Seminar on
Advanced Analysis Methods for Heavy-Ion Data

Introduction: Multivariate Analysis and Machine Learning

Klaus Reygers

3 May 2019

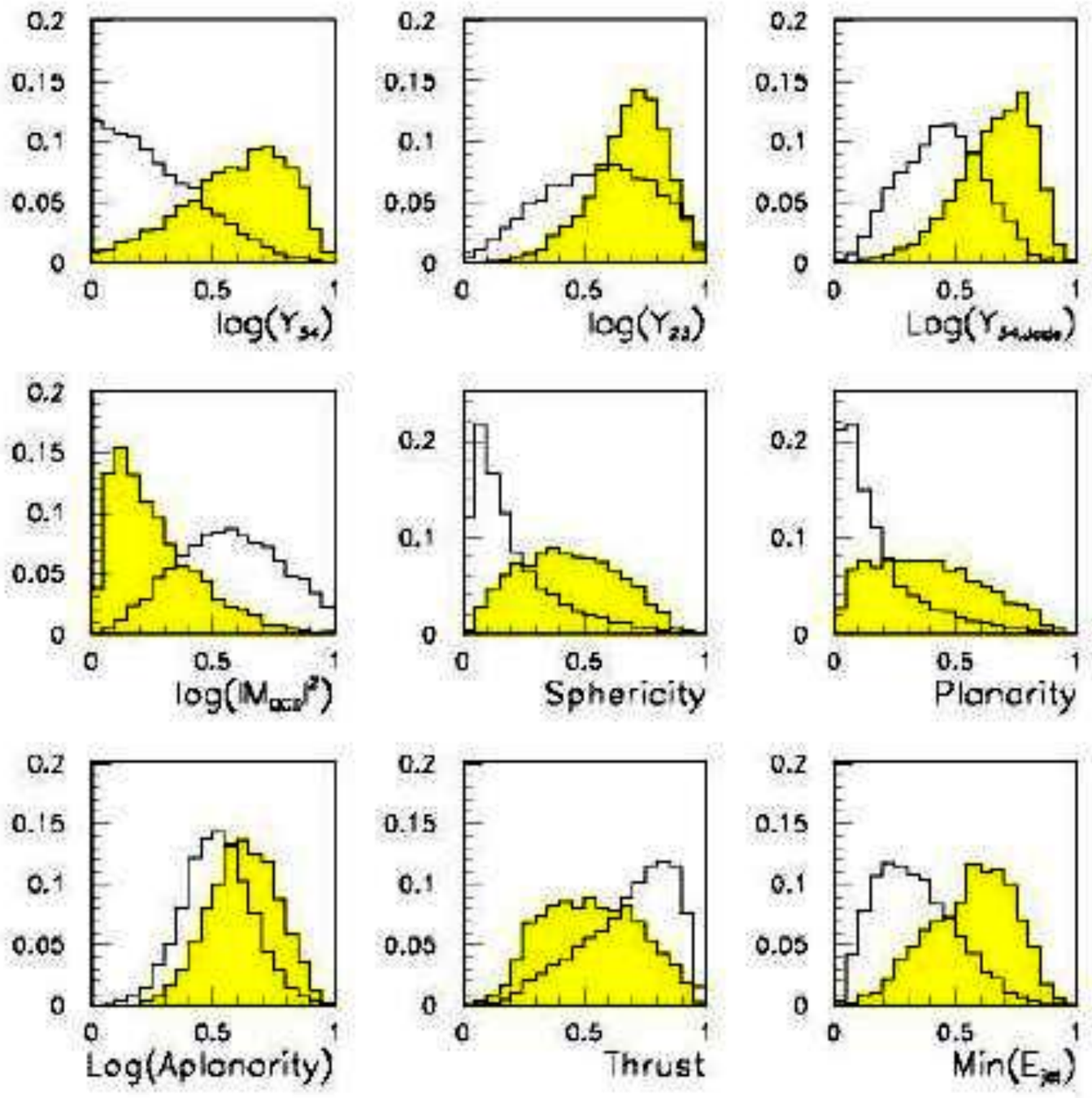
Physikalisches Institut
Heidelberg University

Contents

1. Quick tour of methods and applications
2. A selection of methods for multivariate classification
3. More on neural networks

Multivariate Analysis: An Early Example from Particle Physics

G. Cowan, Lecture on Statistical data analysis



(Garrido, Juste and Martinez, ALEPH 96-144)

Signal: $e^+e^- \rightarrow W^+W^-$

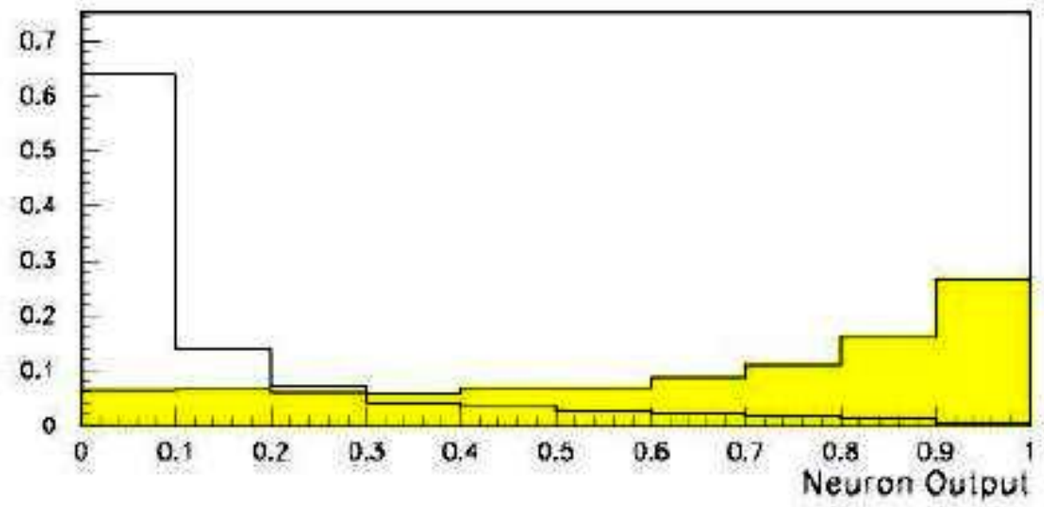
often 4 well separated hadron jets

Background: e+e– → qqgg

4 less well separated hadron jets

← input variables based on jet structure, event shape, ... none by itself gives much separation.

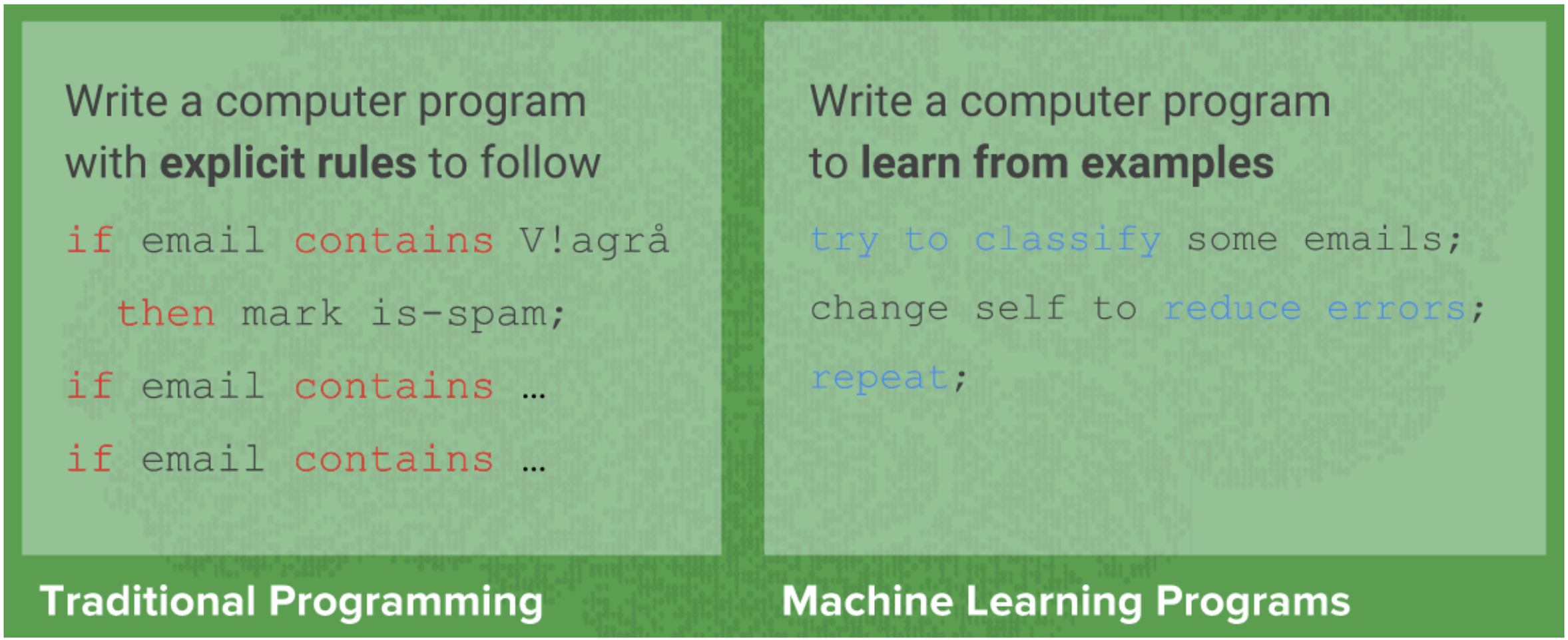
Neural network output:



Machine Learning

"Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed" – Wikipedia

J. Mayes, Machine learning 101

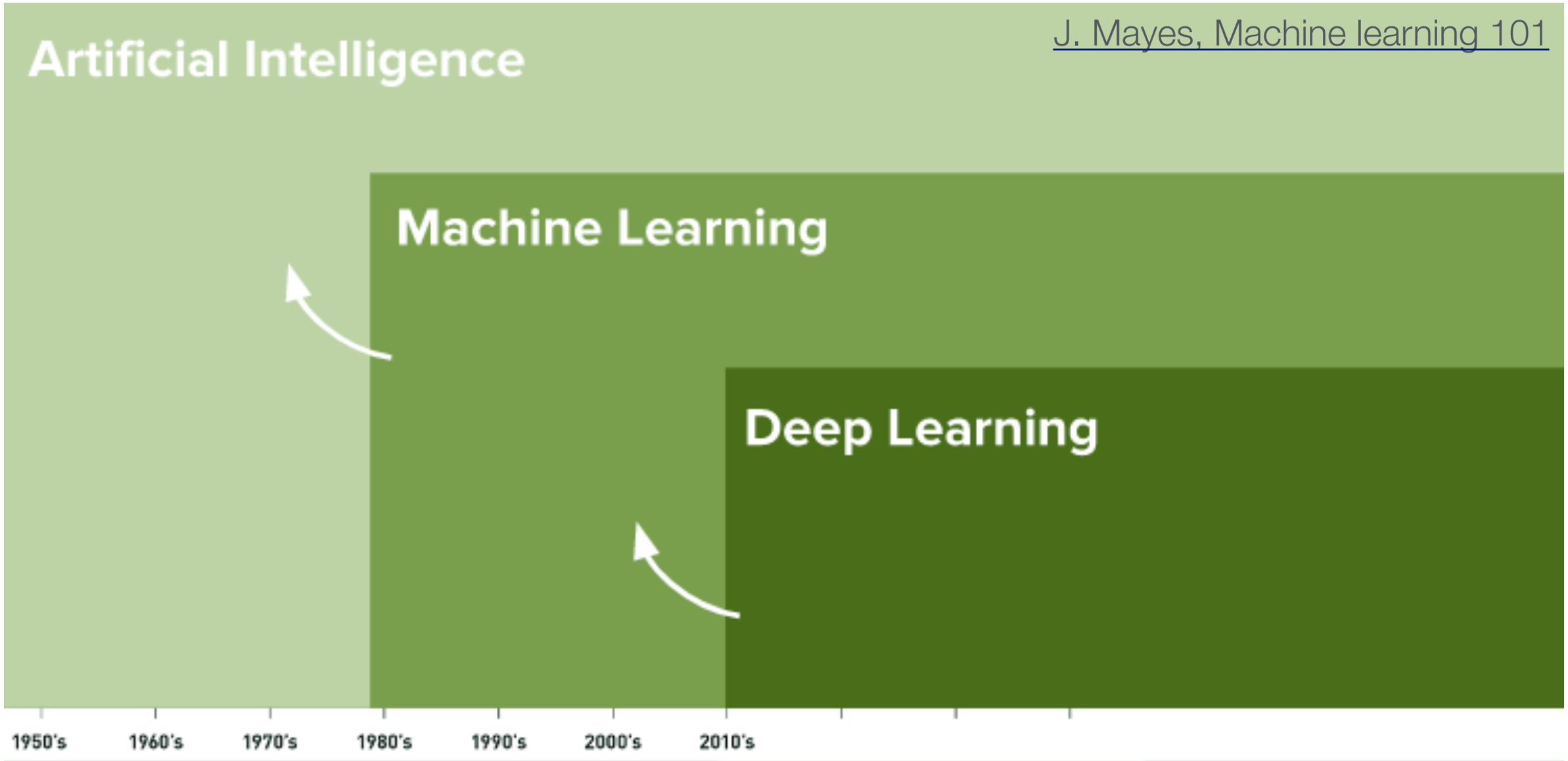


Manual feature engineering vs. automatic feature detection

AI, ML, and DL

"AI is the study of how to make computers perform things that, at the moment, people do better.2

Elaine Rich, Artificial intelligence, McGraw-Hill 1983

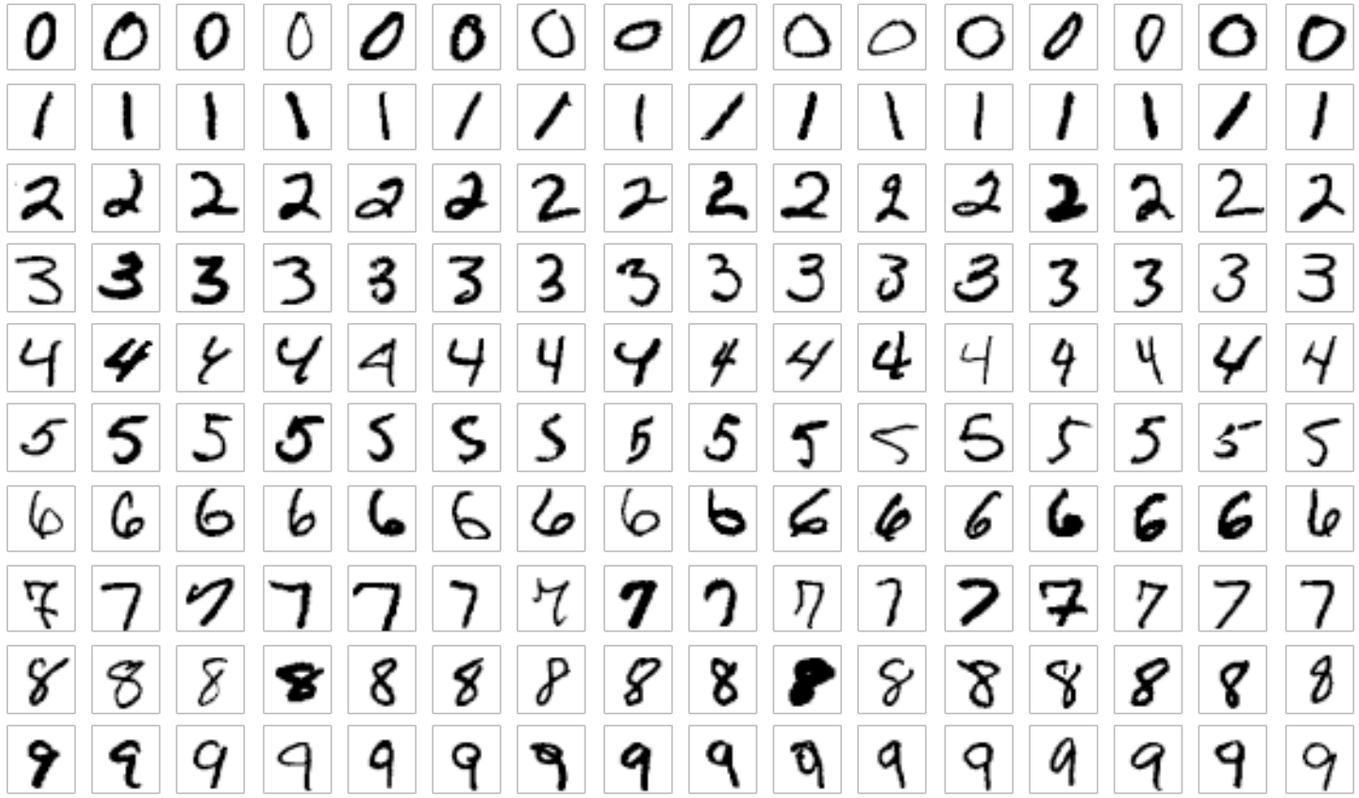


"deep" in deep learning: artificial neural nets with multiple layers of nonlinear processing units for feature extraction

Machine Learning: The "hello world" Problem

Recognition of handwritten digits

- MNIST database (Modified National Institute of Standards and Technology database)
- 60,000 training images and 10,000 testing images labeled with correct answer
- 28 pixel x 28 pixel
- Algorithms have reached "near-human performance"
- Smallest error rate (2018): 0.18%



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

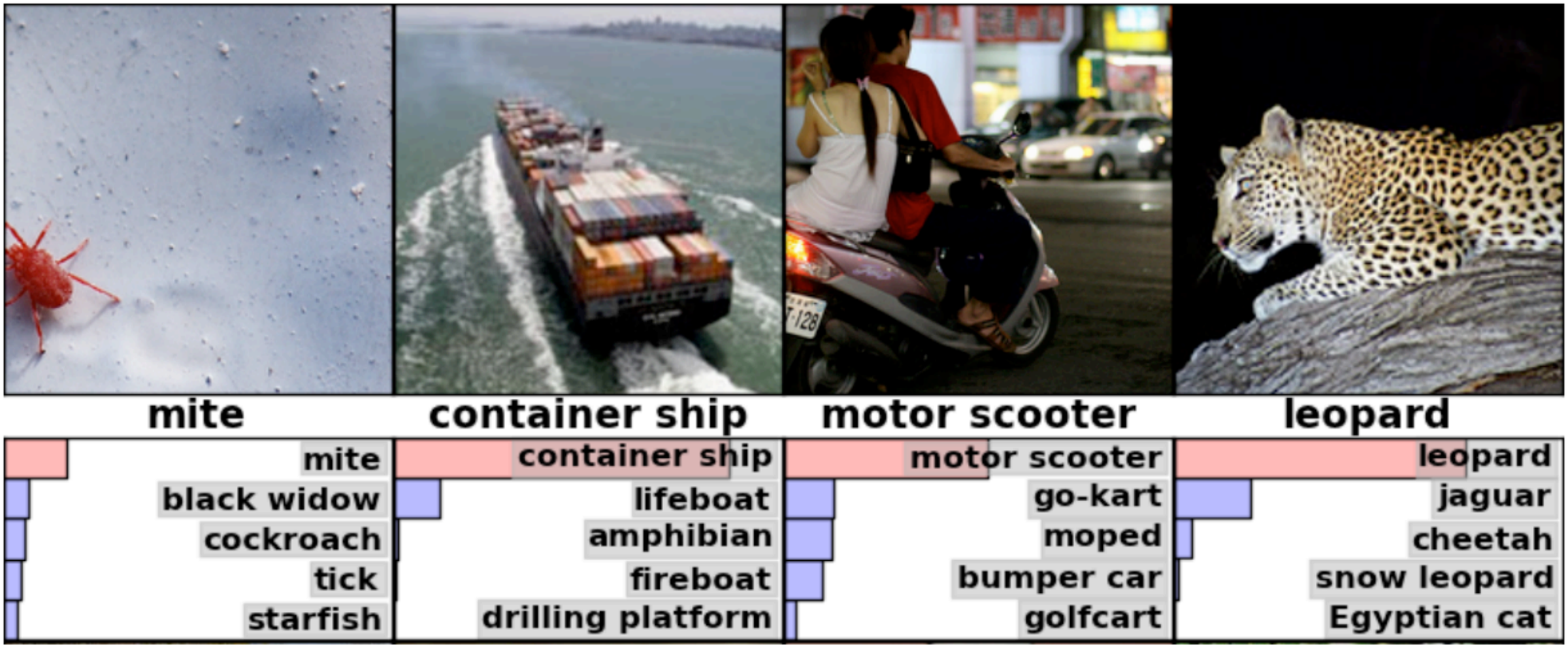
Play with MNIST data set and Keras (Stefan Wunsch, CERN IML Workshop):
https://github.com/stwunsch/iml_tensorflow_keras_workshop

Machine learning: Image Recognition

ImageNet database

- 14 million images, 22,000 categories
- Since 2010, the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC): 1.4 million images, 1000 categories
- In 2017, 29 of 38 competing teams got less than 5% wrong

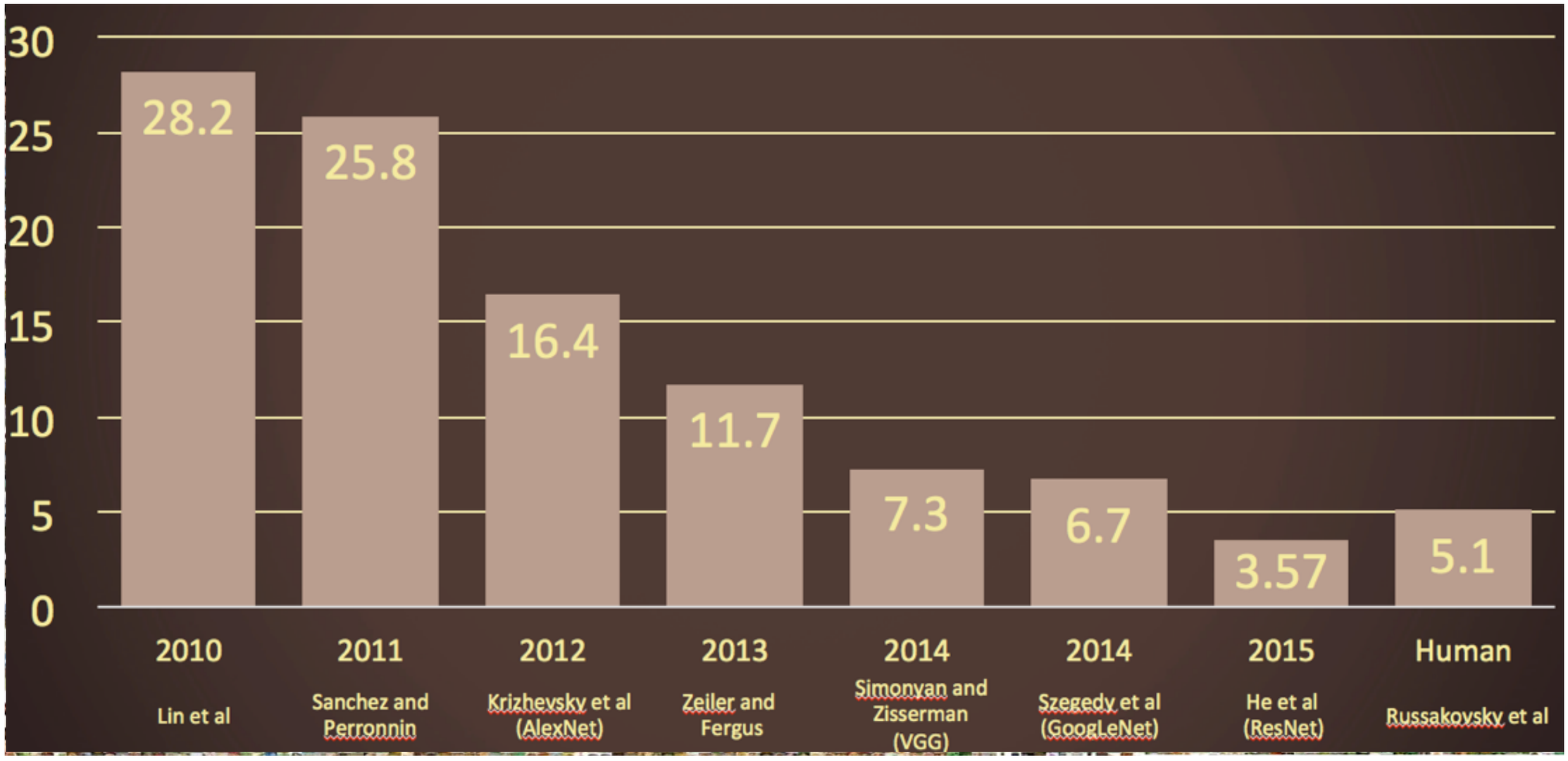
https://en.wikipedia.org/wiki/ImageNet



https://www.tensorflow.org/tutorials/image_recognition

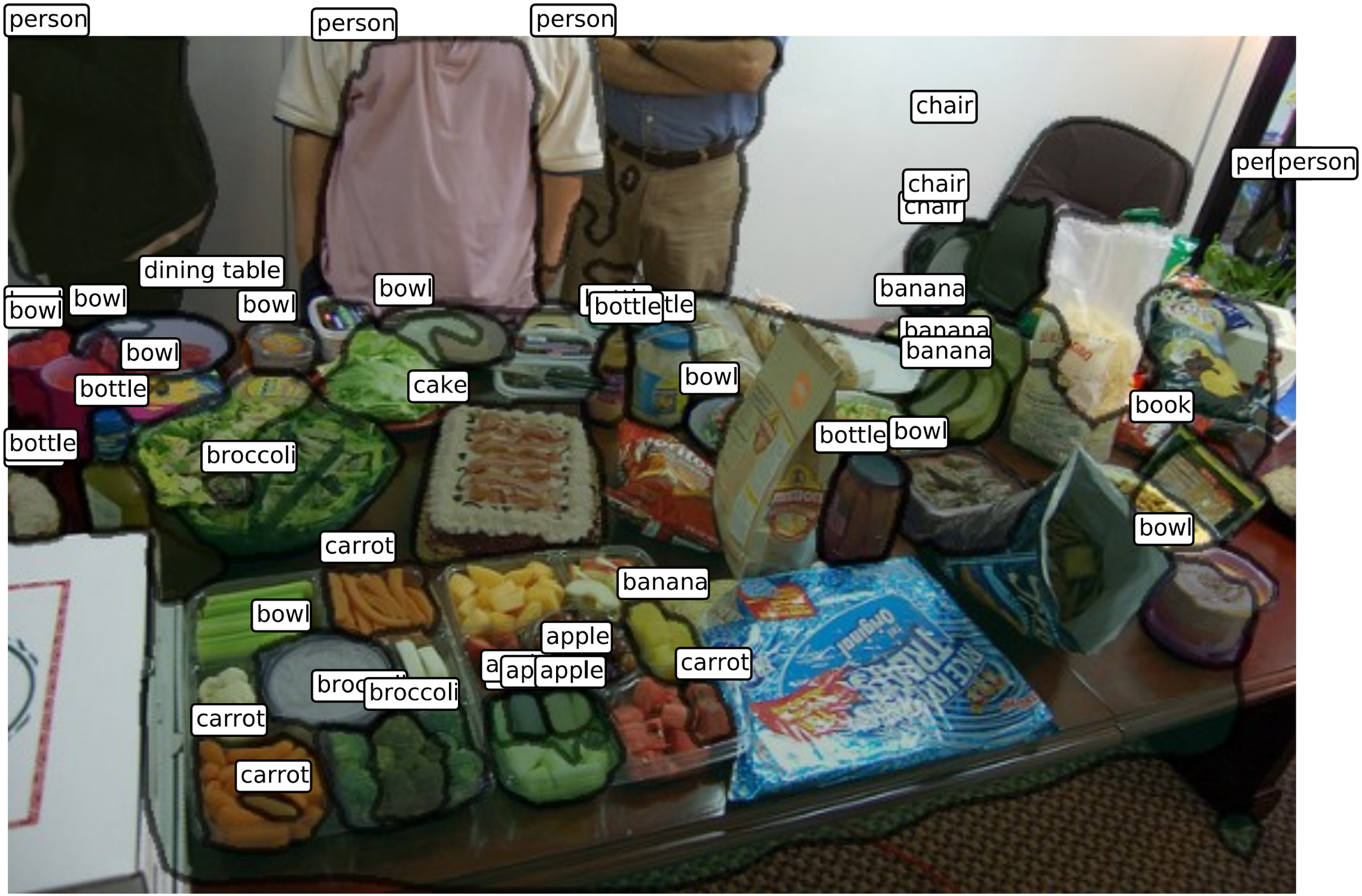
ImageNet: Large Scale Visual Recognition Challenge

Error rate:



O. Russakovsky et al, arXiv:1409.0575

Further examples (I):
Segmenting and Localizing Objects



Further examples (II): Image captioning

- [Lebret, Pinheiro, Collobert 2015] [Kulkarni 11] [Mitchell 12] [Vinyals 14] [Mao 14]

A man riding skis on a snow covered ski slope.

NP: a man, skis, the snow, a person, a woman, a snow covered slope, a slope, a snowboard, a skier, man.
VP: wearing, riding, holding, standing on, skiing down.
PP: on, in, of, with, down.

A man wearing skis on the snow.

A man is doing skateboard tricks on a ramp.

NP: a skateboard, a man, a trick, his skateboard, the air, a skateboarder, a ramp, a skate board, a person, a woman.
VP: doing, riding, is doing, performing, flying through.
PP: on, of, in, at, with.

A man riding a skateboard on a ramp.

The girl with blue hair stands under the umbrella.

NP: a woman, an umbrella, a man, a person, a girl, umbrellas, that, a little girl, a cell phone.
VP: holding, wearing, is holding, holds, carrying.
PP: with, on, of, in, under.

A woman is holding an umbrella.

A slice of pizza sitting on top of a white plate.

NP: a plate, a white plate, a table, pizza, it, a pizza, food, a sandwich, top, a close.
VP: topped with, has, is, sitting on, is on.
PP: of, on, with, in, up.

A table with a plate of pizza on a white plate.

A baseball player swinging a bat on a field.

NP: the ball, a game, a baseball player, a man, a tennis court, a ball, home plate, a baseball game, a batter, a field.
VP: swinging, to hit, playing, holding, is swinging.
PP: on, during, in, at, of.

A baseball player swinging a bat on a baseball field.

A bunch of kites flying in the sky on the beach.

NP: the beach, a beach, a kite, kites, the ocean, the water, the sky, people, a sandy beach, a group.
VP: flying, flies, is flying, flying in, are.
PP: on, of, with, in, at.

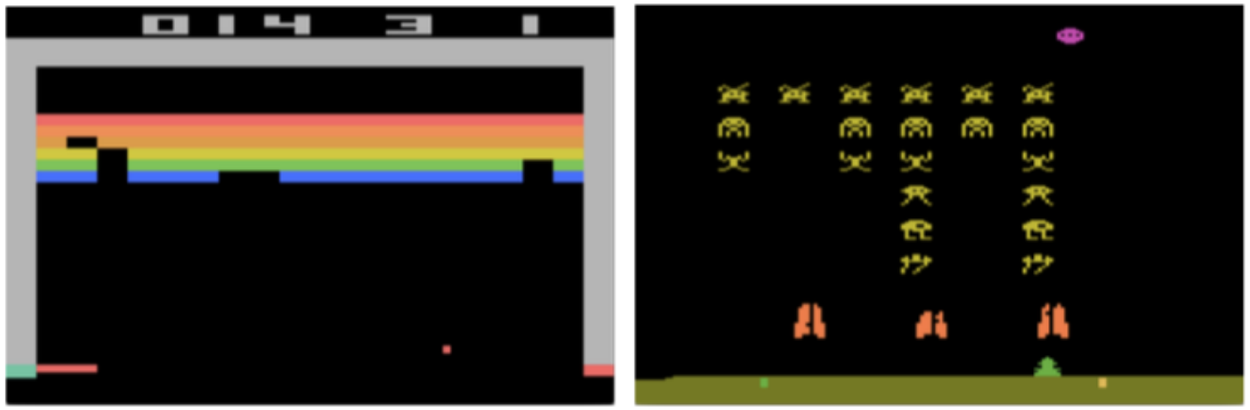
People flying kites on the beach.

Three Types of Learning

LeCun 2018, Power And Limits of Deep Learning,
https://www.youtube.com/watch?v=0tEhw5t6rhc

Reinforcement learning

- The machine ("the agent") predicts a scalar reward given once in a while
- Weak feedback



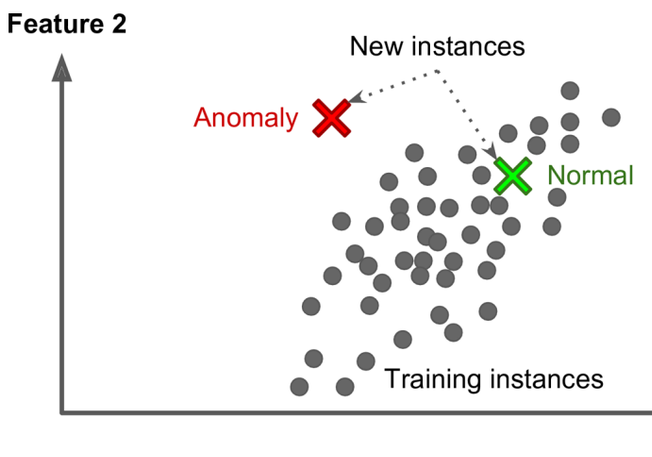
arXiv:1312.5602

Supervised learning

- The machine predicts a category based on labeled training data
- Medium feedback

Unsupervised learning

- Describe/find hidden structure from "unlabeled" data
- Cluster data in different sub-groups with similar properties

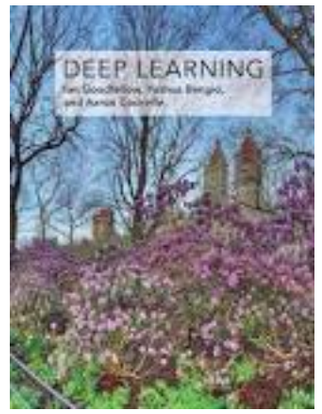


Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

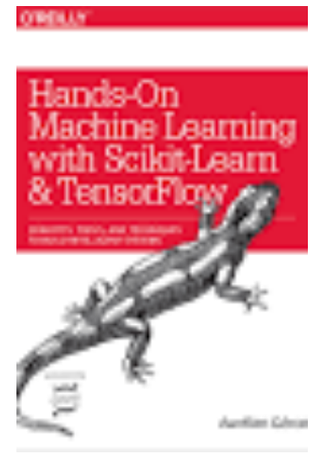
Example: anomaly detection

Books on Machine Learning

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, free online http://www.deeplearningbook.org/



- Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*



Lectures / Talks / Papers

Multivariate analysis

- Pushpalatha C. Bhat, Multivariate Analysis Methods in Particle Physics
- Lecture "Statistical Methods in Particle physics" (WS 2017/18)
  - https://www.physi.uni-heidelberg.de/~reygers/lectures/2017/smipp/stat_methods_ss2017_08_multivariate_analysis.pdf

Machine learning

- Lecture "Convolutional Neural Networks for Visual Recognition"
  - http://cs231n.stanford.edu/slides
- CERN academic training lecture: Michael Kagan, Machine learning
  - https://indico.cern.ch/event/619370/
- CERN IML workshop
  - 2019: https://indico.cern.ch/event/766872
  - 2018: https://indico.cern.ch/event/668017

More Papers

- A high-bias, low-variance introduction to Machine Learning for physicists
  https://arxiv.org/abs/1803.08823
- Machine learning and the physical sciences
  https://arxiv.org/abs/1903.10563

Contents

Multivariate 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"

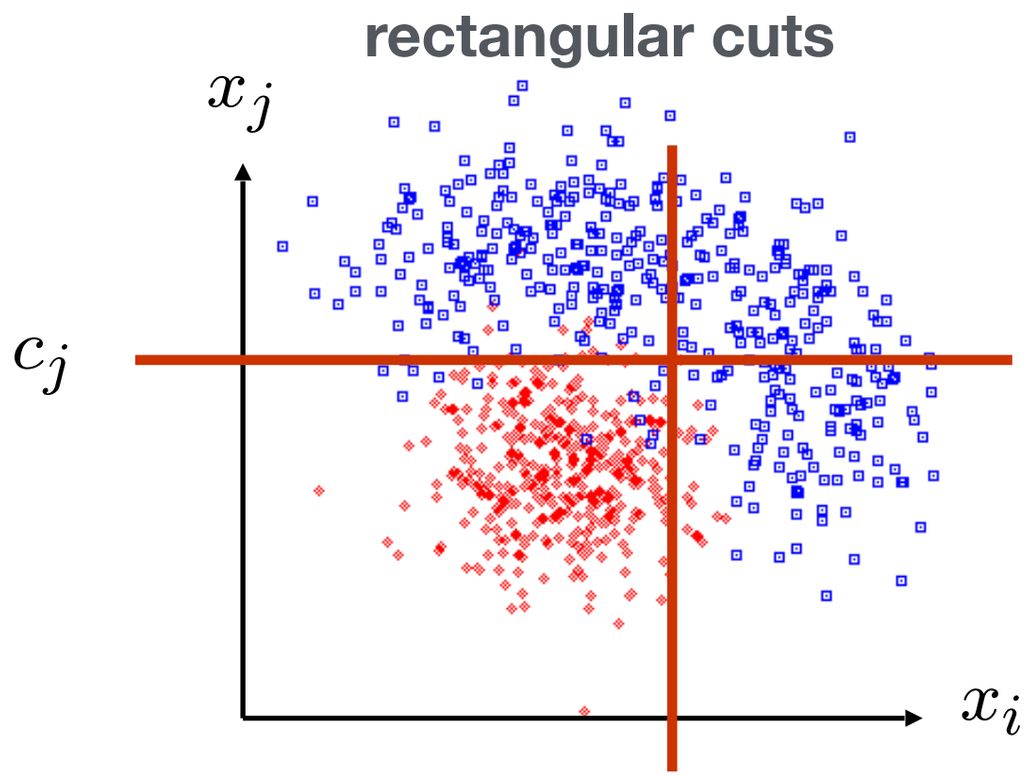
Goal: classify events as signal or background in an optimal way.

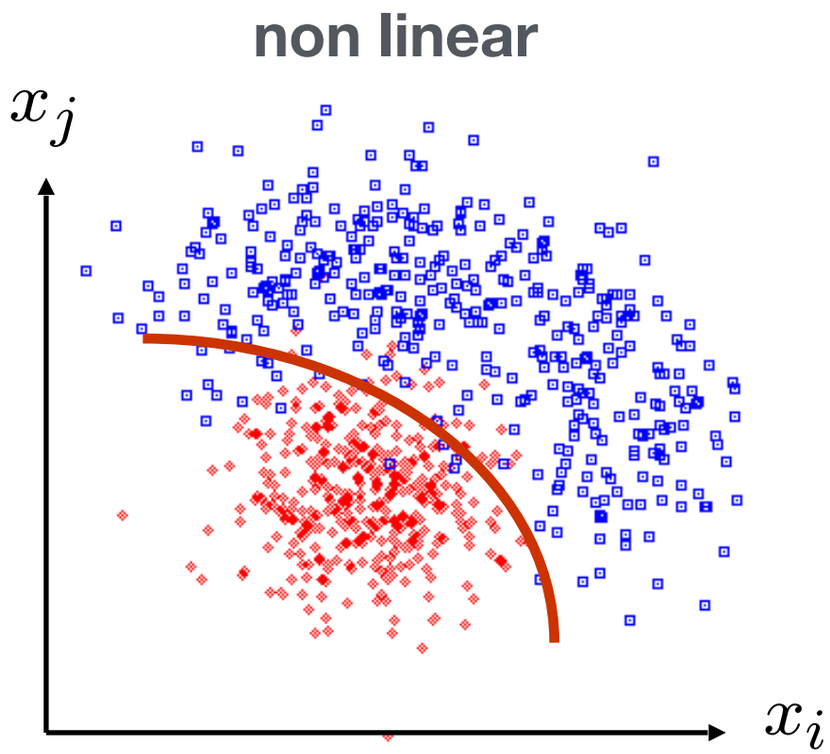
This is usually done by mapping the feature vector to a single variable, i.e., to scalar "test statistic":

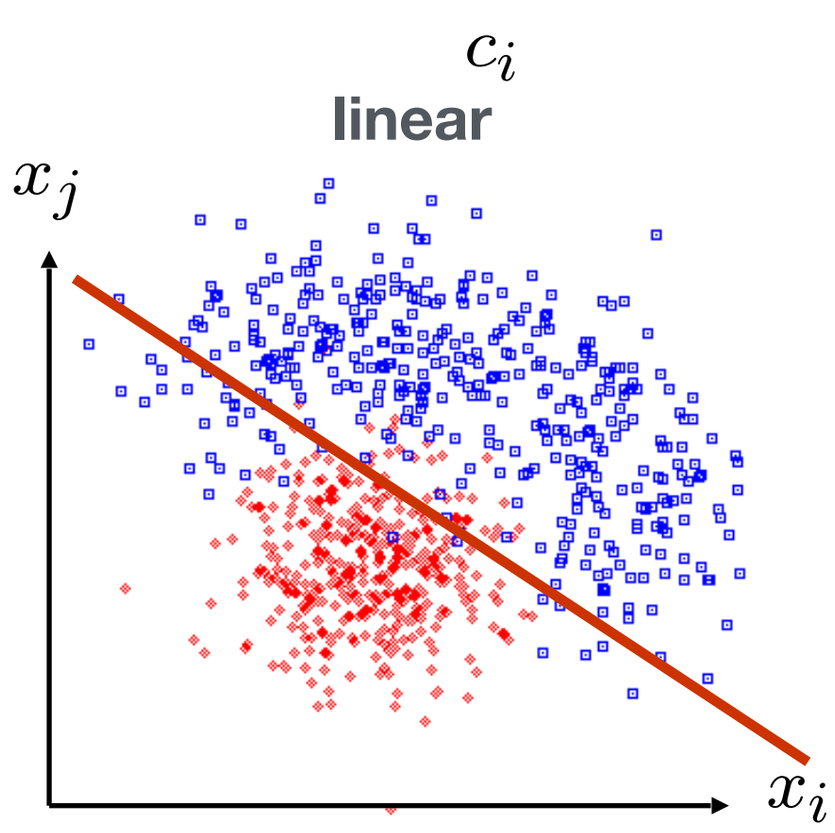
$$\mathbb{R}^n \to \mathbb{R}: \quad y(\vec{x})$$

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.

Classification: Learning Decision Boundaries







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

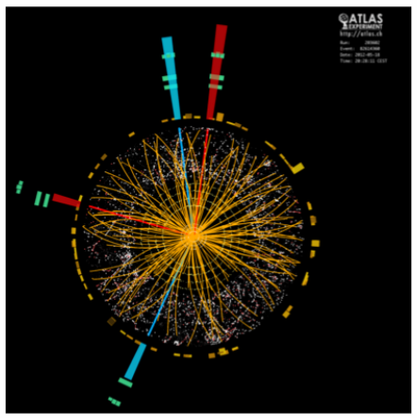
…

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

Supervised Learning in a Nutshell

M. Kagan,
https://indico.cern.ch/event/619370/

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



$y(\vec{x}, \vec{w})$
Function with adjustable parameters

Loss Function

Compare prediction with true label

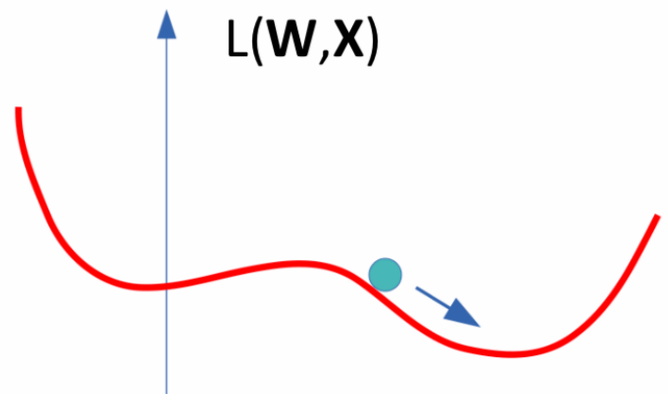
Loss

True labels:
Higgs = 1
Bkg = 0

Design function $y(\vec{x}, \vec{w})$ with ajdustable parameters $\vec{w}$

Design a loss function

Find best parameters which minimize loss



Supervised Learning: Classification and Regression

The codomain Y of the function y: $X \rightarrow Y$ can be a set of labels or classes or a continuous domain, e.g., $\mathbb{R}$

Binary classification: $Y = \{0, 1\}$ e.g., signal or background

Multi-class classification $Y = \{c_1, c_2, ..., c_n\}$

Labels sometimes represented as "**one-hot vector**" (no ordering btw. labels):

$t_a = \{0, 0, ..., 1, ..., 0\}$

Y = finite set of labels → classification

Y = real numbers → regression

"All the impressive achievements of deep learning amount to just curve fitting"

J. Pearl, Turing Award Winner 2011,
https://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/

Supervised Learning:
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 Learning: Cross Validation

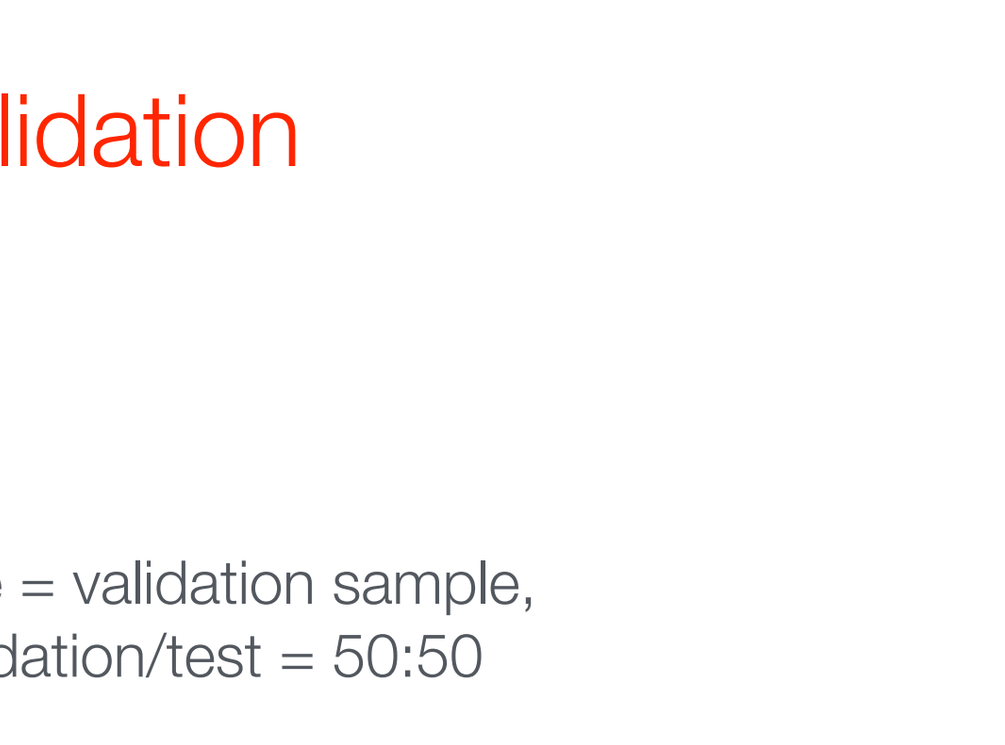
Rule of thumb if training data not expensive

- 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



Often Used Loss Functions

Square Error Loss:

- often used in regression

predicted label

true label

$$E(y(\vec{x}, \vec{w}), t) = (y(\vec{x}, \vec{w}) - t)^2$$

Cross entropy:

- $t \in \{0, 1\}$
- Often used in classification

$$E(y(\vec{x}, \vec{w}), t) = -\, t \log y(\vec{x}, \vec{w}) - (1 - t) \log(1 - y(\vec{x}, \vec{w}))$$

More on Entropy

Self-information of an event x: $I(x) = -\log p(x)$

Shannon entropy: $H(P) = -\sum p_i \log p_i$

- Expected amount of information in an event drawn from a distribution P.
- Measure of the minimum of amount of bits needed on average to encode symbols drawn from a distribution

Cross entropy: $H(P, Q) = -E[\log q_i] = -\sum p_i \log q_i$

- Can be interpreted as a measure of the amount of bits needed when a wrong distribution Q is assumed while the data actually follows a distribution P
- Measure of dissimilarity between distributions P and Q

Cross-Entropy Error Function and Logistic Regression

Outcomes $y \in \{0,1\}$

True probability for a given feature vector : $p_{y=1} = y$, $p_{y=0} = 1 - y$

Predicted probability (logistic regression):

$$q_{y=1} \equiv \hat{y} = g(\vec{x} \cdot \vec{w}), \qquad g(z) = \frac{1}{1+e^{-z}} \qquad q_{y=0} = 1 - \hat{y}$$

logistic function

Cross entropy:

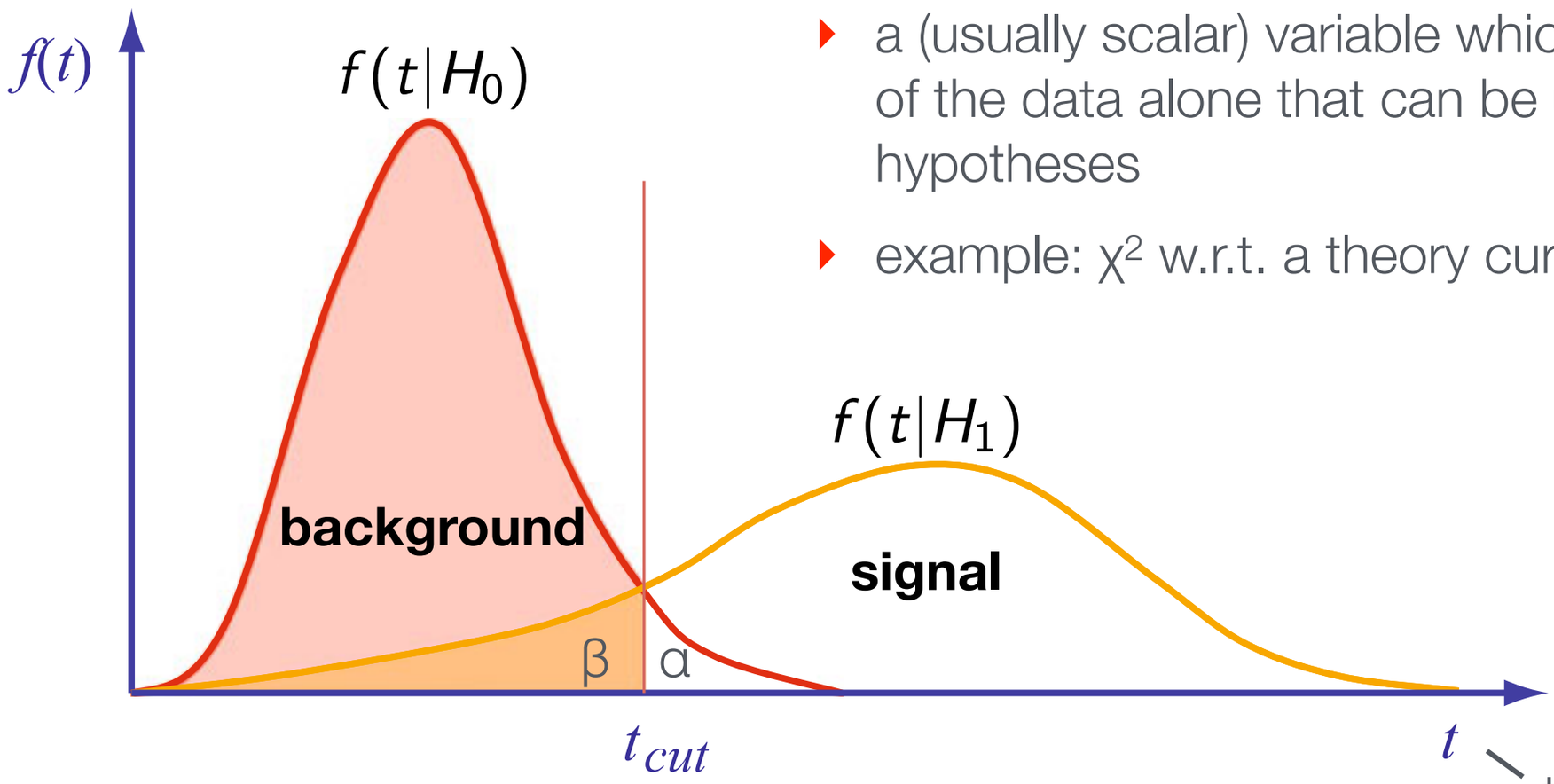
$$H(p,q) = -\sum_i p_i \log q_i = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

Loss function:

$$E(\vec{w}) = \frac{1}{N_{\text{samples}}} \sum_{i=1}^{N_{\text{samples}}} H(p_i, q_i)$$

https://en.wikipedia.org/wiki/Cross_entropy#Cross-entropy_error_function_and_logistic_regression

Hypothesis testing



test statistic:

- a (usually scalar) variable which is a function of the data alone that can be used to test hypotheses
- example: χ^2 w.r.t. a theory curve

$\epsilon_B \equiv \alpha$ "background efficiency", i.e., prob. to misclassify bckg. as signal

$\epsilon_S \equiv 1 - \beta$ "signal efficiency"

	H_0 is true	H_0 is false (i.e., H_1 is true)
H_0 is rejected	Type I error (α)	Correct decision ($1 - \beta$)
H_0 is not rejected	Correct decision ($1 - \alpha$)	Type II error (β)

Neyman–Pearson Lemma

The likelihood ratio

$$t(\vec{x}) = \frac{f(\vec{x}|H_1)}{f(\vec{x}|H_0)} \qquad \begin{aligned} H_1 &: \text{ signal hypothesis} \\ H_0 &: \text{ background hypothesis} \end{aligned}$$

is an optimal test statistic, i.e., it provides highest "signal efficiency" 1 – β for a given "background efficiency" α.

Accept hypothesis if $t(\vec{x}) = \frac{f(\vec{x}|H_1)}{f(\vec{x}|H_0)} > c$

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

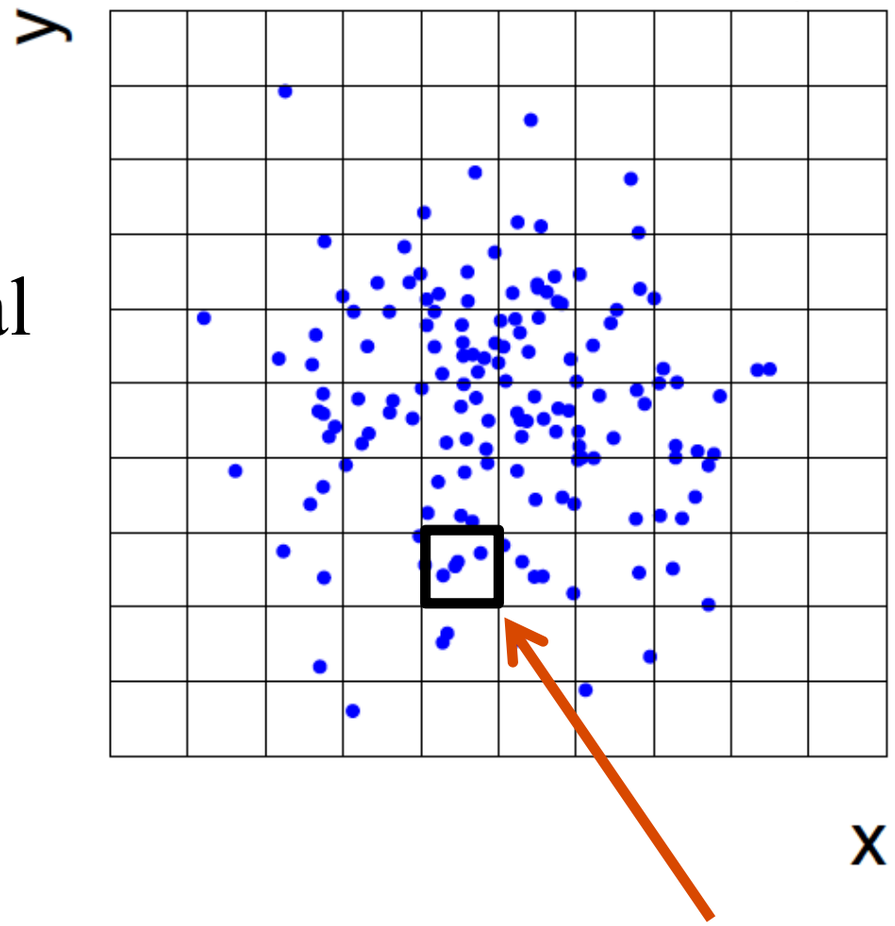
2. Decision boundaries determined directly without approximating the pdf's (linear discriminants, decision trees, neural networks, …)

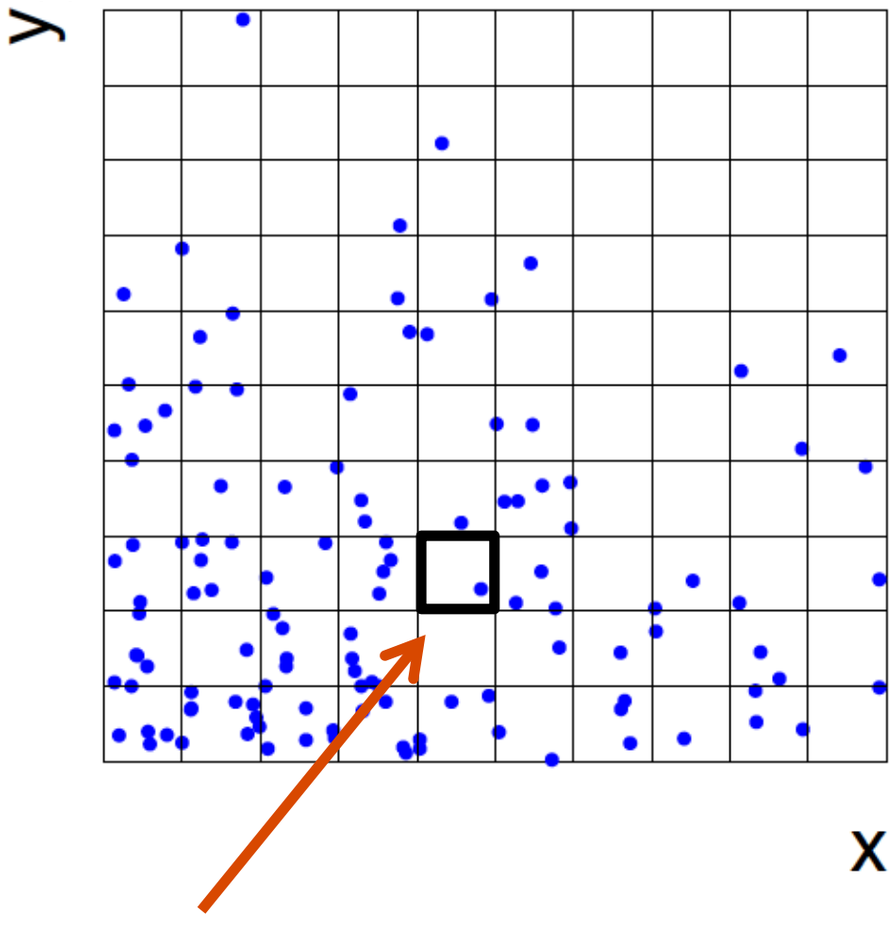
Estimating PDFs from Histograms?

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 $N(x, y|B)$

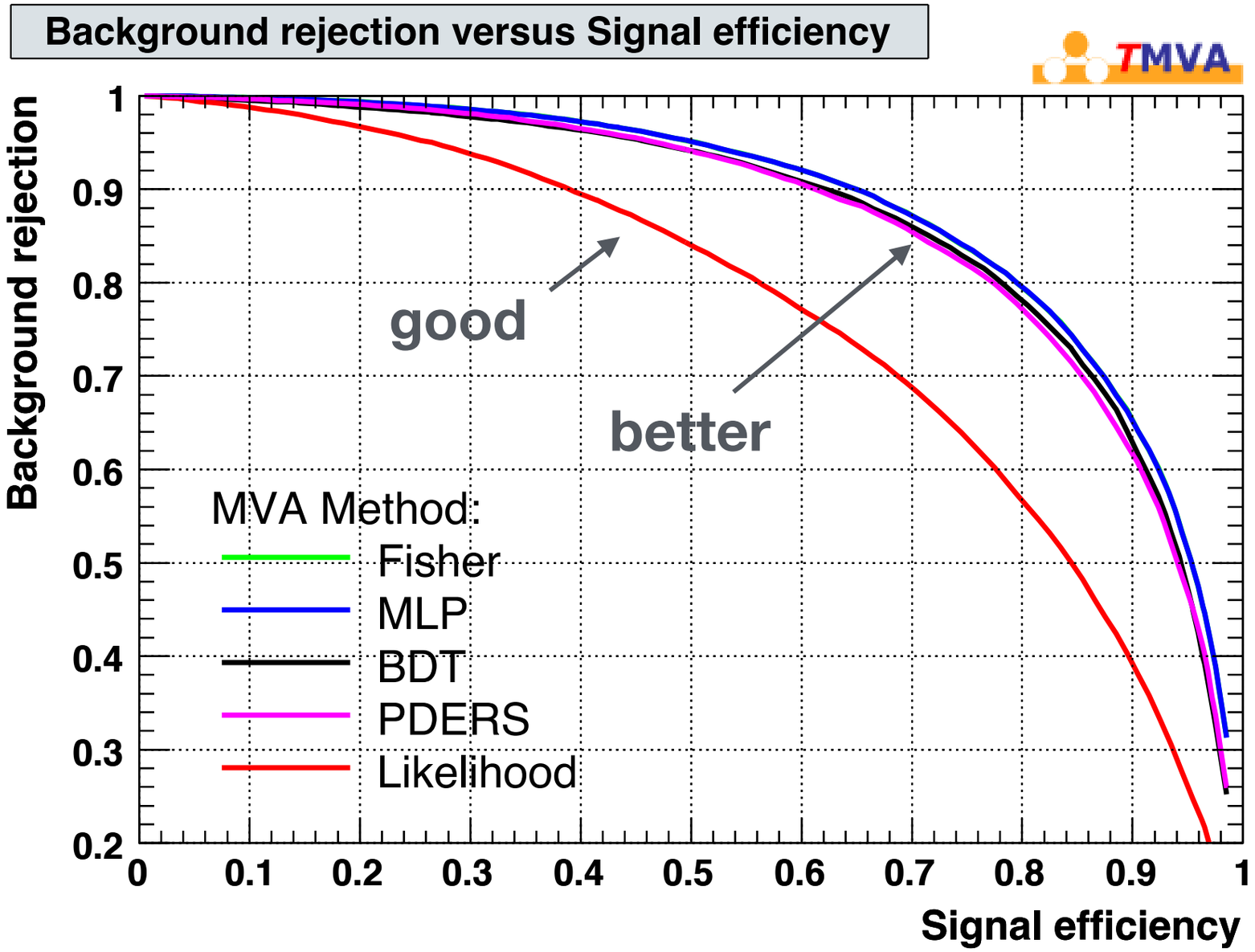
M bins per variable in d dimensions: M^d cells

→ 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**"

ROC Curve

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



$1 - \varepsilon_B$

|

ε_B: background efficiency

Methods discussed in the following slides

Methods based on Neyman-Pearson lemma

- Naïve Bayesian classifier
- *k*-Nearest Neighbor

Other methods

- Fisher's linear discriminant
- Feedforward Neural Network
- Boosted decision trees

Naïve Bayesian Classifier
(also called "Projected Likelihood Classification")

Application of the Neyman-Pearson lemma
(ignoring correlations between the x_i):

$$f(x_1, x_2, ..., x_n) \quad \text{approximated as} \quad L = f_1(x_1) \cdot f_2(x_2) \cdot ... \cdot f_n(x_n)$$

$$\text{where} \quad f_1(x_1) = \int \mathrm{d}x_2 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n)$$

$$f_2(x_2) = \int \mathrm{d}x_1 \mathrm{d}x_3 ... \mathrm{d}x_n \; f(x_1, x_2, ..., x_n)$$

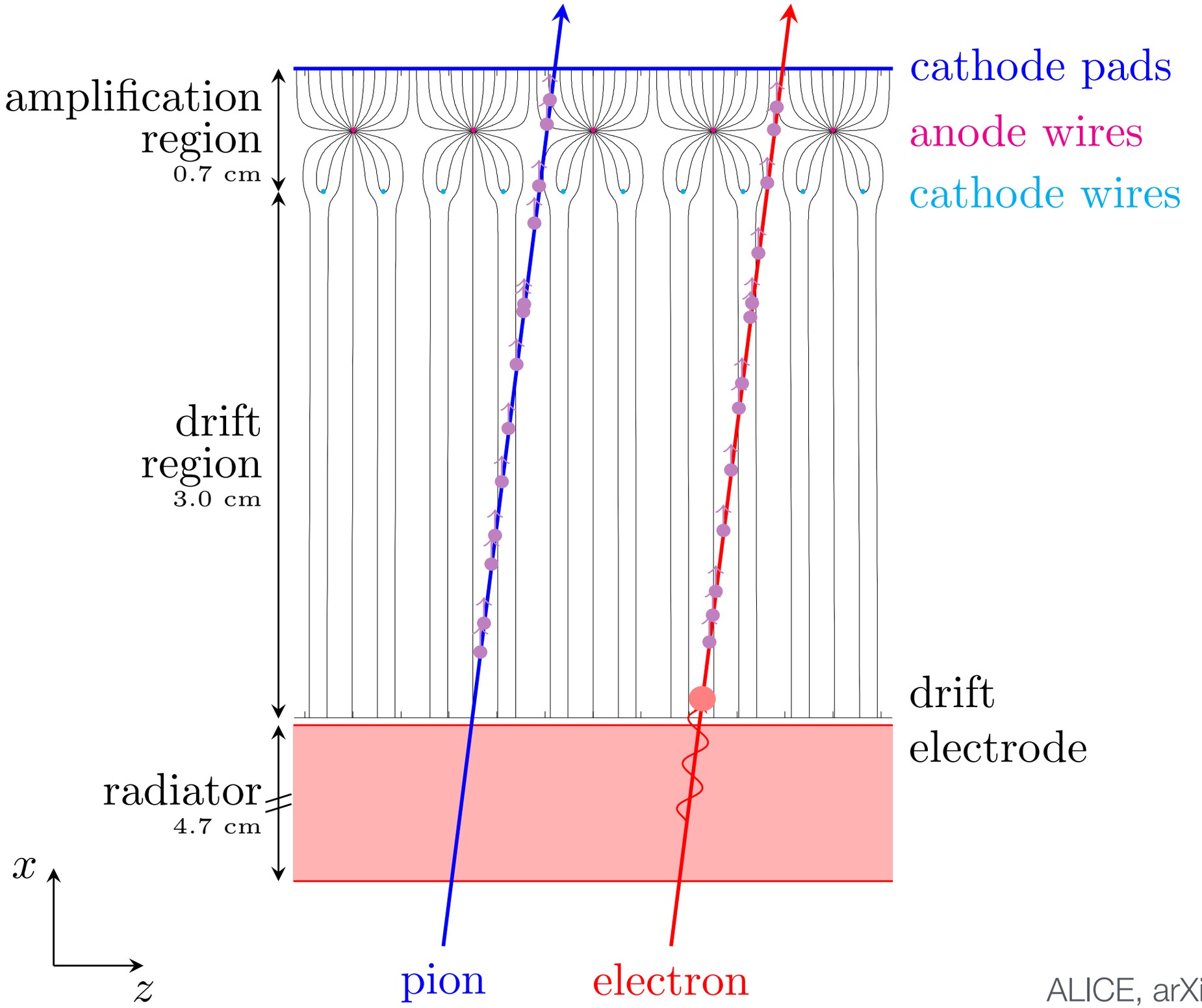
$$\vdots$$

Classification of feature vector $\vec{x}$:

$$y(\vec{x}) = \frac{L_\mathrm{s}(\vec{x})}{L_\mathrm{s}(\vec{x}) + L_\mathrm{b}(\vec{x})} = \frac{1}{1 + L_\mathrm{b}(\vec{x})/L_\mathrm{s}(\vec{x})}$$

Performance not optimal if true PDF does not factorize

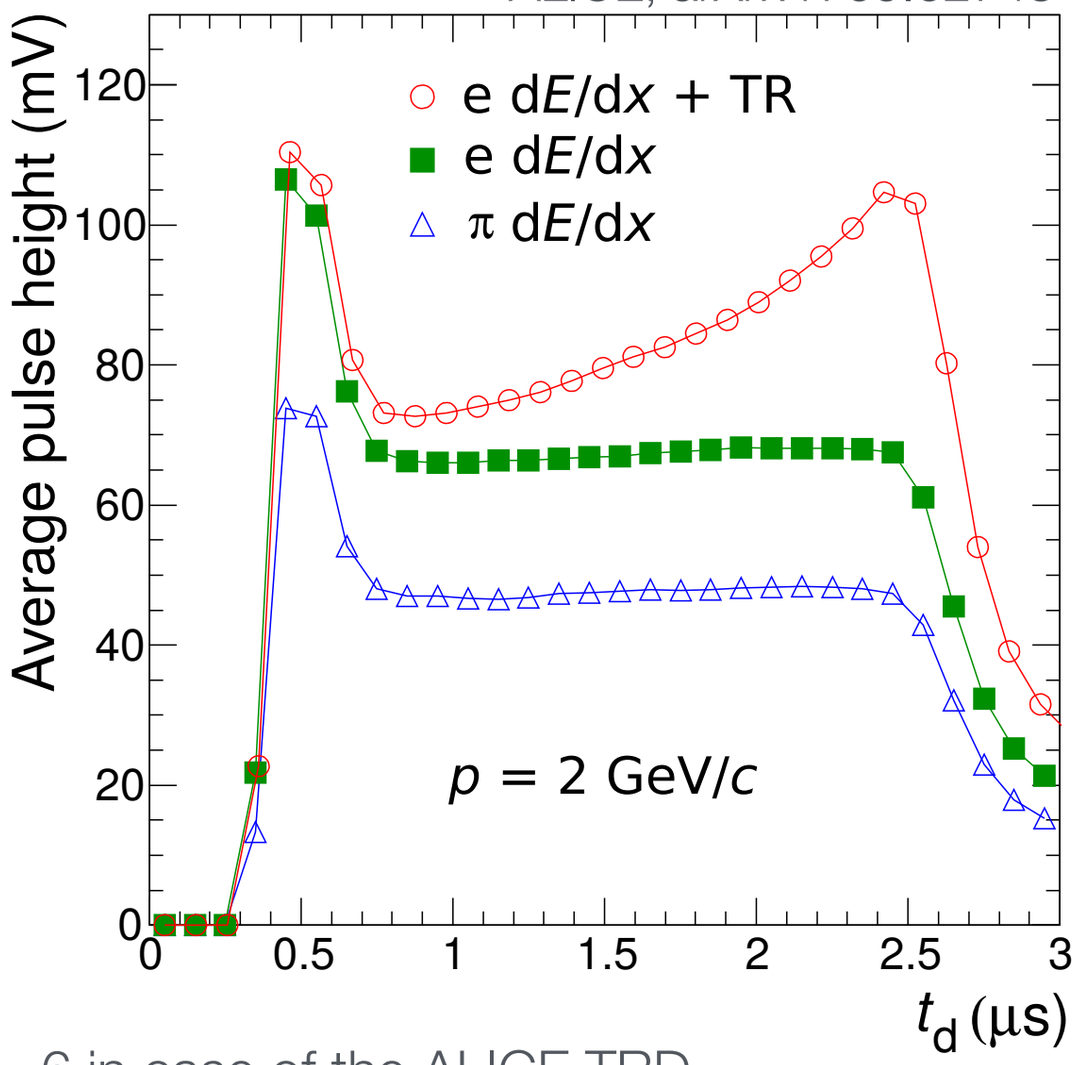
Example: Electron ID with the ALICE TRD (I)



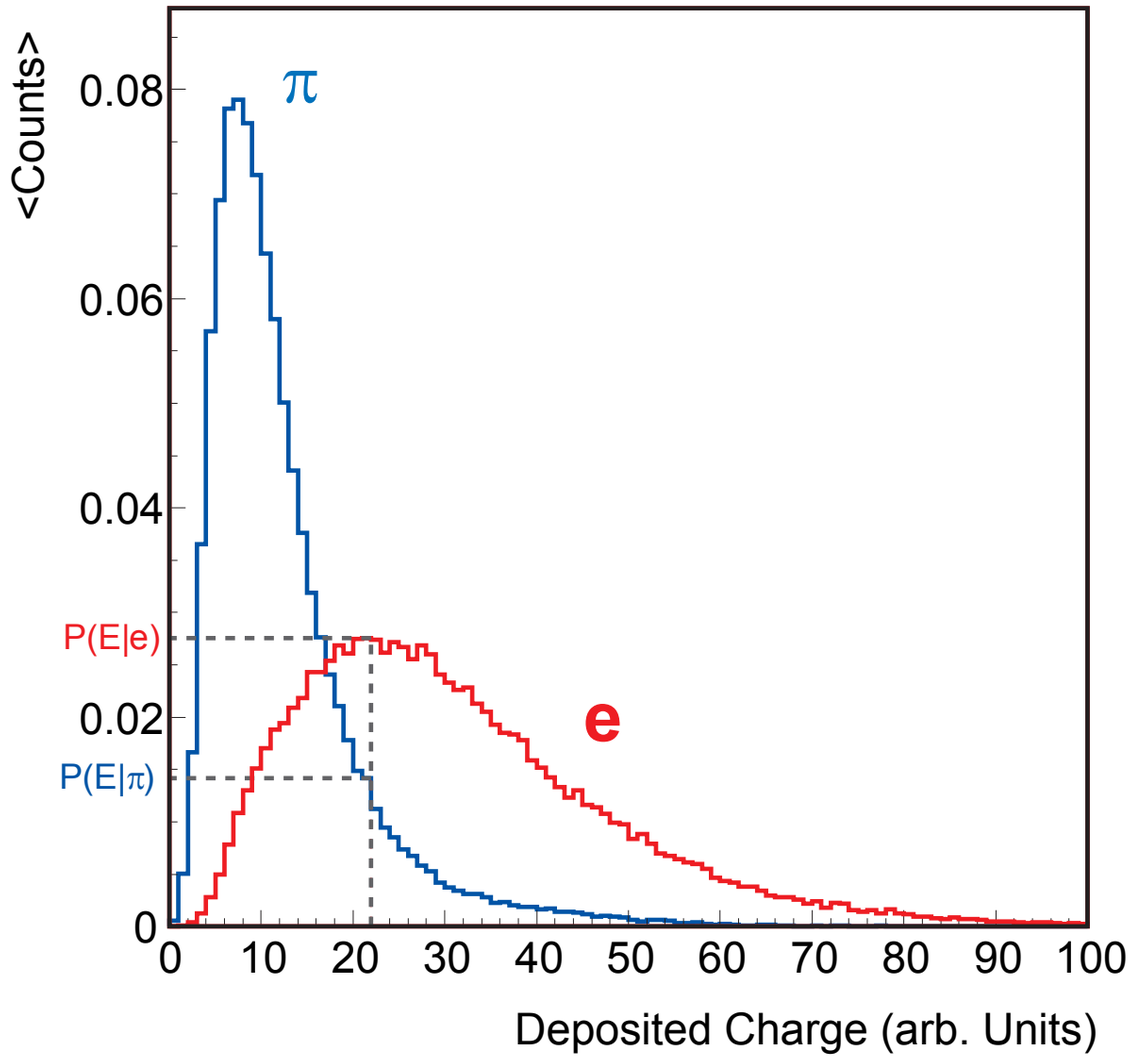
ALICE, arXiv:1709.02743

Example: Electron ID with the ALICE TRD (II)

ALICE, arXiv:1709.02743



Doctoral thesis A. Wilk:
https://inspirehep.net/record/1231193/



6 in case of the ALICE TRD

$$P_e = \prod_{i=1}^{n_{\text{chambers}}} P(E_i|e), \quad P_\pi = \prod_{i=1}^{n_{\text{chambers}}} P(E_i|\pi), \quad \text{test statistic } t = \frac{P_e}{P_e + P_\pi}$$

likelihoods can be multiplied here
(independent information)

high values (close to unity)
indicate high prob. for an electron

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:

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

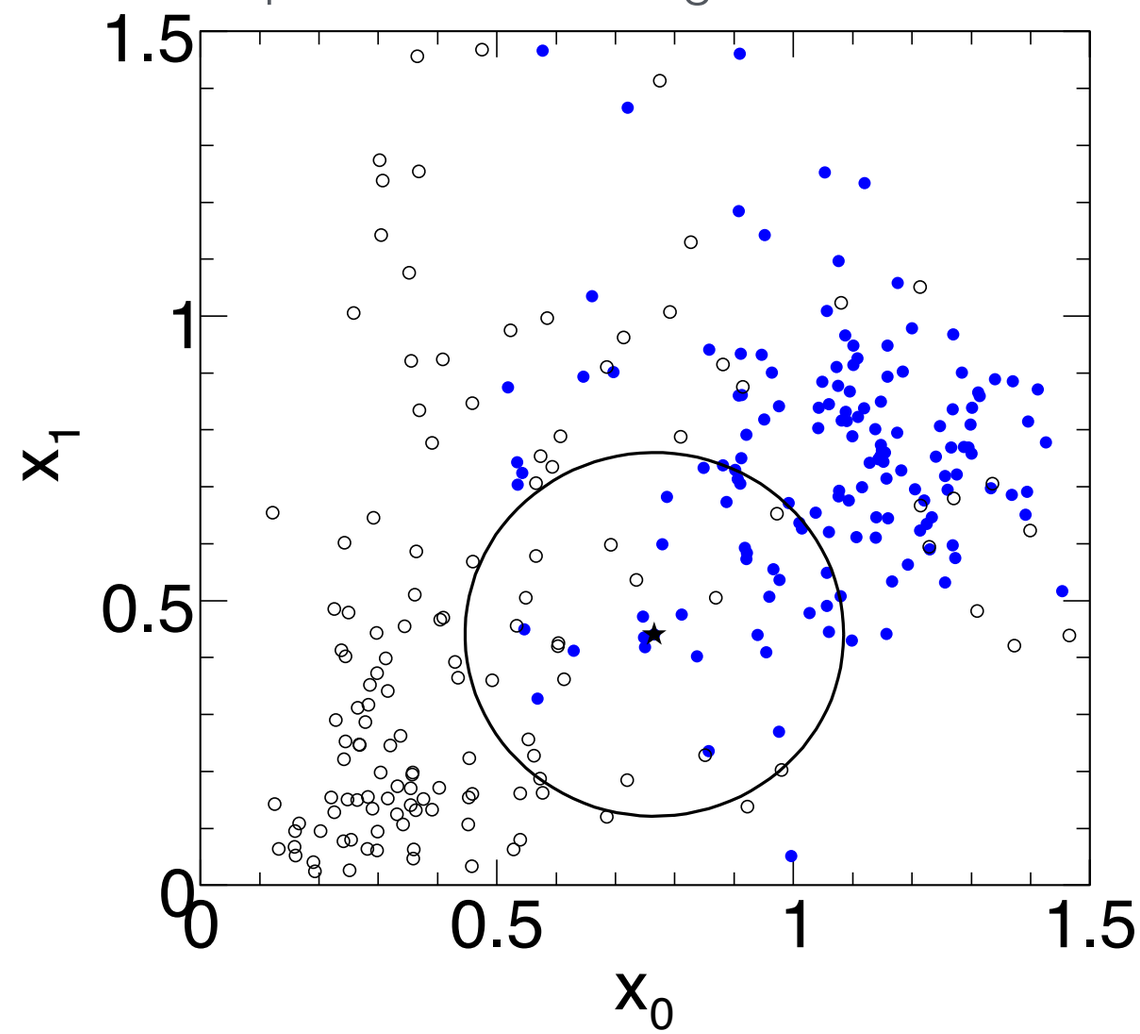
Better: take correlations between variables into account:

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

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.

Fisher Linear Discriminant

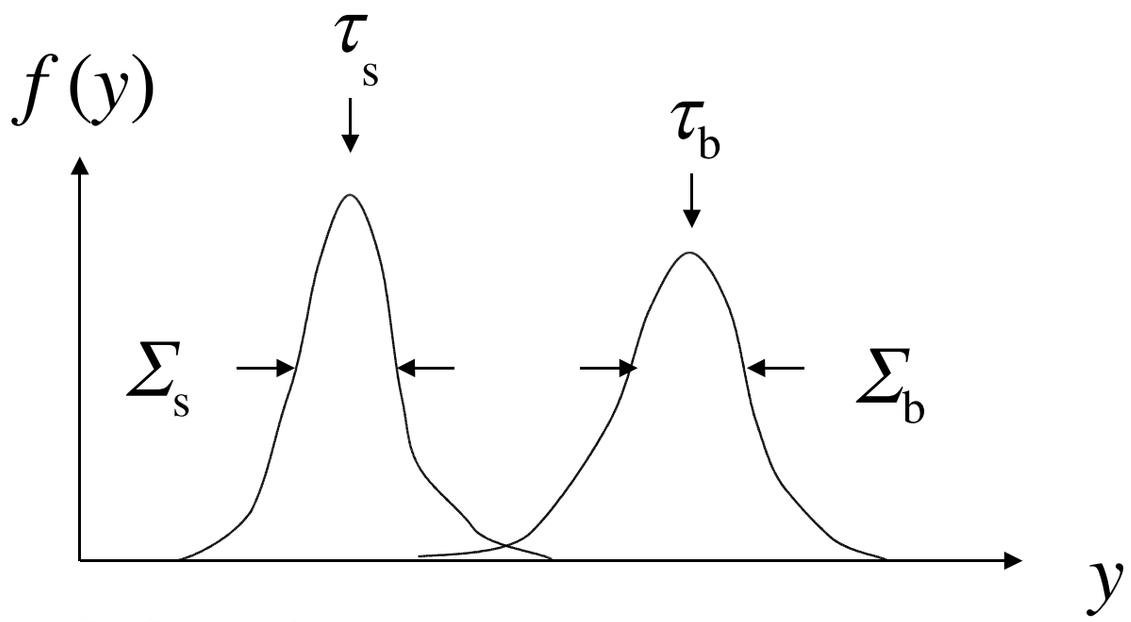
Linear discriminant is simple. Can still be optimal if amount of training data is limited.

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: Determining the Coefficients w_i

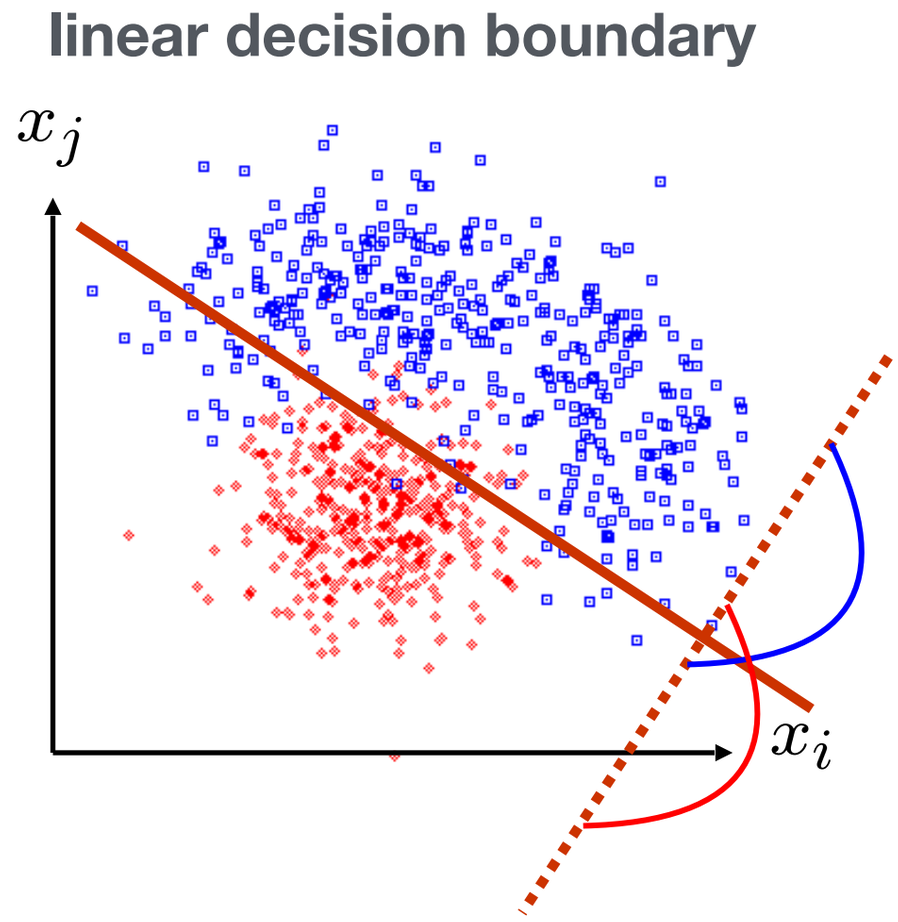
Coefficients are obtained from:

$$\frac{\partial J}{\partial w_i} = 0$$

Linear decision boundaries

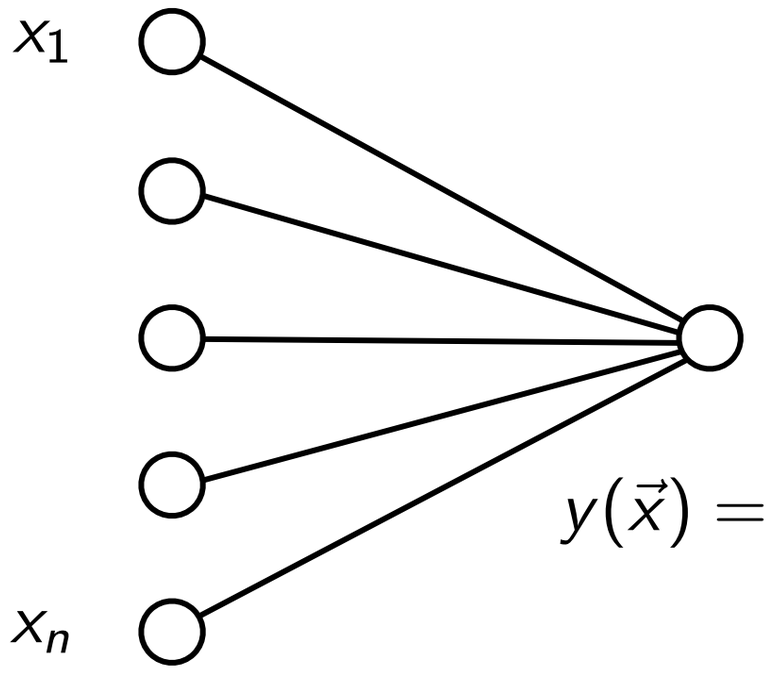
Weight vector $\vec{w}$ can be interpreted as a direction in feature space on which the events are projected.

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



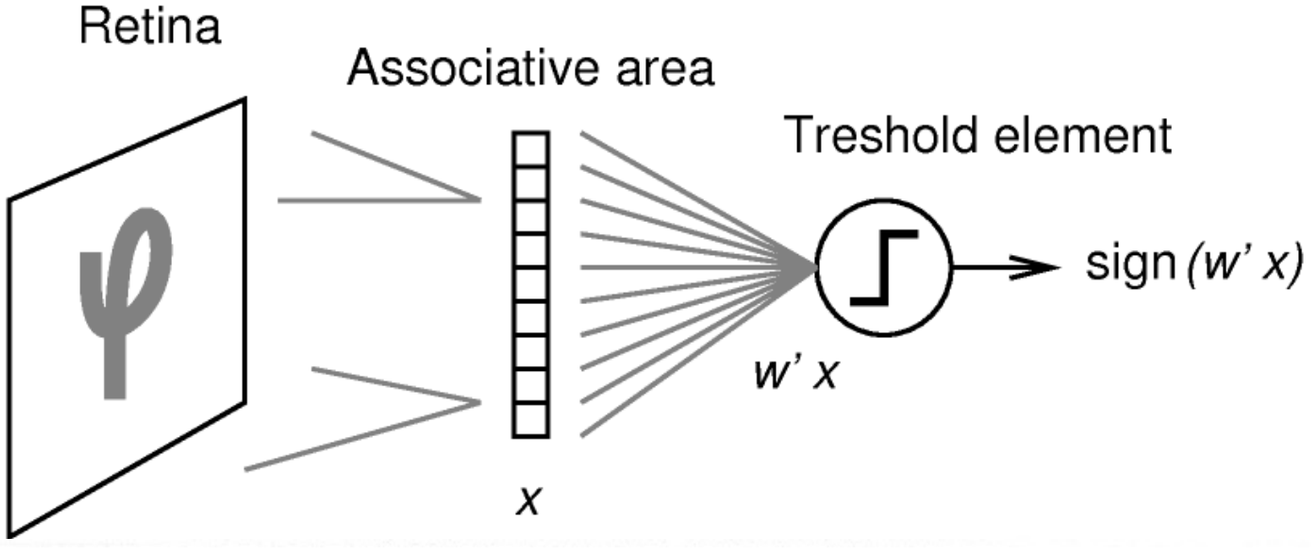
Perceptron (I)

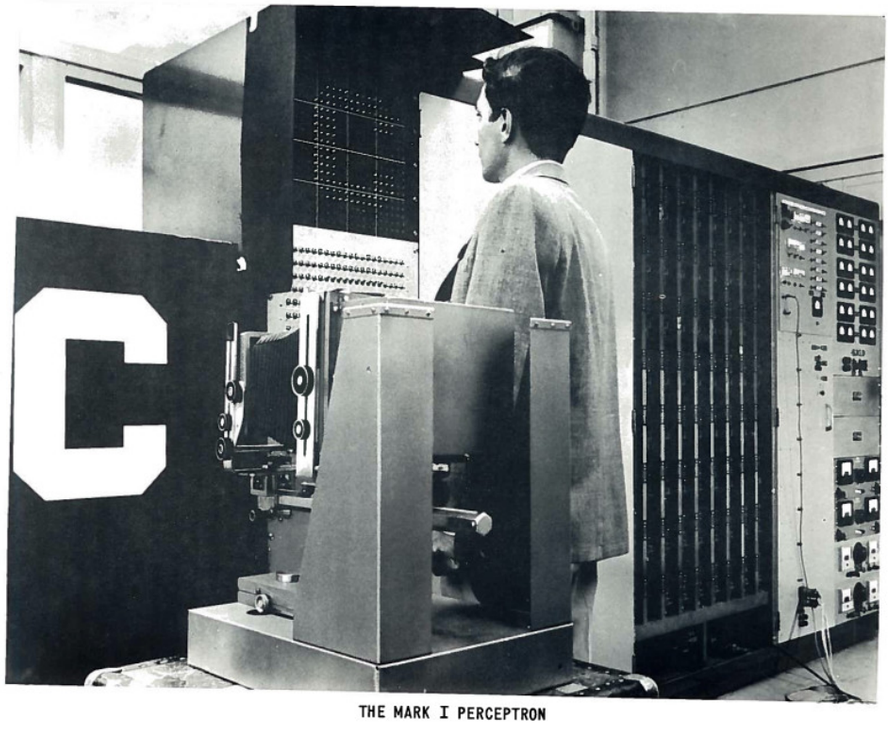
Rosenblatt, 1957



Output:

$$h(\vec{x}) = \begin{cases} 1 & \text{if} \quad \vec{w} \cdot \vec{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$



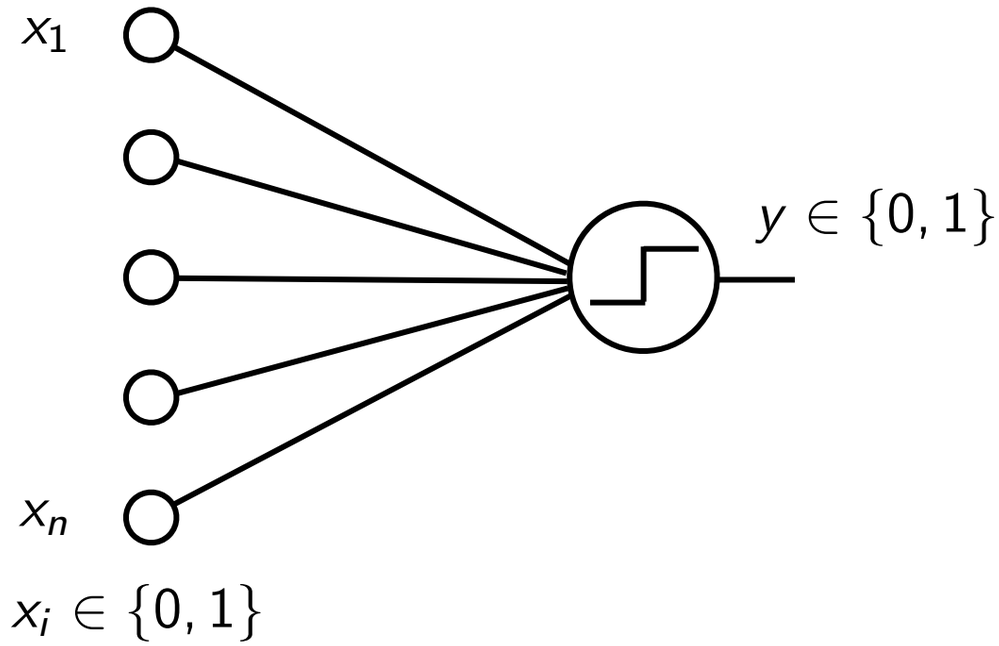


Mark 1 Perceptron. Source: Rosenblatt, Frank (1961) Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms

Perceptron (II)

McCulloch–Pitts (MCP) neuron (1943)

- First mathematical model of a biological neuron
- Boolean input
- Equal weights for all inputs
- Threshold hardcoded

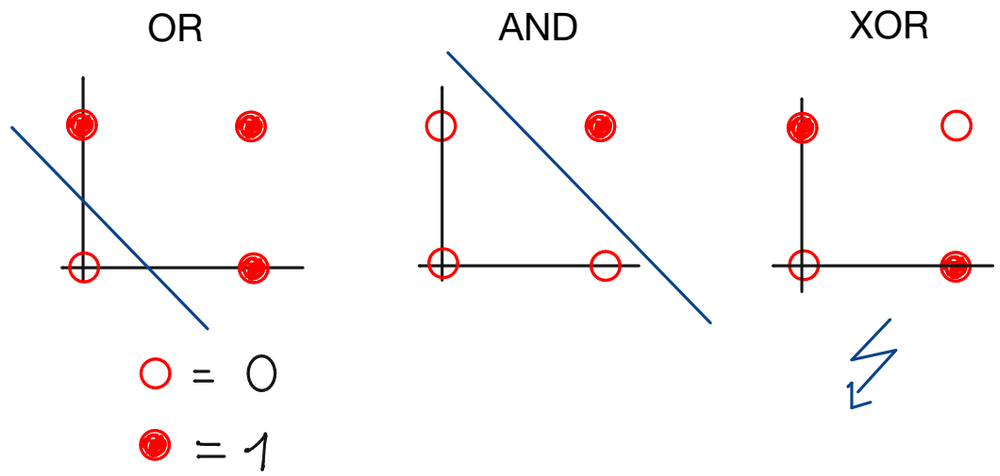


Improvements by Rosenblatt:

- Different weights for inputs
- Algorithm to update weights and threshold given labeled training data

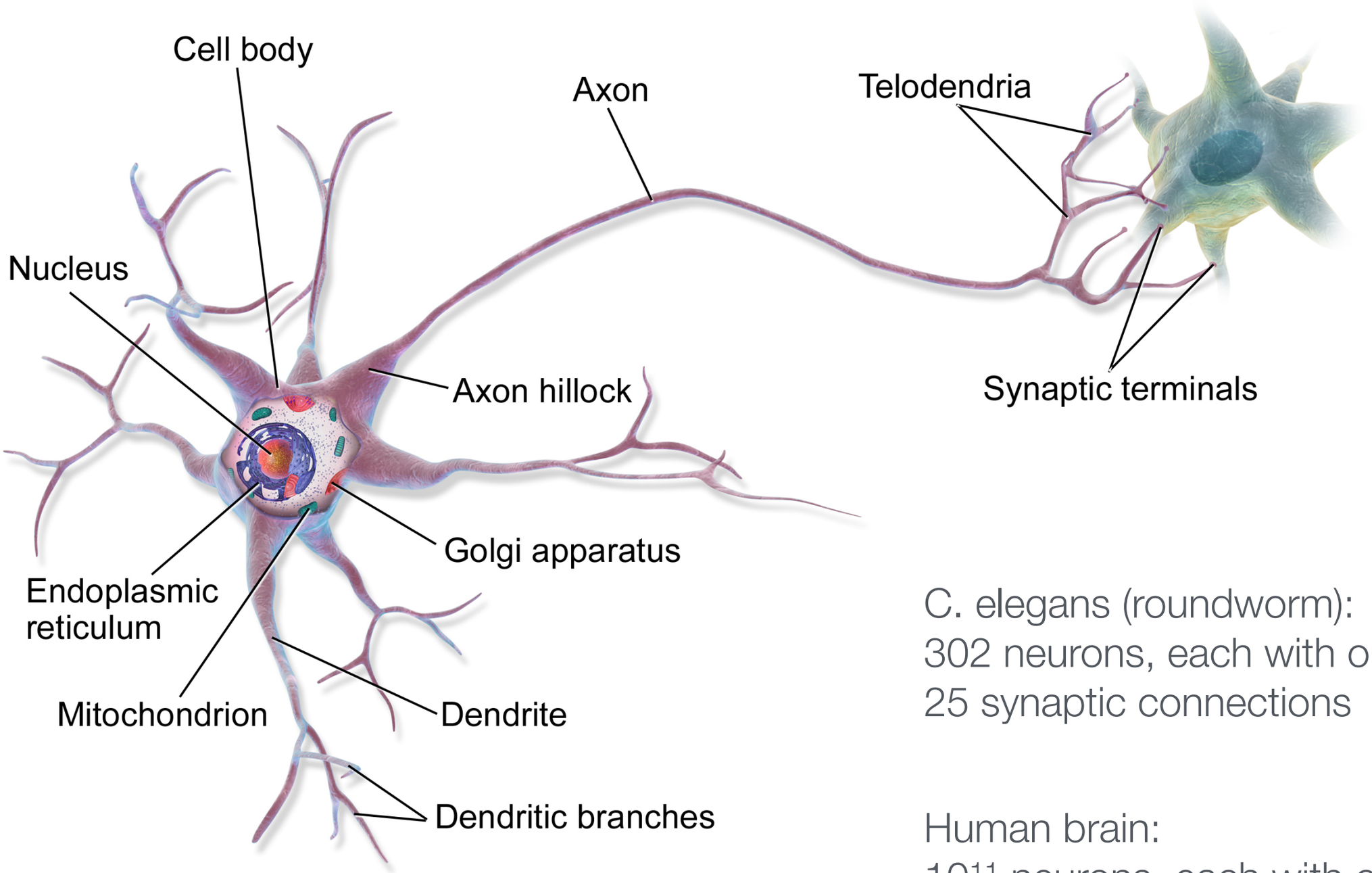
Shortcoming of the perceptron: it cannot learn the XOR function

Minsky, Papert, 1969



XOR: not linearly separable

The Biological Inspiration: the Neuron



C. elegans (roundworm):
302 neurons, each with on average 25 synaptic connections

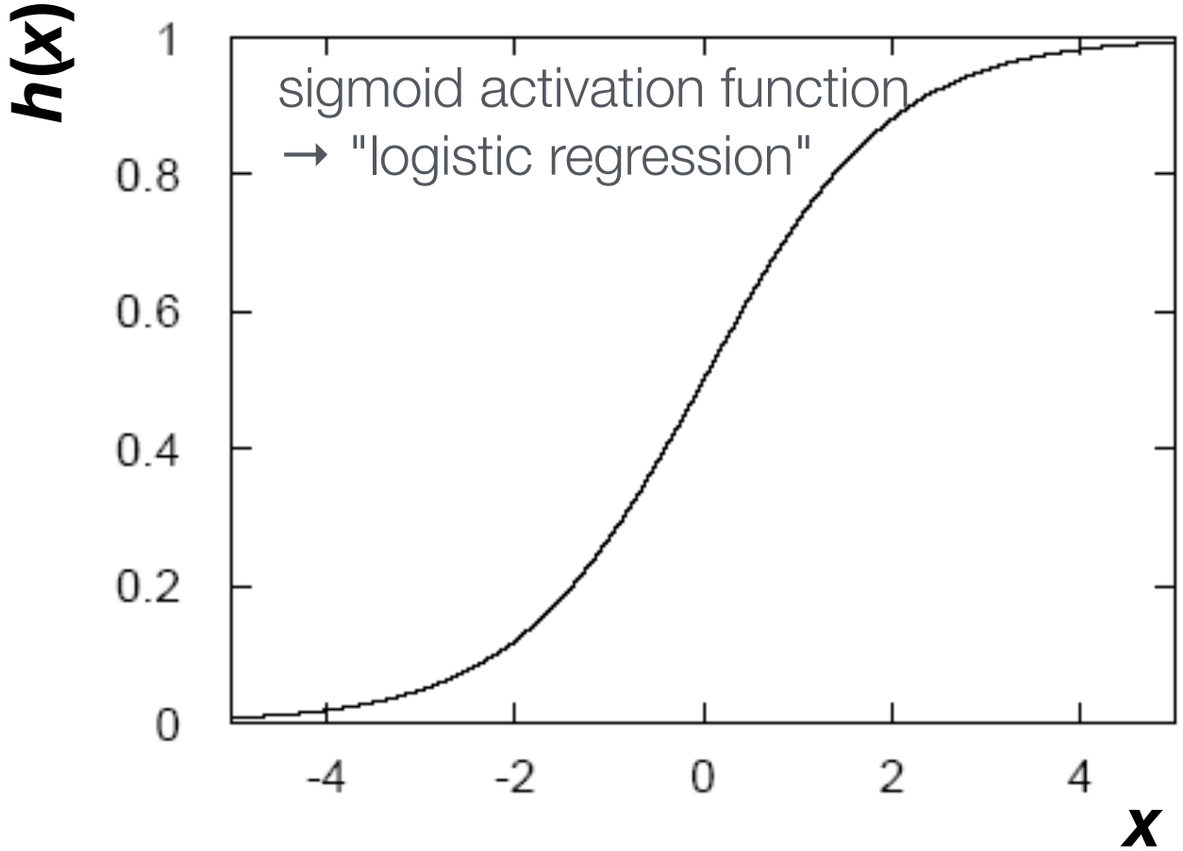
Human brain:
10^{11} neurons, each with on average 7000 synaptic connections

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

Non-linear Transfer / Activation Function

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

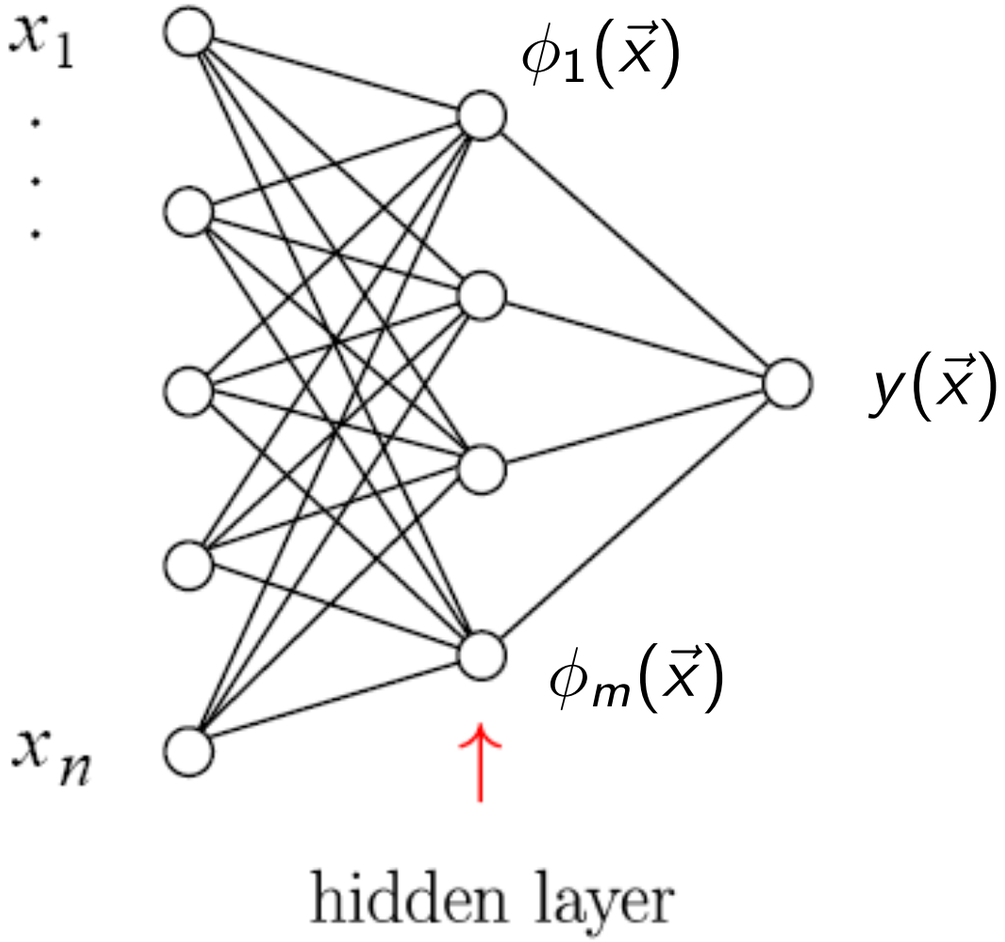
Examples for h: $\frac{1}{1+e^{-x}}$ (”sigmoid” or ”logistic” function), $\tanh x$



Needed in neural networks when feature space is not linearly separable

Neural net with linear activation functions is just a perceptron

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

Loss function (example):

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

Weights are determined by minimizing the loss function (also called error function)

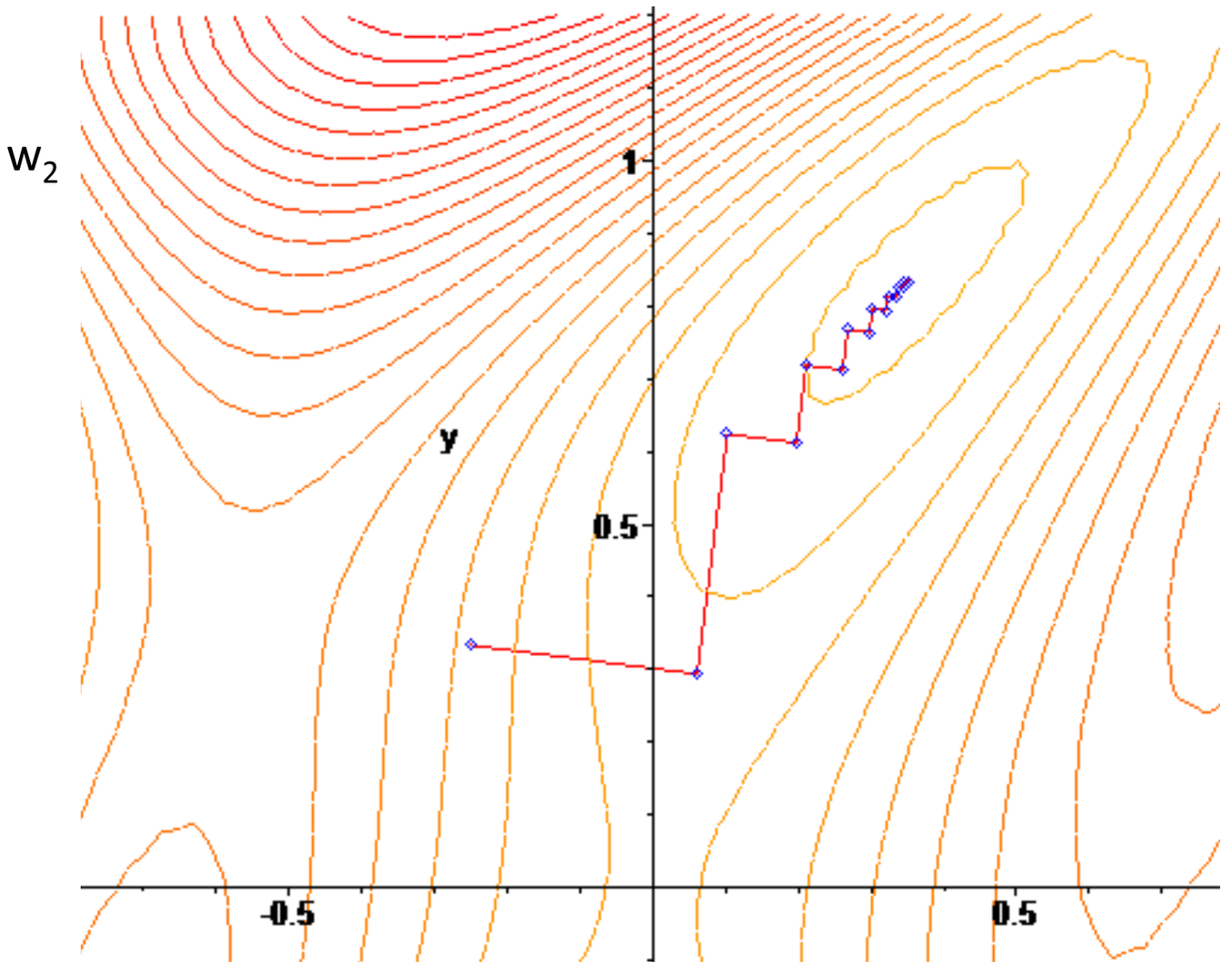
Back-propagation (I)

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

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

learning rate

Gradient descent:



Back-propagation (II)

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.

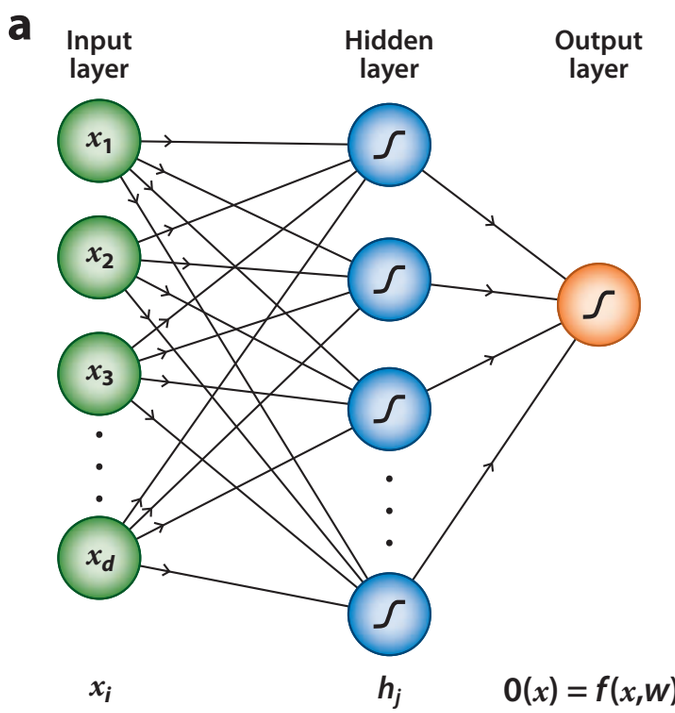
Weights from hidden layer to output:

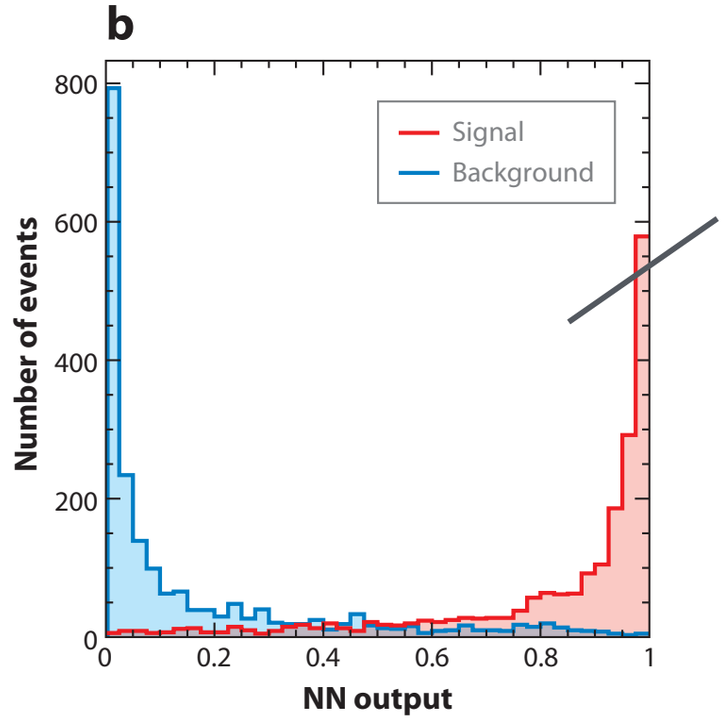
$$E_a = \frac{1}{2}(y_a - t_a)^2 \;\rightarrow\; \frac{\partial E_a}{\partial w_{1j}^{(2)}} = (y_a - t_a) h'(u(\vec{x}_a)) \frac{\partial u}{\partial w_{1j}^{(2)}} = (y_a - t_a) h'(u(\vec{x}_a)) \phi_j(\vec{x}_a)$$

Further application of the **chain rule** gives weights from input to hidden layer.

Neural Network Output and Decision Boundaries

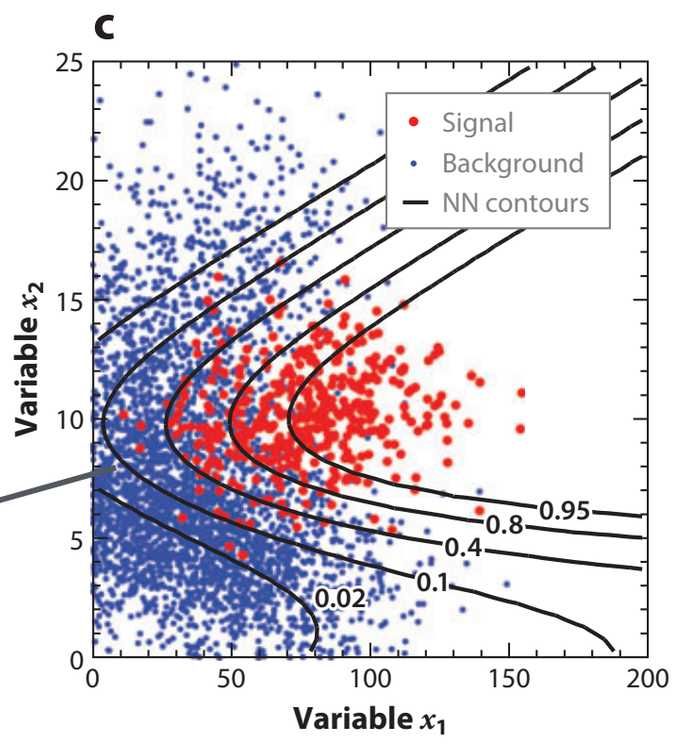
P. Bhat, Multivariate Analysis Methods in Particle Physics, inspirehep.net/record/879273

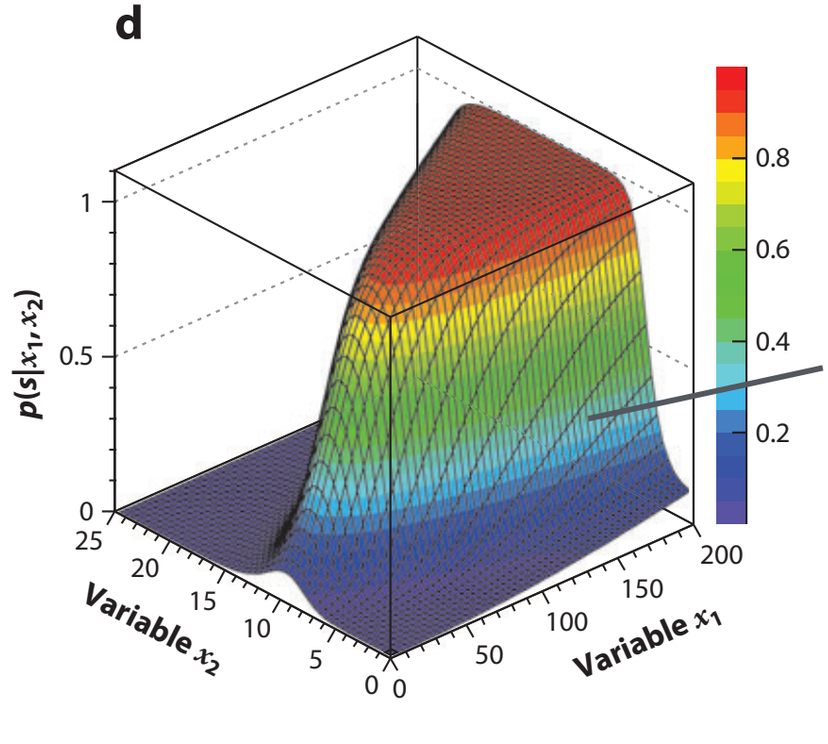




output of neural network

decision boundaries for different cuts on NN output



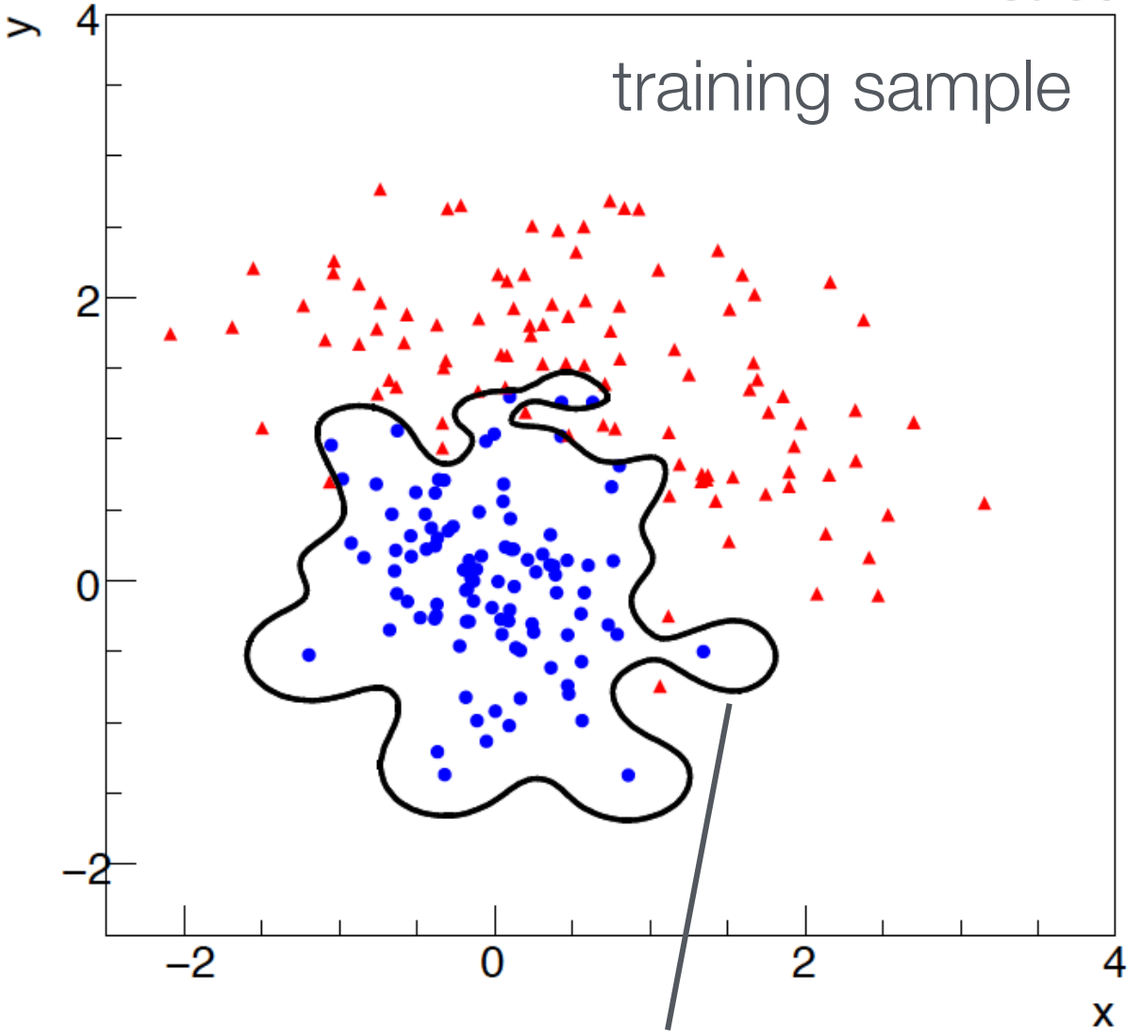


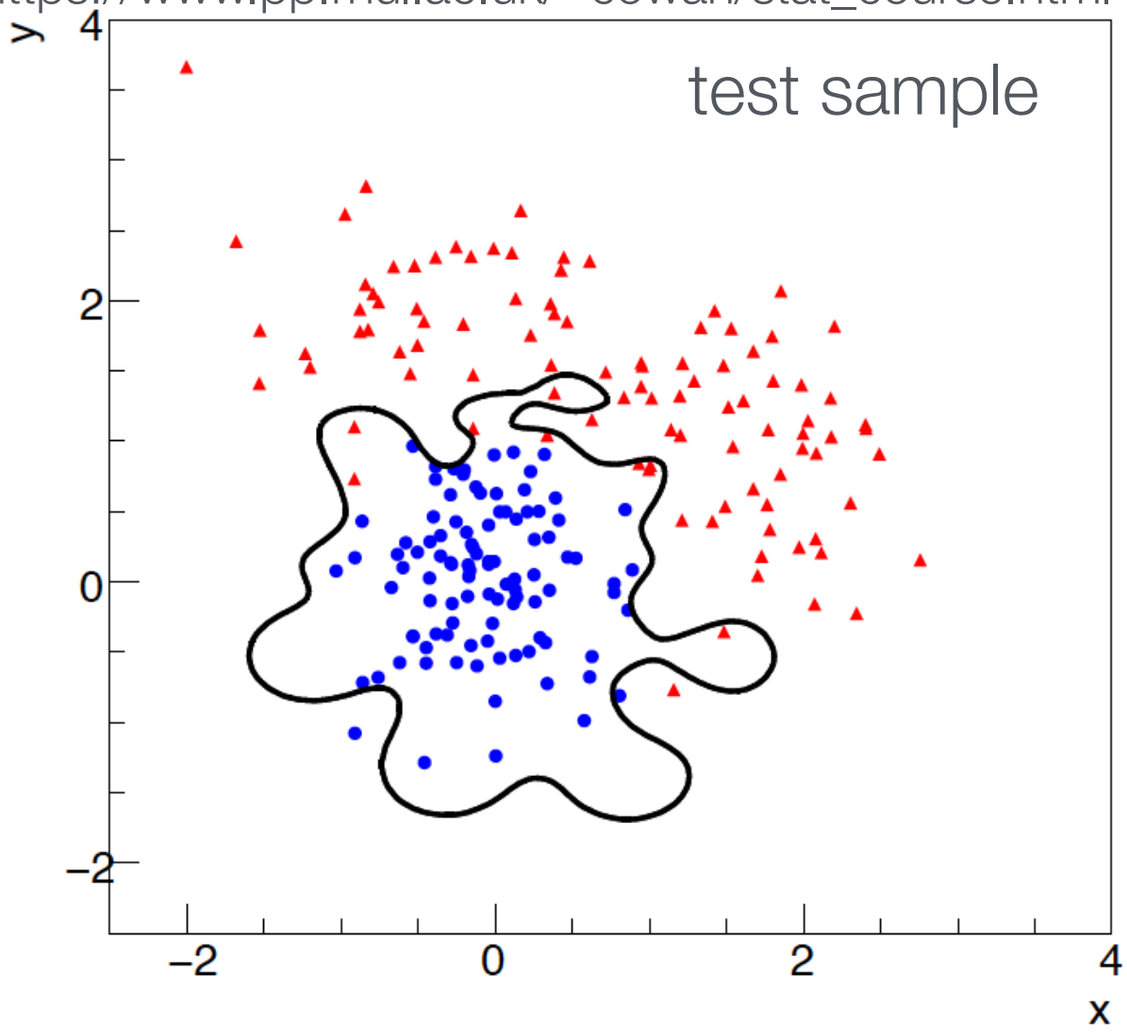
signal probability $p(s \mid x_1, x_2)$

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



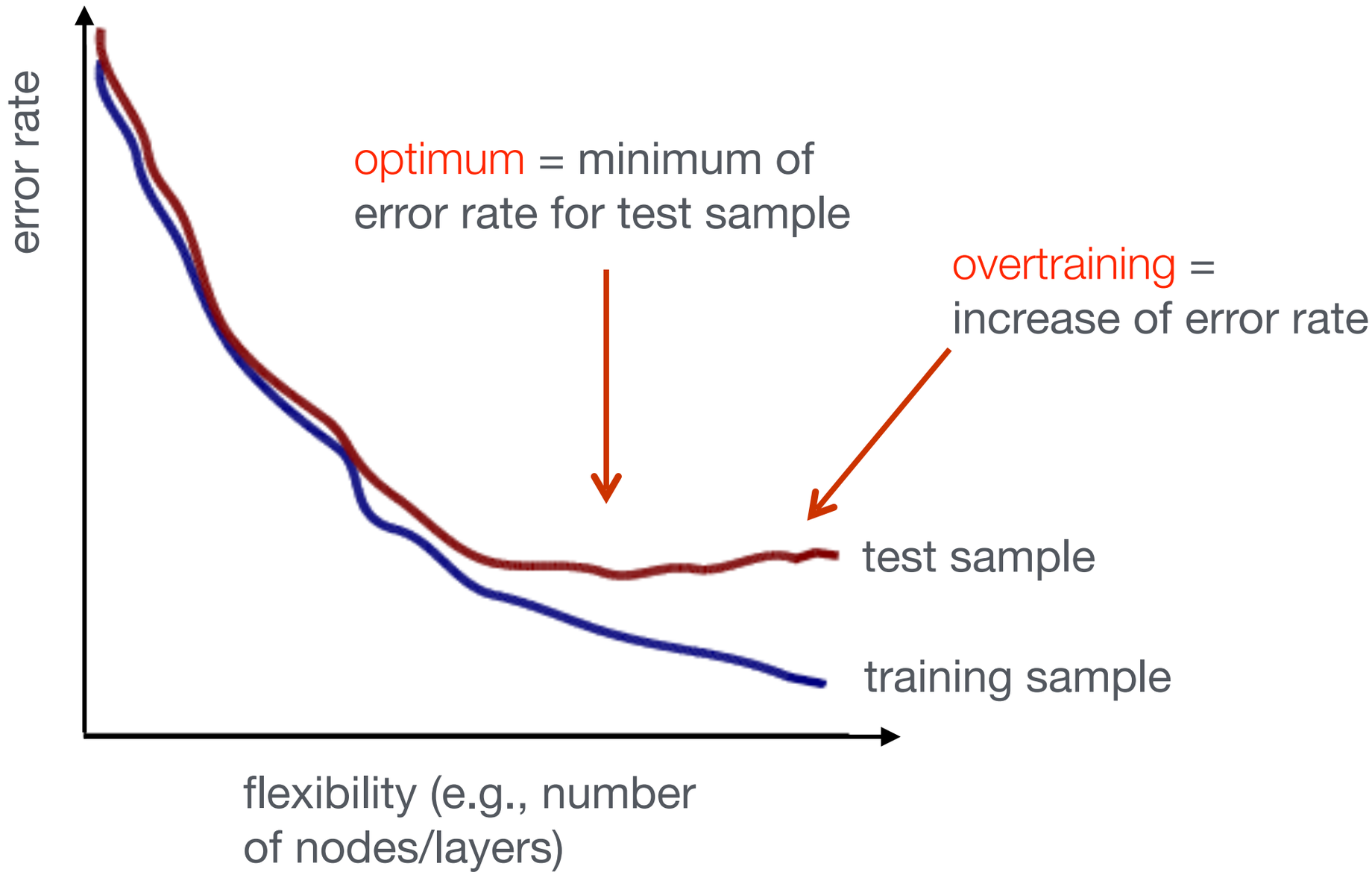


Network "learns" features that are merely statistical fluctuations in the training sample

Monitoring Overtraining

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

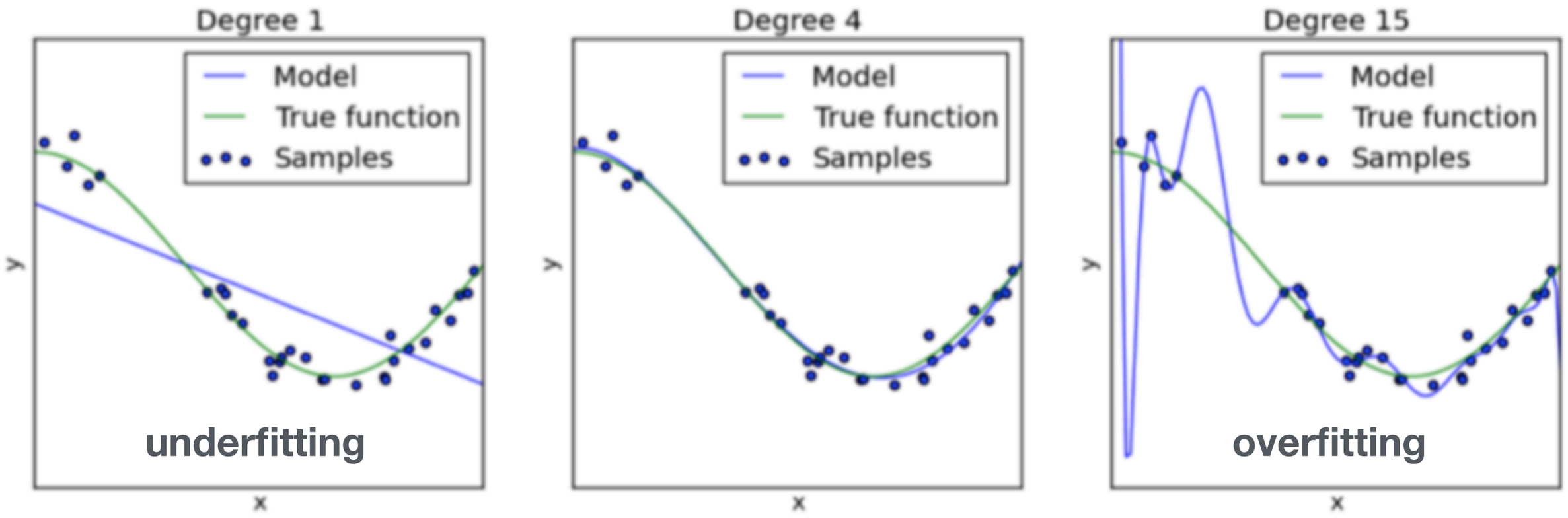
Monitor fraction of misclassified events (or loss function:)



Bias-Variance Tradeoff

Goal: generalization of training data

- Simple models (few parameters): danger of bias
  - 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")
- Complex models (many parameters): danger of overfitting
  - large variance of decision boundaries for different training samples



Deep Neural Networks

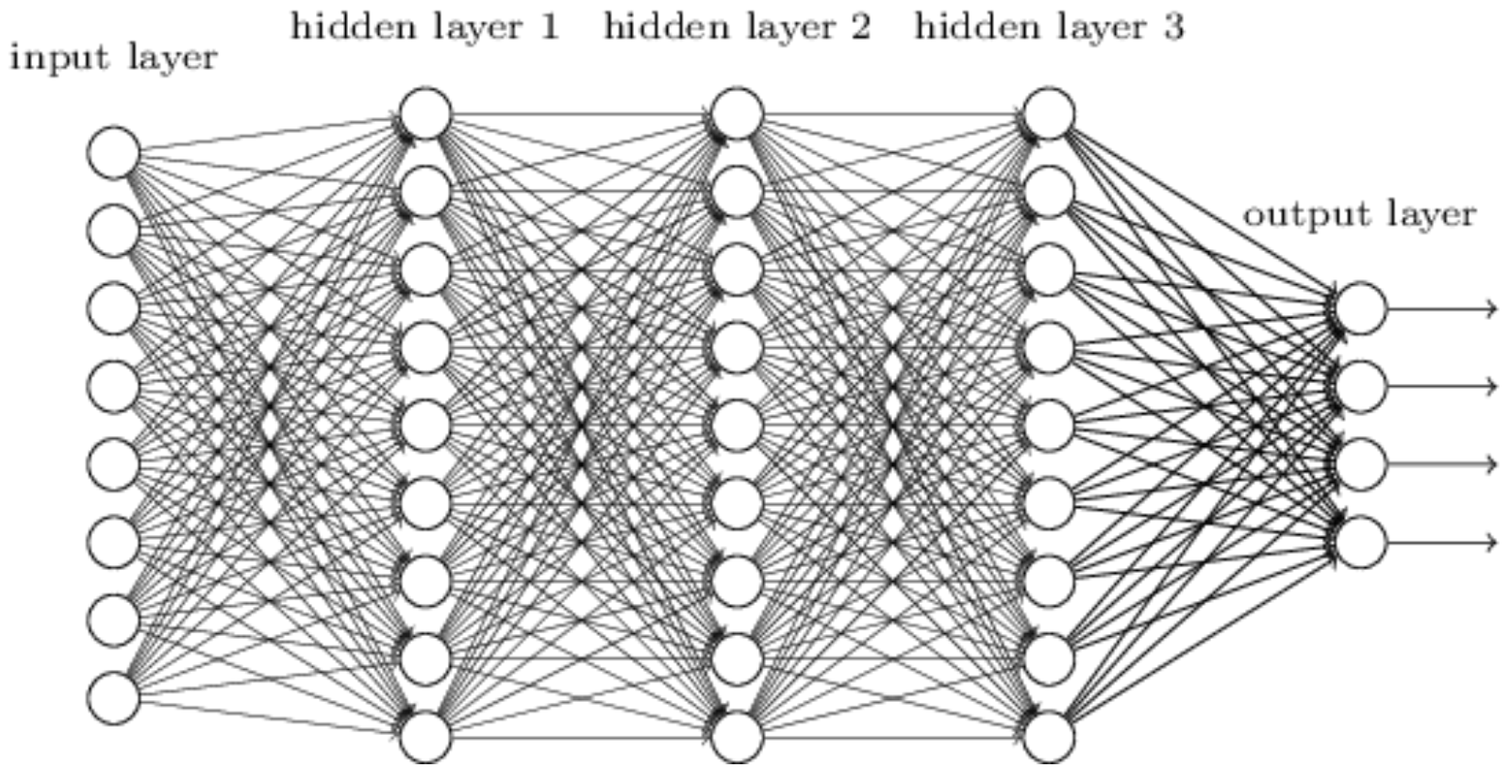
Deep networks: many hidden layers with large number of neurons

Challenges

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

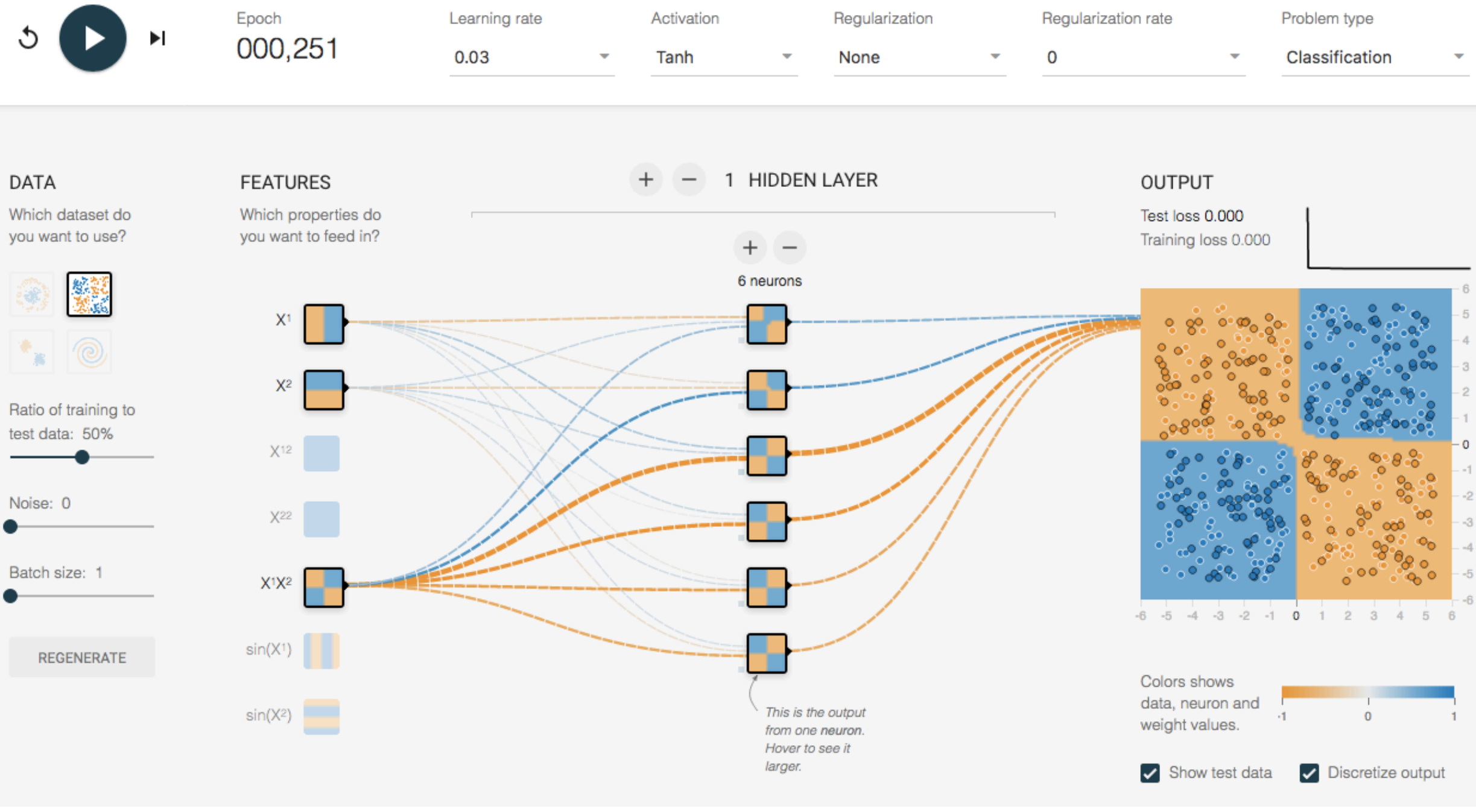
Big progress in recent years

- 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

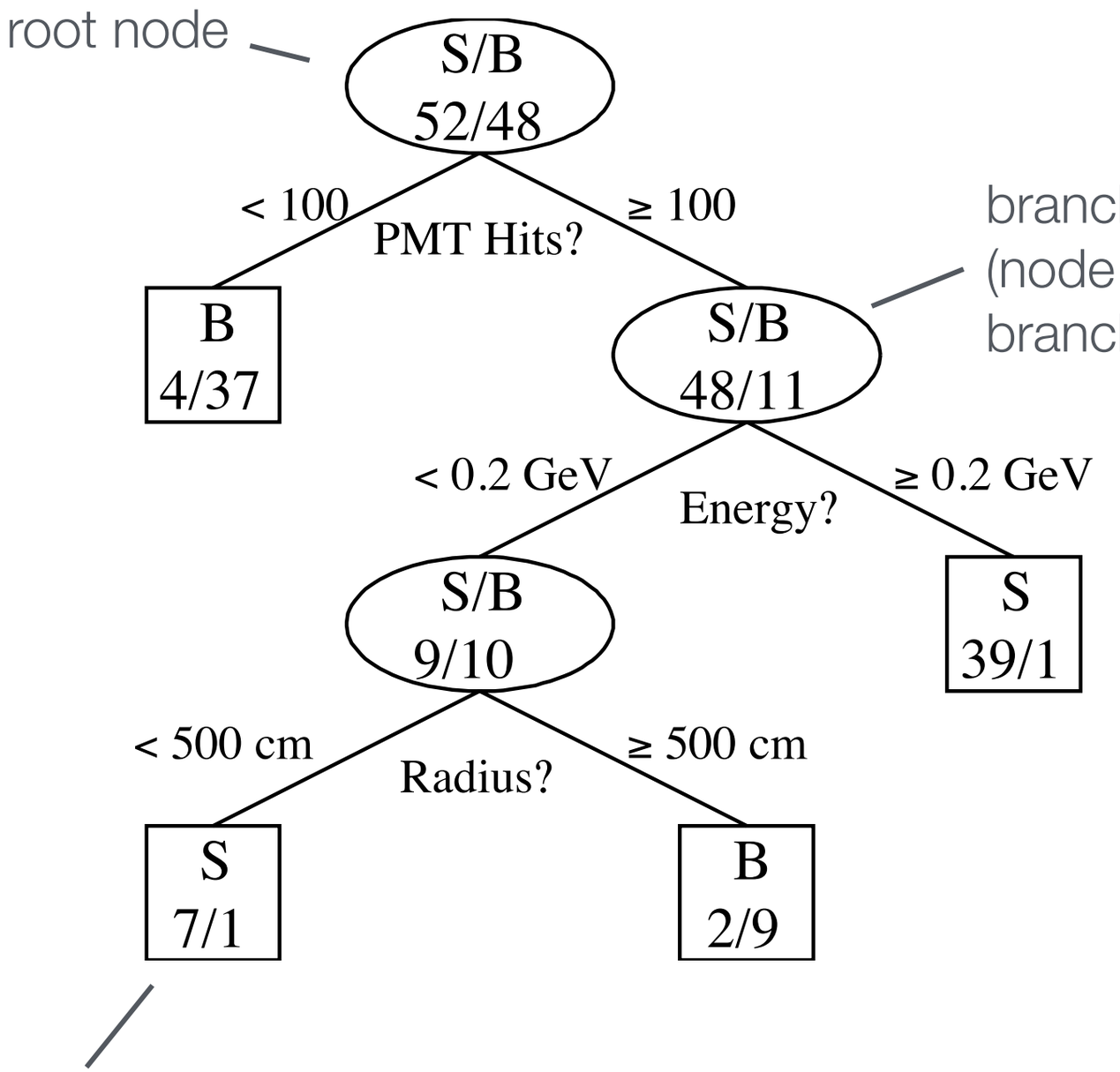
Fun with Neural Nets in the Browser

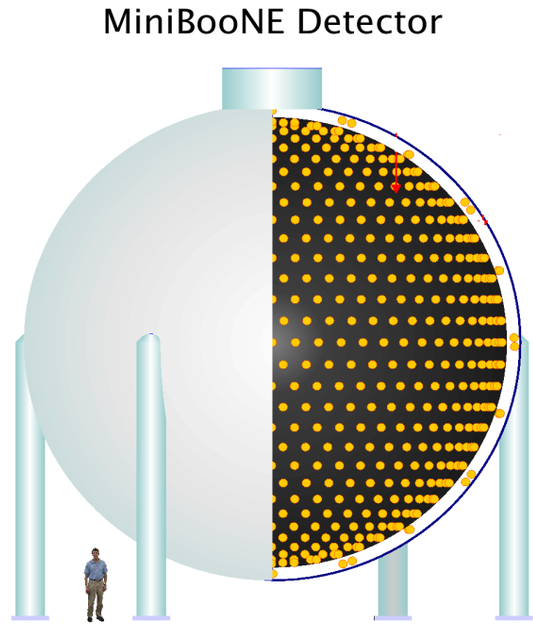


http://playground.tensorflow.org

Decision Trees (I)

arXiv:physics/0508045v1



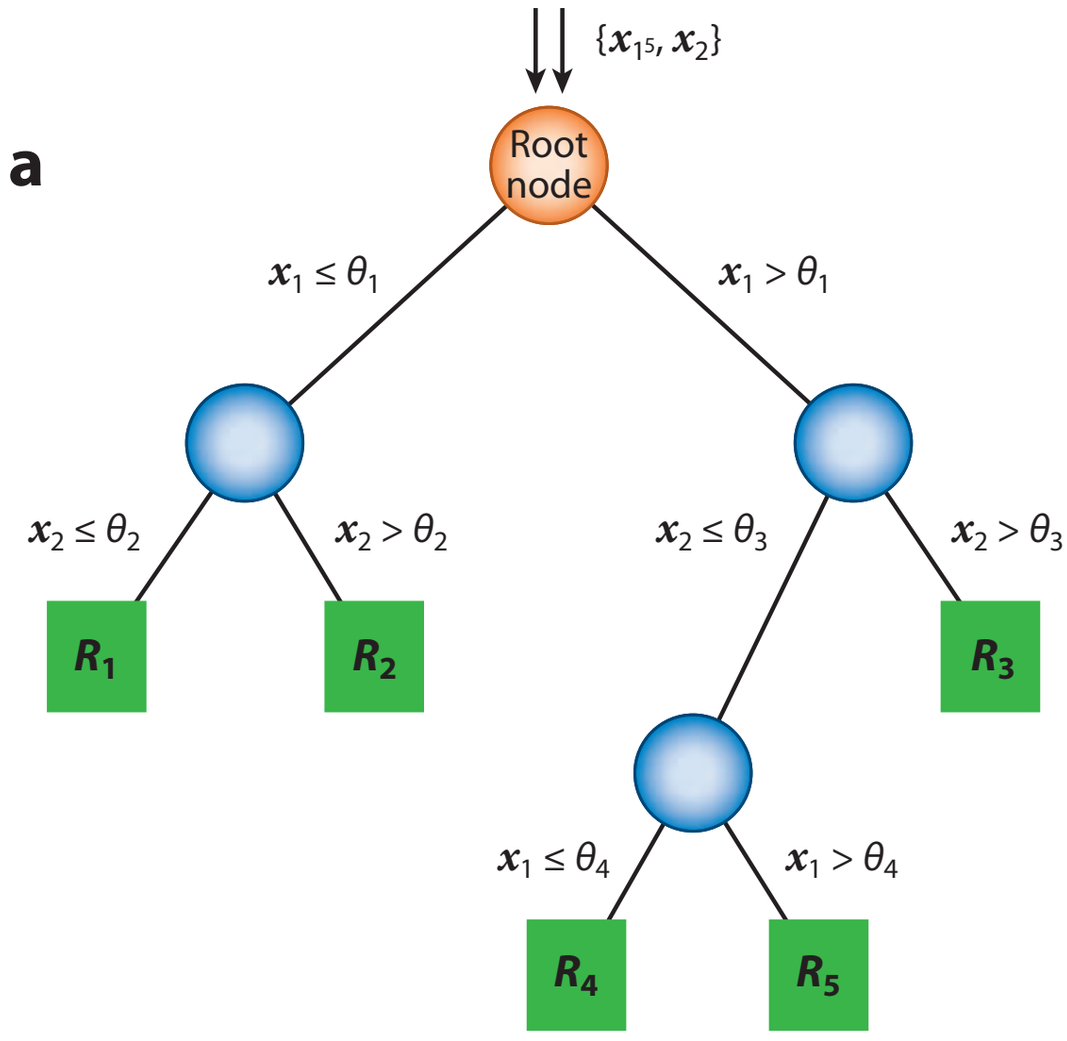


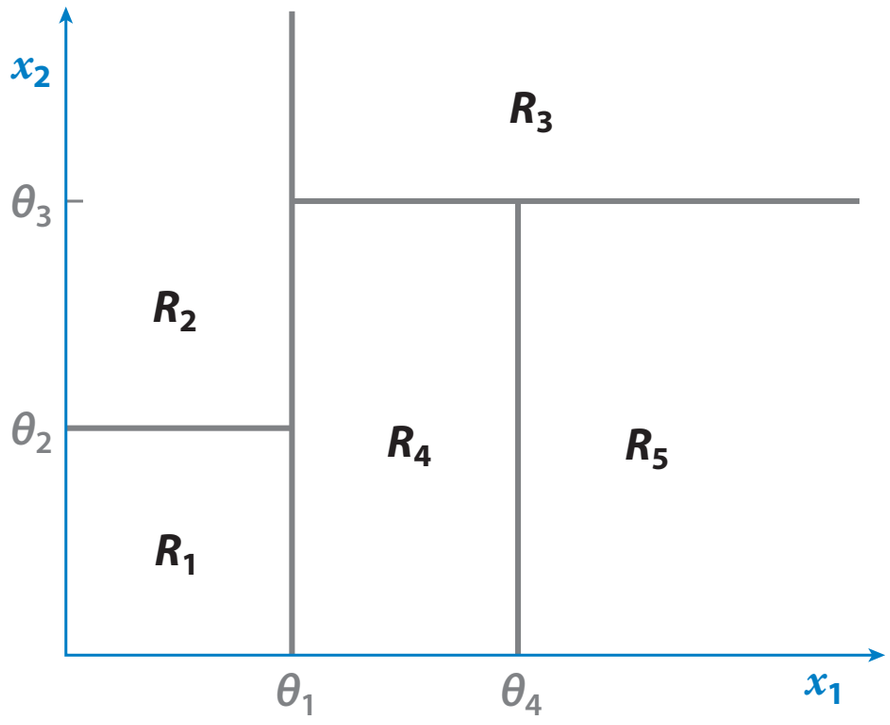
MiniBooNE: 1520 photomultiplier signals, goal: separation of ν_e from ν_μ events

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:

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

$w_i = \text{weight of event } i$

[usefulness of weights will become apparent soon]

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

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

Single Decision Trees: Pros and Cons

Pros:

- Requires little data preparation
- Can use continuous and categorical inputs

Cons:

- Danger of overfitting training data
- Sensitive to fluctuations in the training data
- Hard to find global optimum
- When to stop splitting?

Ensemble Methods: Combine Weak Learners

- **B**ootstrap **Agg**regating (Bagging)
  - Sample training data (with replacement) and train a separate model on each of the derived training sets
  - Classify example with majority vote, or compute average output from each tree as model output

$$y(\vec{x}) = \frac{1}{N_{\text{trees}}} \sum_{i=1}^{N_{trees}} y_i(\vec{x})$$

- Boosting
  - Train *N* models in sequence, giving more weight to examples not correctly classified by previous model
  - Take weighted average to classify examples

$$y(\vec{x}) = \frac{\sum_{i=1}^{N_{\text{trees}}} \alpha_i y_i(\vec{x})}{\sum_{i=1}^{N_{\text{trees}}} \alpha_i}$$

Random Forests

- Use bagging to select random example subset

- Train a tree, but only use random subset of features at each split
  - this reduced the correlation between different trees
  - makes the decision more robust to missing data

Boosted Decision Trees: Idea

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

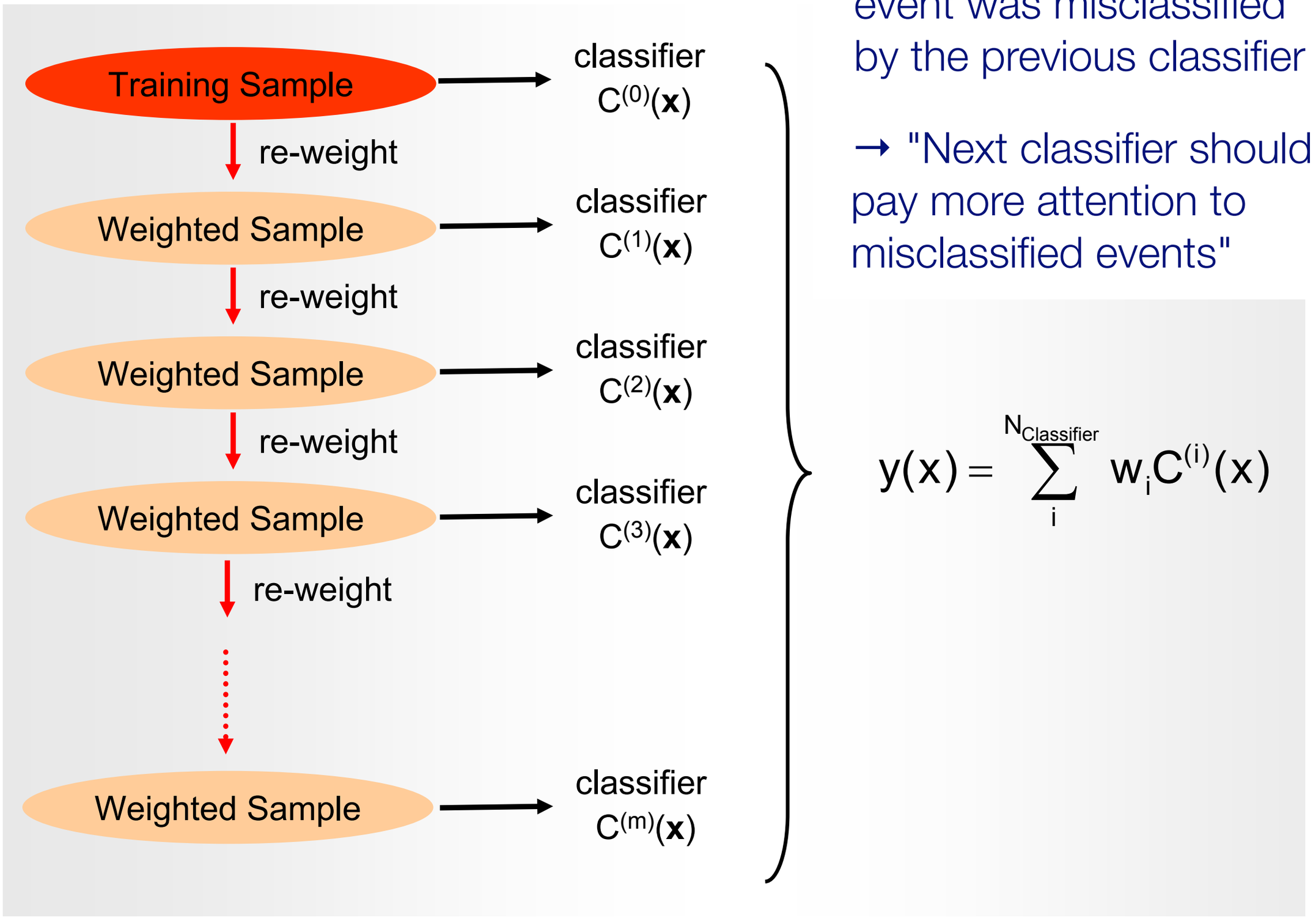
Solution: boosting

- 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)

Boosted Decision Trees: Idea



Weight is increased if event was misclassified by the previous classifier

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

H. Voss, Lecture: Graduierten-Kolleg, http://tmva.sourceforge.net/talks.shtml

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

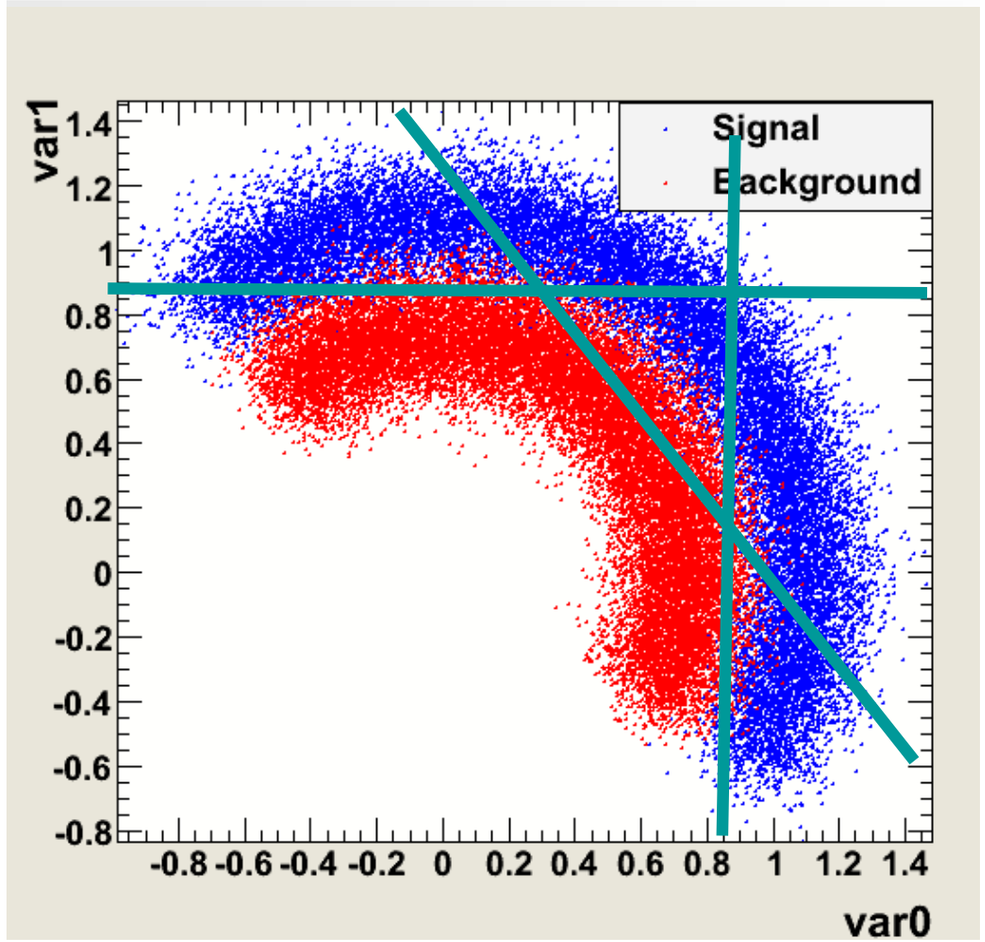
H. Voss, Multivariate Data Analysis and Machine Learning in High Energy Physics
http://tmva.sourceforge.net/talks.shtml

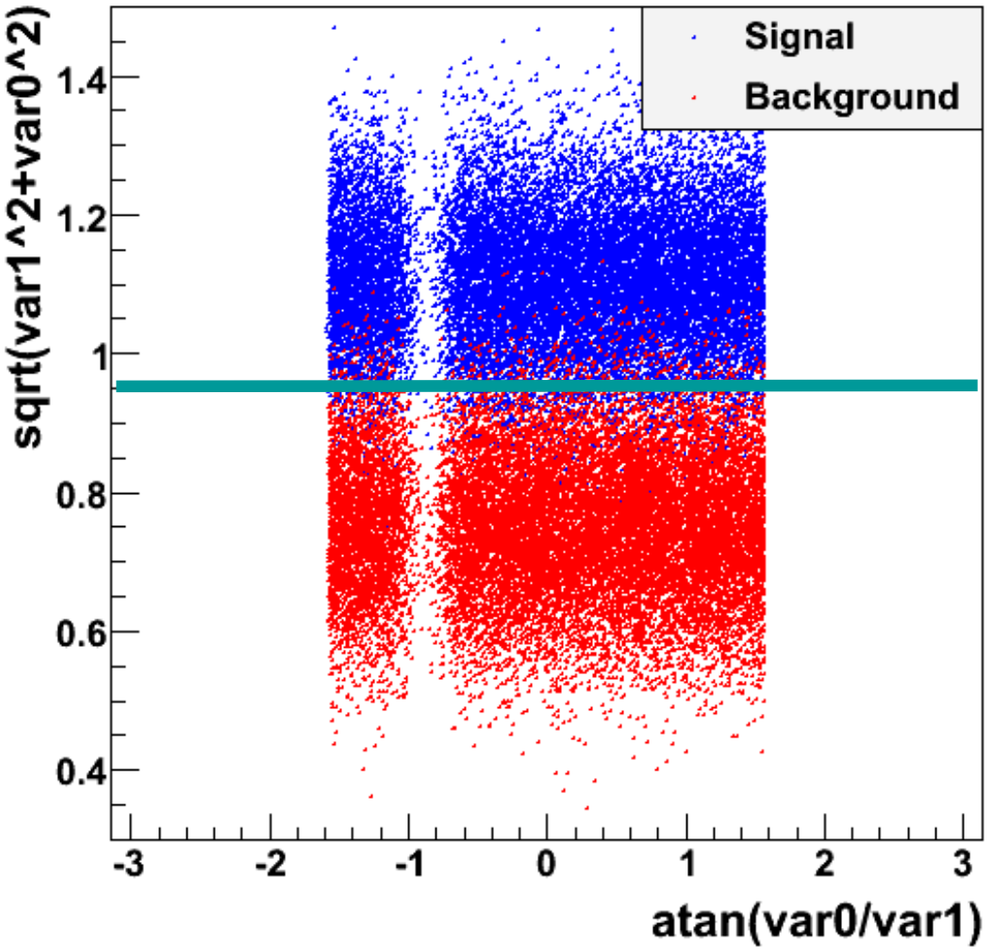
Example of a feature transformation

$$\mathrm{var}0' = \sqrt{\mathrm{var}0^2 + \mathrm{var}1^2}$$

$$\mathrm{var}1' = \mathrm{atan}\left(\frac{\mathrm{var}0}{\mathrm{var}1}\right)$$

In this case a linear classifier works well after feature transformation





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		🙂	🙂	😐	🙂	🙂	☹	☹	☹	☹

Contents

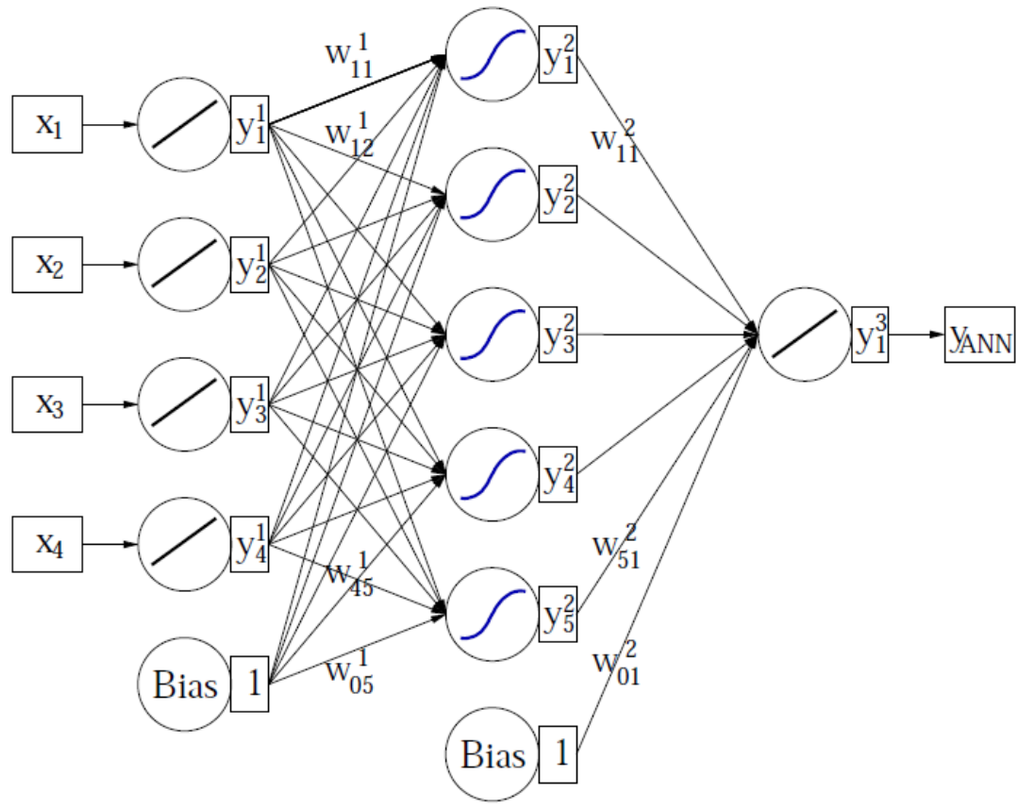
Universal Approximation Theorem

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

"A feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of $\mathbb{R}^n$."

One of the first versions of the theorem was proved by George Cybenko in 1989 for sigmoid activation functions

The theorem does not touch upon the algorithmic learnability of those parameters

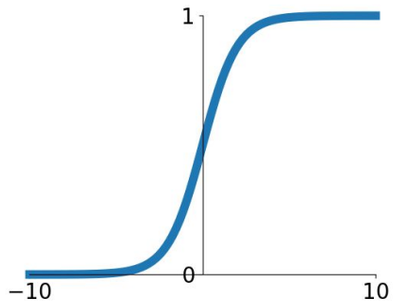


Activation Functions

http://cs231n.stanford.edu/slides

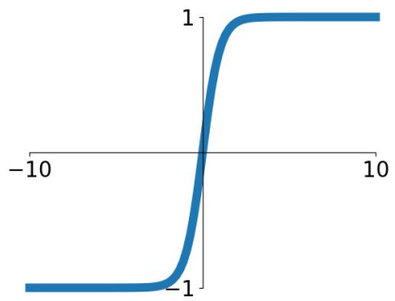
Sigmoid

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



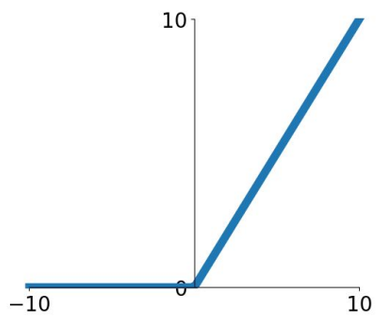
tanh

$$\tanh(x)$$



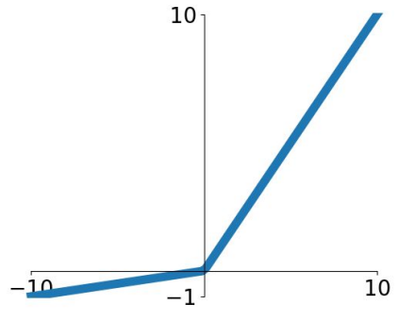
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

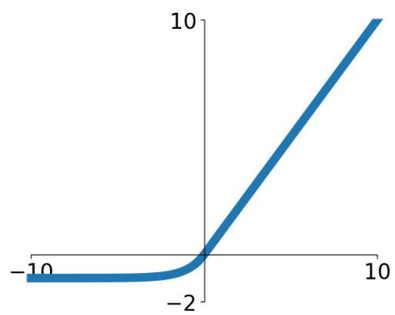


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

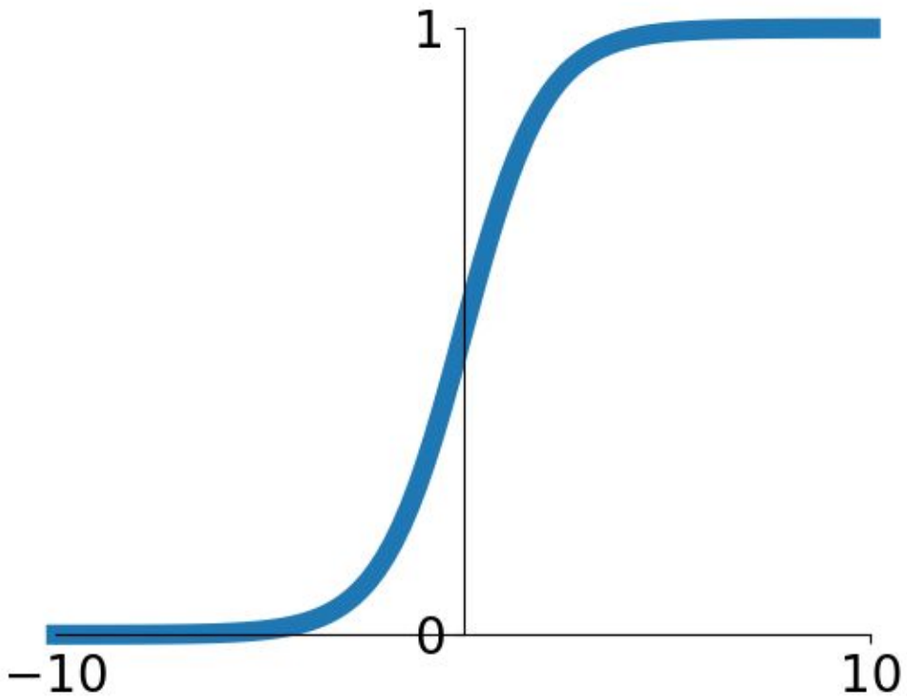
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid activation function

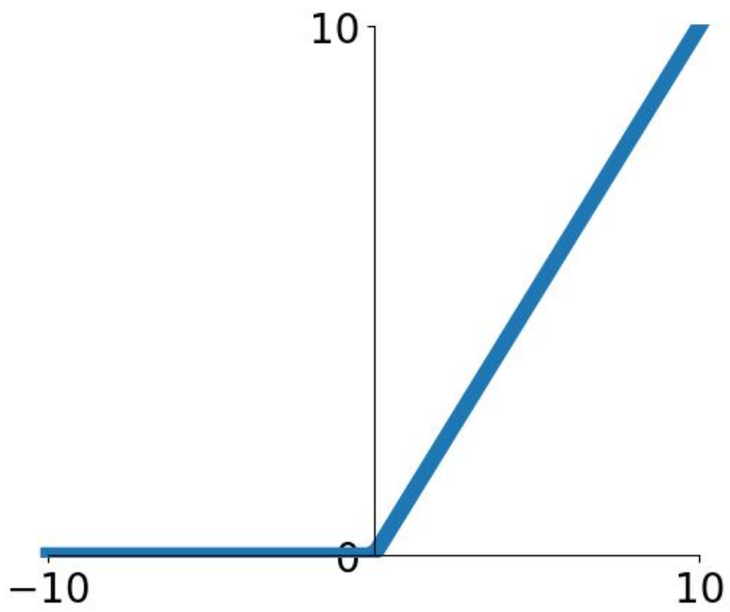
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Sigmoid

- Saturated neurons “kill” the gradients
- Sigmoid outputs are not zero-centered
- exp() is a bit compute expensive

ReLU



ReLU
(Rectified Linear Unit)

$$f(x) = \max(0, x)$$

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

But: gradient vanishes for $x < 0$

Practical Tips

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid

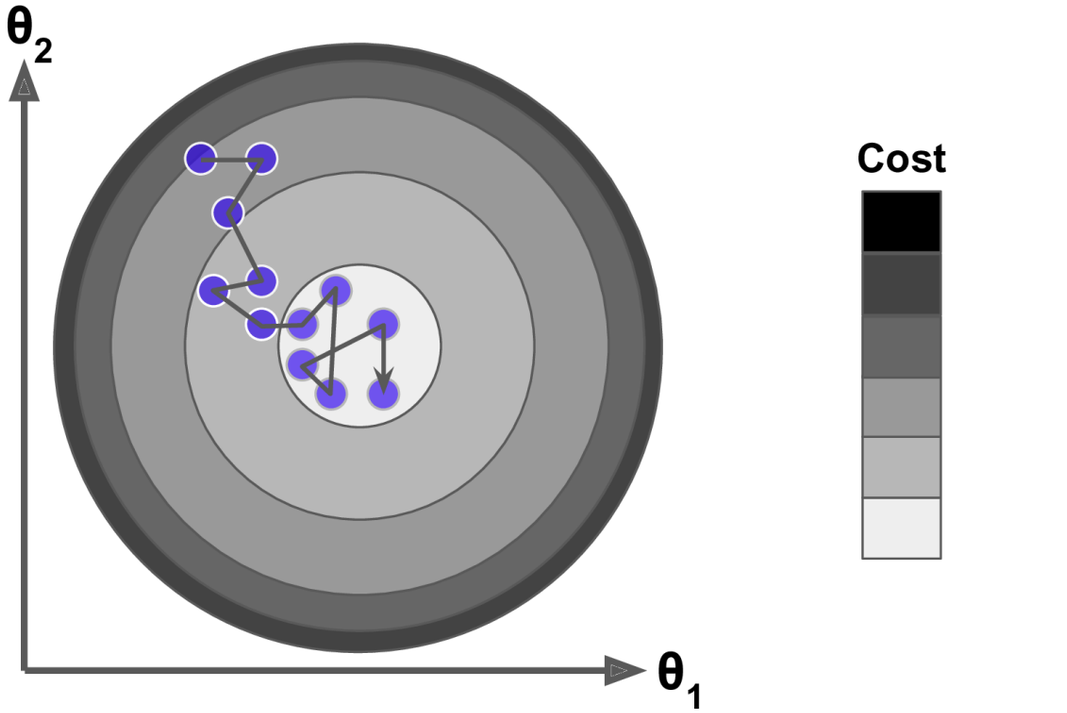
Fei-Fei Li & Justin Johnson & Serena Yeung,
Convolutional Neural Networks for Visual Recognition,
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

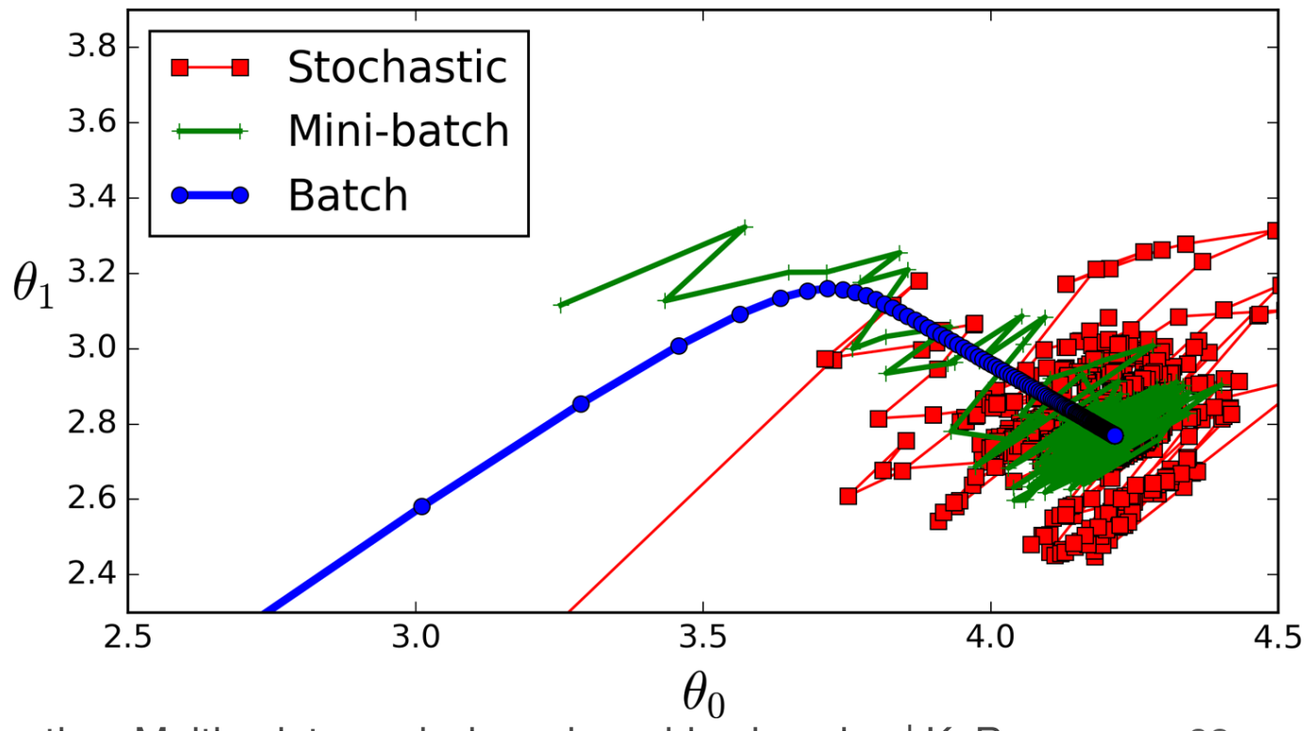
Gradient Descent

Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

- Stochastic gradient descent
  - just uses one training event at a time
  - fast, but quite irregular approach to the minimum
  - can help escape local minima
  - one can decrease learning rate to settle at the minimum ("simulated annealing")
- Batch gradient descent
  - use entire training sample to calculate gradient of loss function
  - computationally expensive
- Mini-batch gradient descent
  - calculate gradient for a random sub-sample of the training set

Stochastic Gradient Descent





Multiclass Classification: Softmax Output Layer

Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

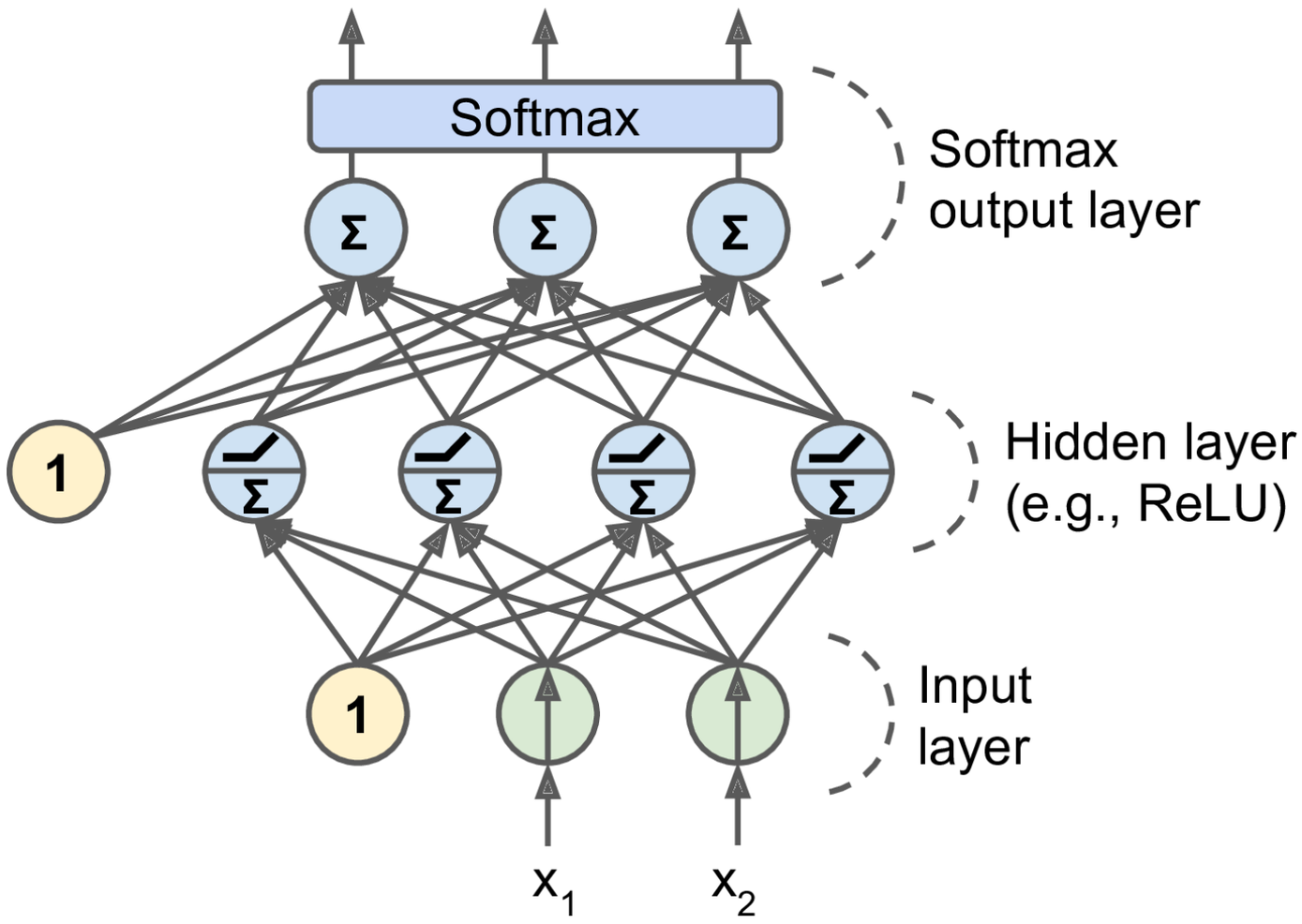
Classification with *n* exclusive categories (*y* = 1, 2, …, *n*):

Score of category *i* for a given input:

$$s_i = \vec{w}_i^{\mathsf{T}} \vec{h} + b_i$$

Softmax function:

$$P_j := P(y = j) = \frac{e^{s_j}}{\sum_{i=1}^{n} e^{s_i}}$$



Translate *n* dimensional vector of "scores" (arbitrary real values) to *n* dimensional vector of values P_j with

$$0 \leq P_j \leq 1, \quad \sum_j P_j = 1$$

Values P_j interpreted as probabilities (cf. *logistic regression* for binary classification).

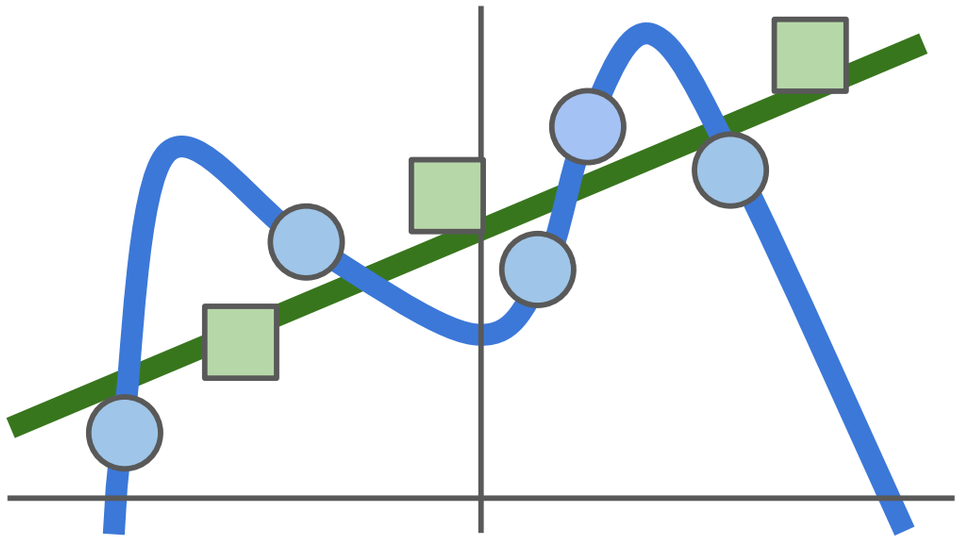
Regularization: Avoid Overfitting

http://cs231n.stanford.edu/slides

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

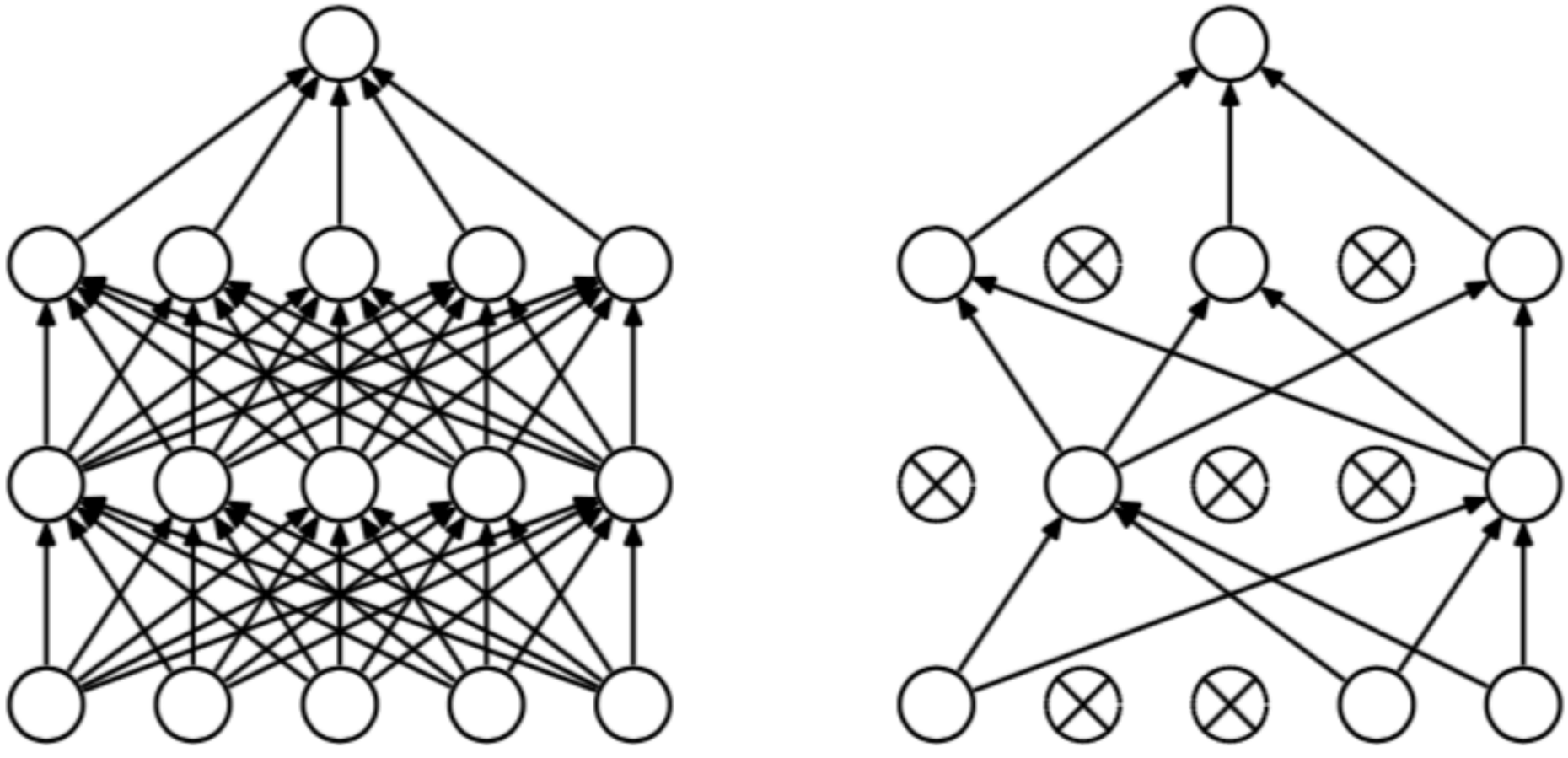
In common use:

L2 regularization $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

Another Approach to Prevent Overfitting: Dropout

- Randomly remove nodes during training
- Avoid co-adaptation of nodes



(a) Standard Neural Net

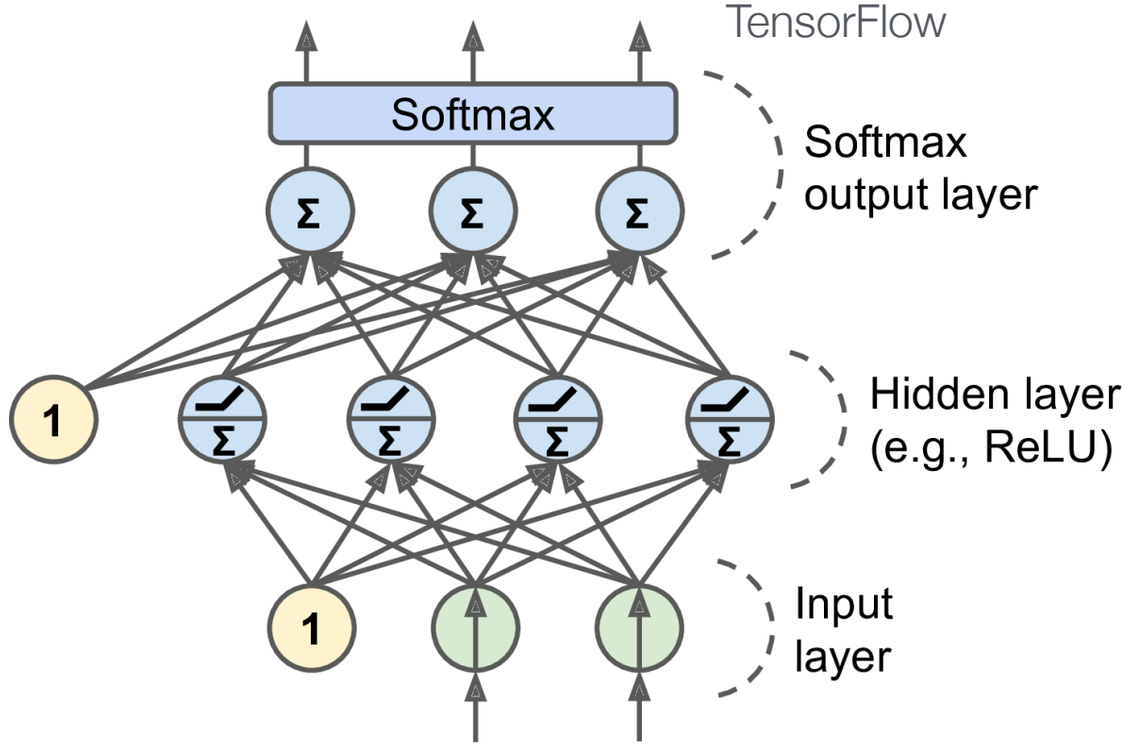
(b) After applying dropout.

Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"

Xavier and He Initialization

- Initial weights determine speed of convergence and whether algorithm converges at all
- Xavier Glorot and Yoshua Bengio
  - Paper "Understanding the Difficulty of Training Deep Feedforward Neural Networks"
  - Idea: Variance of the outputs of each layer to be equal to the variance of its inputs

Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow



Neuron with n_{in} inputs connected to n_{out} neurons in the next layer

Activation function	Uniform distribution $[-r, r]$	Normal distribution ($\mu = 0$)
Logistic	$r = \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}$
tanh	$r = 4\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}$
ReLU (and variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}$

Concluding Remarks

Which Method to Use?

M. Kagan,
https://indico.cern.ch/event/619370/

- Linear model
- Nearest Neighbors
- (Deep?) Neural network
- Decision tree ensemble
- Support vector machine
- ...

No Free Lunch Theoreme

David Wolpert, William Macready, 1997

https://de.wikipedia.org/wiki/No-free-Lunch-Theoreme

"Folkloric" version:

> Any two optimization algorithms are equivalent when their performance is averaged across all possible problems

In other words:

> If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems

How do we pay for our lunch?

> Domain knowledge and/or biases in the choice of the algorithms (link)

> Relevance for practical problem?

Practical Advice – Which Algorithm to Choose?

M. Kagan, https://indico.cern.ch/event/619370/

From Kaggle competitions:

Structured data: "High level" features that have meaning

- feature engineering + decision trees
- Random forests
- XGBoost

Unstructured data: "Low level" features, no individual meaning

- deep neural networks
- e.g. image classification: convolutional NN