# Statistical Methods in Particle Physics

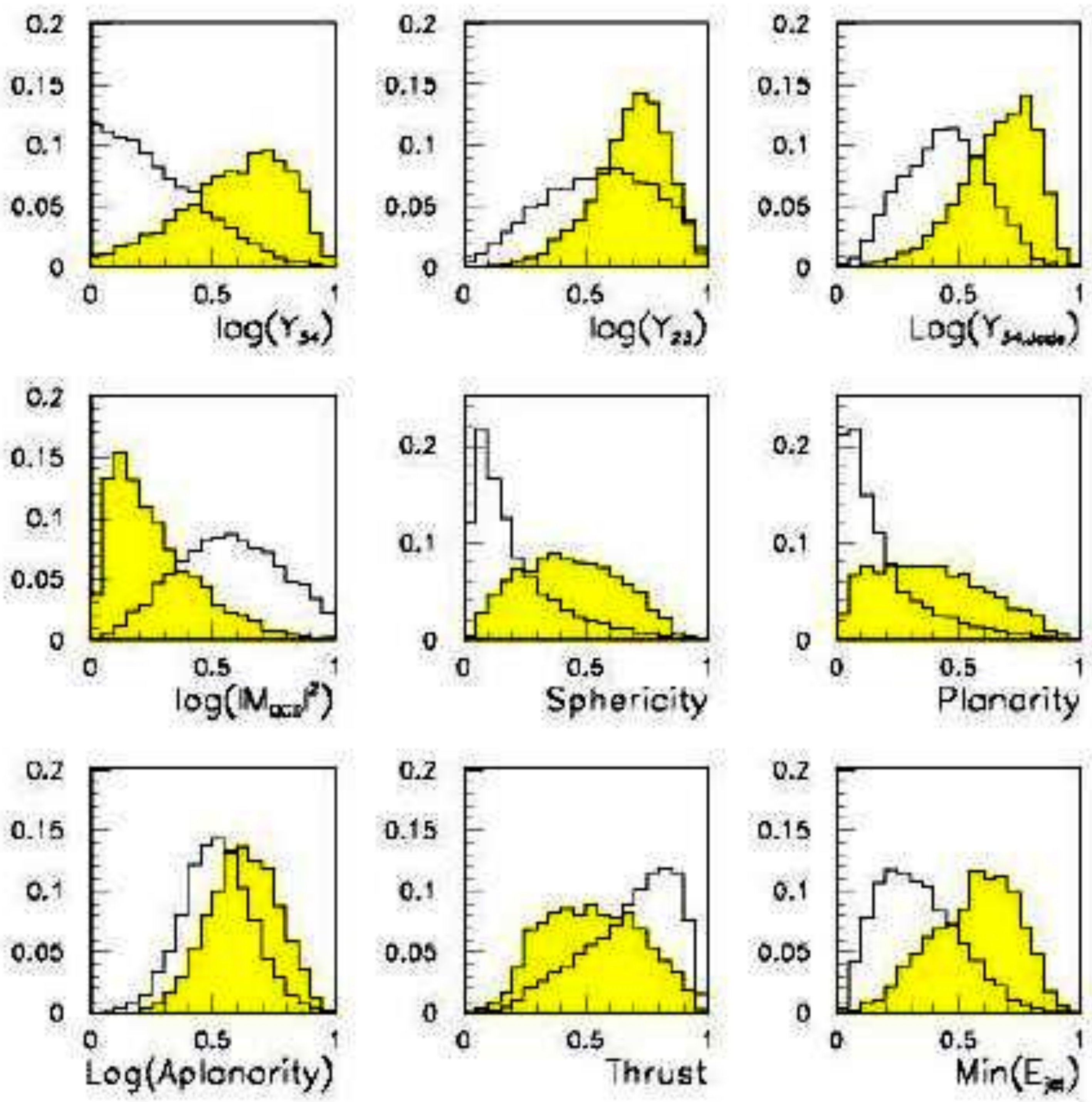
## 9. Machine Learning

Heidelberg University, WS 2020/21

Klaus Reygers (lectures)
Rainer Stamen, Martin Völkl (tutorials)

# Multivariate analysis: An early example from particle physics

G. Cowan, Lecture on Statistical data analysis



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

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

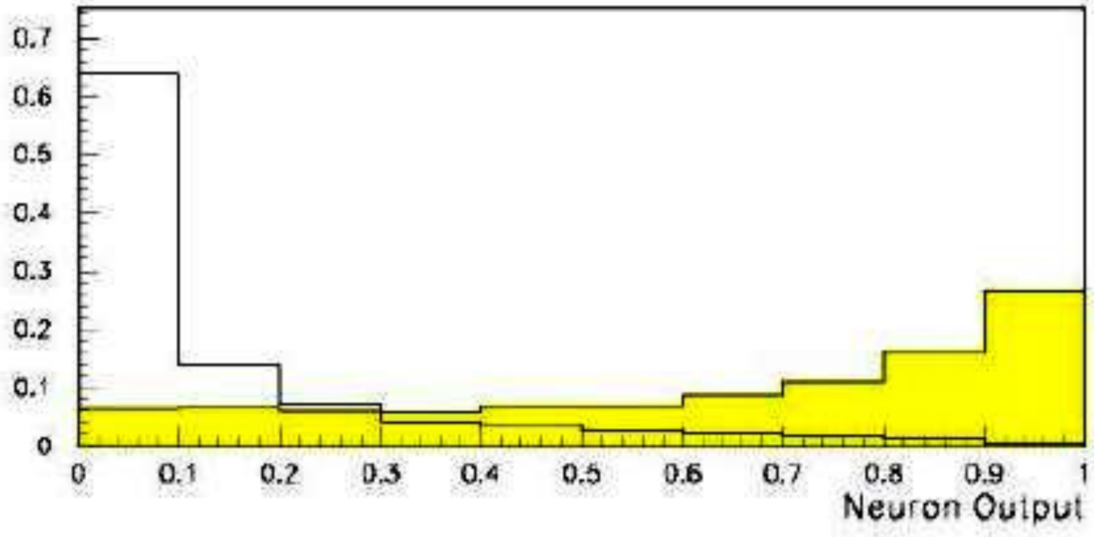
often 4 well separated hadron jets

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:

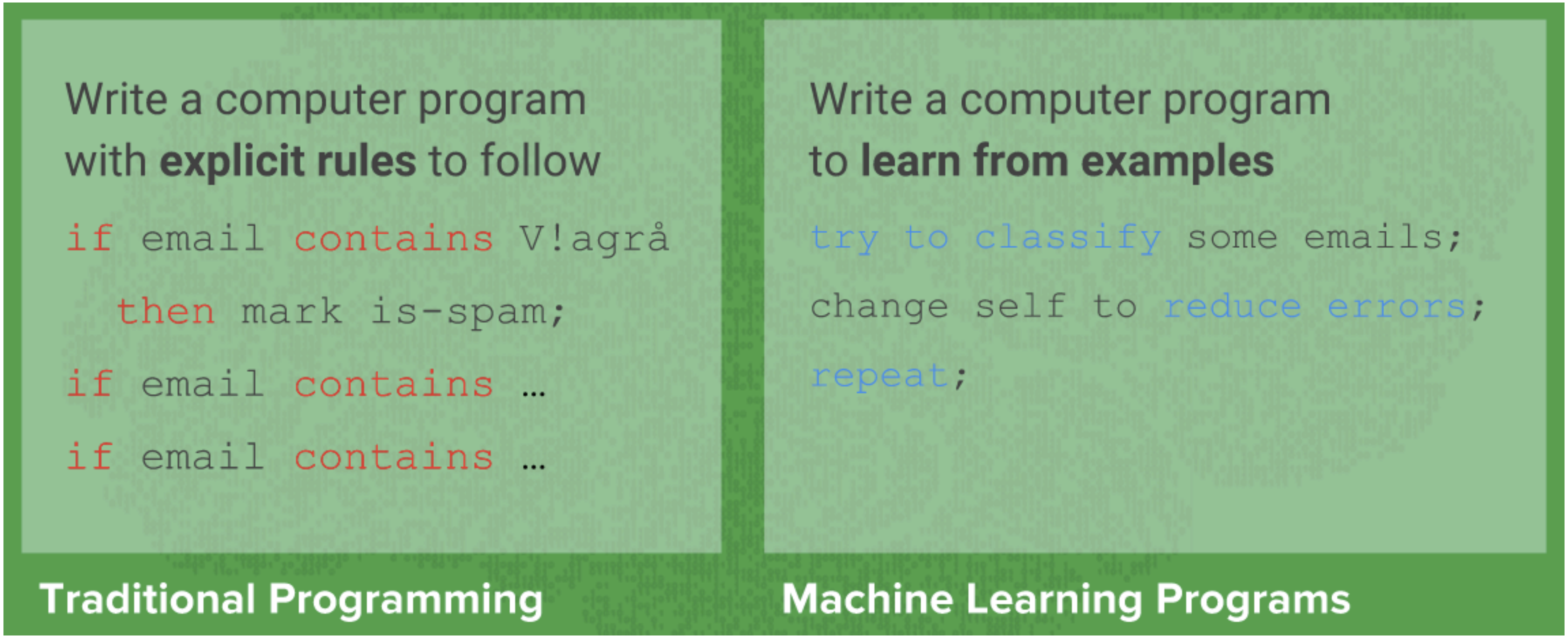


# Machine learning

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

Example: spam detection

J. Mayes, Machine learning 101



Write a computer program with **explicit rules** to follow

```
if email contains V!agrå
  then mark is-spam;
if email contains …
if email contains …
```

**Traditional Programming**

Write a computer program to **learn from examples**

```
try to classify some emails;
change self to reduce errors;
repeat;
```

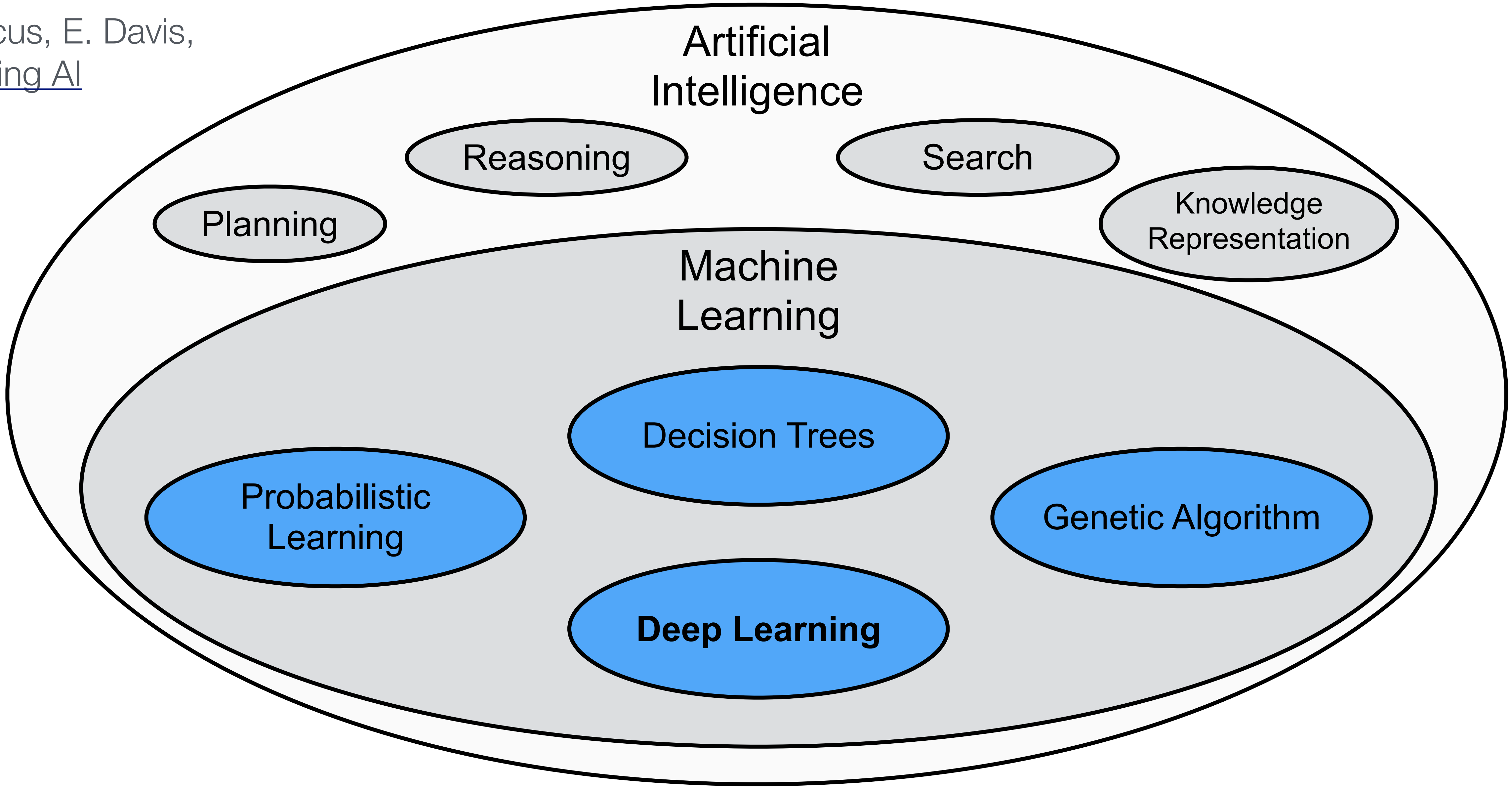
**Machine Learning Programs**

Manual feature engineering vs. automatic feature detection

# AI, ML, and DL

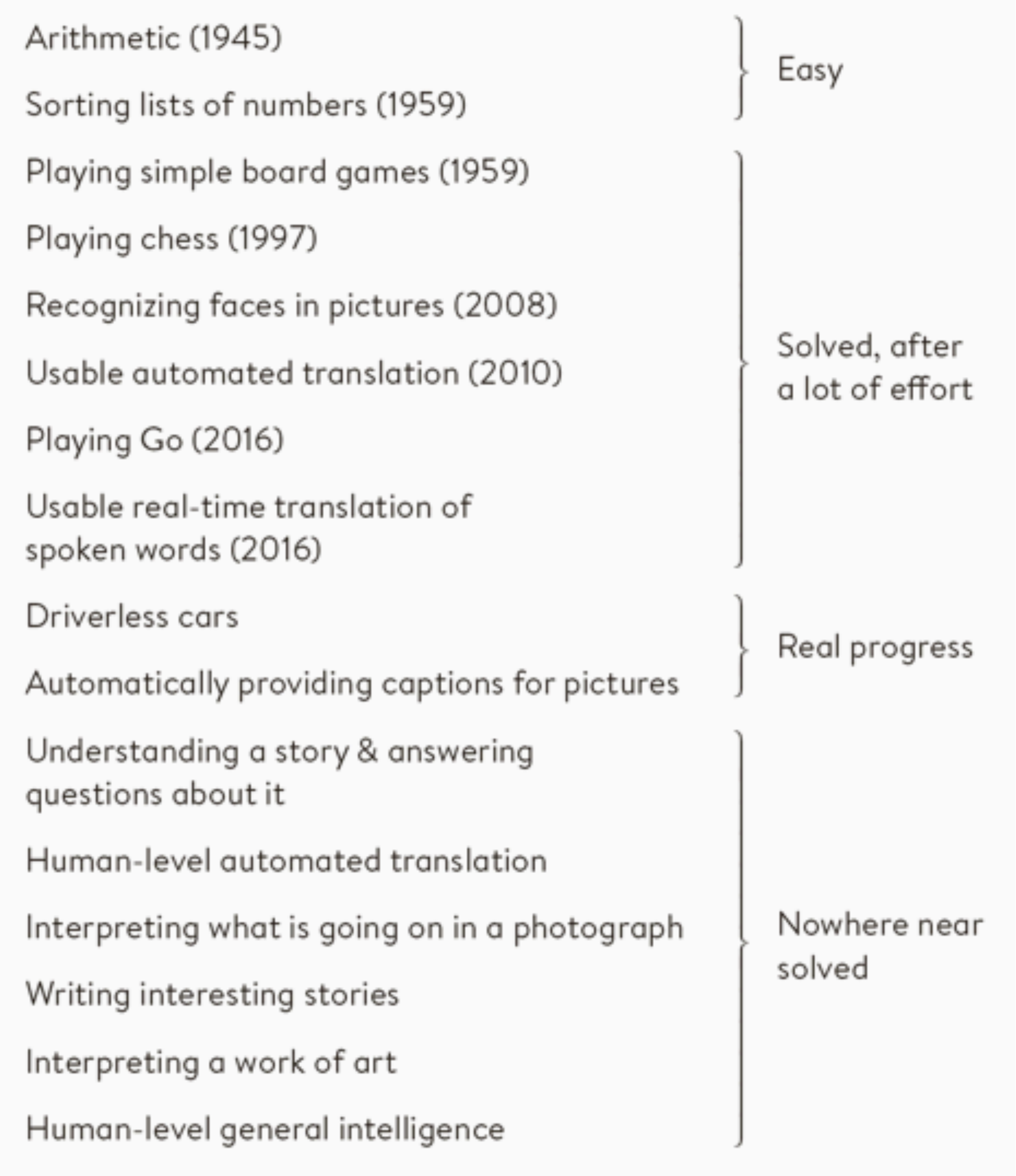
"AI is the study of how to make computers perform things that, at the moment, people do better." Elaine Rich, Artificial intelligence, McGraw-Hill 1983

G. Marcus, E. Davis, Rebooting AI



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

# Some successes and unsolved problems in AI



| | |
|---|---|
| Arithmetic (1945) | Easy |
| Sorting lists of numbers (1959) | |
| Playing simple board games (1959) | Solved, after a lot of effort |
| Playing chess (1997) | |
| Recognizing faces in pictures (2008) | |
| Usable automated translation (2010) | |
| Playing Go (2016) | |
| Usable real-time translation of spoken words (2016) | |
| Driverless cars | Real progress |
| Automatically providing captions for pictures | |
| Understanding a story & answering questions about it | Nowhere near solved |
| Human-level automated translation | |
| Interpreting what is going on in a photograph | |
| Writing interesting stories | |
| Interpreting a work of art | |
| Human-level general intelligence | |

M. Woolridge,
The Road to Conscious Machines

Impressive progress in certain fields:

- Image recognition
- Speech recognition
- Recommendation systems
- Automated translation
- Analysis of medical data

How can we profit from these developments in physics?

# Different modeling approaches

- Simple mathematical representation like linear regression. Favored by statisticians.
- Complex deterministic models based on scientific understanding of the physical process. Favored by physicists.
- Complex algorithms to make predictions that are derived from a huge number of past examples (“machine learning” as developed in the field of computer science). These are often black boxes.
- Regression models that claim to reach causal conclusions. Used by economists.

D. Spiegelhalter, The Art of Statistics – Learning from data

# Application of machine learning in experimental particle physics

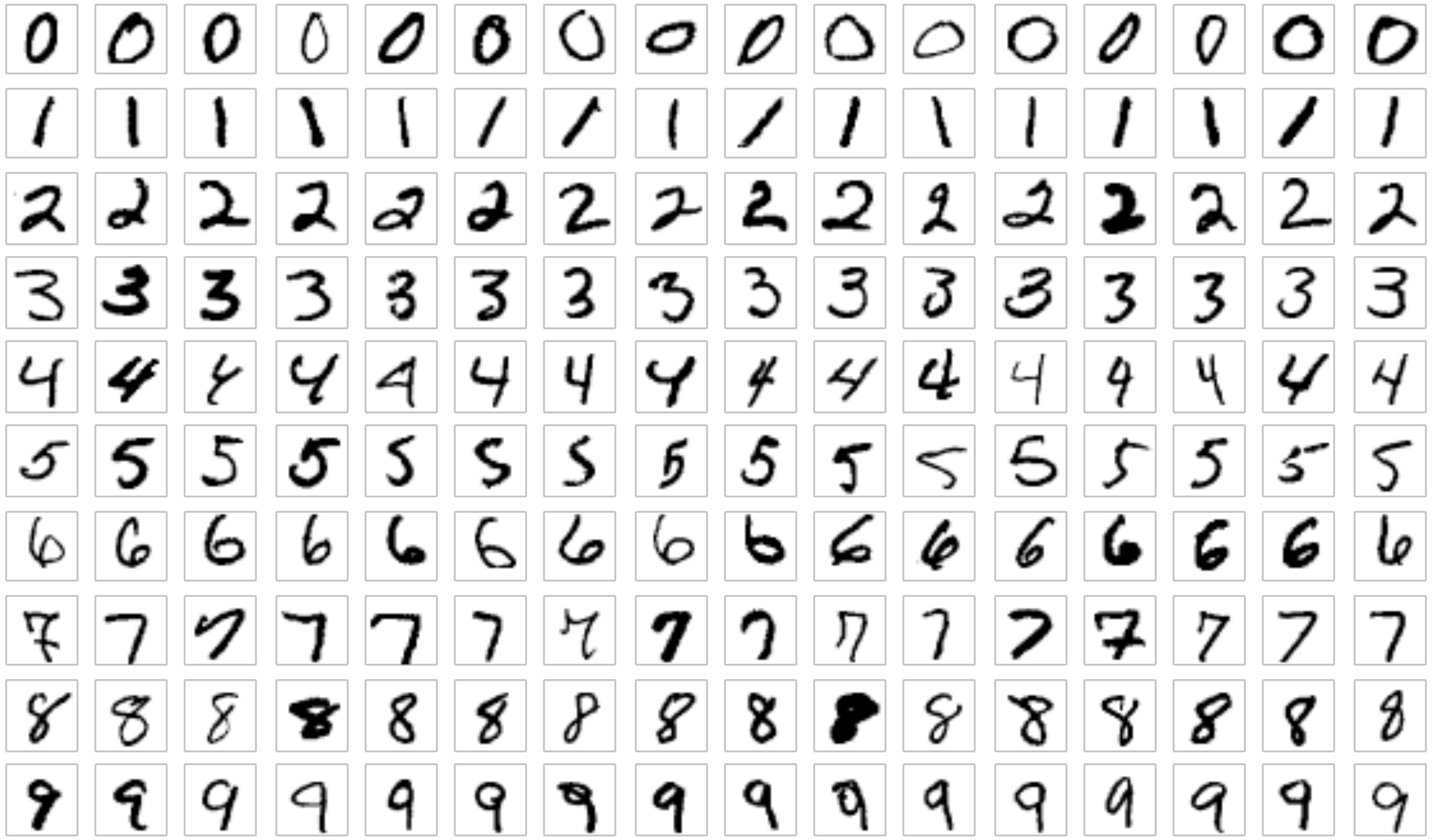
- Monte Carlo simulation
  - use generative models for faster MC event generation
- Event reconstruction and particle identification
- Data acquisition / trigger
  - faster algorithms
- Offline data analysis
  - better algorithms
- Detector monitoring
  - anomaly detection

“Machine Learning in High Energy Physics Community White Paper”, arXiv:1807.02876

# 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

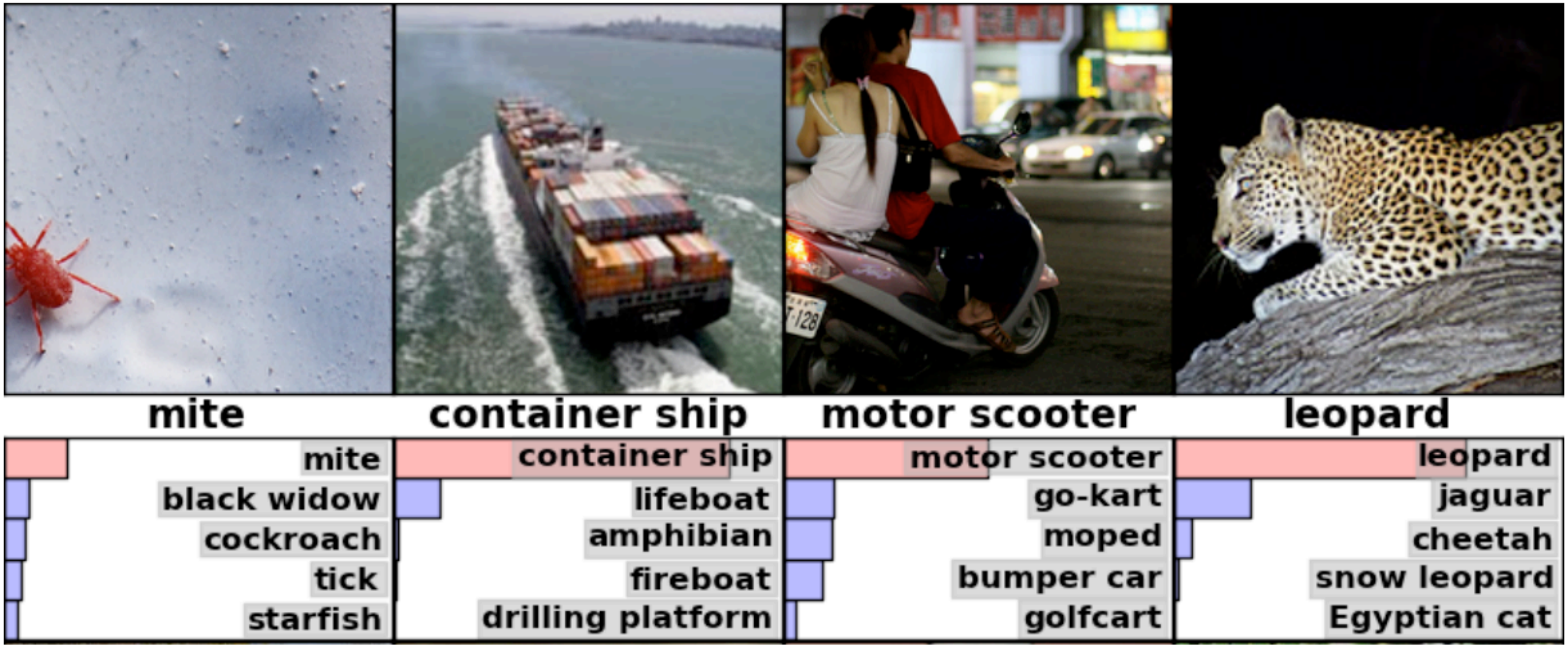
Play with MNIST data set and Keras (Stefan Wunsch, CERN IML Workshop):
https://github.com/stwunsch/iml_tensorflow_keras_workshop

# 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

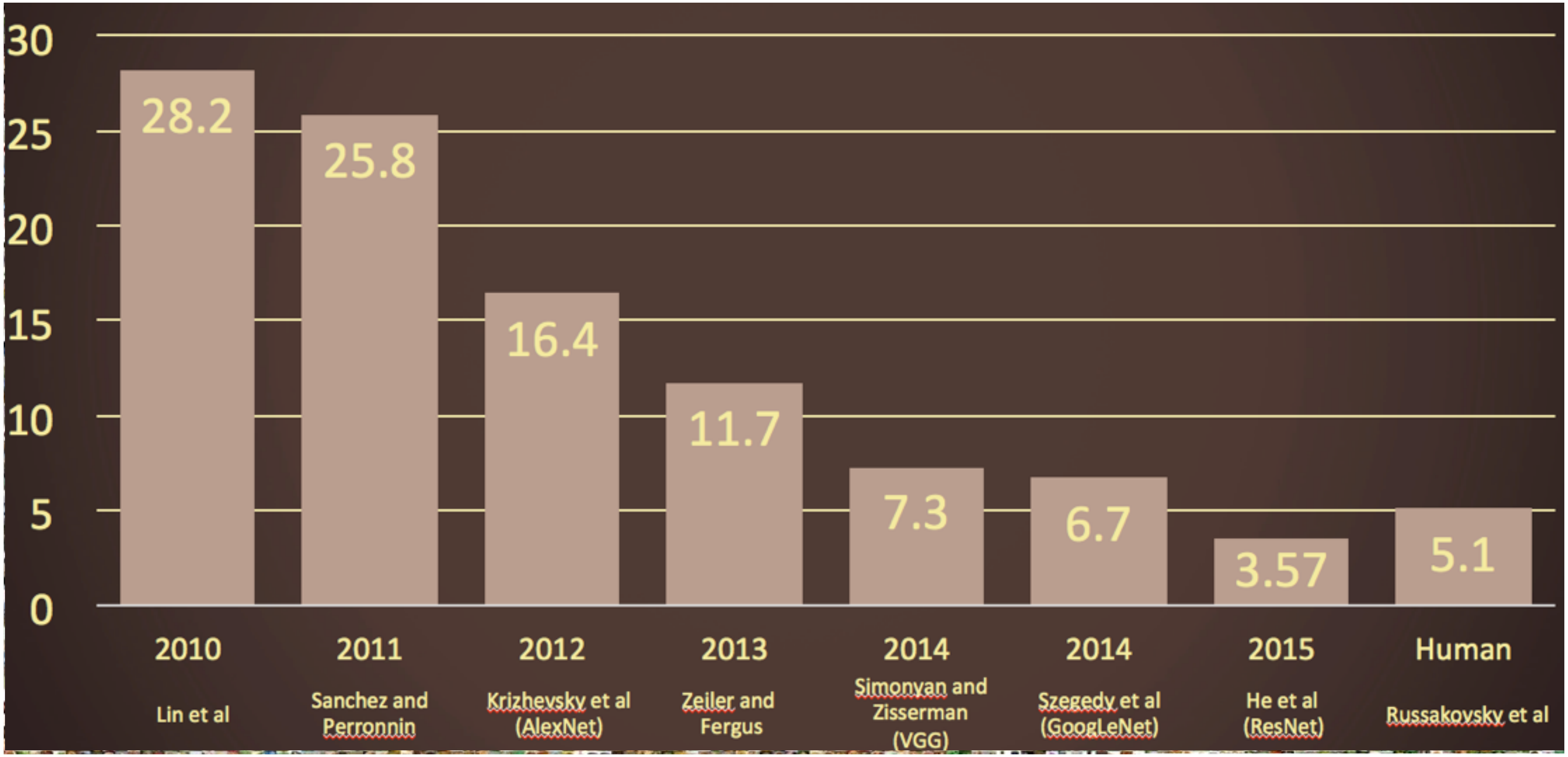
https://en.wikipedia.org/wiki/ImageNet



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

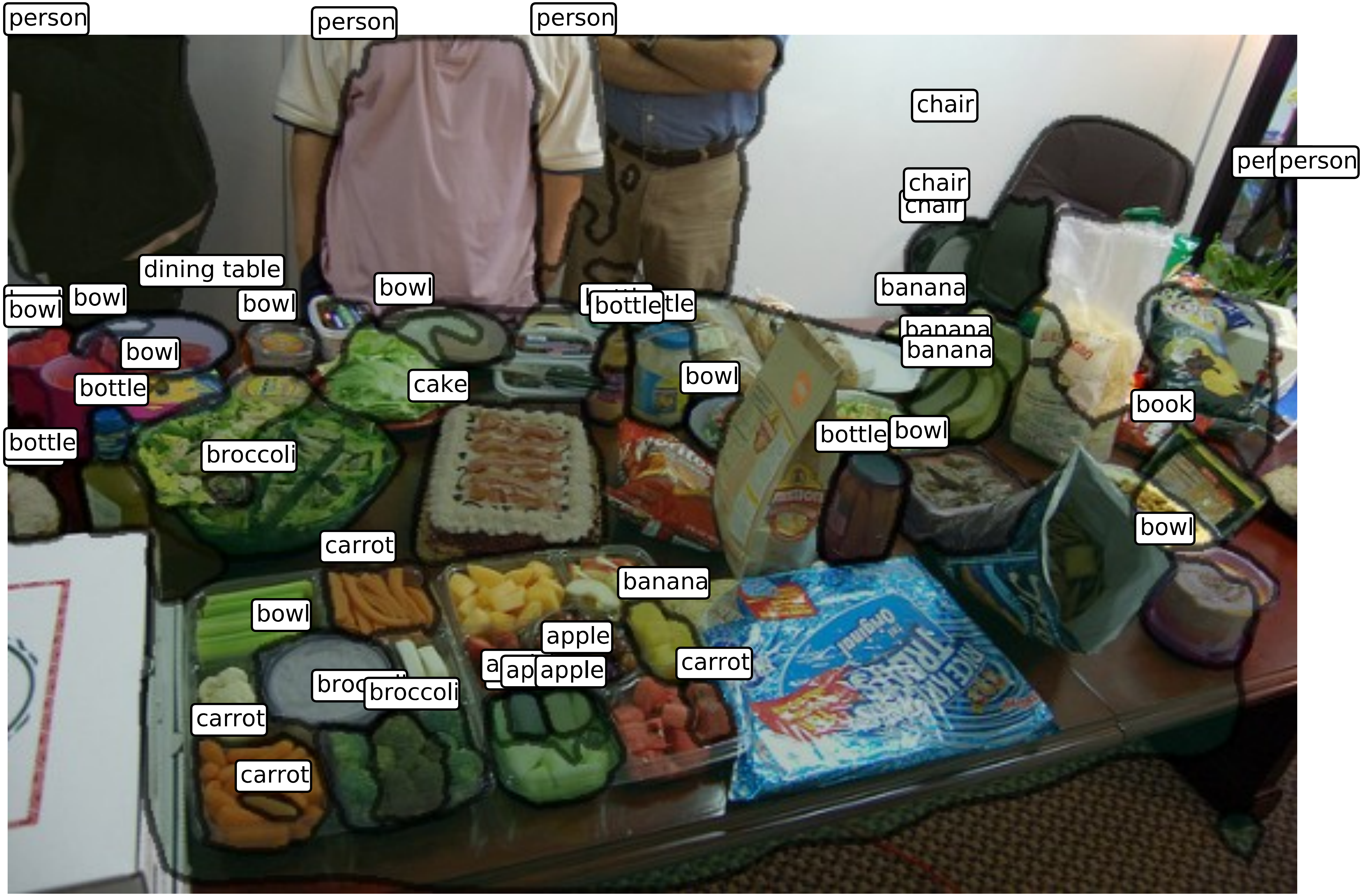
# ImageNet: Large Scale Visual Recognition Challenge

Error rate in percent:



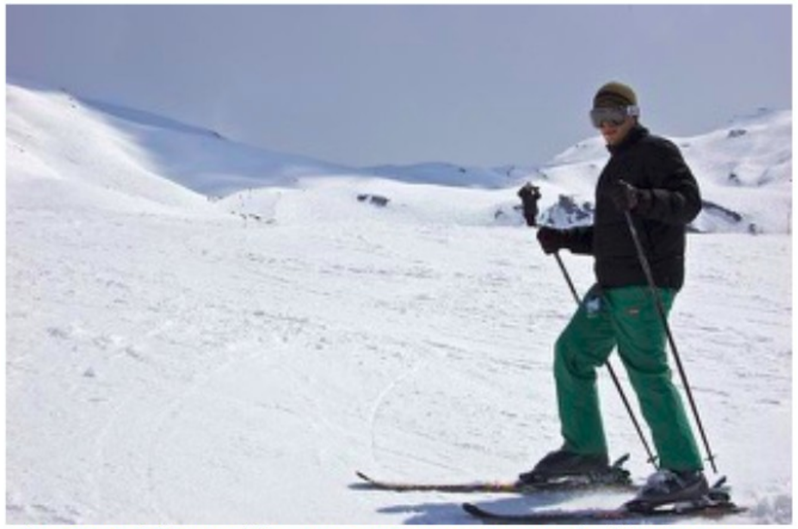
O. Russakovsky et al, arXiv:1409.0575

# Further examples (1): Segmenting and localizing objects



# Further examples (2): Image captioning

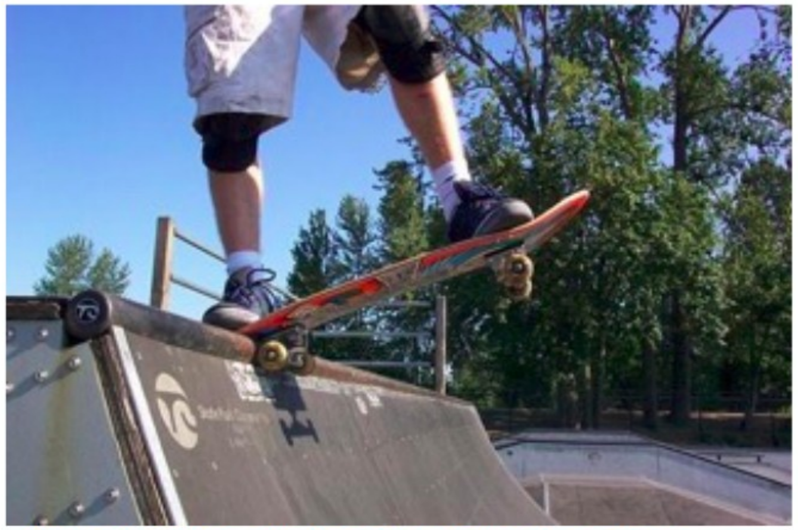
- [Lebret, Pinheiro, Collobert 2015] [Kulkarni 11] [Mitchell 12] [Vinyals 14] [Mao 14]



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.

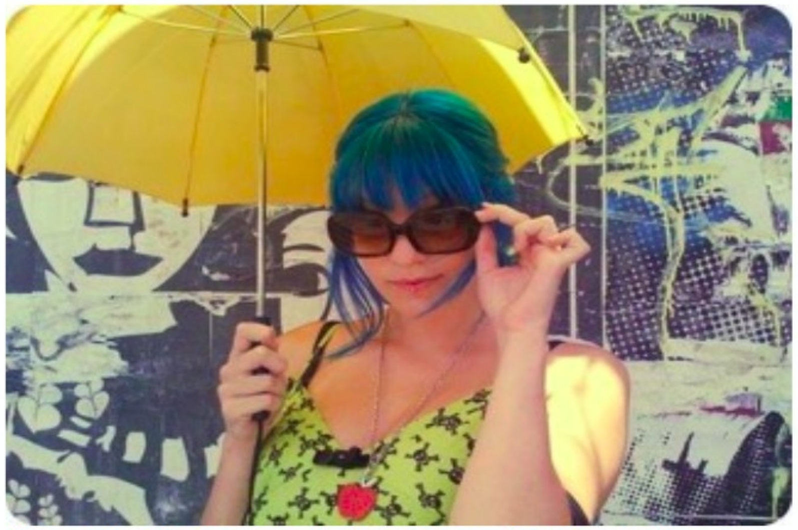
A man wearing skis on the snow.



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.

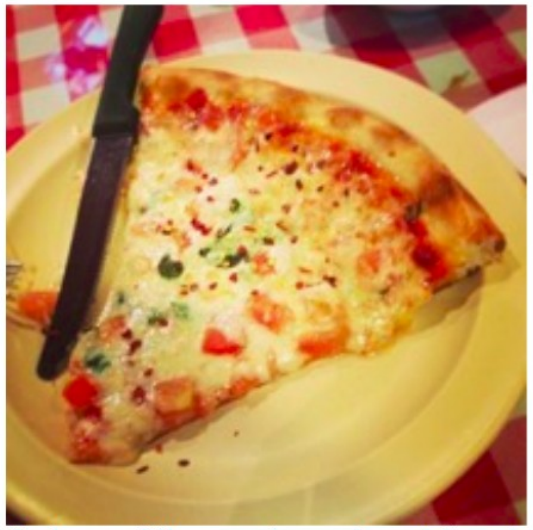
A man riding a skateboard on a ramp.



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.

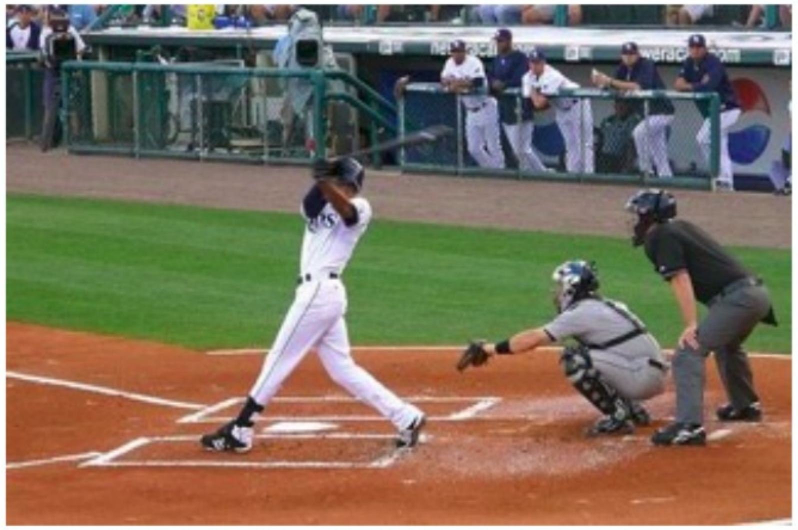
A woman is holding an umbrella.



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.

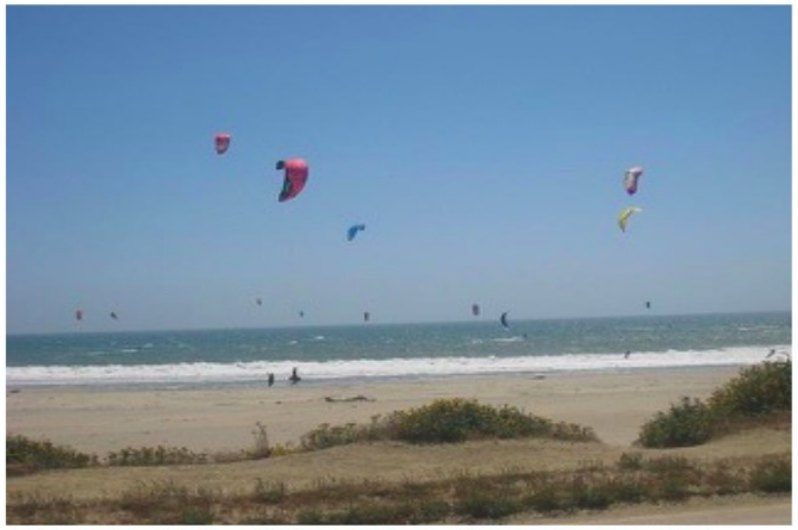
A table with a plate of pizza on a white plate.



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.



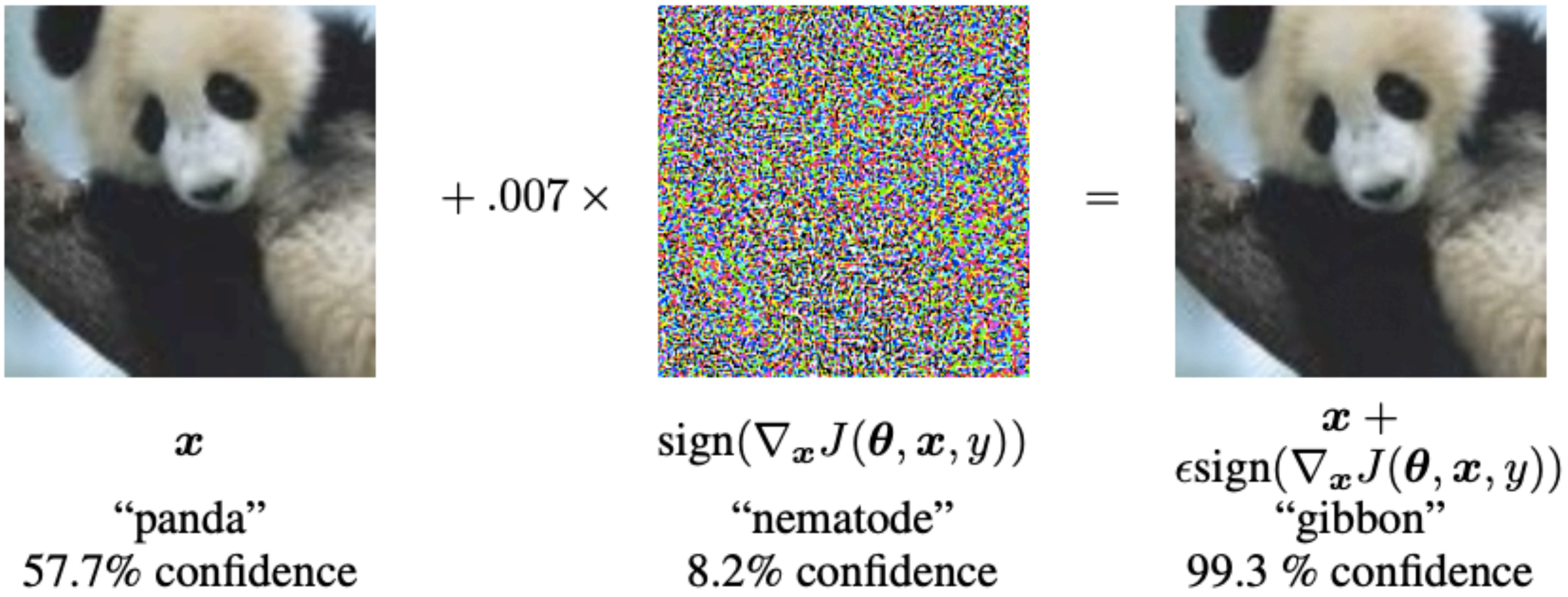
A bunch of kites flying in the sky on the beach.

**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.

# Adversarial examples

Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, arXiv:1412.6572v1

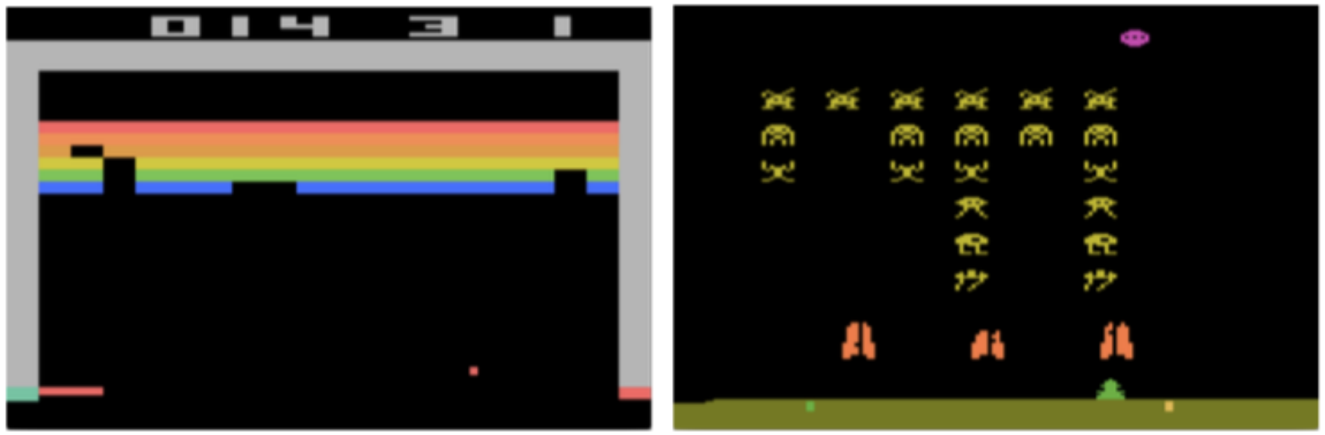

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

# 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



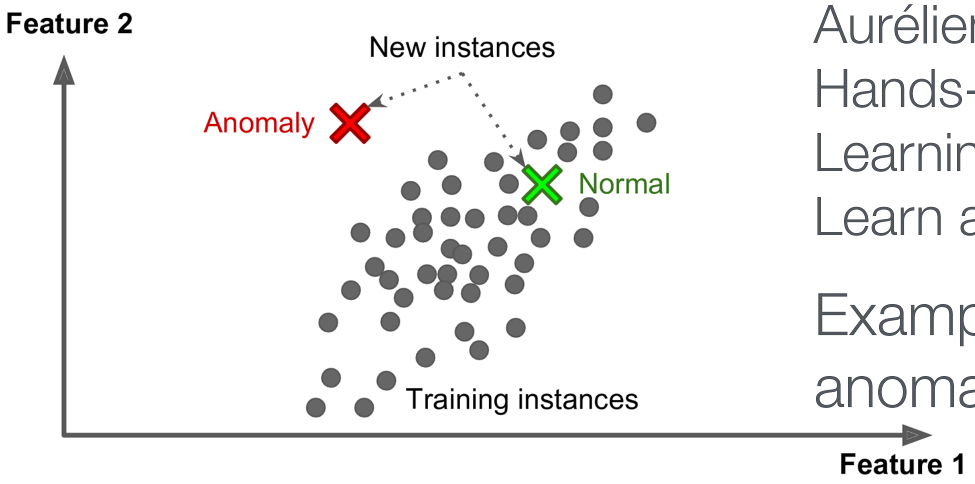
arXiv:1312.5602

## 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

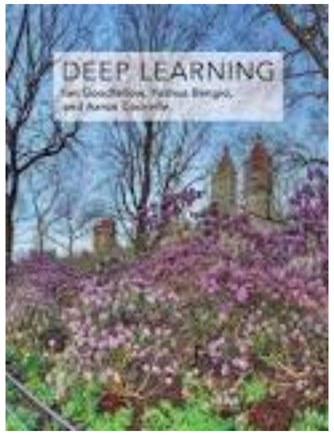


Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

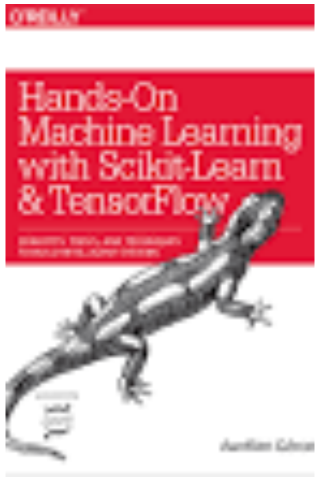
Example: anomaly detection

# 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*



# Multivariate classification

Consider events which can be either signal or background events.

Each event is characterized by *n* observables:

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

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

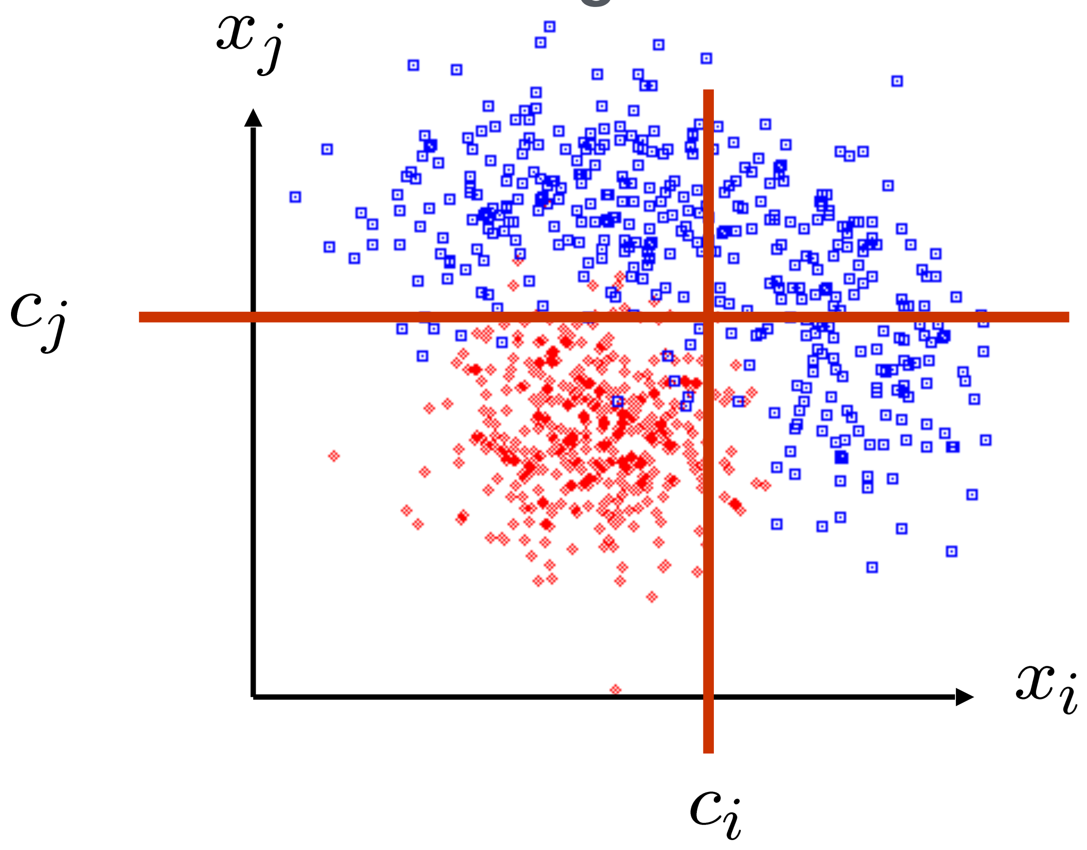
This is usually done by mapping the feature vector to a single variable, i.e., to scalar "test statistic":

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

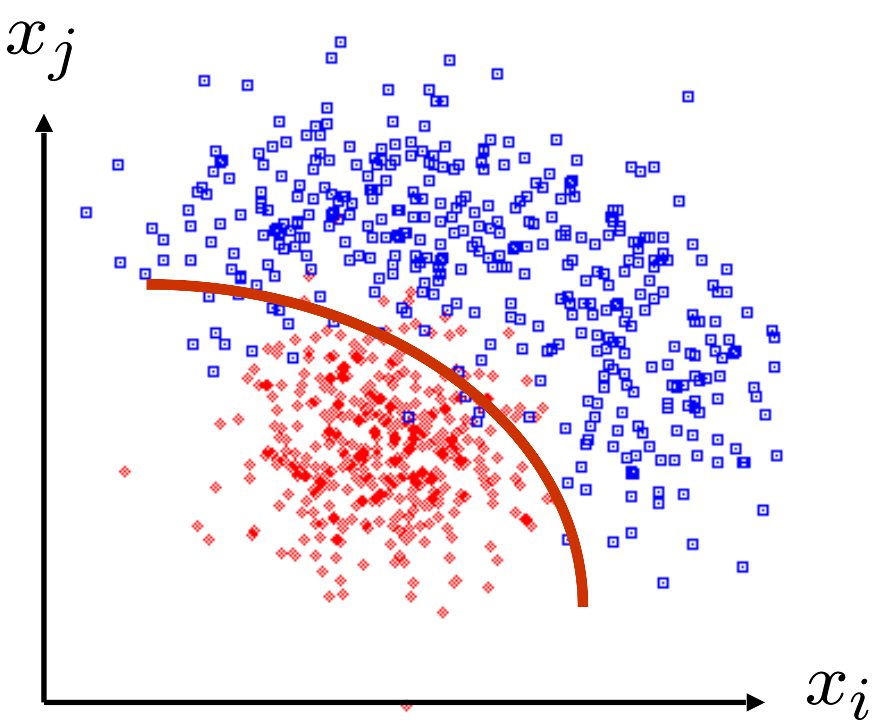
A cut $y > c$ to classify events as signal corresponds to selecting a potentially complicated hyper-surface in feature space. In general superior to classical "rectangular" cuts on the $x_i$.

# Classification: Learning decision boundaries

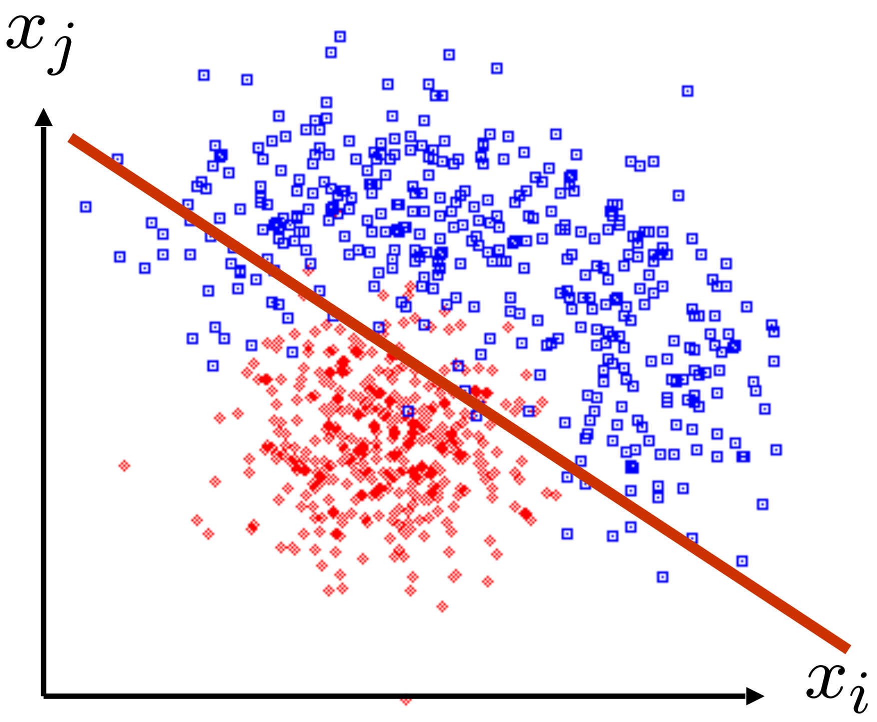
**rectangular cuts**



**non linear**



**linear**



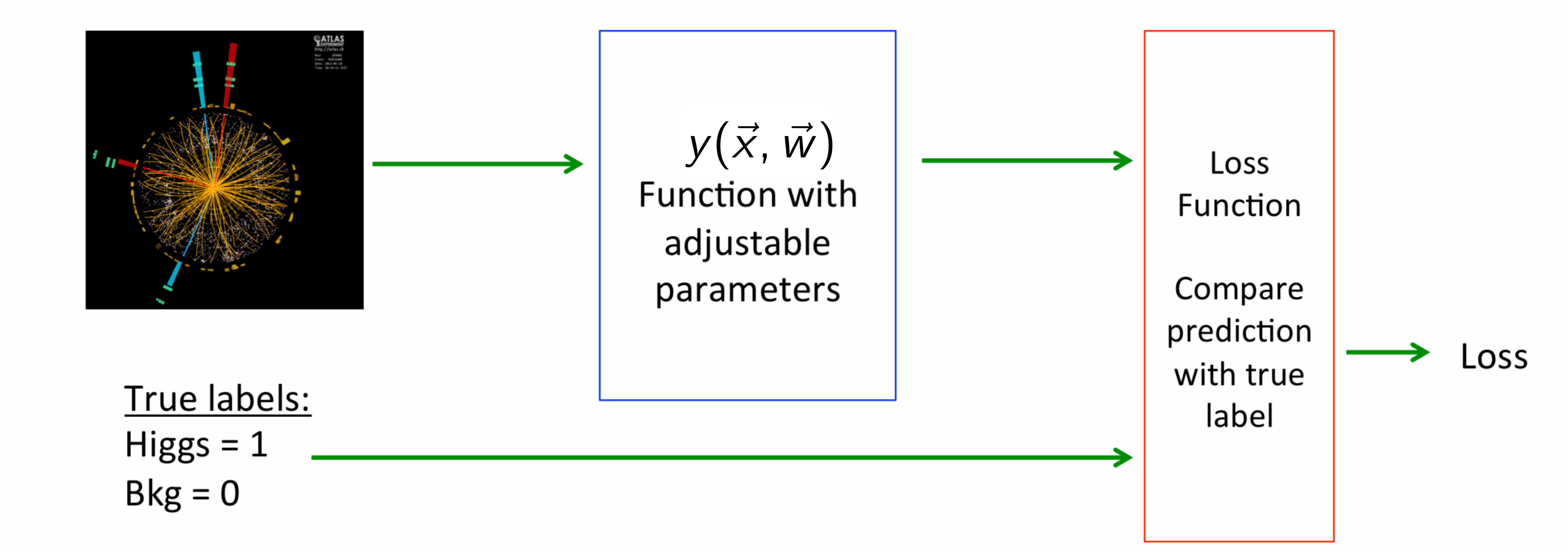
*k*-Nearest-Neighbor,
Boosted Decision Trees,
Multi-Layer Perceptrons,
Support Vector Machines
…

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

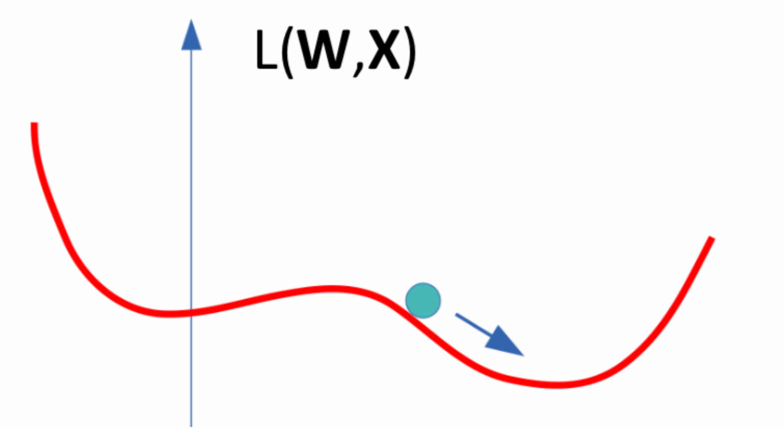
Loss

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* → *Y* can be a set of labels or classes or a continuous domain, e.g., 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

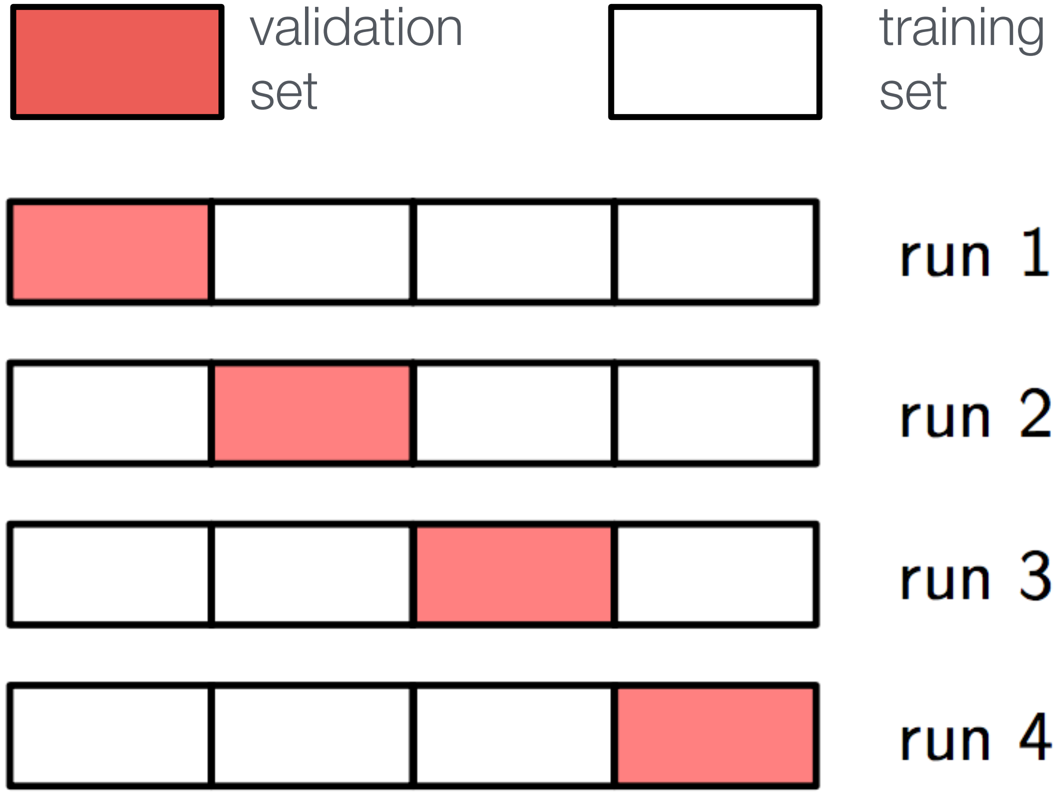
## Rule of thumb if training data not expensive

- 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

**predicted "probability" for outcome t = 1**

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

# 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 (i.e, a measure of how well the model Q describes the true distribution P)

# Cross-entropy error function for logistic regression

Let $Y \in \{0,1\}$ be a random variable; outcome of experiment *i*: $y_i$

Consider one event with feature vector $\vec{x}$ and label $y \in \{0,1\}$

Predicted probability $q_1$ for outcome $Y = 1$:

By construction the right property for predicting a probability

$$q_1 \equiv q(Y=1) = \sigma(\vec{x}; \vec{w}) \equiv \sigma(w_0 + \sum_{i=1}^{n} w_i x_i), \qquad \sigma: \mathbb{R} \mapsto [0,1], \;\; \sigma(z) = \frac{1}{1+e^{-z}}$$

$$q_0 \equiv q(Y=0) = 1 - q(Y=1)$$

logistic function

The true probabilities $p_i$ are either 0 or 1, so we can write $p_1 \equiv p(Y=1) = y, \;\; p_0 = 1 - p_1 \equiv 1 - y$. With this the cross entropy is:

$$H(p,q) \;=\; -\sum_{k=0}^{1} p_k \log q_k \;=\; -\,y\,\log\sigma(\vec{x},\vec{w}) - (1-y)\,\log(1-\sigma(\vec{x},\vec{w}))$$

Loss function from sum over entire data set:

$$E(\vec{w}) = -\sum_{i=1}^{n_\text{samples}} y_i \log\sigma(\vec{x}_i, \vec{w}) + (1-y_i)\,\log(1-\sigma(\vec{x}_i,\vec{w}))$$

# Logistic regression:
# loss function from maximum likelihood

We can write the two predicted probabilities $q_0$ and $q_1$ in the following way:

$$q(Y = y) = \sigma(\vec{x}; \vec{w})^y \cdot (1 - \sigma(\vec{x}; \vec{w}))^{1-y}$$

With this the likelihood can be written as

$$\begin{aligned} L(\vec{w}) &= \prod_{i=1}^{n_\text{samples}} q(Y = y_i) \\ &= \prod_{i=1}^{n_\text{samples}} \sigma(\vec{x}; \vec{w})^{y_i} \cdot (1 - \sigma(\vec{x}; \vec{w}))^{1-y_i} \end{aligned}$$

The corresponding log-likelihood function is

$$\log L(\vec{w}) = \sum_{i=1}^{n_\text{samples}} y_i \, \log \sigma(\vec{x}_i; \vec{w}) + (1 - y_i) \, \log(1 - \sigma(\vec{x}_i; \vec{w}))$$

Thus, minimizing the cross entropy loss function corresponds to finding the maximum likelihood estimate.

# Multinomial logistic regression: Softmax function

In the previous example we considered two classes (0, 1). For multi-class classification, the logistic function can generalized to the softmax function.

Consider $K$ classes and let $z_i$ be the score for class *i*, $\vec{z} = (z_1, \ldots, z_K)$

A probability for class *i* can be predicted with the softmax function:

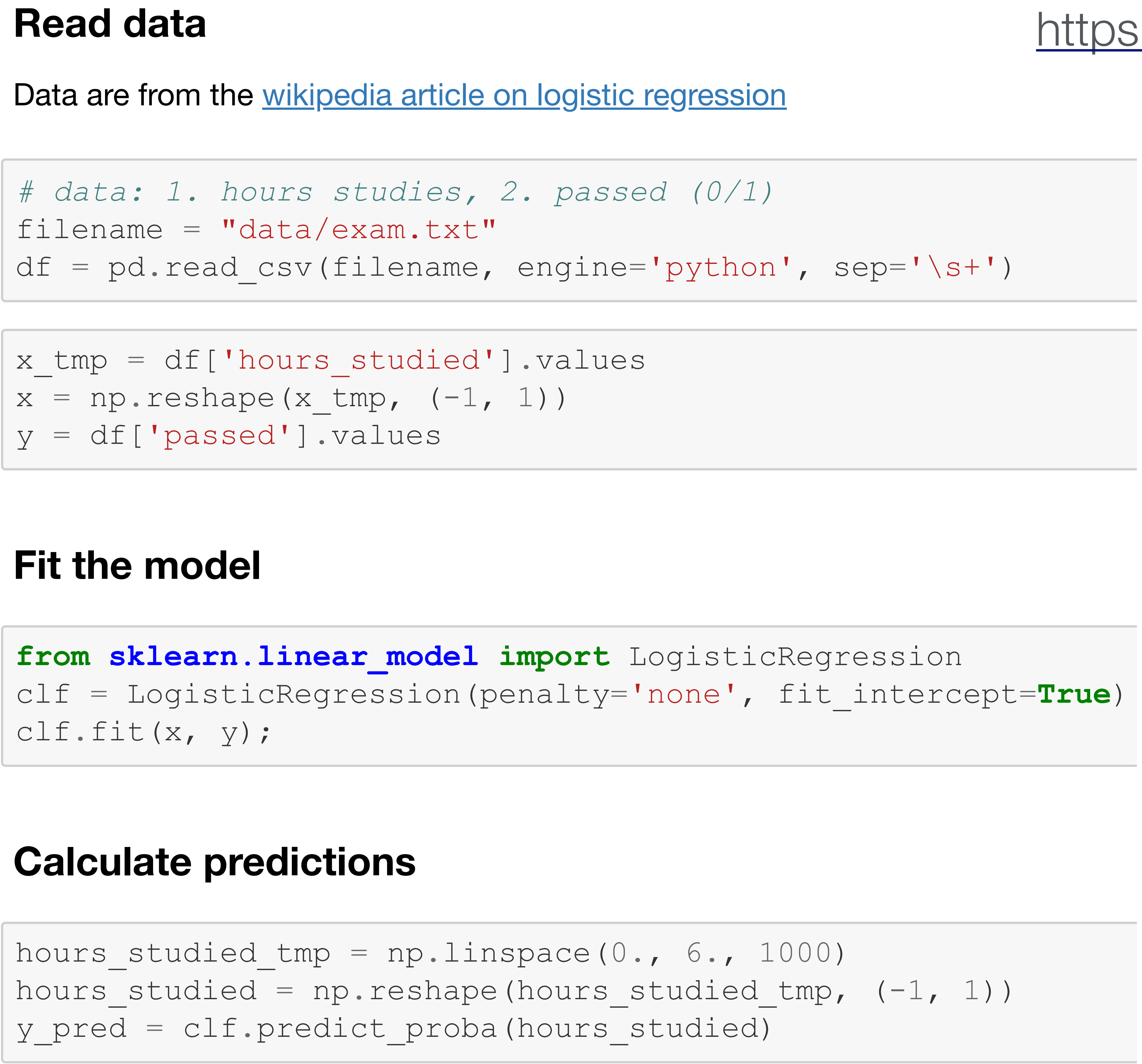
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for} \quad i = 1,\ldots,K$$

The softmax functions is often used as the last activation function of a neural network in order to predict probabilities in a classification task.

Multinomial logistic regression is also known as softmax regression.

# Simple example of logistic regression with scikit-learn (1)

https://scikit-learn.org



## Read data

Data are from the wikipedia article on logistic regression

```
# data: 1. hours studies, 2. passed (0/1)
filename = "data/exam.txt"
df = pd.read_csv(filename, engine='python', sep='\s+')
```

```
x_tmp = df['hours_studied'].values
x = np.reshape(x_tmp, (-1, 1))
y = df['passed'].values
```

## Fit the model

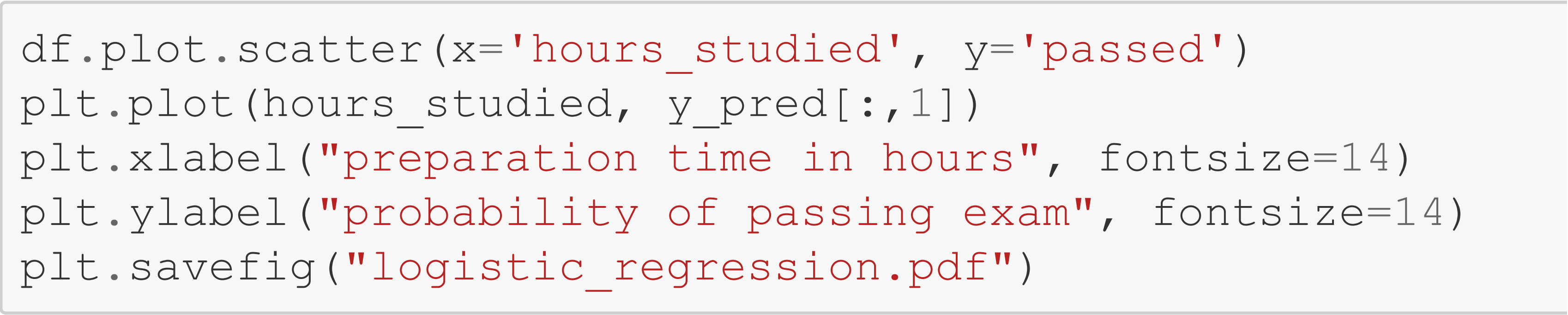
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(penalty='none', fit_intercept=True)
clf.fit(x, y);
```

## Calculate predictions

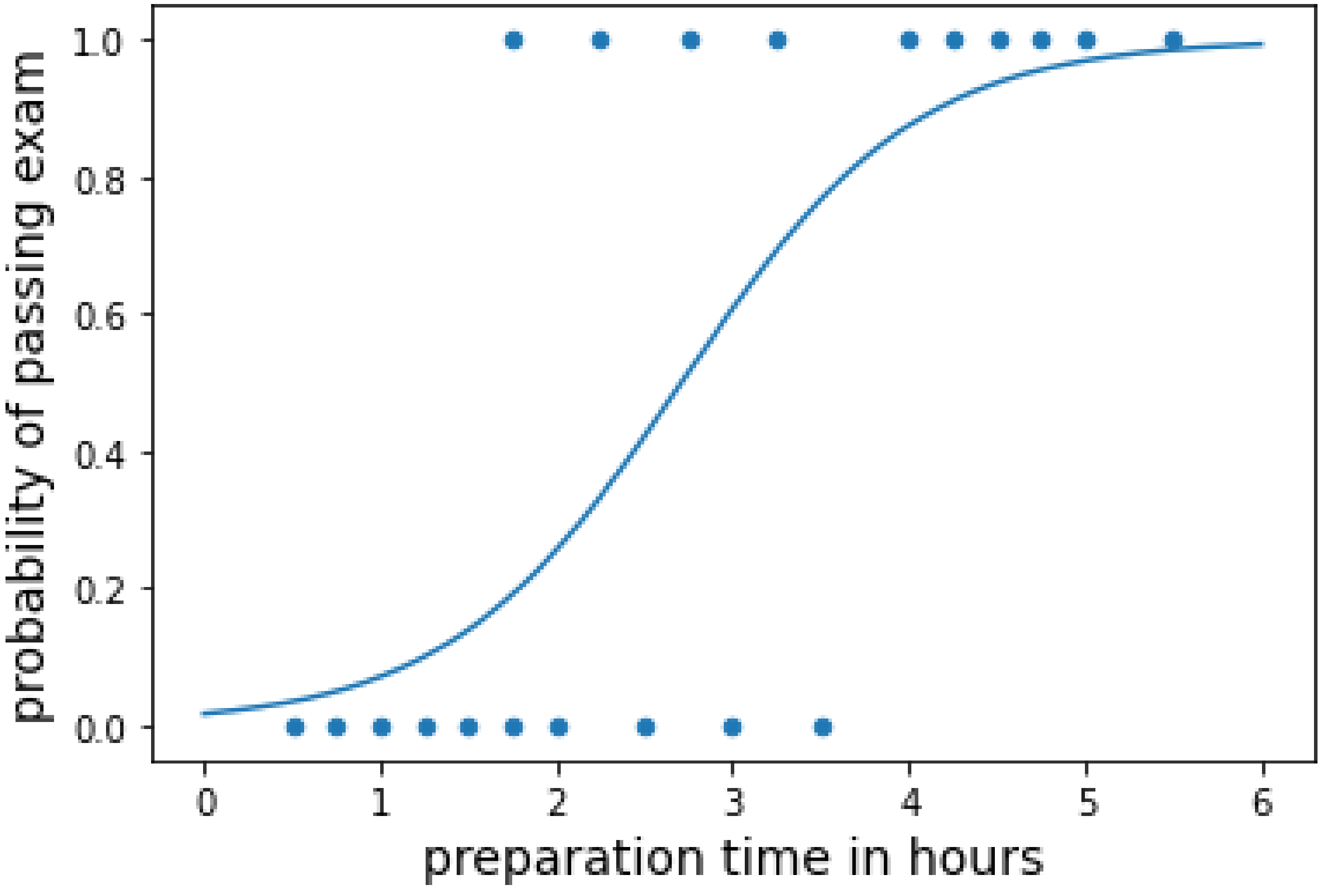
```
hours_studied_tmp = np.linspace(0., 6., 1000)
hours_studied = np.reshape(hours_studied_tmp, (-1, 1))
y_pred = clf.predict_proba(hours_studied)
```

# Simple example of logistic regression with scikit-learn (2)

### Plot result



```
df.plot.scatter(x='hours_studied', y='passed')
plt.plot(hours_studied, y_pred[:,1])
plt.xlabel("preparation time in hours", fontsize=14)
plt.ylabel("probability of passing exam", fontsize=14)
plt.savefig("logistic_regression.pdf")
```



# Reminder: 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:

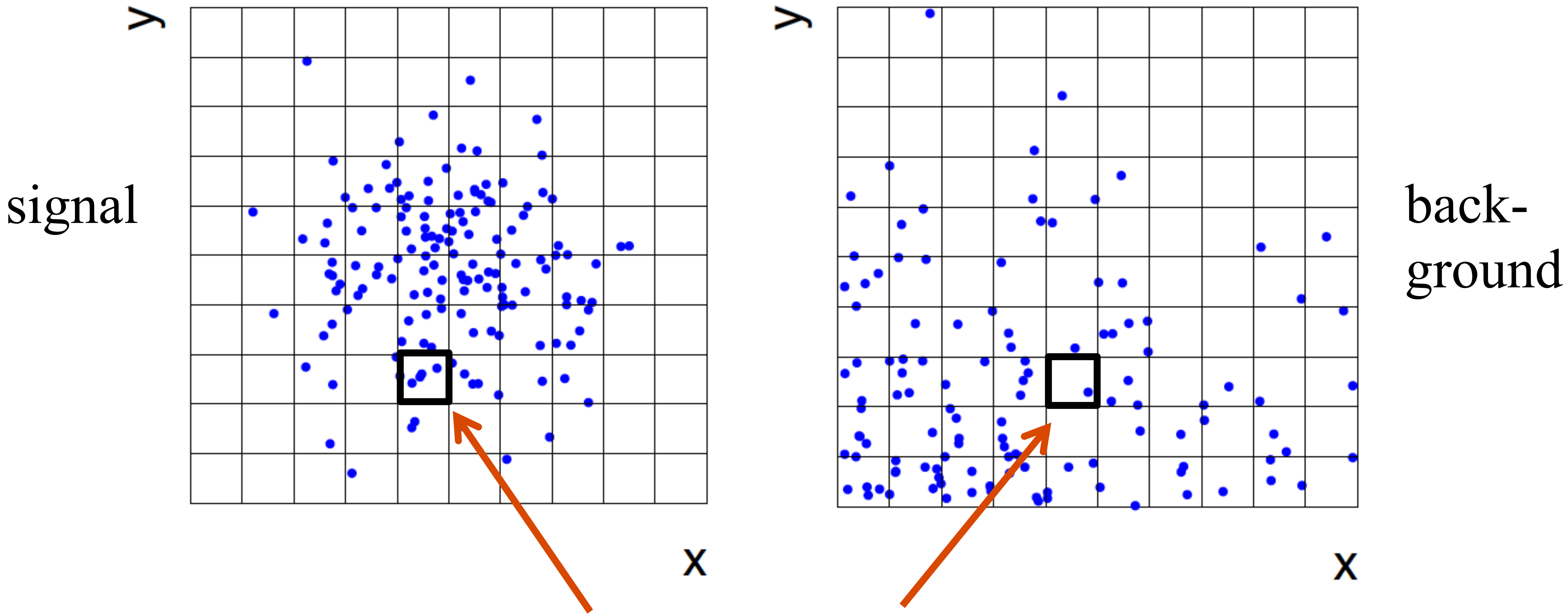
1. Estimate signal and background pdf's and construct test statistic based on Neyman-Pearson lemma

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



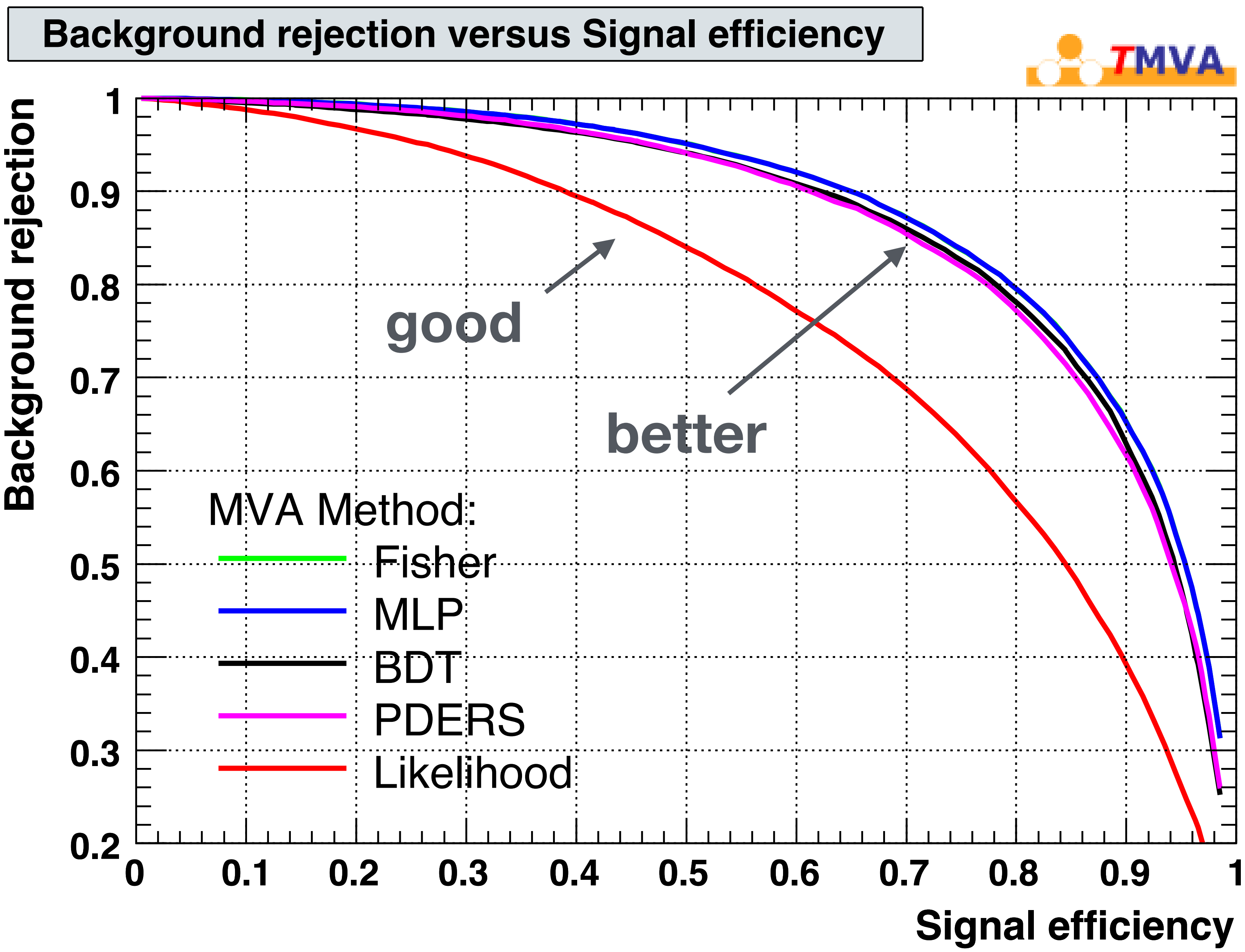
approximate PDF by $N(x, y|S)$ and $N(x, y|B)$

*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)



$1 - \varepsilon_B$

$\varepsilon_B$: background efficiency

# Naïve Bayesian classifier (also called "projected likelihood classification")

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

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

$$\begin{aligned} \text{where} \quad f_1(x_1) &= \int \mathrm{d}x_2 \mathrm{d}x_3 \ldots \mathrm{d}x_n \; f(x_1, x_2, \ldots, x_n) \\ f_2(x_2) &= \int \mathrm{d}x_1 \mathrm{d}x_3 \ldots \mathrm{d}x_n \; f(x_1, x_2, \ldots, x_n) \\ &\vdots \end{aligned}$$

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

# *k*-nearest neighbor method (1)

## *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 (2)

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

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

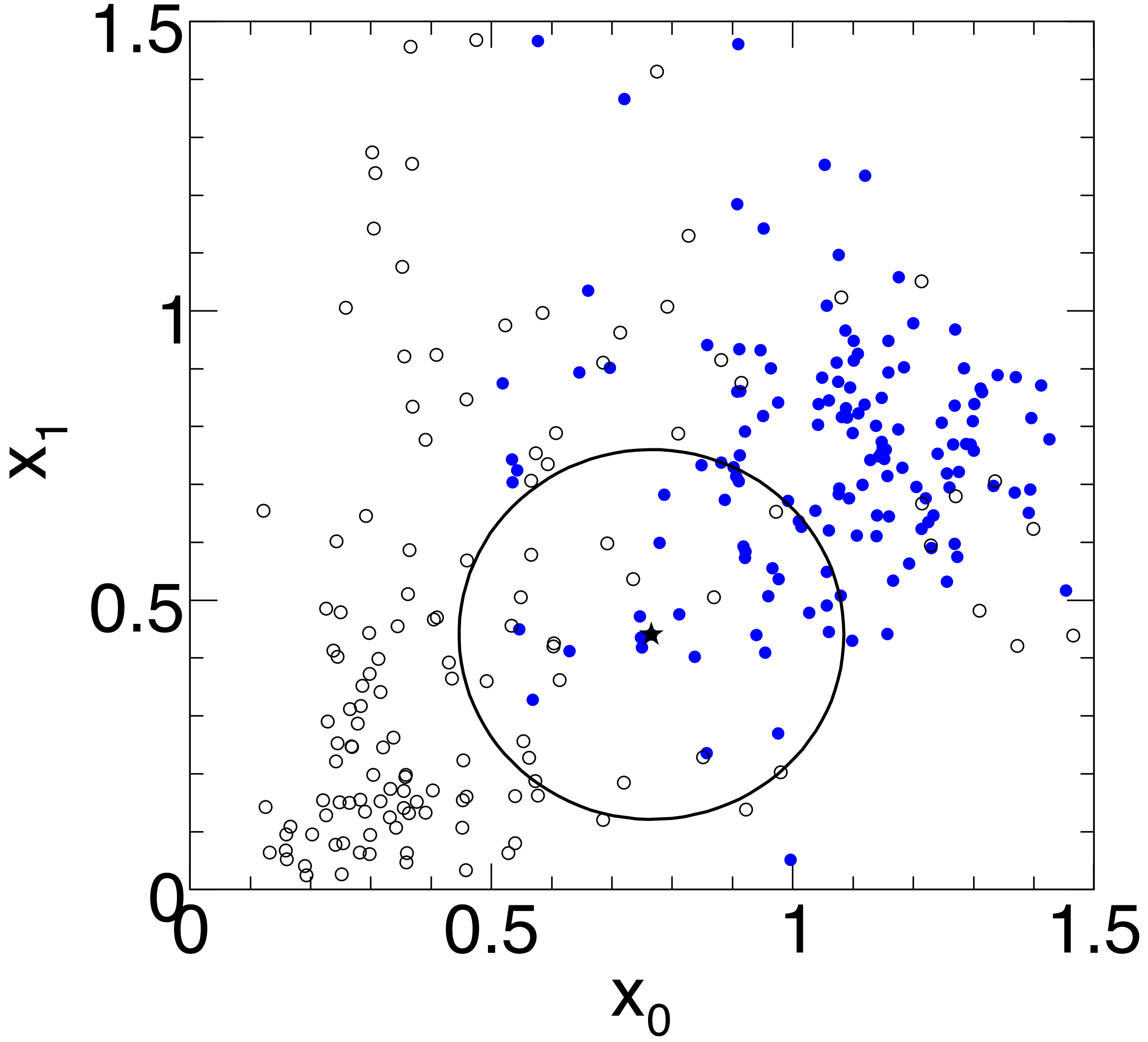
Better: take correlations between variables into account:

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

$$V = \text{covariance matrix}$$

"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

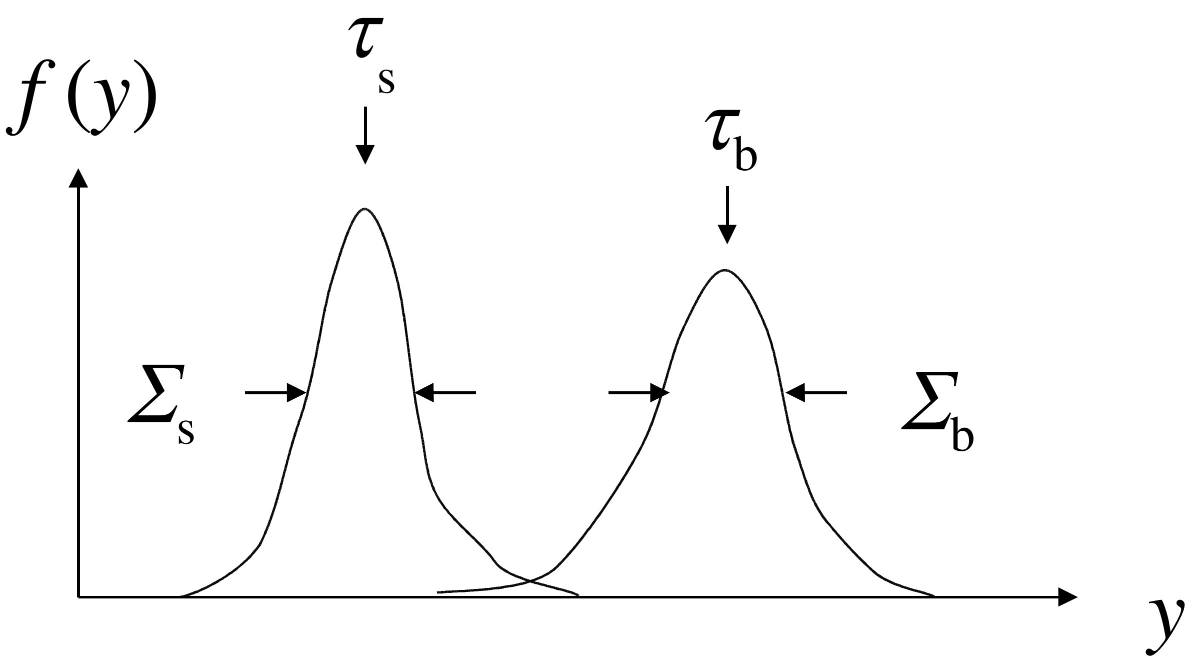
Linear discriminant is simple. Can still be optimal if amount of training data is limited.

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

Choose parameters *wᵢ* 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 variance 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 $J(\vec{w})$ :

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

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

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

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

Maximize:

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

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

# Fisher linear discriminant: Determining the coefficients $w_i$

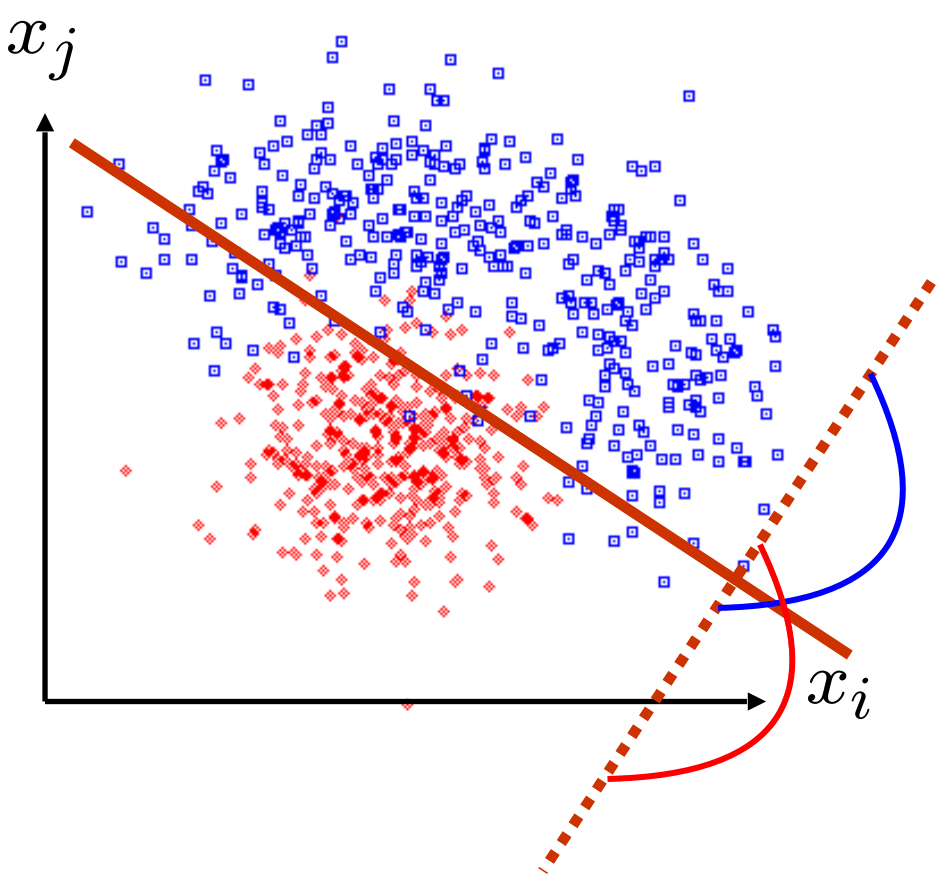
Setting $\frac{\partial J}{\partial w_i} = 0$ gives:

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

We obtain linear decision boundaries.

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

linear decision boundary



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

# Fisher linear discriminant: Remarks

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

$$y(\vec{x}) \propto \ln \frac{f(\vec{x}|H_\mathrm{s})}{f(\vec{x}|H_\mathrm{b})}$$

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).

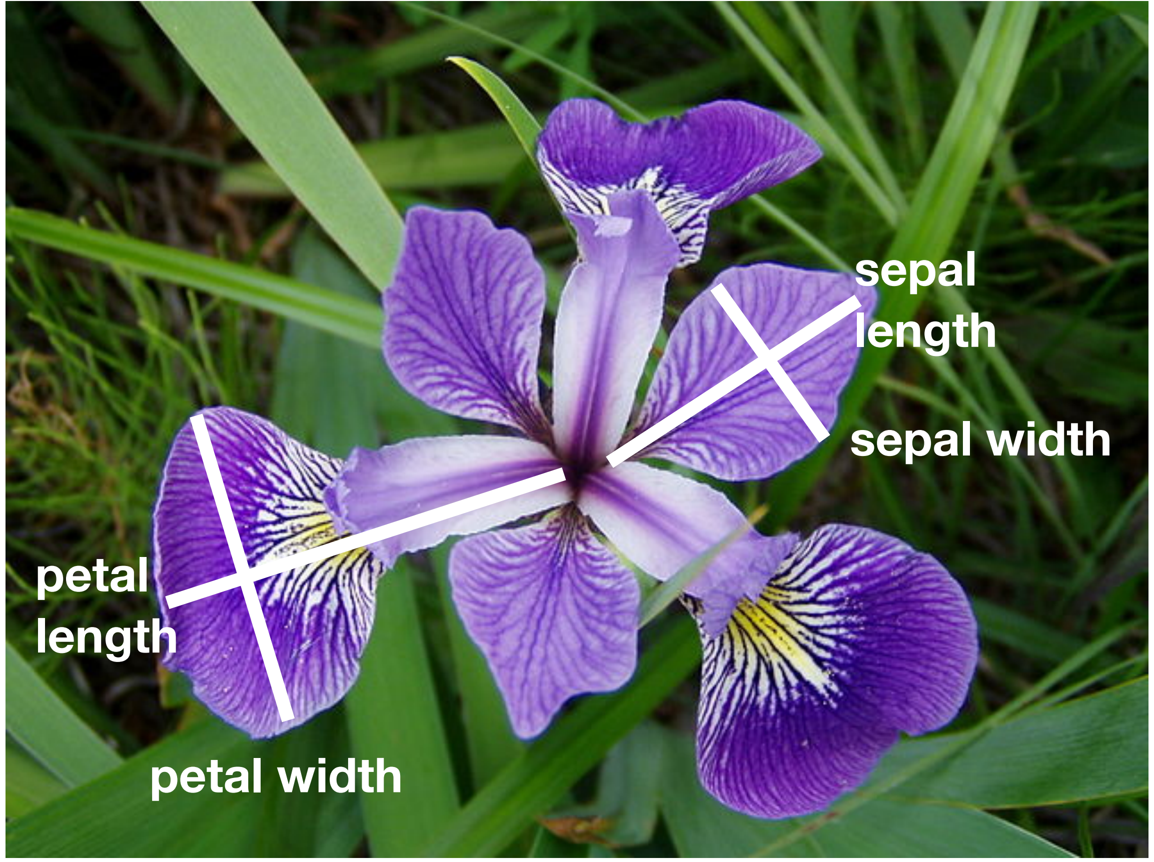
# Example: Classification with scikit-learn (1)

## Iris flower data set

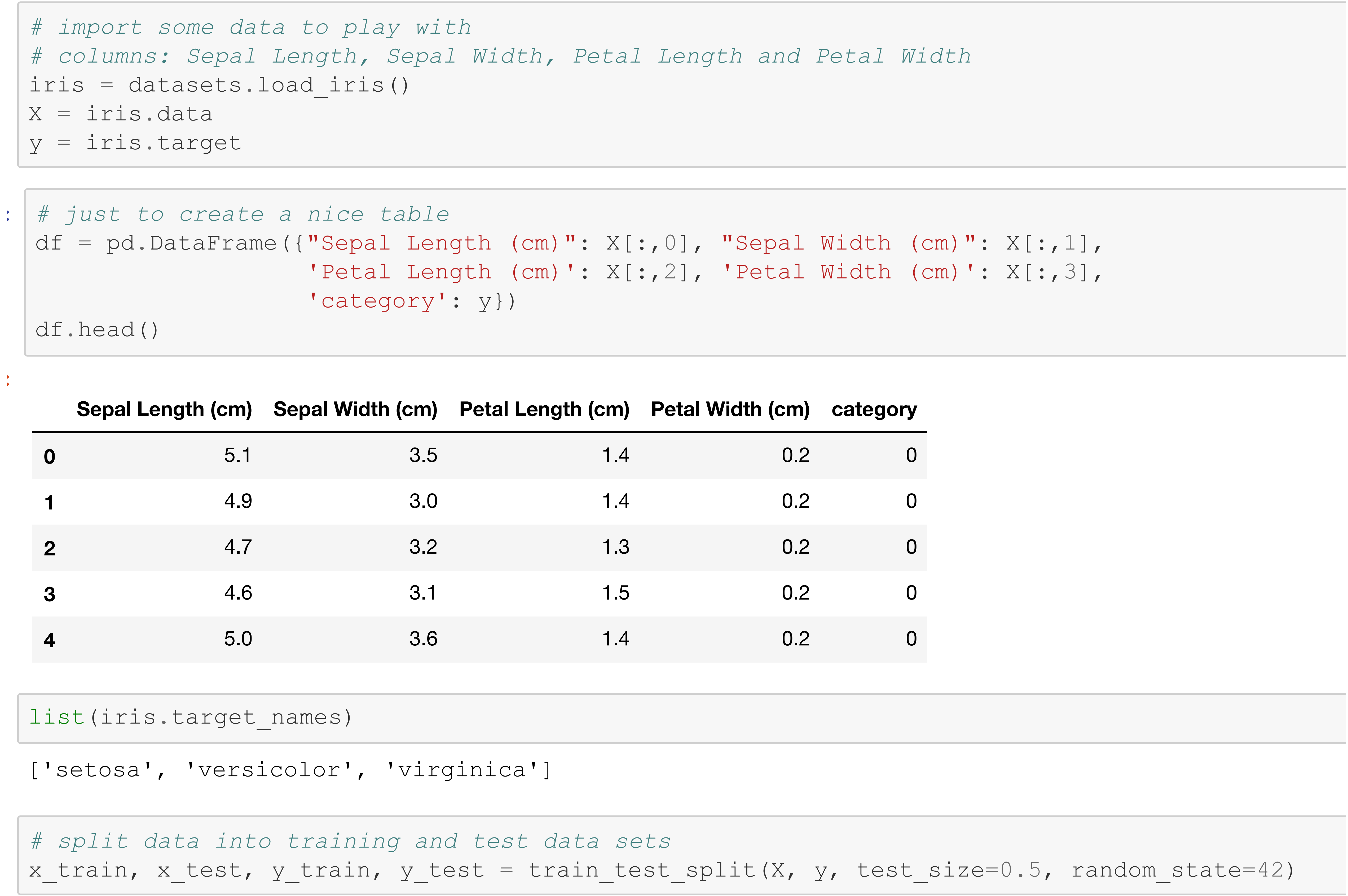
https://archive.ics.uci.edu/ml/datasets/Iris

- Introduced 1936 in a paper by Ronald Fisher
- Task: classify flowers
- Three species: iris setosa, iris virginica and iris versicolor
- Four features: petal width and length, sepal width/length, in centimeters

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



# Example: Classification with scikit-learn (2)



```
# import some data to play with
# columns: Sepal Length, Sepal Width, Petal Length and Petal Width
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
# just to create a nice table
df = pd.DataFrame({"Sepal Length (cm)": X[:,0], "Sepal Width (cm)": X[:,1],
                   'Petal Length (cm)': X[:,2], 'Petal Width (cm)': X[:,3],
                   'category': y})
df.head()
```

| | Sepal Length (cm) | Sepal Width (cm) | Petal Length (cm) | Petal Width (cm) | category |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

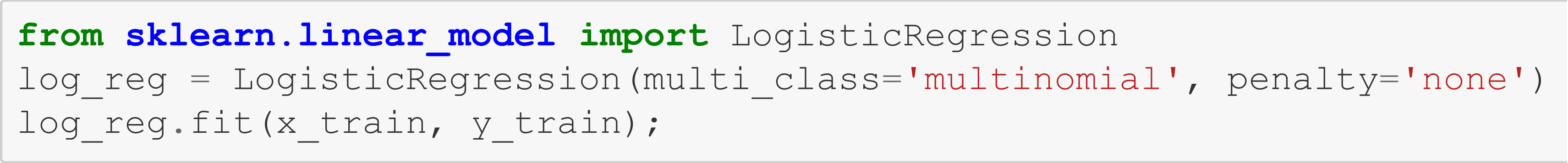
```
list(iris.target_names)
```

```
['setosa', 'versicolor', 'virginica']
```

```
# split data into training and test data sets
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

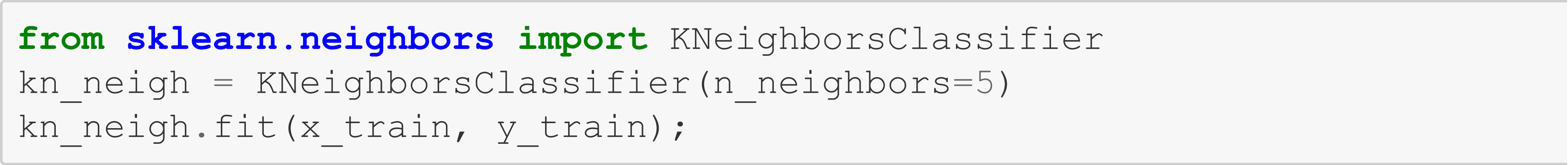
# Example: Classification with scikit-learn (3)

## Softmax regression



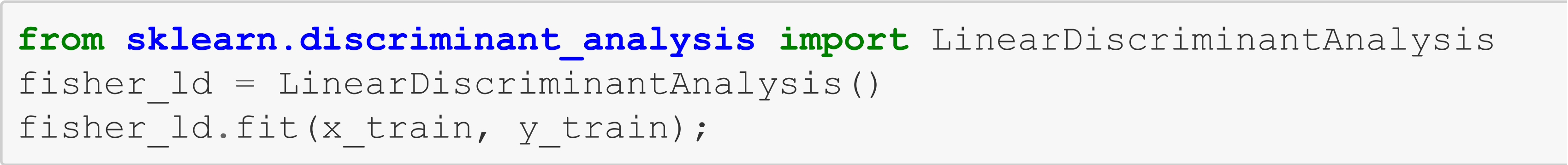
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(multi_class='multinomial', penalty='none')
log_reg.fit(x_train, y_train);
```

## k-nearest neighbor



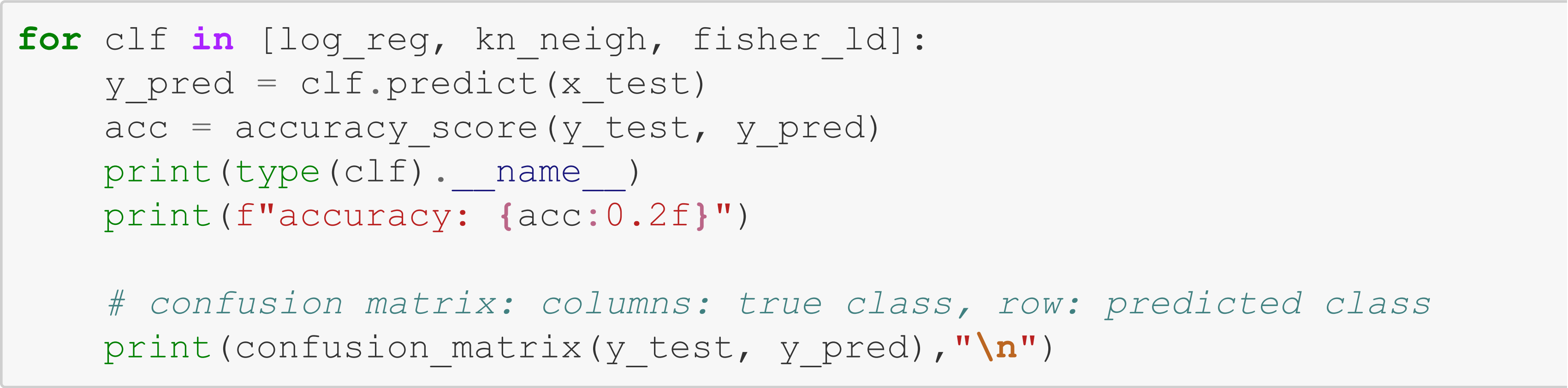
```
from sklearn.neighbors import KNeighborsClassifier
kn_neigh = KNeighborsClassifier(n_neighbors=5)
kn_neigh.fit(x_train, y_train);
```

## Fisher linear discriminant



```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
fisher_ld = LinearDiscriminantAnalysis()
fisher_ld.fit(x_train, y_train);
```

## Classification accuracy



```
for clf in [log_reg, kn_neigh, fisher_ld]:
    y_pred = clf.predict(x_test)
    acc = accuracy_score(y_test, y_pred)
    print(type(clf).__name__)
    print(f"accuracy: {acc:0.2f}")

    # confusion matrix: columns: true class, row: predicted class
    print(confusion_matrix(y_test, y_pred),"\n")
```

Output:

```
LogisticRegression
accuracy: 0.96
[[29  0  0]
 [ 0 23  0]
 [ 0  3 20]]

KNeighborsClassifier
accuracy: 0.95
[[29  0  0]
 [ 0 23  0]
 [ 0  4 19]]

LinearDiscriminantAnalysis
accuracy: 0.99
[[29  0  0]
 [ 0 23  0]
 [ 0  1 22]]
```

With scikit-learn it is extremely simple to test and apply different classification methods

# Precision and recall

TP: true positives
FP: false positives
FN: false negatives

## Precision:

Fraction of correctly classified instances among all instances that obtain a certain class label.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

“purity”

## Recall:

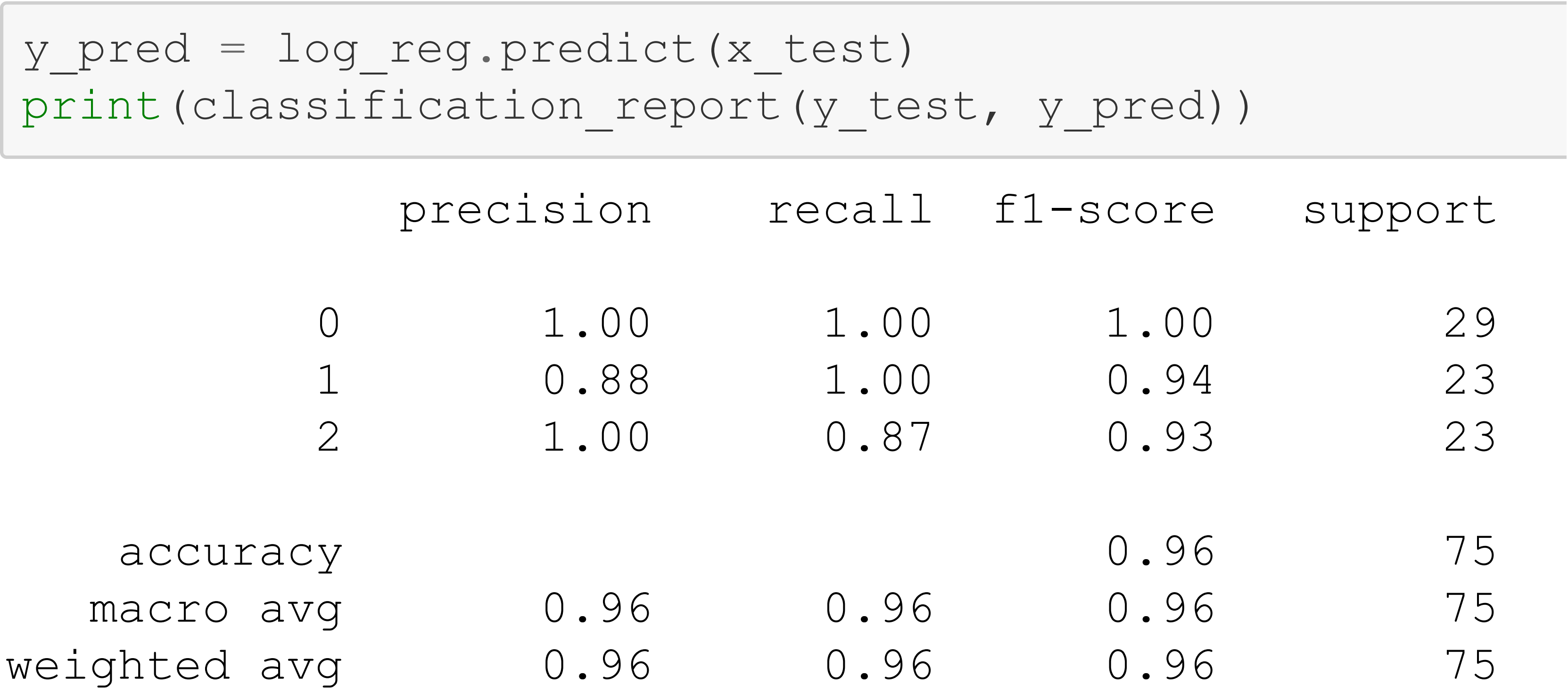
Fraction of positive instances that are correctly classified.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

“efficiency”

Iris classification example:
precision and recall for softmax classification

see sklearn.metrics.classification_report



```
y_pred = log_reg.predict(x_test)
print(classification_report(y_test, y_pred))
```

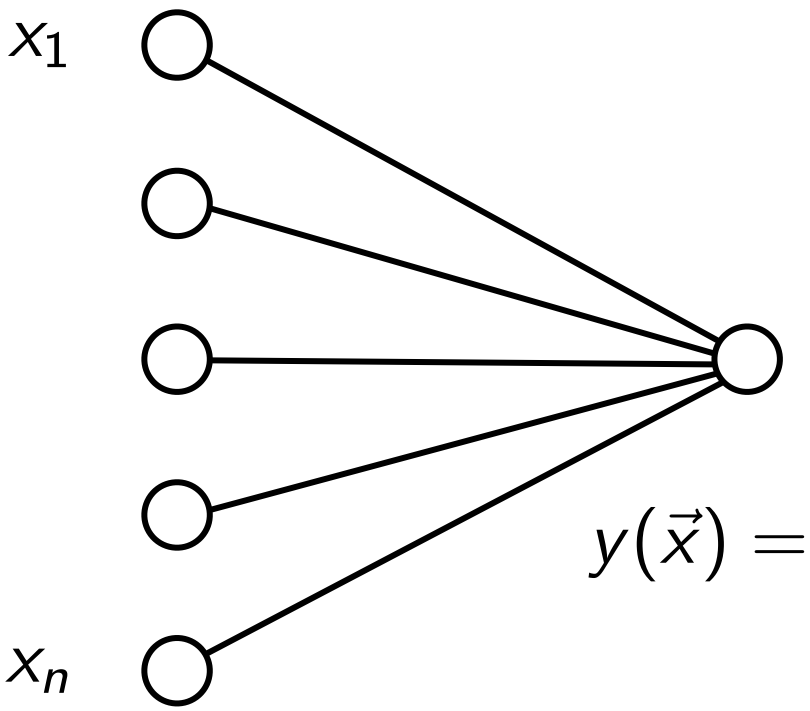
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        29
           1       0.88      1.00      0.94        23
           2       1.00      0.87      0.93        23

    accuracy                           0.96        75
   macro avg       0.96      0.96      0.96        75
weighted avg       0.96      0.96      0.96        75
```

# Perceptron (1)

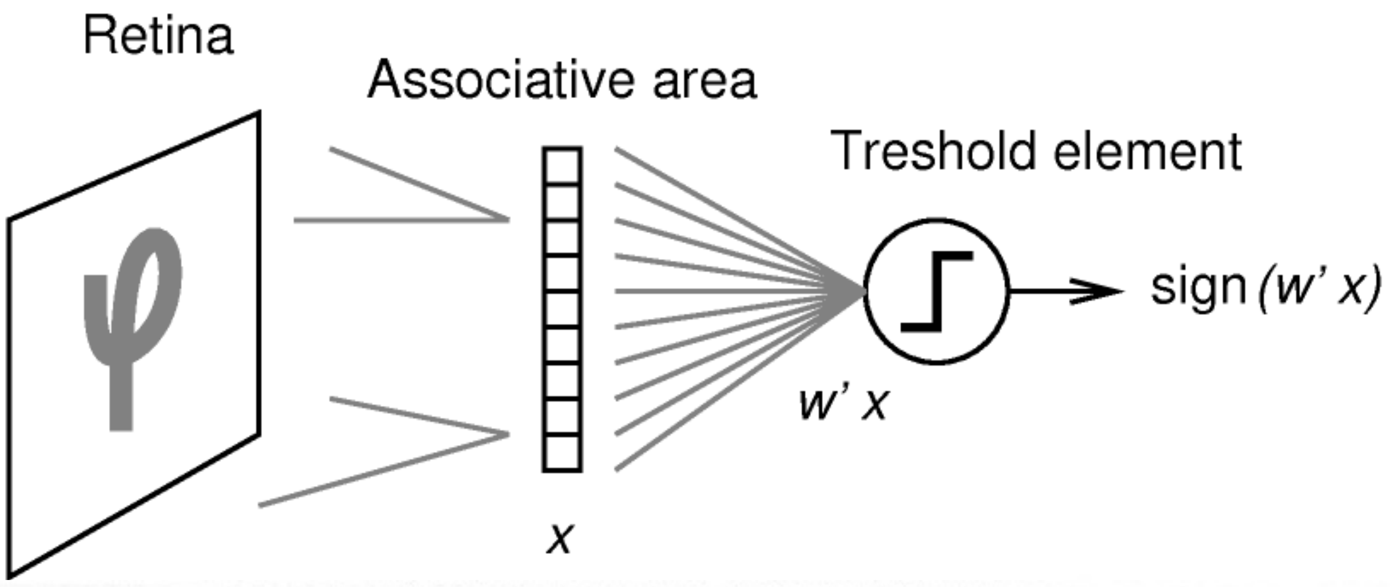
Rosenblatt, 1957



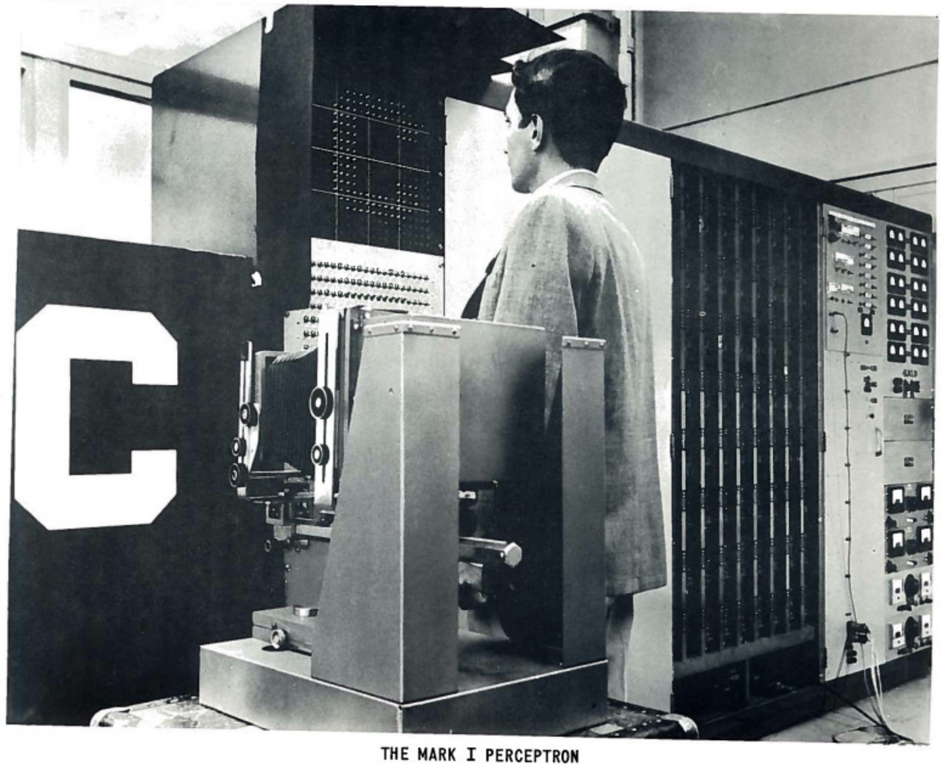
$$y(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

Output: “binary classifier”

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



The perceptron was designed for image recognition. It was first implemented in hardware (400 photocells, weights = potentiometer settings).

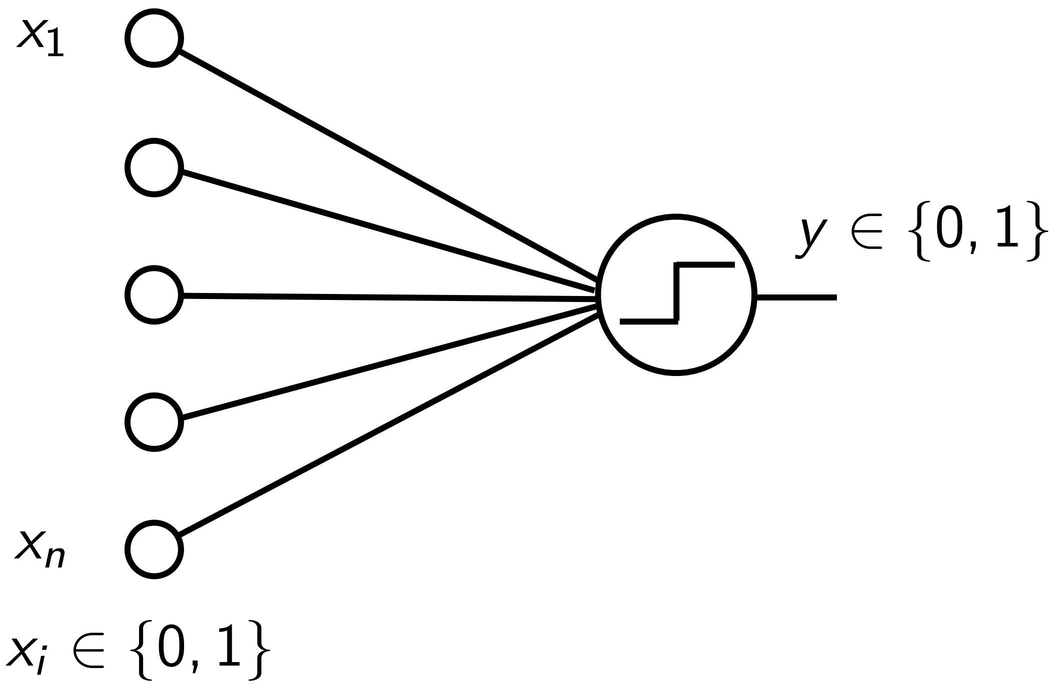


Mark 1 Perceptron. Source: Rosenblatt, Frank (1961) Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms

# Perceptron (2)

## McCulloch–Pitts (MCP) neuron (1943)

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

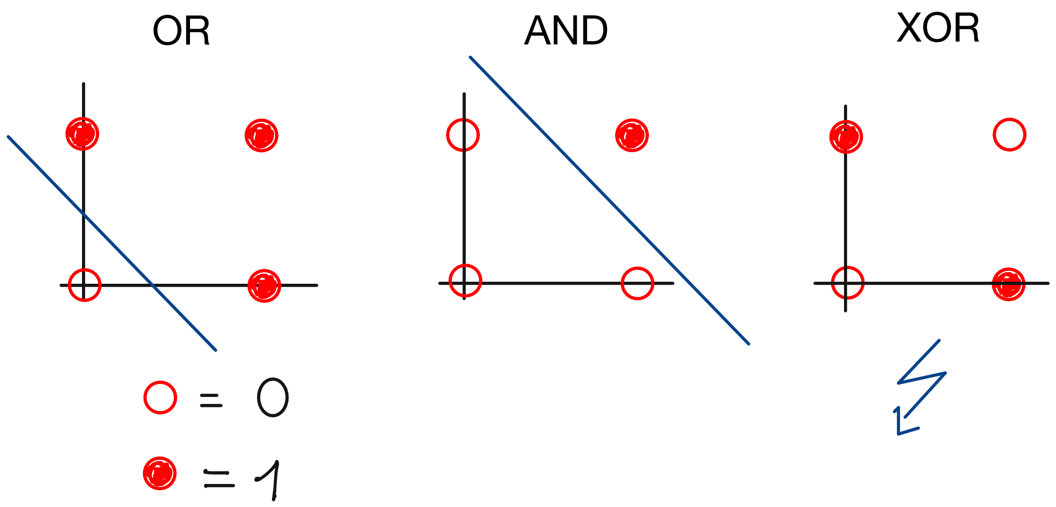


## Improvements by Rosenblatt:

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

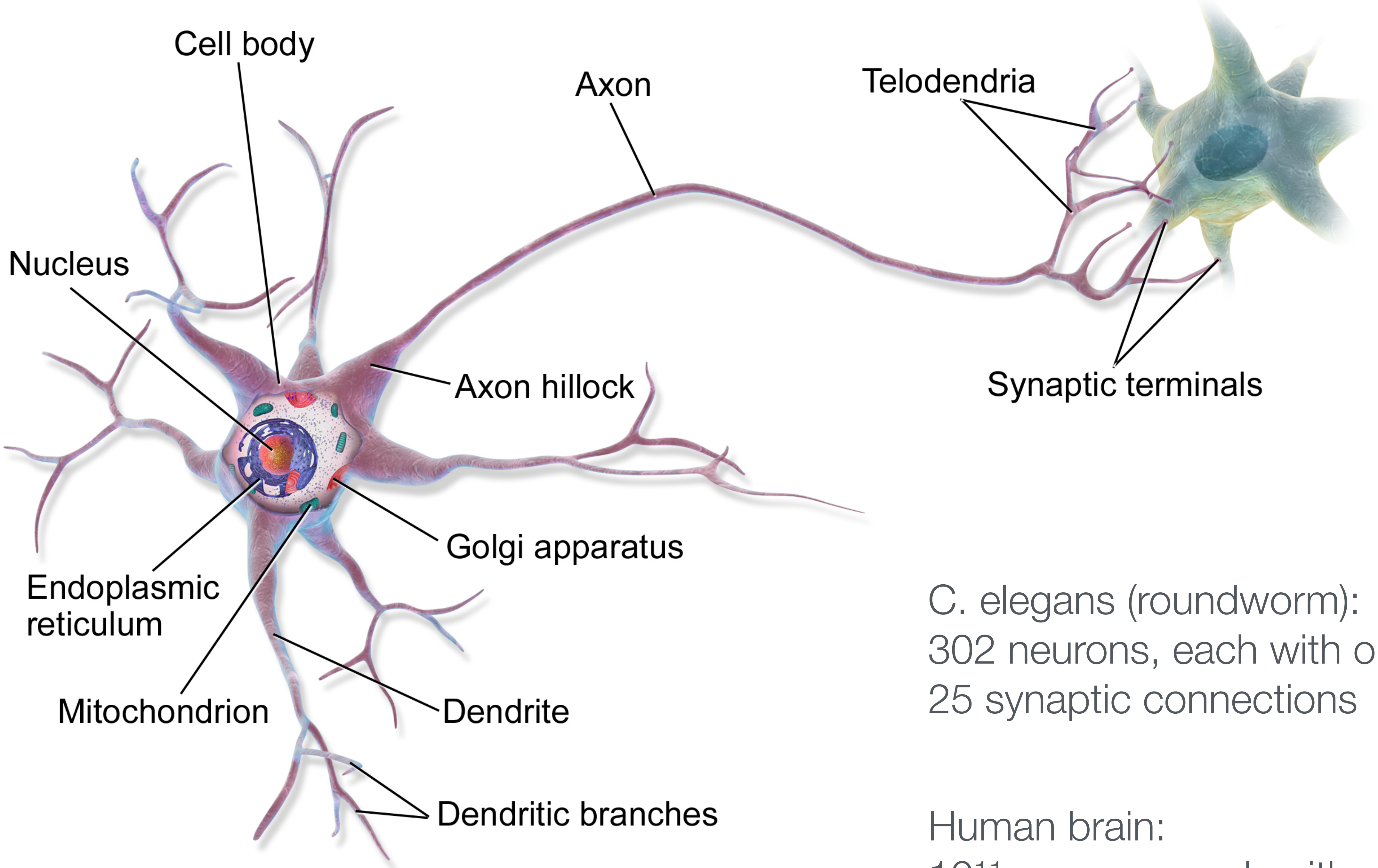
## Shortcoming of the perceptron: it cannot learn the XOR function

Minsky, Papert, 1969



XOR: not linearly separable

# The biological inspiration: the neuron



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

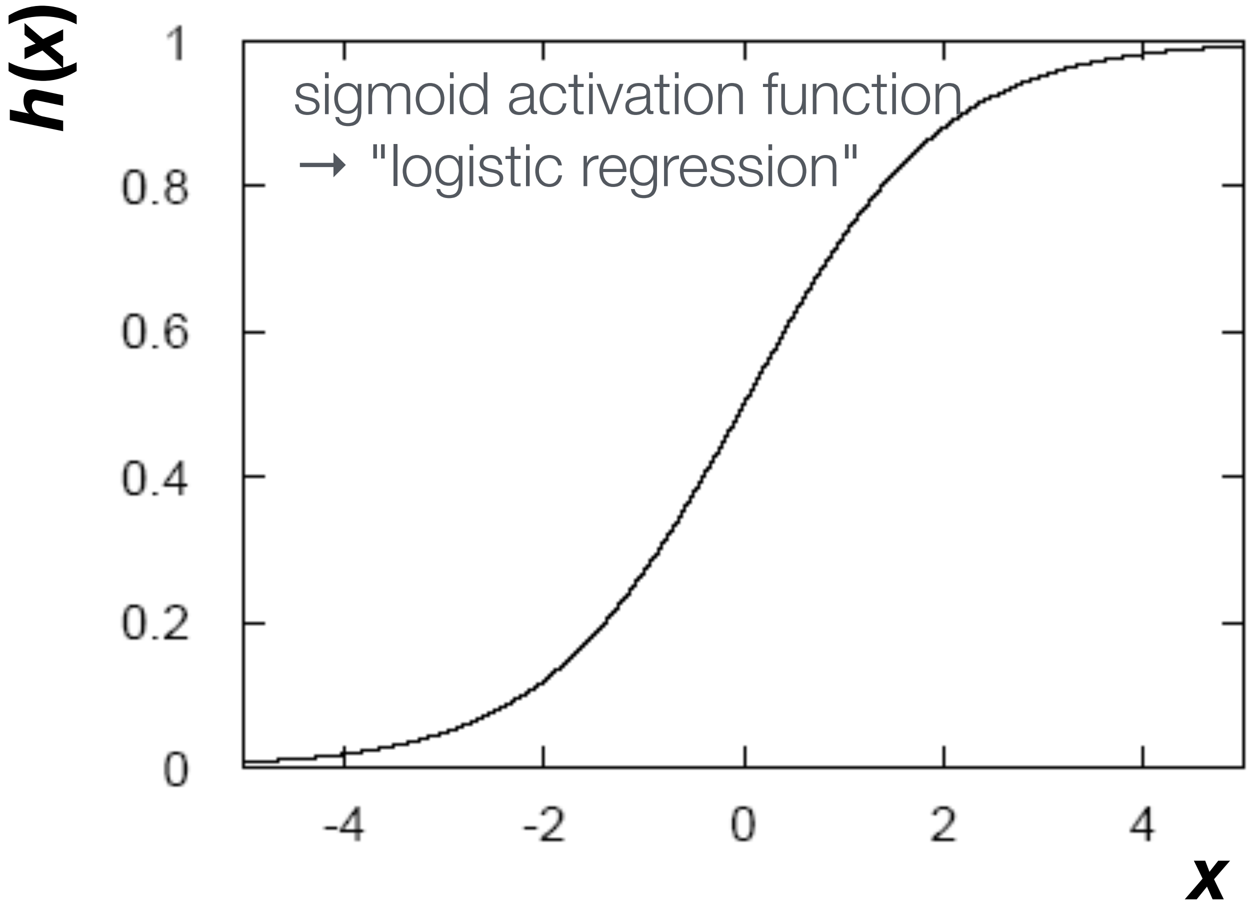
Human brain:
$10^{11}$ neurons, each with on average
7000 synaptic connections

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

# Non-linear transfer / activation function

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

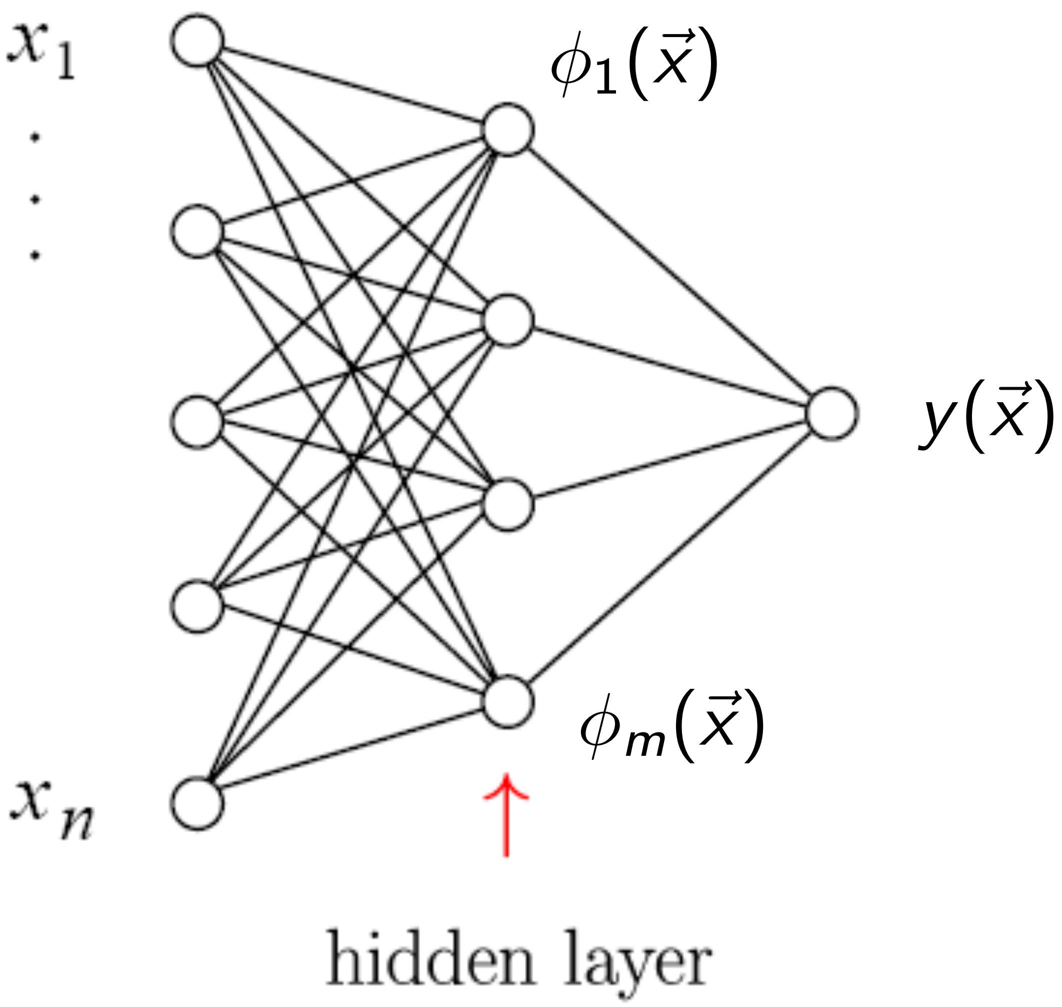
Examples for *h*: $\frac{1}{1+e^{-x}}$ ("sigmoid" or "logistic" function), $\tanh x$



Non-linear activation function 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

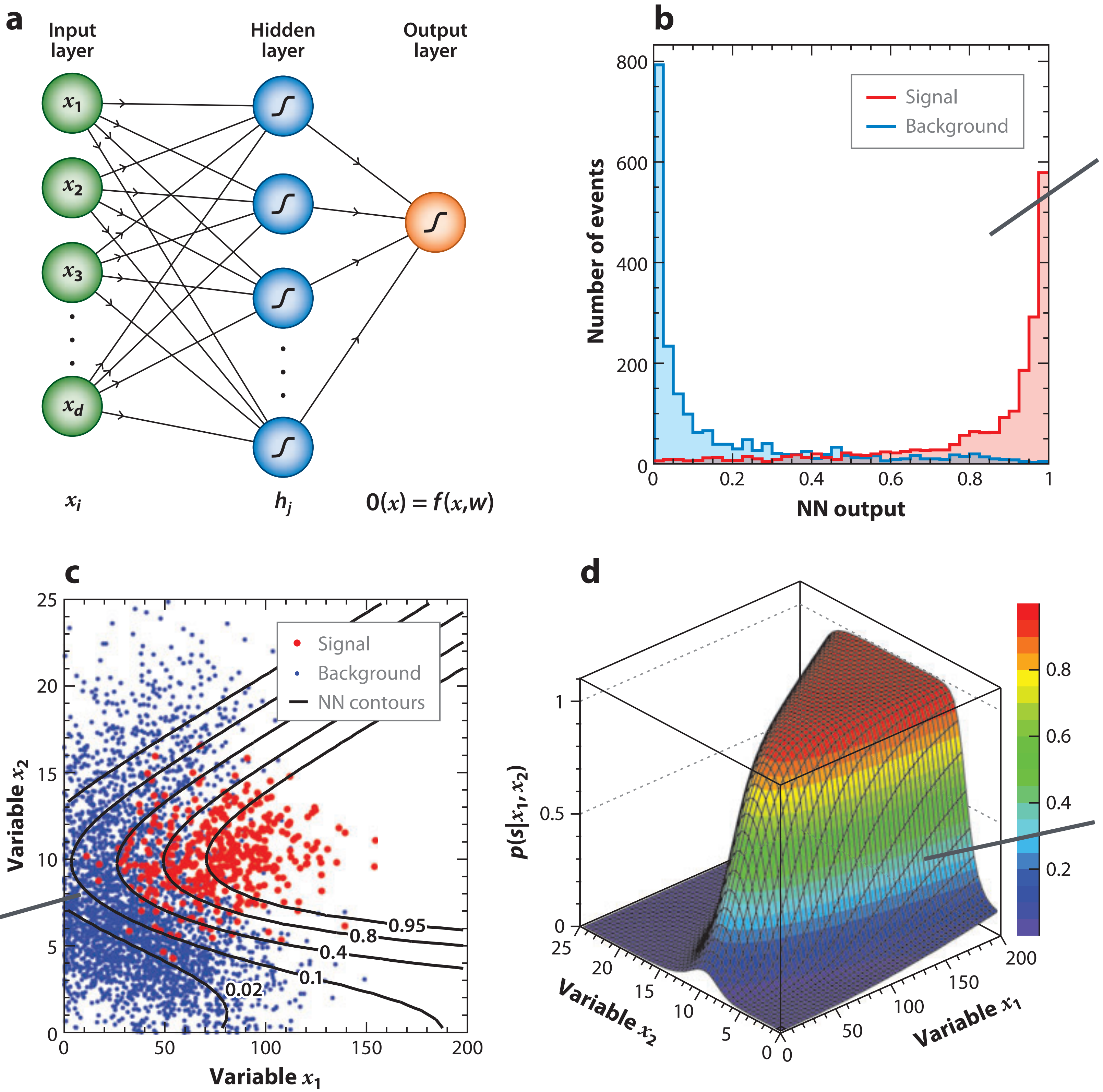
$$\phi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right)$$

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

Straightforward to generalize to multiple hidden layers

# Neural network output and decision boundaries

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

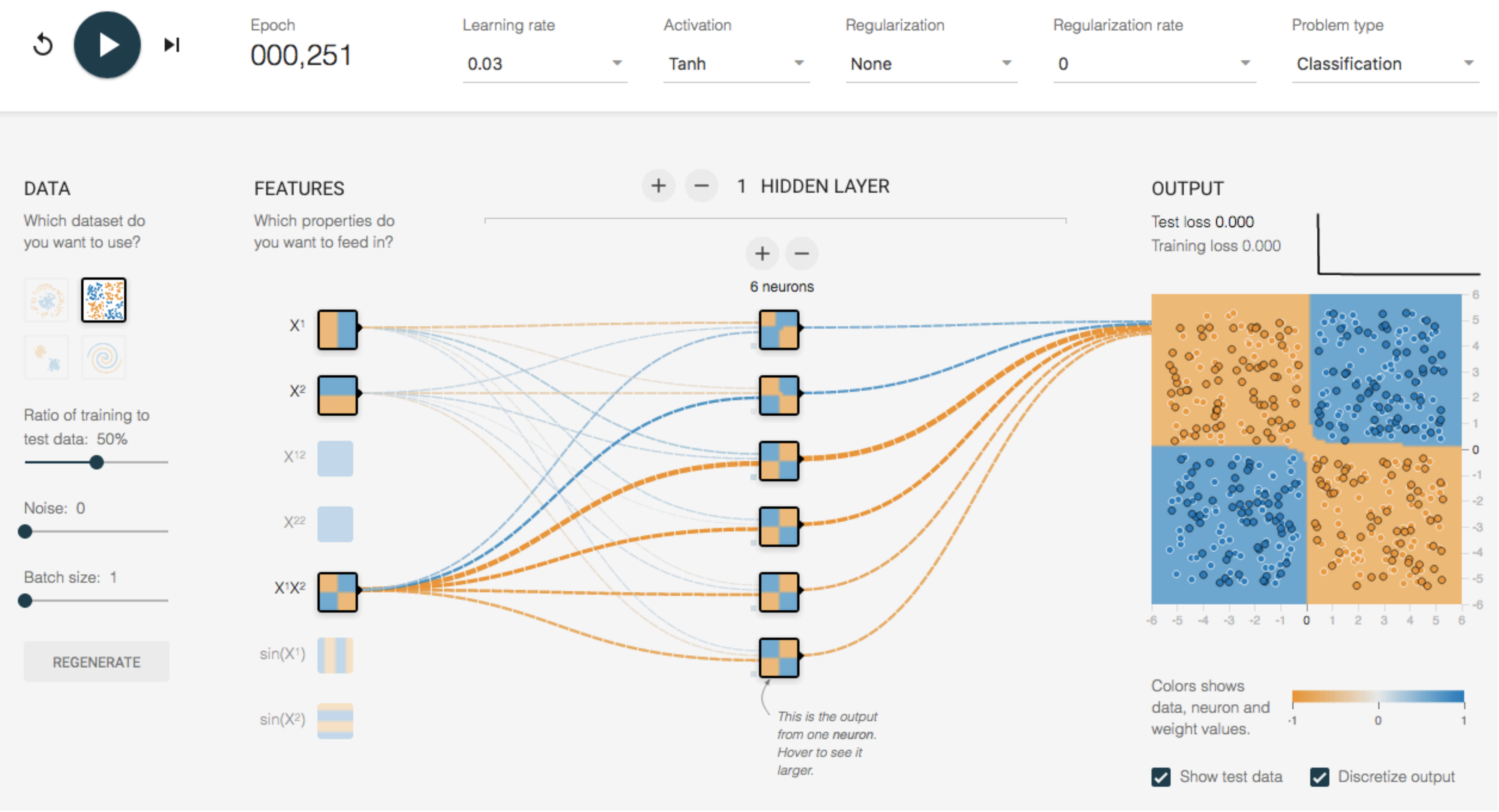


output of neural network

decision boundaries for different cuts on NN output

signal probability p(s | $x_1$, $x_2$)

# Fun with neural nets in the browser



Epoch 000,251

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 1

REGENERATE

FEATURES

Which properties do you want to feed in?

1 HIDDEN LAYER

6 neurons

*This is the output from one neuron. Hover to see it larger.*

OUTPUT

Test loss 0.000

Training loss 0.000

Colors shows data, neuron and weight values.

☑ Show test data ☑ Discretize output

[http://playground.tensorflow.org](http://playground.tensorflow.org)

# Network training

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

$t_a$ : 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):

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

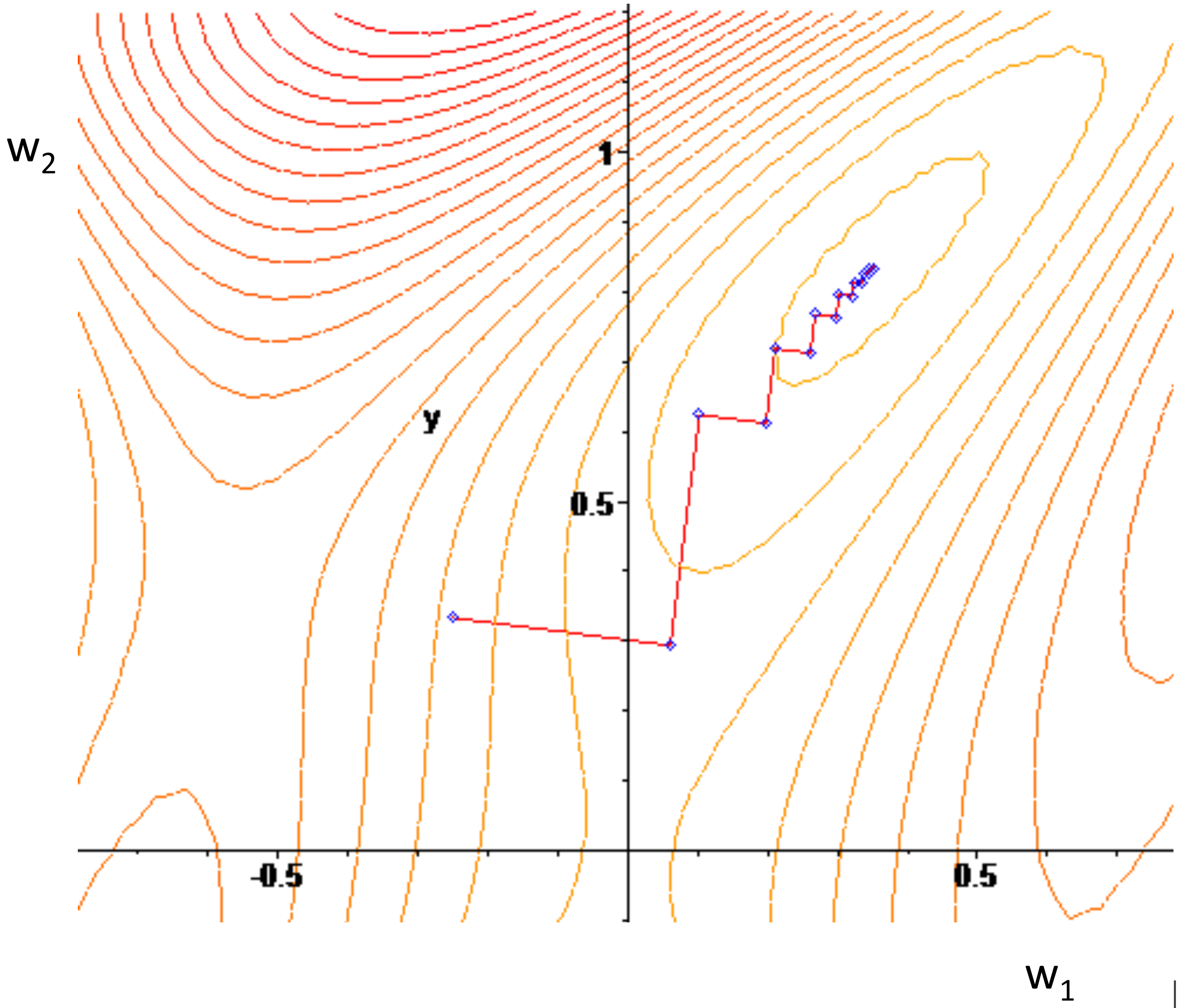
Weights are determined by minimizing the loss function (also called error function)

# Back-propagation (1)

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

$$\vec{w}^{(\tau+1)} = \vec{w}^{(\tau)} - \underbrace{\eta}_{\text{learning rate}} \nabla E_a(\vec{w}^{(\tau)})$$

Gradient descent:



# Back-propagation (2)

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:

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

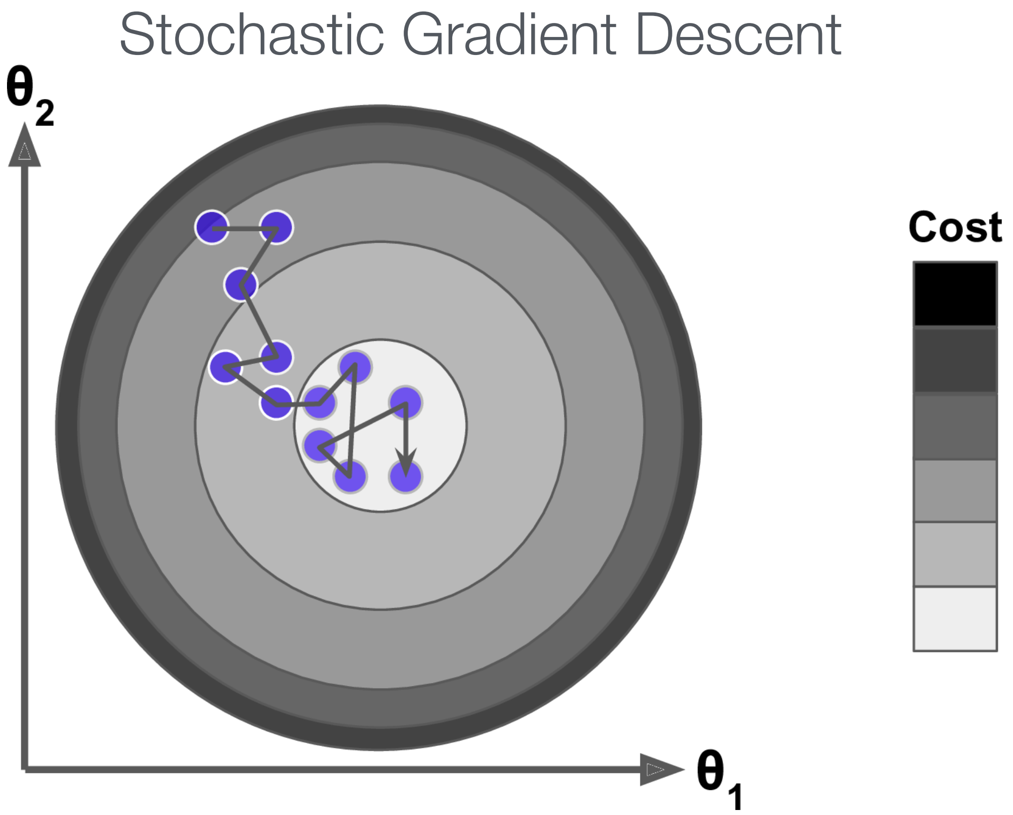
Further application of the **chain rule** gives weights from input to hidden layer.

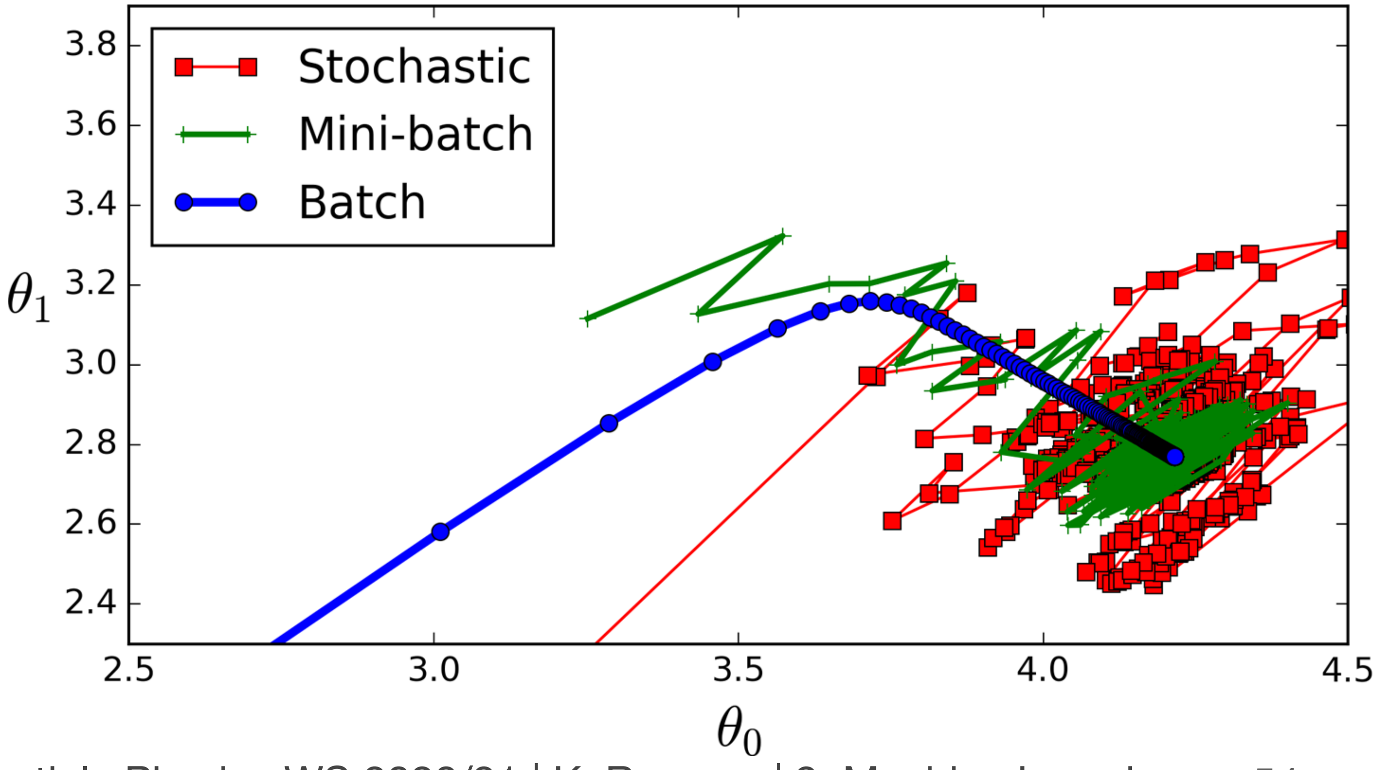
"Learning representations by back-propagating errors.",
Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams

# More on 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





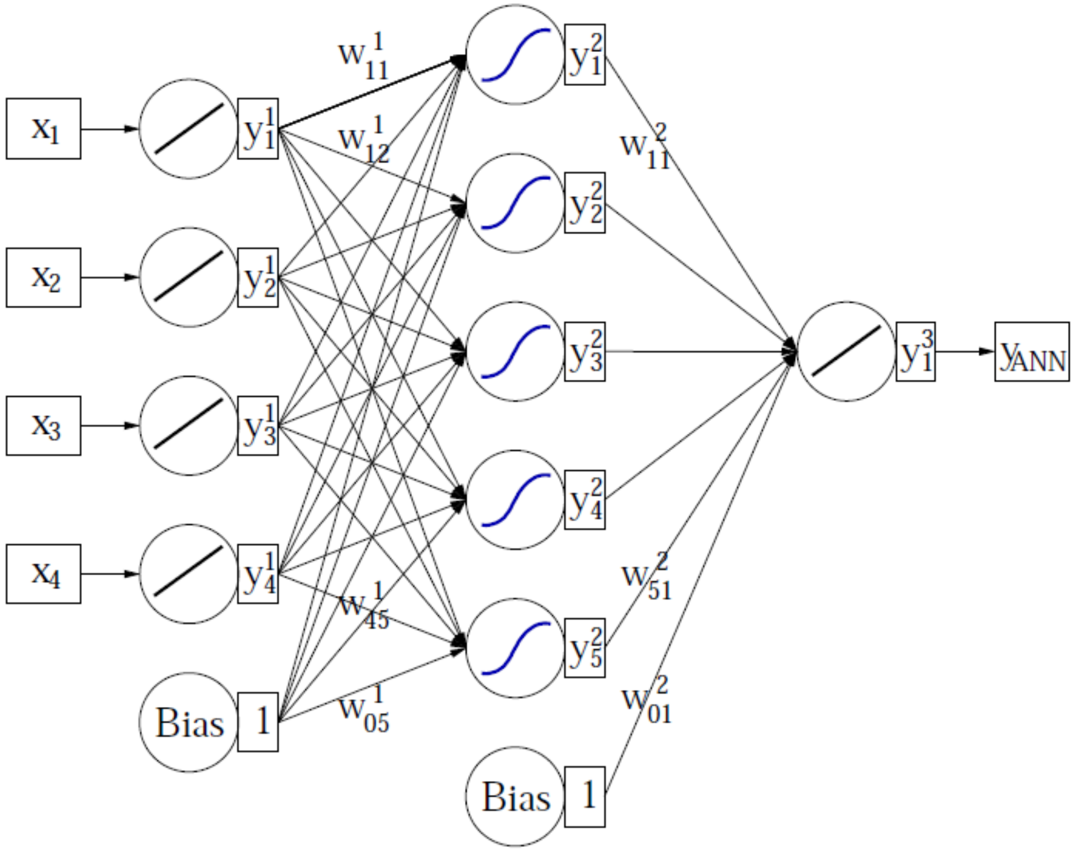
# Universal approximation theorem

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



# Deep neural networks

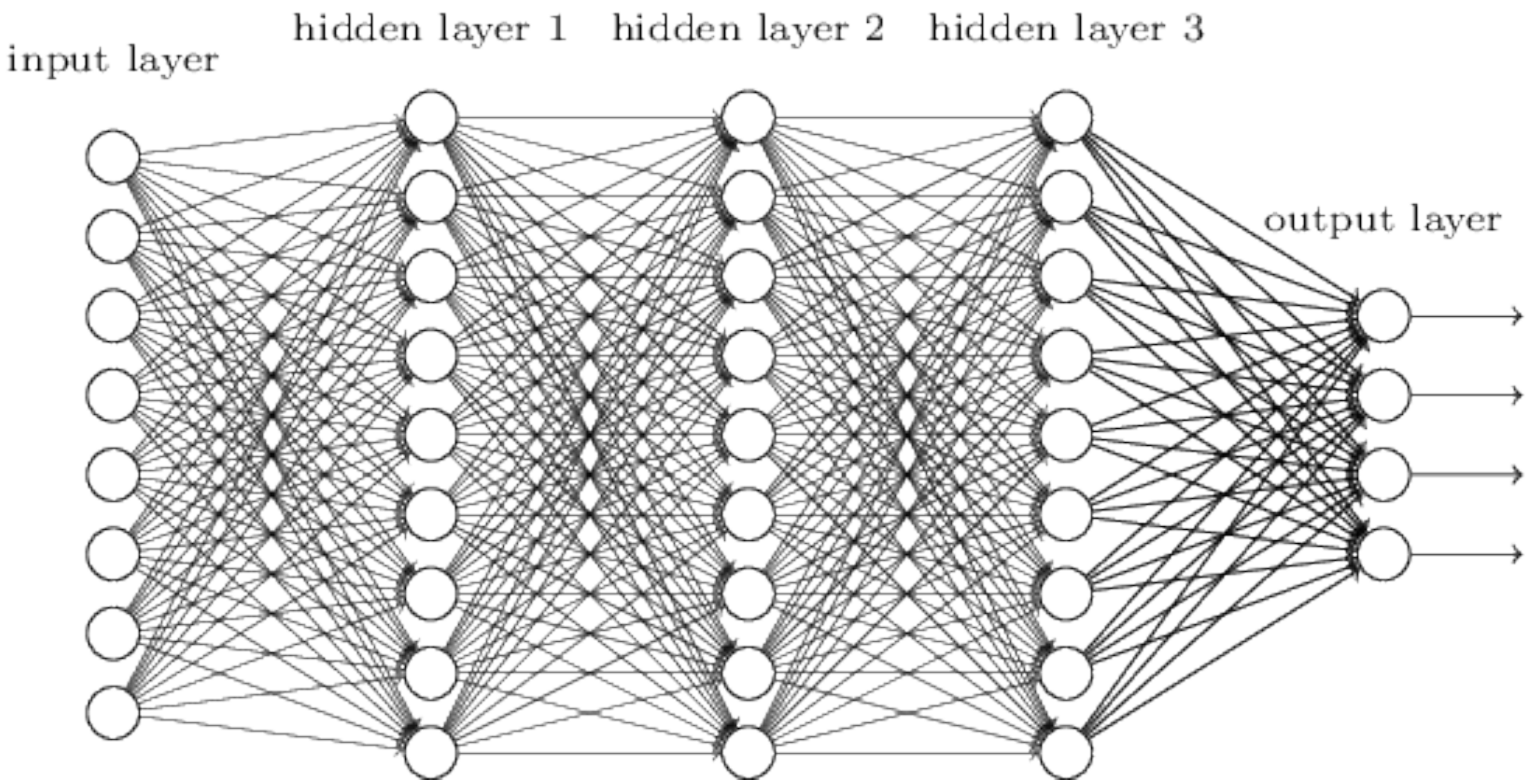
## Deep networks: many hidden layers with large number of neurons

### 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, …

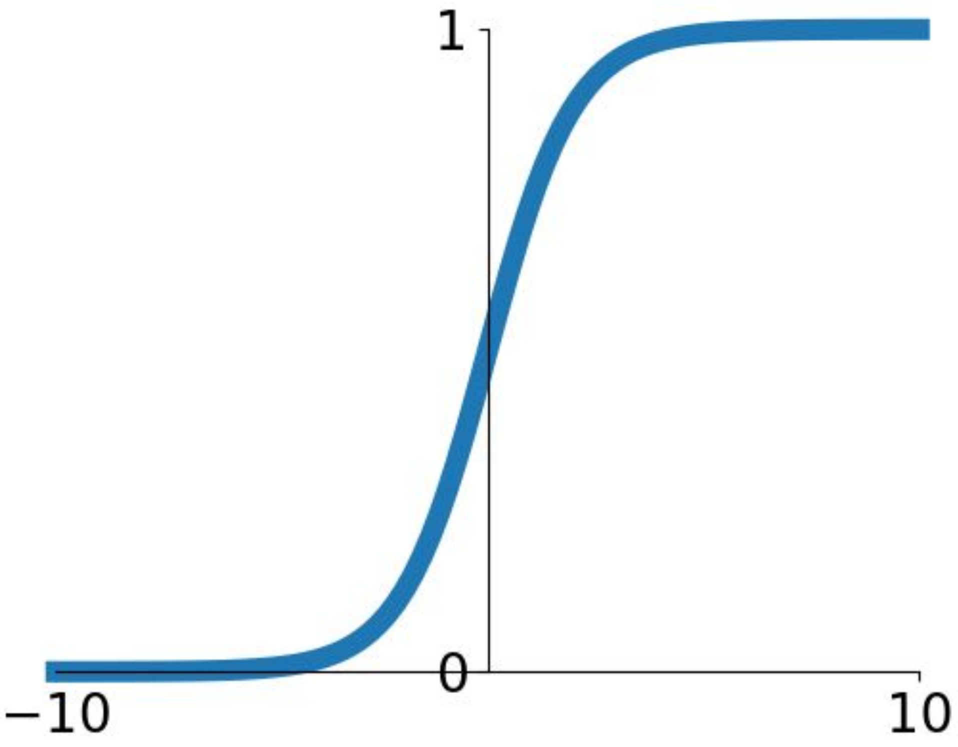


http://neuralnetworksanddeeplearning.com

# Drawbacks of the sigmoid activation function

http://cs231n.stanford.edu/slides

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



**Sigmoid**

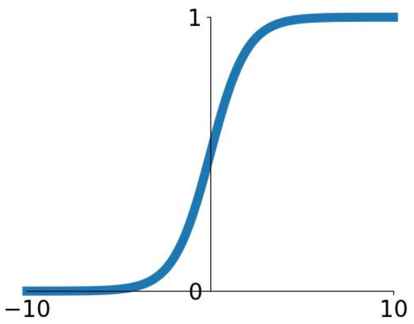
- Saturated neurons “kill” the gradients
- Sigmoid outputs are not zero-centered
- exp() is a bit compute expensive

# Activation functions

http://cs231n.stanford.edu/slides

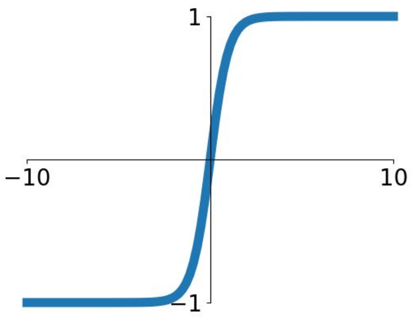
**Sigmoid**

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



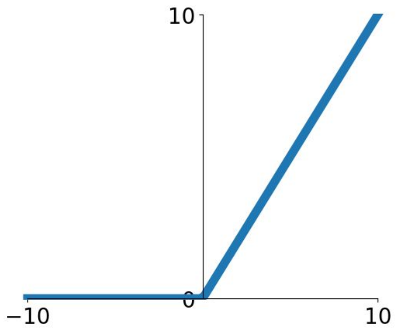
**tanh**

$\tanh(x)$



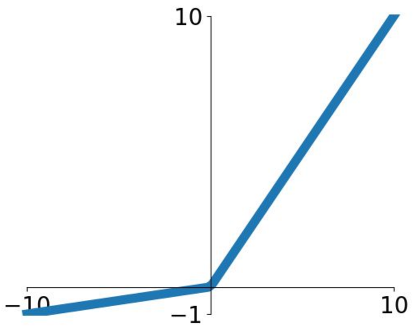
**ReLU**

$\max(0, x)$



**Leaky ReLU**

$\max(0.1x, x)$

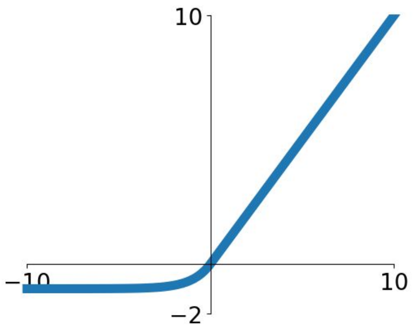


**Maxout**

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

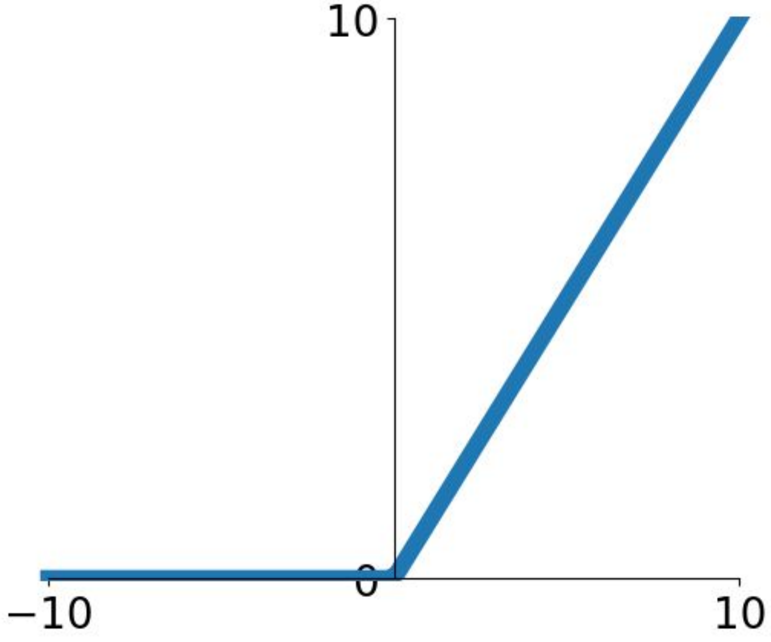
**ELU**

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



# ReLU

http://cs231n.stanford.edu/slides



**ReLU**
(Rectified Linear Unit)

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

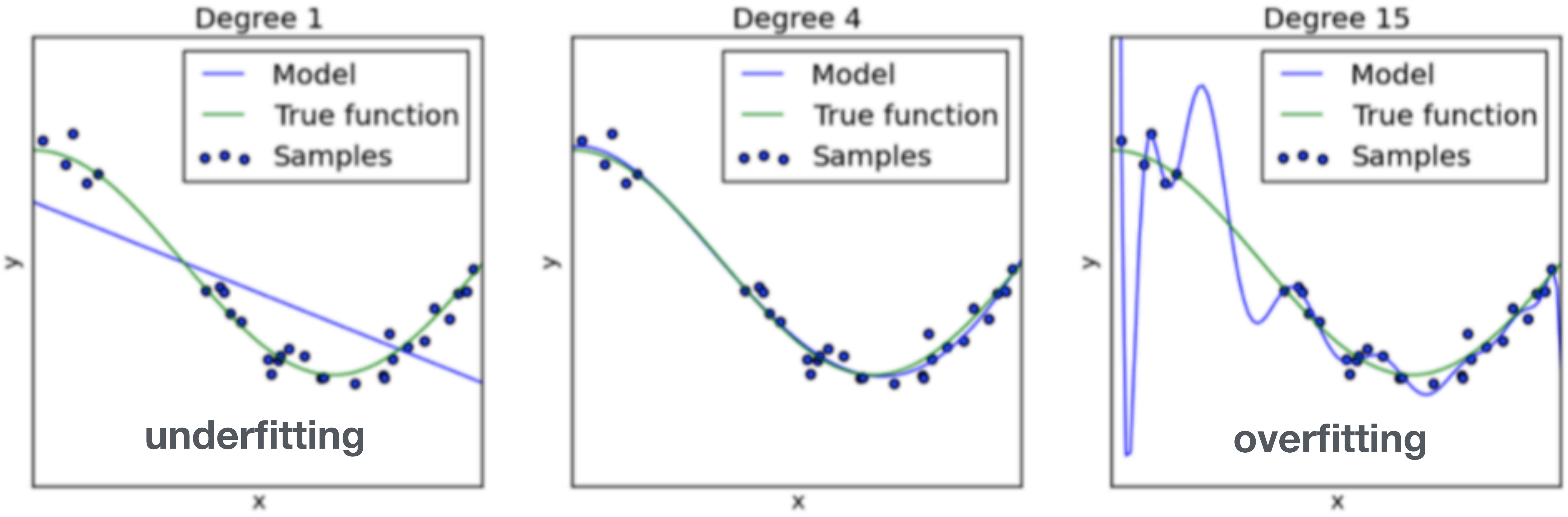
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

But: gradient vanishes for $x < 0$

# Bias-variance tradeoff

Goal: generalization of training data

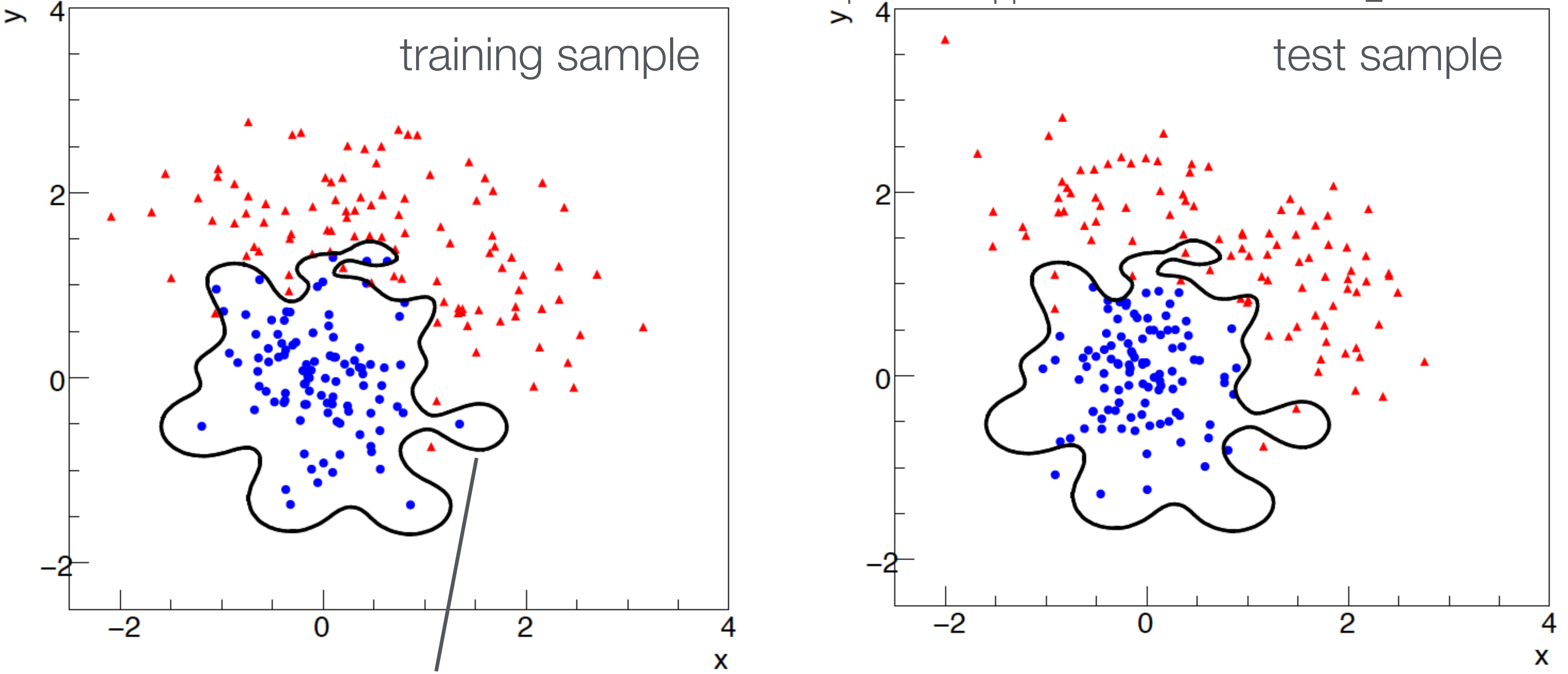
- Simple models (few parameters): danger of bias
  - Classifiers with a small number of degrees of freedom are less prone to statistical fluctuations: different training samples would result in similar classification boundaries ("small variance")
- Complex models (many parameters): danger of overfitting
  - large variance of decision boundaries for different training samples



# 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

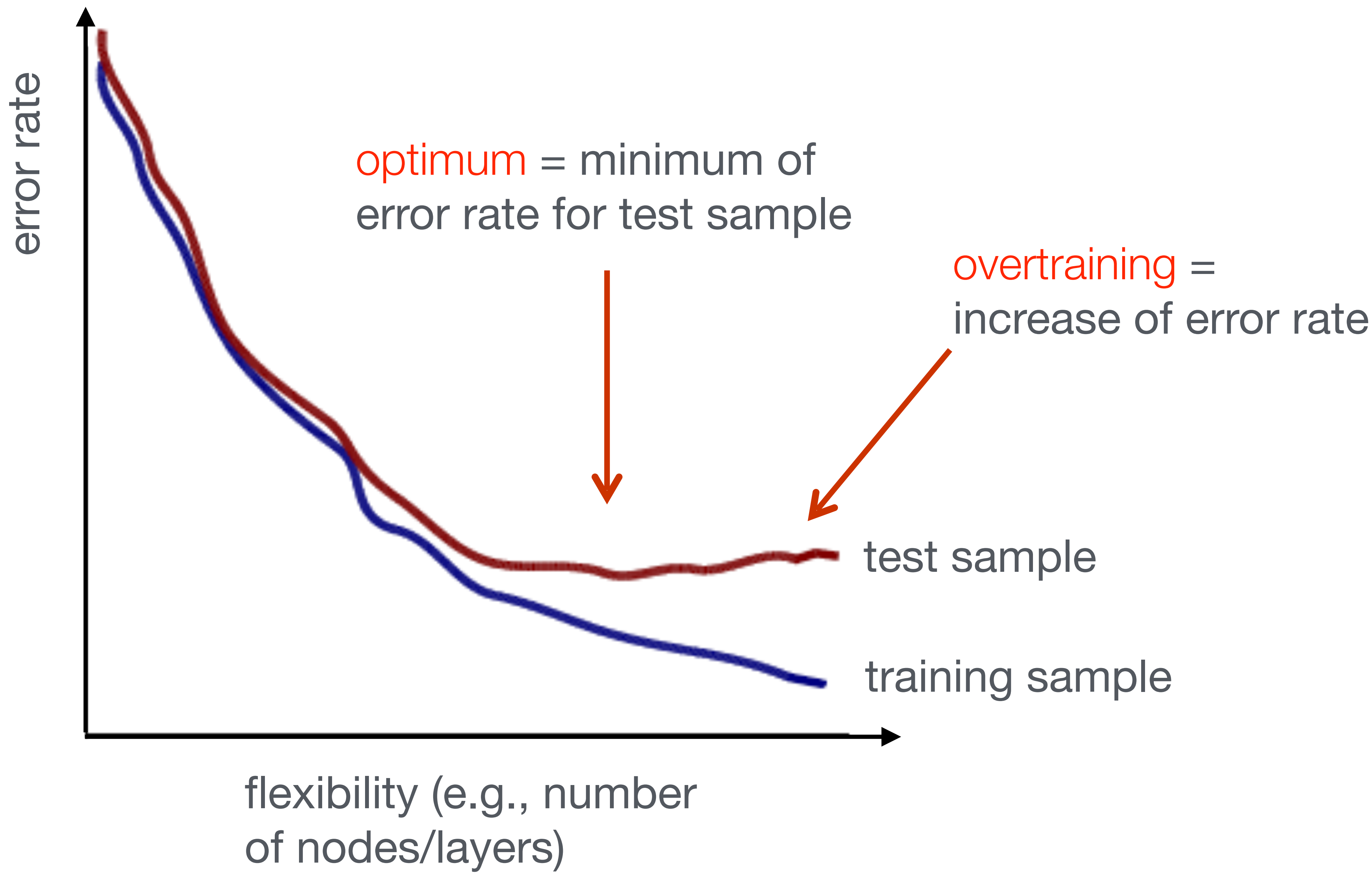


Network "learns" features that are merely statistical fluctuations in the training sample

# Monitoring overtraining

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

Monitor fraction of misclassified events (or loss function:)



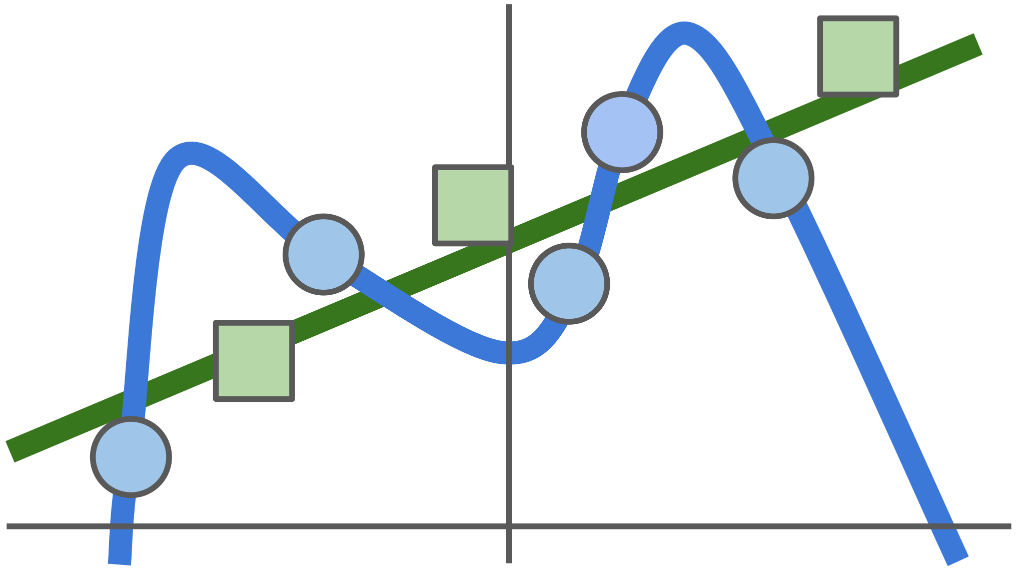
# Regularization: Avoid overfitting

http://cs231n.stanford.edu/slides

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

**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

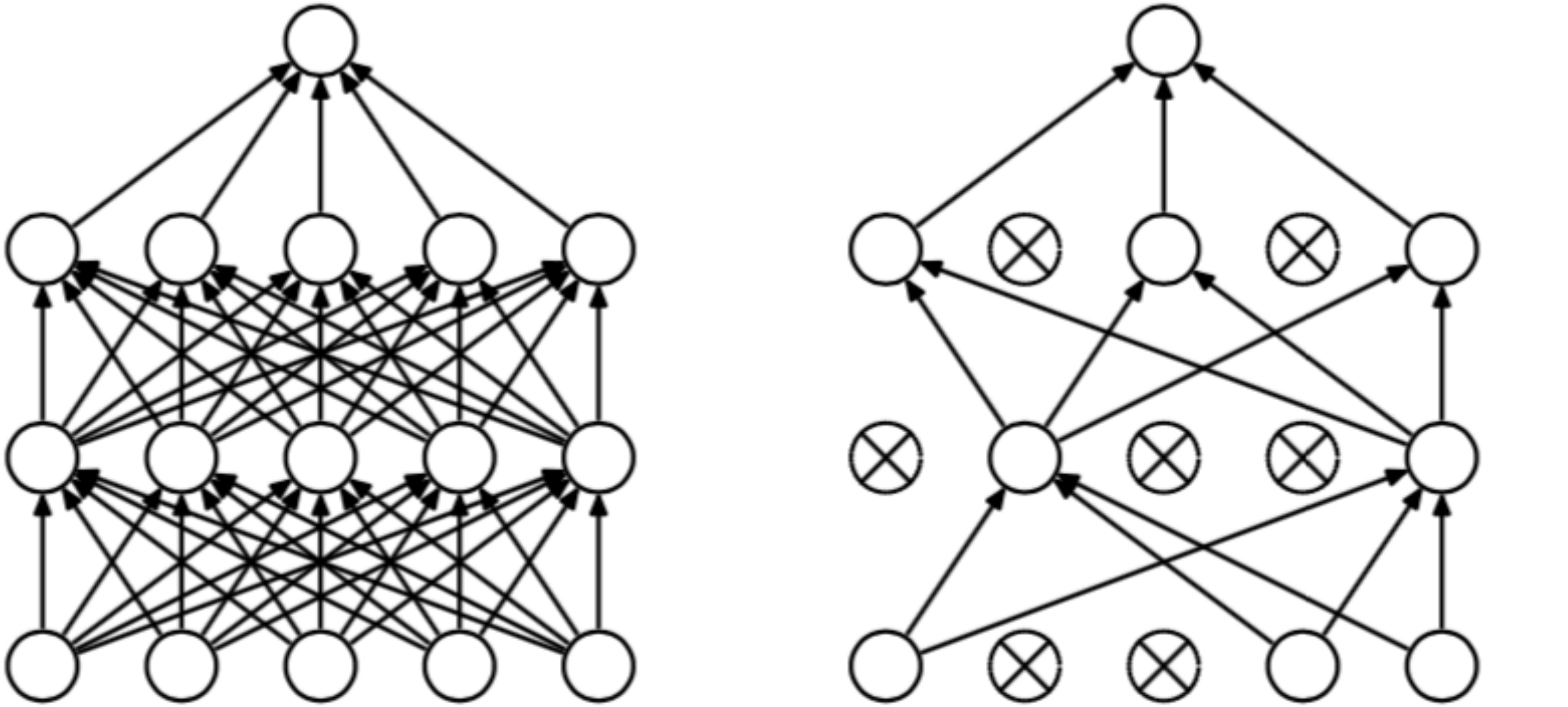
In common use:

**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

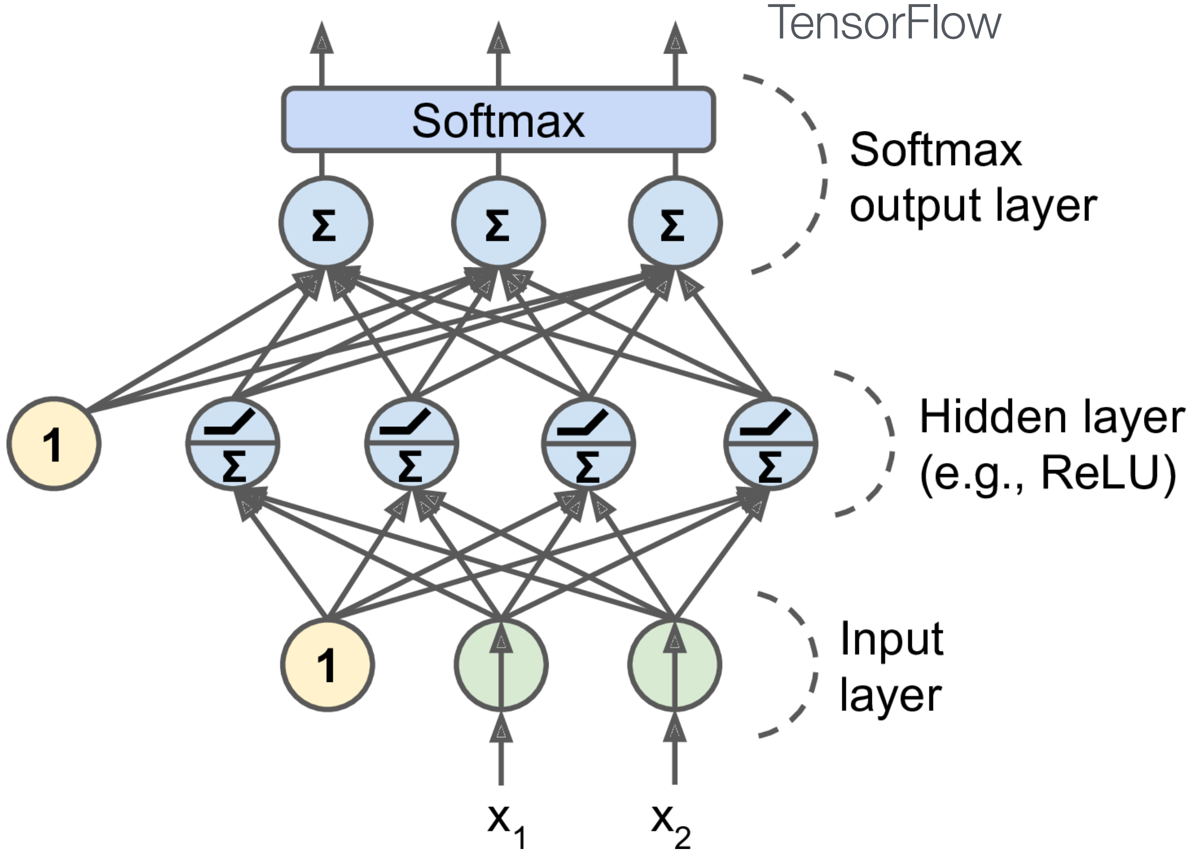
(b) After applying dropout.

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

# Xavier and He initialization

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

- 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



Layer with $n_{\text{in}}$ inputs connected to $n_{\text{out}}$ 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) | $r = \sqrt{2}\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$ | $\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}$ |

# Pros and cons of multi-layer perceptrons

**Pros:**

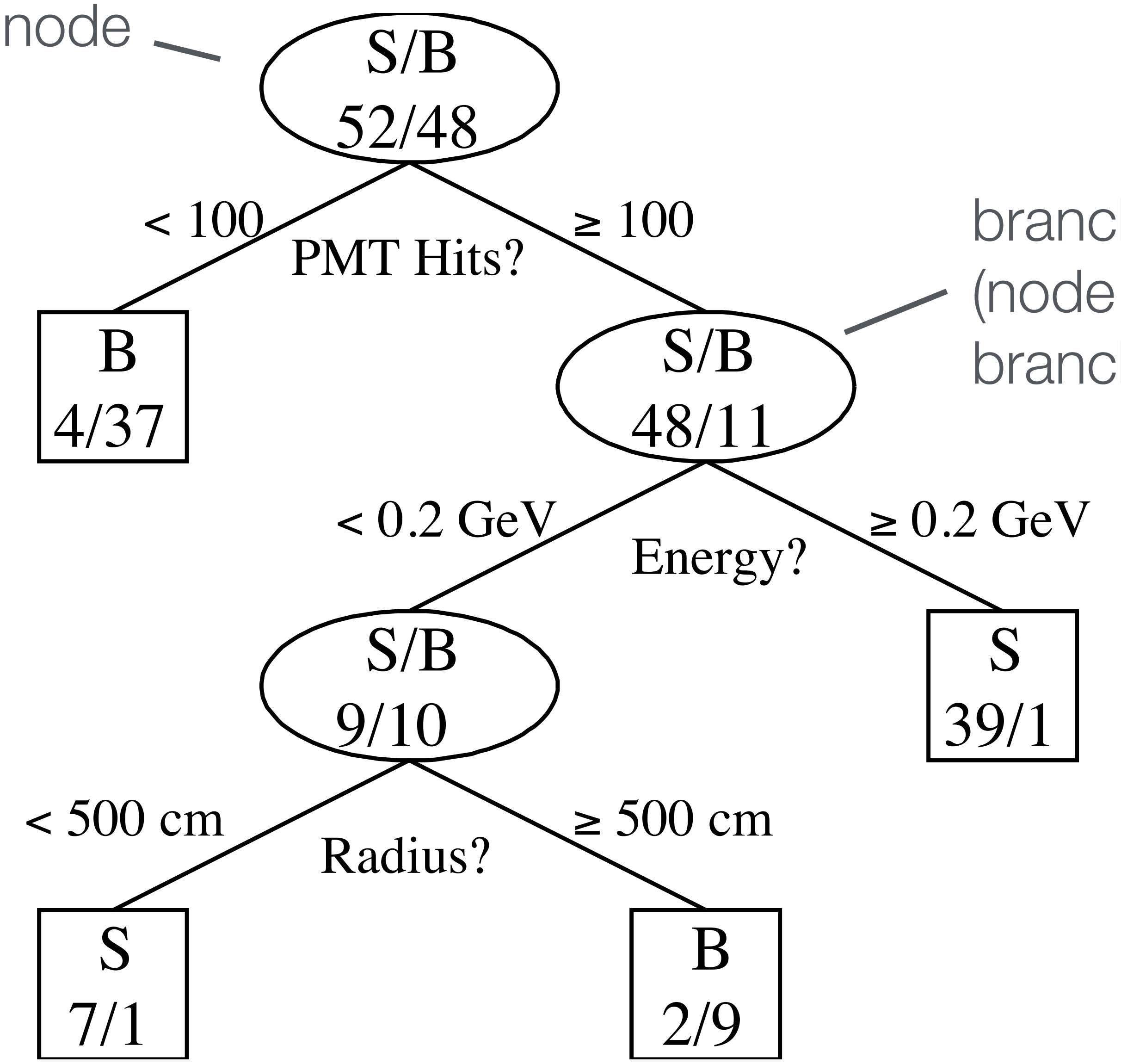
- Capability to learn non-linear models

**Cons:**

- Loss function can have several local minima
- Hyperparameters need to be tuned
  - number of layers, neurons per layer, and training iterations
- Sensitive to feature scaling
  - preprocessing needed (e.g., scaling of all feature to range [0,1])

# Decision trees

root node



S/B 52/48

PMT Hits?

< 100

≥ 100

B 4/37

S/B 48/11

Energy?

< 0.2 GeV

≥ 0.2 GeV

S/B 9/10

S 39/1

Radius?

< 500 cm

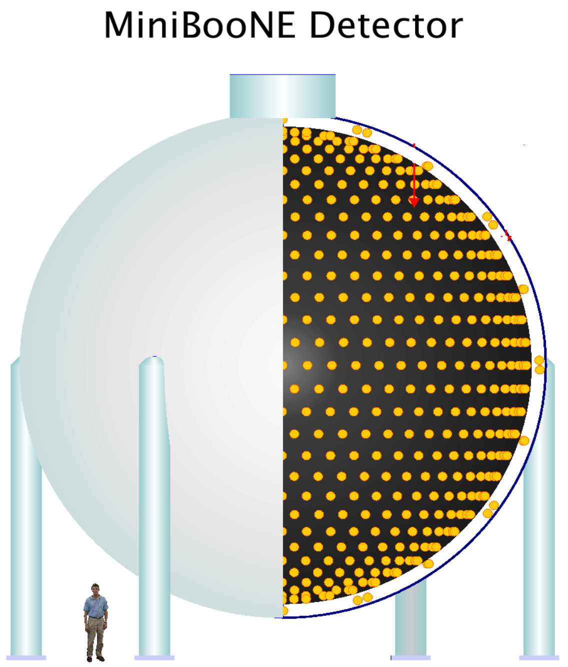
≥ 500 cm

S 7/1

B 2/9

branch node (node with further branching)

leaf node (no further branching)



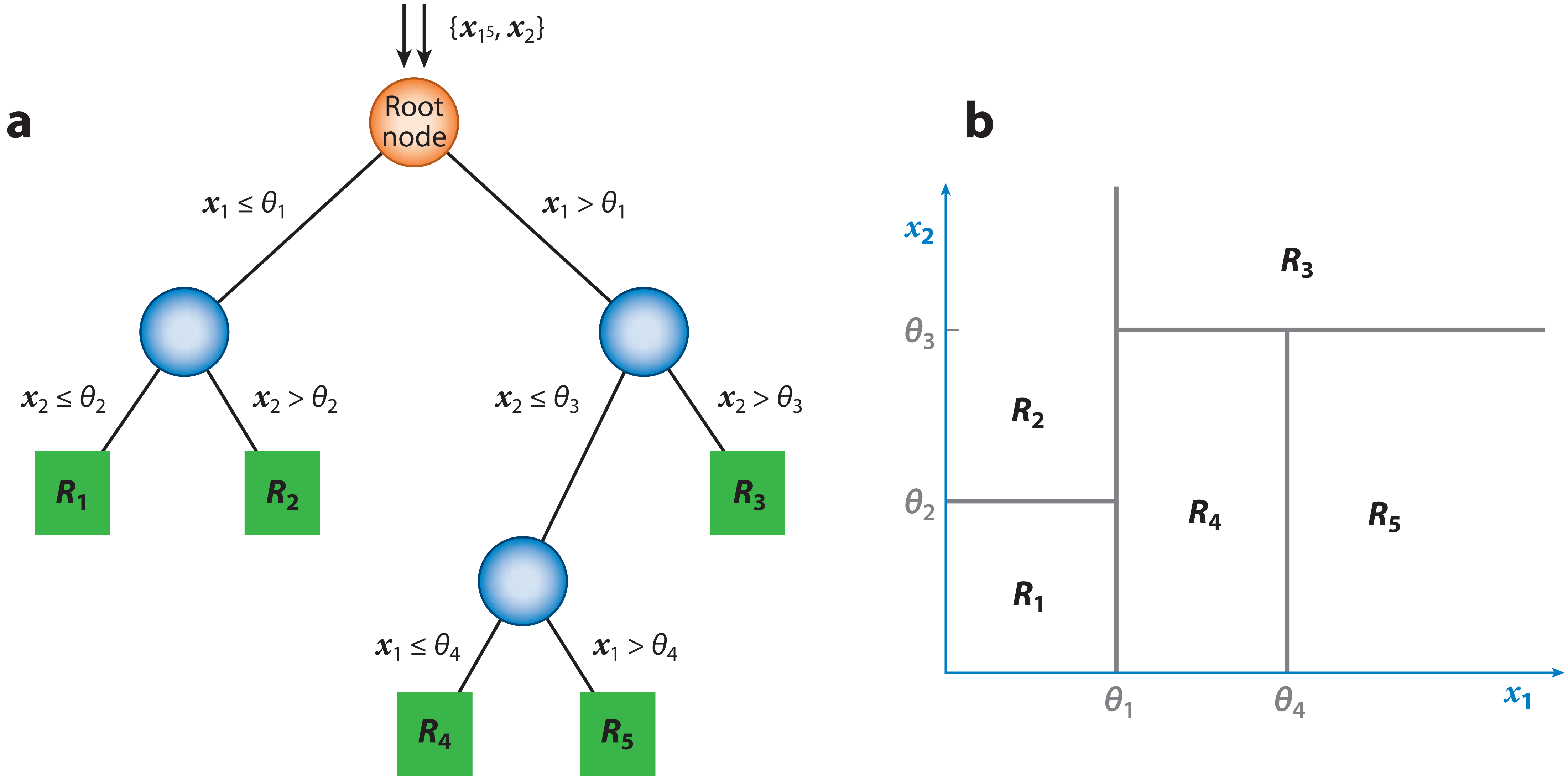
MiniBooNE: 1520 photomultiplier signals, goal: separation of $\nu_e$ from $\nu_\mu$ events

arXiv:physics/0508045v1

Leaf nodes classify events as either signal or background

# Decision trees

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* (or *Gini impurity*):

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

Here *p* is the purity:

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

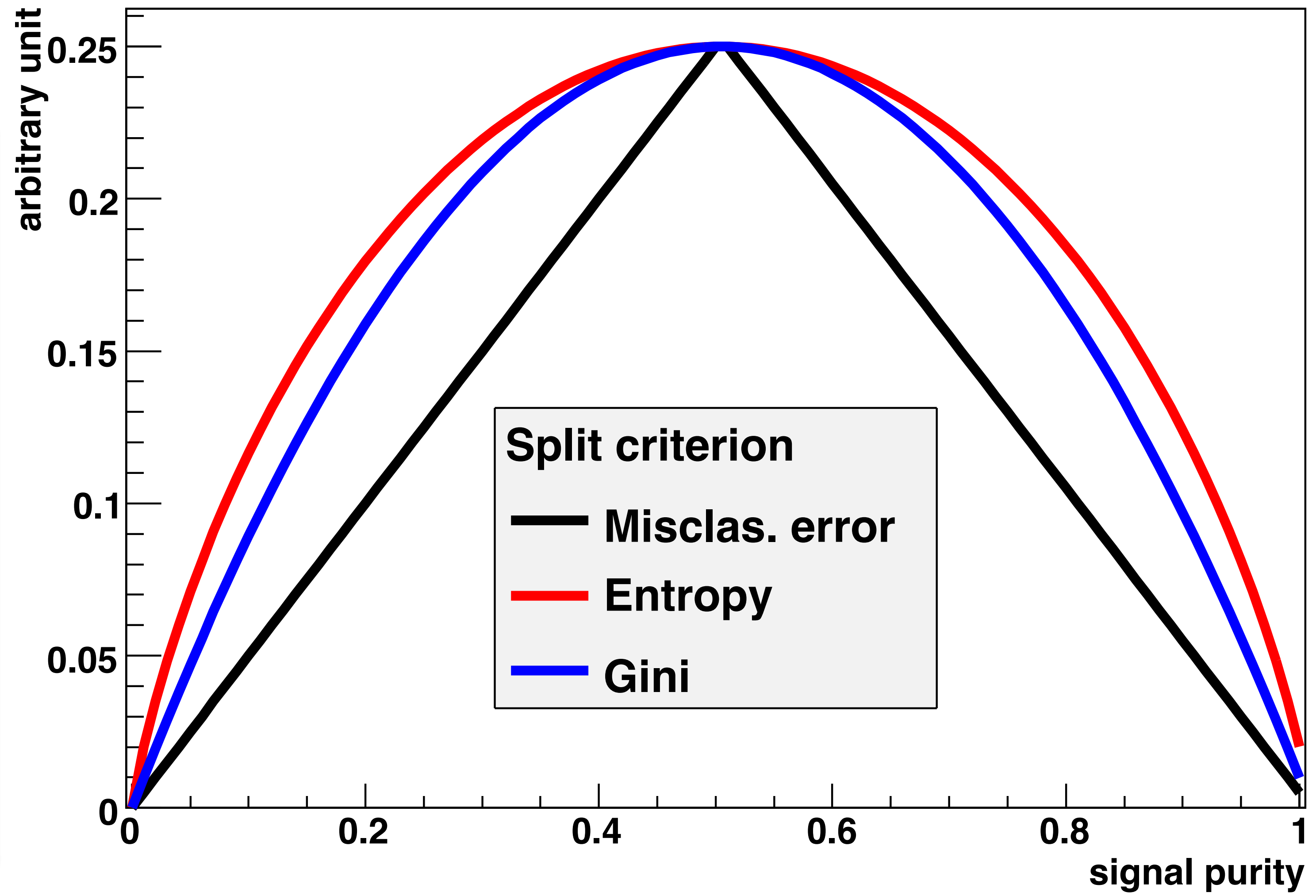
$w_i$ = weight of event $i$

[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



$p$

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

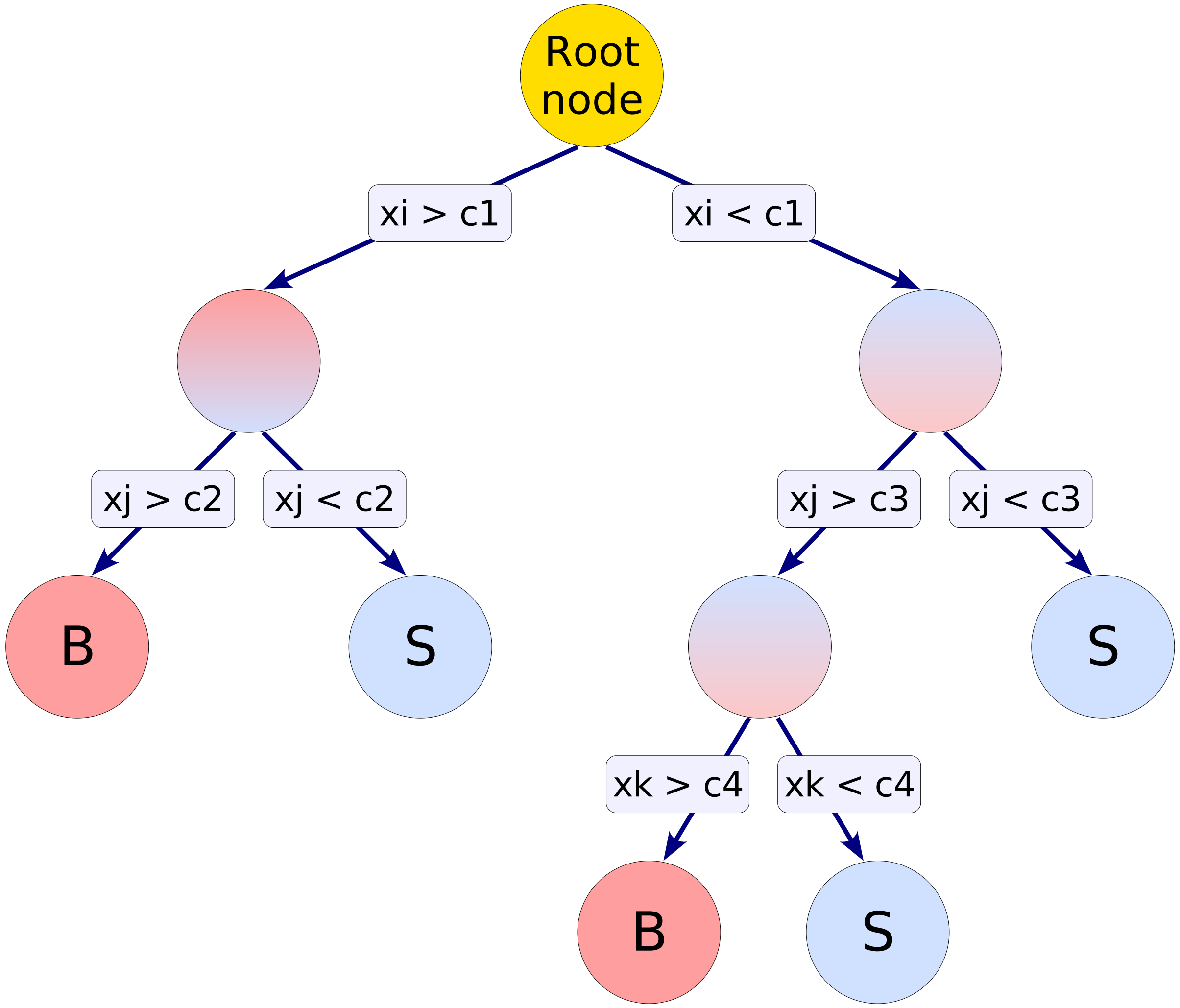
# Decision tree pruning

## 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, i.e., replace nodes and subtrees by leaves



# 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

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

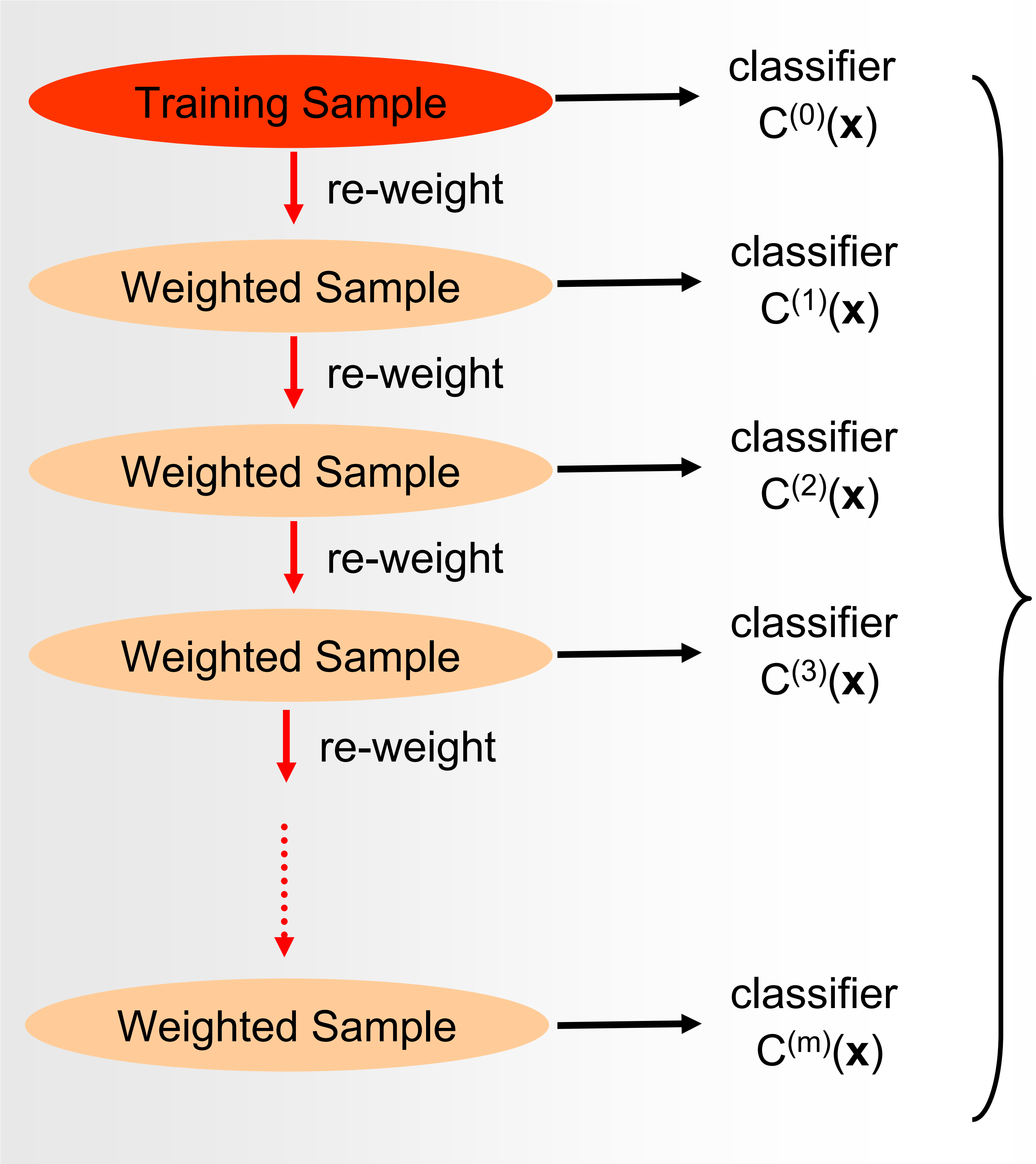
- Boosting
  - Train *N* models in sequence, giving more weight to examples not correctly classified by previous model
  - Take weighted average to classify examples

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

# Random forests

- “One of the most widely used and versatile algorithms in data science and machine learning” (arXiv:1803.08823v3)
- Use bagging to select random example subset
- Train a tree, but only use random subset of features at each split
  - this reduces the correlation between different trees
  - makes the decision more robust to missing data

# Boosted decision trees: Idea


Training Sample
classifier $C^{(0)}(\mathbf{x})$
re-weight
Weighted Sample
classifier $C^{(1)}(\mathbf{x})$
re-weight
Weighted Sample
classifier $C^{(2)}(\mathbf{x})$
re-weight
Weighted Sample
classifier $C^{(3)}(\mathbf{x})$
re-weight
Weighted Sample
classifier $C^{(m)}(\mathbf{x})$

Weight is increased if event was misclassified by the previous classifier

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

$$y(x) = \sum_{i}^{N_{\text{Classifier}}} w_i C^{(i)}(x)$$

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

# AdaBoost (short for *Adaptive Boosting*)

Initial training sample

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

# AdaBoost: 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

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

# AdaBoost: 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

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

# Gradient boosting

- Like in AdaBoost, decision trees are iteratively added to an ensemble
- Can be applied to classification and regression
- Basic idea
  - Train a first decision tree
  - Then train a second one on the residual errors made by the first tree
  - And so on …

Labeled training data: $\{\vec{x}_i, y_i\}$

Model prediction at iteration *m*: $F_m(\vec{x}_i)$

New model: $F_{m+1}(\vec{x}) = F_m(\vec{x}) + h_m(\vec{x})$

Find $h_m(\vec{x})$ by fitting it to

$$\{(\vec{x}_1, y_1 - F_m(\vec{x}_1)), (\vec{x}_2, y_2 - F_m(\vec{x}_2)), \ldots (\vec{x}_n, y_n - F_m(\vec{x}_n))\}$$