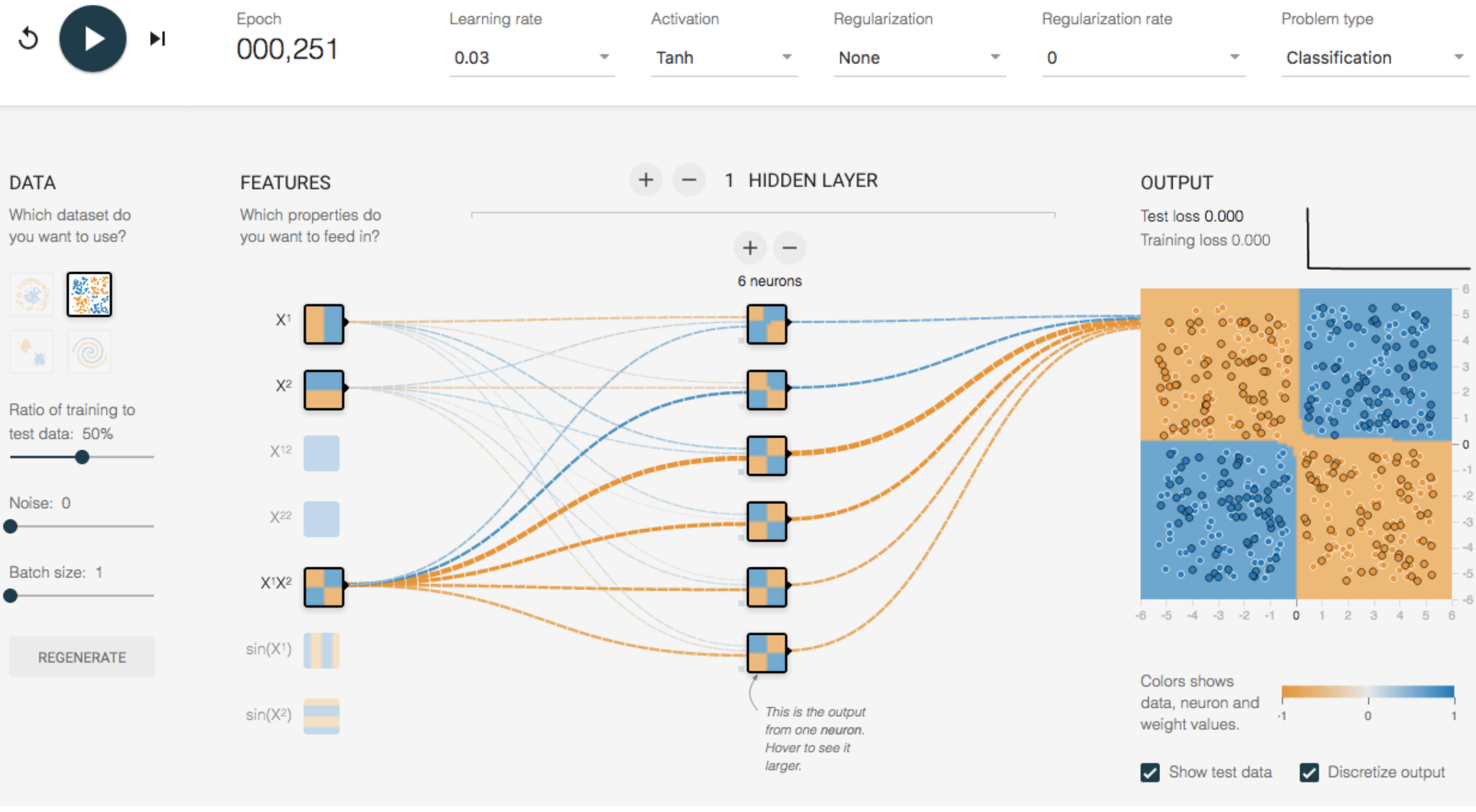
Big progress in recent years

- Interest in NN waned before ca. 2006
- Milestone: paper by G. Hinton (2006): "learning for deep belief nets"
- Image recognition, AlphaGo, …
- Soon: self-driving cars, …



http://neuralnetworksanddeeplearning.com

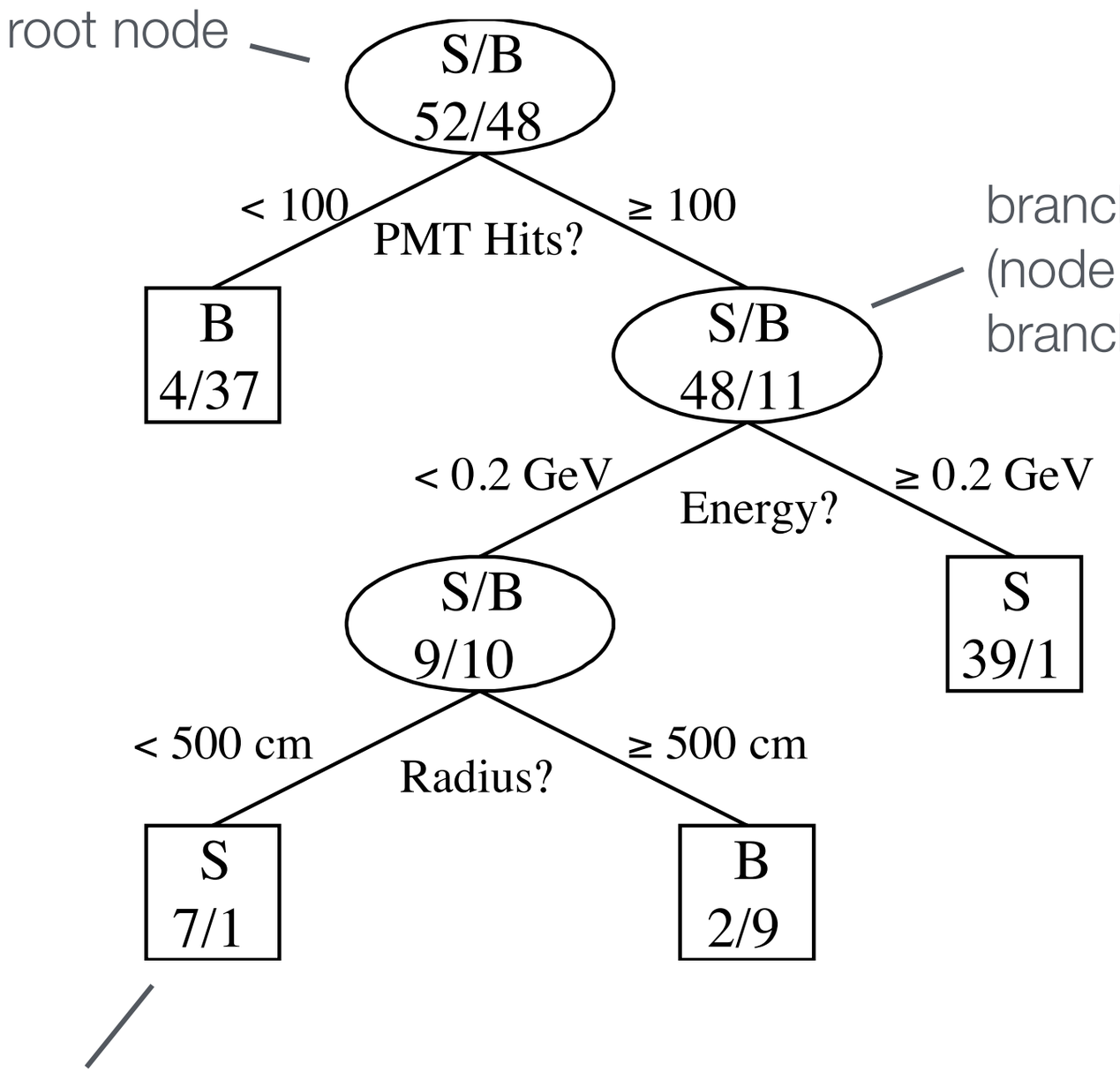
Fun with Neural Nets in the Browser

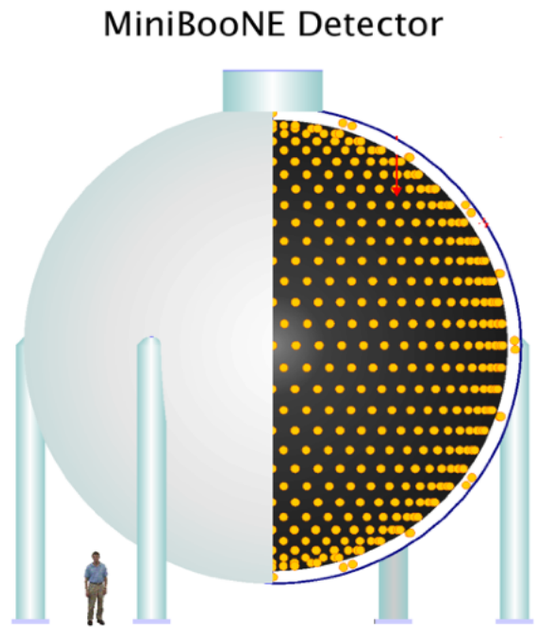


http://playground.tensorflow.org

Decision Trees (I)

arXiv:physics/0508045v1



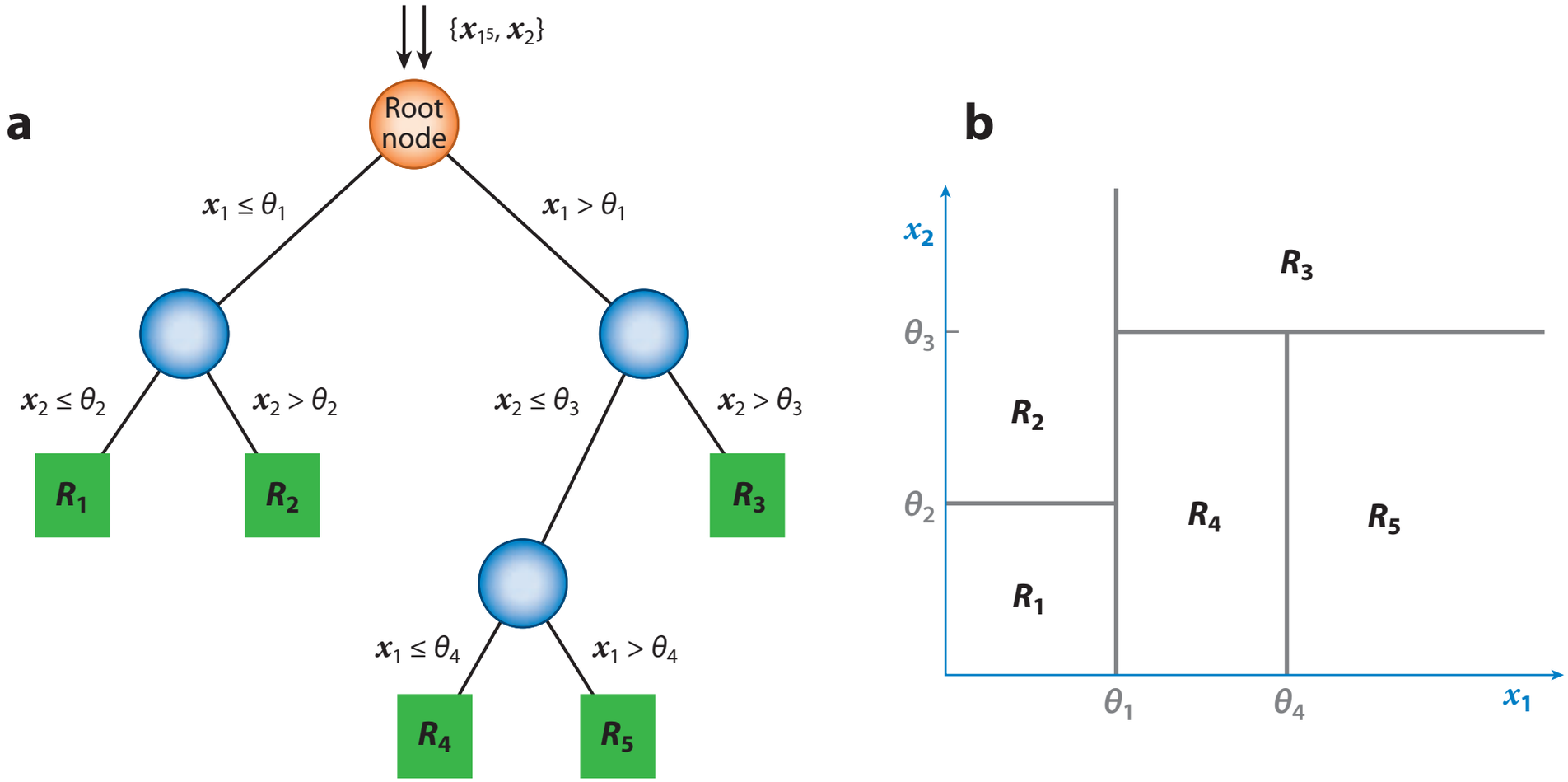


MiniBooNE: 1520 photomultiplier signals, goal: separation of ν_e from ν_μ events

Leaf nodes classify events as either signal or background

Decision Trees (II)

Ann.Rev.Nucl.Part.Sci. 61 (2011) 281-309



Easy to interpret and visualize:
Space of feature vectors split up into rectangular volumes (attributed to either signal or background)

How to build a decision tree in an optimal way?

Finding Optimal Cuts

Separation btw. signal and background is often measured with the *Gini index*:

$$G = p(1-p)$$

Here p is the purity:

$$p = \frac{\sum_{\text{signal}} w_i}{\sum_{\text{signal}} w_i + \sum_{\text{background}} w_i}$$

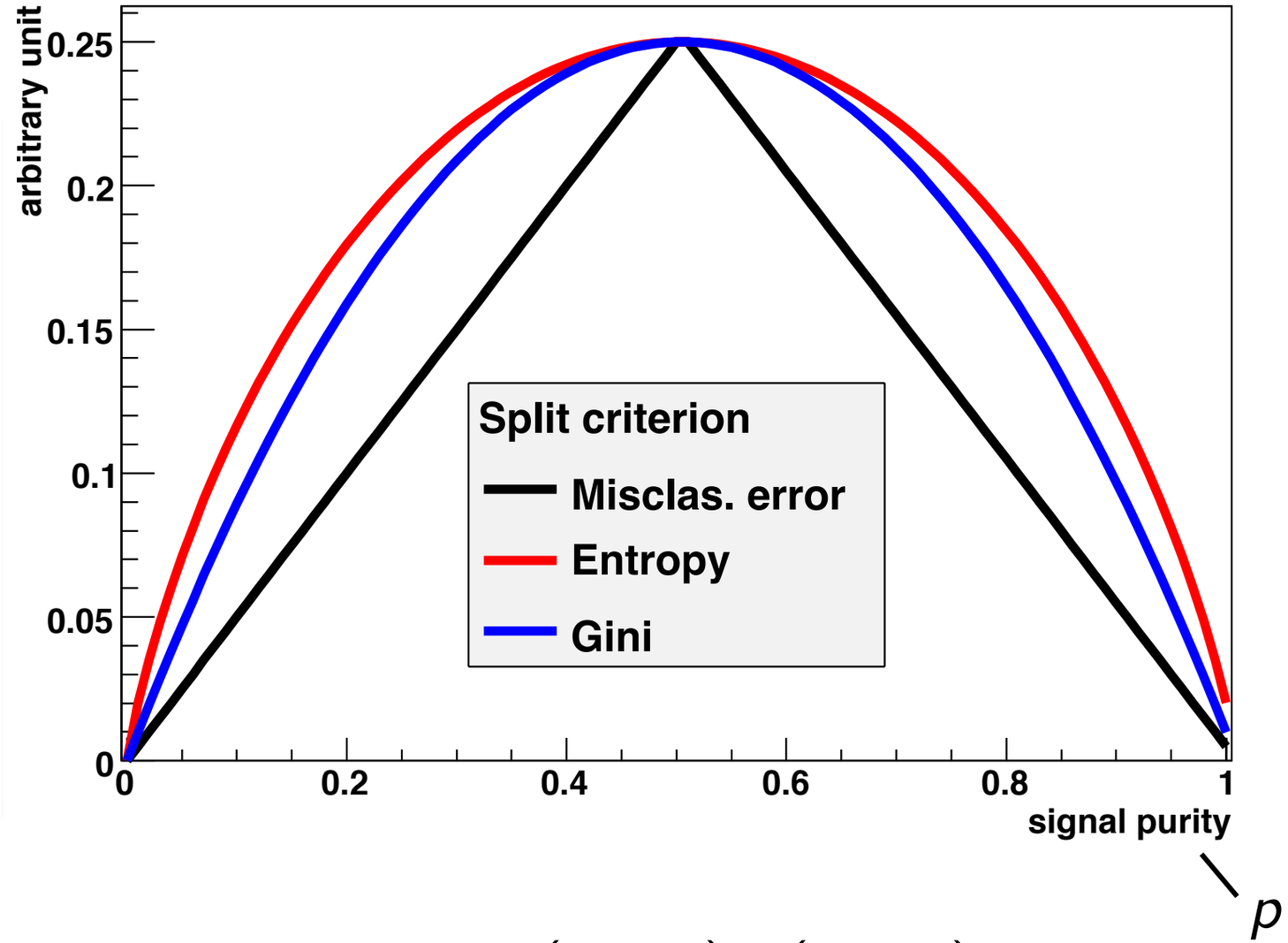
w_i = weight of event i

[usefulness of weights will become apparent soon]

Improvement in signal/background separation after splitting a set A into two sets B and C:

$$\Delta = W_A G_A - W_B G_B - W_C G_C \quad \text{where} \quad W_X = \sum_X w_i$$

Separation Measures



Cross entropy: $-p \ln p - (1-p) \ln(1-p)$

Gini index: $p(1-p)$ [after Corrado Gini, used to measure income and wealth inequalities, 1912]

Misclassification rate: $1 - \max(p, 1-p)$

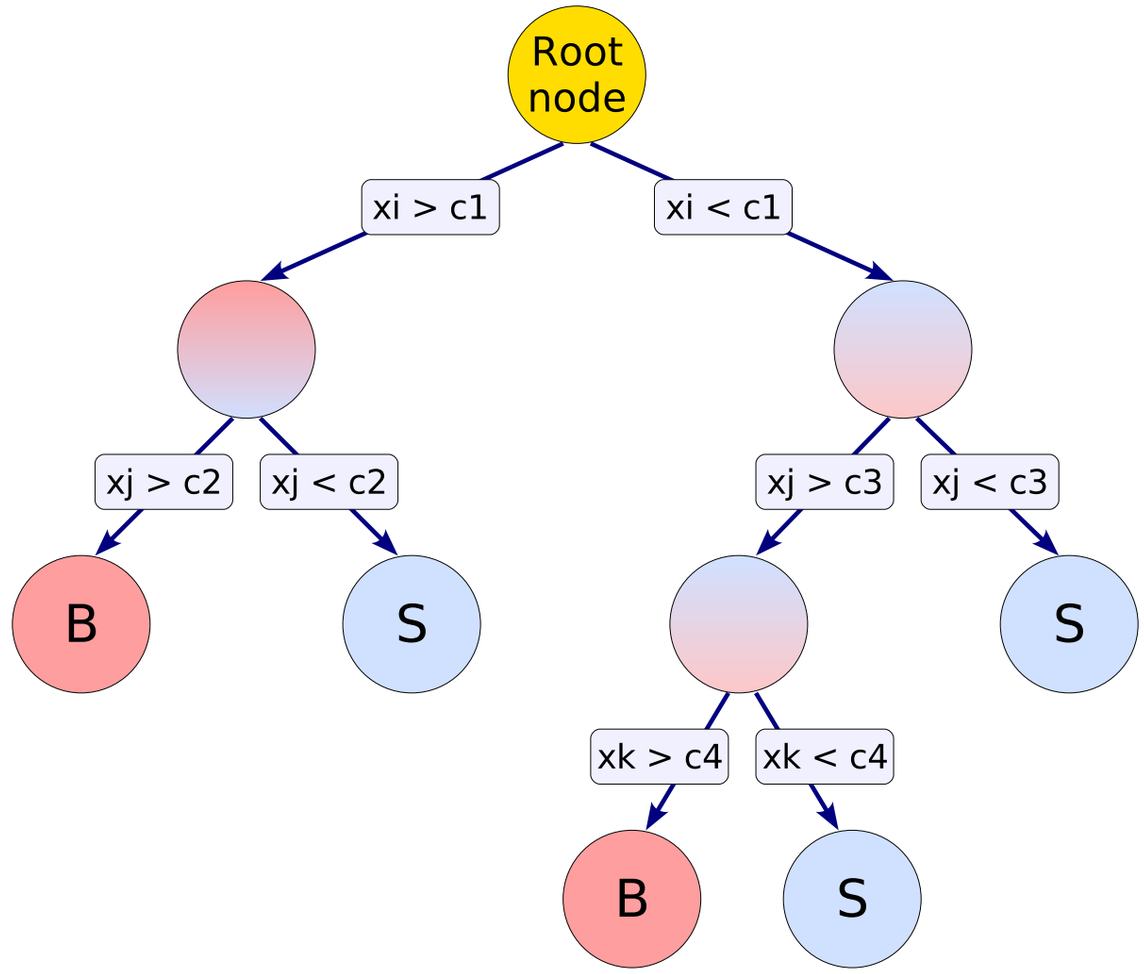
Decision Tree Pruning

When to stop growing a tree?

- When all nodes are essentially pure?
- Well, that's overfitting!

Pruning

- Cut back fully grown tree to avoid overtraining



Boosted Decision Trees: Idea

Drawback of decisions trees:
very sensitive to statistical fluctuations in training sample

Solution: boosting

- One tree → several trees ("*forrest*")
- Trees are derived from the same training ensemble by reweighting events
- Individual trees are then combined: weighted average of individual trees

Boosting is a general method of combining a set of classifiers (not necessarily decisions trees) into a new, more stable classifier with smaller error.

Popular example: AdaBoost (Freund, Schapire, 1997)

AdaBoost (short for *Adaptive Boosting*)

Initial training sample

$\vec{x}_1, ..., \vec{x}_n$:	multivariate event data
$y_1, ..., y_n$:	true class labels, $+1$ or -1
$w_1^{(1)}, ..., w_n^{(1)}$	event weights

with equal weights normalized as

$$\sum_{i=1}^{n} w_i^{(1)} = 1$$

Train first classifier f_1:

$f_1(\vec{x}_i) > 0$	classify as signal
$f_1(\vec{x}_i) < 0$	classify as background

Updating Events Weights

Define training sample k+1 from training sample k by updating weights:

$$w_i^{(k+1)} = w_i^{(k)} \frac{e^{-\alpha_k f_k(\vec{x}_i) y_i / 2}}{Z_k}$$

i = event index

normalization factor so that $\sum_{i=1}^{n} w_i^{(k)} = 1$

Weight is increased if event was misclassified by the previous classifier

→ "Next classifier should pay more attention to misclassified events"

At each step the classifier f_k minimizes error rate

$$\varepsilon_k = \sum_{i=1}^{n} w_i^{(k)} I(y_i f_k(\vec{x}_i) \leq 0), \quad I(X) = 1 \text{ if } X \text{ is true, 0 otherwise}$$

Assigning the Classifier Score

Assign score to each classifier according to its error rate:

$$\alpha_k = \ln \frac{1 - \varepsilon_k}{\varepsilon_k}$$

Combined classifier (weighted average):

$$f(\vec{x}) = \sum_{k=1}^{K} \alpha_k f_k(\vec{x})$$

It can be shown that the error rate of the combined classifier satisfies

$$\varepsilon \leq \prod_{k=1}^{K} 2\sqrt{\varepsilon_k(1 - \varepsilon_k)}$$

General Remarks on Multi-Variate Analyses

MVA Methods

- More effective than classic cut-based analyses
- Take correlations of input variables into account

Important: find good input variables for MVA methods

- Good separation power between S and B
- Little correlations among variables
- No correlation with the parameters you try to measure in your signal sample!

Pre-processing

- Apply obvious variable transformations and let MVA method do the rest
- Make use of obvious symmetries: if e.g. a particle production process is symmetric in polar angle θ use |cos θ| and not cos θ as input variable
- It is generally useful to bring all input variables to a similar numerical range

H. Voss, Multivariate Data Analysis and Machine Learning in High Energy Physics
http://tmva.sourceforge.net/talks.shtml

Classifiers and Their Properties

H. Voss, Multivariate Data Analysis and Machine Learning in High Energy Physics
http://tmva.sourceforge.net/talks.shtml

Criteria		Classifiers								
		Cuts	Likeli-hood	PDERS / k-NN	H-Matrix	Fisher	MLP	BDT	RuleFit	SVM
Perfor-mance	no / linear correlations	😐	🙂	🙂	😐	🙂	🙂	😐	🙂	🙂
	nonlinear correlations	😐	☹	🙂	☹	☹	🙂	🙂	😐	🙂
Speed	Training	☹	🙂	🙂	🙂	🙂	😐	☹	😐	☹
	Response	🙂	🙂	☹/😐	🙂	🙂	🙂	😐	😐	😐
Robust-ness	Overtraining	🙂	😐	😐	🙂	🙂	☹	☹	😐	😐
	Weak input variables	🙂	🙂	☹	🙂	🙂	😐	😐	😐	😐
Curse of dimensionality		☹	🙂	☹	🙂	🙂	😐	🙂	😐	😐
Transparency		🙂	🙂	😐	🙂	🙂	☹	☹	☹	☹