

# Machine Learning

## Lecture 1: Introduction to Machine Learning

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

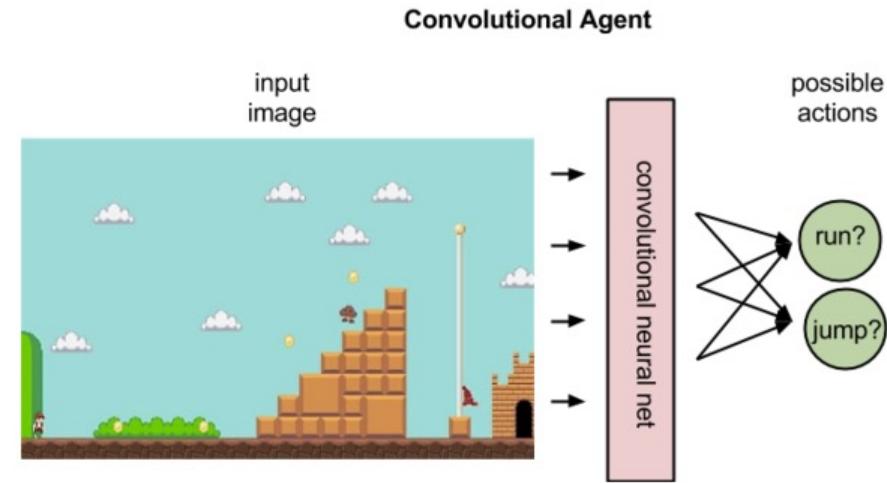
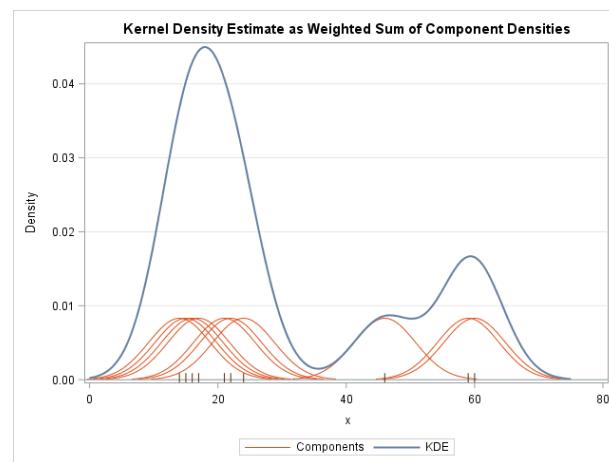
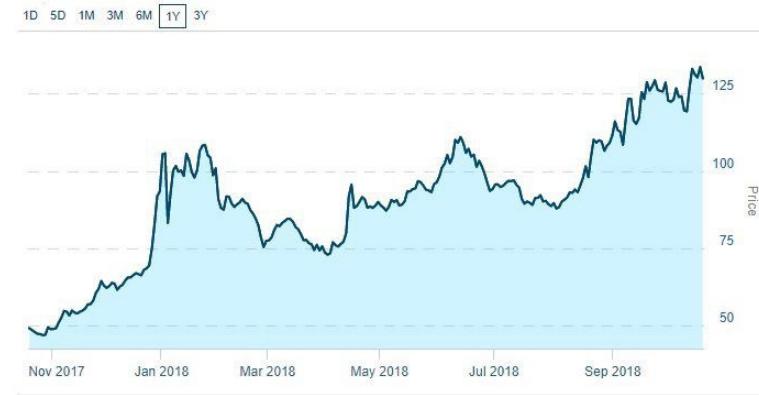
# Course outline

1. Introduction
2. Bayesian decision theory
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Lecture outline

- Introduction
  - Typical machine learning problems and selected examples
- Machine learning
  - Concepts, supervised learning, unsupervised learning
- Memory-based learning
- Model-based learning

# Typical machine learning problems



# Classification examples

- Object Recognition on IMAGENET (Russakovsky et al., 2015)



flamingo



cock



ruffed grouse



quail



partridge

...



Egyptian cat



Persian cat



Siamese cat



tabby



lynx

...



dalmatian



keeshond



miniature schnauzer



standard schnauzer

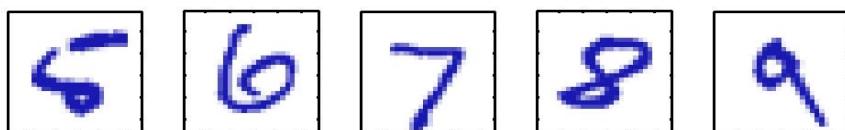
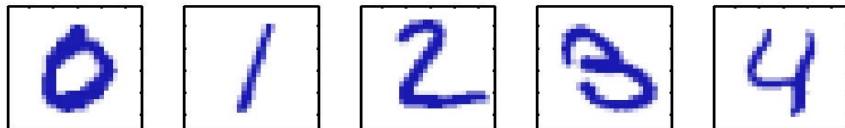


giant schnauzer

...

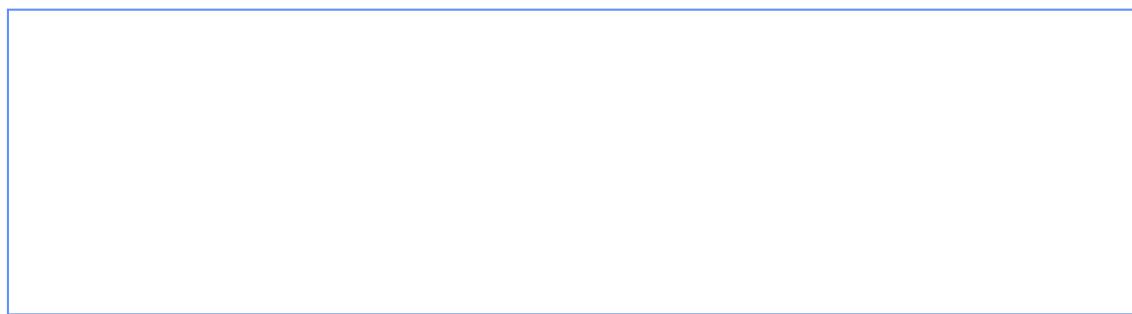
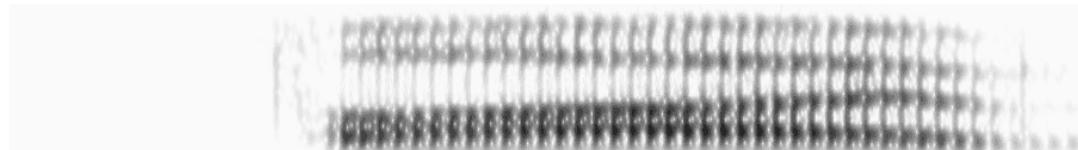
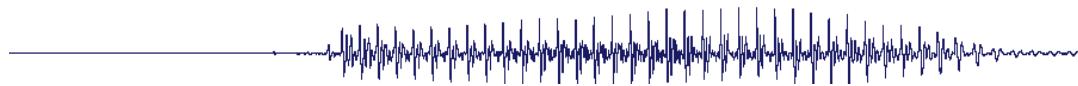
# Classification examples - cont'd

## Handwritten Digit Recognition



0 0 0 0 1 1 1 1 1 1 2  
2 2 2 2 2 2 2 3 3 3  
3 4 4 4 4 4 5 5 5 5  
6 6 6 6 7 7 7 7 8 8  
8 8 8 8 9 9 9 9 9 9

# Classification examples - cont'd



# Classification examples - cont'd



Income  $x_1$   
Savings  $x_2$   
Low-risk / high-risk y

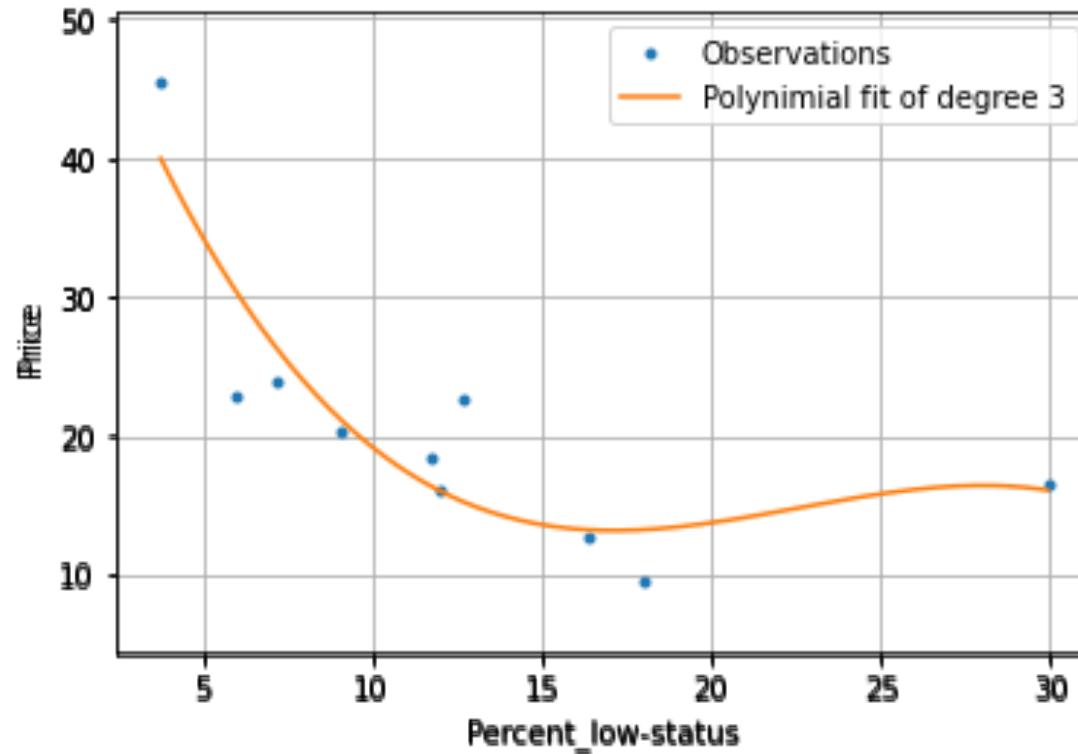


versus



# Regression example

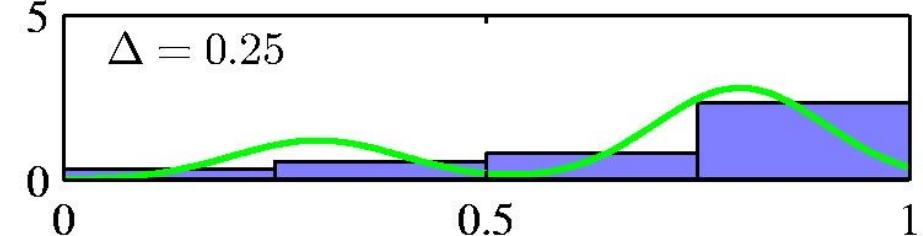
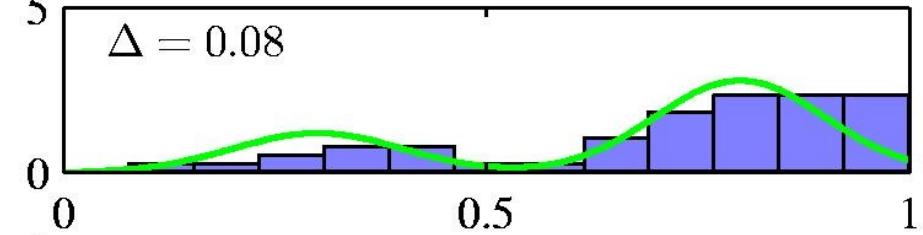
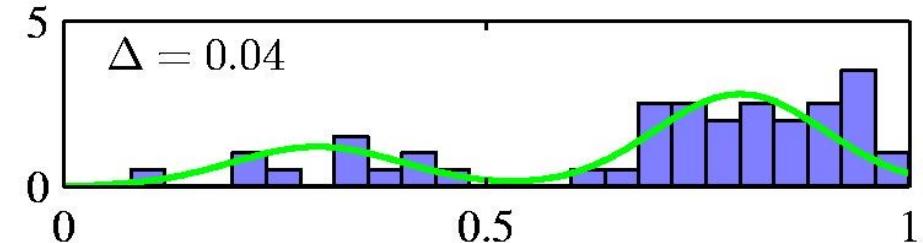
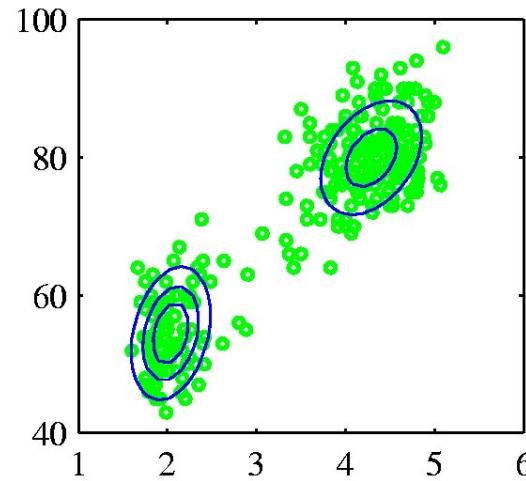
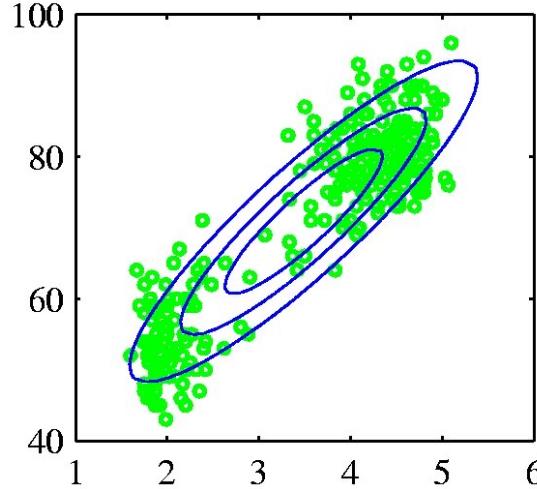
Polynomial Curve Fitting for the Boston Housing Dataset



$x$  - Percent\_low\_status  
 $t$  - Price

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$$

# Density estimation examples



from Bishop

# Lecture outline

- Introduction
  - Typical machine learning problems and selected examples
- Machine learning
  - Concepts, supervised learning, unsupervised learning
- Memory-based learning
- Model-based learning

# What is machine learning?

- Machine refers to computing devices.
- Learning is improving performance through experience.
- Machine learning is concerned with the design and development of algorithms that allow computers to optimize a performance criterion (i.e., learn) using examples or experiences.

# Machine learning's WHs

- What: Get the computer to learn **density, discriminant or regression functions** by showing examples of inputs (and outputs).
- How: We write a “parameterized” program and let the learning algorithm find the set of parameters that best approximates the desired function or behavior.

# Types of machine learning

- **Supervised learning:** given inputs along with corresponding outputs (labeled data), find the ‘correct’ outputs for test inputs
  - Classification: 1-of-N discrete output (pattern recognition)
  - Regression: real-valued output (prediction)
- **Unsupervised learning:** given only inputs without outputs (unlabeled data) as training, find structure in the space
  - density estimation
  - clustering
  - dimensionality reduction
- **Reinforcement learning:** given inputs from the environment, take actions that affect the environment and produce action sequences that maximize the expected scalar reward or punishment. This is similar to animal learning.

# Supervised learning {input, output}

- **Classification**: assign each input to one of **a finite number of discrete categories**. Learn a decision boundary that separates one class from the other.
- **Regression**: the desired output consists of one or more **continuous variables**. Learn a continuous input-output mapping from a limited number of examples. Regression is also known as “curve fitting” or “function approximation”.

# Supervised learning {input, output}

- How to represent the inputs and outputs
- How to select a both powerful and searchable hypothesis space to represent the relationship between inputs and outputs

# Unsupervised learning {input}

- Discover the unknown structure of the inputs.
  - density estimation: determine the probability density distribution of data within the input space, e.g. histogram.
  - dimensionality reduction: project the data from a high-dimensional space down to low dimensions.
  - clustering: discover groups of similar examples (clumps) within the data, e.g. k-means.

# Lecture outline

- Introduction
  - Typical machine learning problems and selected examples
- Machine learning
  - Concepts, supervised learning, unsupervised learning
- Memory-based learning
- Model-based learning

# Learning is more than memorization

- Constructing a lookup table is easy
  - Simply store all the inputs and their corresponding outputs in the training data.
  - For a new input, compare it to all the samples and produce the output associated with the matching prototype.
- Problem
  - In general, new inputs are different from training prototypes.
  - The key of learning is **generalization**: the ability to produce correct outputs or behavior on previously unseen inputs.

## Learning from examples

# Memory based learning: a simple trick

- Compute the “distances” between the input and all the stored prototypes, instead of identity requirement.
  - 1-nearest neighbor search: choose the class of the nearest prototype.
  - K-nearest neighbor search: choose the class that has the majority among the K nearest prototypes.
  - so called lazy learning, or memory-based learning.
- Challenges
  - What is the right similarity measure?
  - High computational intensity for large number of prototypes
  - The curse of dimensionality and data sparsity.

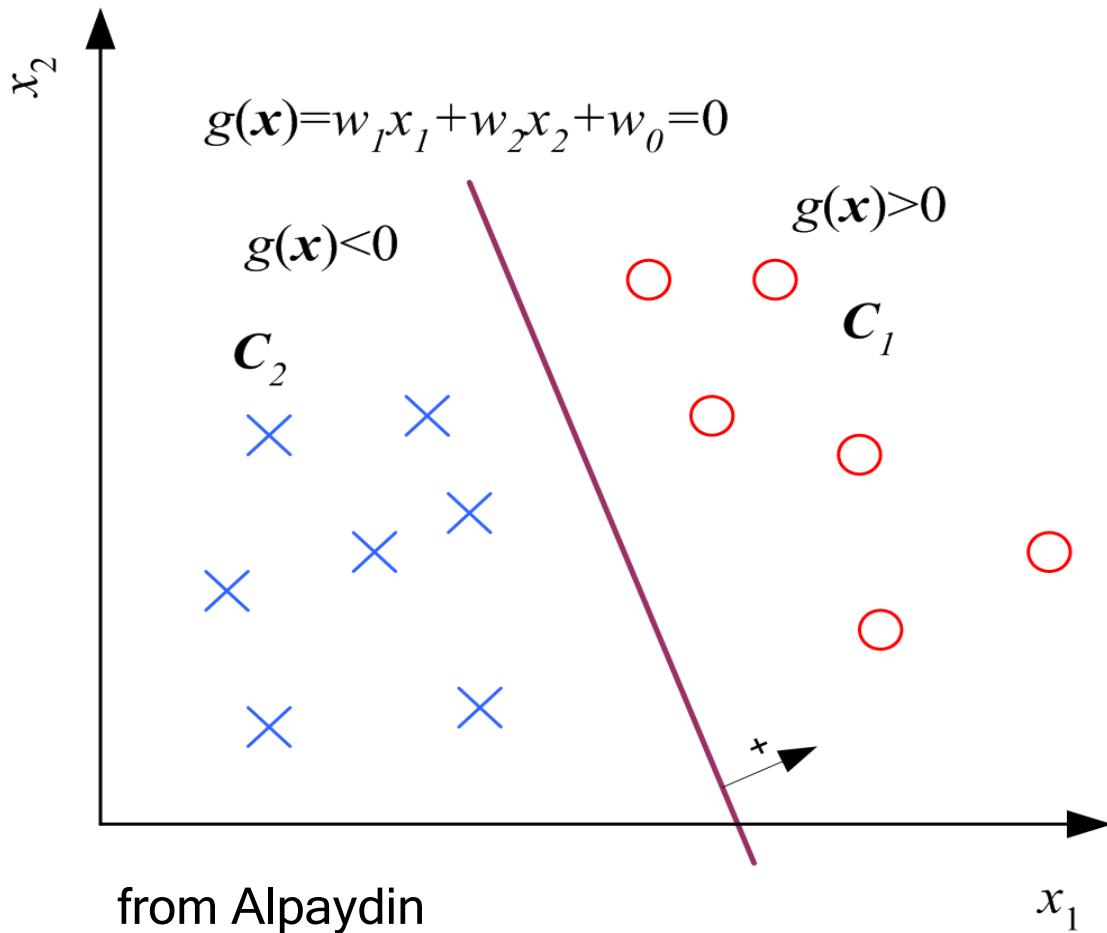
# Lecture outline

- Introduction
  - Typical machine learning problems and selected examples
- Machine learning
  - Concepts, supervised learning, unsupervised learning
- Memory-based learning
- Model-based learning
  - Over-fitting, bias-variance trade-off
  - Function view of machine learning

# Model based learning

- Build a **model** that is a good approximation to the data, or construct a general, explicit description of the target function
  - linear vs nonlinear
  - parametric vs nonparametric
  - shallow vs deep
- Discard learning examples when they are processed
  - efficient computationally and in memory use.
- It is limited by the used **learning bias (also known as inductive bias)** - a coarse approximation of the target function.

# Linear classifier - two classes

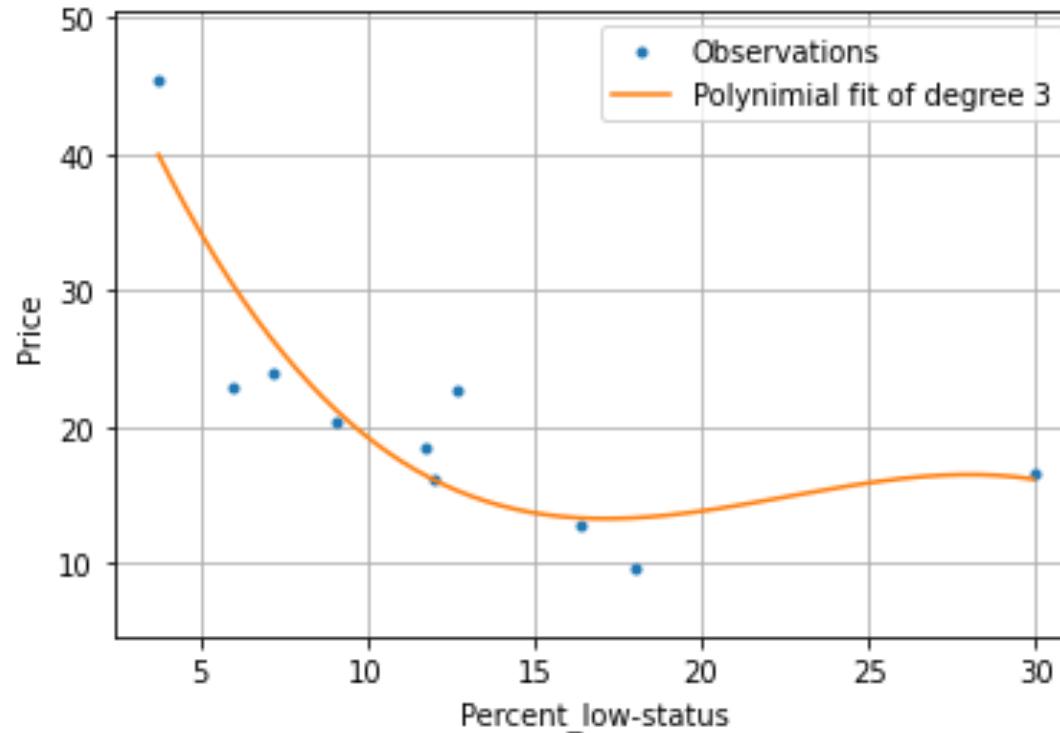


$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

choose  $\begin{cases} C_1 & \text{if } g(\mathbf{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$

# Regression example - again

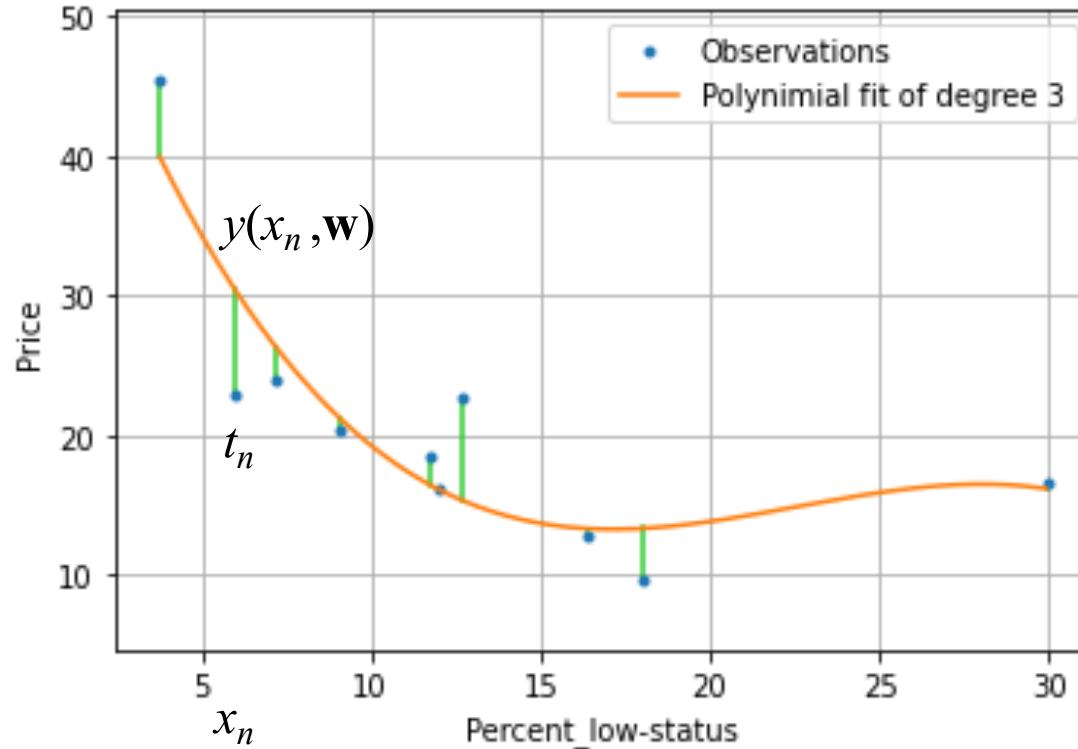
Polynomial curve fitting for the Boston Housing Dataset



$x$  - Percent\_low\_status  
 $t$  - Price

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$$

# Sum-of-squares error function

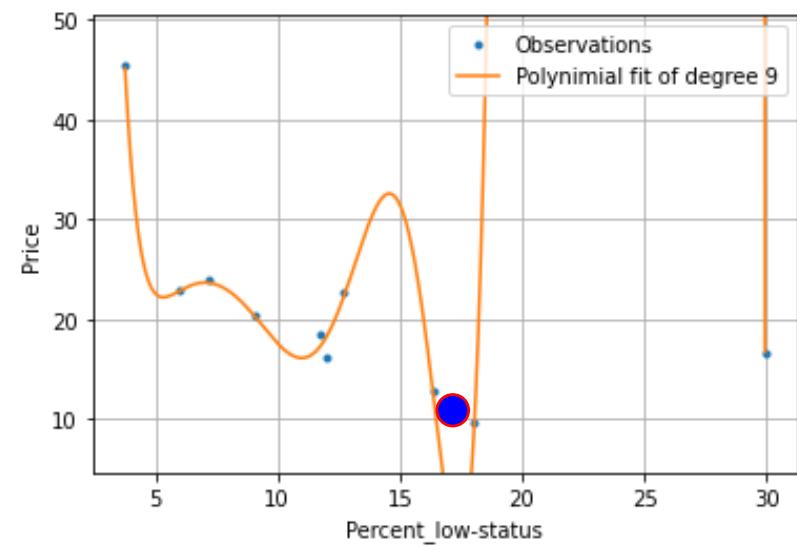
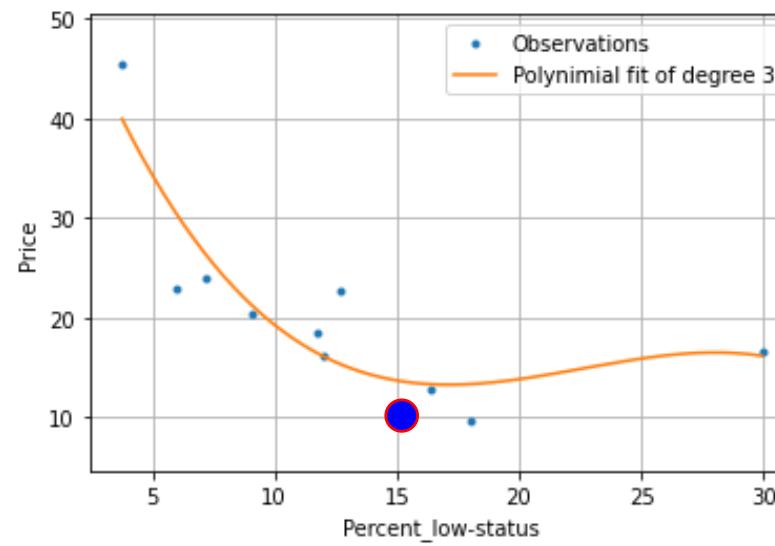
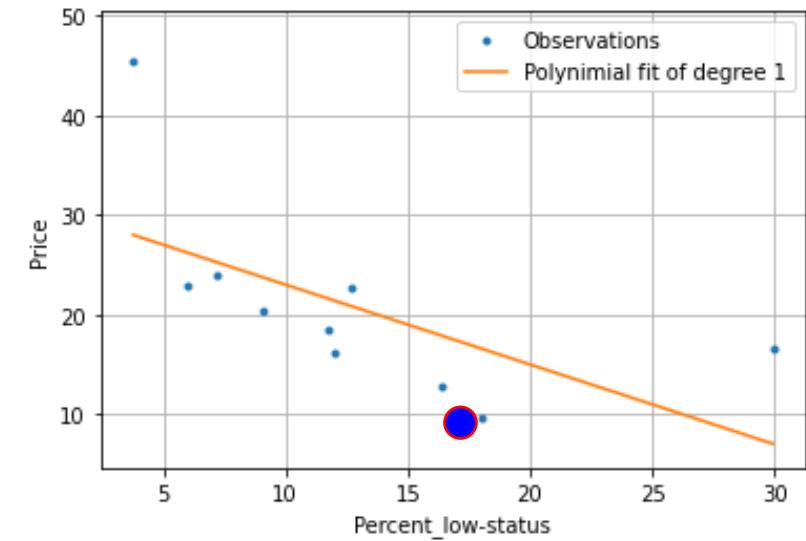
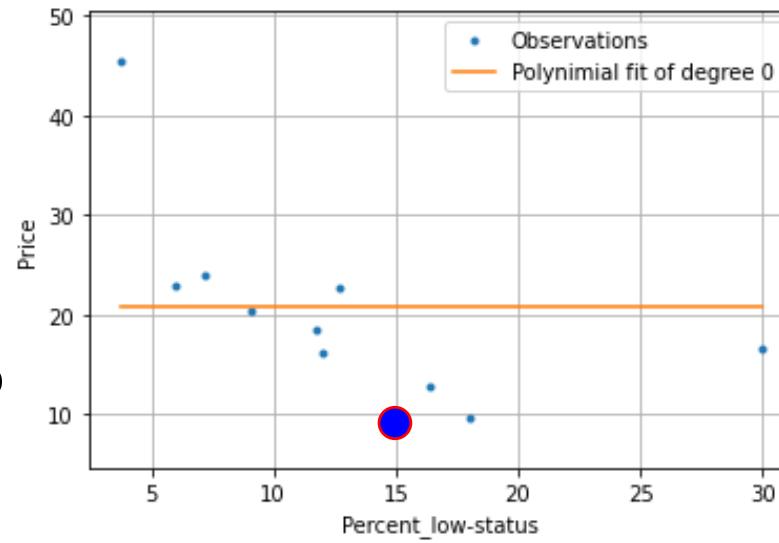


$x$  - Percent\_low\_status  
 $t$  - Price

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

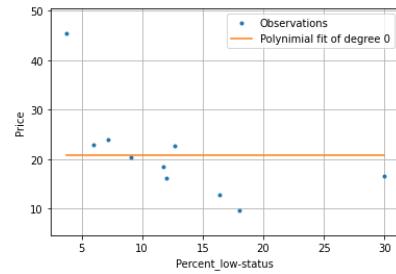
# Polynomials - model selection

Underfit to the training data (room to improve)

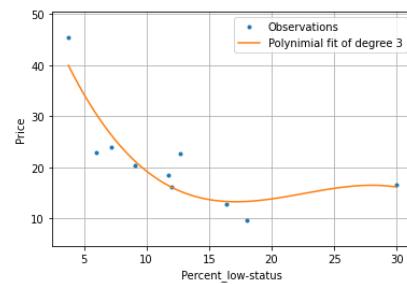


# Polynomials - coeffs, MSE

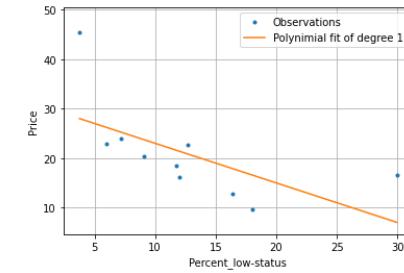
$M = 0$   
 $(\text{array}([20.9]),$   
 $86.0)$



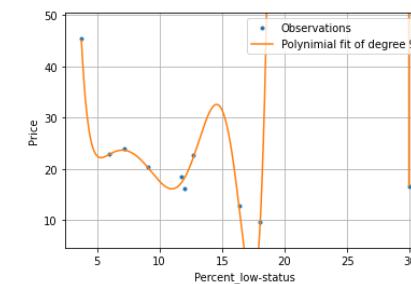
$M = 3$   
 $(\text{array}([ 6.26e+01, -7.23e+00,$   
 $3.40e-01, -5.02e-03]),$   
 $16.5)$



$M = 1$   
 $(\text{array}([31.0, -0.801]),$   
 $53.3)$



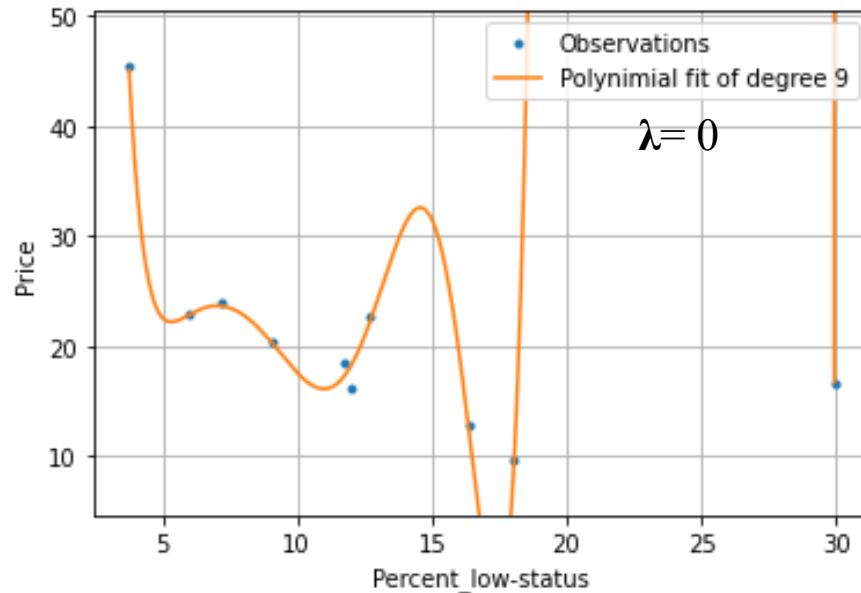
$M = 9$   
 $(\text{array}([ 1.10e+05, -1.10e+05, 4.62e+04, -$   
 $1.09e+04, 1.58e+03, -1.46e+02, 8.69e+00,$   
 $-3.19e-01, 6.54e-03, -5.71e-05]),$   
 $5.18e-14)$



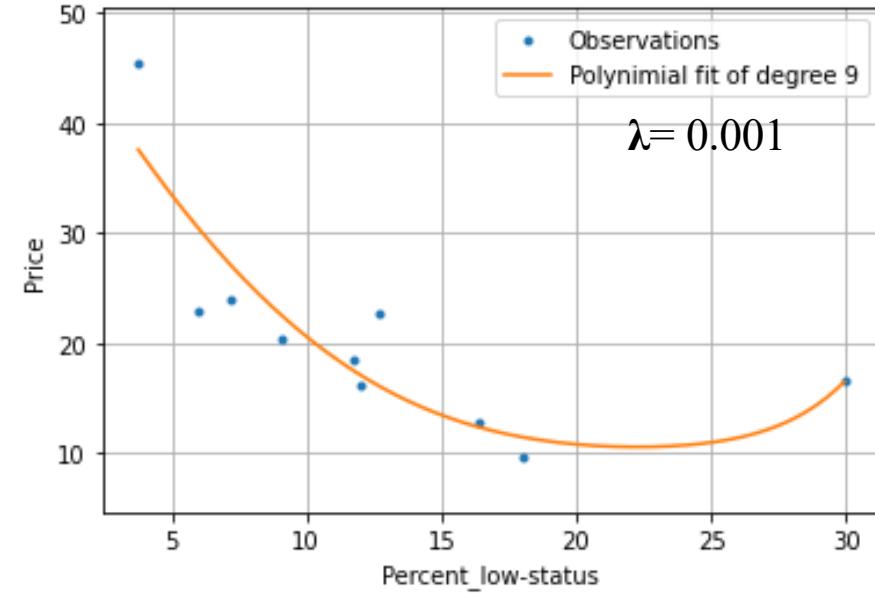
# Ridge regression / L2-regularization

- When using L2-regularization the error function to be minimized becomes (the addition of the l2-term)

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$



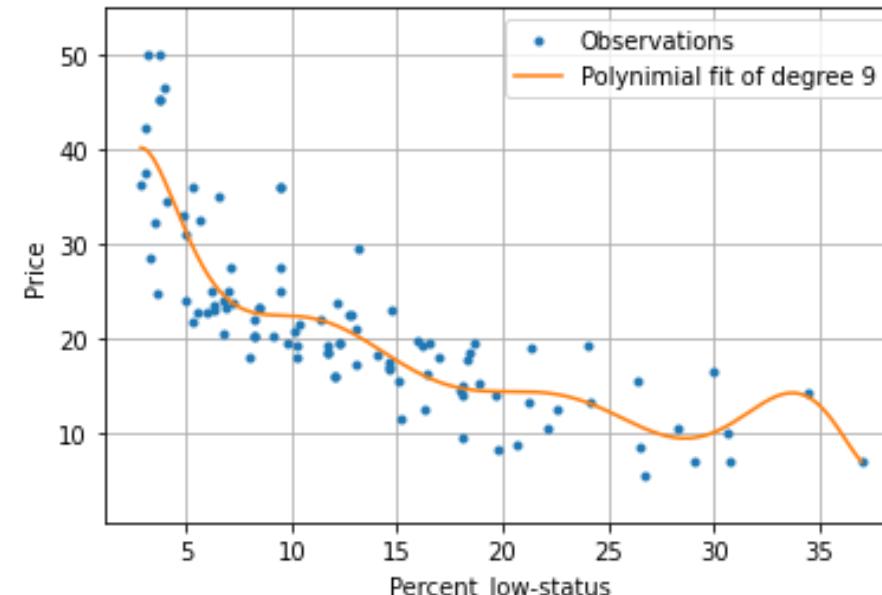
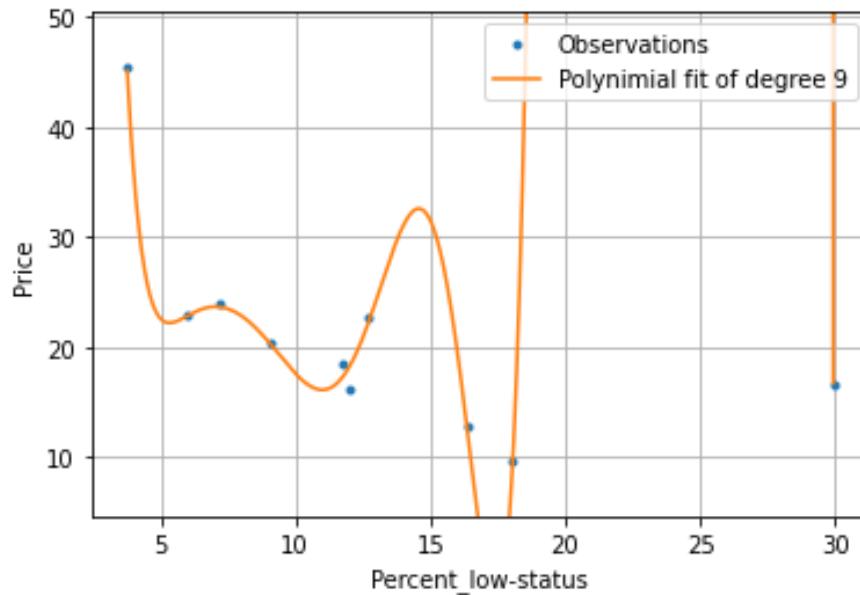
Some coeffs > 1.10e+5  
MSE = 5.18e-14



All coeffs < 60  
MSE = 18.1

# Data set size

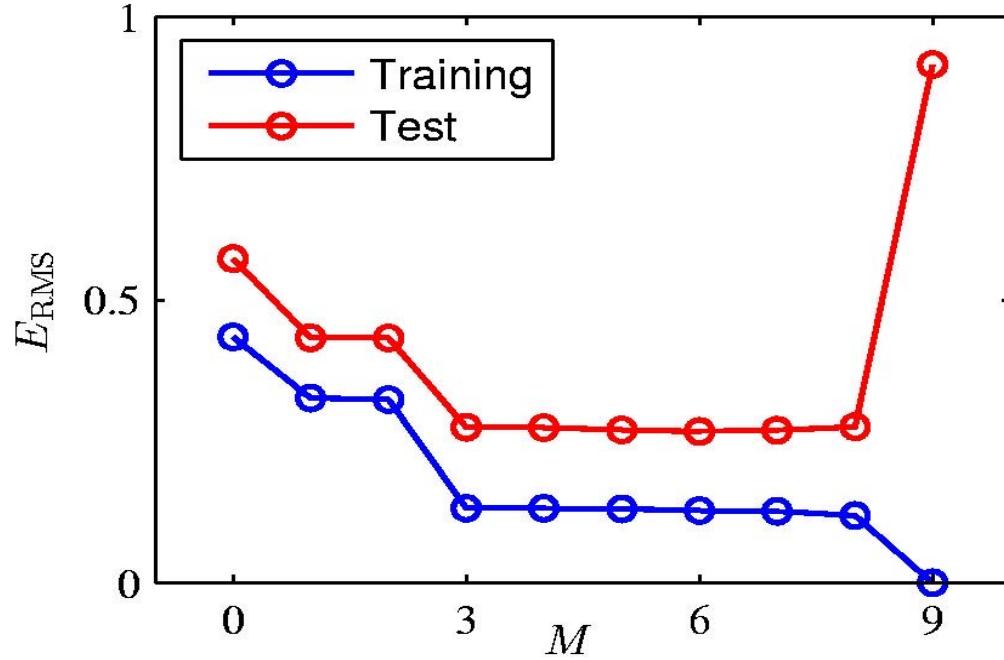
## 9<sup>th</sup> Order Polynomial



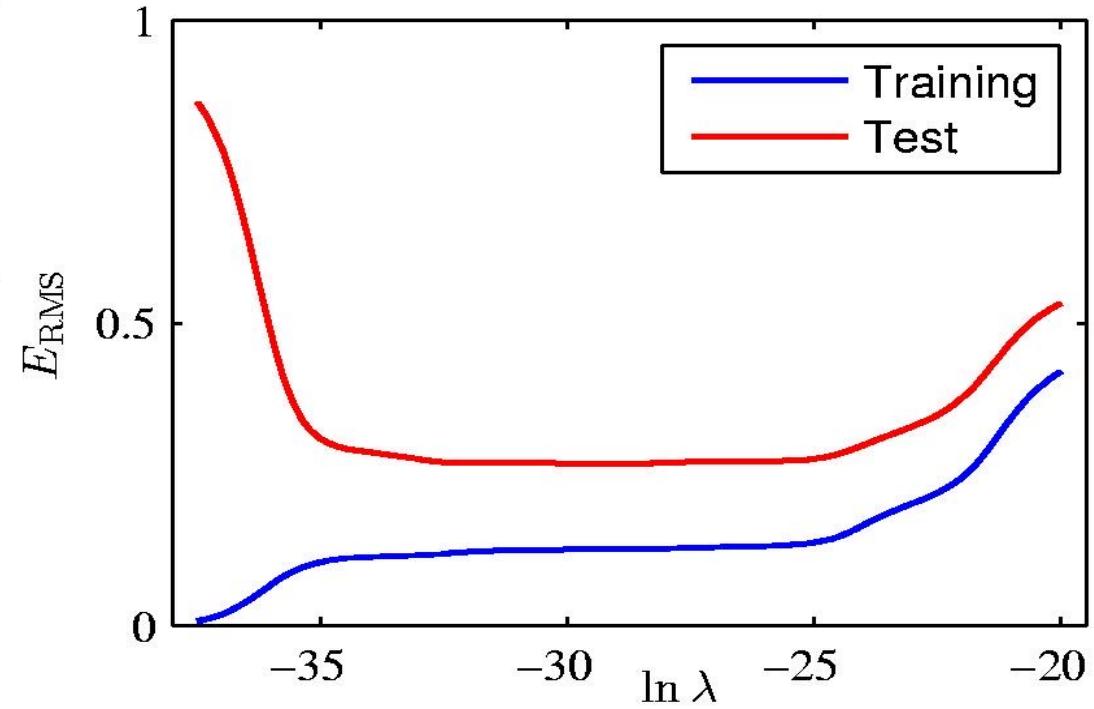
For a given model complexity, the over-fitting problem become less severe as the size of the data set increases.

# Over-fitting and regularization

from Bishop

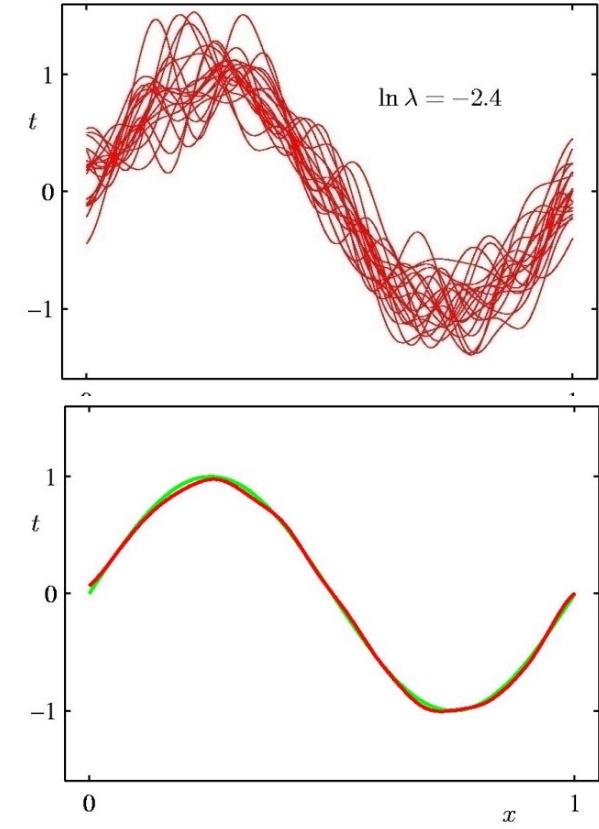
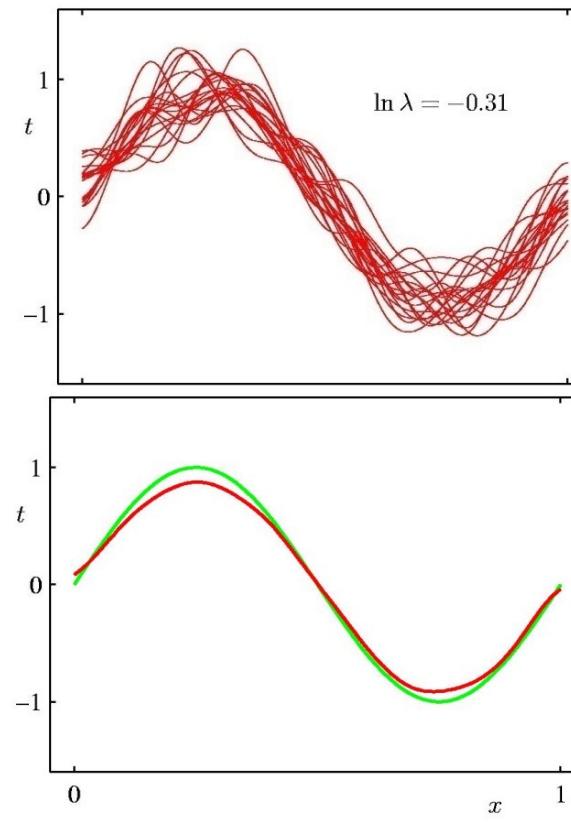
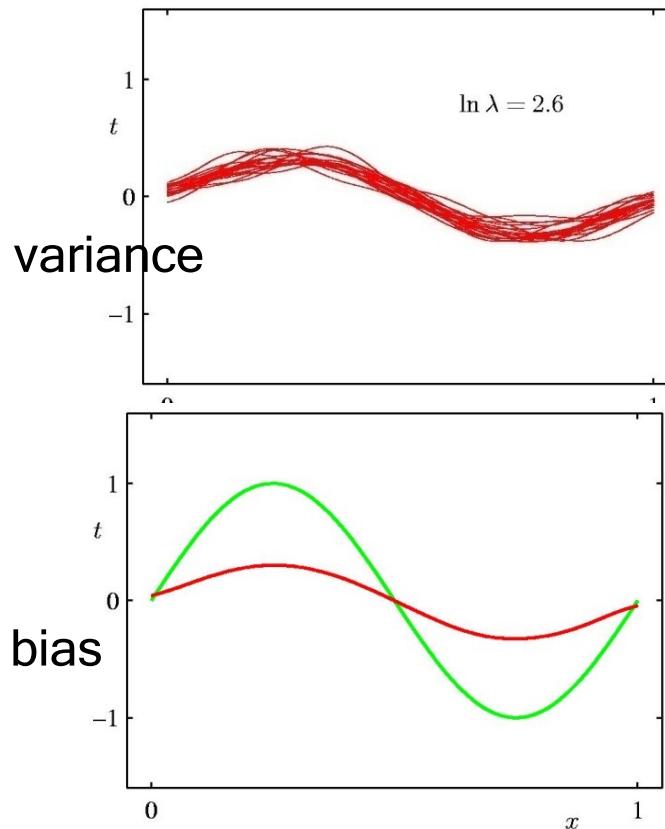


The need for a separate validation (or hold-out) set for model selection.



# Number of data sets for 9<sup>th</sup> order

- 100 data sets; training multiple polynomials and then averaging them, the contribution from the variance term tended to cancel, leading to improved predictions.
- **The dependence of bias and variance on model complexity**

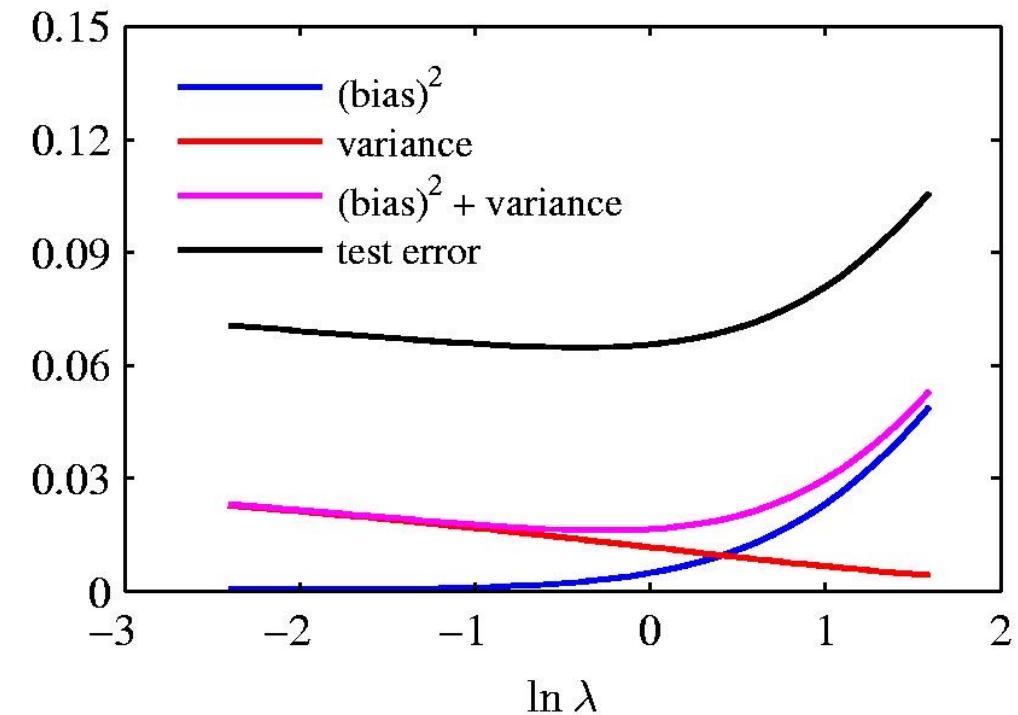


# Bias-variance trade-off

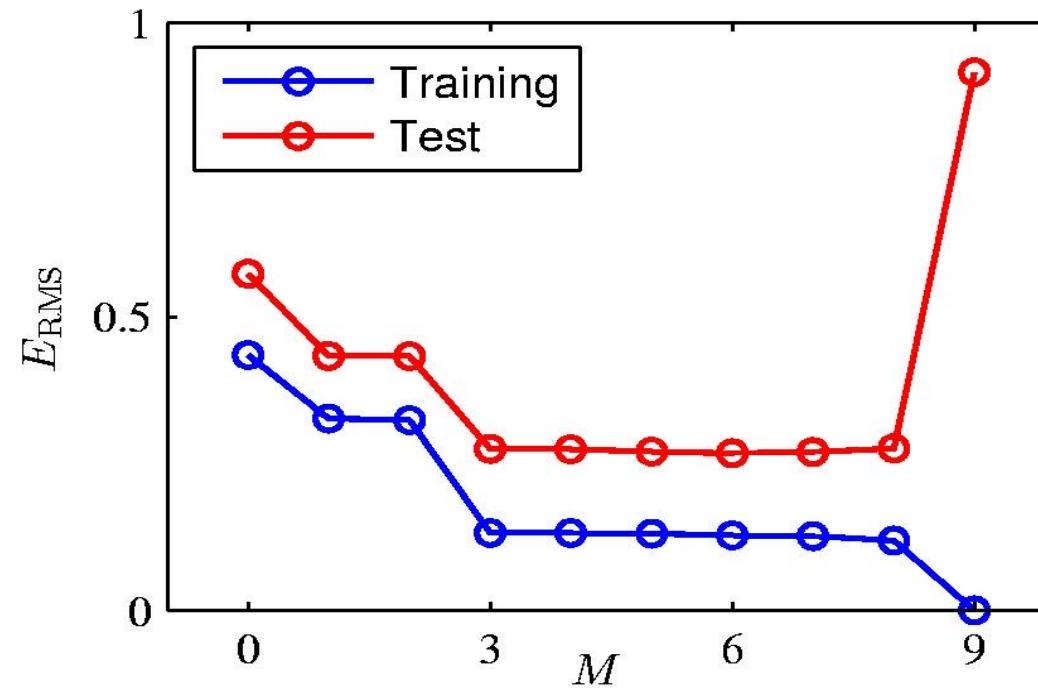
- There is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance.

Mean square error of the estimator  $d$  for unknown parameter  $\theta$ :

$$\begin{aligned}
 r(d, \theta) &= E[(d - \theta)^2] \\
 &= (E[d] - \theta)^2 + E[(d - E[d])^2] \\
 &= \text{Bias}^2 + \text{Variance}
 \end{aligned}$$



# Over-fitting - a still unsolved problem!



from Bishop

The least squares approach to finding the model parameters resorts to intuition and represents a specific case of **maximum likelihood**, and that the over-fitting problem can be understood as a general property of ML. → More principled approach - **probability theory**, foundation for machine learning.

# Function view of machine learning

- Known examples of function:
  - Circumference of a circle is a function of radius (also diameter):

$$A=2\pi r$$

- Instantaneous velocity  $v$  of a falling object after elapsed time  $t$  is

$$v=gt$$

where  $g$  is Gravitational acceleration. Near the surface of the Earth, the acceleration due to gravity is  $g = 9.807 \text{ m/s}^2$

- In essence, a machine learning problem is a mathematical modelling problem, i.e., coming up with a mathematic function to solve a problem, e.g., predicting a label or a value or calculating a likelihood, given a data input.

# Function view of machine learning - cont'd

1. Given an input  $x$ , we define a parametric function  $f_{\theta}(x)$  where  $\theta$  is the set of parameters of  $f$ , to predict the output  $y$  for a given input  $x$ .
2. To learn  $\theta$ , we define an objective/loss function
  - Regression, (continuous numbers, e.g., mean squared error),
  - Classification (discrete categories, e.g., cross-entropy)
  - Density (continuous numbers, e.g., likelihood)
3. Training: for given training data, an optimizer minimizes the loss to find  $\theta$ , e.g., stochastic gradient descent (SGD)
4. Inference: apply the learned function to unknown data

Design for uncertainty - a probabilistic perspective!

# Statistical machine learning approach

- Steps for statistical machine learning after problem formulation:
  1. Data Collection (cleaning and labelling):
    - Large sample of data; often costly and time-consuming
  2. Model Selection
    - Bias-variance tradeoff, regularization
  3. Parameter Estimation (training)
    - Calculate parameter values using the data
  4. Search based on the learned model (evaluation)
    - Find optimal solution to the given problem

# An example

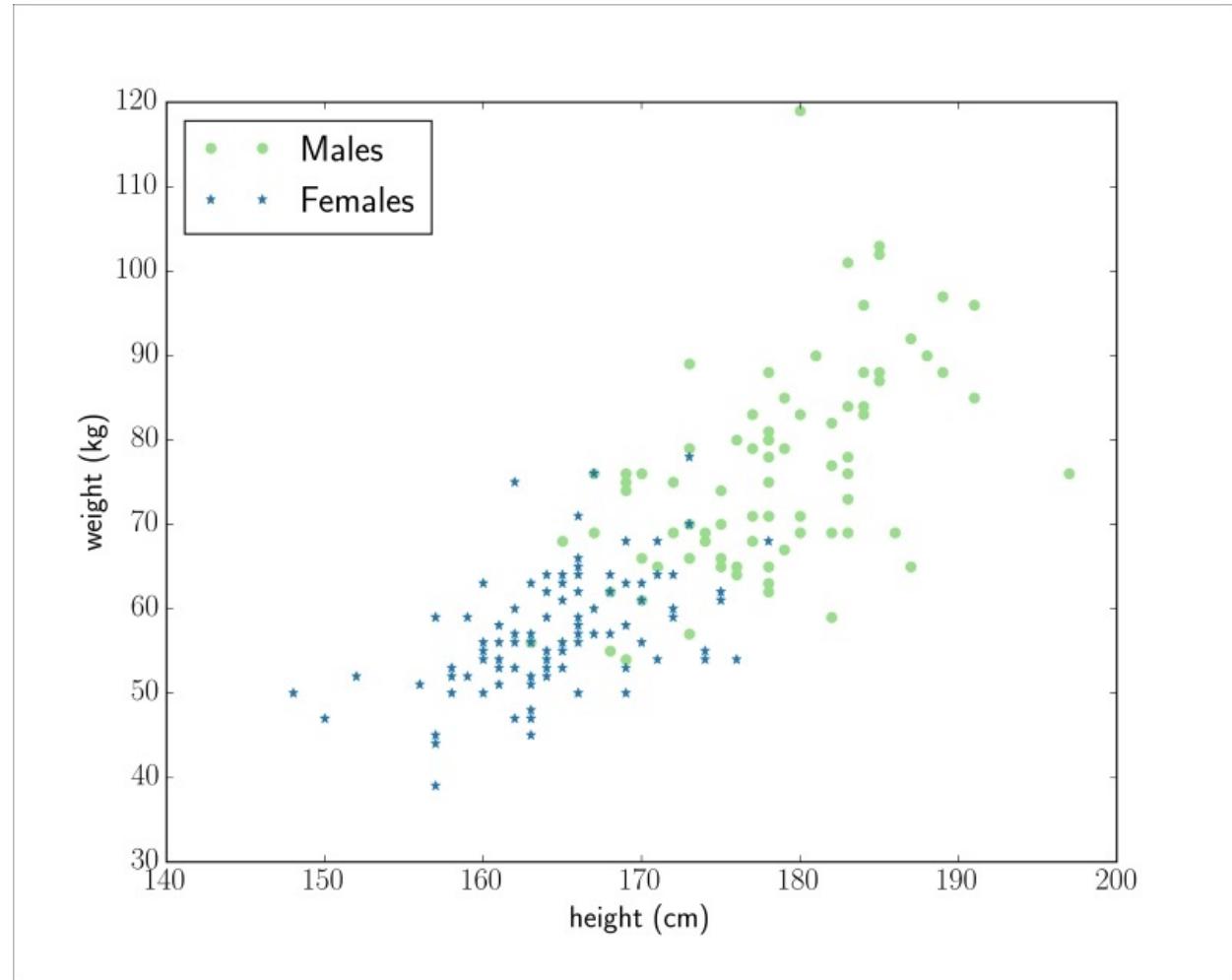


Figure from (Scala for Data Science by P Bugnion)

- Regression:  $(x_n, y_n)$
- Classification:  $([x_{1,n}, x_{2,n}]^T, y_n)$
- Apply machine learning
  - Function or model
  - Loss function
  - Optimization
  - Inference
- Similarity and differences btw regression and classification
- Density estimation's role in classification

# Lecture outline

- Introduction
  - Typical machine learning problems and selected examples
- Machine learning
  - Concepts, supervised learning, unsupervised learning
- Memory-based learning
- Model-based learning
  - Over-fitting, bias-variance trade-off
  - Function view of machine learning

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 2: Bayesian Decision Theory

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Lecture outline

- Probability theory
- Bayes' theorem
- Classification

# Probability theory

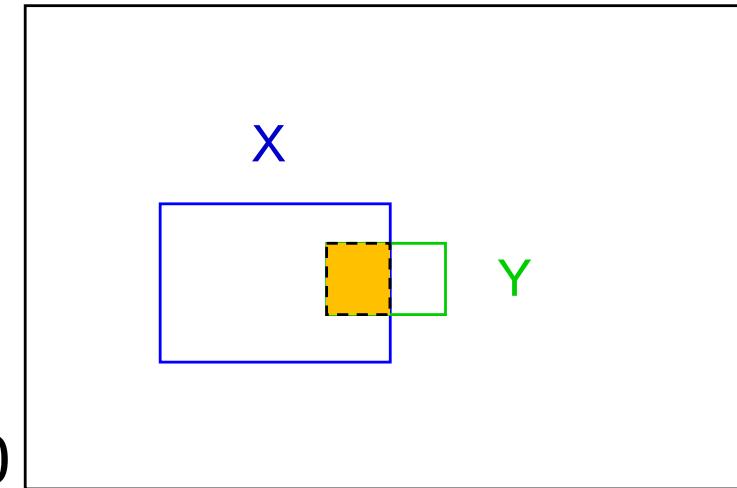
- A key concept in pattern recognition is that of **uncertainty**, which arises through both
  - noise on measurements
  - the finite size of data sets.
- **Probability theory** provides a consistent framework for the quantization and manipulation of uncertainty.

# Random variables

- Random variable, used in the study of probability (wiki)
  - a function mapping the sample space of a random process to the real numbers
  - to which a probability distribution is assigned.
- Discrete random variables
  - $X$  is a Boolean-valued random variable if  $X$  denotes an event, and there is some degree of uncertainty as to whether  $X$  occurs.
  - Examples
- $X = \text{You wake up tomorrow with a gift delivered from a postman.}$
- Probabilities
  - Denote  $p(X)$  as “the fraction of possible worlds in which  $X$  is true”
  - $P(X = x_i) = p_i$ .

# Probability theory

- $p(X|Y)$ =Fraction of worlds in which Y is true that also have X true.
- $X$  = “Receive a gift”;  $Y$  = “Get a prize”
- $p(X) = 1/10$ ;  $p(Y) = 1/40$ ;  $p(X|Y) = 1/2$
- $p(X|Y) = p(X, Y)/P(Y)$   
 $= \text{area of "X and Y"}/\text{area of "Y"}$
- **Product rule:**  $p(X, Y)=p(X|Y) p(Y) = 1/80$
- $p(X|Y) = \text{area of "Y and X"}/\text{area of "Y"}$   
 $= p(Y, X)/p(Y)$   
 $= p(Y|X)p(X)/p(Y)$



**Sum rule:**

$$p(X) = \sum_Y p(X, Y)$$

(e.g., both true and false for Y)

which is the **Bayes' theorem**.

# Probability theory - general example

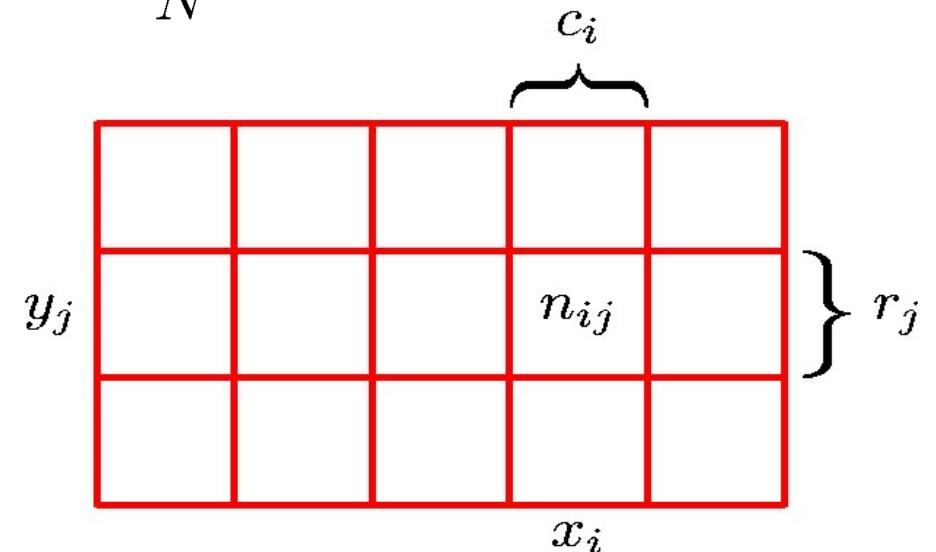
- Marginal Probability  $p(X = x_i) = \frac{c_i}{N}.$
- Conditional Probability  $p(Y = y_j | X = x_i) = \frac{n_{ij}}{c_i}$
- Joint Probability  $p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$
- **Sum Rule**

$$p(X = x_i) = \frac{c_i}{N} = \frac{1}{N} \sum_{j=1}^L n_{ij}$$

$$= \sum_{j=1}^L p(X = x_i, Y = y_j)$$

- **Product rule**

$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j | X = x_i)p(X = x_i) \end{aligned}$$



# The rules of probability

- **The rules of probability**

- sum rule

$$p(X) = \sum_Y p(X, Y)$$

- product rule

$$p(X, Y) = p(X | Y)p(Y)$$

- Based on the product rule and the symmetry property  $p(X, Y) = p(Y, X)$ , we have **Bayes' theorem**

$$p(X|Y) = p(Y|X) p(X) / p(Y)$$

which plays a central role in machine learning and pattern recognition.

# Bayes' theorem

**Posterior probability**  
of  $C_i$  given data  $\mathbf{x}$

$$p(C_i | \mathbf{x}) = \frac{p(\mathbf{x} | C_i) p(C_i)}{p(\mathbf{x})}$$

/ OR likelihood function of  $C_i$ 
— Prior probability of  $C_i$ 
Evidence  $\sum_k p(C_k)p(\mathbf{x} | C_k)$

- $P(\mathbf{x}|C_i)$  is the probability of seeing  $\mathbf{x}$  as the input when it is known to belong to class  $C_i$ ,  $i=1,\dots,K$ .
- $p(\mathbf{x})$  is the normalization constant required to ensure that the sum of the conditional probability  $p(C_i|\mathbf{x})$  over all values of  $\mathbf{x}$  is equal to one.
- Bayes' theorem converts a prior probability into a posterior probability by incorporating the observed data.

# Lecture outline

- Probability theory
- Bayes' theorem
- Classification

# Bayes' theorem for classification

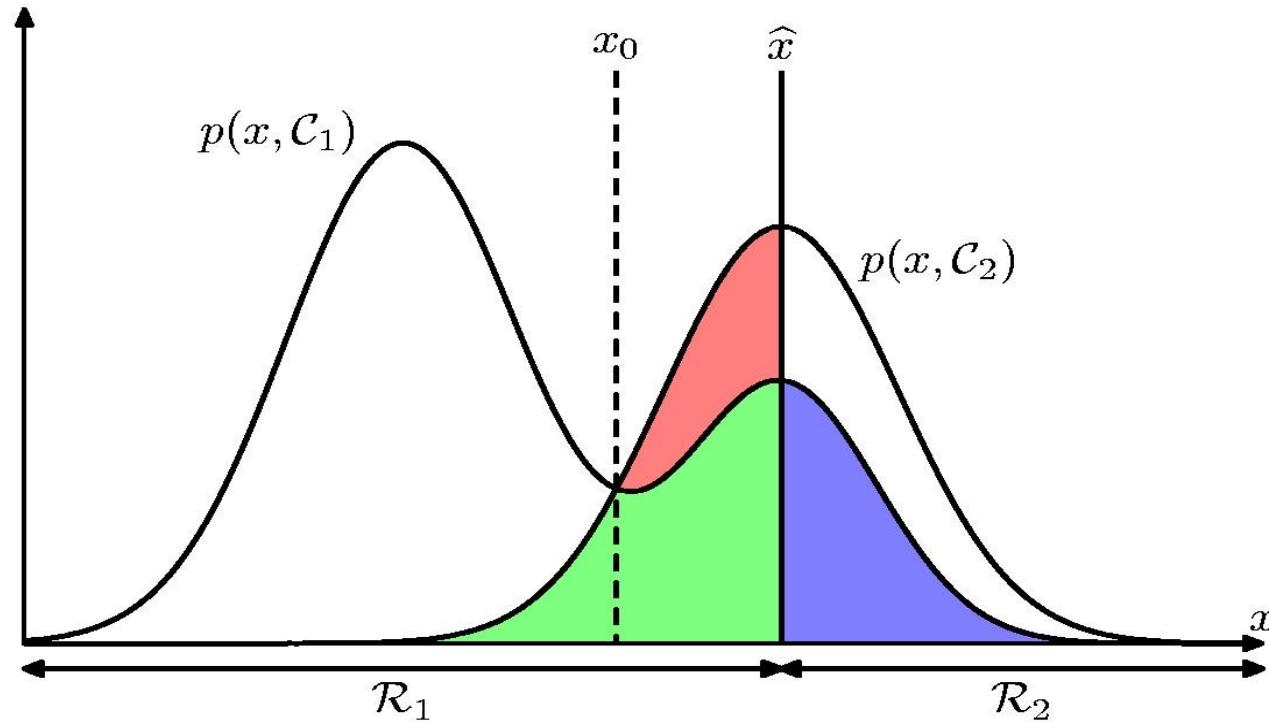
- Combining the prior and what the data tells us, we calculate the posterior probability:

$$p(C_i | \mathbf{x}) = \frac{p(C_i)p(\mathbf{x} | C_i)}{p(\mathbf{x})} = \frac{p(C_i)p(\mathbf{x} | C_i)}{\sum_{k=1}^K p(C_k)p(\mathbf{x} | C_k)}$$

- For minimum error, the Bayes' classifier chooses the class with the highest posterior probability, i.e.,

choose  $C_i$  if  $p(C_i | \mathbf{x}) = \max_k p(C_k | \mathbf{x})$

# Minimum misclassification rate



$$\begin{aligned}
 p(\text{mistake}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) \\
 &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) d\mathbf{x}.
 \end{aligned}$$

# Classification example

- Credit scoring: Inputs are income  $x_1$  and savings  $x_2$ .  
Output is low-risk ( $C=0$ ) vs high-risk ( $C=1$ )
- When we get a new input  $\mathbf{x} = [x_1, x_2]^T$ , find out output  $C \in \{0,1\}$
- Prediction:

$$\text{choose } \begin{cases} C = 1 \text{ if } P(C = 1 | x_1, x_2) > 0.5 \\ C = 0 \text{ otherwise} \end{cases}$$

or equivalently

$$\text{choose } \begin{cases} C = 1 \text{ if } P(C = 1 | x_1, x_2) > P(C = 0 | x_1, x_2) \\ C = 0 \text{ otherwise} \end{cases}$$

# Losses and risks

- Decisions may not be equally costly.
  - Actions  $a_i$  is the decision to assign the input the class  $C_i$ .
  - Loss  $\lambda_{ik}$  is the loss for taking action  $a_i$  when the input actually belongs to  $C_k$ .
- Expected risk
$$R(\alpha_i | \mathbf{x}) = \sum_{k=1}^K \lambda_{ik} P(C_k | \mathbf{x})$$
- Choose the action with minimum risk
  - choose  $\alpha_i$  if  $R(\alpha_i | \mathbf{x}) = \min_k R(\alpha_k | \mathbf{x})$
- **Decision theory:** make a decision to minimize cost.

# Losses and risks: 0/1 loss

Loss  $\lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ 1 & \text{if } i \neq k \end{cases}$

Risk 
$$\begin{aligned} R(\alpha_i | \mathbf{x}) &= \sum_{k=1}^K \lambda_{ik} P(C_k | \mathbf{x}) \\ &= \sum_{k \neq i} P(C_k | \mathbf{x}) \\ &= 1 - P(C_i | \mathbf{x}) \end{aligned}$$

To minimize risk, choose the most probable class.

# Losses and risks: reject

When wrong decisions have very high cost, e.g. in postal codes reading, we define an additional action of reject,  $\alpha_{K+1}$

$$\text{Loss } \lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ \lambda & \text{if } i = K + 1, \quad 0 < \lambda < 1 \\ 1 & \text{otherwise} \end{cases}$$

Risk of reject

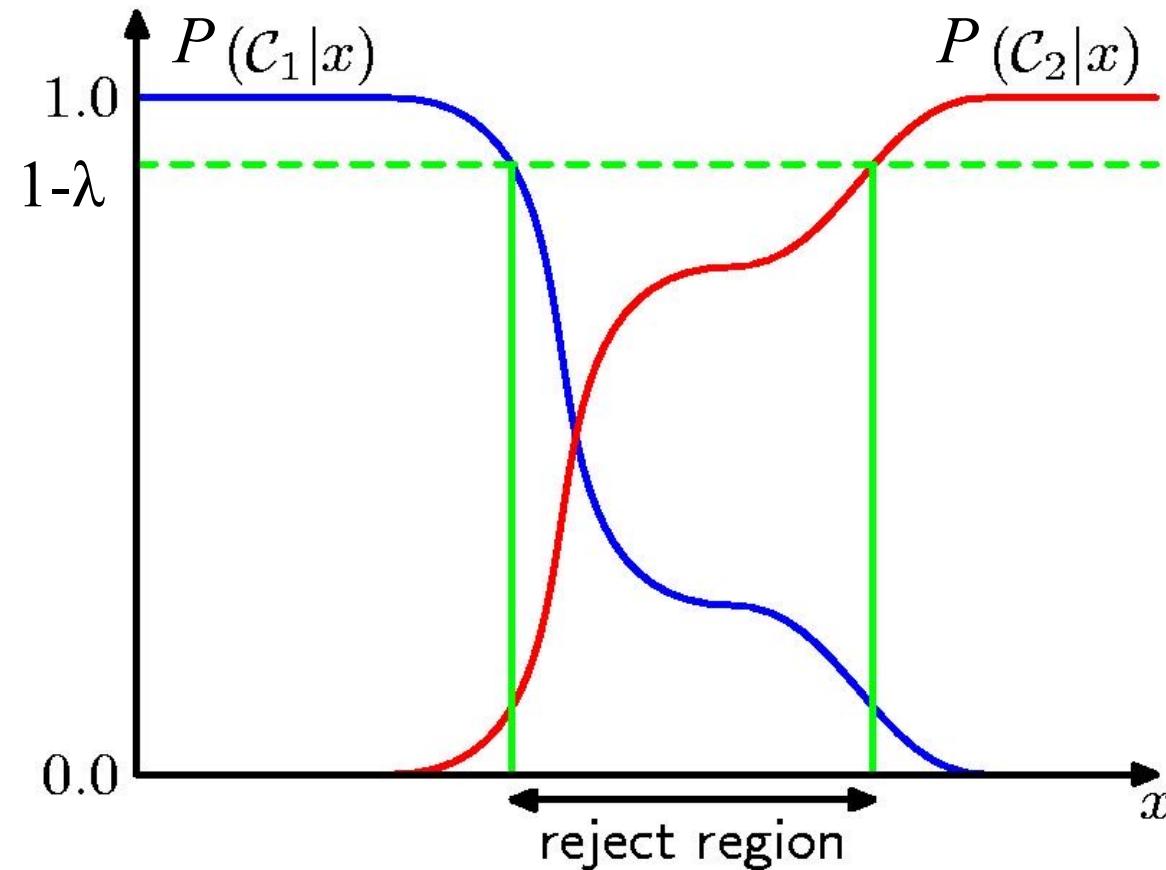
$$R(\alpha_{K+1} | \mathbf{x}) = \sum_{k=1}^K \lambda P(C_k | \mathbf{x}) = \lambda$$

Risk of choosing  $C_i$

$$R(\alpha_i | \mathbf{x}) = \sum_{k \neq i} P(C_k | \mathbf{x}) = 1 - P(C_i | \mathbf{x})$$

$$\begin{cases} \text{choose } C_i & \text{if } P(C_i | \mathbf{x}) > P(C_k | \mathbf{x}) \ \forall k \neq i \text{ and } P(C_i | \mathbf{x}) > 1 - \lambda \\ \text{reject} & \text{otherwise} \end{cases}$$

# Reject option



# Lecture outline

- Probability theory
- Bayes' theorem
- Classification

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. **Parametric and nonparametric methods**
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 3: Parametric and nonparametric methods

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. **Parametric and nonparametric methods**
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Unsupervised learning

Goals - discover the unknown structure of the inputs

- **Model data density** (Lecture 3)
  - Use: Facilitate other learning
- Reduce dimensionality (Lecture 4)
  - Use: Facilitate other learning, compress data
- Find clusters (Lecture 5)
  - Use: Compress data, detect outliers

# Statistical machine learning approach

Steps:

1. Data Collection (cleaning and labeling):
  - Large sample of data; often costly and time-consuming
2. Model Selection
  - Bias-variance tradeoff, regularization
3. Parameter Estimation (training)
  - Calculate parameter values by using the data
4. Search based on the learned model (evaluation)
  - Find optimal solution to the given problem

# Density Estimation

**Density estimation:** model the probability distribution  $p(x)$  of a random variable  $x$ , given a finite set of observations.

- Model selection is a central issue.
- **Parametric:** Assume a single model for  $p(x | C_i)$ , e.g., Gaussian density.
- **Semiparametric:**  $p(x | C_i)$  is a mixture of densities, e.g., Gaussian mixture model (GMM).
  - E.g., different phonemes in speech.
  - Being a clustering problem as well.
- **Nonparametric:** No model; data speaks for itself, e.g., histogram estimator.

# Lecture outline

- Parametric methods
  - Parameter estimation
  - Application: Parametric classification
  - Application: Parametric regression
- Nonparametric methods

# Probability density estimation

- How to estimate probabilities from a given training set?
- Parametric methods for density estimation
  - Model with a small number of parameters

# Parametric estimation

- Assume an independent and identically distributed (iid) sample  $\mathcal{X} = \{x^t\}_{t=1}^N$  where  $x^t \sim p(x)$
- Parametric estimation:
  - Assume a form for  $p(x | \theta)$  and estimate  $\theta$ , its sufficient statistics, using  $\mathcal{X}$
  - e.g.,  $\mathcal{N}(\mu, \sigma^2)$  where  $\theta = \{\mu, \sigma^2\}$

# Maximum likelihood estimation

- Likelihood of  $\theta$  given the iid sample  $\mathcal{X}$

$$l(\theta | \mathcal{X}) = p(\mathcal{X} | \theta) = \prod_t p(x^t | \theta)$$

- Log likelihood

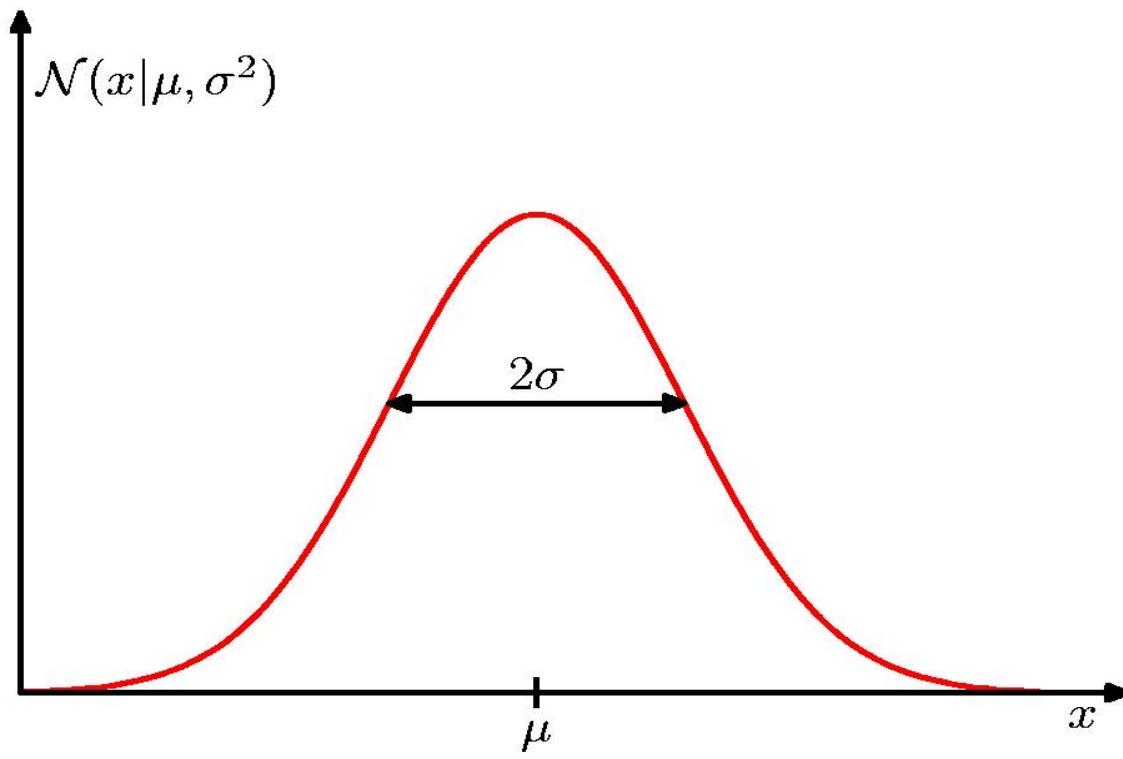
$$\mathcal{L}(\theta | \mathcal{X}) = \ln l(\theta | \mathcal{X}) = \sum_t \ln p(x^t | \theta)$$

- Maximum likelihood estimator (MLE)

$$\theta^* = \operatorname{argmax}_\theta \mathcal{L}(\theta | \mathcal{X})$$

# An example: Gaussian distribution

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x - \mu)^2\right\}$$



$$\mathcal{N}(x|\mu, \sigma^2) > 0$$

$$\int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) dx = 1$$

**Mean**

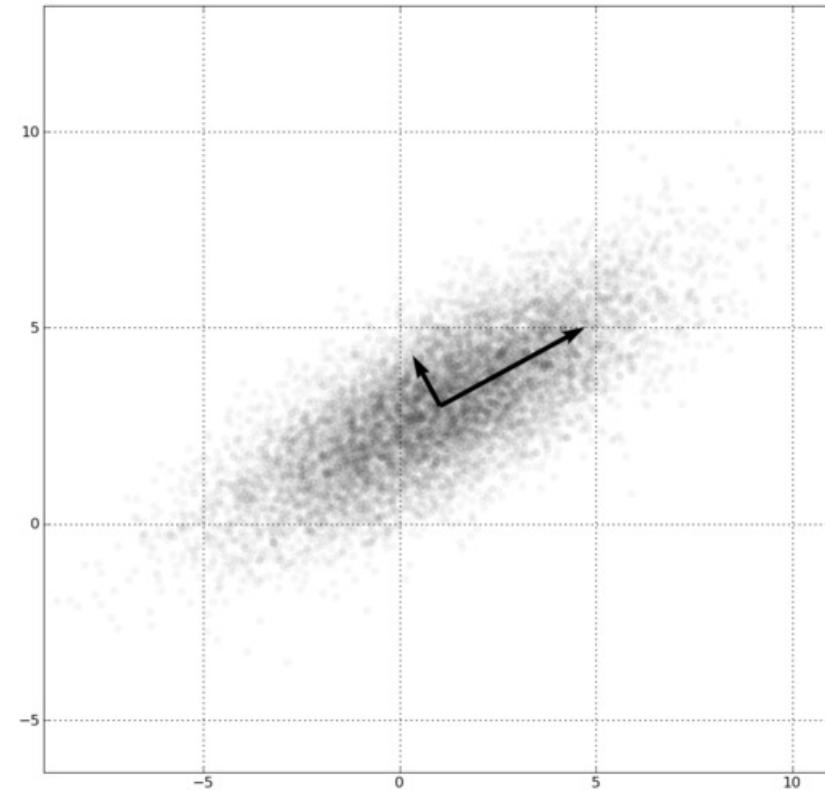
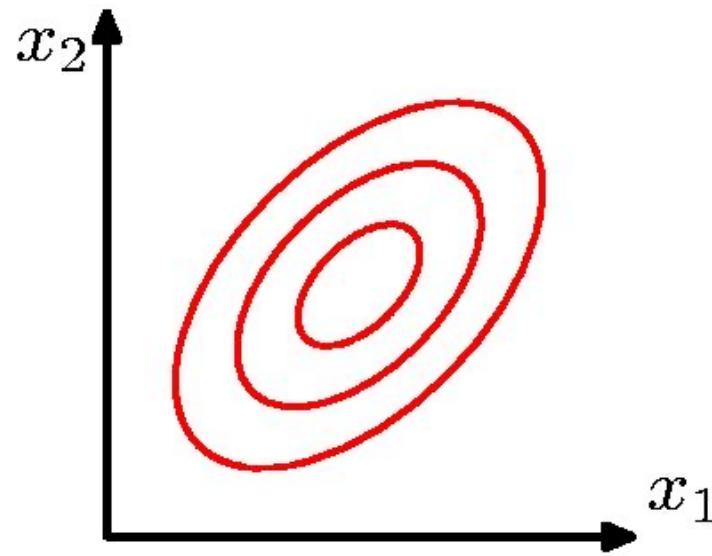
$$\mathbb{E}[x] = \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x dx = \mu$$

**Variance**

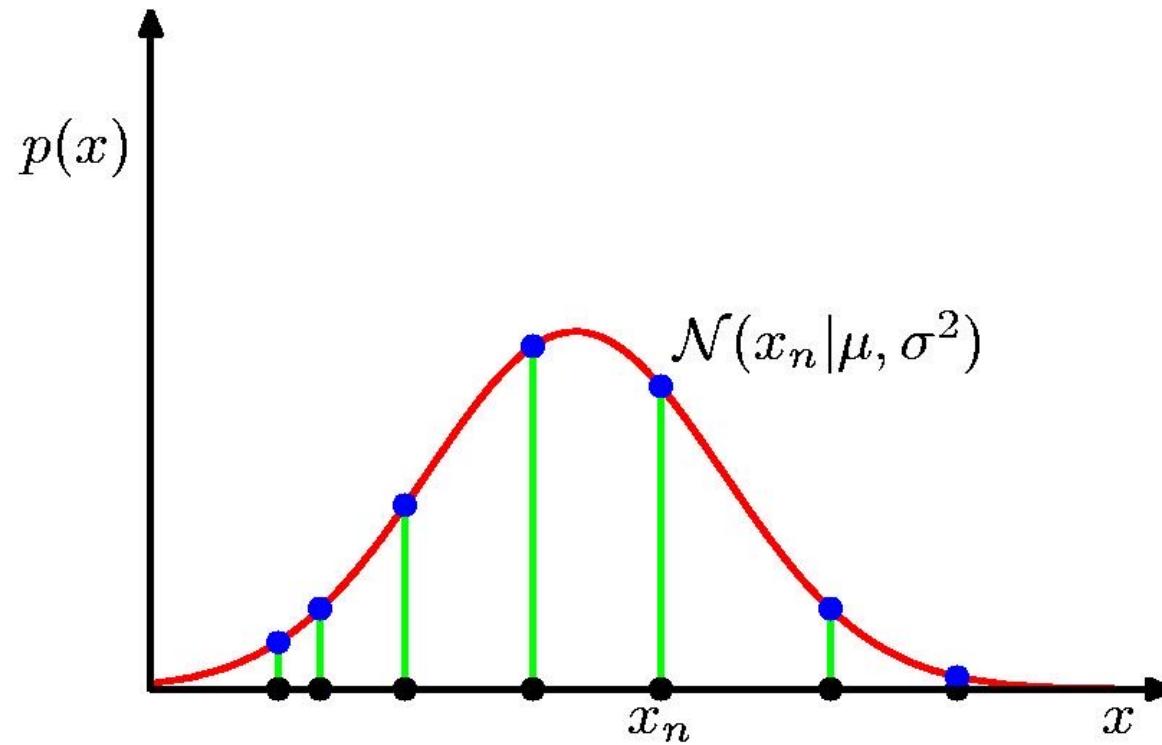
$$\text{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2$$

# Multivariate Gaussian

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp \left\{ -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$



# Likelihood function for the Gaussian



$$l(\theta|\mathcal{X}) = p(\mathcal{X}|\theta) = p(\mathcal{X}|\mu, \sigma^2) = \prod_{n=1}^N \mathcal{N}(x_n | \mu, \sigma^2)$$

# Gaussian parameter estimation - maximum (log) likelihood

$$L(\mu, \sigma^2 | x) = \ln p(x | \mu, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$$

$$\partial L / \partial \mu = 0 \text{ and } \partial L / \partial \sigma = 0$$

$$m = \mu_{\text{ML}} = \frac{1}{N} \sum_{n=1}^N x_n \quad s^2 = \sigma_{\text{ML}}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_{\text{ML}})^2$$

# Bayesian estimation

- ML has serious deficiencies with small training data sets.
- We may have some **prior** information on the possible value range that a parameter  $\theta$  may take, and turn towards Bayesian treatment, which
  - provides powerful insights into the issues of over-fitting and
  - leads to practical techniques for addressing the model complexity issue.

# Bayesian estimation - cont'd

- Treat  $\theta$  as a random variable with prior  $p(\theta)$  .
- The sample data tell the likelihood density  $p(\mathcal{X}|\theta)$  .
- Posterior density of  $\theta$  :  $p(\theta|\mathcal{X}) = p(\mathcal{X}|\theta)p(\theta) / p(\mathcal{X})$ .
- Parameter estimation
  - Maximum likelihood (ML):  $\theta_{\text{ML}} = \operatorname{argmax}_{\theta} p(\mathcal{X}|\theta)$
  - Maximum *a posteriori* (MAP):  $\theta_{\text{MAP}} = \operatorname{argmax}_{\theta} p(\theta|\mathcal{X})$   
ML estimator is a MAP estimator for the uniform prior.
  - Bayesian estimate:  $\theta_{\text{Bayes}} = E[\theta|\mathcal{X}] = \int \theta p(\theta|\mathcal{X}) d\theta$ , i.e., expectation

# ML vs. MAP (point estimation)

- Maximum likelihood is an estimator, when no prior distribution is available, in which  $\theta$  is set to the value that maximizes the likelihood function  $l(\theta|\mathcal{X}) = p(\mathcal{X}|\theta)$ .
  - $\theta_{\text{ML}} = \operatorname{argmax}_{\theta} p(\mathcal{X}|\theta)$
- Maximum a posteriori is an estimator in which  $\theta$  is determined by maximizing the posterior distribution  $p(\theta|\mathcal{X})$ .
  - $\theta_{\text{MAP}} = \operatorname{argmax}_{\theta} p(\theta|\mathcal{X})$

# Bayesian estimation

- Advantage of the Bayesian approach
  - The inclusion of prior knowledge - a reasonable prior - leads to a much less extreme conclusion, especially when the amount of training data is small.
- Disadvantage
  - The need to marginalize (sum or integrate) over the whole of parameter space.
  - $\theta_{\text{Bayes}} = E[\theta | \mathcal{X}] = \int \theta p(\theta | \mathcal{X}) d\theta$

# Lecture outline

- Parametric methods
  - Parameter estimation
  - Application: Parametric classification
  - Application: Parametric regression
- Nonparametric methods

# Bayesian parametric classification

- Two separate stages:
  - **Inference** stage: use a training data to learn a model for posterior  $p(C_k|x)$ .
  - **Decision** stage: use these posterior probabilities to make optimal class assignments.
- Computing posterior probabilities lies at the heart of **Bayesian classification**.
  - Choose the class that gives maximal posterior probability  $p(C_k|x)$  OR  $p(x|C_k) p(C_k)$

# An example: Gaussian

If we can assume that  $p(x|C_i)$  are Gaussian, then

- Given the sample  $\mathcal{X} = \{x^t, r^t\}_{t=1}^N$  where  $x \in \Re$  is 1-D and

$$r_i^t = \begin{cases} 1 & \text{if } x^t \in C_i \\ 0 & \text{if } x^t \in C_j, j \neq i \end{cases}$$

- ML estimates** are

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N}, \quad m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t}, \quad s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t}$$

- Compute posterior probabilities  $p(C_i | x)$  OR joint distribution  $p(x|C_k) p(C_k)$  for classification.

# Procedure as in the exercise

1. Compute priors if not given

$$\hat{P}(C_i) = \frac{\sum r_i^t}{N}$$

1. Computer means (`np.mean`) and variances (`np.var`) for 1-D / covariance matrix (`np.cov`) for multivariate

$$m_i = \frac{\sum_t x^t r_i^t}{\sum_t r_i^t}, \quad s_i^2 = \frac{\sum_t (x^t - m_i)^2 r_i^t}{\sum_t r_i^t}, \quad \Sigma_i = \frac{1}{\sum_t r_i^t} \sum_t (x^t - m_i)(x^t - m_i)^T r_i^t$$

2. Computer likelihood  $p(x|C_k)$  for test data
3. Compute posterior probabilities  $p(C_i | x)$  OR joint probability  $p(x|C_k) p(C_k)$  for classification
4. Calculate accuracy using the labels

# Partial code for exercise (a)

```
import numpy as np
from scipy.io import loadmat
from scipy.stats import multivariate_normal as norm
import matplotlib.pyplot as plt
```

```
#Load dataset
file = "data/dataset1_G_noisy.mat"
data = loadmat(file)
```

# Partial code for exercise (a)

- #Trainsets
- trn\_x = data["trn\_x"]
- trn\_y = data["trn\_y"]
- trn\_x\_class = data["trn\_x\_class"]
- trn\_y\_class = data["trn\_y\_class"]
  
- #Testsets
- tst\_xy = data["tst\_xy"]
- tst\_xy\_class = data["tst\_xy\_class"]
- tst\_xy\_126 = data["tst\_xy\_126"]
- tst\_xy\_126\_class = data["tst\_xy\_126\_class"]

# Partial code for exercise (a)

```
#Uniform Prior
```

```
prior_x = 0.5
```

```
prior_y = 0.5
```

```
#Estimate mean value and covariance for X and Y
```

```
mean_x = np.mean(trn_x, axis = 0)
```

```
mean_y = np.mean(trn_y, axis = 0)
```

```
cov_x = np.cov(trn_x.T)
```

```
cov_y = np.cov(trn_y.T)
```

```
#Create multivariate Gaussian distributions for X and Y
```

```
l_x = norm(mean = mean_x, cov = cov_x)
```

```
l_y = norm(mean = mean_y, cov = cov_y)
```

# Partial code for exercise (a)

```
#Posteriori probability (ignoring the normalization constant) on the test set
```

```
p1 = prior_x * l_x.pdf(tst_xy)
```

```
...
```

```
#Maximum a posteriori prediction
```

```
...
```

```
#Compute accuracy
```

```
...
```

# Classification accuracy

- Classification accuracy is defined as the percentage of correct predictions:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}$$

- Error/misclassification rate:

$$\text{error rate} = 1 - \text{accuracy}$$

- For binary classification, accuracy can also be calculated as

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP=true positive, TN=true negative, FP=false positive and FN=false negative

# Confusion matrix

		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

TN              True Negative  
 FP              False Positive  
 FN              False Negative  
 TP              True Positive

## Model Performance

Accuracy               $= (TN+TP)/(TN+FP+FN+TP)$

Precision               $= TP/(FP+TP)$

Sensitivity               $= TP/(TP+FN)$

Specificity               $= TN/(TN+FP)$

- Accuracy measures overall correct classification of the model.
- Precision measures accuracy of a class, when predicts "Yes" how often it is correct.
- Sensitivity or Recall: When it is actually "yes" how often it predicts "Yes".
- Specificity: Specificity measures true negative rate. When it is actually "No", how often it is "No".

# Lecture outline

- Parametric methods
  - Parameter estimation
  - Application: Parametric classification
  - Application: Parametric regression
- Nonparametric methods

# Regression: ML estimation

- Output = function of input + random noise:

$$r = f(x) + \varepsilon \text{ where } \varepsilon \sim N(0, \sigma^2).$$

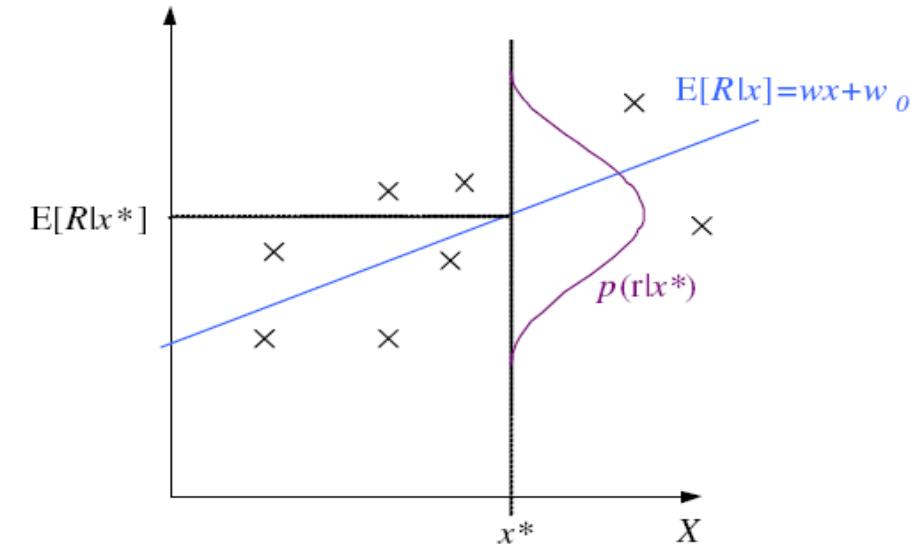
Let  $g(x|\theta)$  be the estimator of  $f(x)$ , then ML estimator of  $r$  is

$$p(r|x) \sim N(g(x|\theta), \sigma^2)$$

- Log likelihood of parameters  $\theta$

$$\mathcal{L}(\theta | \mathcal{X}) = \ln \prod_{t=1}^N p(x^t, r^t)$$

$$= \ln \prod_{t=1}^N p(r^t | x^t) + \ln \prod_{t=1}^N p(x^t)$$



where the pairs  $(x^t, r^t)$  in the training set  $\mathcal{X}$  are drawn from an unknown joint probability density  $p(x, r)$ .

# Regression: from ML to error function

Ignore the second term in ML estimator, then:

$$\begin{aligned} \operatorname{argmax}_{\theta} \mathcal{L}(\theta | \mathcal{X}) &= \ln \prod_{t=1}^N \frac{1}{\sqrt{2\pi}\sigma} \exp \left[ -\frac{[r^t - g(x^t | \theta)]^2}{2\sigma^2} \right] \\ &= -N \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum_{t=1}^N [r^t - g(x^t | \theta)]^2 \end{aligned}$$

Maximizing  $L(\theta|X)$  equal to minimizing the following error function:

$$E(\theta | \mathcal{X}) = \frac{1}{2} \sum_{t=1}^N [r^t - g(x^t | \theta)]^2$$

which is the most frequently used **error function**, and  $\theta$  that minimize it are called the **least squares estimates**

- least squares estimates are equivalent to ML solution under an assumed Gaussian noise model.

# Lecture outline

- Parametric methods
  - Parameter estimation
  - Application: Parametric classification
  - Application: Parametric regression
- Nonparametric methods

# Nonparametric methods

- **Nonparametric** methods: statistical techniques used when the underlying data distribution is unknown or cannot be described by a specific mathematical distribution.
- Non-parametric methods make fewer assumptions about the data and are more flexible.
- Such models still contain parameters, but they control the model complexity rather than the form of the distribution.
- Rational: Similar inputs have similar outputs.
- Keep the training data; let the data speak for itself.

“Non-parametric model is not none-parametric.”

# Nonparametric methods - cont'd

- Do not assume a model valid over the whole space (parametric) or a part of it (semiparametric); postpone the computation of the model until they are given a test instance.
- Have increased need for memory and computation.
- Aka lazy/memory-based learning.

# Nonparametric methods

- **Histograms**: greatly depend on the value of the **bin width**.
  - In high dimensionality, the quantity of data needed to provide meaningful estimates of local density would be prohibitive.
- $p(x) = \frac{K}{NV}$ , N is the number of observations, K the number of points that lie inside R, and V the volume of R.
  - Either we can fix V and determine K from the data, giving rise to the **kernel approach**, or
  - we can fix K and determine the value of V from the data, which gives rise to the **K-nearest-neighbour** techniques.

# Non-parametric density estimation

- Given the training set  $X = \{x^t\}_t$ , drawn iid from  $p(x)$
- Divide data into bins of size  $h$
- Histogram:

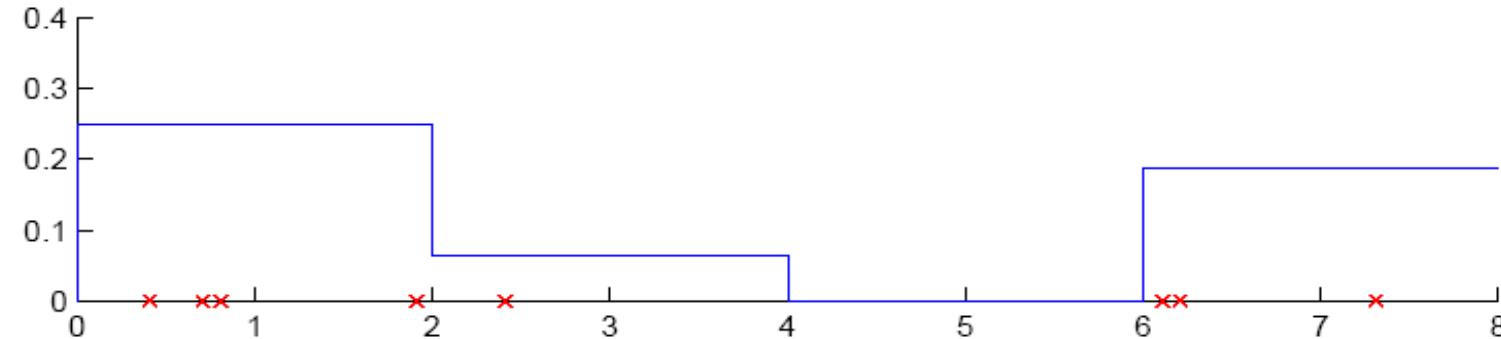
$$\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh}$$

N is the total number of observations

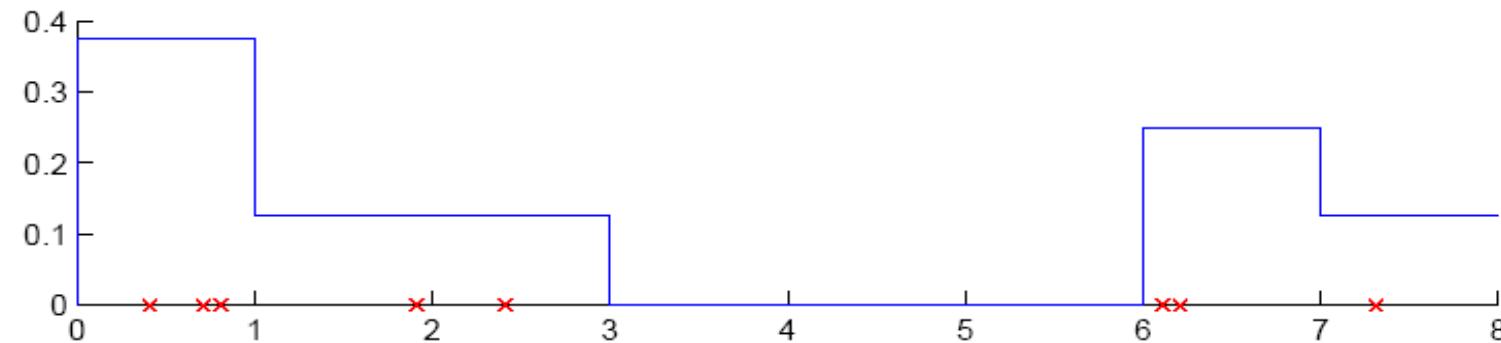
Fixed bins:  $[x_0 + mh, x_0 + mh + h]$

# Density Estimation

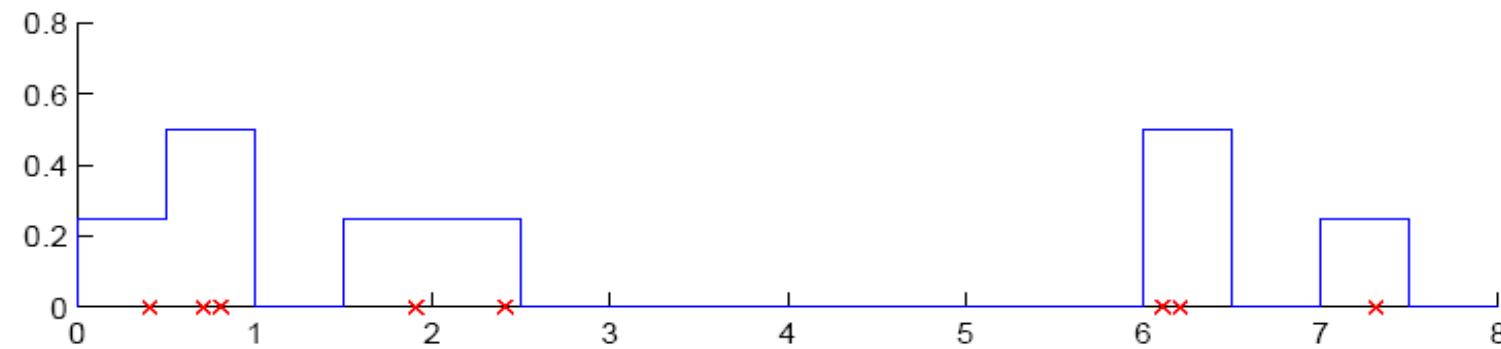
Histogram:  $h=2$



$h=1$



$h=0.5$



# Kernel Estimator

- To get a smooth estimate, we use a smooth weight function, called a Kernel function. The most popular one is Gaussian kernel:

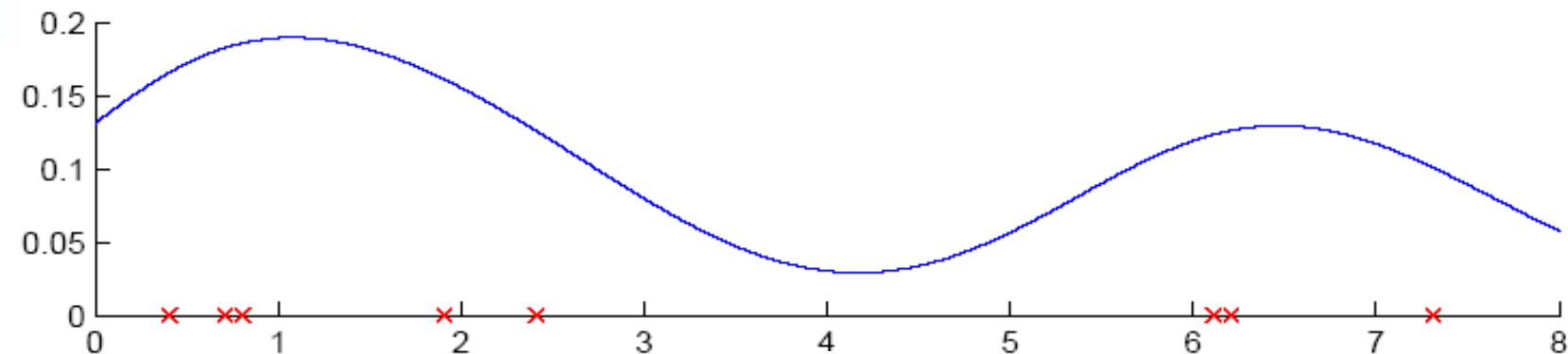
$$K(u) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{u^2}{2}\right]$$

- The Kernel estimator (also called Parzen windows)

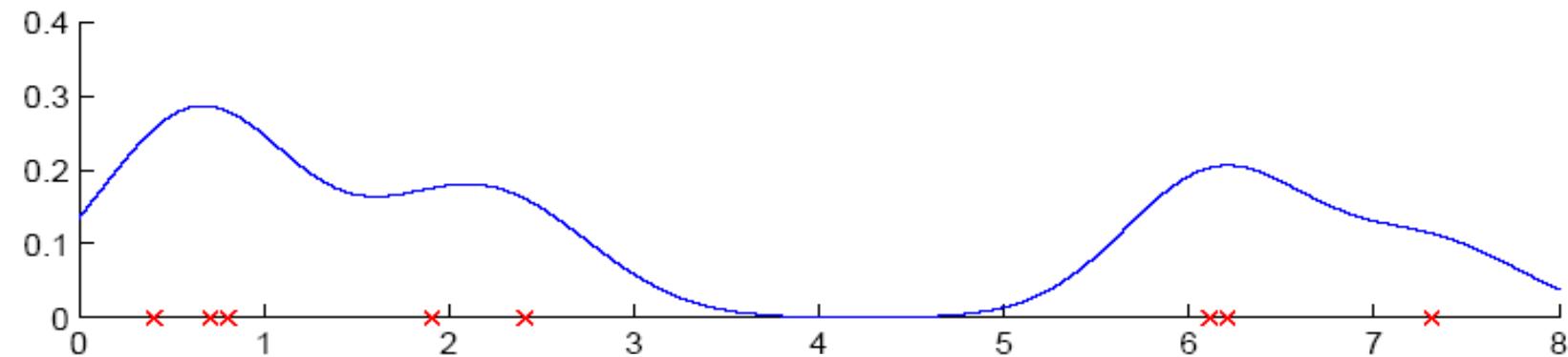
$$\hat{p}(x) = \frac{1}{Nh} \sum_{t=1}^N K\left(\frac{x - x^t}{h}\right) \longrightarrow \begin{array}{l} \text{All observations} \\ \text{Smoothed} \end{array}$$

- (histogram:  $\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh}$ )  $\longrightarrow$  Observations in the  
same bin only  
Rectangular kernel

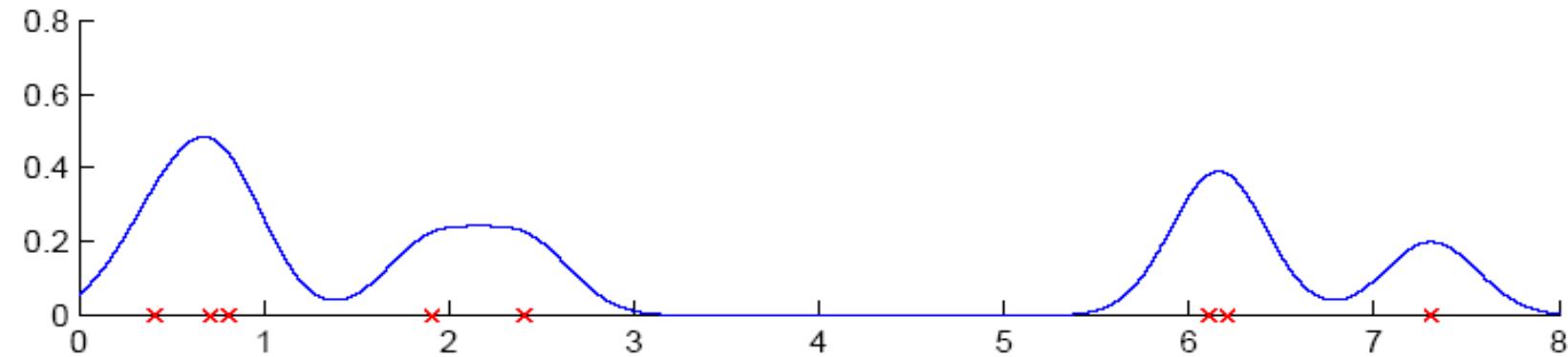
Kernel estimator:  $h=1$



$h=0.5$



$h=0.25$



# k-Nearest Neighbor Estimator

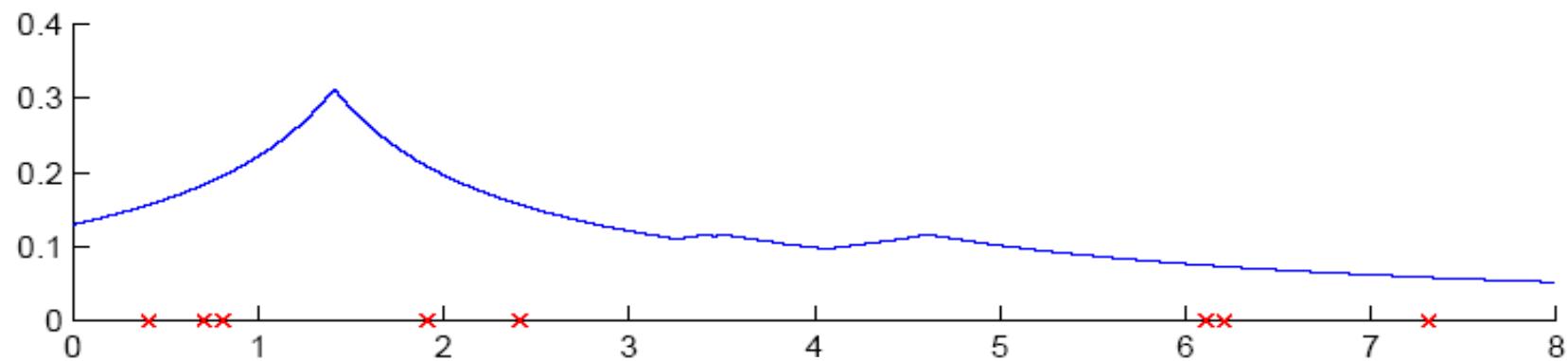
- Instead of fixing bin width  $h$  and counting the number of instances, fix the instances (neighbors)  $k$  and compute bin width

$$\hat{p}(x) = \frac{k}{2Nd_k(x)}$$

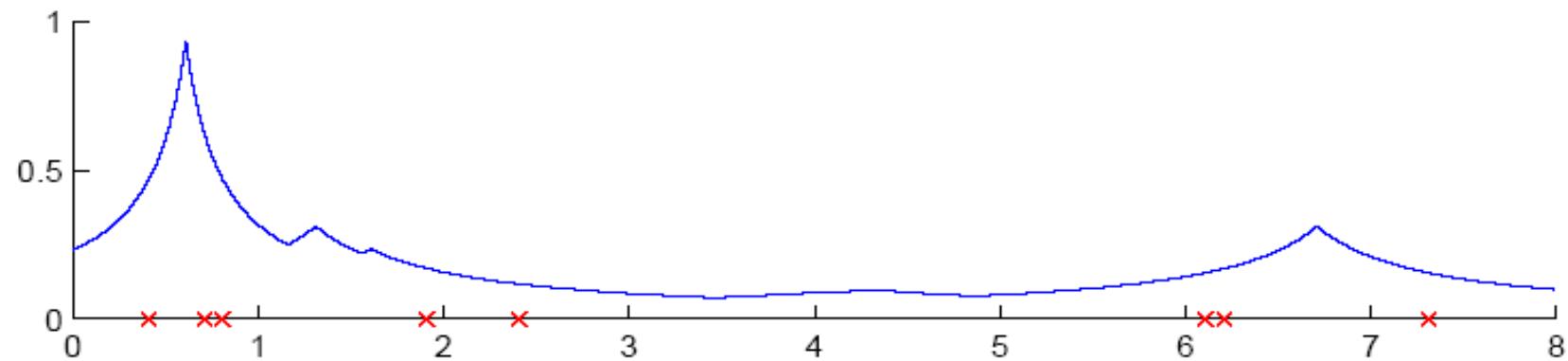
where  $d_k(x)$  is the distance to the  $k$ th closest instance to  $x$ .

(histogram:  $\hat{p}(x) = \frac{\#\{x^t \text{ in the same bin as } x\}}{Nh}$ )

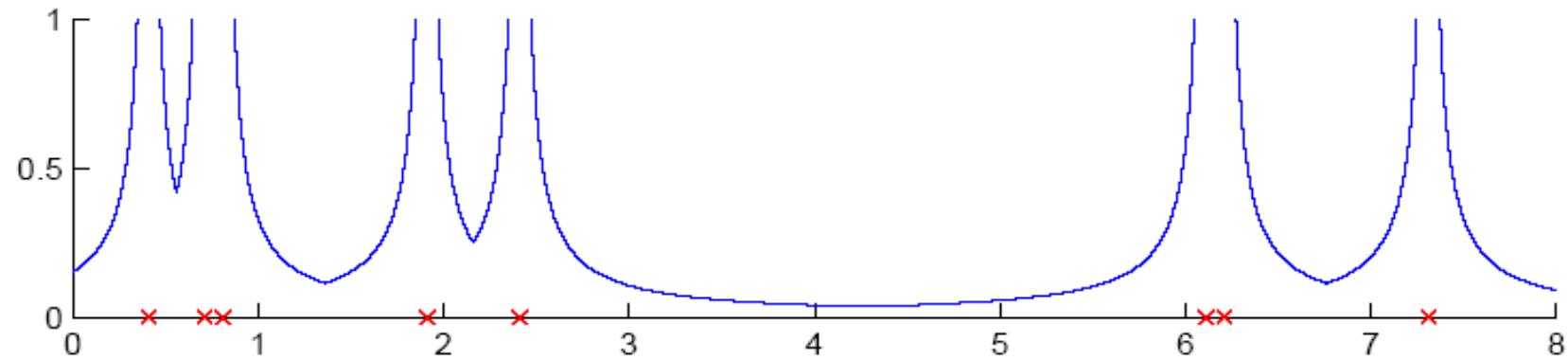
k-NN estimator: k=5



k=3



k=1



# Nonparametric Classification

- Estimate  $p(x|C_i)$  and use Bayes' rule
- $K$ -NN estimator

$$\hat{p}(x|C_i) = \frac{k_i}{N_i V^k(x)}$$

$$\hat{p}(x) = \frac{k}{2Nd_k(x)}$$

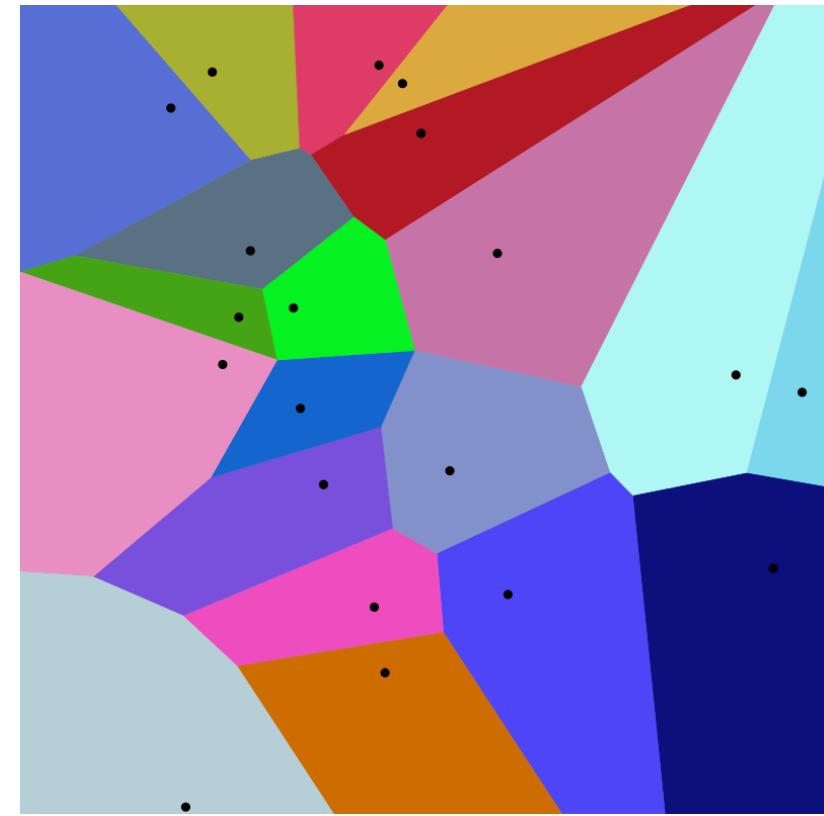
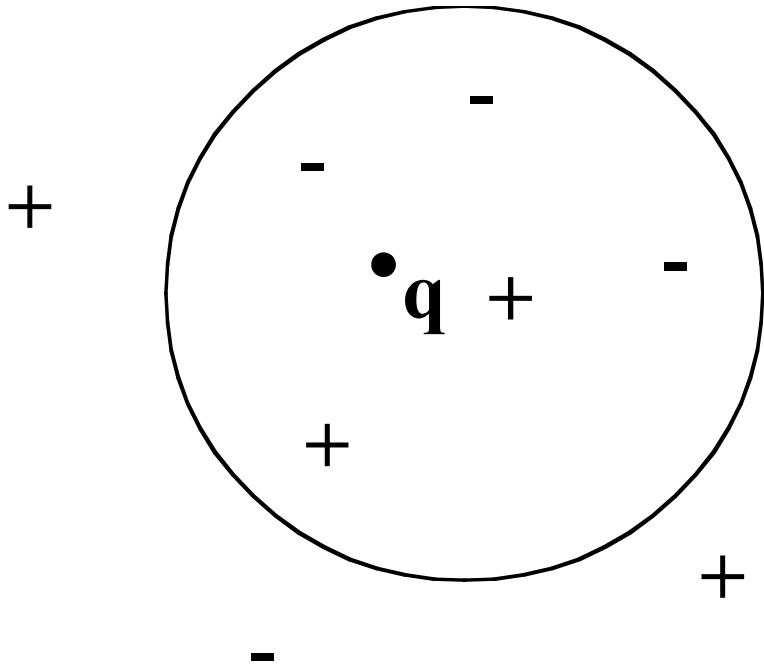
$$\hat{P}(C_i | x) = \frac{\hat{p}(x | C_i) \hat{P}(C_i)}{\hat{p}(x)} = \frac{k_i}{K}$$

k-NN is a widely used classification method.

# K-nearest neighbor classification

- The simplest, most used instance-based learning algorithm
- $k$ -NN assumes that all instances are points in some  $n$ -dimensional space and defines neighbors in terms of distance
- Training:
  - Store the training examples
- Inference/prediction:
  - Find the  $k$  training examples  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_k, y_k)$  that are closest to the test example  $\mathbf{x}$
  - Predict the most frequent class among those  $y_i$ 's.

# K-nearest neighbor classification



Voronoi diagram. All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample

# How to Choose $k$ or $h$ ?

- When  $k$  or  $h$  is small, single instances matter; bias is small, variance is large (undersmoothing): High complexity
- As  $k$  or  $h$  increases, we average over more instances and variance decreases but bias increases (oversmoothing): Low complexity
- Cross-validation is used to fine-tune  $k$  or  $h$ .

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 4: Dimensionality Reduction

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification (including figures)*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Unsupervised learning: goals

Discover the unknown structure of the inputs

- Model data density (Lecture 3): determine probability density distribution (pdf) of data within input space, e.g. histogram, k-NN, kernel
- **Reduce dimensionality** (Lecture 4)
- Find clusters (Lecture 5): discover groups of similar examples (clumps of points) within the data, e.g. k-means, EM

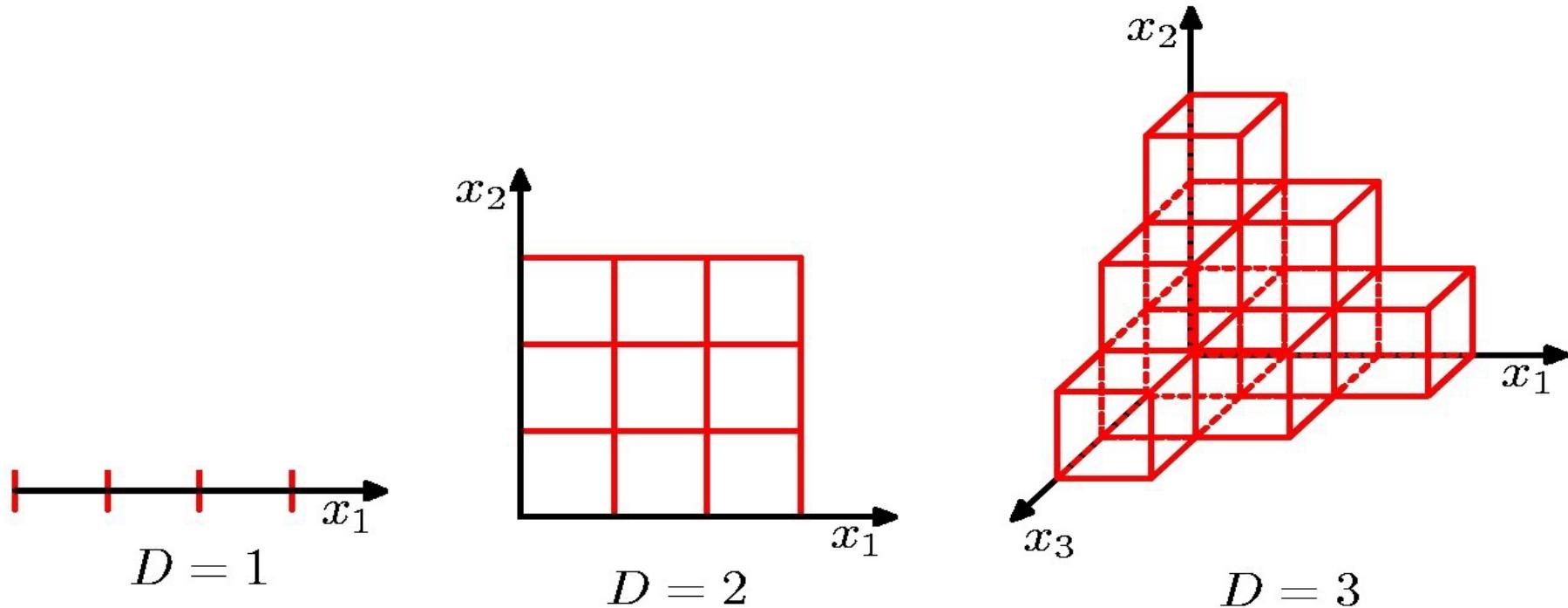
# Lecture outline

- The curse of dimensionality
- Feature selection
- Feature extraction
  - Principal component analysis
  - Linear discriminant analysis

# The curse of dimensionality

- Refer to the exponential growth of volume as a function of dimensionality. One example is VQ.
- Features
  - May involve hundreds, or even hundreds of thousands features.
  - Each feature may not provide independent information.
- Classifiers
  - Classification accuracy depends on the dimensionality (and amount of training data)
  - Computational complexity of the classifier

# The curse of dimensionality



# Motivation to reduce dimensionality

- Reduce time complexity: Less computation
- Reduce space complexity: Less parameters
- Simpler models are more robust on small datasets
- More interpretable; simpler explanation
- Data visualization (structure, groups, outliers, etc.) if plotted in 2 or 3 dimensions

# Dimensionality reduction

- It is the process of reducing the number of random variables:
  - **Feature selection** approaches try to find a subset of the original variables
  - **Feature extraction** is to apply a mapping of the multidimensional space into a space of fewer dimensions - the original feature space is transformed.
- It is a slightly simpler problem than density estimation:
  - Find a low-dimensional surface (a manifold) that is as close as possible to the training samples.

# Feature selection vs. extraction

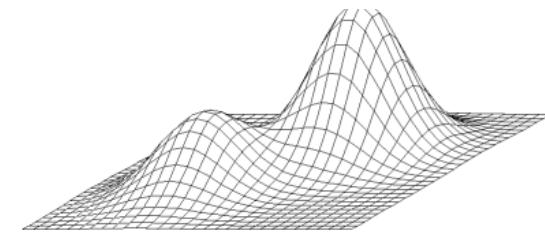
- **Feature selection:** Choosing  $M < D$  important features, ignoring the remaining  $D - M$ 
  - subset selection algorithms
- **Feature extraction:** Project the original  $x_i$ ,  $i = 1, \dots, D$  dimensions to new  $M < D$  dimensions,  $z_j$ ,  $j = 1, \dots, M$ 
  - principal components analysis (PCA),
  - linear discriminant analysis (LDA),

# Lecture outline

- The curse of dimensionality
- Feature selection
- Feature extraction
  - Principal component analysis
  - Linear discriminant analysis

# Subset Selection

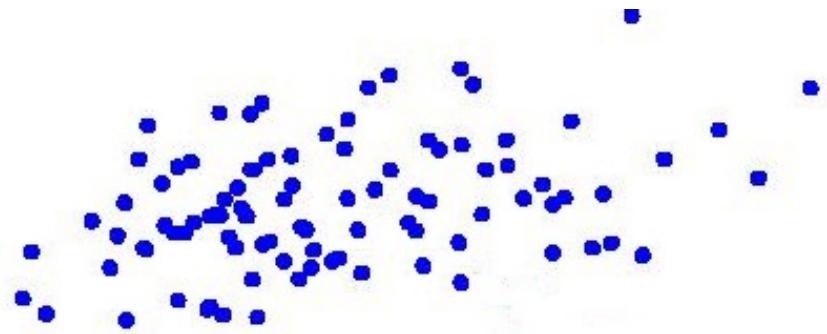
- There are  $2^D$  possible subsets of  $D$  features
- Forward search: Add the best feature at each step
  - Set of features  $F$  initially  $\emptyset$ .
  - At each iteration, find the best new feature
$$j = \operatorname{argmin}_i E(F \cup x_i)$$
    - Add  $x_j$  to  $F$  if  $E(F \cup x_j) < E(F)$
- Hill-climbing algorithm
- Backward search: Start with all features and remove one at a time, if possible.
- Floating search (Add  $k$ , remove  $l$ )



# Lecture outline

- The curse of dimensionality
- Feature selection
- Feature extraction
  - Principal component analysis
  - Linear discriminant analysis

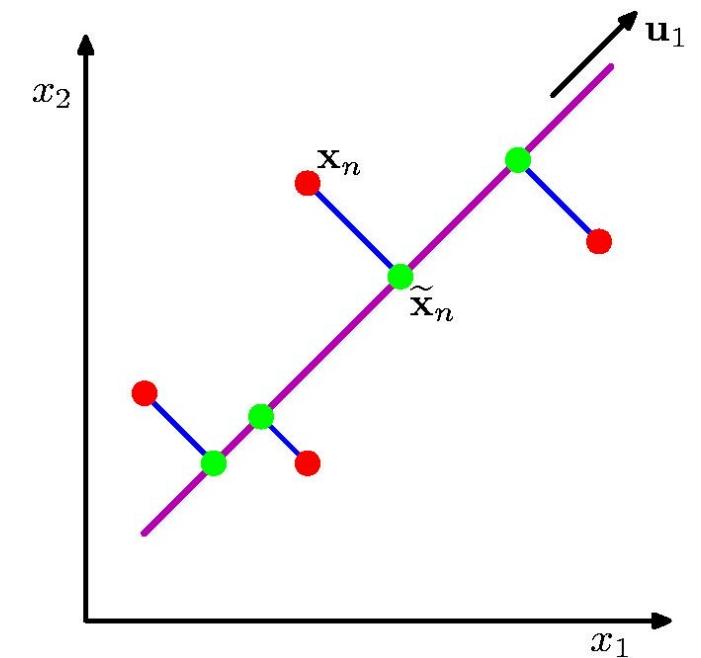
# Projection for dimensionality reduction



- How to project it into one dimension?

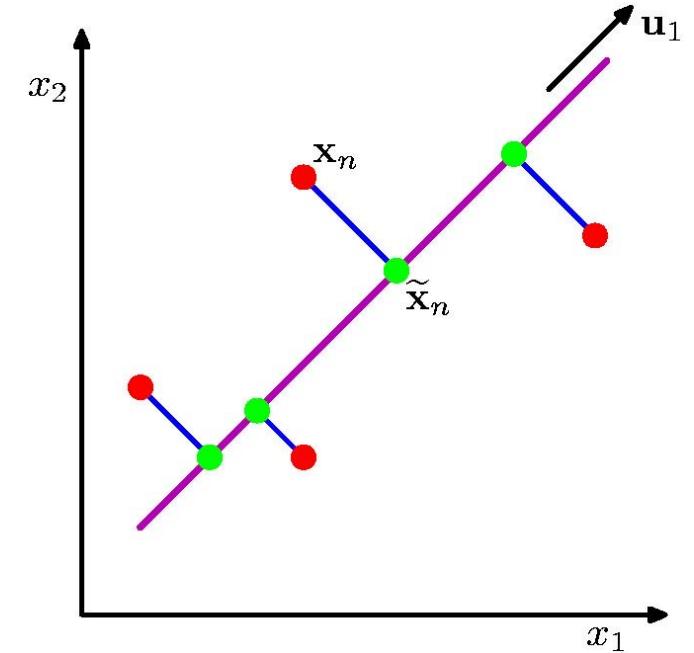
# Principal components analysis

- The main linear tech for **dimensionality reduction.**
- Search for direction in data space which has highest variance, and subsequently project the data onto it.
  - It is an orthogonal projection.
  - the **variance** of the projected data is **maximized**.
- Maximize the variance of the projected points (**green dots**).



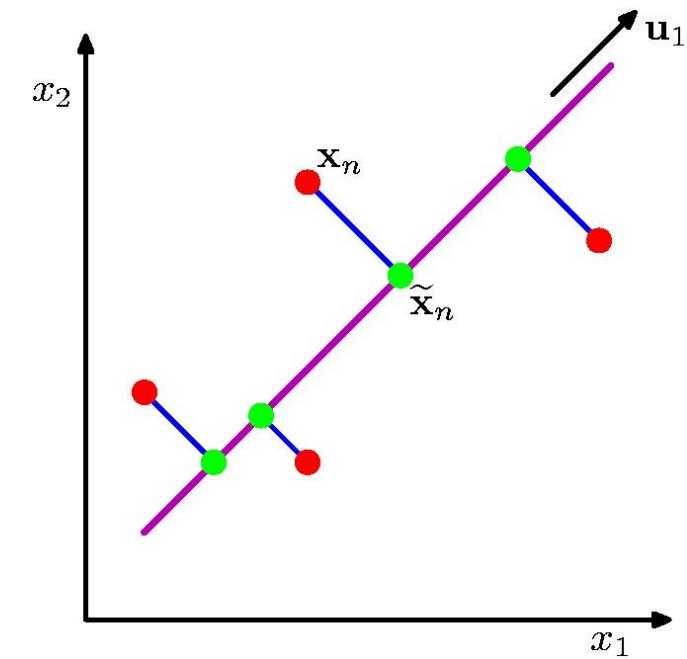
# Principal components analysis

- Equivalently, it can be defined as the linear projection that **minimizes** the average **projection cost**
  - The cost is defined as the mean squared distance between the data points and their projections.
  - Minimize the sum of squares of the projection errors (**blue lines**).
- PCA is the most popular instance of **projection** methods, and the 2nd main class of unsupervised learning methods (the other is clustering).



# Maximum variance formulation

- Consider a dataset of observations  $\{\mathbf{x}_n\}$  where  $n=1, \dots, N$  and  $\mathbf{x}_n$  is a Euclidean variable with dimensionality  $D$ .
- The aim is to project the data onto a space having dimensionality  $M < D$  while maximizing the variance of the projected data.



# Project onto a 1-D space

- Define the direction of this 1-*D* space using a *D*-dimensional vector  $\mathbf{w}_1$ , which is a unit vector so that  $\mathbf{w}_1^T \mathbf{w}_1 = 1$ .
- Each data point  $\mathbf{x}_n$  is then projected onto a scalar value  $\mathbf{w}_1^T \mathbf{x}_n$ . The projected data mean is  $\mathbf{w}_1^T \mathbf{m}$  where  $\mathbf{m}$  is the sample set mean

$$\mathbf{m} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$$

- and the **variance** of the projected data is given by

$$\frac{1}{N} \sum_{n=1}^N \{\mathbf{w}_1^T \mathbf{x}_n - \mathbf{w}_1^T \mathbf{m}\}^2 = \mathbf{w}_1^T \Sigma \mathbf{w}_1$$

where the data covariance matrix of *DxD* dims is defined by

$$\Sigma = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \mathbf{m})(\mathbf{x}_n - \mathbf{m})^T$$

# Project onto a one-dimensional space

- We now maximize the projected variance  $\mathbf{w}_1^T \Sigma \mathbf{w}_1$  with respect to  $\mathbf{w}_1$  with the constraint  $\mathbf{w}_1^T \mathbf{w}_1 = 1$ . To do so, we introduce a Lagrange multiplier,  $\lambda$ , and then make an unconstrained maximization of

$$\mathbf{w}_1^T \Sigma \mathbf{w}_1 + \lambda(1 - \mathbf{w}_1^T \mathbf{w}_1)$$

- By setting the derivative with respect to  $\mathbf{w}_1$  equal to zero, we have

$$\Sigma \mathbf{w}_1 = \lambda \mathbf{w}_1$$

- which says that  $\mathbf{w}_1$  must be an eigenvector of  $\Sigma$ . The variance is given by

$$\mathbf{w}_1^T \Sigma \mathbf{w}_1 = \lambda_1$$

- and so the variance will be a maximum when setting  $\mathbf{w}_1$  equal to the eigenvector having the largest eigenvalue  $\lambda_1$ . This eigenvector is the first principal component.

# Find additional principal components

- Incrementally choose each new direction to be the one that maximizes the projected variance amongst all possible directions orthogonal to those already considered.
- The optimal linear project is defined by the  $M$  eigenvectors  $\mathbf{w}_1, \dots, \mathbf{w}_M$  of the data covariance matrix corresponding to the  $M$  largest eigenvalues  $\lambda_1, \dots, \lambda_M$ .

# Principal components analysis

- In practice, the covariance matrix of data is constructed and eigenvectors on this matrix are computed.
  - Use statistics from the training data, not from test data.
  - Unsupervised learning, so pool training data of all classes together to find the PCA solution; not one PCA project space per class.
- The **eigenvectors** corresponding to the largest eigenvalues (the principal components) can now be used to reconstruct a large fraction of the variance of the original data.
- The original space has been reduced (with data loss, but hopefully retaining the most important variance) to the space spanned by a few eigenvectors.

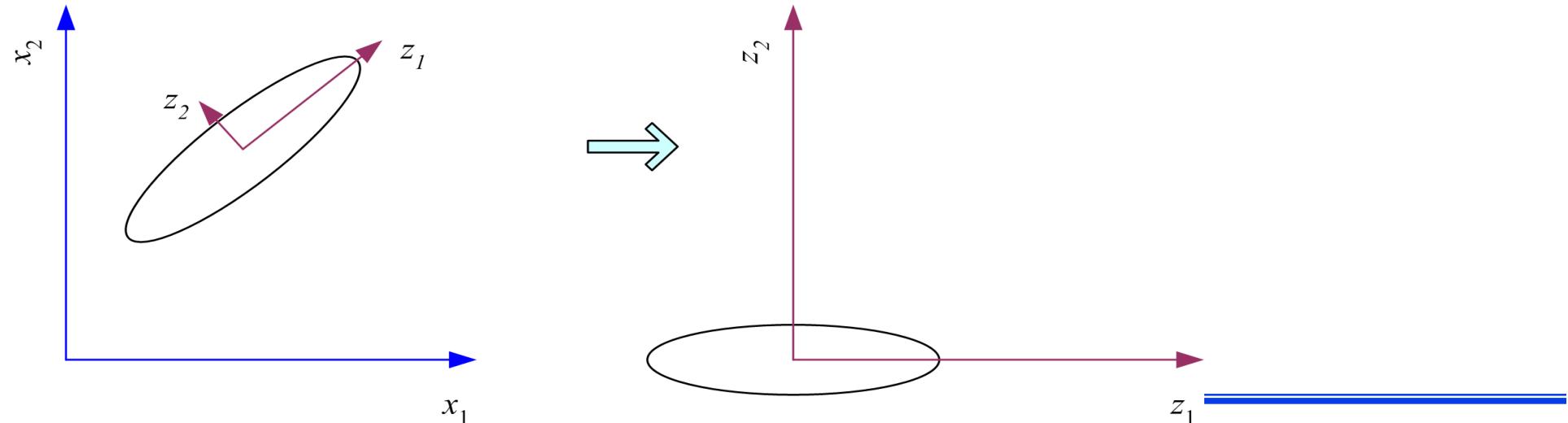
# What PCA does

- The projection of  $\mathbf{x}$  on the direction of  $\mathbf{w}$  is:  $z = \mathbf{w}^T \mathbf{x}$ . We define

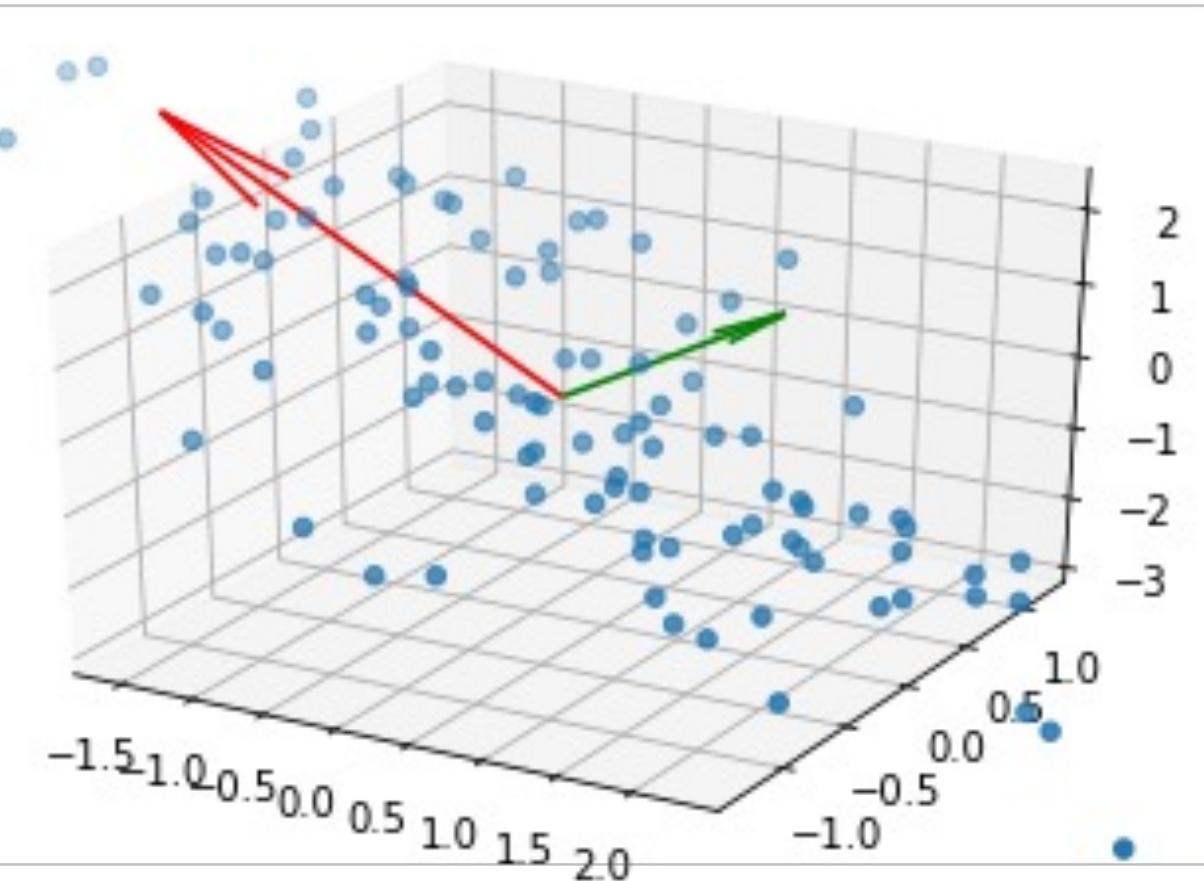
$$\mathbf{z} = \mathbf{W}^T(\mathbf{x} - \mathbf{m})$$

where the columns of  $\mathbf{W}$  are the eigenvectors of  $\Sigma$ , and  $\mathbf{m}$  is sample mean.

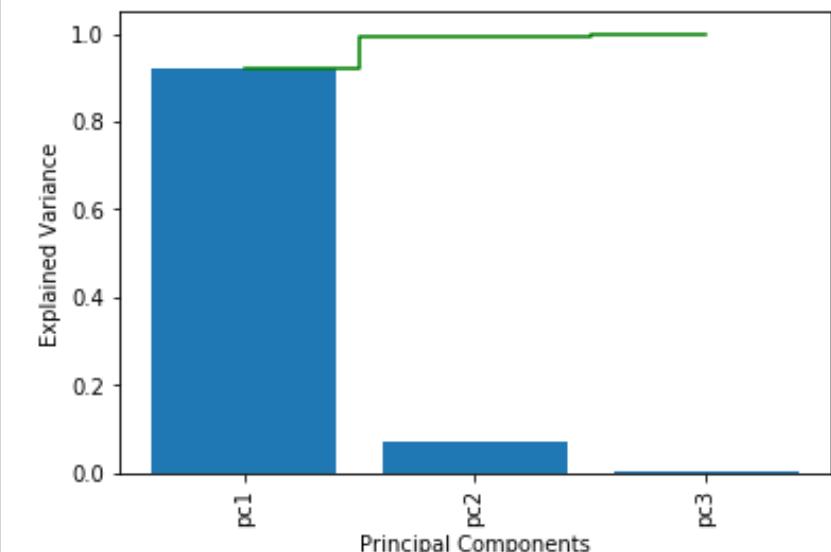
- Centers the data at the origin and rotates the axes.
- Reconstruction  $\mathbf{x}' = \mathbf{W}\mathbf{z} + \mathbf{m}$



# Principal components and variance

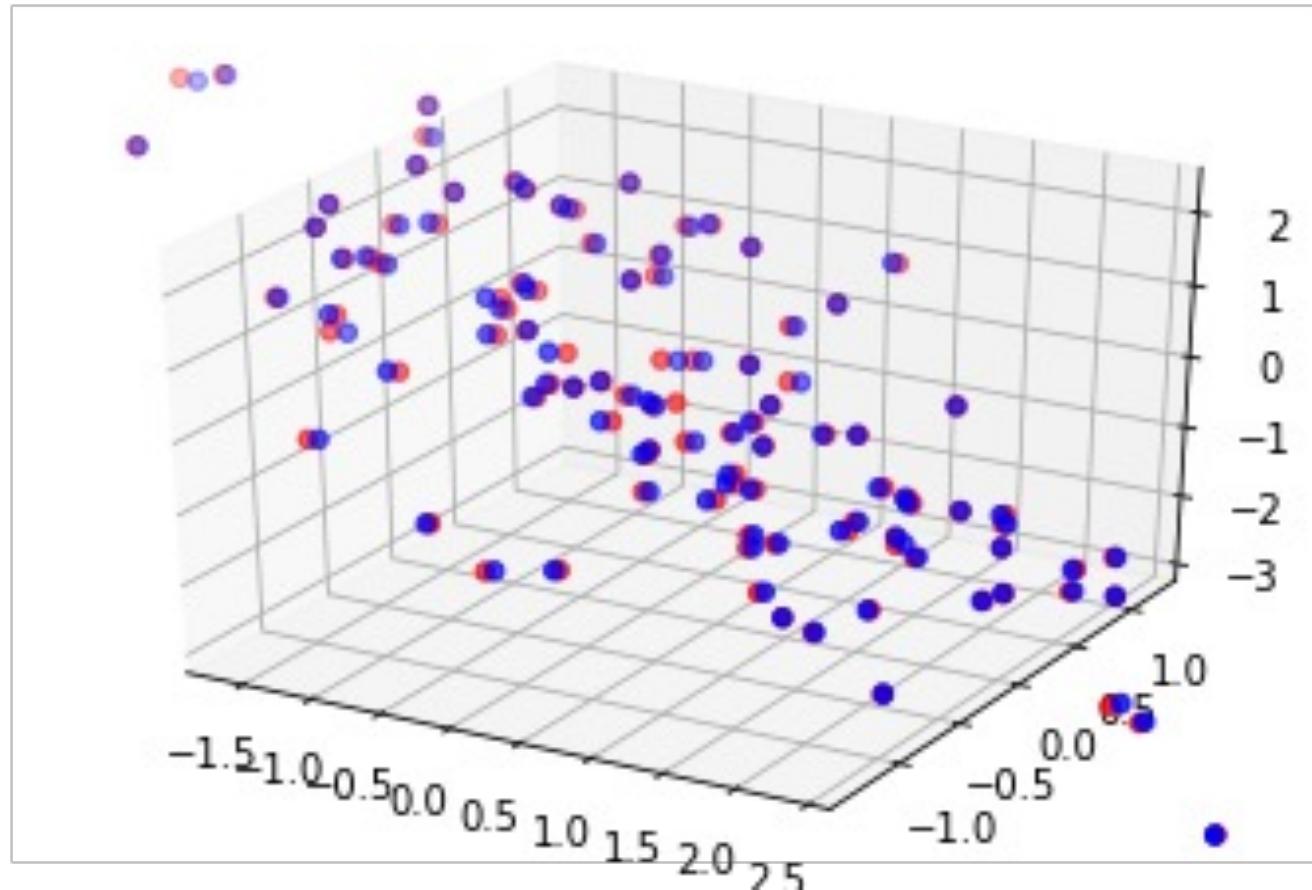


First two principal components



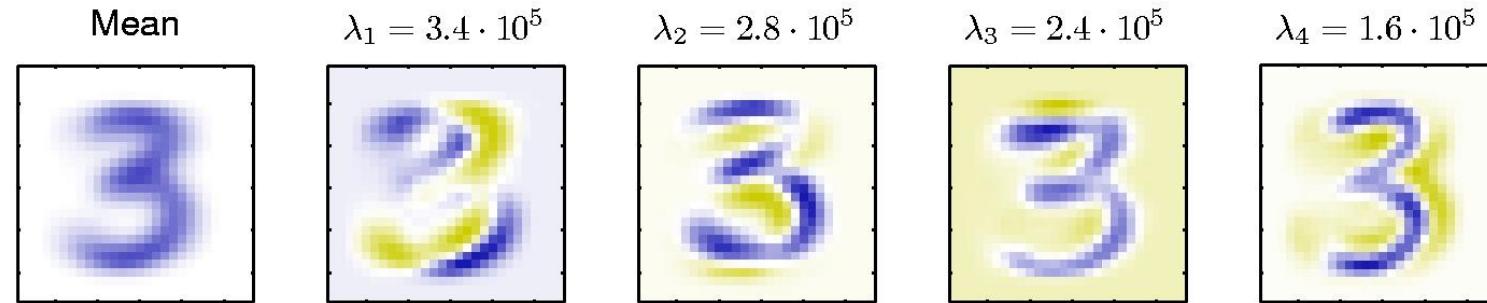
The eigenvalue magnitude signifies on how much variance each component explains.

# Reconstruction

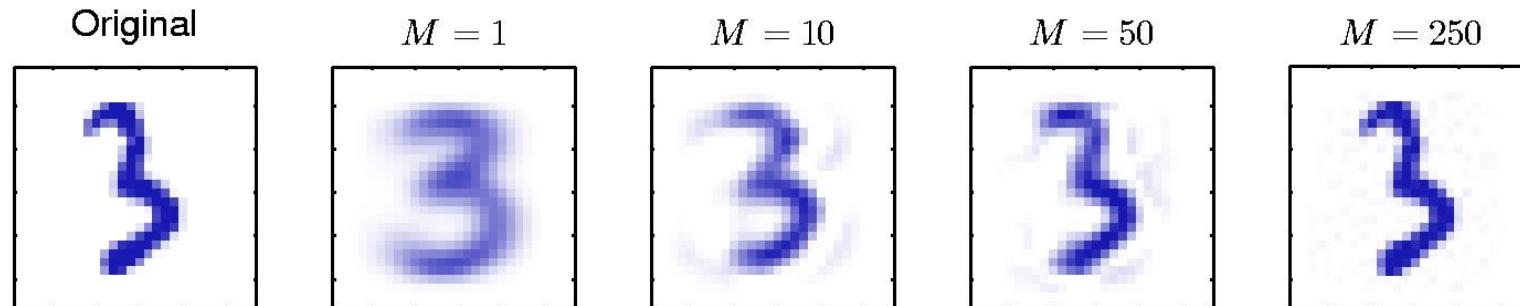


# PCA for data compression

- The mean vector along with the first 4 PCA eigenvectors for a digits dataset.



- An original example from the digits dataset together with its PCA reconstructions by retaining  $M$  principal components. When  $M=D=28\times 28=784$ , it would be perfect.



# Face recognition using eigenfaces



<http://www.cs.princeton.edu/~cdecoro/eigenfaces/>

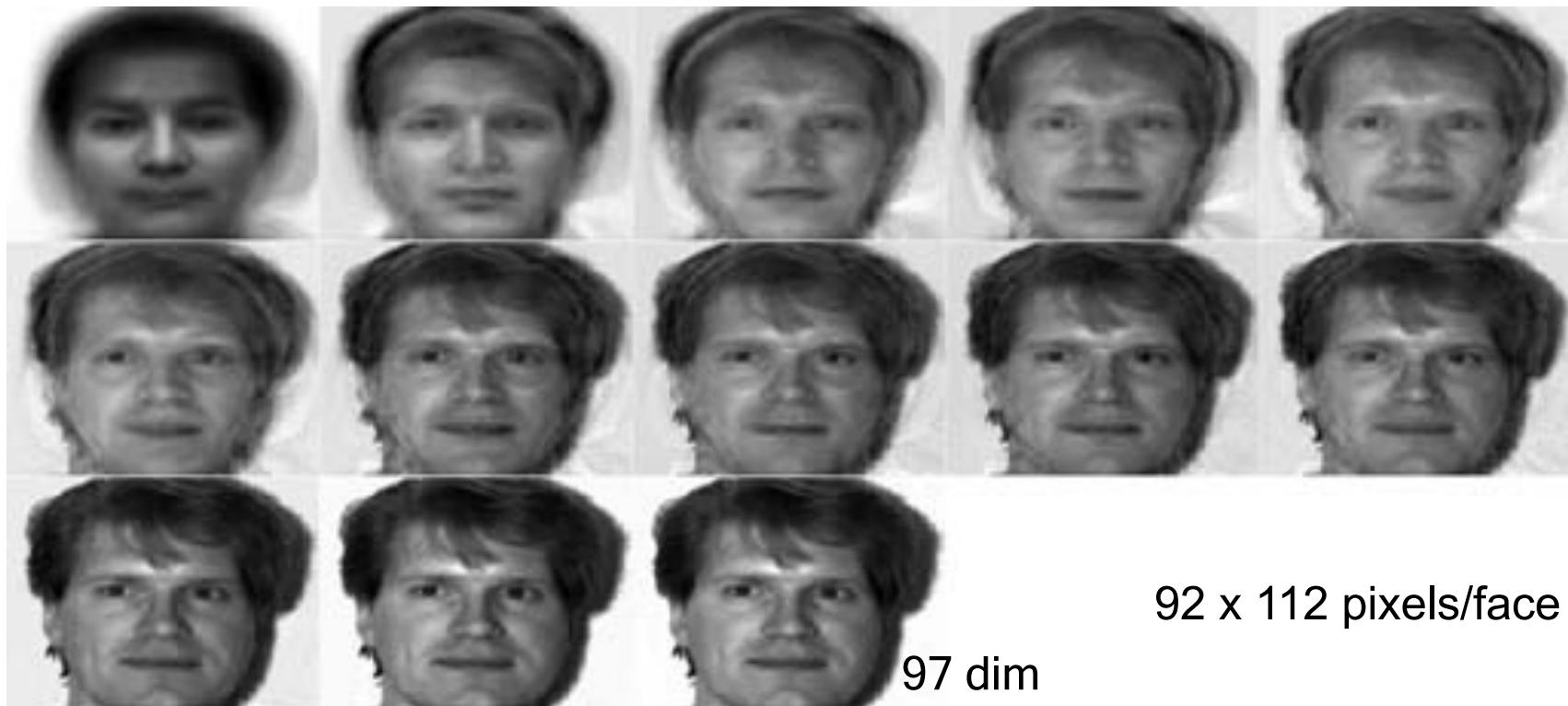
# Face recognition using eigenfaces - cont.

Each new image from left to right corresponds to using 1 additional principle component for reconstruction.



# Face recognition using eigenfaces - cont.

Each additional face representing an additional 8 principle components.



# How to choose M (<D) ?

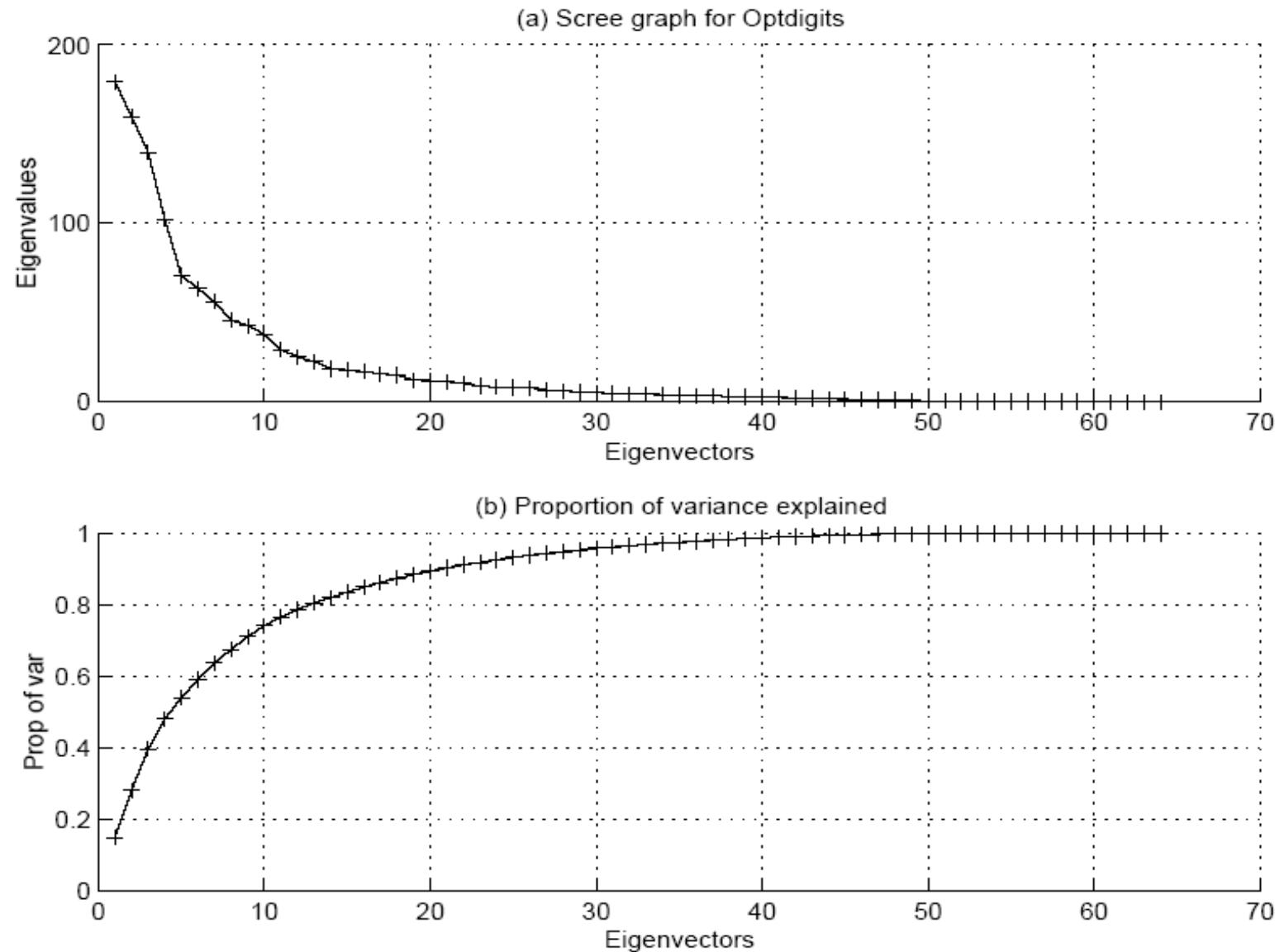
- Proportion of Variance (PoV) is

$$\frac{\lambda_1 + \lambda_2 + \dots + \lambda_M}{\lambda_1 + \lambda_2 + \dots + \lambda_M + \dots + \lambda_D}$$

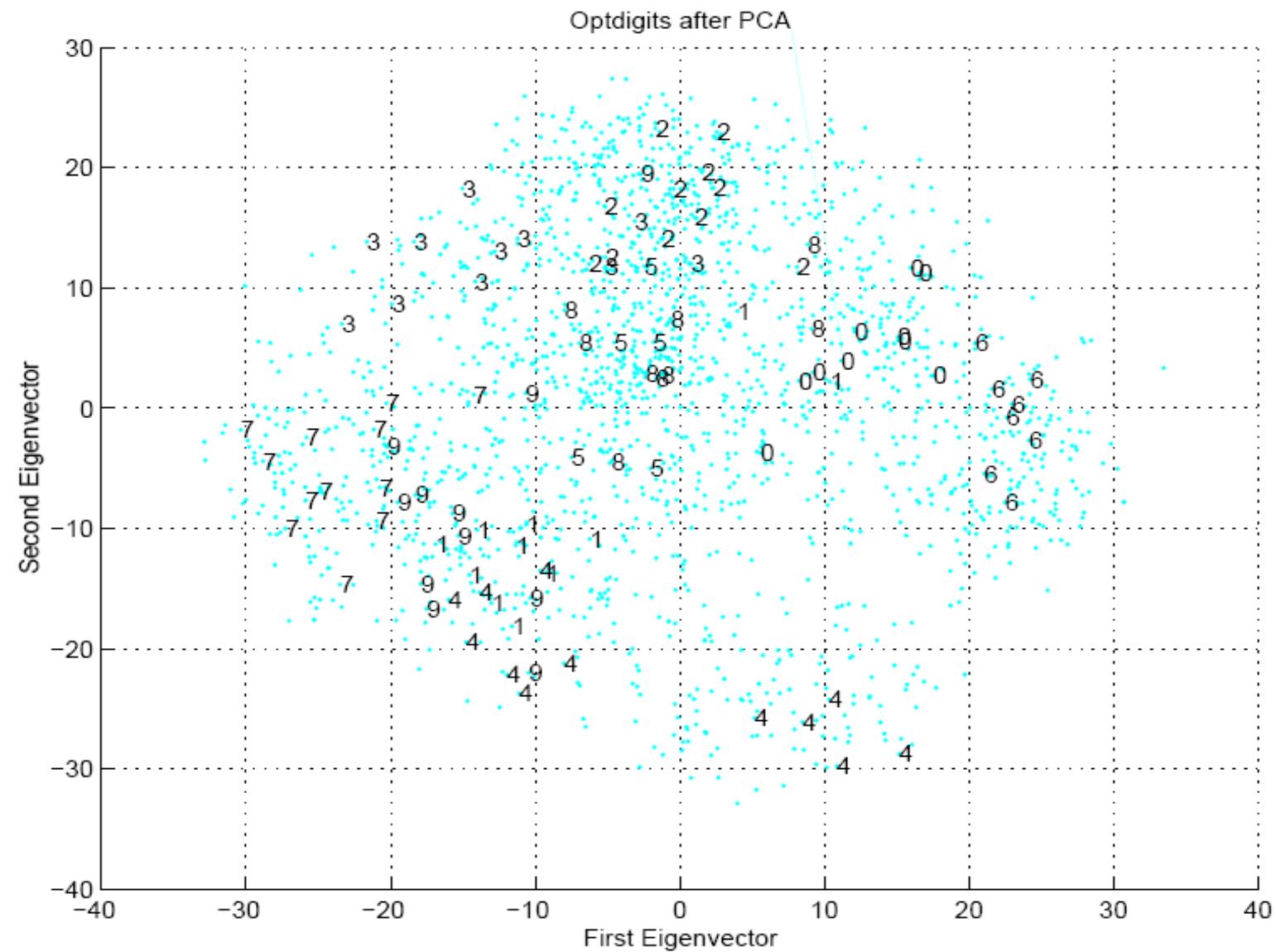
when  $\lambda_i$  are sorted in descending order.

- Typically, stop at PoV>0.9.

# Principal components analysis



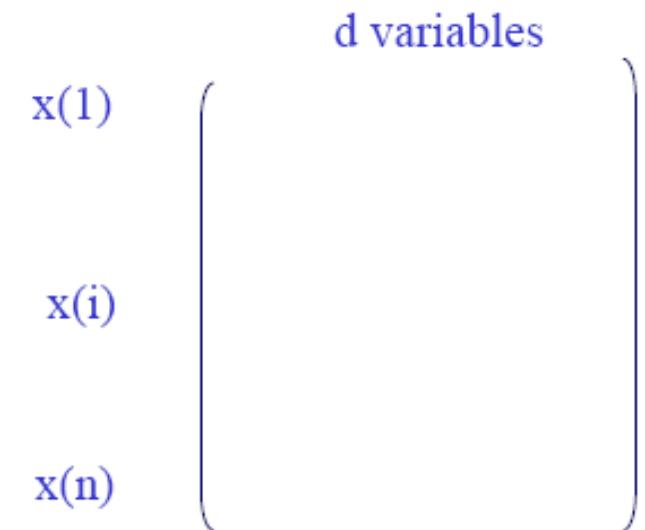
# Principal components analysis



# Dimensionality reduction and clustering

- PCA (variance-oriented) reduces dimensionality by forming linear combinations of the features.
- Given the data matrix, whose  $n$  rows are the  $d$ -dimensional samples:
  - dimensionality reduction can be thought of as a grouping of the columns, with combined features being used to represent the data.
  - clustering can be thought of as grouping the rows, with a smaller number of cluster centers being used to represent the data

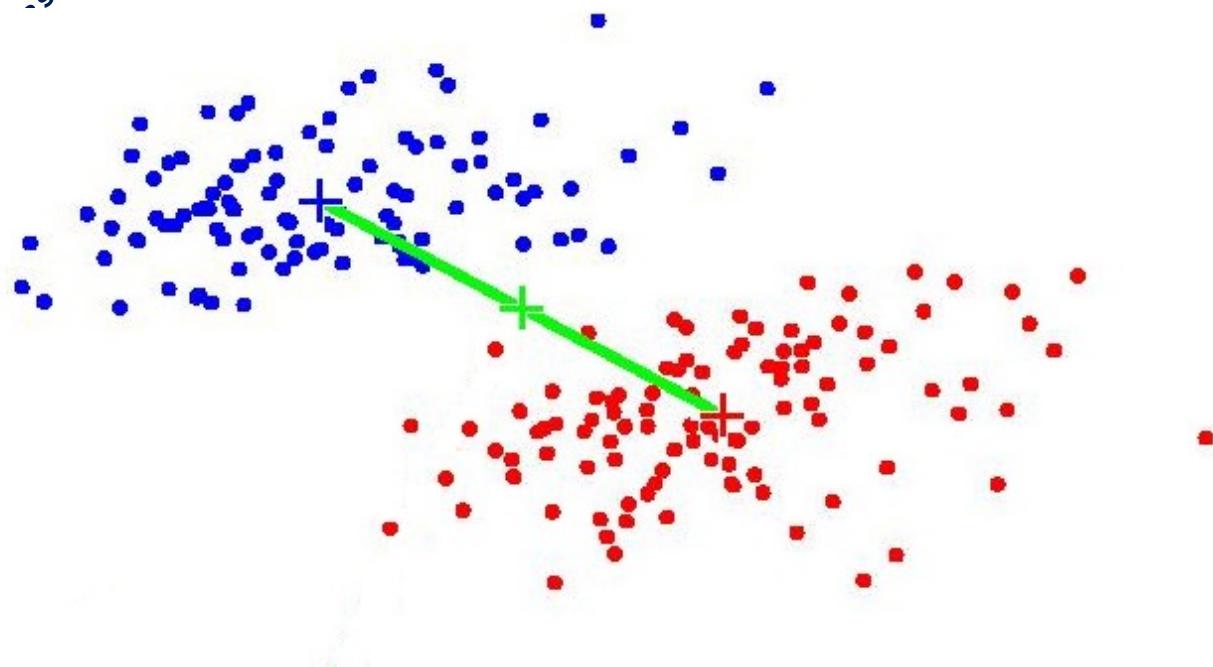
Let  $X$  be a  $n \times d$  *data matrix* of  $n$  samples



# Limit of representation-based dim reduction

- The dim reduction approaches are concerned with faithful representation of the data
- But for classification, we are interested in discrimination.
- The directions that are discarded by PCA might be exactly the ones needed for distinguishing classes.
  - Considering letters O and Q, PCA might discover the gross features that characterize the letters but ignore the tail that distinguishes them.
  - Discriminant analysis seeks directions that are efficient for discrimination.

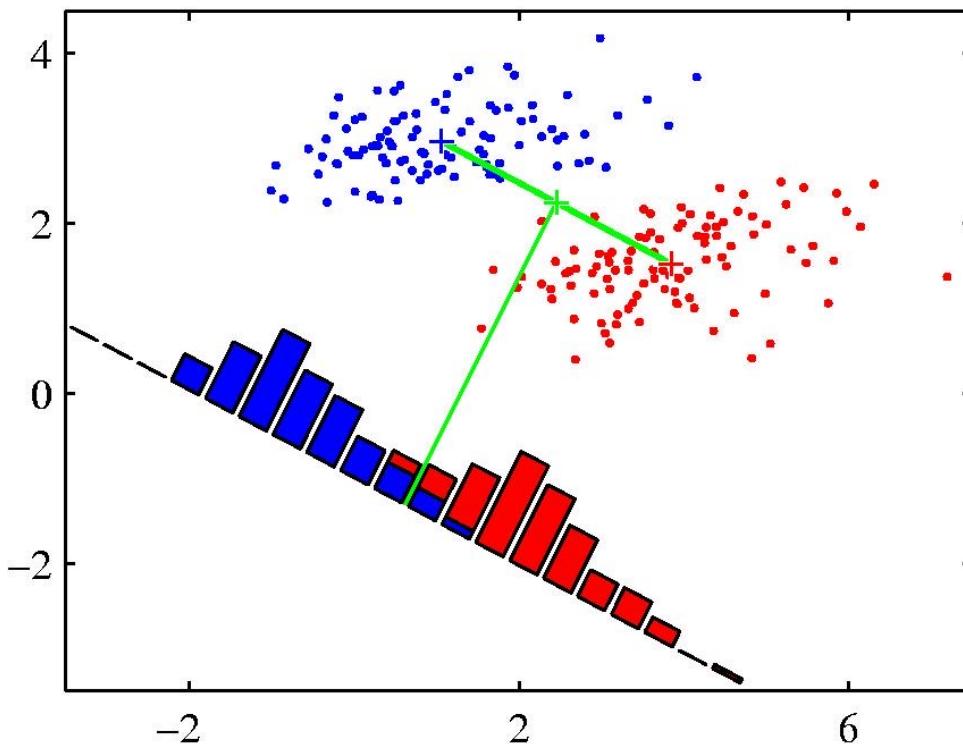
# Projection for dimensionality reduction



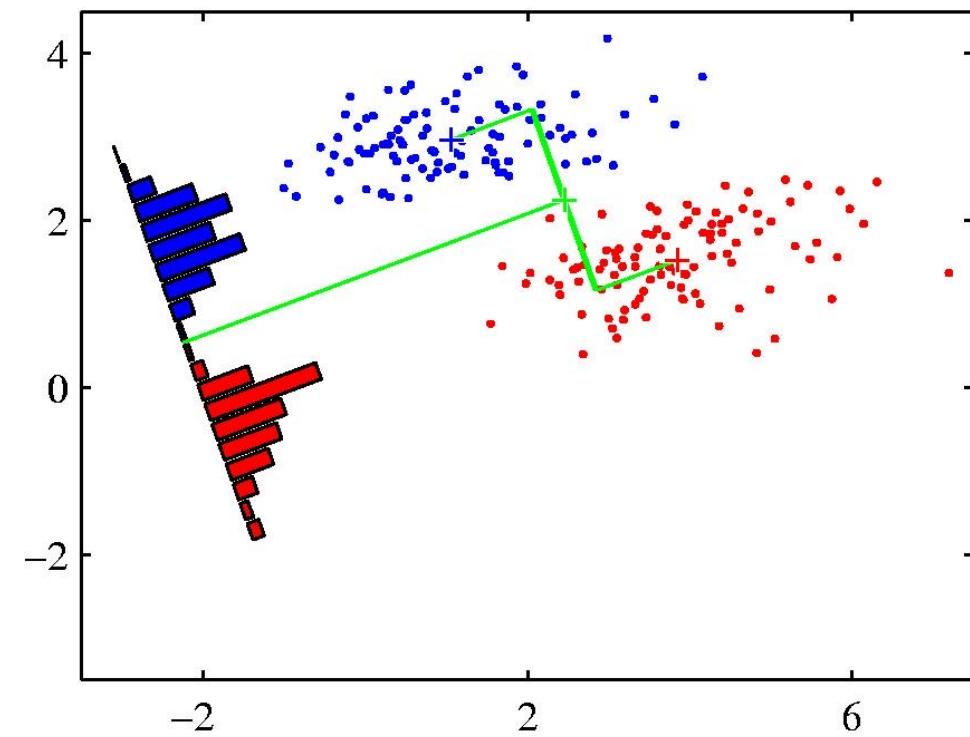
- How to project it into one dimension?
- Representation vs. discrimination
- Supervised and unsupervised

# Two different projections

view a linear classification model in terms of dimensionality reduction - Fisher's linear discriminant



Projection onto the line joining  
the class means



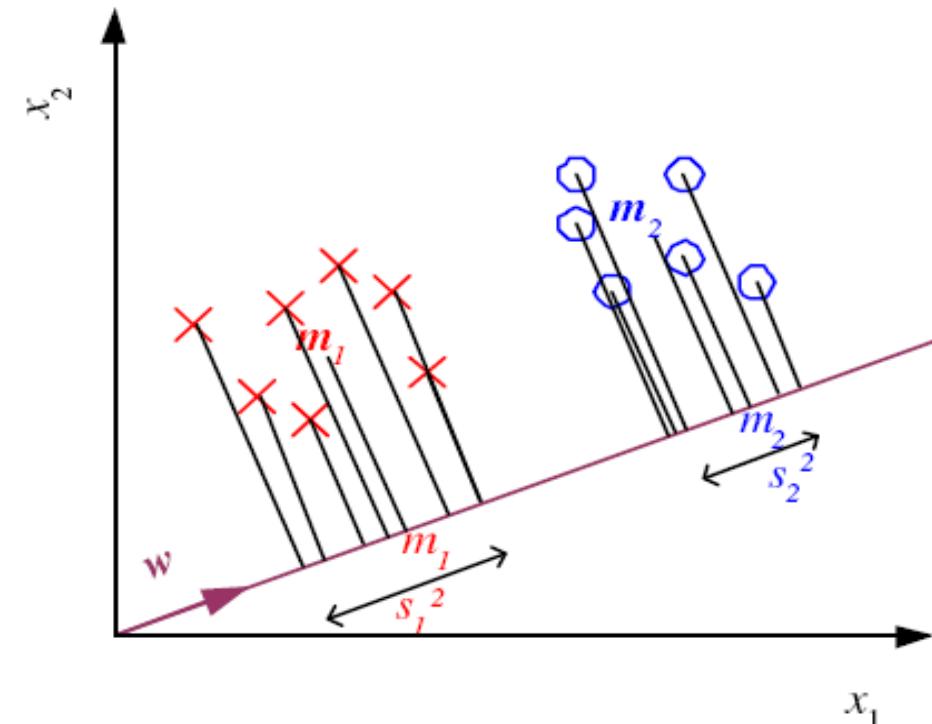
Projection based on the Fisher  
linear discriminant

# Linear discriminant analysis

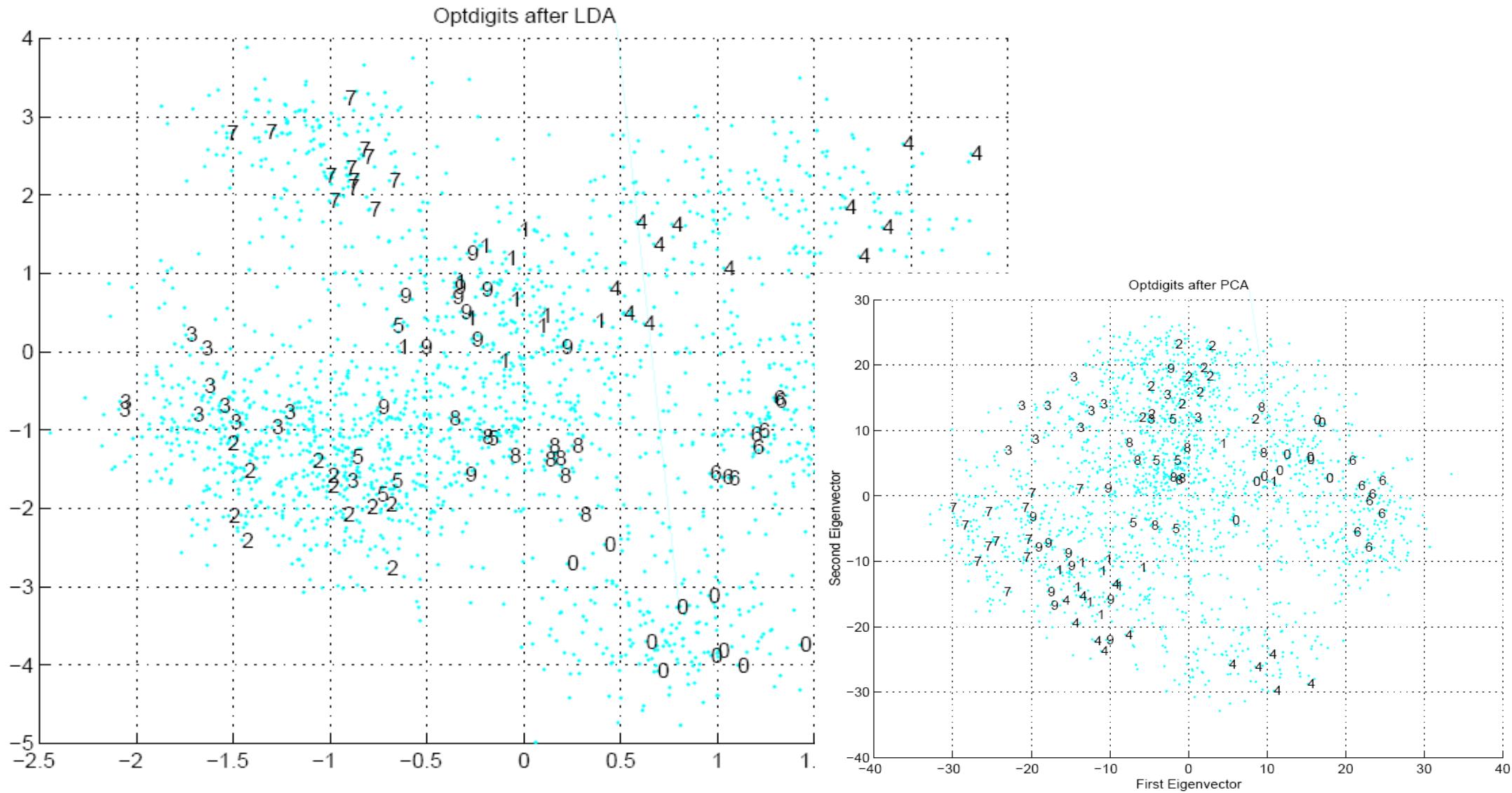
- Find a low-dimensional space such that when  $\mathbf{x}$  is projected, classes are well-separated.
- Find  $\mathbf{w}$  that maximizes the objective function  $J$

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$m_1 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t r^t}{\sum_t r^t} \quad s_1^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t$$



# Linear discriminant analysis



# Eigenfaces (upper) vs fisherfaces (lower)

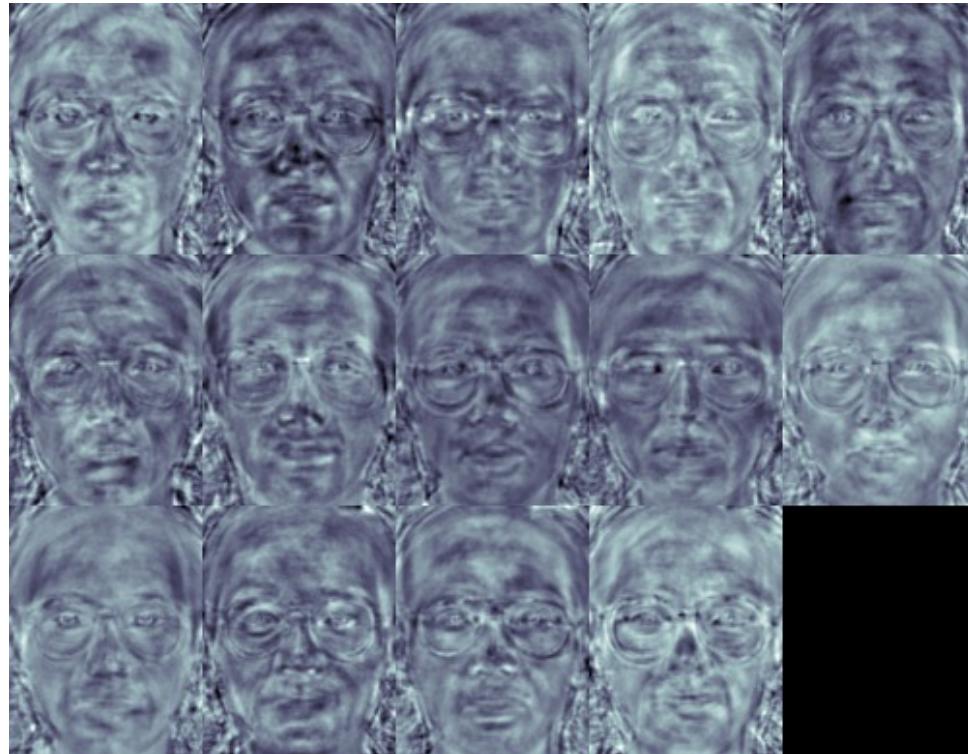
AT&T database



<http://www.lemonpolka.com/wp-content/uploads/2012/05/facial-recognition-system.pdf>

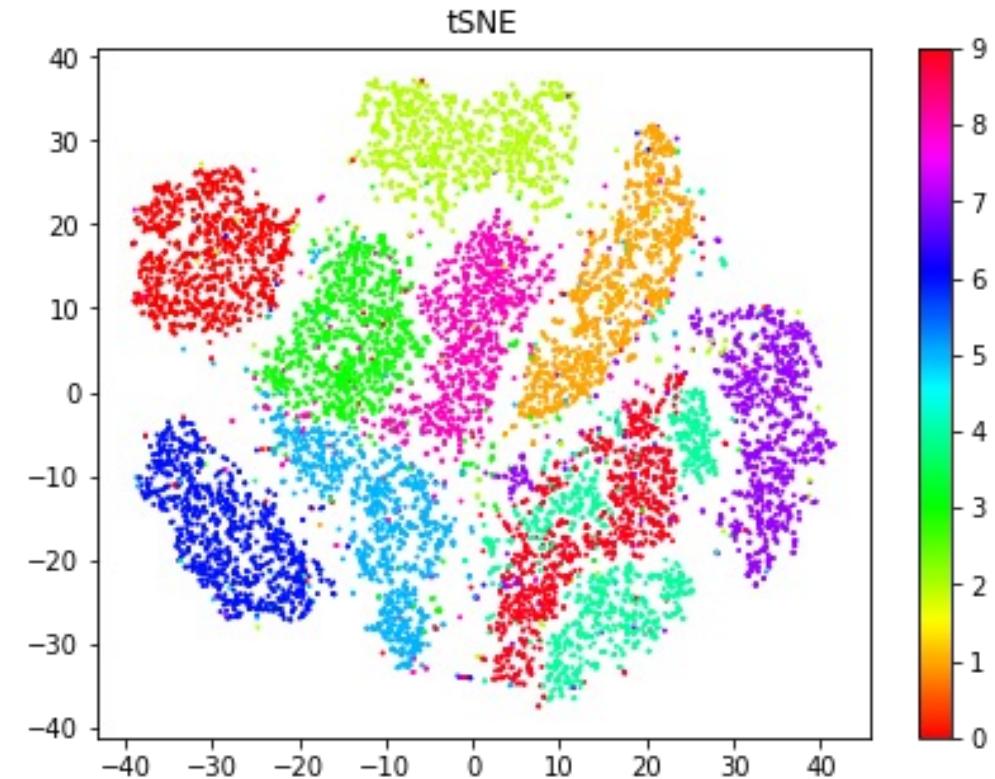
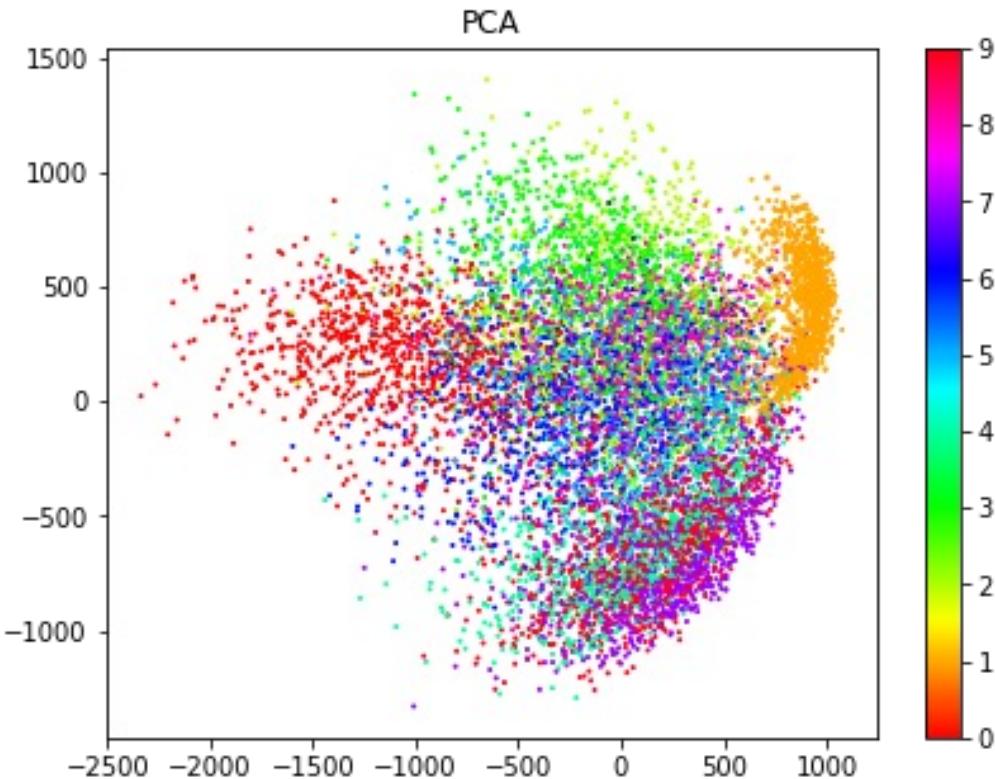
# Fisherfaces and reconstruction

Yale Facedatabase A



[http://docs.opencv.org/modules/contrib/doc/facerec/facerec\\_tutorial.html](http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html)

# t-distributed Stochastic Neighbor Embedding (tSNE)



- Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov), 2579-2605.

# Lecture outline

- The curse of dimensionality
- Feature selection
- Feature extraction
  - Principal component analysis
  - Linear discriminant analysis

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 5: Clustering

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Unsupervised learning

Goals - discover the unknown structure of the inputs

- Model data density (Lecture 3)
  - Use: Facilitate other learning
- Reduce dimensionality (Lecture 4)
  - Use: Facilitate other learning, compress data
- **Find clusters** (Lecture 5)
  - Use: Compress data, detect outliers

# Clustering

- The process of organizing data into groups/clusters whose members are similar in some way.
- **Clustering** is the **most important unsupervised learning problem**, aimed at finding a structure in a collection of unlabeled data.
  - Reference vectors
  - Mixture models
- Vector quantization is the modeling of probability density functions through prototype vectors.
  - K-means

# K-means clustering

- Find  $k$  reference vectors (prototypes/codebook vectors/codewords) which best represent data
- Reference vectors,  $\mathbf{m}_j, j = 1, \dots, k$
- Use nearest (most similar) reference:
$$\|\mathbf{x}^t - \mathbf{m}_i\| = \min_j \|\mathbf{x}^t - \mathbf{m}_j\|$$
- Reconstruction error

$$E(\{\mathbf{m}_i\}_{i=1}^k | \mathcal{X}) = \sum_t \sum_i b_i^t \|\mathbf{x}^t - \mathbf{m}_i\|$$

$$b_i^t = \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\| = \min_j \|\mathbf{x}^t - \mathbf{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

# K-means clustering

Initialize  $\mathbf{m}_i, i = 1, \dots, k$ , for example, to  $k$  random  $\mathbf{x}^t$

Repeat

For all  $\mathbf{x}^t \in \mathcal{X}$

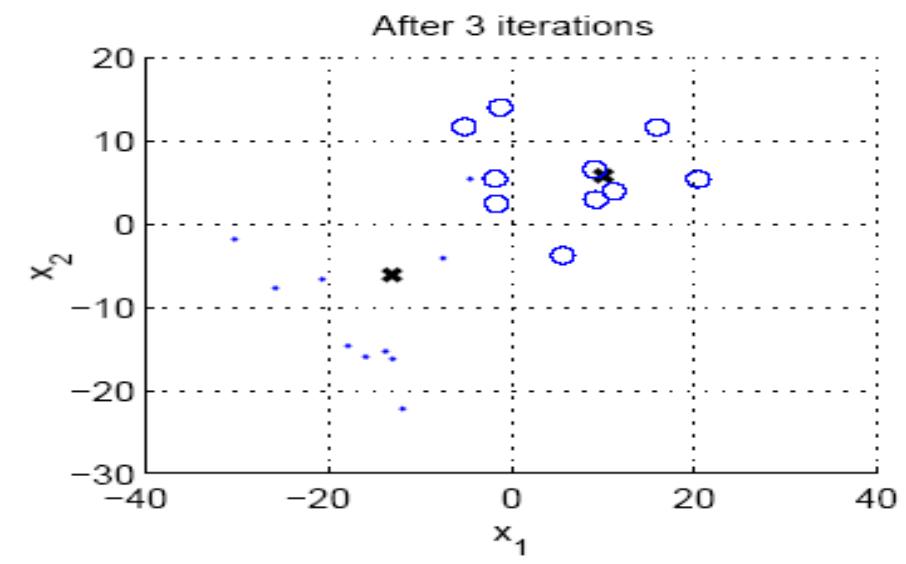
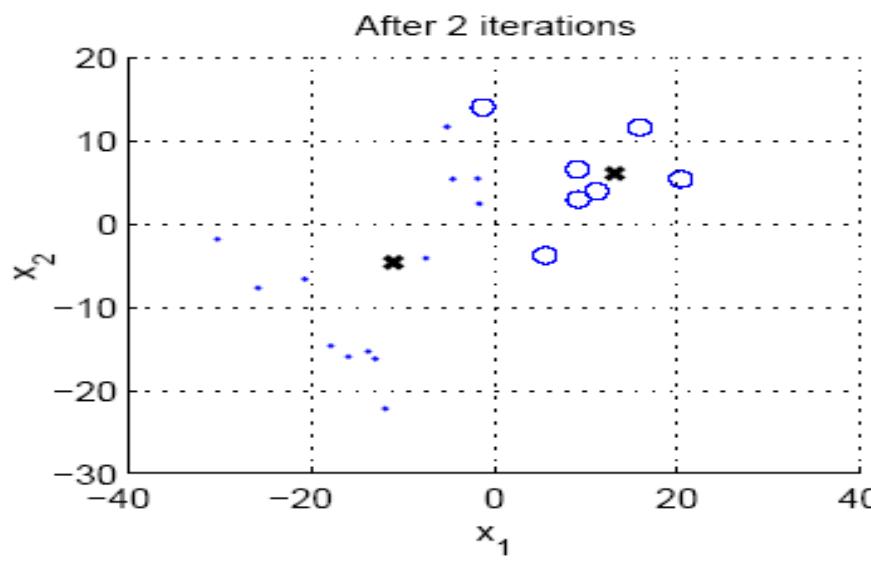
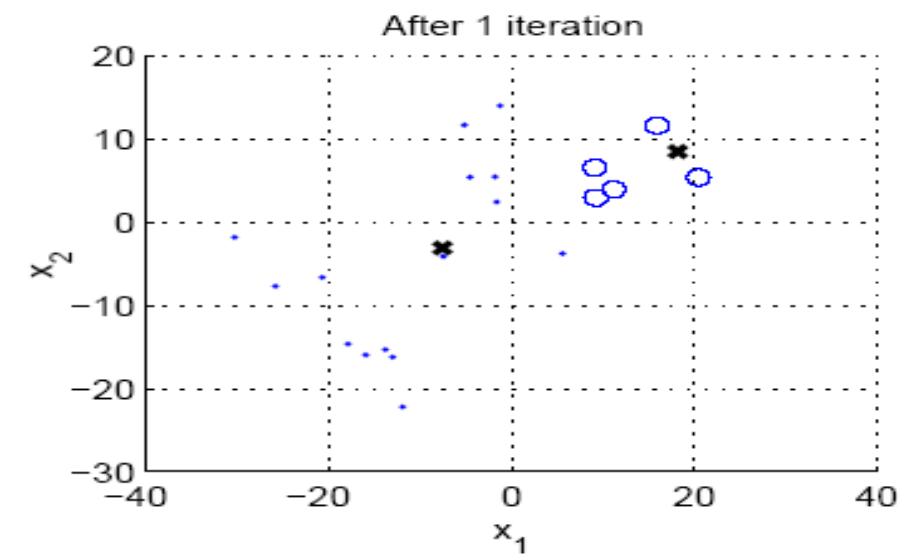
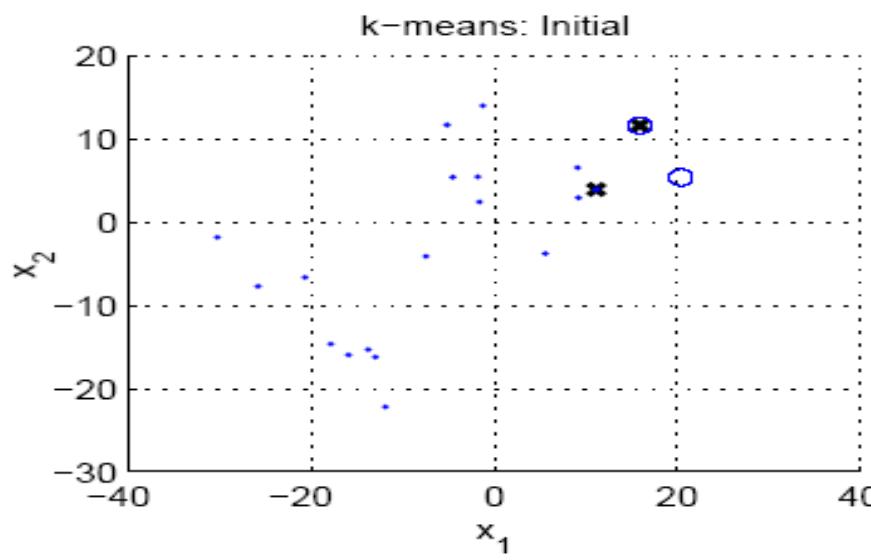
$$b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\| = \min_j \|\mathbf{x}^t - \mathbf{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

For all  $\mathbf{m}_i, i = 1, \dots, k$

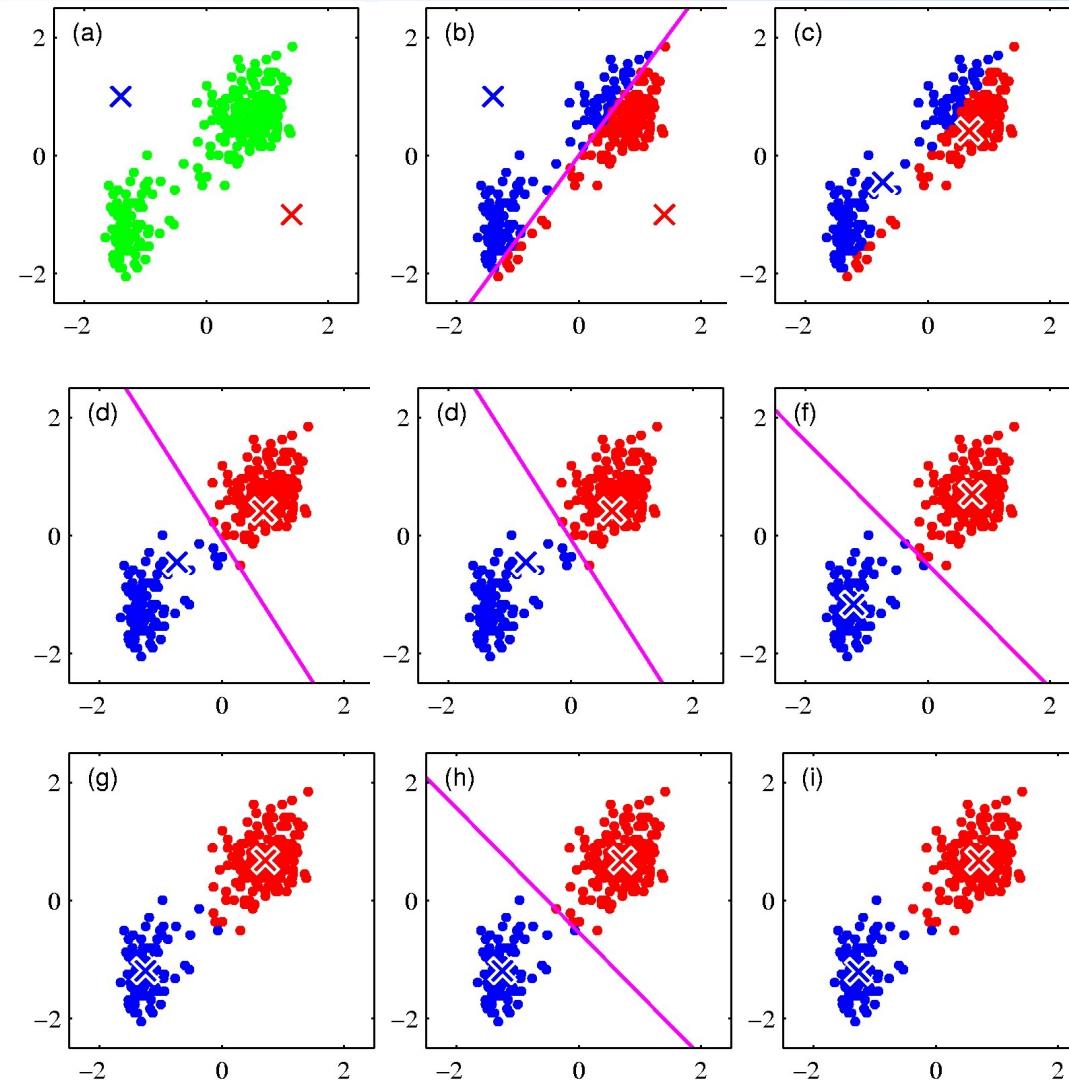
$$\mathbf{m}_i \leftarrow \sum_t b_i^t \mathbf{x}^t / \sum_t b_i^t$$

Until  $\mathbf{m}_i$  converge

# K-means clustering

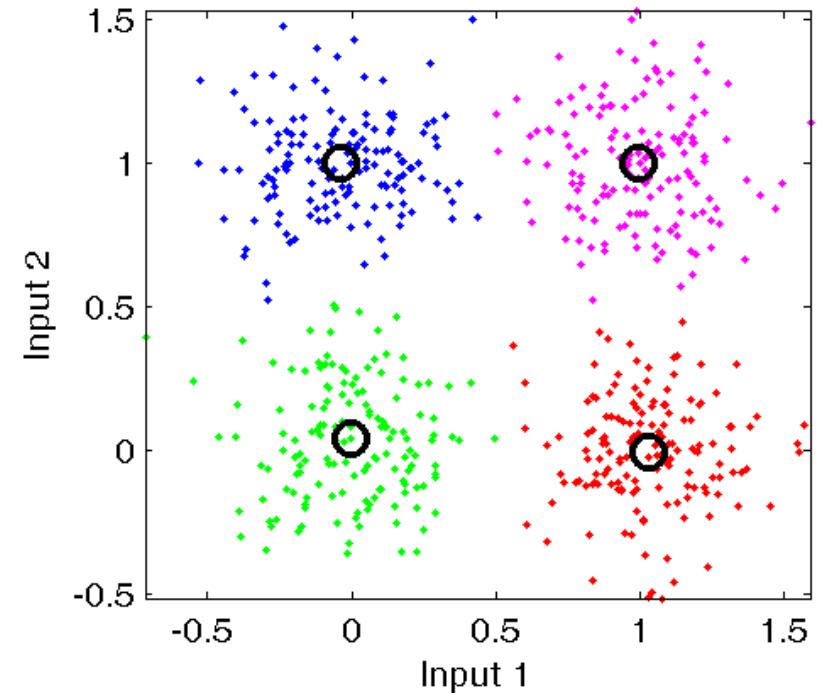


# K-means clustering - cont.

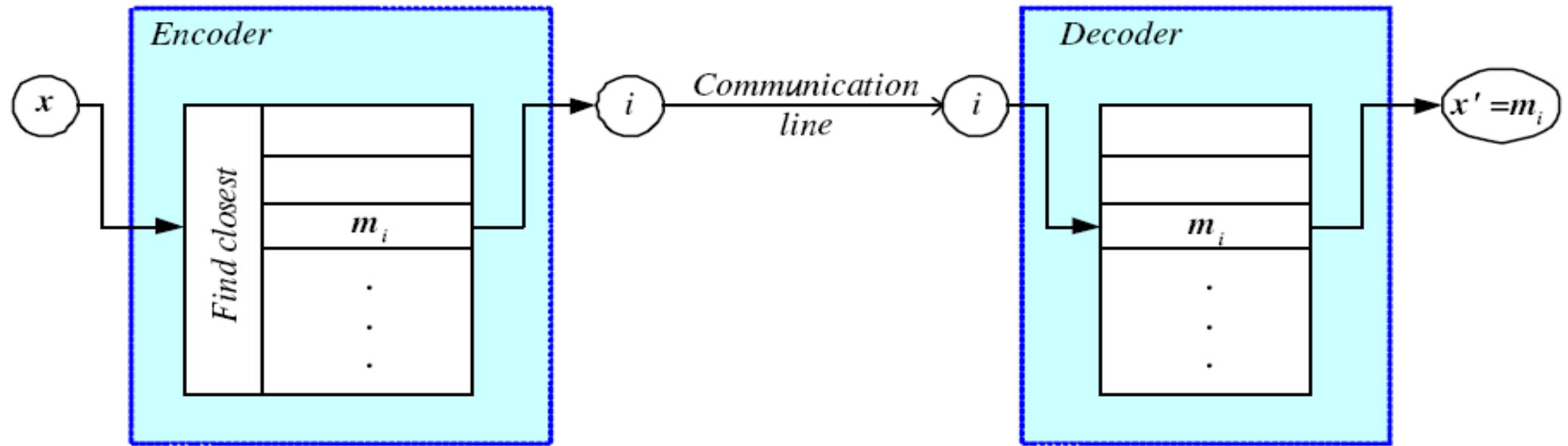


# K-means clustering

- Application 1: Discovering hidden categories.
- Application 2: Lossy data compression - to code a vector, find the prototype,  $M_k$ , that is closest to it, and transmit  $k$ . This process is called **Vector Quantization**.

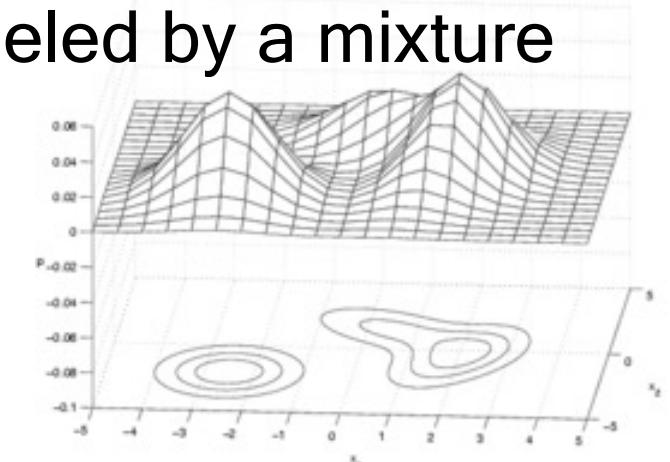


# VQ encoding and decoding



# From prototypes to model based clustering

- In K-means, clustering is to find reference vectors that minimize total reconstruction errors.
- Instead, we can **use certain models for clusters** and optimize the fit between the data and the models.
  - Each cluster is mathematically represented by a parametric distribution, like a Gaussian.
  - The entire data set is therefore modeled by a mixture of these distributions.
  - An individual distribution used to model a specific cluster is often referred to as a component distrib.



# Mixture Densities

Probabilistic view; finding the component density parameters that maximize the likelihood. Mixture densities (e.g. GMM):

$$p(\mathbf{x}) = \sum_{i=1}^k p(\mathbf{x} | G_i) P(G_i)$$

where  $G_i$  the components/groups/clusters,

$P(G_i)$  mixture proportions (priors),

$p(\mathbf{x}|G_i)$  component densities

Gaussian mixture where  $(\mathbf{x}|G_i) \sim N(\boldsymbol{\mu}_i, \Sigma_i)$  parameters  $\Phi = \{P(G_i), \boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^k$

Based on unlabeled sample  $\{\mathbf{x}^t\}_{t=1}^N$  (unsupervised learning)

# Classes vs. clusters

- **Supervised:**  $\mathcal{X} = \{\mathbf{x}^t, r^t\}_t$
- Classes  $C_i, i=1, \dots, K$

$$p(\mathbf{x}) = \sum_{i=1}^K p(\mathbf{x} | C_i) P(C_i)$$

where  $(\mathbf{x} | C_i) \sim N(\boldsymbol{\mu}_i, \Sigma_i)$

- $\boldsymbol{\theta} = \{P(C_i), \boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^K$

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \quad \mathbf{m}_i = \frac{\sum_t r_i^t \mathbf{x}^t}{\sum_t r_i^t}$$

$$\mathbf{S}_i = \frac{\sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T}{\sum_t r_i^t}$$

- **Unsupervised :**  $\mathcal{X} = \{\mathbf{x}^t\}_t$
- Clusters (Groups)  $G_i, i=1, \dots, k$

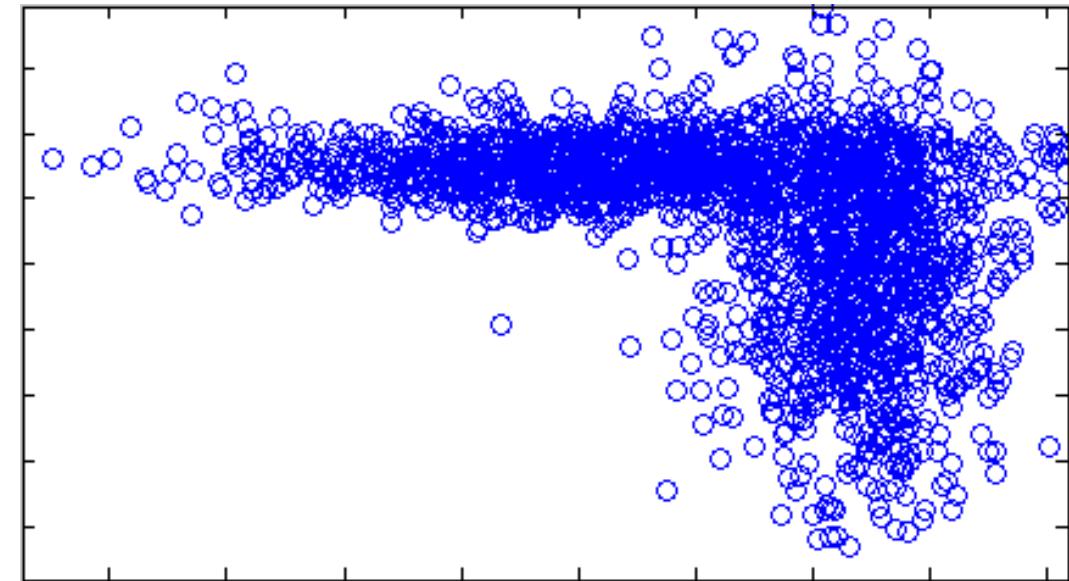
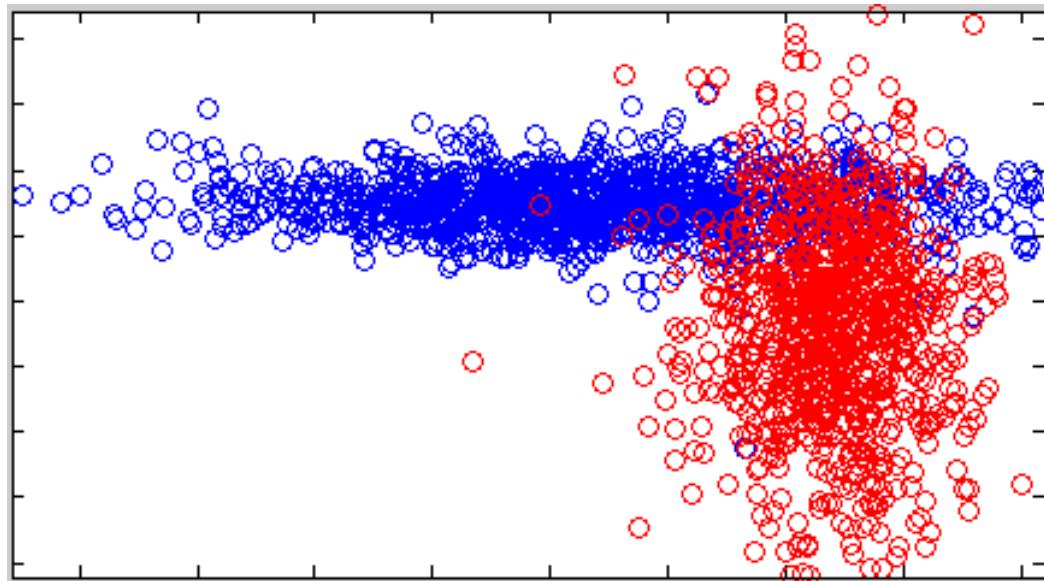
$$p(\mathbf{x}) = \sum_{i=1}^k p(\mathbf{x} | G_i) P(G_i)$$

where  $(\mathbf{x} | G_i) \sim N(\boldsymbol{\mu}_i, \Sigma_i)$

- $\Phi = \{P(G_i), \boldsymbol{\mu}_i, \Sigma_i\}_{i=1}^k$

- How to estimate  $\Phi$ ?
- Labels,  $r^t_i$  ?

# Supervised vs unsupervised



# How to estimate parameters $\Phi$ ?

- Find component density parameters that **maximize the likelihood**.
- Log likelihood with a mixture model

$$\mathcal{L}(\Phi | \mathcal{X}) = \ln \prod_t p(\mathbf{x}^t | \Phi)$$

Parameters  $\Phi$ : priors and sufficient statistics of component densities.

$$= \sum_t \ln \sum_{i=1}^k p(\mathbf{x}^t | G_i) P(G_i)$$

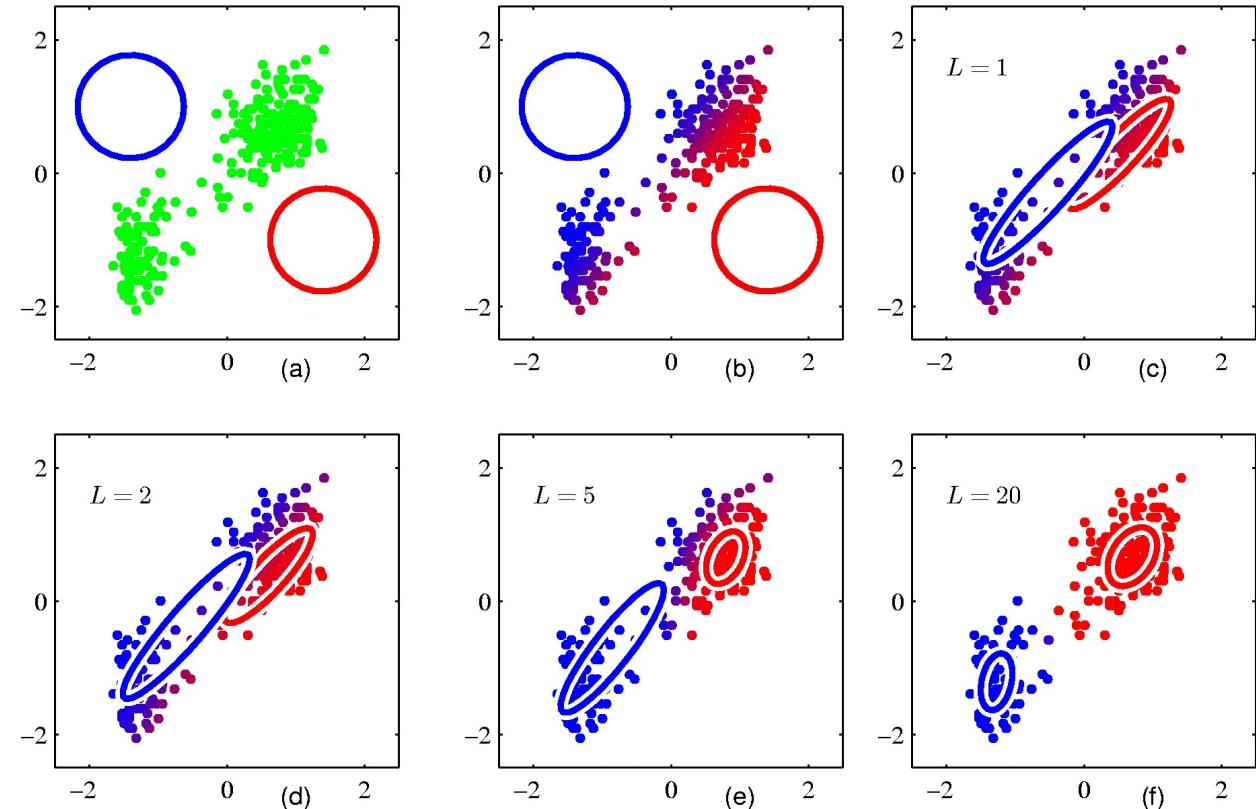
If  $k=1, : )$

- No analytical solution -> iterative optimization
- Expectation-maximization (EM) algorithm for maximum likelihood estimation

# Example - fitting a mixture of Gaussians

After initialization, EM algorithm alternates between two steps:

- **E-step:** Compute the posterior probability that each Gaussian generates each data point -> which Gaussian generated which point.
- **M-step:** Assuming that the data really was generated this way, change the parameters of each Gaussian to maximize the probability that it would generate the data it is currently responsible for.



# The general EM algorithm: E- and M-steps

- Given a joint distribution  $p(\mathcal{X}, \mathcal{Z}|\Phi)$  over observed variables  $\mathcal{X}$  and **latent variables**  $\mathcal{Z}$ : e.g.  $z_i^t = 1$  if  $x^t$  belongs to  $G_i$ , 0 otherwise (labels  $r_i^t$  of supervised learning), governed by parameters  $\Phi$ , the goal is to maximize the likelihood  $p(\mathcal{X}|\Phi)$  wrt  $\Phi$ .
- Initialize  $\Phi$ , and iterate the two steps
  - E-step: Estimate  $\mathcal{Z}$  given  $\mathcal{X}$  and current  $\Phi$
  - M-step: Find new  $\Phi^{l+1}$  given  $\mathcal{Z}$ ,  $\mathcal{X}$ , and old  $\Phi$ .

$$\Phi^{l+1} = \arg \max_{\Phi} Q(\Phi | \Phi^l)$$

where  $Q(\Phi | \Phi^l) = E[\mathcal{L}_C(\Phi | \mathcal{X}, \mathcal{Z}) | \mathcal{X}, \Phi^l]$ , i.e. the expectation  
 An increase in  $Q$  increases incomplete likelihood (likelihood of data):  $\mathcal{L}(\Phi^{l+1} | \mathcal{X}) \geq \mathcal{L}(\Phi^l | \mathcal{X})$

- 
- The simple k-means clustering is a special case of the EM algorithm.

# EM in Gaussian Mixtures

- Assume  $(\mathbf{x}|G_i) \sim N(\boldsymbol{\mu}_i, \Sigma_i)$
- E-step: the expected value of the hidden variable is the posterior probability that  $\mathbf{x}^t$  is generated by  $G_i$

$$E[z_i^t | \mathcal{X}, \Phi^l] = \frac{p(\mathbf{x}^t | G_i, \Phi^l) P(G_i)}{\sum_j p(\mathbf{x}^t | G_j, \Phi^l) P(G_j)}$$

$$= P(G_i | \mathbf{x}^t, \Phi^l) \equiv h_i^t$$

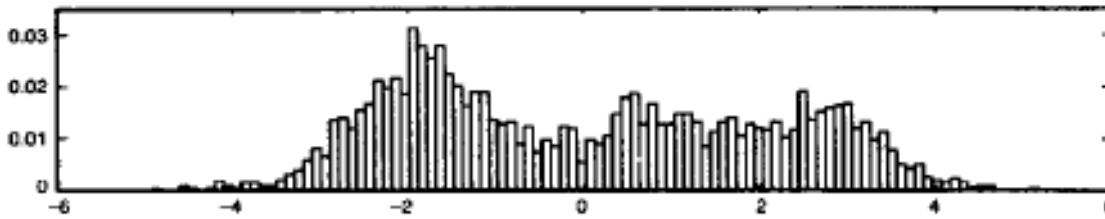
- M-step:

$$P(G_i) = \frac{\sum_t h_i^t}{N} \quad \mathbf{m}_i^{l+1} = \frac{\sum_t h_i^t \mathbf{x}^t}{\sum_t h_i^t}$$

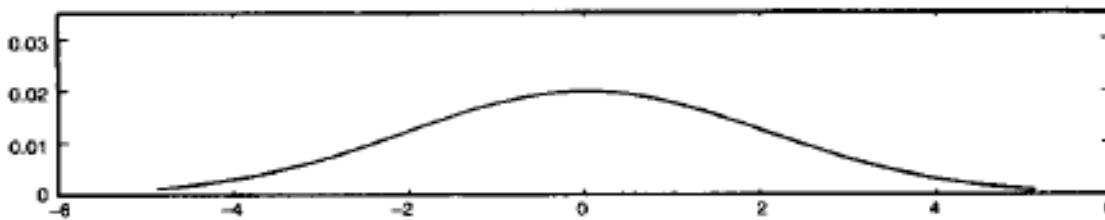
$$\mathbf{S}_i^{l+1} = \frac{\sum_t h_i^t (\mathbf{x}^t - \mathbf{m}_i^{l+1})(\mathbf{x}^t - \mathbf{m}_i^{l+1})^T}{\sum_t h_i^t}$$

Use estimated labels  
in place of unknown  
labels

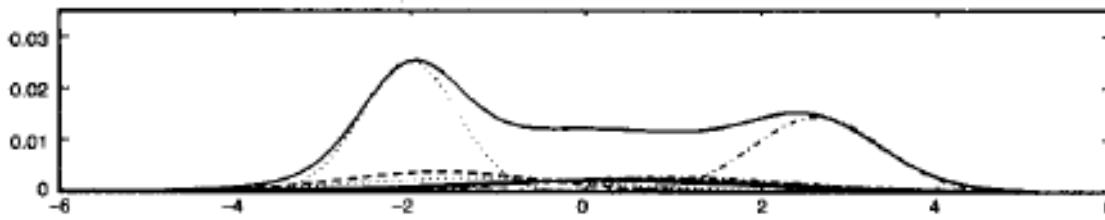
# Comparison of distribution modelling



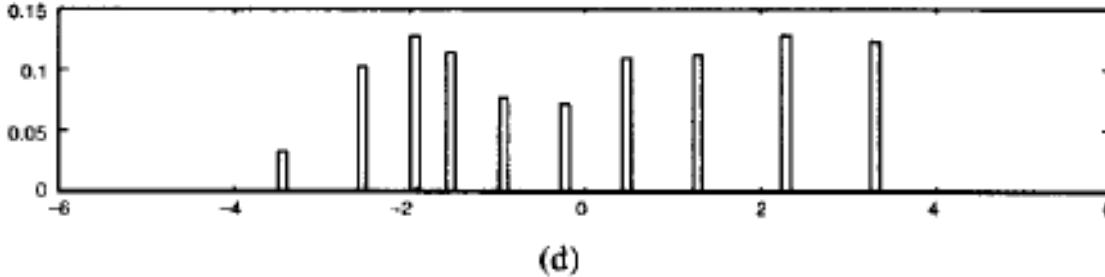
(a)



(b)



(c)



(d)

Reynolds & Rose, 1994

# Clustering applications

- Compared with dimensionality reduction
  - Dimensionality reduction methods find correlations between features and group features
  - Clustering methods find similarities between instances and group instances
- Allows knowledge extraction through
  - number of clusters,
  - prior probabilities,
  - cluster parameters, i.e., center, range of features.
  - preprocessing for classification and regression

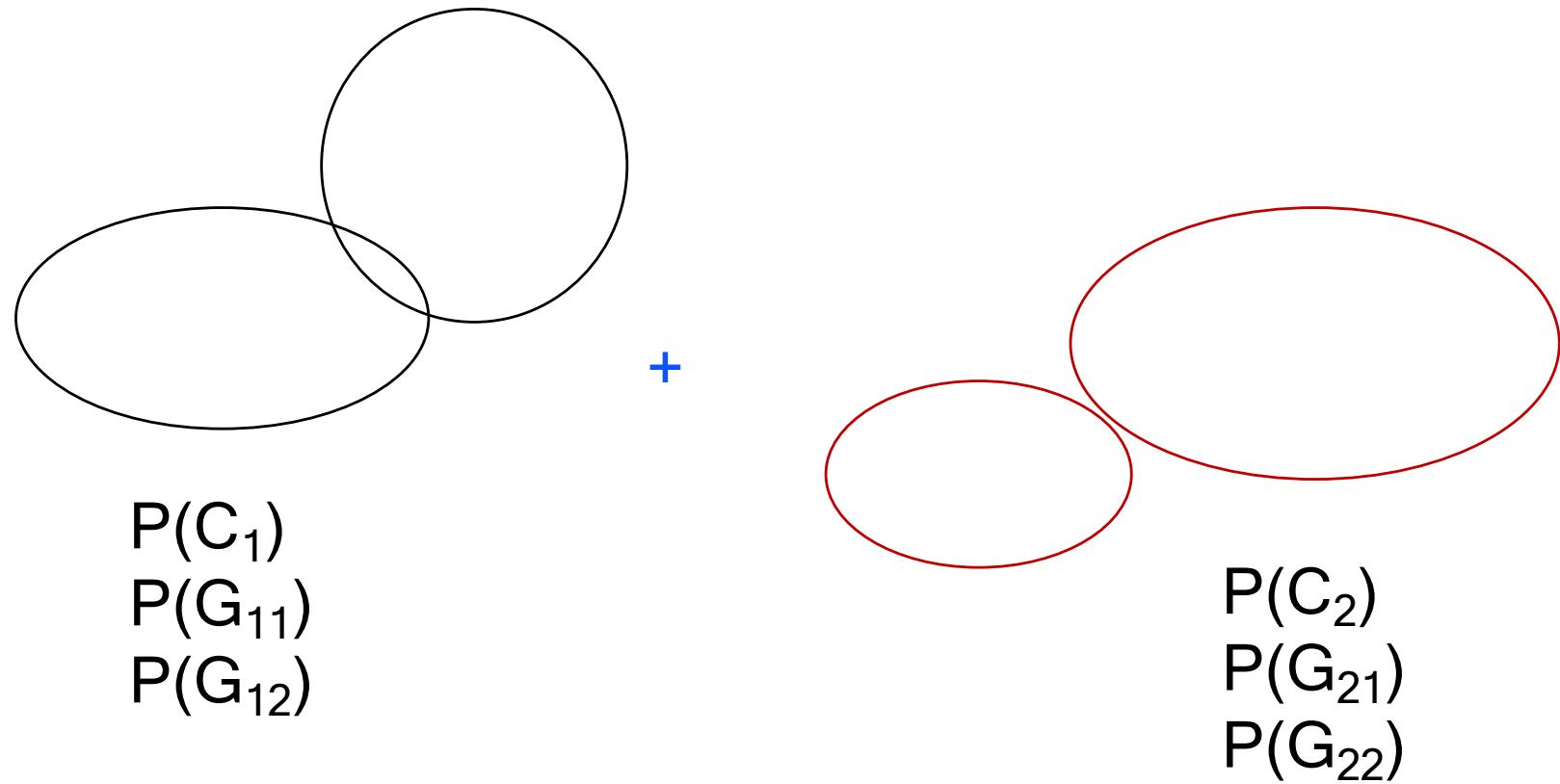
# Mixture of Mixtures

- In classification, the input comes from a mixture of classes (supervised).
- If each class is also a mixture, e.g., of Gaussians, (unsupervised), we have a mixture of mixtures:

$$p(\mathbf{x} | C_i) = \sum_{j=1}^{k_i} p(\mathbf{x} | G_{ij}) P(G_{ij})$$

$$p(\mathbf{x}) = \sum_{i=1}^K p(\mathbf{x} | C_i) P(C_i)$$

# Mixture of Mixtures - cont.



# Choosing the number of clusters - k

- Defined by the application, e.g., image quantization
- Plot data (after PCA) and check for clusters
- Incremental algorithm: Add one at a time until “elbow” (reconstruction error/log likelihood/intergroup distances)
- Manual check for meaning

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. **Linear discrimination**
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 6: Linear Classification Methods

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. **Linear discrimination**
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

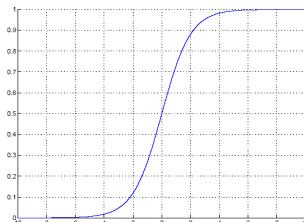
# Lecture outline

## Linear classification methods

- Probabilistic generative models
  - A revisit
- Probabilistic discriminative models
- Linear discriminant functions and learning approaches

# Probabilistic generative models

- Use a separate generative model for each class, and see which model makes test data most probable, based on Bayes' rule.
- In 2-class case, the posterior probability of class 1 is given by:



$$p(C_1 | \mathbf{x}) = \frac{p(C_1)p(\mathbf{x} | C_1)}{p(C_1)p(\mathbf{x} | C_1) + p(C_0)p(\mathbf{x} | C_0)} = \boxed{\frac{1}{1 + e^{-z}}} = \sigma(z)$$

Sigmoid function

where  $z = \ln \frac{p(C_1)p(\mathbf{x} | C_1)}{p(C_0)p(\mathbf{x} | C_0)} = \ln \frac{p(C_1 | \mathbf{x})}{1 - p(C_1 | \mathbf{x})} = \boxed{\ln \frac{\sigma}{1 - \sigma}}$

$z(\mathbf{x})$  may be linear function of  $\mathbf{x}$ :  $p(C_1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0)$

Consequently,  $p(C_i | \mathbf{x})$  is governed by a **generalized linear model**.

$z$  is called the **logit**

# A simple example

- Assume that the input vectors for each class are from a **Gaussian distribution**, and all classes have the **same covariance matrix**.

$$p(\mathbf{x} | C_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right\}$$

- For two-class case, we can show that the posterior is a logistic:

$$p(C_1 | \mathbf{x}) = \sigma\left(\ln \frac{p(C_1)p(\mathbf{x} | C_1)}{p(C_0)p(\mathbf{x} | C_0)}\right) = \boxed{\sigma(\mathbf{w}^T \mathbf{x} + w_0)}$$

$$\mathbf{w} = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$$

$$w_0 = -\frac{1}{2} \boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_0^T \Sigma^{-1} \boldsymbol{\mu}_0 + \ln \frac{p(C_1)}{p(C_0)}$$

# K>2 classes

A multi-class generalization of the logistic sigmoid, known as normalized exponential or softmax function.

$$\begin{aligned} P(C_i | \mathbf{x}) &= \frac{p(\mathbf{x} | C_i)p(C_i)}{\sum_{j=1}^K p(\mathbf{x} | C_j)p(C_j)} \\ &= \boxed{\frac{\exp(a_i)}{\sum_{j=1}^K \exp(a_j)}}, i = 1, \dots, K \end{aligned}$$

softmax

$$a_i = \ln p(\mathbf{x} | C_i)p(C_i)$$

# Lecture outline

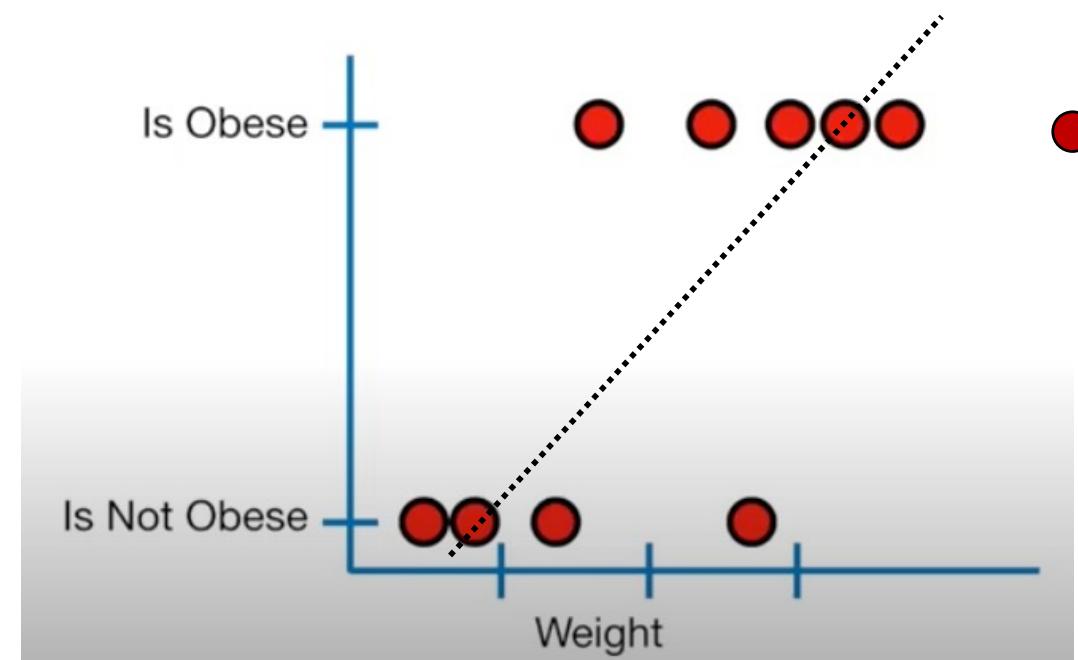
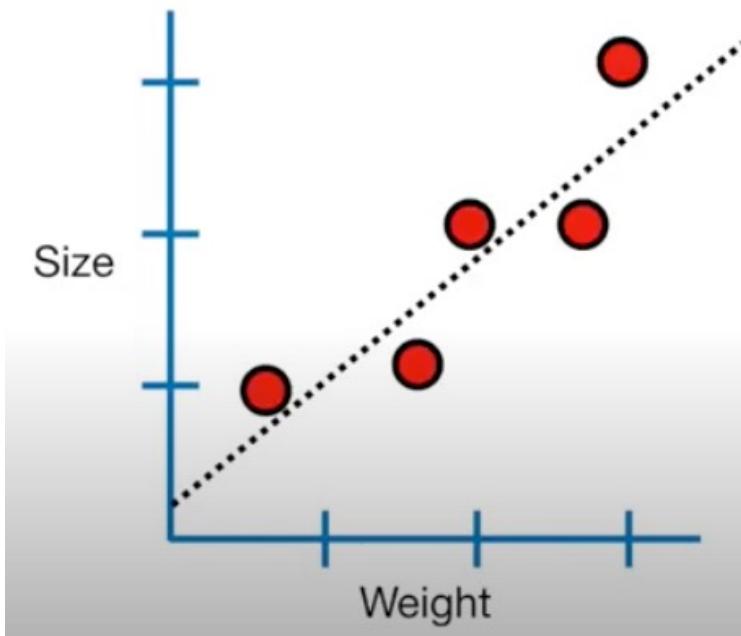
## Linear classification methods

- Probabilistic generative models
- Probabilistic discriminative models
  - Logistic regression
- Linear discriminant functions and learning approaches

# Logistic regression

An example of linear regression vs logistic regression: mouse

- regression: (weight, size)
- classification: (weight, obese or not)



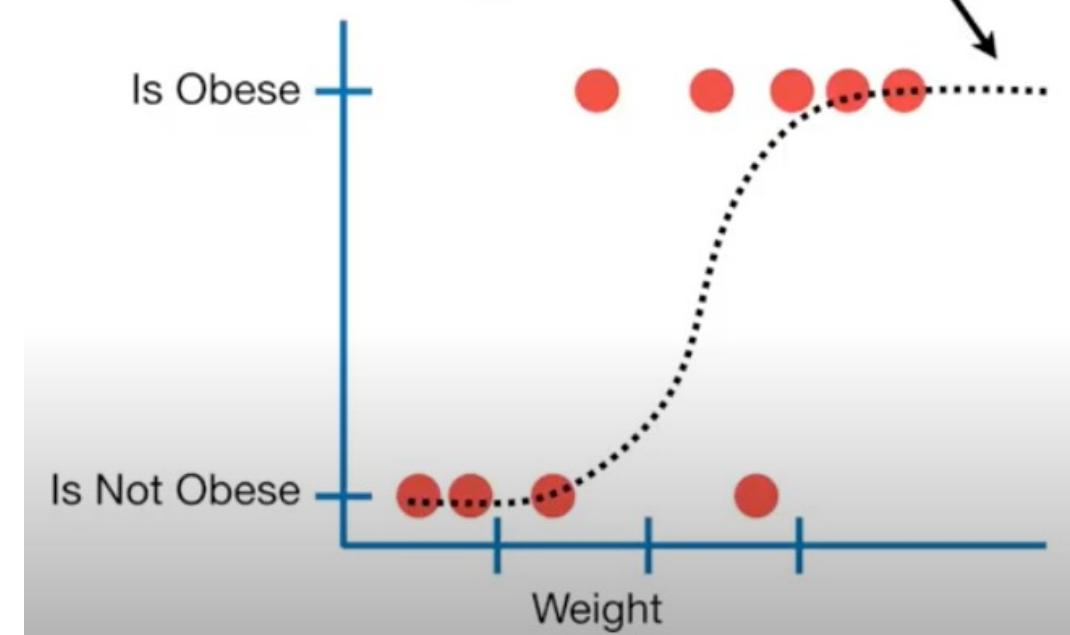
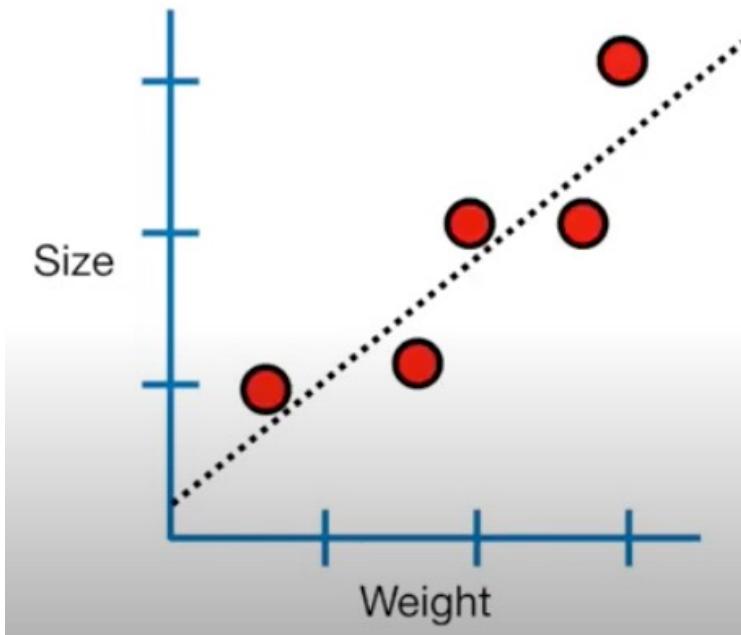
(StatQuest)

# Logistic regression – cont'd

An example of linear regression vs logistic regression: mouse

- regression: (weight, size)
- classification: (weight, obese or not)

...also, instead of fitting a line to the data, logistic regression fits an "S" shaped "logistic function".



(StatQuest)

# Logistic regression – cont'd

- Linear regression
  - Function:  $h(x_n) = y(x_n, \mathbf{w}) = w_1 x_n + w_0$
  - Loss function: SE / MSE
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$
- Logistic regression (LR)
  - Function:  $h(x_n) = \sigma(w_1 x_n + w_0) = 1/(1+e^{-(w_1 x_n + w_0)})$
  - By using maximum likelihood to determine the parameters of LR, we will derive its loss function: cross entropy error

# Logistic regression – cont'd

- Model the posterior probability directly using the logistic regression method
- As shown in probabilistic generative models, when the distributions underlying the data are Gaussians with a common covariance matrix, the posterior probability for  $C_1$

$$p(C_1 | \mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + w_0) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + w_0)}}$$

- In LR, we adopt this model irrespective of the data distribution.
- It is a model for classification rather than regression.

# Logistic regression – cont'd

- The likelihood function of the set of training samples  $(t_n, \mathbf{x}_n), n = 1, \dots N, t_n \in [0,1]$ :

$$p(t_1 \dots t_N; \mathbf{w}, w_0) \\ = \prod_{n=1}^N (\sigma(\mathbf{w}^T \mathbf{x} + w_0))^{t_n} (1 - \sigma(\mathbf{w}^T \mathbf{x} + w_0))^{1-t_n}$$

- Consider the negative log-likelihood (convex) as cost function

$$L(\mathbf{w}, w_0) = - \sum_{n=1}^N (t_n \ln \sigma(\mathbf{w}^T \mathbf{x} + w_0) + (1 - t_n) \ln(1 - \sigma(\mathbf{w}^T \mathbf{x} + w_0)))$$

- This cost function is also known as the cross-entropy error.  
(Entropy:  $H(X) = - \sum_{n=1}^N p(x_n) \log p(x_n)$ )

# Logistic regression – cont'd

- Minimize  $L(\mathbf{w}, w_0)$  with respect to  $\mathbf{w}$ ,  $w_0$  is carried out iteratively by an iterative minimization scheme such as gradient descent
- Take the gradient of the error function with respect to  $\mathbf{w}$  and  $w_0$ , we have

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{n=1}^N (y_n - t_n) \mathbf{x}_n \quad \text{and} \quad \frac{\partial L}{\partial w_0} = \sum_{n=1}^N (y_n - t_n)$$

- After finding the gradients, we subtract the gradients with the original  $\mathbf{w}$  and  $w_0$ .

# Lecture outline

## Linear classification methods

- Probabilistic generative models
- Probabilistic discriminative models
- Linear discriminant functions and learning approaches
  - Least square
  - LDA
  - Perceptron (gradient descent)

# Linear discriminant functions

- The decision boundaries are linear functions of the input vector  $\mathbf{x}$  and are therefore defined by  $(D-1)$ -dimensional hyperplanes within the  $D$ -dimensional input space.
- Linear classification means that **the part that adapts is linear**
  - The adaptive part is followed by a **fixed non-linearity** and may also be preceded by a **fixed non-linearity** (e.g. nonlinear basis functions  $\phi(\mathbf{x})$  ).

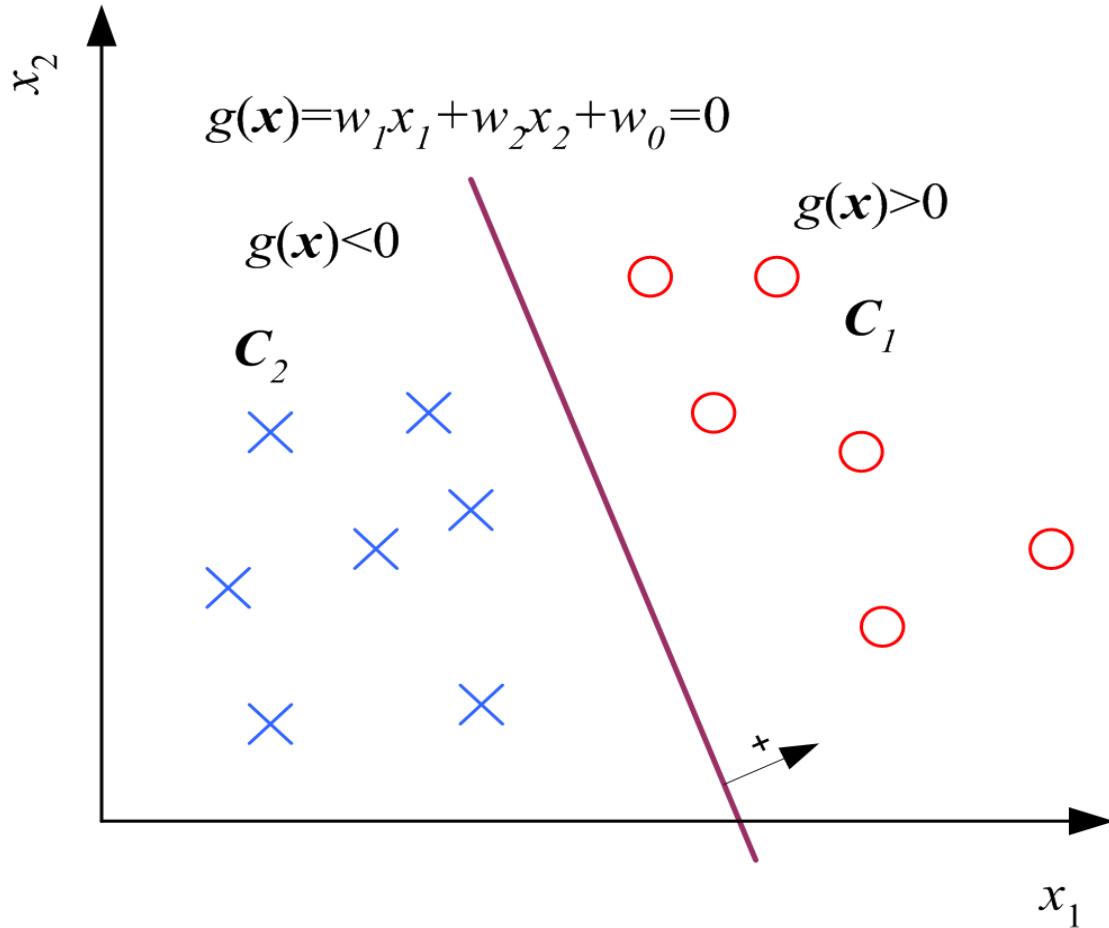
$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0,$$

↑  
adaptive linear  
function

$$Decision = f(g(\mathbf{x}))$$

↑  
fixed non-linear  
activation function

# Two classes



$$\begin{aligned}
 g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) \\
 &= (\mathbf{w}_1^T \mathbf{x} + w_{10}) - (\mathbf{w}_2^T \mathbf{x} + w_{20}) \\
 &= (\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x} + (w_{10} - w_{20}) \\
 &= \mathbf{w}^T \mathbf{x} + w_0
 \end{aligned}$$

choose  $\begin{cases} C_1 & \text{if } g(\mathbf{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$

$f()$

# Three approaches to learning w

**Three approaches** to learning the parameters of linear discriminant functions

- Least-squares
  - each class is described by its own linear model
  - pleasant analytical properties
  - lack of robustness to outliers.
- Fisher's linear discriminant
  - view a linear classification model as dimensionality reduction.
- The perceptron algorithm (gradient descent)
  - Rosenblatt

# Least squares for classification

- Consider a classification problem with  $K$  classes, with a 1-of- $K$  binary coding scheme for the target vector  $t$ .
- Using least squares
- Each class is described by its own linear model so that

$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

a new input  $\mathbf{x}$  is then assigned to the class for which the output  $g_i$  is the largest.

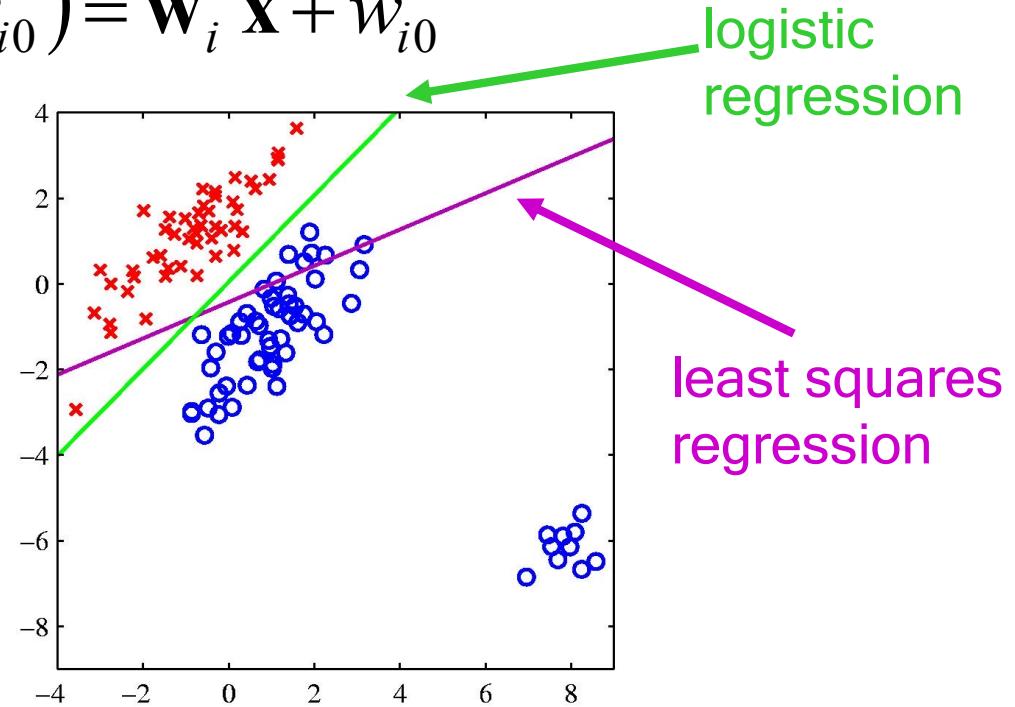
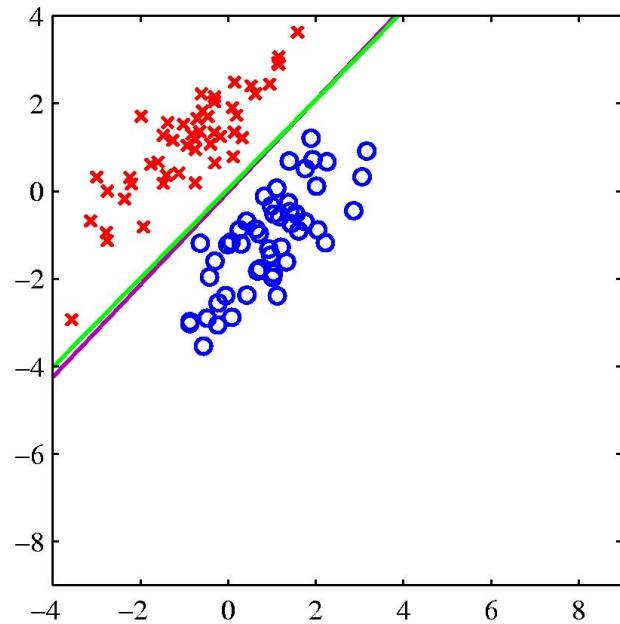
# Least squares for classification

- It is easy:
  - It reduces classification to least squares regression, whose optimal weights can be solved with some matrix algebra.
  - It gives an exact closed-form solution for the discriminant function parameters.
  - When there are more than two classes, we treat each class as a separate problem.
- This is however not the right thing to do and it doesn't work as well as better methods.

# Least squares for classification – cont.

- Least-squares solutions lack robustness to outliers.
- This failure is due to that least-squares corresponds to ML under **the assumption of a Gaussian conditional distribution**.

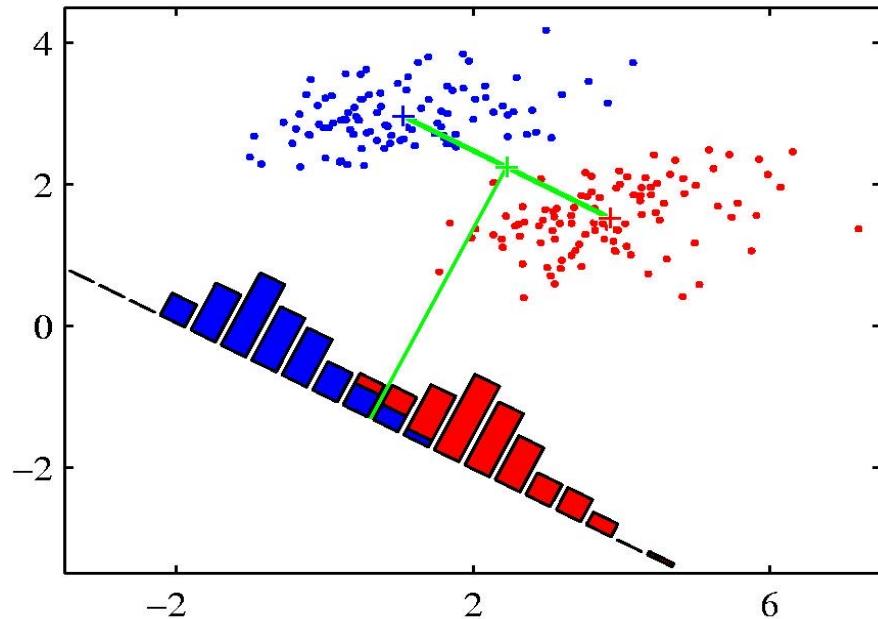
$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$



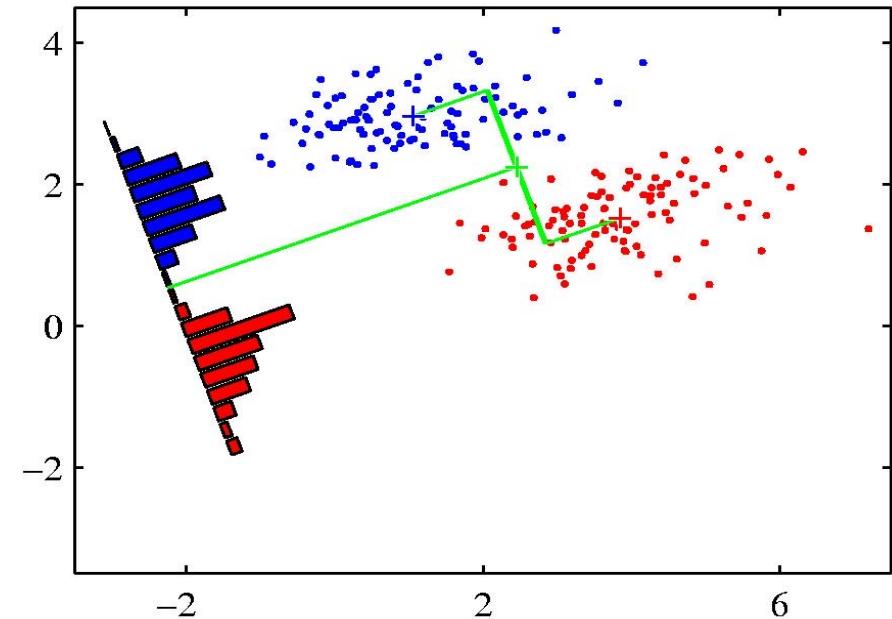
# Fisher's linear discriminant

- One way to view a classification model is in terms of **dimensionality reduction** (projecting the data down to **1-D**).
  - So choose the projection that gives the best separation of the classes.
- One choice: the line joining the class means.
  - But if the main direction of variance in each class is not orthogonal to this line, it won't give good separation.
- Fisher's method chooses the direction that maximizes the ratio of **between class scatter** to **within class scatter**, after the projection.

# Fisher's linear discriminant – cont.



When projected onto the line joining the class means, the classes are not well separated.



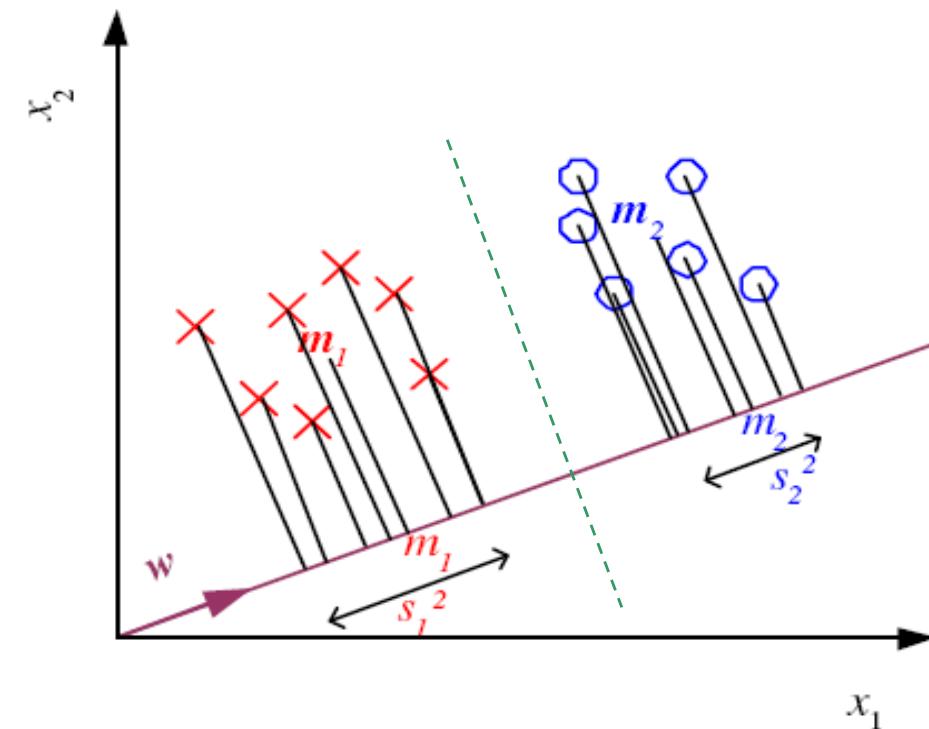
Fisher chooses a direction that makes the projected classes much tighter, even though their projected means are less far apart.

# Fisher's linear discriminant – cont.

- Find a low-dimensional space such that when  $\mathbf{x}$  is projected, classes are well-separated.
- Find  $\mathbf{w}$  that maximizes

$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$m_1 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t r^t}{\sum_t r^t} \quad s_1^2 = \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t$$



# Fisher's linear discriminant – cont.

- Between-class scatter:

$$\begin{aligned}
 (\mathbf{m}_1 - \mathbf{m}_2)^2 &= (\mathbf{w}^T \mathbf{m}_1 - \mathbf{w}^T \mathbf{m}_2)^2 \\
 &= \mathbf{w}^T (\mathbf{m}_1 - \mathbf{m}_2) (\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{w} \\
 &= \mathbf{w}^T \mathbf{S}_B \mathbf{w} \text{ where } \mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2) (\mathbf{m}_1 - \mathbf{m}_2)^T
 \end{aligned}$$

- Within-class scatter:

$$\begin{aligned}
 s_1^2 &= \sum_t (\mathbf{w}^T \mathbf{x}^t - m_1)^2 r^t \\
 &= \sum_t \mathbf{w}^T (\mathbf{x}^t - \mathbf{m}_1) (\mathbf{x}^t - \mathbf{m}_1)^T \mathbf{w} r^t = \mathbf{w}^T \mathbf{S}_1 \mathbf{w}
 \end{aligned}$$

$$\text{where } \mathbf{S}_1 = \sum_t (\mathbf{x}^t - \mathbf{m}_1) (\mathbf{x}^t - \mathbf{m}_1)^T r^t$$

$$s_1^2 + s_2^2 = \mathbf{w}^T \mathbf{S}_W \mathbf{w} \text{ where } \mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$$

# Fisher' s linear discriminant – cont.

- Find  $w$  that max

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

- Differentiate  $J(w)$  with respect to  $w$  and set to zero:

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

- LDA solution:

$$\underline{S_W^{-1} S_B w = \lambda w}$$

Choose the eigenvector correspoding to largest eigenvalue.

The maximal number of directions/eigenvectors is N\_class-1.

# Fisher's linear discriminant – cont.

- LDA solution:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}$$

OR since  $\mathbf{S}_B \mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)(m_1 - m_2)$

i.e.,  $\mathbf{S}_B \mathbf{w}$  is always in the direction of  $\mathbf{m}_1 - \mathbf{m}_2$ ,

we can simply set  $\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$  and normalize it.

# Fisher's linear discriminant

- LDA chooses the projection that gives the best separation of the classes.
  - What does “best separation” mean?
- Joining the class means does not lead to a good solution as *only means* are considered.
- Fisher’s method chooses the direction that maximizes the ratio of **between class scatter** (a function of *projected means*) to **within class scatter**.
  - Fisher criterion maximizes class separation in the output space. (vs. SVM is a classifier in the original space.)
  - *More commonly used for dim reduction before classification than for classification*

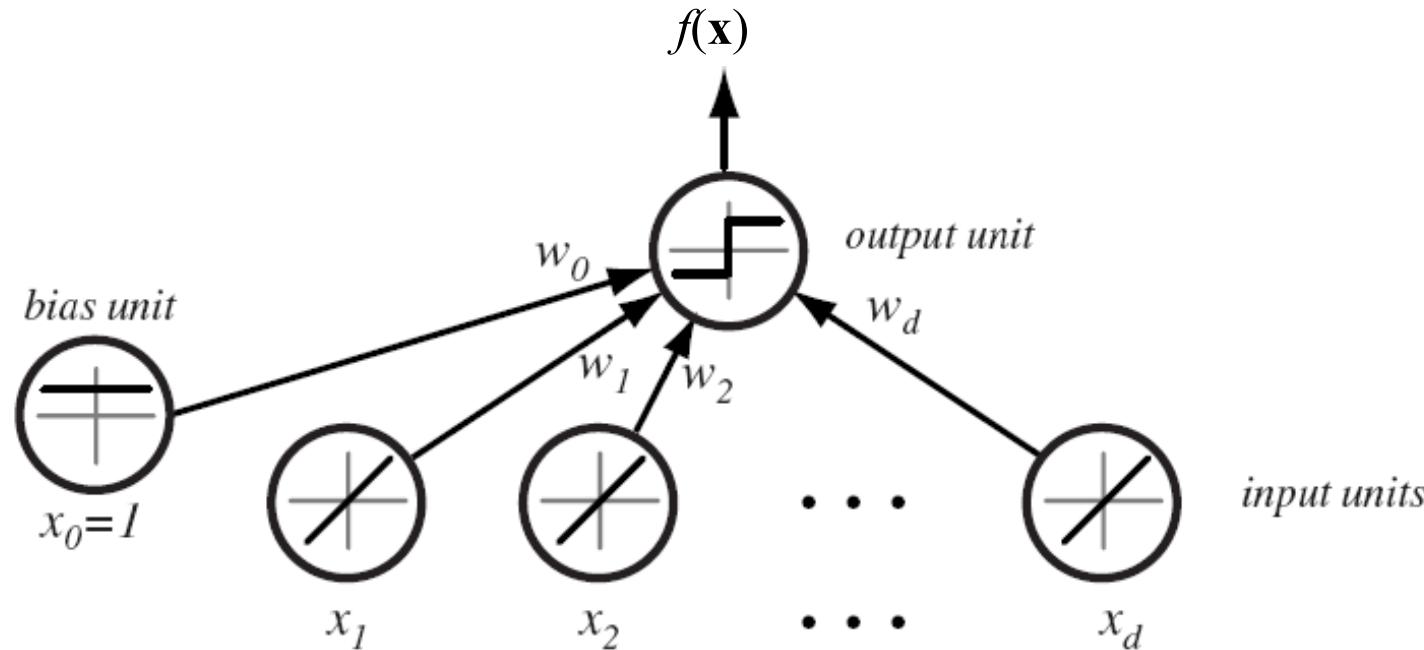
# The perceptron algorithm

- A linear discriminant model introduced by Rosenblatt (1962).
  - Simplest feedforward neural network: a linear classifier.
  - Grand claims about their abilities led to lots of controversy, e.g. researchers in symbolic AI emphasized their limitations.
- It corresponds to a two-class model in which the input vector  $\mathbf{x}$  is first transformed using a fixed nonlinear transformation to give a feature vector  $\phi(\mathbf{x})$ , and this is then used to construct a generalized linear model of the form  $y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$

Nonlinear sign activation function  $f(\cdot)$ : 
$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

If a later stage needs the posterior prob., use sigmoid function.

# A simple perceptron



**FIGURE 5.1.** A simple linear classifier having  $d$  input units, each corresponding to the values of the components of an input vector. Each input feature value  $x_i$  is multiplied by its corresponding weight  $w_i$ ; the effective input at the output unit is the sum all these products,  $\sum w_i x_i$ . We show in each unit its effective input-output function. Thus each of the  $d$  input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit always emits the constant value 1.0. The single output unit emits a +1 if  $\mathbf{w}^t \mathbf{x} + w_0 > 0$  or a -1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons,

# Parameter estimation: gradient descent

- Probability based approach – recap
  - The parameters are the sufficient statistics of  $p(x|C_i)$  and  $p(C_i)$ .
  - The method to estimate the parameters is ML.
- Discriminant-based approach
  - The parameters are those of the discriminants, optimized to minimize the classification error on the training set.

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w} | X)$$

- Often no analytical solution<sup>w</sup> and so resort to iterative optimization methods: most commonly **gradient descent**.

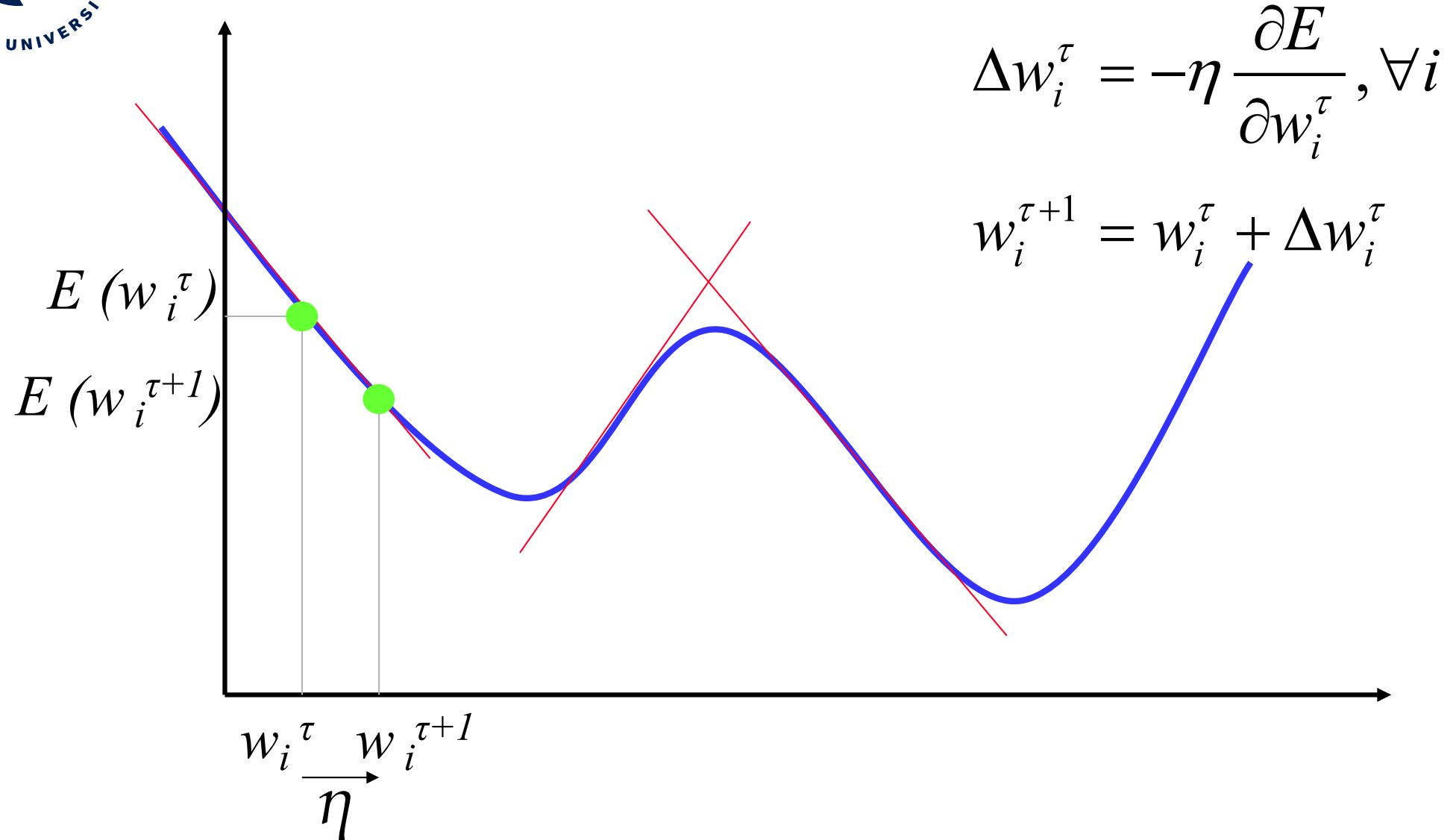
# Gradient descent of error function

- Objective function  $J(\mathbf{w}) = E(\mathbf{w} | \mathcal{X})$  is error with parameters  $\mathbf{w}$  on sample  $\mathcal{X}$

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w} | \mathcal{X})$$

- Gradient  $\nabla_{\mathbf{w}} E = \left[ \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_d} \right]^T$
- Gradient descent:
  - Starts from random  $\mathbf{w}$  and updates  $\mathbf{w}$  iteratively in the negative direction of gradient.
  - Do it for each  $w_i$  separately.

# Gradient descent of error function



# Learning $w$ of the perceptron

Choices of error function and their minimization

- The total number of misclassified patterns
  - a piecewise constant function of  $w$ , which does not lead to a simple learning algorithm to be able to change  $w$  using the gradient of the error function.
- Perceptron criterion – an alternative error function being differentiable

Using the  $\underline{t \in \{-1,+1\}}$  target coding scheme, all patterns should satisfy  $\mathbf{w}^T \phi(\mathbf{x}_n) t_n > 0$ . So perceptron criterion is

$$E_P(\mathbf{w}) = - \sum_{n \in M} \mathbf{w}^T \phi_n t_n$$

$M$  denotes the set of all misclassified patterns.

# Gradient descent algorithm for perceptron criterion

- Error function

$$E_P(\mathbf{w}) = - \sum_{n \in M} \mathbf{w}^T \phi_n t_n$$

- Apply the gradient descent algorithm to this error function. The change in the weight vector  $\mathbf{w}$  is then given by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \phi_n t_n$$

# The perceptron convergence procedure

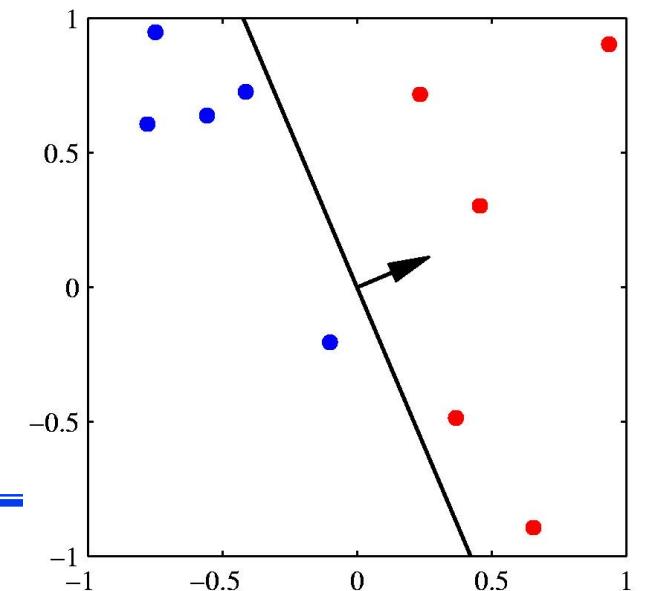
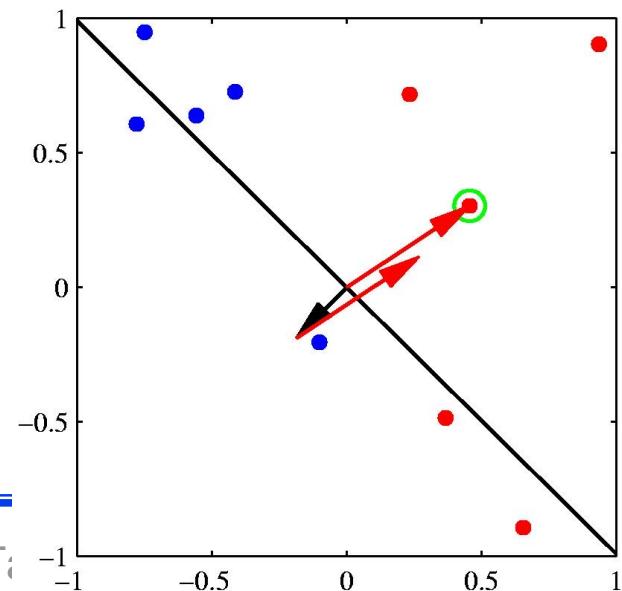
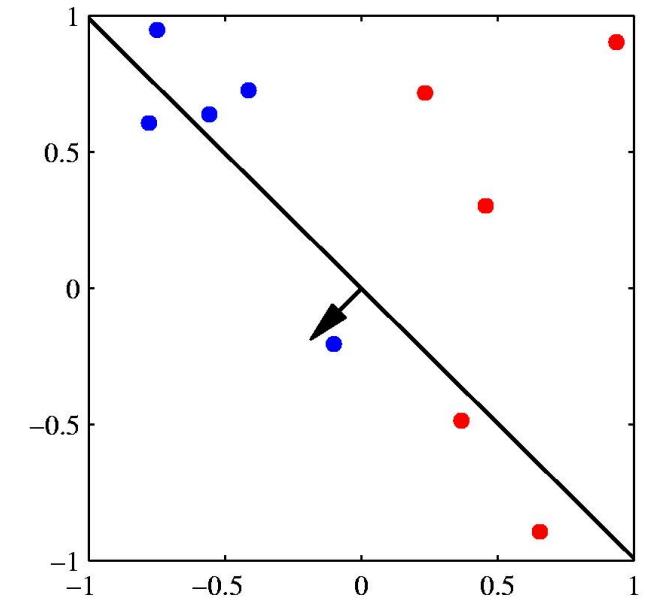
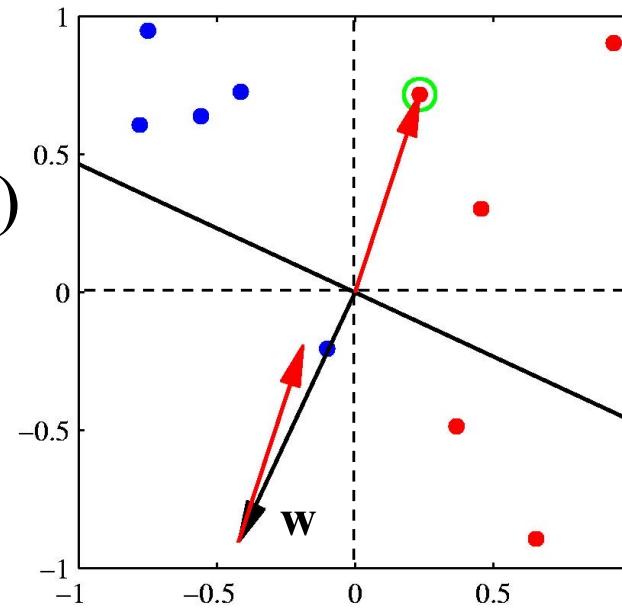
- Add an extra component (=1) to each feature vector. The weight on this “bias” component (unit) is minus the threshold. Then we can forget the threshold.
- Pick training instances (online training)
  - If the pattern is correctly classified, weight vector remains unchanged.
  - If the output is -1 but should be 1, add the vector  $\Phi(\mathbf{x}_n)$  to the weight vector.
  - If the output is 1 but should be -1, subtract the vector  $\Phi(\mathbf{x}_n)$  from the weight vector
- **Theorem:** If the classes are linearly separable, then the Perceptron learning procedure will converge to a solution in a finite number of steps. And no need to choose a learning rate.

# Illustration of the convergence

$$\begin{aligned}\mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) \\ &= \mathbf{w}^{(\tau)} + \eta \phi_n(\mathbf{x}) t_n\end{aligned}$$

$\mathbf{w}$  points towards the red-class region.

Data point circled in green is misclassified.

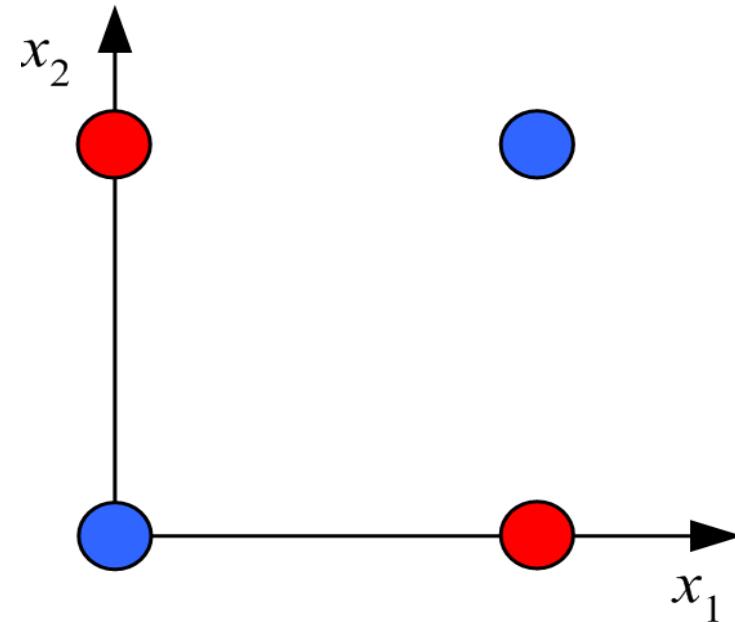


# Perceptrons cannot learn XOR

$x_1$	$x_2$	$r$
0	0	0
0	1	1
1	0	1
1	1	0

- No  $w_0, w_1, w_2$  satisfy:

$$\begin{array}{l} w_0 \leq 0 \\ w_2 + w_0 > 0 \\ w_1 + w_0 > 0 \\ w_1 + w_2 + w_0 \leq 0 \end{array}$$

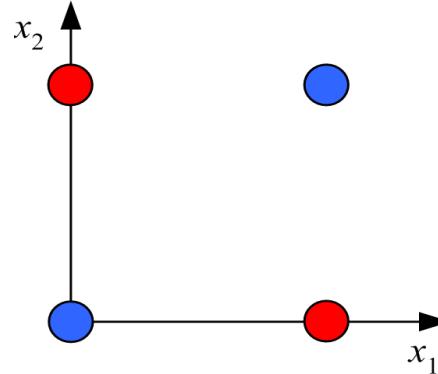


(Minsky and Papert, *Perceptrons*.  
MIT Press, Cambridge 1969)

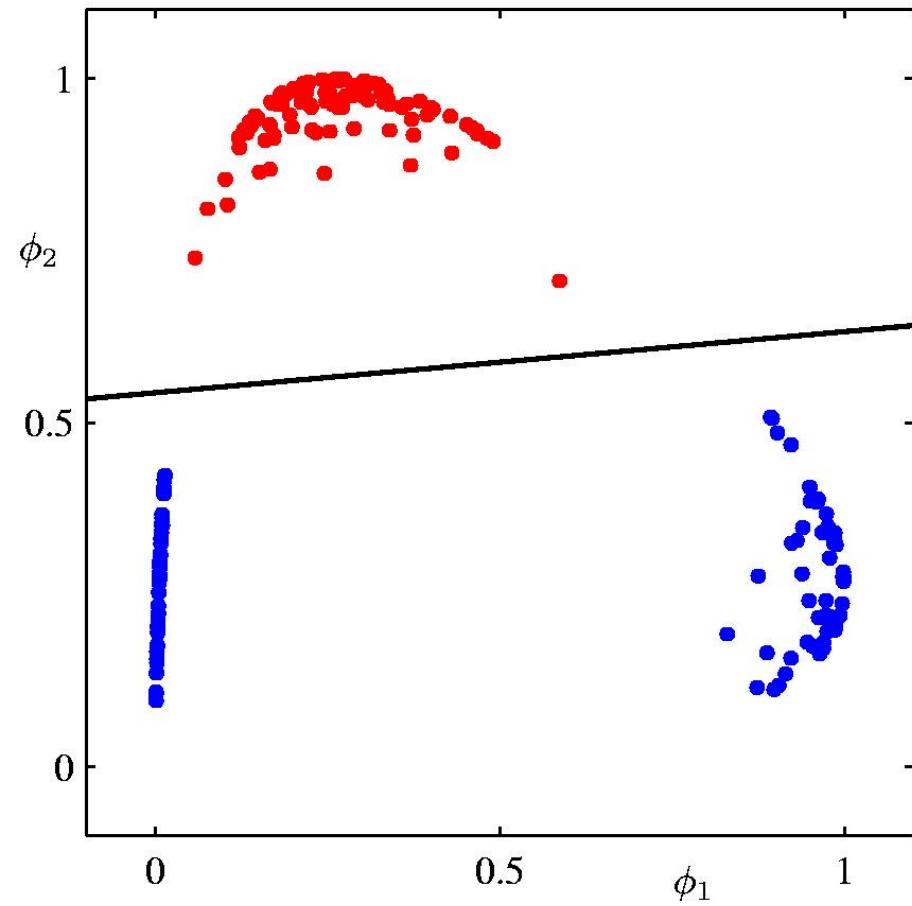
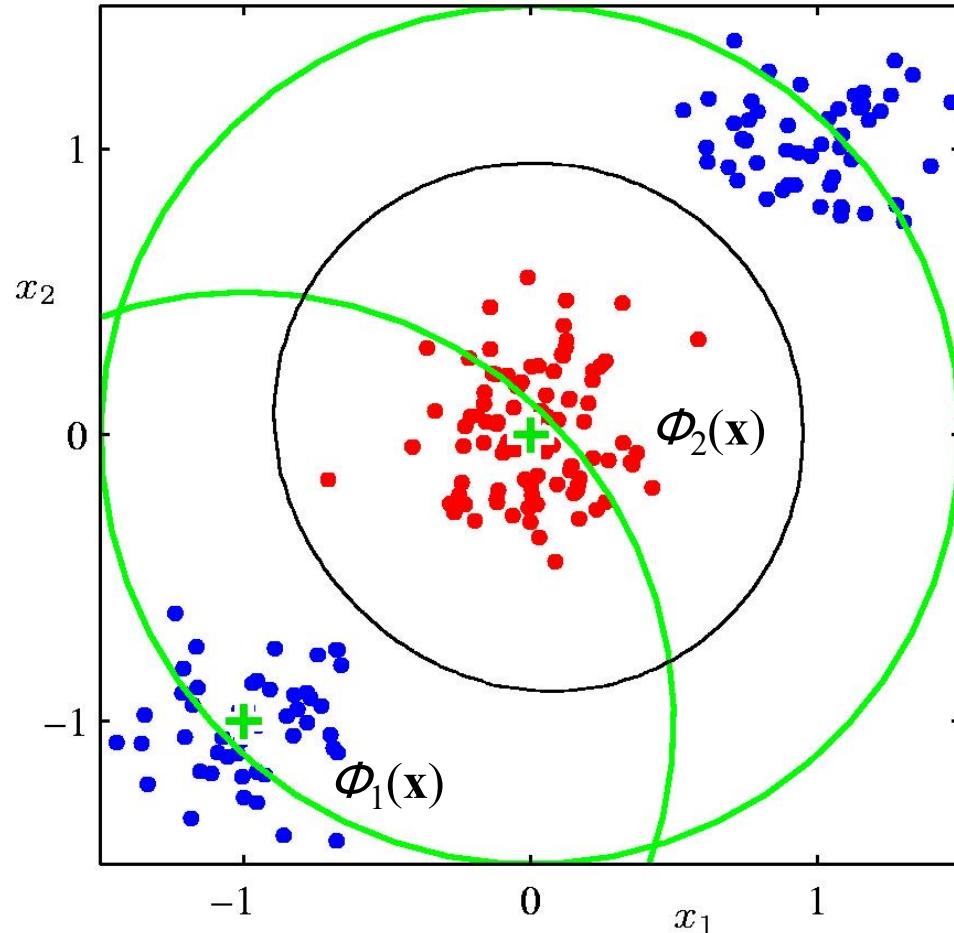
The adaptive part of a perceptron  
cannot even tell if two single bit  
features have the same value!

# Limitations of linear machines

- The linearly separable dichotomies are the partitions that are realizable by a linear classifier (the boundary between the classes is a hyperplane).
- Some seemingly simple dichotomies are not linearly separable.
- Question: How do we make a given problem linearly separable?



# Role of nonlinear basis function



# Linearly separable?

- The perceptron learning procedure can learn a linear decision surface, if there is such one. Otherwise, it will wobble around.
- There are many interesting applications where the data can be represented in a way that makes the classes (nearly) linearly separable
  - e.g. text classification using “bag of words” representations (e.g. for spam filtering).
- Unfortunately, really interesting applications are generally not linearly separable. This is why most people abandoned the field between the late 60's and the early 80's.
  - We will come back to the linear separability problem later.

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. **Support vector machines**
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 7: Support Vector Machines

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

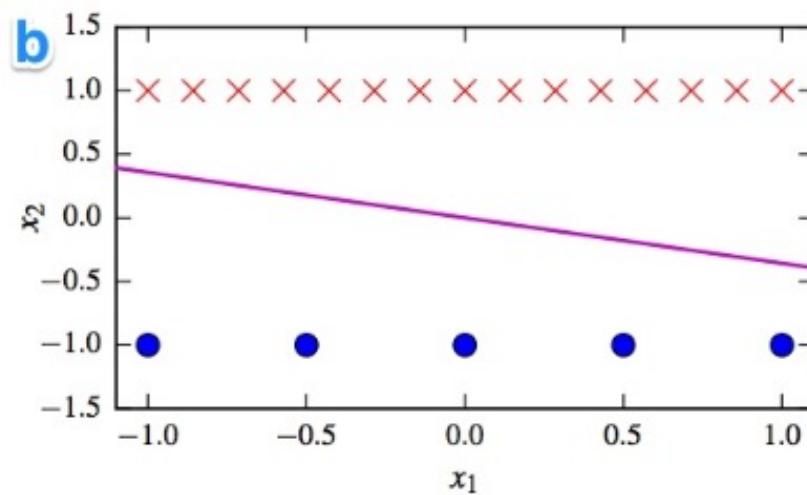
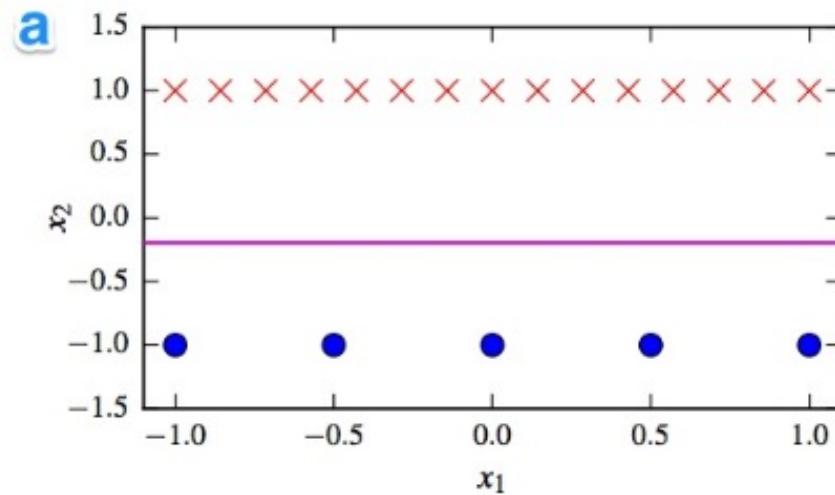
[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. **Support vector machines**
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Linear classifiers for two-class case

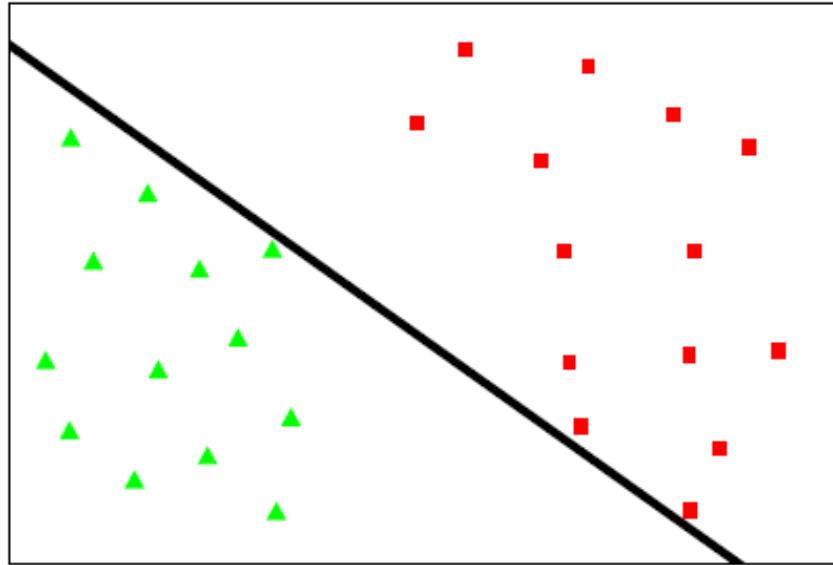
- Perceptron
- Logistic regression



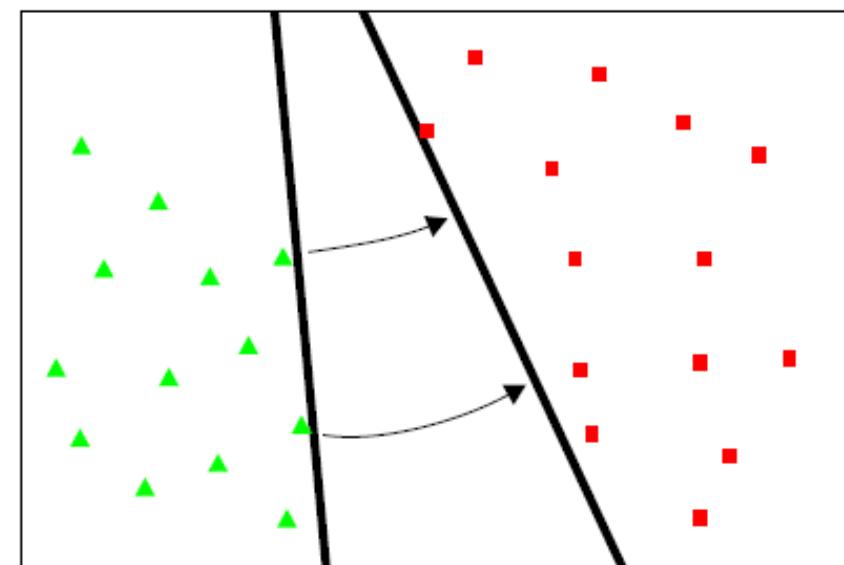
(stackexchange)

# Maximal margin

- Separating hyperplane is not unique



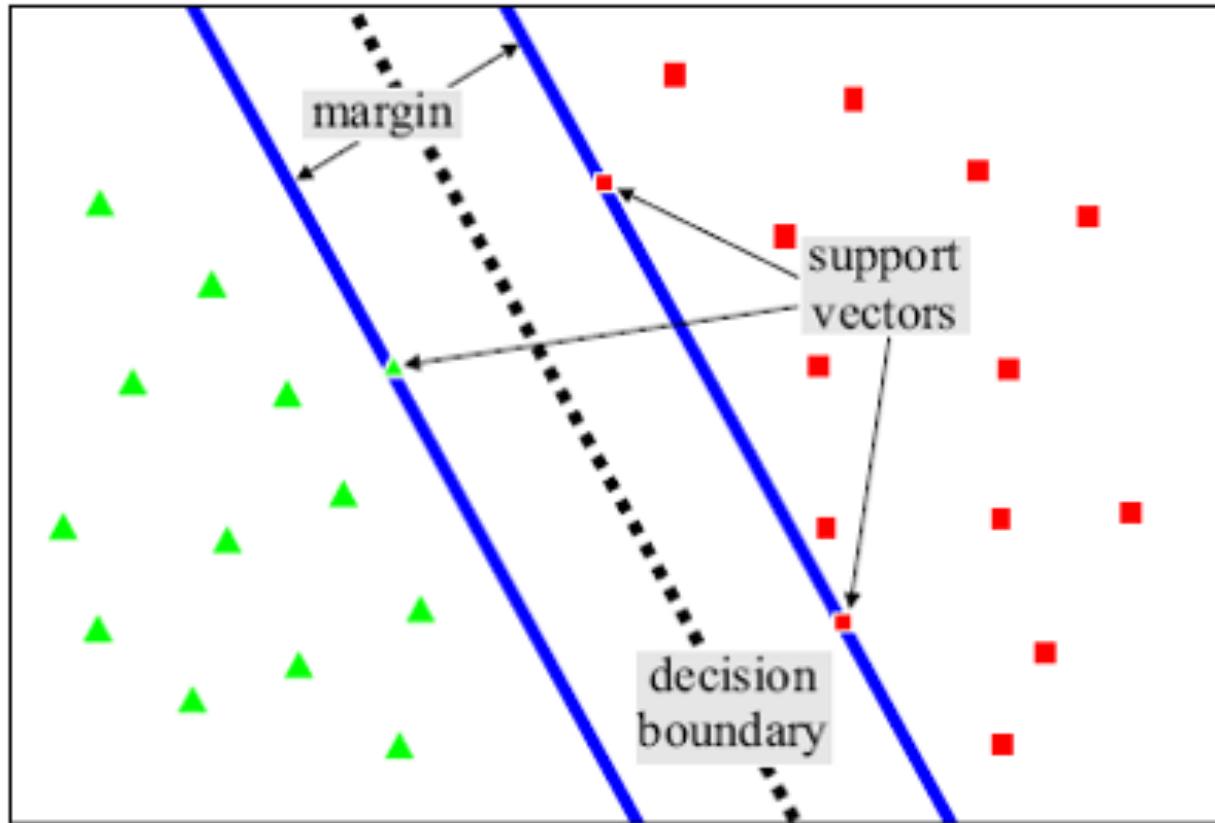
a: An arbitrary linear decision boundary.



b: Any one of an infinite number of boundaries can be chosen.

# Maximal margin – cont.

- Choose the one with maximal margin



The margin is maximised by moving two solid parallel lines as far apart as possible without either making a classification error. The decision boundary (dotted) is halfway between the solid lines. The support vectors are the points that lie on the margin edges.

- What is SVM?
  - A (mostly) binary classifier
  - A linear classifier
  - Supervised learning
  - Good generalization property
- Key ideas
  - Maximal margin
  - Dual space
  - Kernel trick

# Linear Support Vector Machines

- Training data

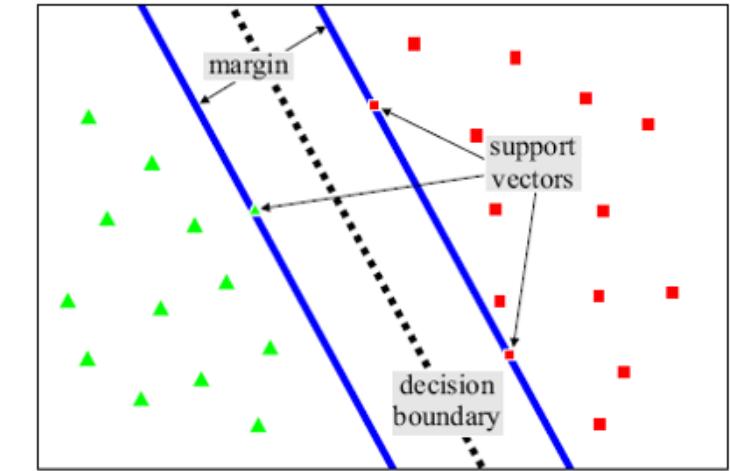
$$\{\mathbf{x}_i, y_i\}, i = 1, \dots, N, y_i \in \{-1, 1\}, \mathbf{x}_i \in \mathbf{R}^d$$

- The points  $\mathbf{x}$  lying on the hyperplane satisfy

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0,$$

where  $\mathbf{w}$  is normal to the hyperplane

- For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with largest margin.



# Linear Support Vector Machines – cont.

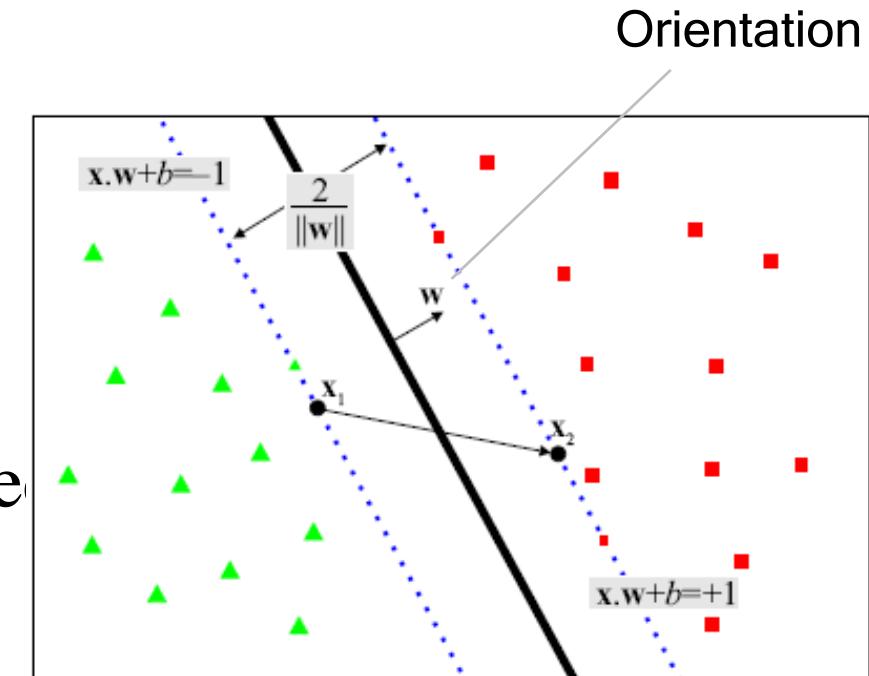
- Formulated as follows

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \geq +1 \text{ for } y_i = +1 \quad (1)$$

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \leq -1 \text{ for } y_i = -1 \quad (2)$$

These can be combined into one set of ine

$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i \quad (3)$$



e:  $\mathbf{w}$  is rescaled so that the classifier gives an output of  $\pm 1$  on the margin edges. The margin width is  $2/\|\mathbf{w}\|$  and can be obtained by finding the component of some vector  $(\mathbf{x}_2 - \mathbf{x}_1)$  in the direction of  $\mathbf{w}$ , the normal to the decision boundary. Maximising the margin is equivalent to minimising  $\|\mathbf{w}\|$ .

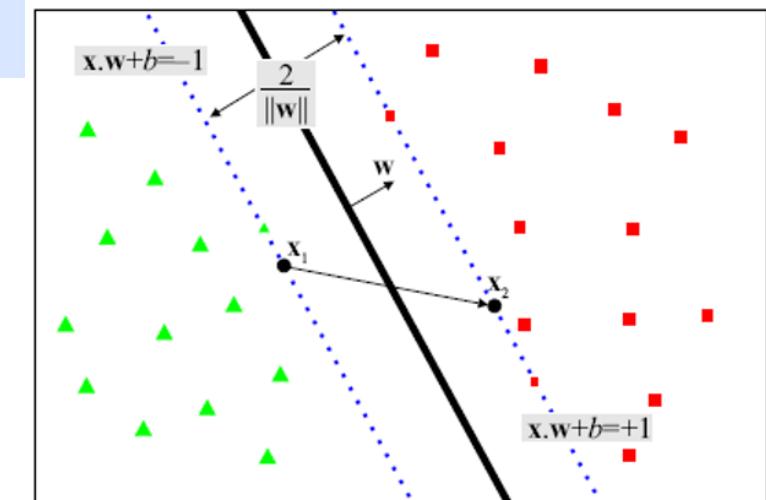
(from Burges 1998; Wan 2003)

# Linear Support Vector Machines – cont.

- Formulated as follows

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \geq +1 \text{ for } y_i = +1 \quad (1)$$

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \leq -1 \text{ for } y_i = -1 \quad (2)$$



These can be combined into one set of inequalities:

$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i \quad (3)$$

The margin is simply  $2/\|\mathbf{w}\|$

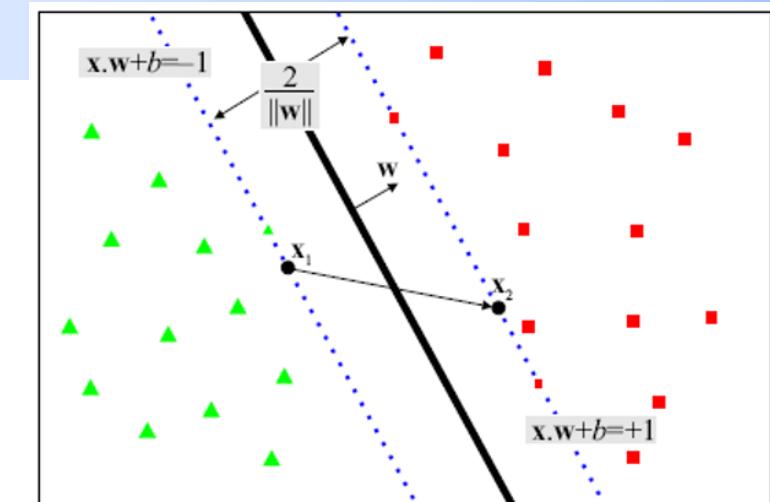
which can be found by projecting  $(\mathbf{x}_2 - \mathbf{x}_1)$  onto  $\mathbf{w}/\|\mathbf{w}\|$ , so we have  $\mathbf{w}^T(\mathbf{x}_2 - \mathbf{x}_1)/\|\mathbf{w}\| = ((\mathbf{w}^T \mathbf{x}_2 + b) - (\mathbf{w}^T \mathbf{x}_1 + b))/\|\mathbf{w}\| = 2/\|\mathbf{w}\|$ .

# Linear Support Vector Machines – cont.

- Formulated as follows

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \geq +1 \text{ for } y_i = +1 \quad (1)$$

$$\mathbf{w}^T \cdot \mathbf{x}_i + b \leq -1 \text{ for } y_i = -1 \quad (2)$$



These can be combined into one set of inequalities:

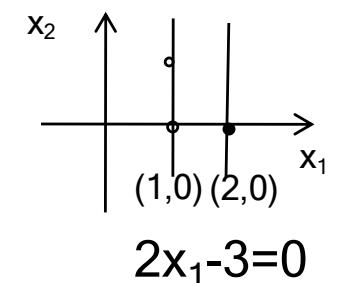
$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i \quad (3)$$

The margin is simply  $2/\|\mathbf{w}\|$

- Thus we can find the pair of hyperplanes which gives the maximum margin by minimizing  $\frac{1}{2} \|\mathbf{w}\|^2$ , subject to constraints (3).

a quadratic programming problem

Objective function



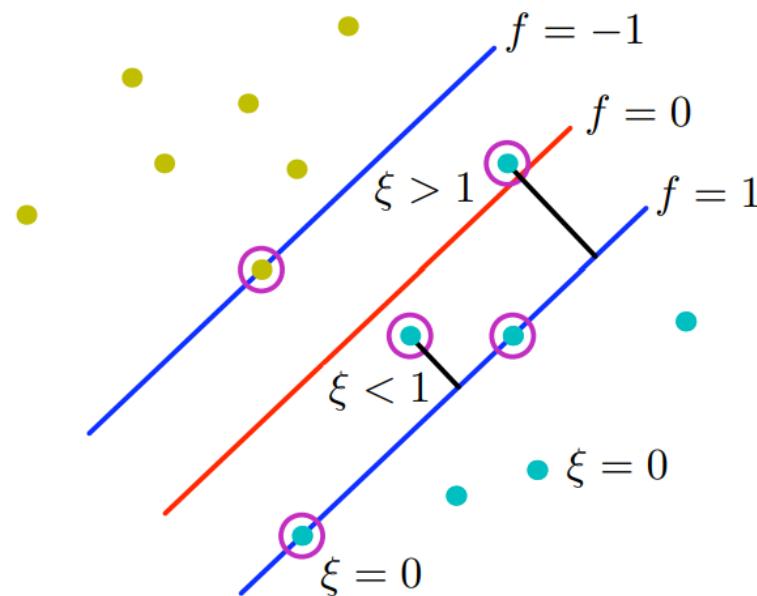
# Soft margin SVMs

(optional self-study)

- If data is not separable, we introduce a penalty term /slack variables (violation of margin constraints)

$$\min_{\mathbf{w}, b, \xi_{1:N}} \sum_i \xi_i + \lambda \frac{1}{2} \|\mathbf{w}\|^2$$

such that, for all  $i$ ,  $y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i$  and  $\xi_i \geq 0$



# Lagrangian formulation of the problem

- We introduce positive Lagrange multipliers  $\alpha_i, i = 1, \dots, N$  one for each of the inequality constraints (3).
- The Lagrangian is formed through subtracting the objective function by the constraint equations multiplied by positive Lagrange multipliers:

$$L_P(\mathbf{w}, b, \boldsymbol{\alpha}) \equiv \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i (\mathbf{w}^T \cdot \mathbf{x}_i + b) - 1) \quad (4)$$

- The **Lagrange function** is given by the primal objective function minus the sum of all products between constraints and corresponding Lagrange multipliers: minimizing w.r.t.  $\mathbf{w}$  and  $b$  and maximizing w.r.t.  $\boldsymbol{\alpha}$ .

# Lagrangian formulation of the problem – cont.

$$L_P \equiv \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i y_i (\mathbf{w}^T \cdot \mathbf{x}_i + b) + \sum_{i=1}^N \alpha_i \quad (4)$$

Minimize  $L_P$  w.r.t.  $\mathbf{w}$  and  $b$  by setting the derivatives of  $L_P$  w.r.t.  $\mathbf{w}$  and  $b$  to zero, we obtain the following two conditions:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (5) \quad \& \quad \sum_i \alpha_i y_i = 0 \quad (6)$$

Substitute them into (4), and give a different label ( $\mathbf{w}$  and  $b$  disappear):

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \cdot \mathbf{x}_j \quad (7)$$

which we maximize w.r.t.  $\alpha$  subject to (6) and  $\alpha_i \geq 0$ .

$L_P$  and  $L_D$  arise from same objective function but with diff. constraints; and solution is found by minimizing  $L_P$  or by maximizing  $L_D$  - Dual problem.

# Lagrangian formulation of the problem

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (5) \quad \& \quad \sum_i \alpha_i y_i = 0 \quad (6)$$

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \cdot \mathbf{x}_j \quad (7)$$

SVM training is to maximize  $L_D$  with respect to the  $\alpha_i$  (7), subject to constraints (6) and positive  $\alpha_i$ , with solution given by (5).

Note there is one  $\alpha_i$  for every training point. In the solution, those points for which  $\alpha_i > 0$  are called “support vectors” lying on  $H_1$  or  $H_2$ . Other points have  $\alpha_i = 0$ .

$b$  is found by using Eq. (3) for supporting vectors (equal to 0).

# Decision function

- The decision function of SVM can be defined as

$$f(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + b = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} + b \quad (8)$$


---

- The SVM classifier is defined in terms of training examples.

SVs ( $\alpha_i, y_i, \mathbf{x}_i$ ):

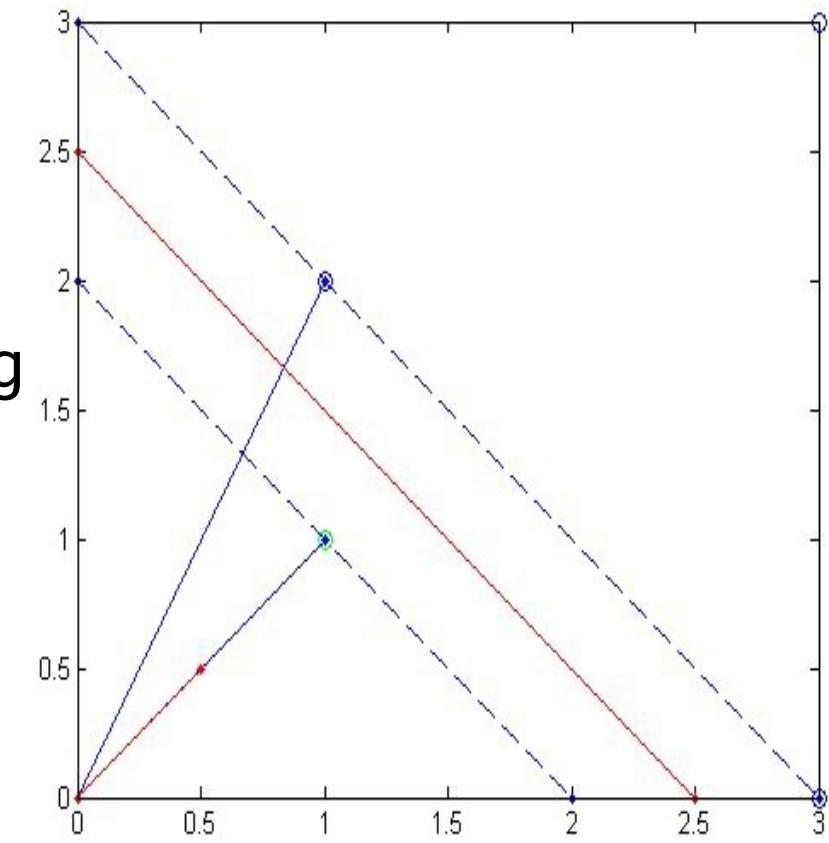
0.25 [3 0];  $\mathbf{w}$  [0.5 0.5] before scaling

0.75 [1 2];  $\mathbf{w}$  [2 2] after scaling

-1 [1 1]

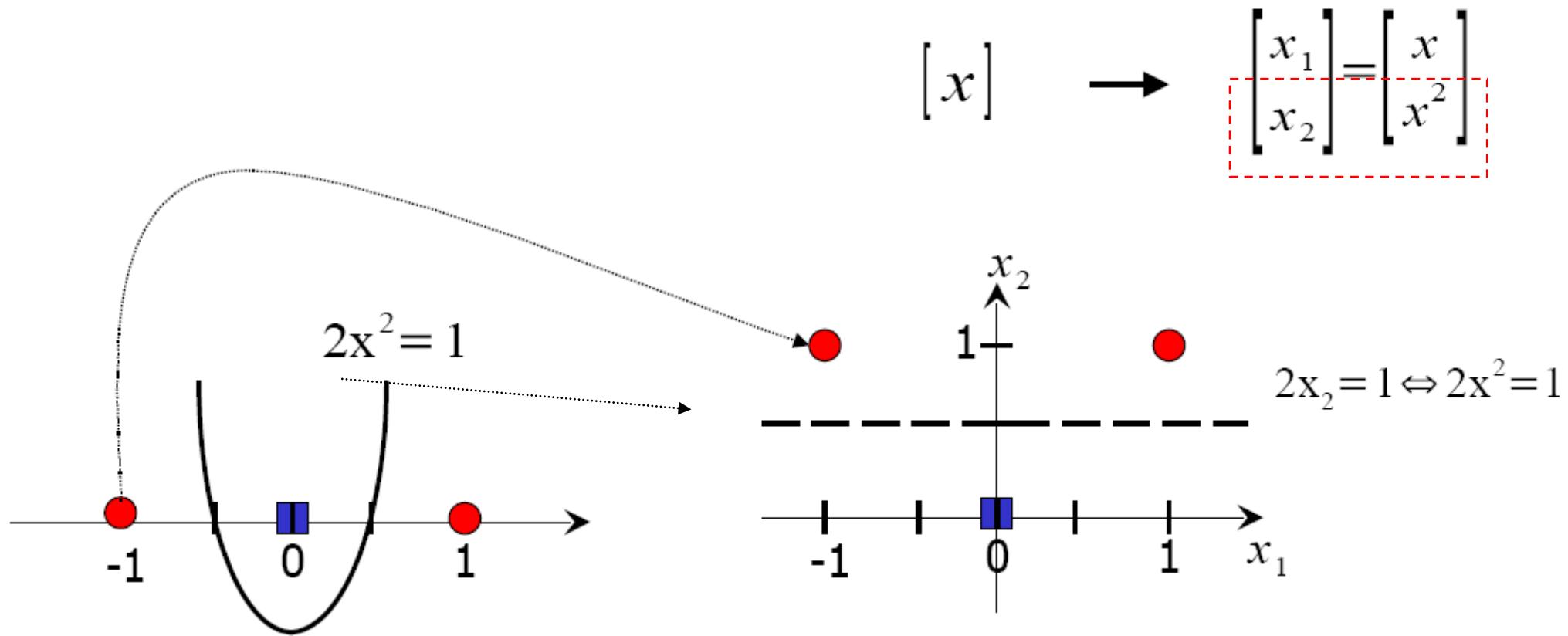
$$2x_1 + 2x_2 - 5 = 0$$


---

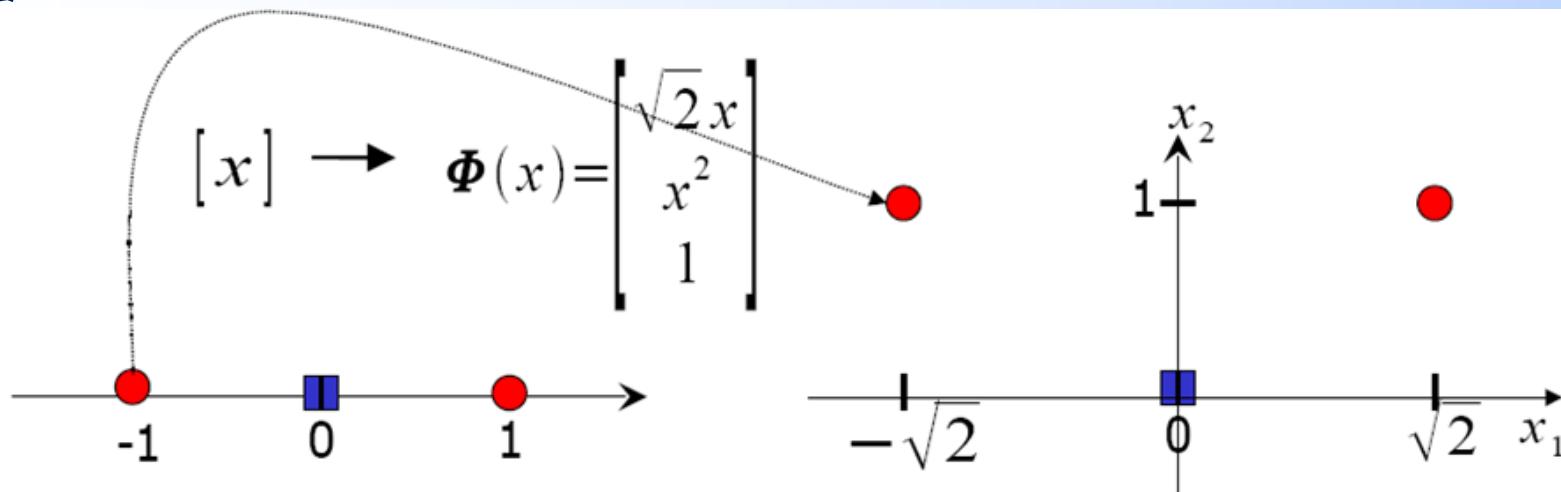


# Non-separable data

- Solutions: (Huang)
  - Nonlinear classifier
  - Increase the dimension



# Example



$$\langle \Phi(x_i)^T \cdot \Phi(x_j) \rangle = 2x_i x_j + x_i^2 x_j^2 + 1 = (x_i x_j + 1)^2 = K(x_i, x_j)$$

Therefore,  $K(x_i, x_j) = (x_i x_j + 1)^2$

No need to calculate  $\Phi(x_i)$  and  $\Phi(x_j)$ , only  $\Phi(x_i)^T \cdot \Phi(x_j)$ .

$$f(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} + b = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i^T \cdot \mathbf{x} + b \quad (8)$$

$$f(\Phi(\mathbf{x})) = \sum_{i=1}^N \alpha_i y_i \Phi(\mathbf{x})_i^T \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^N \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

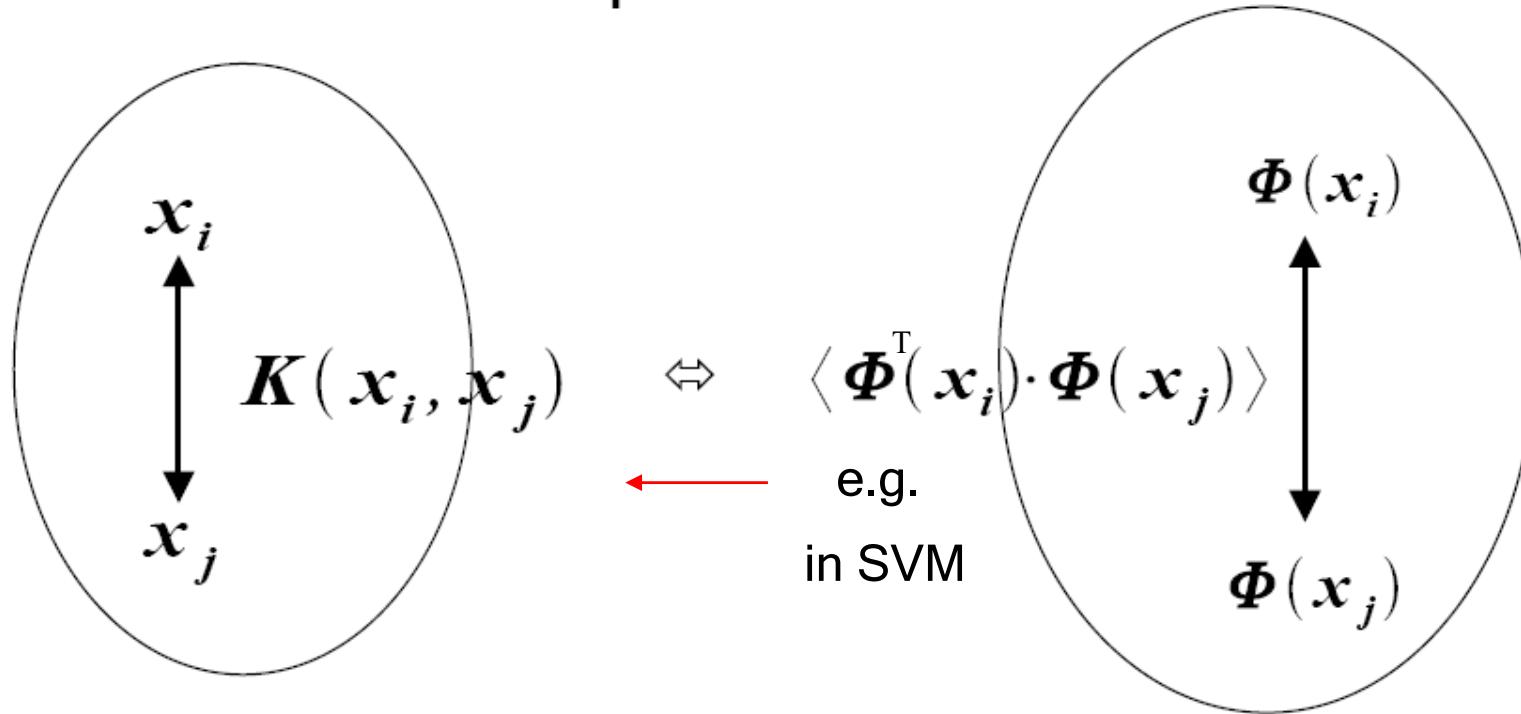
# Kernel trick

## Kernel function

- in the original space

## Inner product

- In the feature space with increased dimension



# Kernel function

- Training vectors  $\mathbf{x}_i$  are mapped into a higher (maybe infinite) dimensional space by the function  $\phi$ . Then SVM finds a linear separating hyperplane in this higher dimensional space.
- $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$  is called the kernel function.
  - No need to calculate  $\phi(\mathbf{x}_i)$ , simply use  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$
  - No need to define  $\phi(\mathbf{x}_i)$ , only define  $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$  as above
  - Kernel trick is thus limited to algorithms using multiplication between vectors, e.g. SVMs.

$$f(\mathbf{x}) = \sum_{i=1}^N \alpha_i y_i \underline{\mathbf{x}_i^T \cdot \mathbf{x}} + b = \mathbf{w}^T \cdot \mathbf{x} + b \quad (8)$$

$$f(\Phi(\mathbf{x})) = \sum_{i=1}^N \alpha_i y_i \Phi(\mathbf{x})_i^T \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^N \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$


---

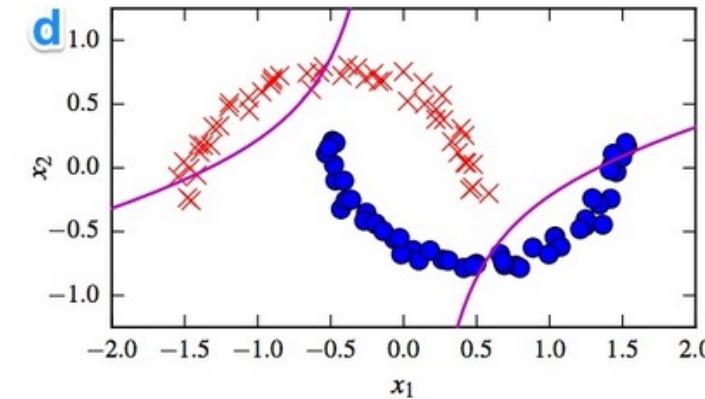
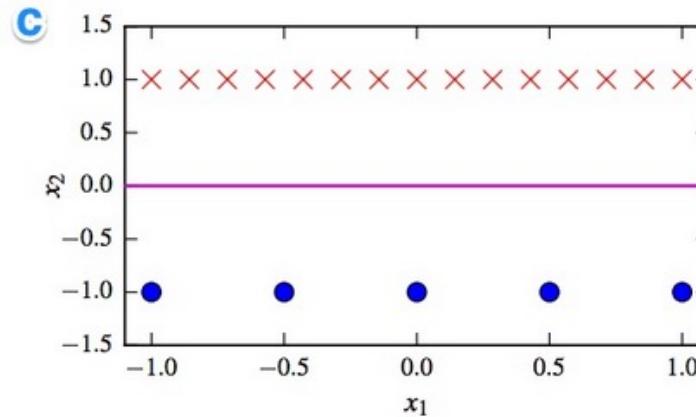
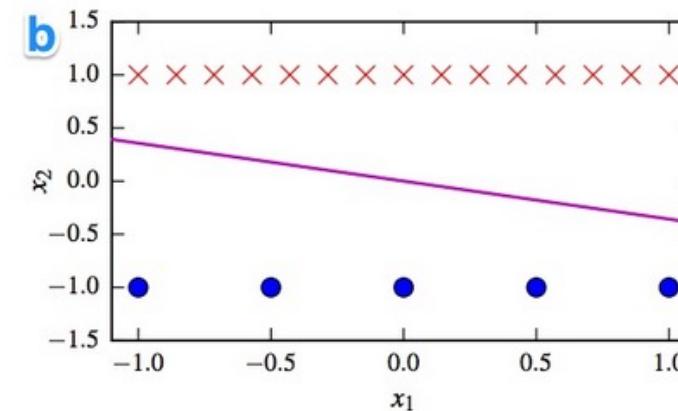
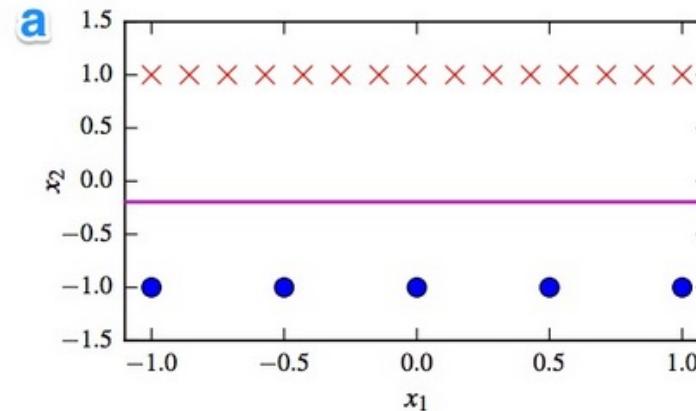
# Kernel functions

- Basic kernels:

- Linear:  $K(x_i, x_j) = x_i^T x_j$
- Polynomial:  $K(x_i, x_j) = (\gamma x_i^T x_j + c)^d, \gamma > 0$
- Radial basis function (RBF), also called Gaussian RBF:  
$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2), \gamma > 0$$
- Sigmoid:  $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$

# Linear classifiers for two-class case

- Perceptron, logistic regression, SVM

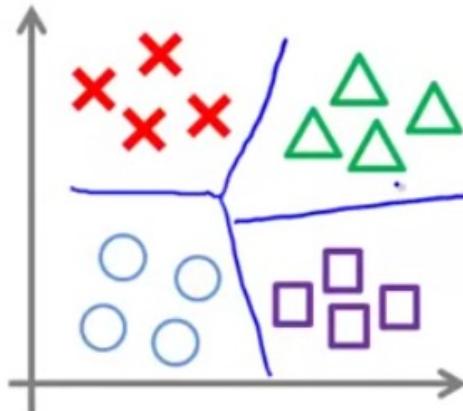


(stackexchange)

# Multi-class classification with SVM

(optional self-study)

## Multi-class classification



$$y \in \{1, 2, 3, \dots, K\}$$



Many SVM packages already have built-in multi-class classification functionality.

Otherwise, use one-vs.-all method. (Train  $K$  SVMs, one to distinguish

$y = i$  from the rest, for  $i = 1, 2, \dots, K$ ), get  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$

~~Pick class  $i$  with largest  $(\theta^{(i)})^T x$~~



13:05 / 21:02



Andrew

Lecture 12.6 – Support Vector Machines | Using An SVM – [ Machine Learning | Andrew Ng]

# Pros and cons

- Works very well
- Fool-proof
  - Only a few kernels to choose from (often polynomial and Gaussian)
  - Very few hand-crafted parameters
  - Error bound easy to get
- Slow
  - Training: quadratic programming
  - Testing: depend on number of SV
- Big
  - Curse of sample size

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 8: Multilayer Perceptrons / Feed-forward Neural Networks

Zheng-Hua Tan, Sarthak Yadav

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt, sarthaky@es.aau.dk](mailto:zt, sarthaky@es.aau.dk),

<http://kom.aau.dk/~zt>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

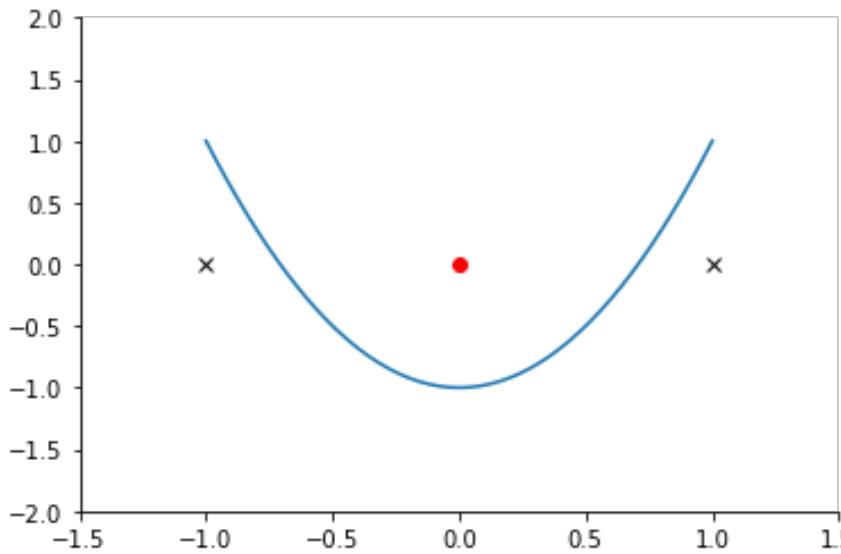
1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Time series models
9. Multilayer perceptrons
10. Deep learning and transfer learning
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Previously on Machine Learning

- Learned about decision boundaries
- Learned about linearly separable v/s non-separable problems
- Learned about perceptron
  - What it is
  - How to train it
  - Why it's not enough
- Learned about Support Vector Machines
  - Fixed basis functions

# Non-separable data and representation

Nonlinear classifier:  $y = f(x) = 2x^2 - 1$

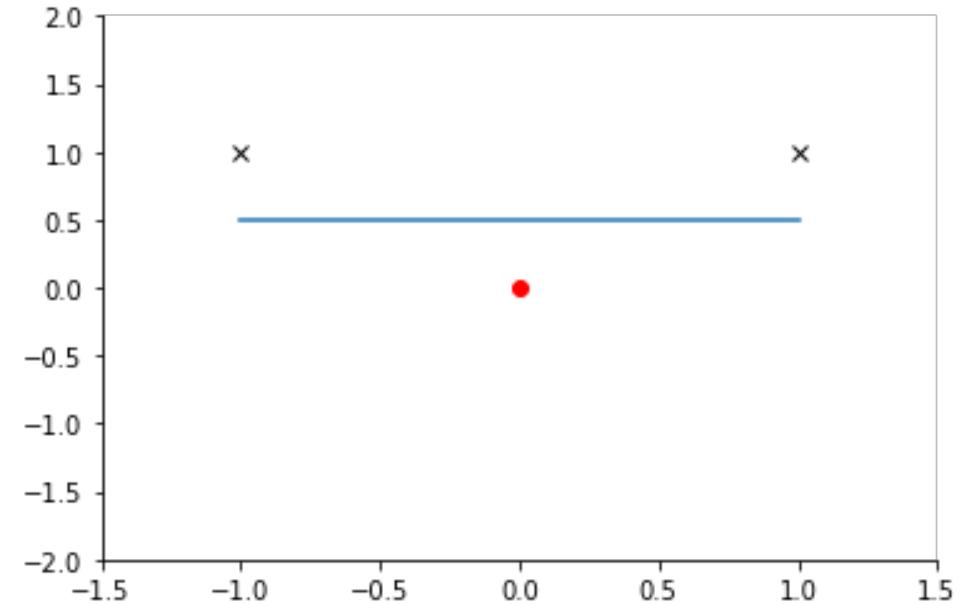


Neural networks are supervised learning methods for learning representations - mapping from one space to another!

From nonlinear separable to linear separable

Represent the data in higher dimension from  $[x]$  to  $[x_1, x_2]^T = [x, x^2]^T$

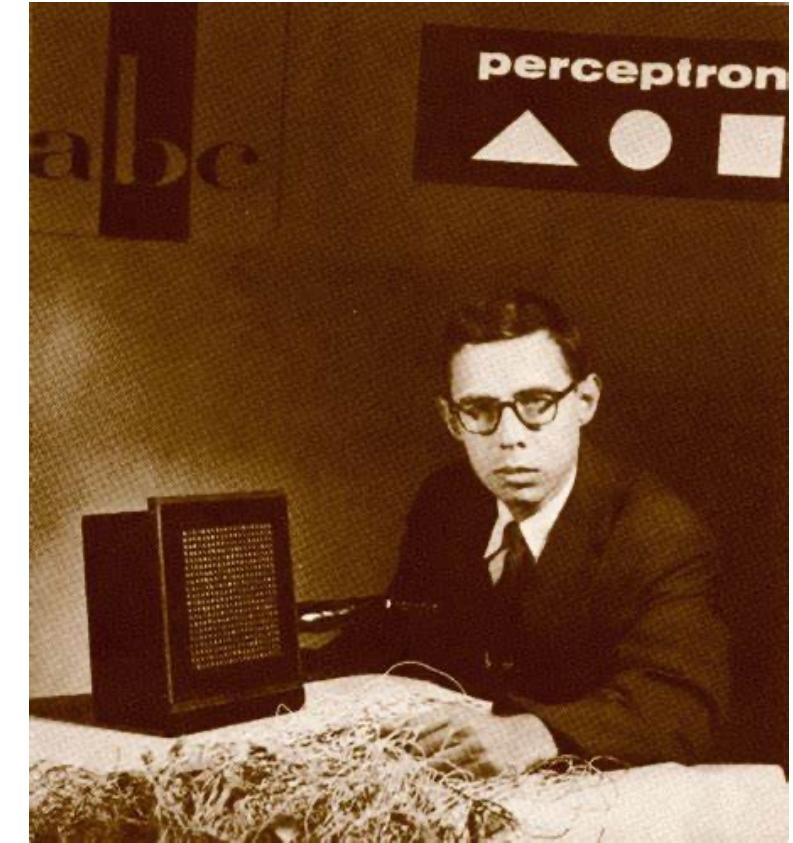
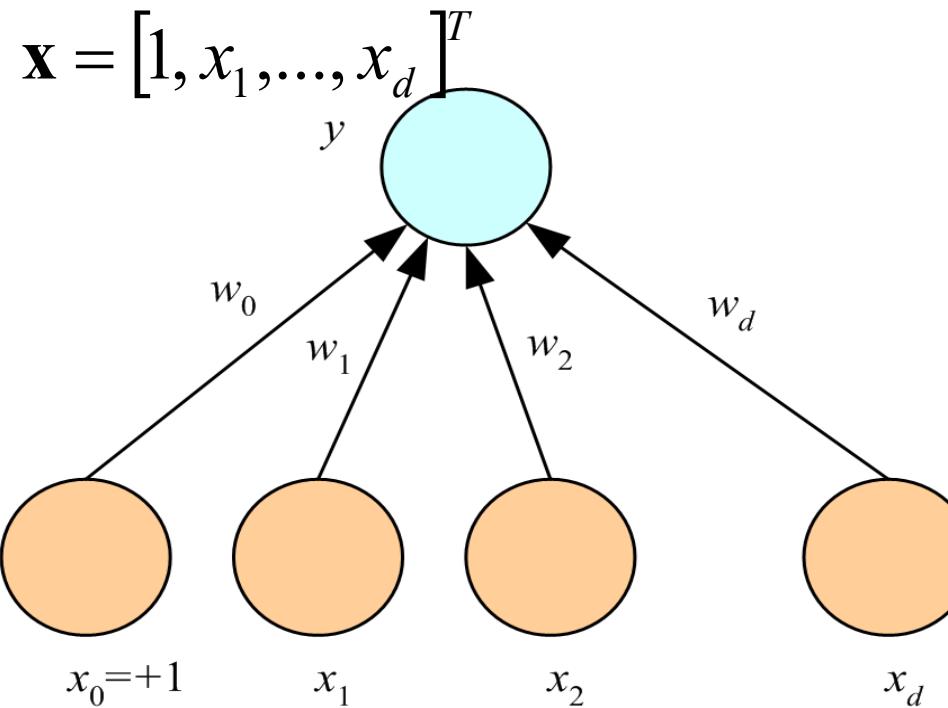
Linear classifier:  $y = f([x_1, x_2]^T) = 2x_2 - 1$



# Perceptron concept

$$y = \sum_{j=1} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

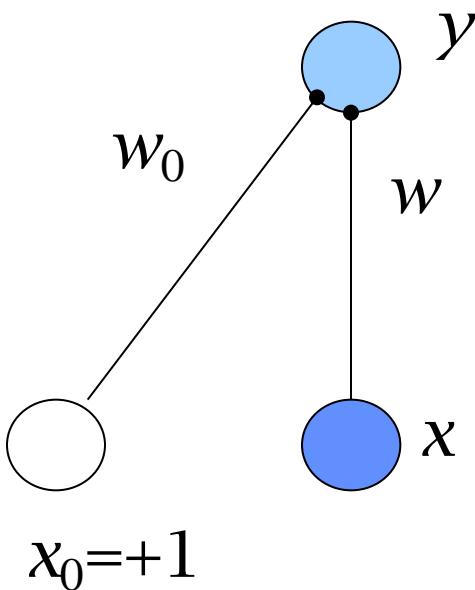
$$\mathbf{w} = [w_0, w_1, \dots, w_d]^T$$



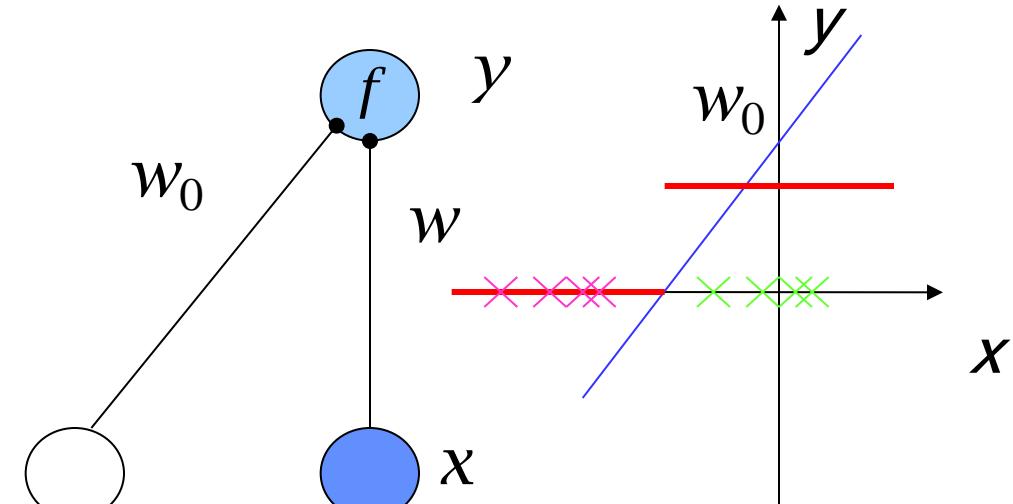
(Rosenblatt, 1962)

# What a perceptron (1 output) does

- Regression:  $y = w_0 + wx$



- Classification:  $y = 1(wx + w_0 > 0)$



Threshold function

If posterior probability is needed later

$$y = \text{sigmoid}(o) = \frac{1}{1 + \exp[-\mathbf{w}^T \mathbf{x}]}$$

# K outputs

Regression:

$$y_i = \sum_{j=1}^d w_{ij} x_j + w_{i0} = \mathbf{w}_i^T \mathbf{x}$$

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

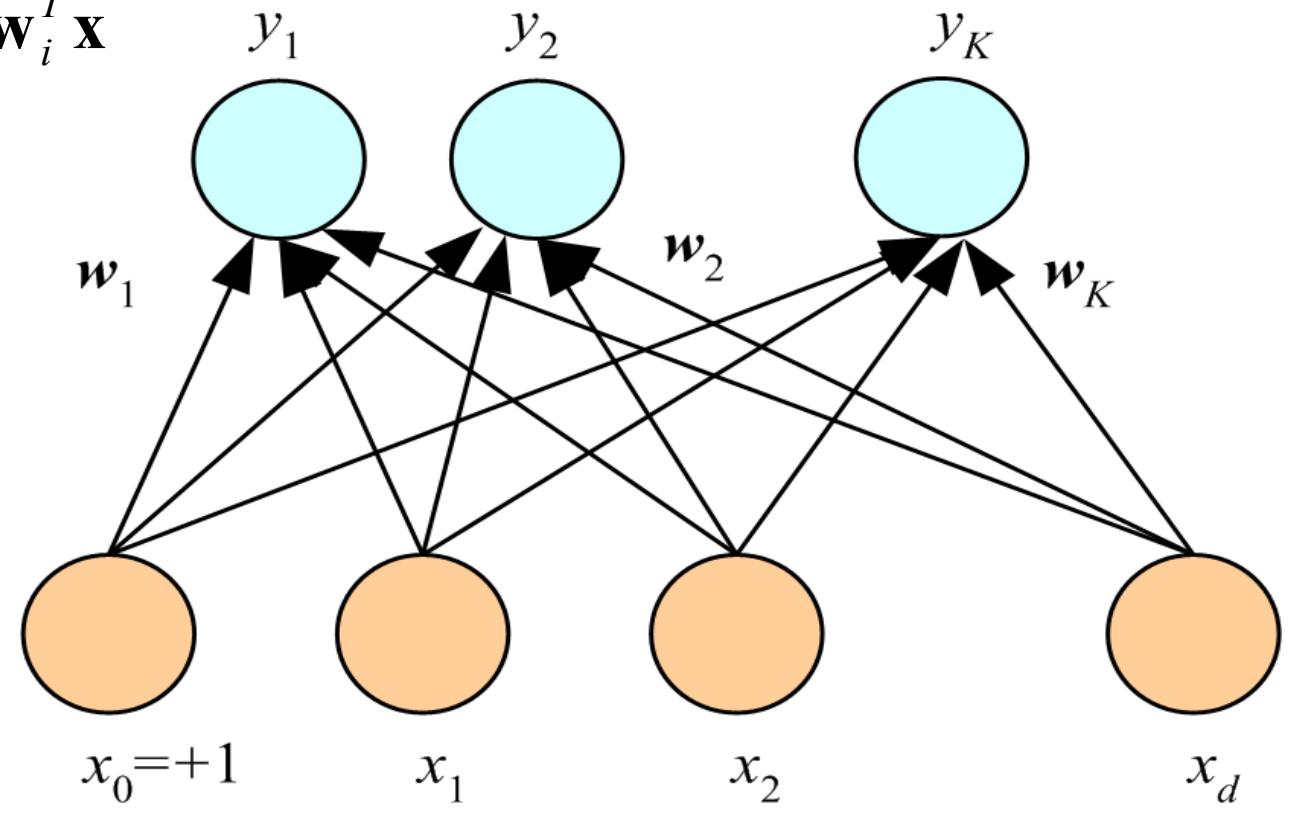
Classification:

$$o_i = \mathbf{w}_i^T \mathbf{x}$$

$$y_i = \frac{\exp o_i}{\sum_k \exp o_k}$$

choose  $C_i$

if  $y_i = \max_k y_k$



# Feed-forward NN functions

- The linear models are based on linear combinations of fixed nonlinear basis functions  $\phi_j(\mathbf{x})$  and take the form

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^M w_j \phi_j(\mathbf{x})\right)$$

- where  $f(\cdot)$  is a nonlinear activation function in classification and is an identity function in regression.
- Our goal is to extend this model **by making the basis functions  $\phi_j(\mathbf{x})$  depend on parameters and then to allow these parameters to be adjusted**, along with the coefficients  $\{w_j\}$ , during training. With multiple outputs,

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$

# Feed-forward neural networks

Three key elements:

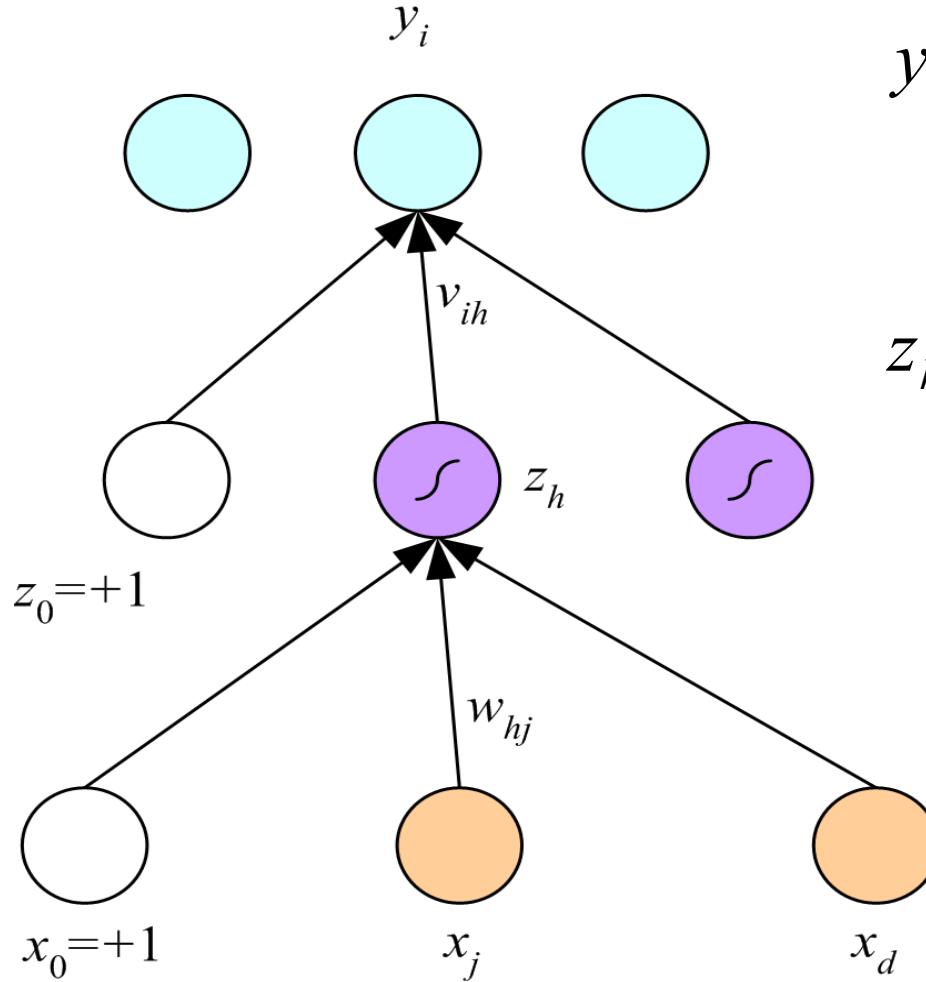
- Hypothesis / hypothesis function / model
  - A non-linear mapping function  $f_\theta(\mathbf{x})$ , defined by the network architecture
- Objective/loss/error functions, e.g.
  - Squared error for regression:  $\sum_{n=1}^N (y_n - f_\theta(x_n))^2$
  - Cross entropy for classification:
$$\sum_{n=1}^N (-y_n \log f_n - (1 - y_n) \log(1 - f_n))$$
- Optimization/training algorithms to find  $\theta$ , e.g.
  - Gradient descent, error backpropagation (BP)

# Feed-forward NN functions - cont.

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^M w_{kj}^{(2)} h\left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$

- where the differentiable, nonlinear activation functions  $h(.)$  and  $f(.)$  are often chosen to be logistic sigmoid functions; for  $h(.)$ , ReLU is also often used.
- The NN model is simply **a nonlinear function** from a set of input variables  $\{\mathbf{x}_i\}$  to a set of output variables  $\{y_i\}$  controlled by adjustable parameters  $\mathbf{W}$ .
- The process of evaluating the above equation can be interpreted as a **forward propagation** of information through the network.
- The neural network model comprises two stages of processing, each of which resembles the perceptron model and thus it is also known as the multilayer perceptron (MLP).

# Multilayer perceptrons for regression



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^H v_{ih} z_h + v_{i0}$$

Adaptive basis function

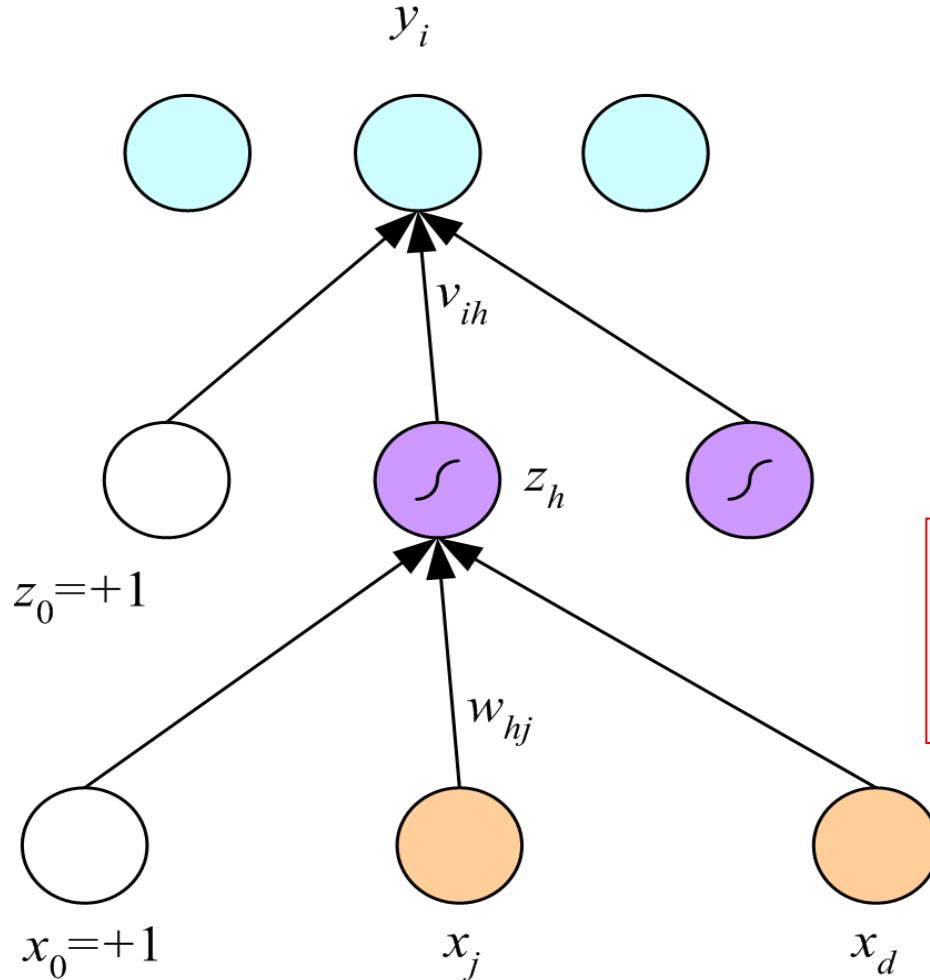
$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x})$$

$$= \frac{1}{1 + \exp[-(\sum_{j=1}^d w_{hj} x_j + w_{h0})]}$$

# Architecture of multilayer perceptron

- Each unit performs a linear combination of its inputs, and passes the result through a nonlinear function, e.g., sigmoid function.
- Each layer is composed of a number of units.
  - Each unit in one layer gets inputs from every unit in the previous layer.
  - “hidden” layers: not directly constrained from the outside (nor provided to the outside), so not directly related to  $\{x_i\}$  or  $\{y_i\}$ .
- Multilayer NNs can be seen as networks of logistic regressors.

# Error backpropagation (BP)



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^H v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x})$$

$$= \frac{1}{1 + \exp[-(\sum_{j=1}^d w_{hj} x_j + w_{h0})]}$$

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}$$

# BP training: regression

Regression: only one output neuron

$$y^t = \sum_{h=1}^H v_h z_h^t + v_0$$

$$E(\mathbf{W}, \mathbf{v} | \mathcal{X}) = \frac{1}{2} \sum_t (r^t - y^t)^2$$

$$\Delta v_h = -\eta \frac{\partial E}{\partial v_h} = \eta \sum_t (r^t - y^t) z_h^t$$

*Backward*

*Forward*

$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x})$$

$\mathbf{x}$

$$\begin{aligned} \Delta w_{hj} &= -\eta \frac{\partial E}{\partial w_{hj}} \\ &= -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}} \\ &= -\eta \sum_t -(r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t \\ &= \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t \end{aligned}$$

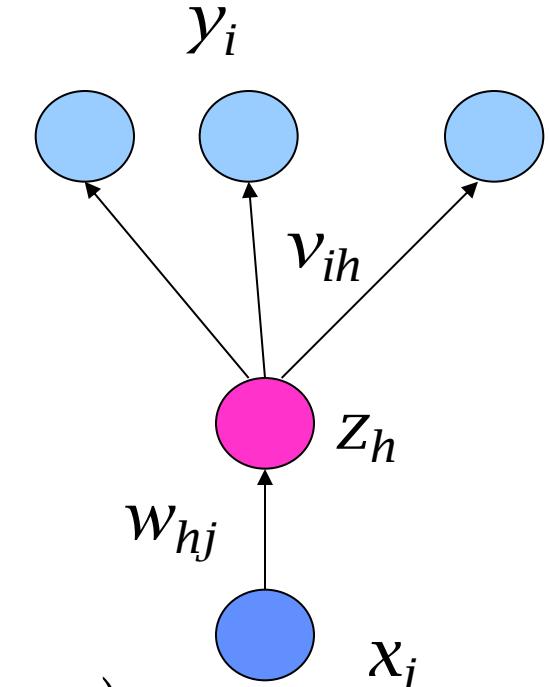
# Regression with multiple outputs

$$E(\mathbf{W}, \mathbf{V} | \mathcal{X}) = \frac{1}{2} \sum_t \sum_i (r_i^t - y_i^t)^2$$

$$y_i^t = \sum_{h=1}^H v_{ih} z_h^t + v_{i0}$$

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[ \sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t$$



# BP training: regression

Initialize all  $v_{ih}$  and  $w_{hj}$  to  $\text{rand}(-0.01, 0.01)$

Repeat

    For all  $(\mathbf{x}^t, r^t) \in \mathcal{X}$  in random order

        For  $h = 1, \dots, H$

$$z_h \leftarrow \text{sigmoid}(\mathbf{w}_h^T \mathbf{x}^t)$$

        For  $i = 1, \dots, K$

$$y_i = \mathbf{v}_i^T \mathbf{z}$$

        For  $i = 1, \dots, K$

$$\Delta \mathbf{v}_i = \eta(r_i^t - y_i^t) \mathbf{z}$$

        For  $h = 1, \dots, H$

$$\Delta \mathbf{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \mathbf{x}^t$$

        For  $i = 1, \dots, K$

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta \mathbf{v}_i$$

        For  $h = 1, \dots, H$

$$\mathbf{w}_h \leftarrow \mathbf{w}_h + \Delta \mathbf{w}_h$$

Until convergence

# MLP for two-class classification

- One sigmoid output  $y^t$  for  $P(C_1|x^t)$  and  $P(C_2|x^t) \equiv 1 - y^t$
- The likelihood function is Bernoulli distribution:
$$p(r|w) = \prod_{t=1}^T (y_t)^{r_t} * (1 - y_t)^{(1 - r_t)}$$
- The error function is as usual defined as taking the negative logarithm of the likelihood, which gives the cross entropy error function in next slide

# MLP for two-class classification - cont'd

The error function is cross entropy:

$$E(\mathbf{W}, \mathbf{v} | \mathcal{X}) = -\sum_t \left( r^t \log y^t + (1 - r^t) \log (1 - y^t) \right)$$

Take gradient:

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

which is the same as for regression on an early slide:

$$\Delta v_h = -\eta \frac{\partial E}{\partial v_h} = \eta \sum_t (r^t - y^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t (r^t - y^t) v_h z_h^t (1 - z_h^t) x_j^t$$

Same as in logistic regression.

# K>2 Classes

$$o_i^t = \sum_{h=1}^H v_{ih} z_h^t + v_{i0} \quad y_i^t = \frac{\text{exp} o_i^t}{\sum_k \text{exp} o_k^t} \equiv P(C_i | \mathbf{x}^t)$$

Multiclass logistic regression takes cross-entropy error function

$$E(\mathbf{W}, \mathbf{v} | \mathcal{X}) = - \sum_t \sum_i r_i^t \log y_i^t$$

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t$$

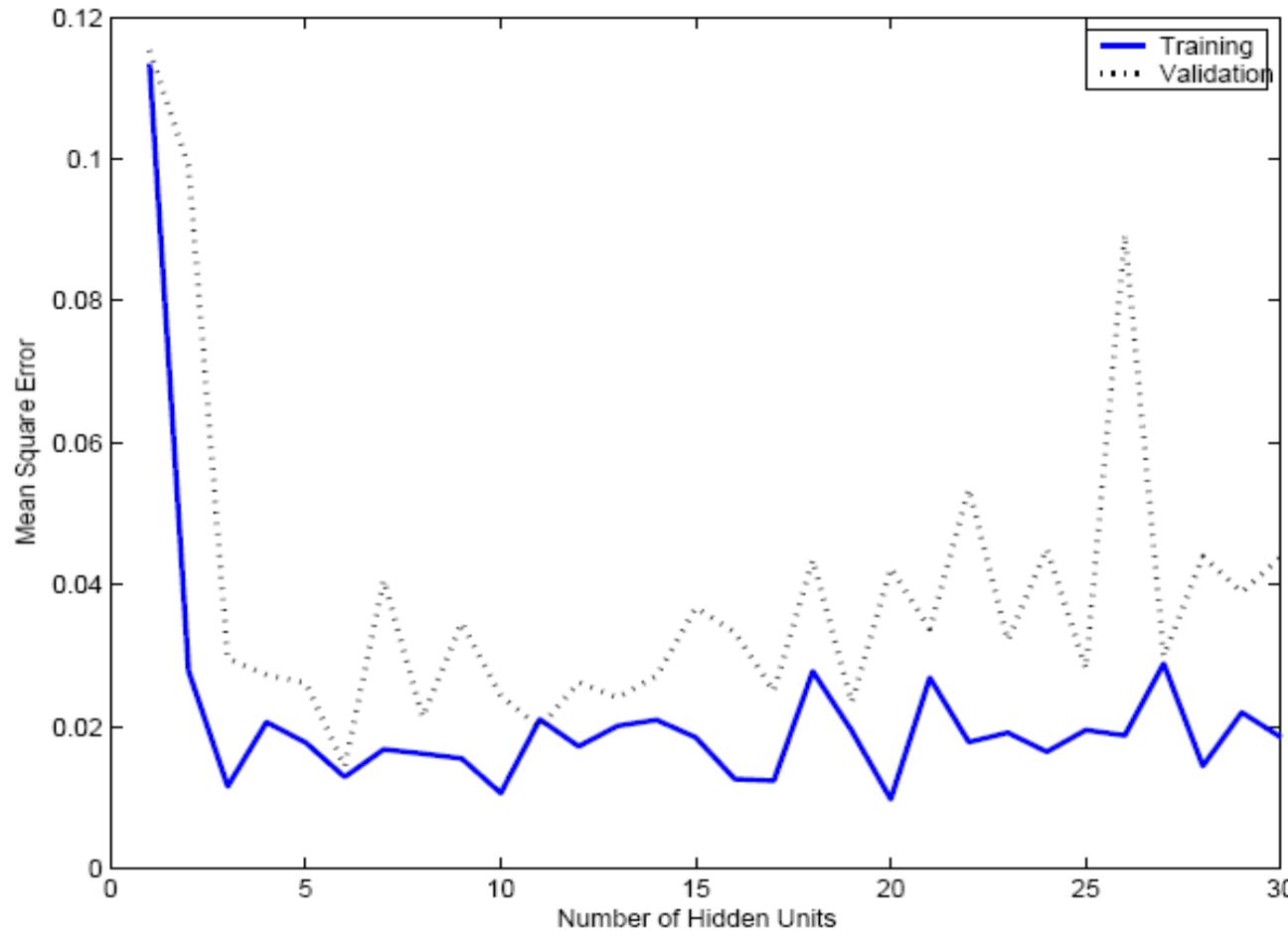
$$\Delta w_{hj} = \eta \sum_t \left[ \sum_i (r_i^t - y_i^t) v_{ih} \right] z_h^t (1 - z_h^t) x_j^t$$

# Training a (multilayer) perceptron

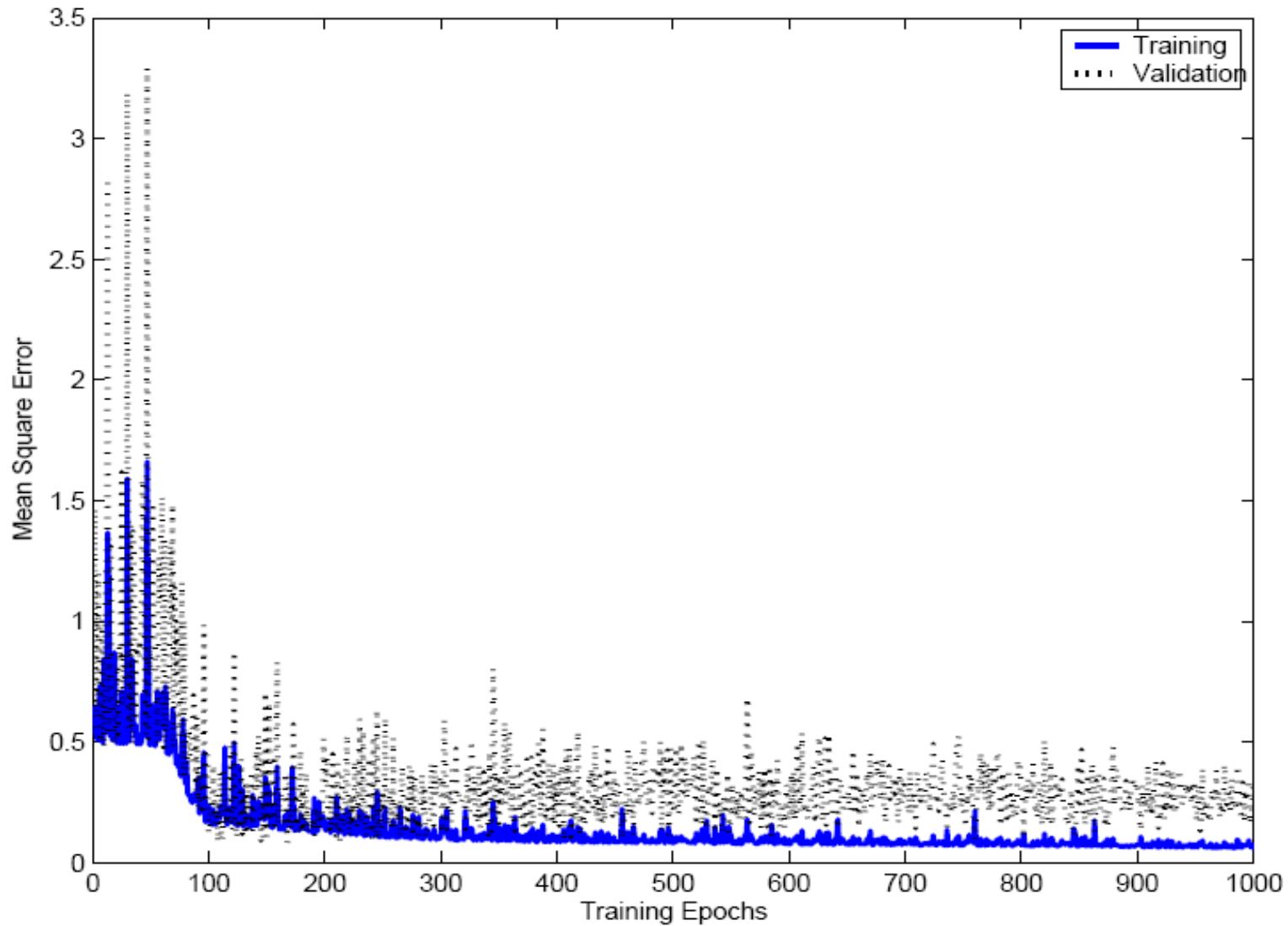
- Online (instances seen one by one) vs. batch/offline (whole sample) learning
- Mini-batch, with one instance and the whole sample as two extreme cases.
- Epoch: One epoch means all data being processed by the algorithm one times.
- Iteration: One iteration means one mini-batch being processed one times. For NN, it means a single forward pass and backward pass.
- Stochastic gradient-descent (SGD): update after a single pattern or a mini-batch

# Overfitting

Number of weights:  $H(d+1)+(H+1)K$



# Overfitting/Overtraining

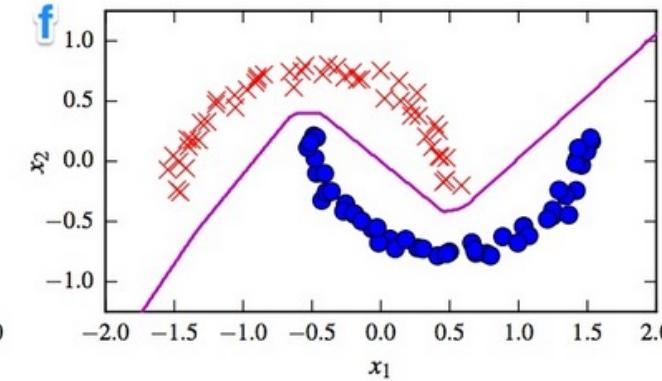
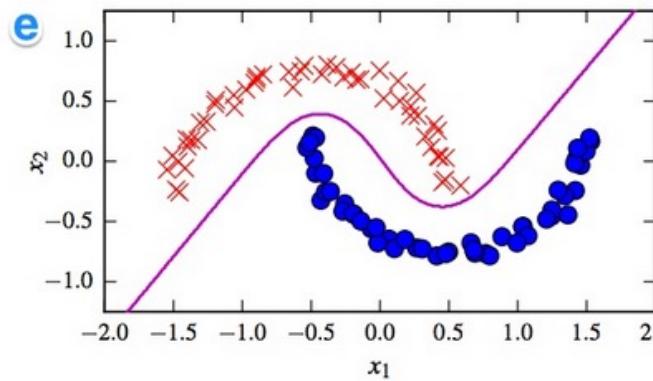
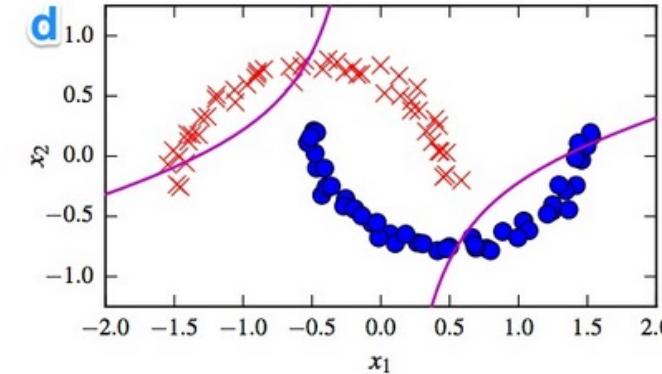
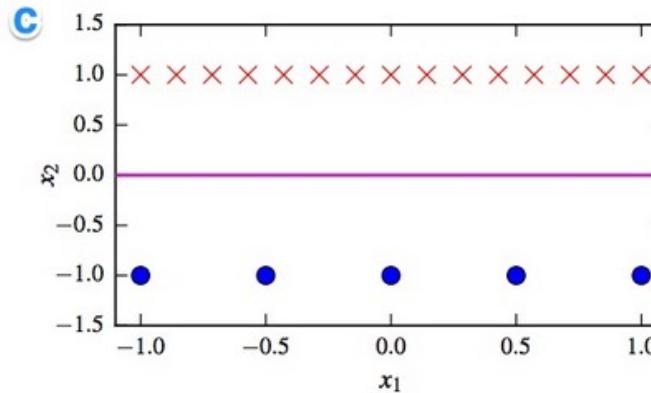


# Designing NN

- Central problems
  - Find a suitable architecture
  - Find the corresponding weights of the network: BP remains the most effective brute-force training algorithm as of today
- General approach
  - Trial and error
  - Fixed architecture during the learning process
  - Parameters are trained by gradient-based algorithms, liable to a local-minimum

# Linear classifiers for two-class case

- SVM, ANN



(stackexchange)

# Feed-forward neural networks

- Hypothesis
  - An intrinsic **non-linear** mapping function  $f(\mathbf{x})$

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$

- In contrast to shallow models, e.g., SVM that is **shallow** with possible **untrainable** nonlinear kernels, neural networks can go **deep**.

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(L)} \dots g\left(\sum_{u=0}^U w_{vu}^{(2)} h\left(\sum_{i=0}^D w_{ui}^{(1)} x_i\right)\right)\right)$$

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Time series models
9. Multilayer perceptrons
10. Deep learning and transfer learning
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Quiz

- Design a MLP neural network for MNIST digit classification
- Complexity of MLP

# Machine Learning

## Lecture 9: Deep Learning

Zheng-Hua Tan, Sarthak Yadav

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt,sarthaky@es.aau.dk](mailto:zt,sarthaky@es.aau.dk), <http://kom.aau.dk/~zt>

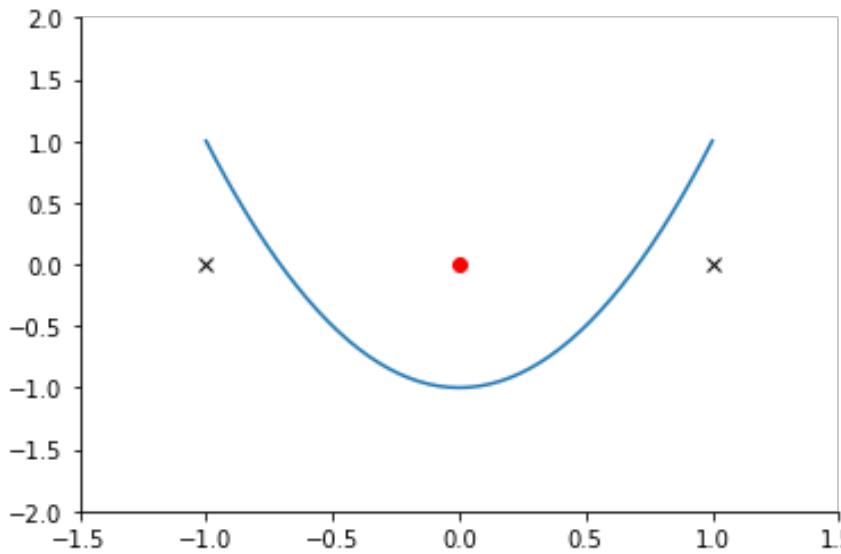
(Some slides courtesy of Dr. Dong Yu, noted as (Yu, 2015) on each slide when used)

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Time series models
9. Multilayer perceptrons
10. Deep learning
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Non-separable data and representation

Nonlinear classifier:  $y = f(x) = 2x^2 - 1$

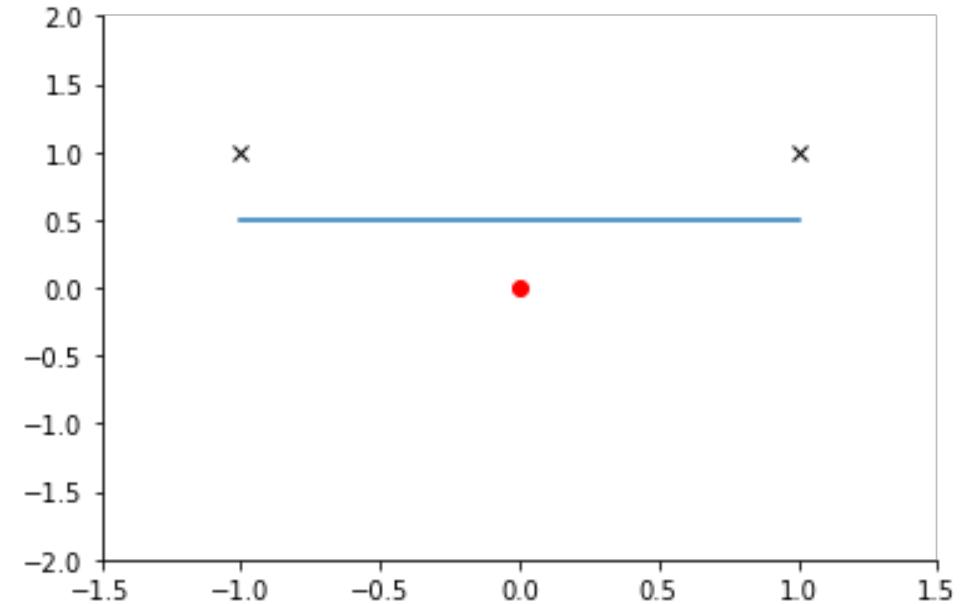


Neural networks are supervised learning methods for **learning representations** - mapping from one space to another!

From nonlinear separable to linear separable

Represent the data in higher dimension from  $[x]$  to  $[x_1, x_2]^T = [x, x^2]^T$

Linear classifier:  $y = f([x_1, x_2]^T) = 2x_2 - 1$



# Feed-forward neural networks

- Hypothesis
  - An intrinsic **non-linear** mapping function  $f(\mathbf{x})$

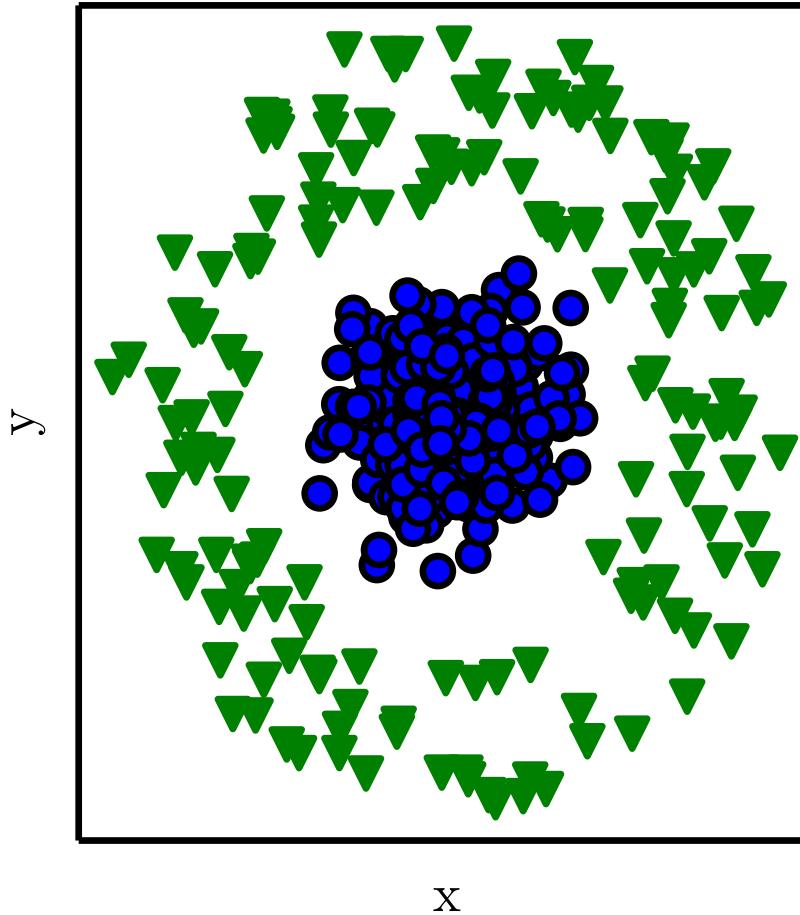
$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$

- In contrast to shallow models, e.g., SVM that is **shallow** with possible **untrainable** nonlinear kernels, neural networks can go **deep**.

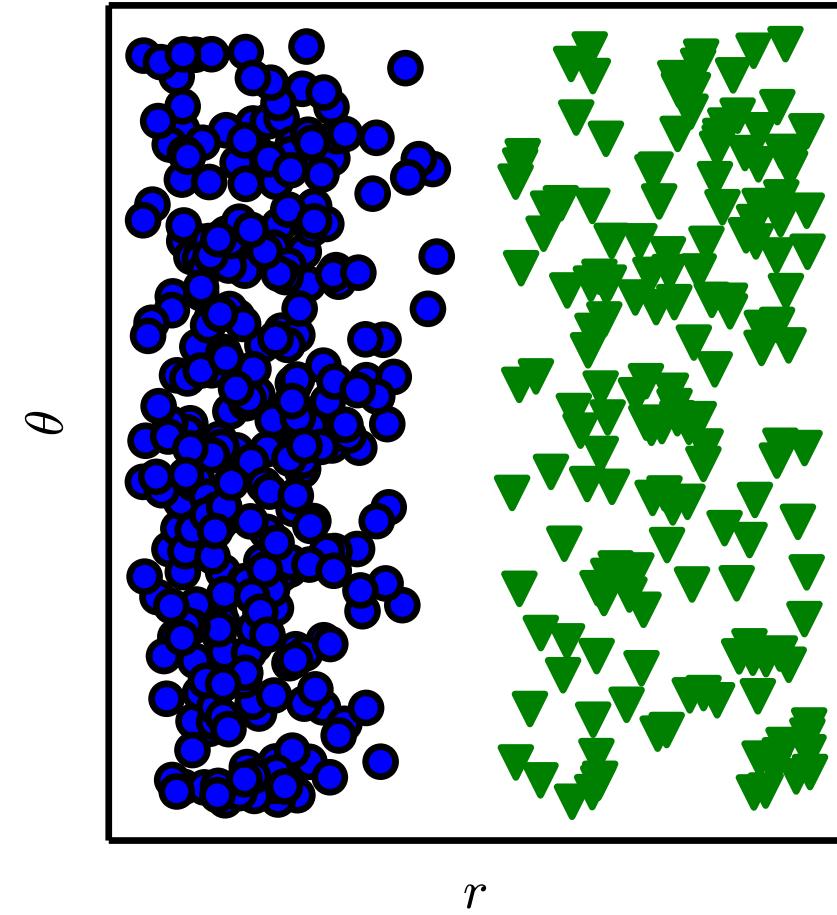
$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(L)} \dots g\left(\sum_{u=0}^U w_{vu}^{(2)} h\left(\sum_{i=0}^D w_{ui}^{(1)} x_i\right)\right)\right)$$

# Representation Matters

Cartesian coordinates

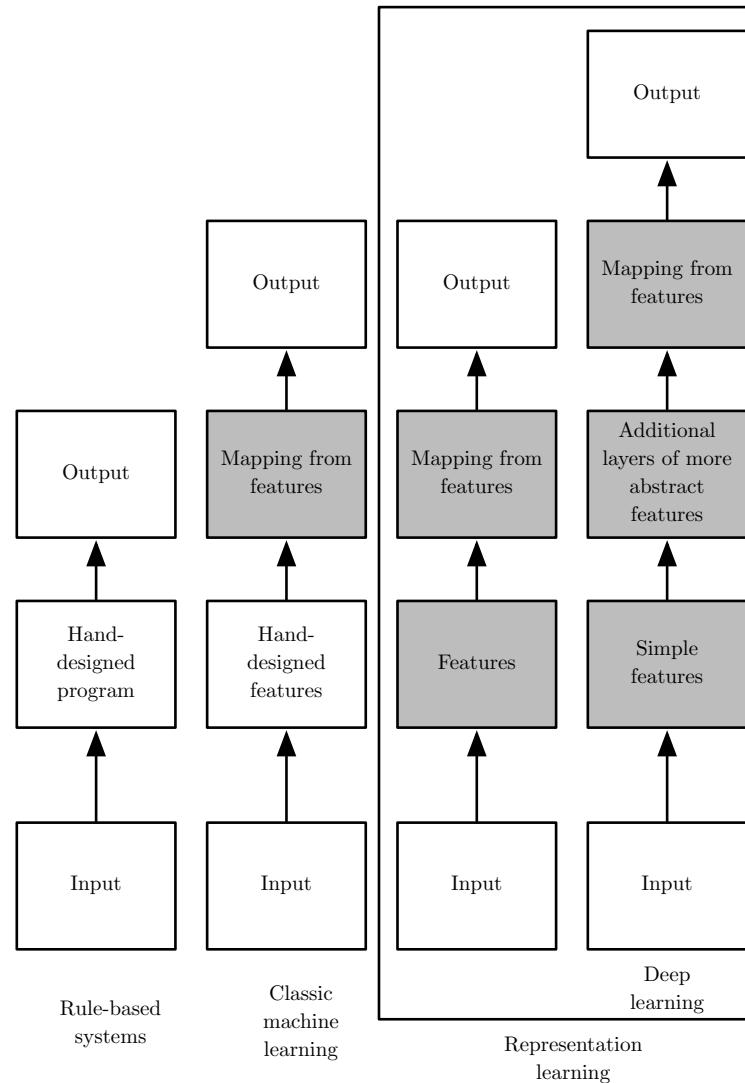


Polar coordinates



(Goodfellow, 2016)

# Learning multiple components



(Goodfellow, 2016)

# Outline

## 1. Brief history and basic concepts

- Intro
- Brief History
- Notable Events
- Basic Concepts

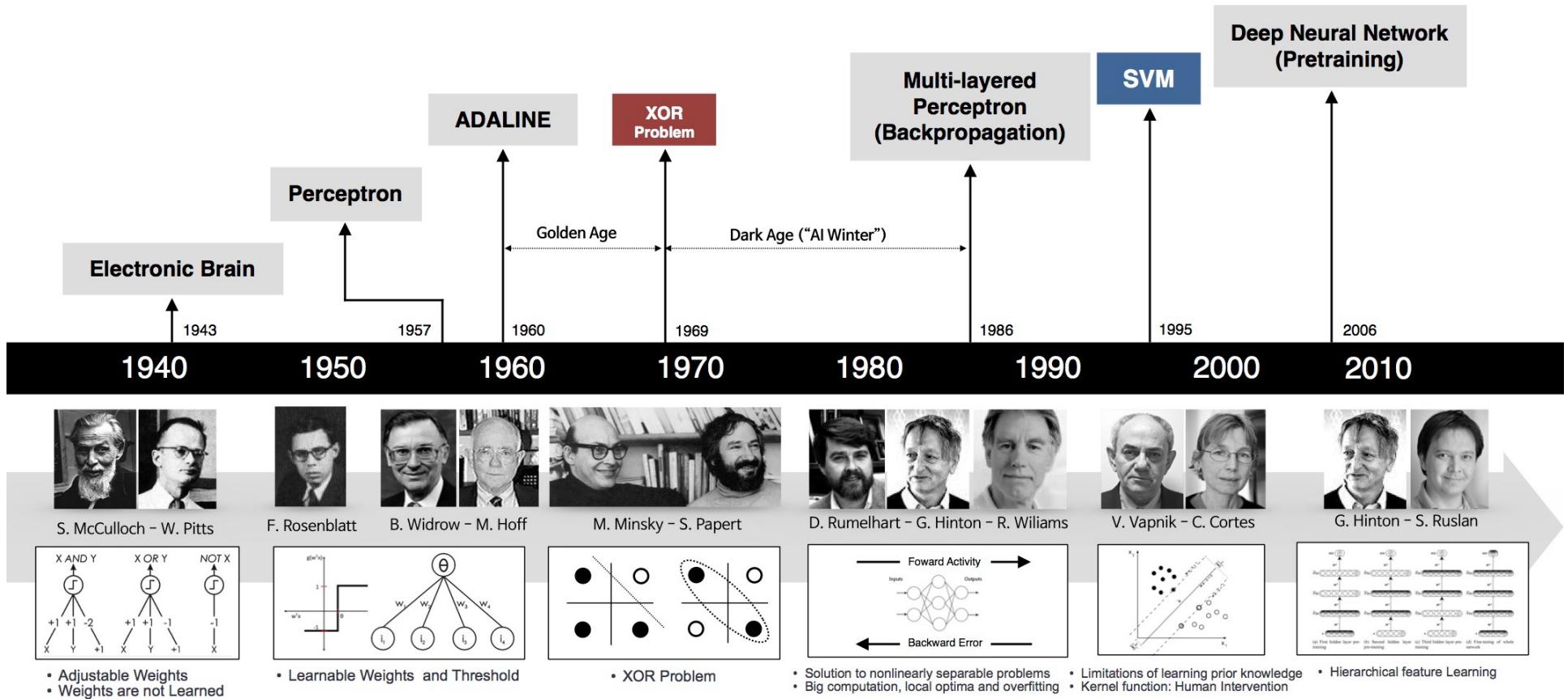
## 2. Key models

- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM) RNNs
- Generative Adversarial Networks (GANs)

# What and Why of Deep Learning

- Deep learning
  - A branch of machine learning
  - It models high-level abstractions in data by
    - using architectures composed of many layers of nonlinear processing units and
    - using supervised or unsupervised learning algorithms
- Why deep learning
  - Big data (labeled and unlabeled)
  - Ever-increasing computation power (Moore's law)
  - Hierarchical structure
  - Superior performance

# History



(VUNO, 2015)

# Deep Learning

(2006 - now)

- Hinton, G. E., Osindero, S. and Teh, Y. (2006), A fast learning algorithm for deep belief nets. *Neural Computation*, 18, pp 1527-1554
  - Proposed to use generative-pretraining before discriminative training (this helped to bring back ANN but not essential)
- Bengio, Y. (2009) Learning deep architectures for AI



G. Hinton



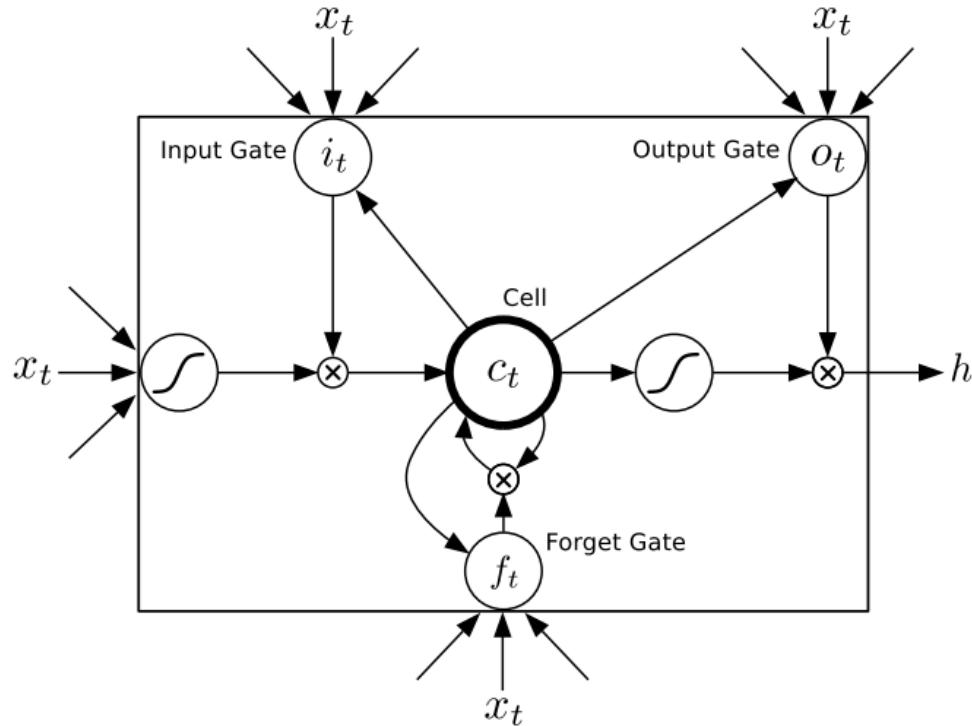
Y. LeCun



Y. Bengio

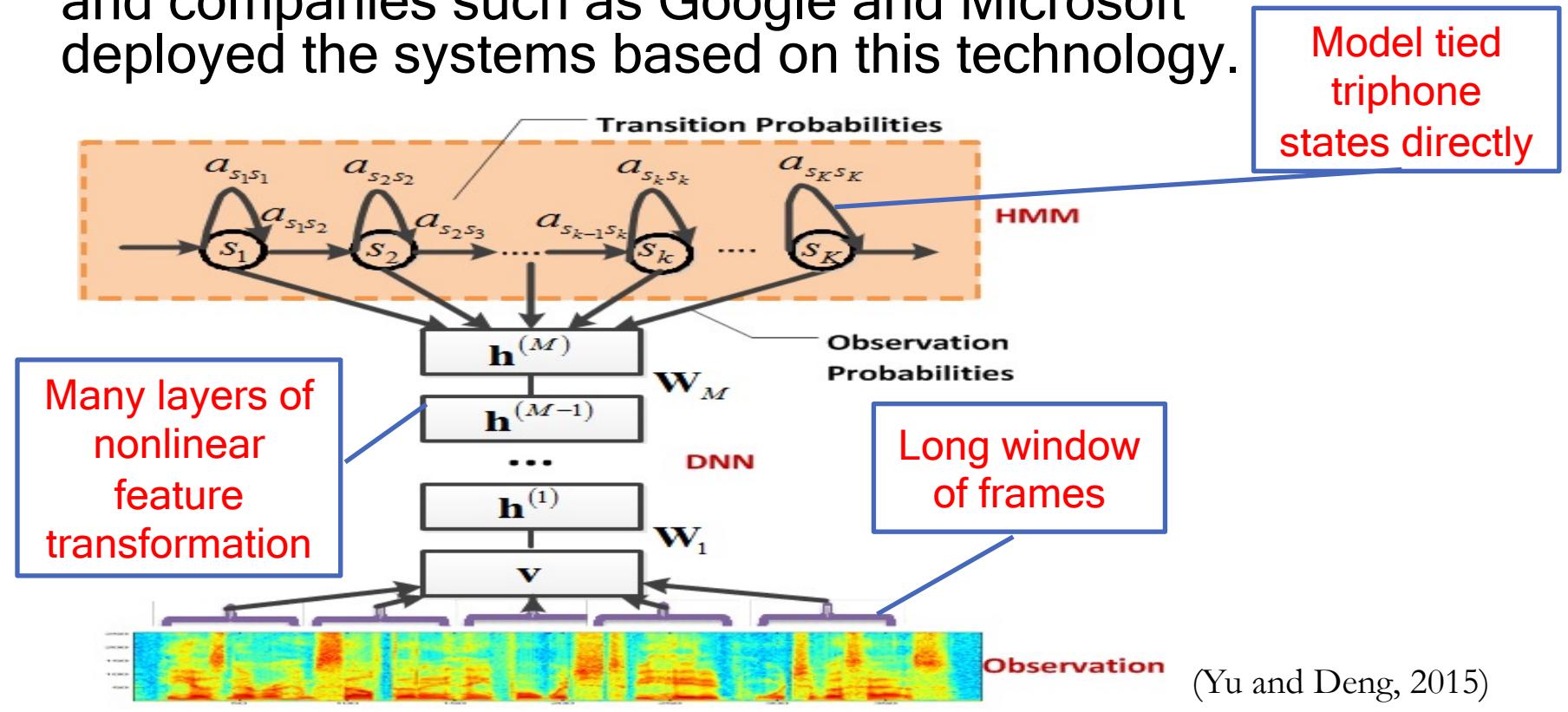
# Deep Learning: Notable Events

- 2009: LSTM won three connected handwriting recognition competitions at ICDAR 2009 (the work was not well known and not related to deep learning until after 2012)



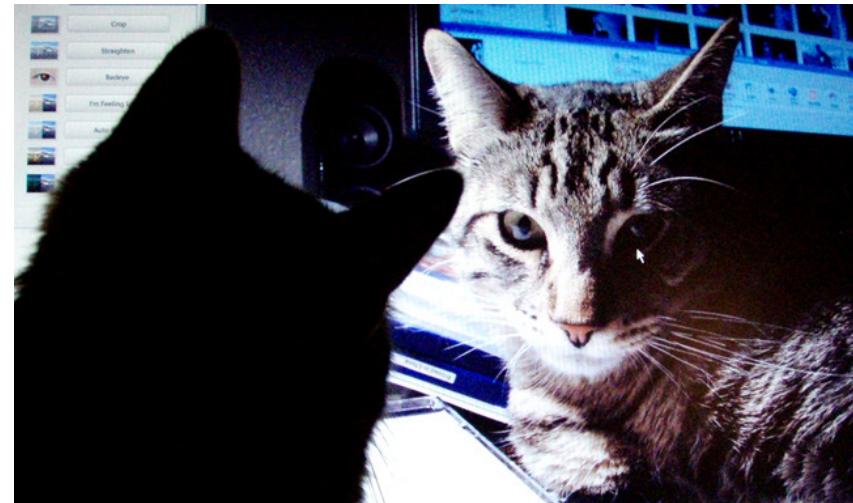
# Deep Learning: Notable Events

- 2010: Microsoft published the first result on large vocabulary speech recognition, which became well-known in 2011 after Switchboard result was published and companies such as Google and Microsoft deployed the systems based on this technology.



# Deep Learning: Notable Events

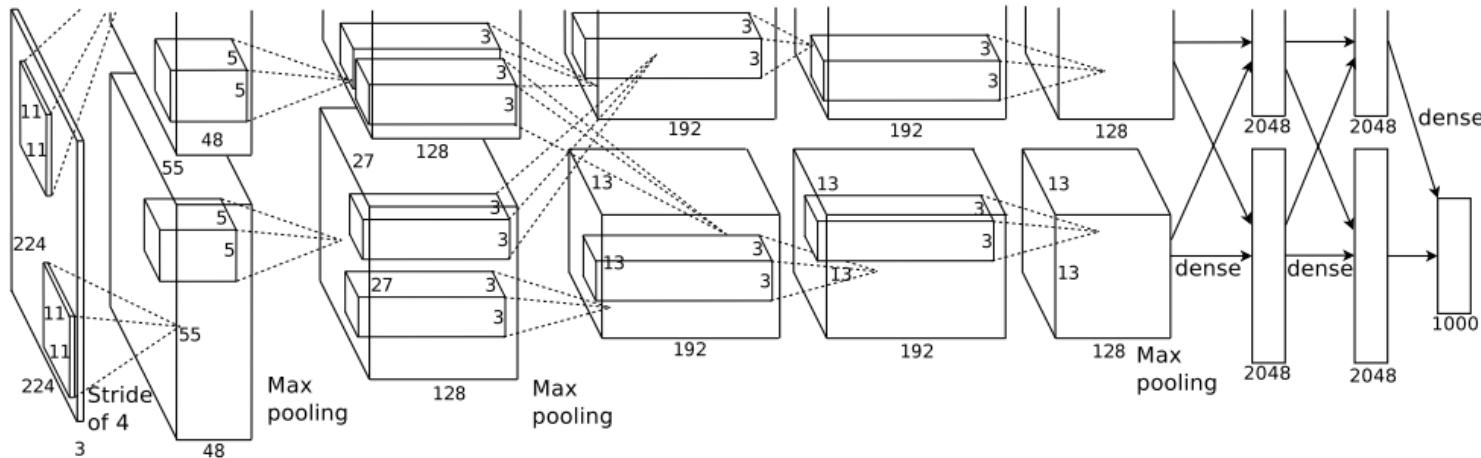
- 2012: Google published results on a large DBN trained with unlabeled images (11m YouTube video thumbnails) and 16,000 CPU cores for three days. The network learns what a cat looks like automatically. No real scientific progress, but further raised awareness of deep learning, especially to the public.



Using sparse deep autoencoder.

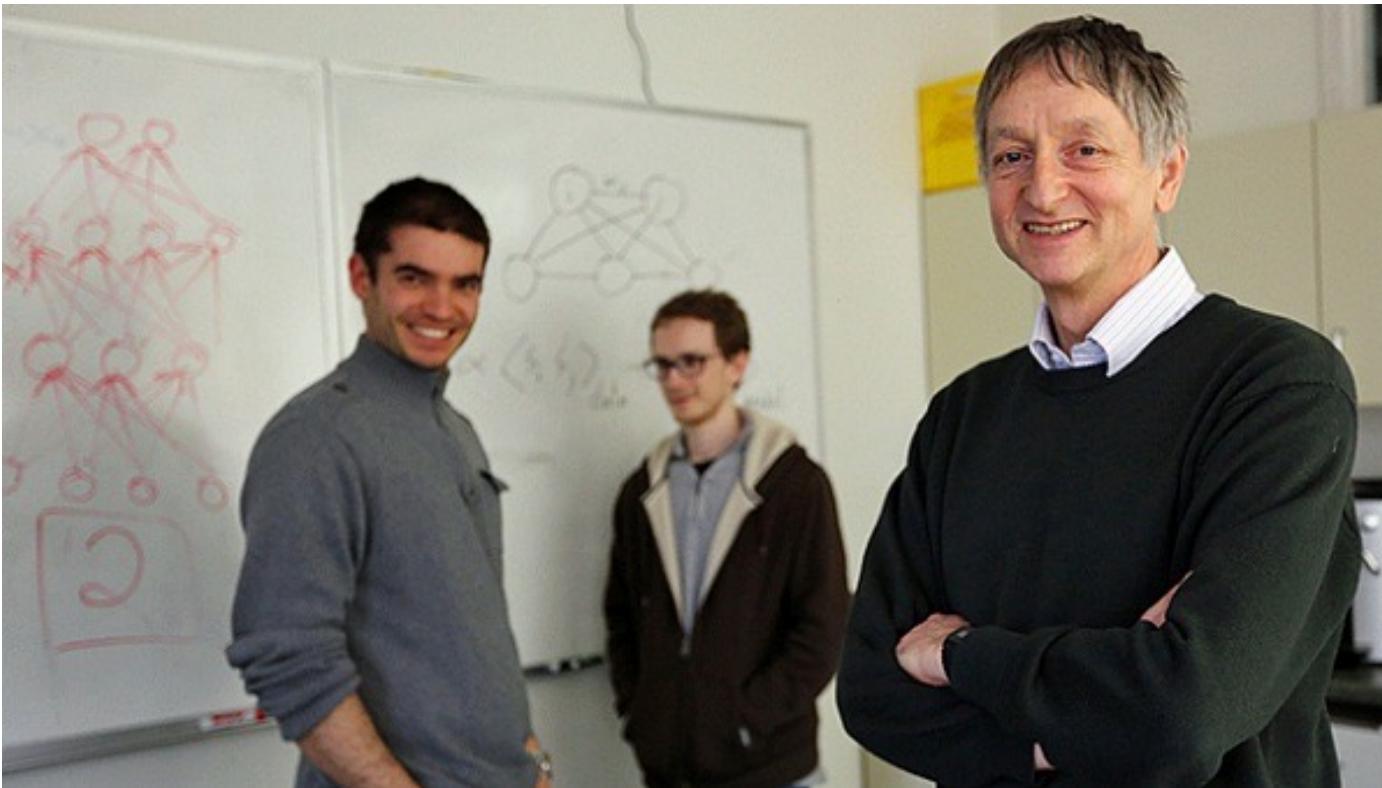
# Deep Learning: Notable Events

- 2012: Hinton's group won Large Scale Visual Recognition Challenge, organized by Li Fei-Fei, with top-5 error rate 16% (compare to the second best system 26%)



# Deep Learning: Notable Events

- 2013: Google bought Hinton's three-person company DNNResearch

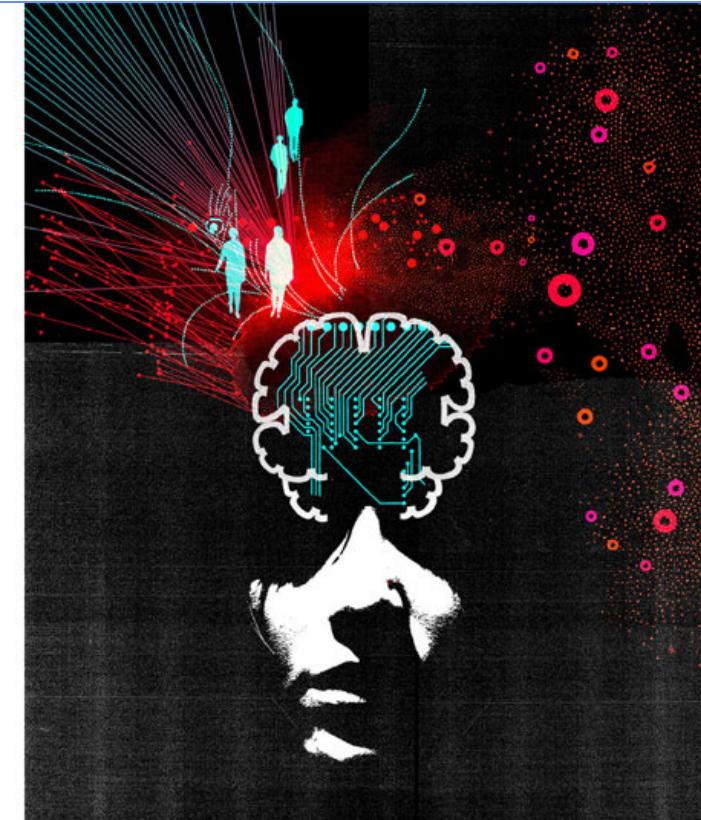


# Deep Learning: Notable Events

- 2013: MIT tech review named Deep Learning one of the 10 breakthroughs

## Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.



# Deep Learning: Notable Events

- 2013: Facebook established artificial intelligence research lab and appointed Prof. Yann LeCun as the head.

**Yann LeCun**  
December 9, 2013 ·

Big news today!

Facebook has created a new research laboratory with the ambitious, long-term goal of bringing about major advances in Artificial Intelligence.

I am thrilled to announce that I have accepted the position of director of this new lab. I will remain a professor at New York University on a part-time basis, and will maintain research and teaching activities at NYU.

Simultaneously, Facebook and New York University's Center for Data Science are entering a partnership to carry out research in data science, machine learning, and AI.

The new AI Group at Facebook will have locations in Menlo Park, CA, in London, UK, and at Facebook's new facility in New York City, one block away from NYU's main campus.

Facebook CEO Mark Zuckerberg, CTO Michael Schroepfer and I are at the Neural Information Processing Systems Conference in Lake Tahoe today. Mark will announce the news during his presentation at the NIPS Workshop on Deep Learning later today.

And we are hiring!

# Deep Learning: Notable Events

- 2014: Google bought London-based artificial intelligence company DeepMind for over \$500M



# Deep Learning: Notable Events

- 2015: Achieved state-of-the-art results in tasks such as machine translation, image captioning, and image generation

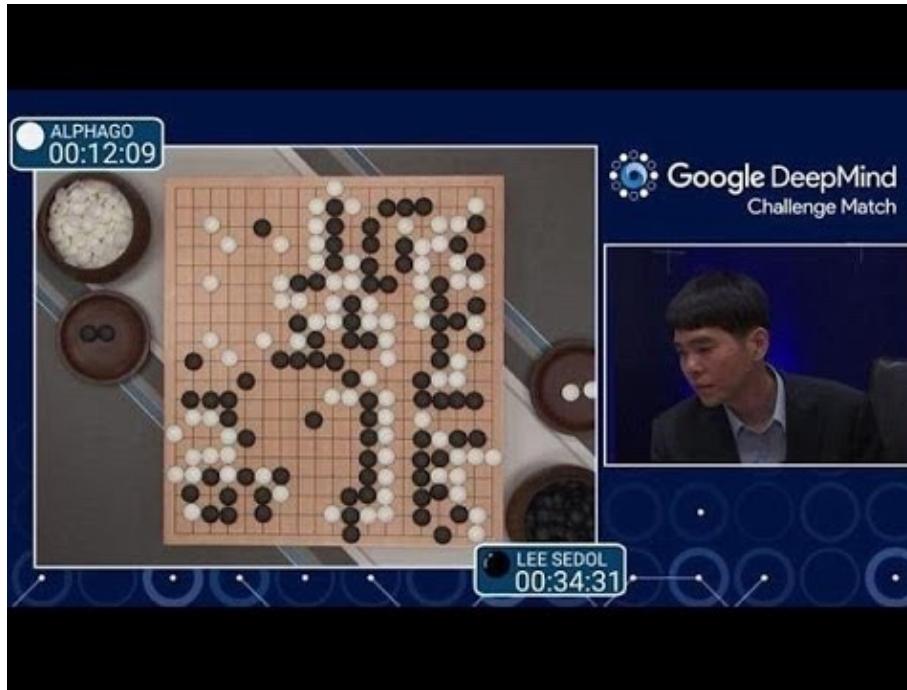


a man riding a  
skateboard  
down  
a street

(Yu, 2015)

# Deep Learning: Notable Events

- 2016: AlphaGo
- 2017: AlphaGo Zero

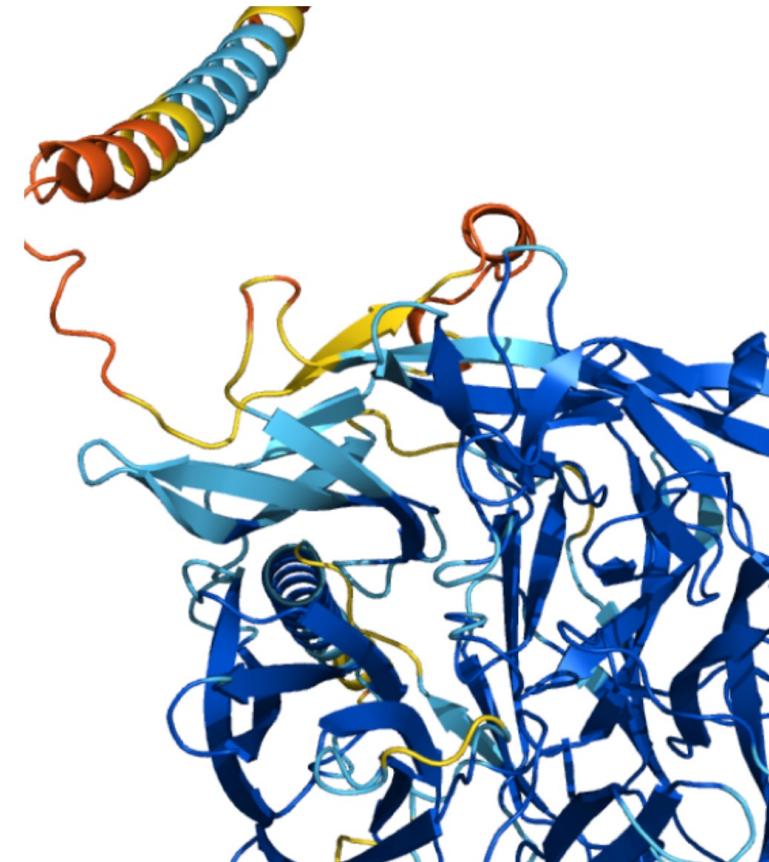


# Deep Learning: Notable Events

- 2018: AlphaFold- breakthrough in protein folding

AlphaFold is an AI system developed by [Google DeepMind](#) that predicts a protein's 3D structure from its amino acid sequence. It regularly achieves accuracy competitive with experiment.

Google DeepMind and EMBL's European Bioinformatics Institute ([EMBL-EBI](#)) have partnered to create AlphaFold DB to make these predictions freely available to the scientific community. The latest database release contains over 200 million entries, providing broad coverage of [UniProt](#) (the standard repository of protein sequences and annotations). We provide individual [downloads](#) for the human proteome and for the proteomes of 47 other key organisms important in research and global health. We also provide a download for the manually curated subset of UniProt ([Swiss-Prot](#)).



Q8I3H7: May protect the malaria parasite against attack by the immune system.  
Mean pLDDT 85.57.

[View protein](#)

# Deep Learning: Notable Events

- 2022: ChatGPT

why does winching up an anchor raise the anchor instead of pulling the ship down

Bing, 6:10 PM

Winching up an anchor raises the anchor instead of pulling the ship down because of several factors, such as:

- The design of the windlass, which is the device that operates the anchor cable. The windlass is not meant to haul the ship, but only to lift the anchor and reduce the tension in the cable. The windlass motor is the "weak link" in the anchoring system and can be damaged by excessive force 1. Therefore, the windlass is used carefully and slowly, and the ship is maneuvered towards the anchor to assist the recovery process 2.
- The weight of the anchor and the cable, which are much smaller than the weight of the ship. The anchor and the cable together may weigh a few tons, while a typical ship may weigh a few hundred or thousand tons. Therefore, the force exerted by the windlass on the anchor is not enough to overcome the inertia and buoyancy of the ship.
- The friction between the anchor and the sea bottom, which resists the pull of the windlass. The anchor works by digging into the sea bottom with its fluke or pointed projection, creating a strong grip. To break out the anchor, the chain has to be locked in a chain stopper and the ship has to move in a way that pulls out the anchor from the sea bottom 2. If the anchor is stuck or buried too deep, it may be difficult to retrieve it with the windlass alone 3.
- The angle of the cable, which affects the direction of the force applied by the windlass. The cable should be vertical or "up and down" when heaving up the anchor, so that the force is directed upwards. If the cable is inclined or horizontal, it means that the ship is too far from the anchor and that the force is



# Deep Learning: Notable Events

- 2022: Image generation from natural language prompts (DALL-E, Stable Diffusion, Midjourney)



"A harlequin dancing in a field on a sunny beautiful day in a beau..."

# Deep Learning: Notable Events

- 2023: Video generation from natural language prompts (SORA)



Prompt: A movie trailer featuring the adventures of the 30 year old space man wearing a red wool knitted motorcycle helmet, blue sky, salt desert, cinematic style, shot on 35mm film, vivid colors.

# Why DL Became Popular Only Now

- We have significantly (100x than 20 years ago) increased the computing power
  - Allows us to increase the model complexity and power significantly, e.g., with more layers and more units in each layer
  - Allows us to adjust model structure and learning procedure much more frequently
- We have tremendous amount data since we are now in the big-data era
  - Allows us to generalize the model well to unseen scenarios since you require more interpolation (relatively reliable) than extrapolation (often questionable)

(Yu, 2015)

# What Is Deep Learning?

- **Definition 1:** A class of machine learning techniques that exploit **many layers** of **non-linear** information processing for **feature extraction** and transformation and for pattern analysis and classification.
- **Definition 2:** A sub-field within machine learning that is based on algorithms for **learning multiple levels of representation** in order to model complex relationships among data. Higher-level features and concepts are thus defined in terms of lower-level ones, and such a hierarchy of features is called a deep architecture.

(Yu, 2015)

# Why Deep?

- In 1980s we already know we can approximate any function as close as we want with shallow architecture. Why deep?
  - E.g., kernel machines and single hidden layer neural network are universal approximators
- Deep machines
  - Can represent more complex functions with less parameters
  - Can learn the same function with less training data by reusing low-level feature detectors
  - Can learn more invariant high-level features

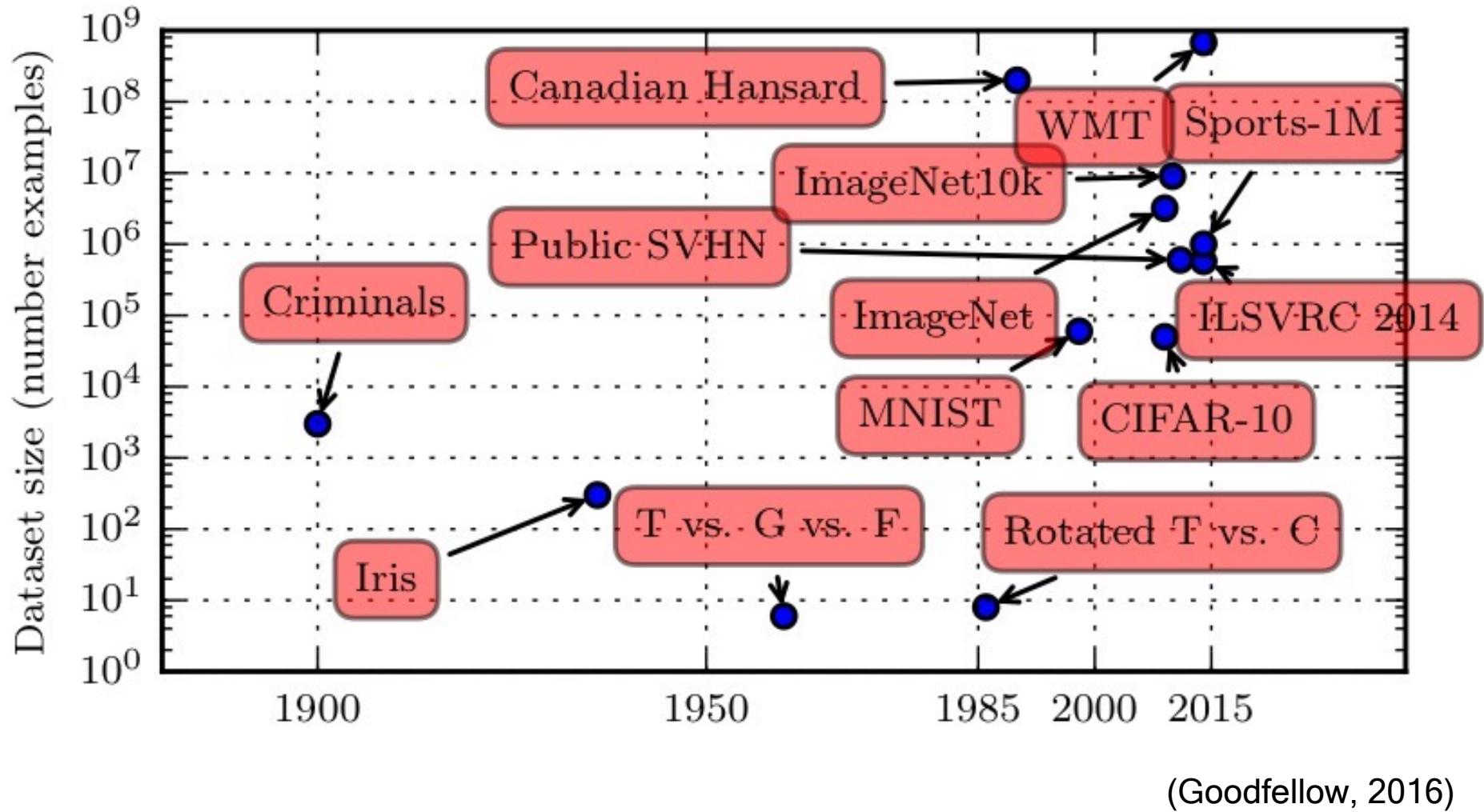
(Yu, 2015)

# Traditional Machine Learning

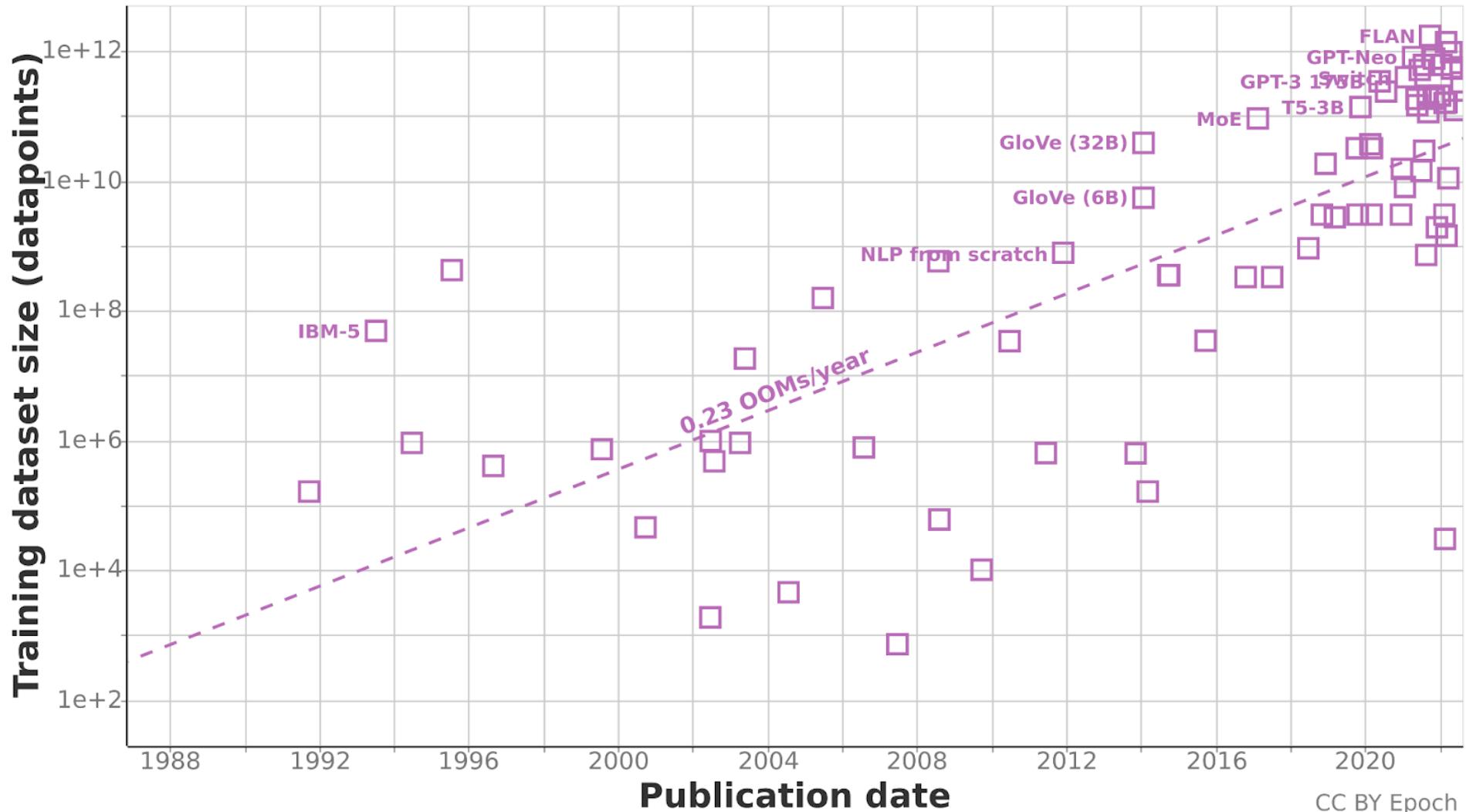
- Feature engineering
  - Design features that work well on a specific task
  - Requires domain knowledge, error analysis, and trial and error
- Simple model
  - Linear model
  - Support vector machine (SVM)
  - Gaussian mixture model (GMM)
  - Single-hidden layer neural network (SHLNN)
- Manually designing good features that can work effectively with simple models

(Yu, 2015)

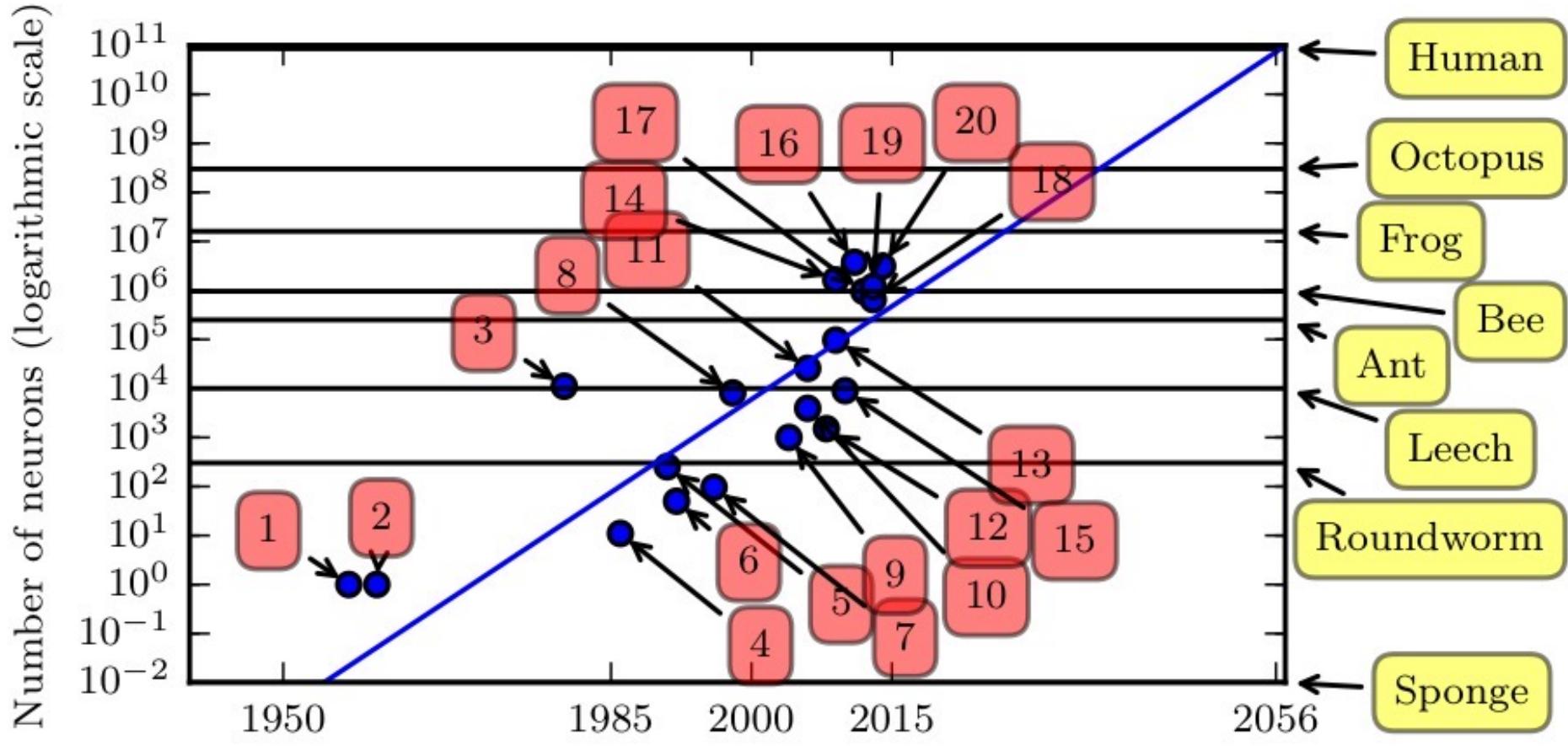
# Historical Trends: Growing Datasets



# Historical Trends: Growing Datasets



# Number of neurons

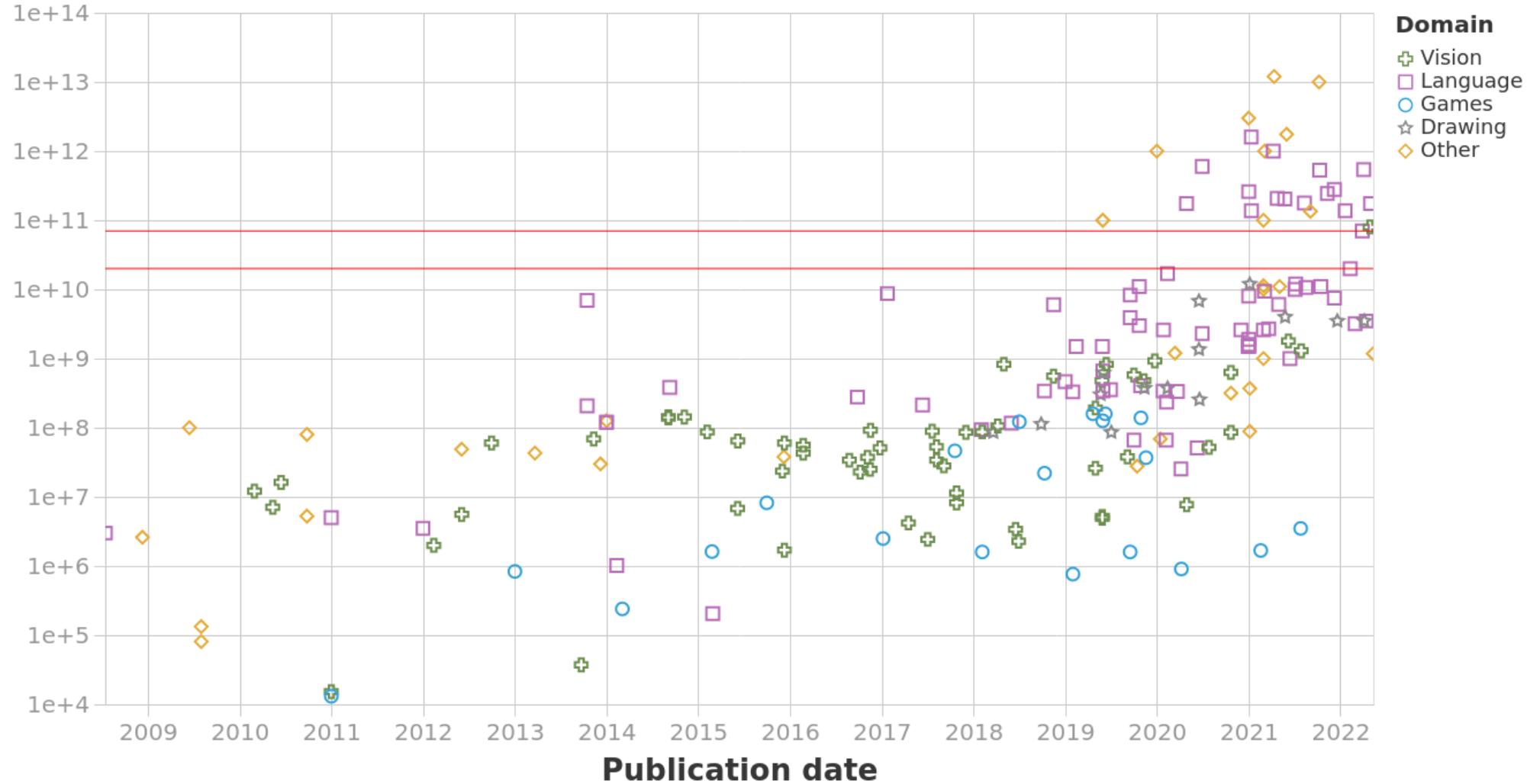


(Goodfellow, 2016)

# Number of neurons

Parameters of milestone Machine Learning systems over time

n = 203



<https://epochai.org/blog/machine-learning-model-sizes-and-the-parameter-gap>

# Course Outline

## 1. Brief history and basic concepts

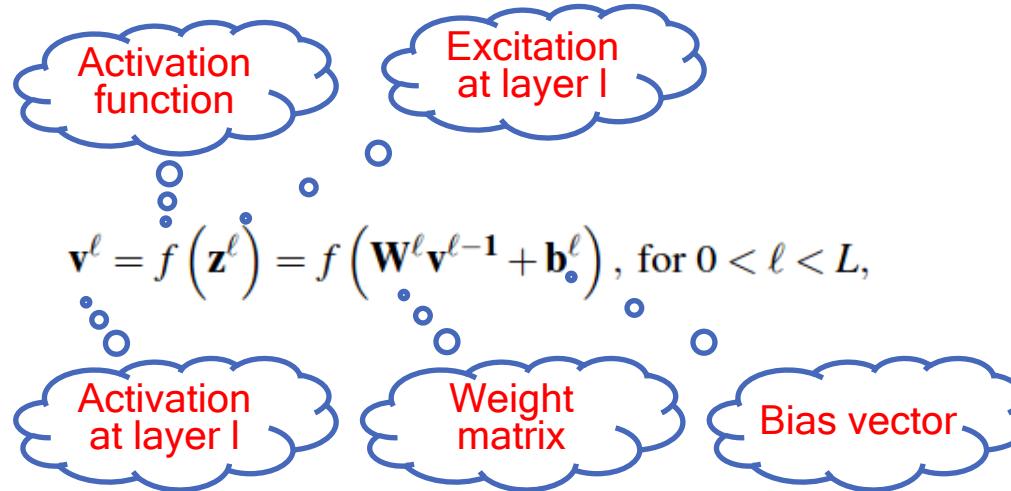
- Intro
- Brief History
- Notable Events
- Basic Concepts

## 2. Key models

- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM) RNNs
- Generative Adversarial Networks (GANs)

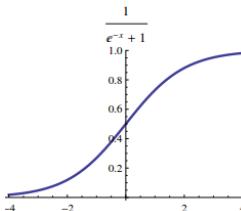
# Deep Neural Network

- A fancy name for multi-layer perceptron (MLP) with many hidden layers.

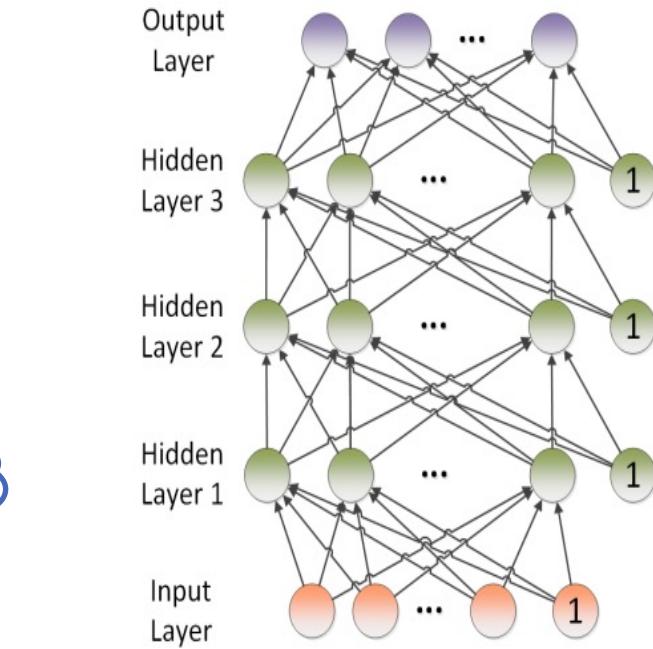
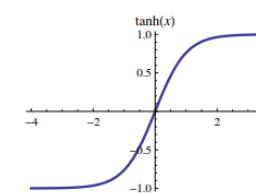


- Typical activation functions

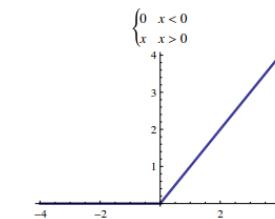
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



$$ReLU(z) = \max(0, z)$$



(Goodfellow, 2016)

# Deep Neural Network

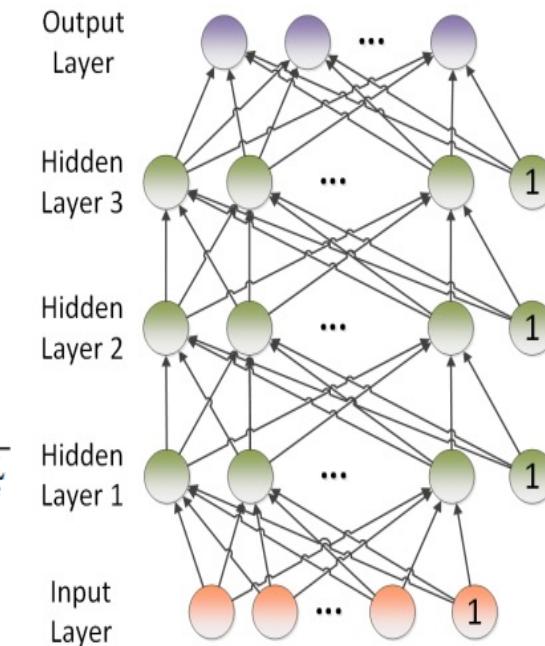
- Output layer is often a softmax layer

i-th unit of the output      i-th element of the excitation at the output layer

$$v_i^L = P_{dnn}(i|\mathbf{o}) = \text{softmax}_i(\mathbf{z}^L) = \frac{e^{z_i^L}}{\sum_{j=1}^C e^{z_j^L}}$$

- Guarantees that the output is a probability distribution

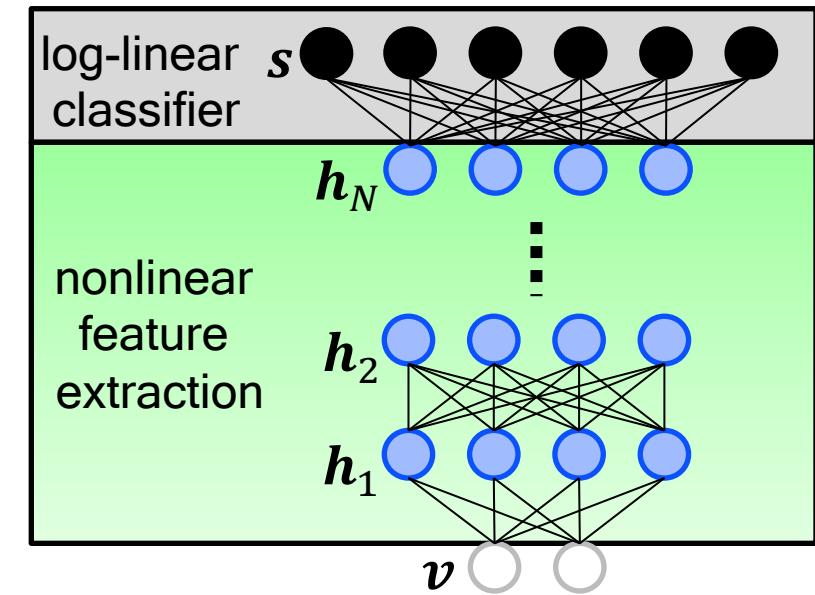
$$v_i^L \geq 0 \text{ and } \sum_{i=1}^C v_i^L = 1$$



(Yu, 2015)

# Why DNNs Perform So Well

- It's a combination of nonlinear feature extraction and log-linear classifier
- Many simple nonlinearities combine to form arbitrarily complex nonlinearities for better feature transformation
- Joint feature learning & classifier design
- Lower-layer feature representations are exploited by the higher layer feature detectors
- Features at higher layers more **invariant** and **discriminative** than at lower layers



(Yu, 2015)

# Limitations of DNNs

- DNNs obtain features, which are **discriminative** and **invariant**, through many layers of non-linear transformations with supervision.
  - **Discriminative**: transfer the raw feature non-linearly into a higher dimensional space in which things that were non-separable become separable
  - **Invariant**: pool or aggregate features in the new space to introduce invariance
- However,
  - DNNs **do not explicitly** exploit known structures (e.g., translational variability) in the input data
  - DNNs **do not explicitly** apply operations that reduces variability (e.g., pooling and aggregation)
- Can we build these properties directly in the neural networks?
  - Yes, e.g., convolutional neural networks (CNNs)

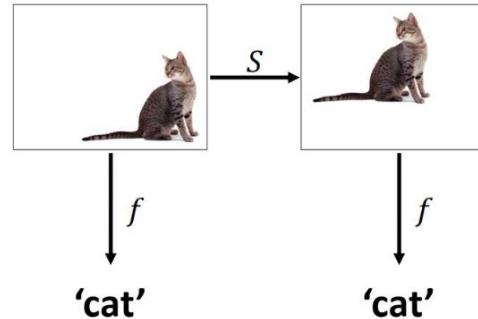
(Yu, 2015)

# CONVOLUTIONAL NEURAL NETWORKS

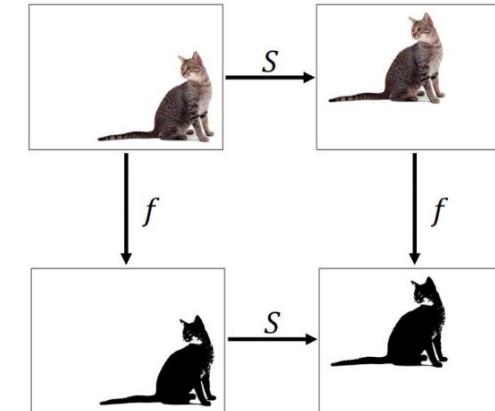
# Invariance vs equivariance

## Invariance vs equivariance

Invariance



Equivariance



Deep Learning – Bernhard Kainz

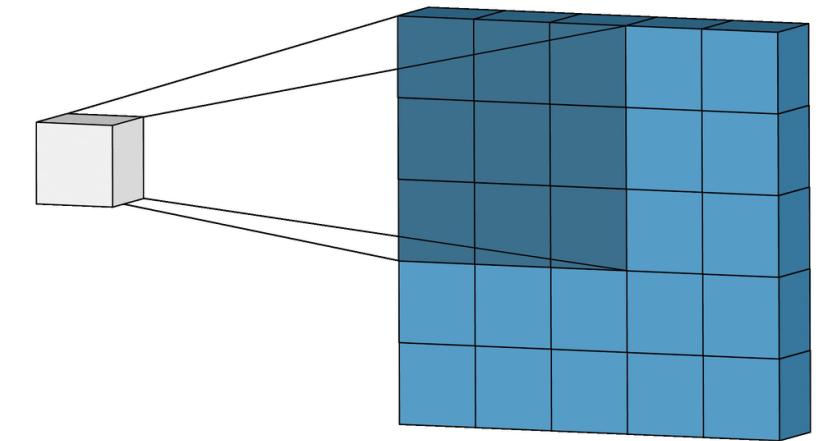
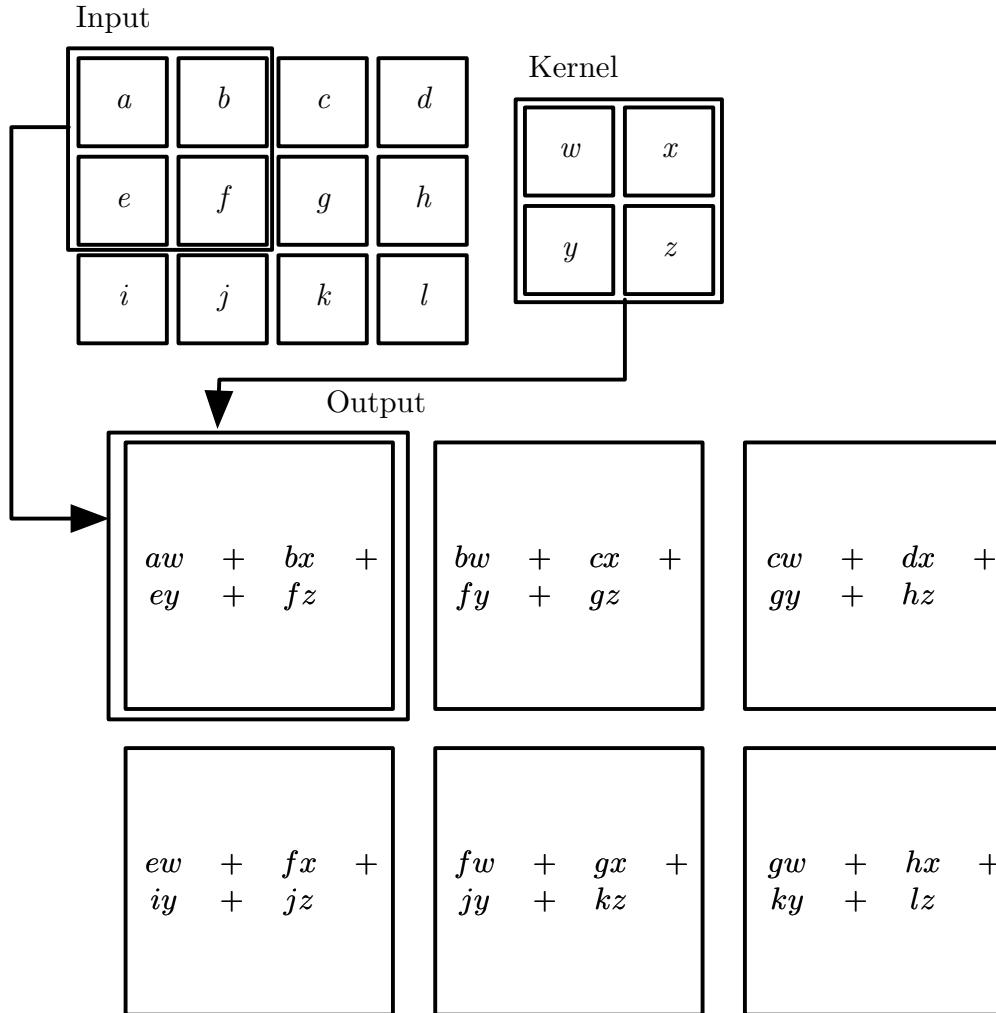
<https://wp.doc.ic.ac.uk/bkainz/teaching/70010-deep-learning/>

# Convolutional Networks

- Scale up neural networks to process very large images / video sequences
  - Sparse connections
  - Parameter sharing
- Automatically generalize across spatial translations of inputs
- Applicable to any input that is laid out on a grid (1-D, 2-D, 3-D, ...)
- Replace matrix multiplication in neural nets with convolution
- Everything else stays the same (Maximum likelihood, Backpropagation, etc.)

(Goodfellow, 2016)

# 2D Convolution



<https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1>

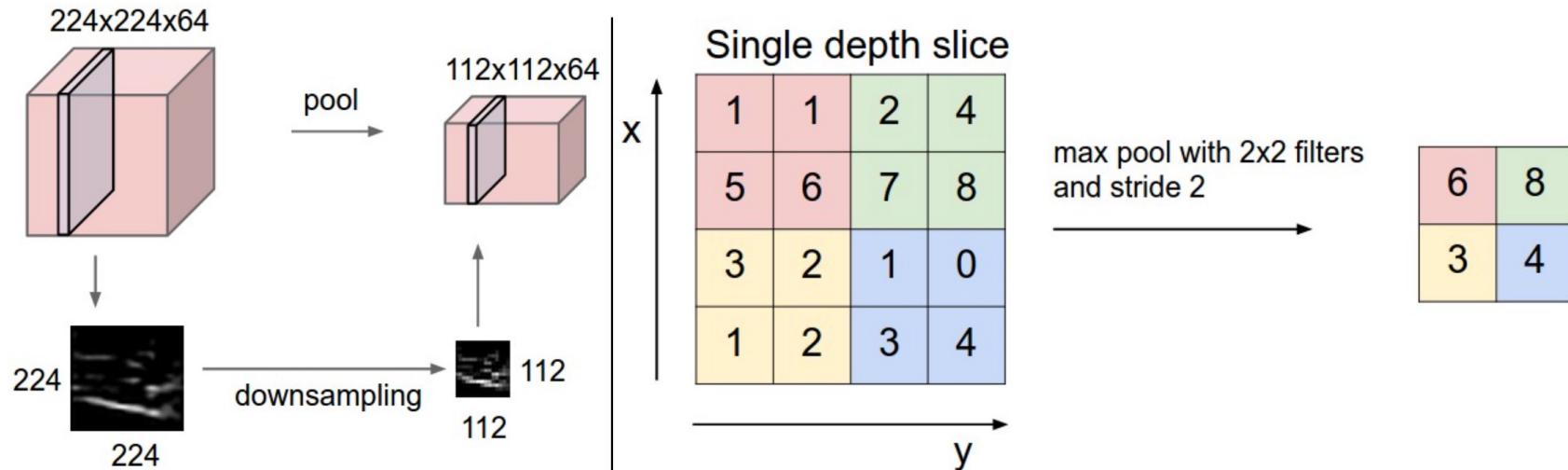
(Goodfellow, 2016)

# 2D Convolution

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) is the result of performing a valid convolution of the  $d$ -th filter over the input volume with a stride of  $S$ , and then offset by  $d$ -th bias.

<https://cs231n.github.io/convolutional-networks/>

# Max Pooling and Invariant to Translation

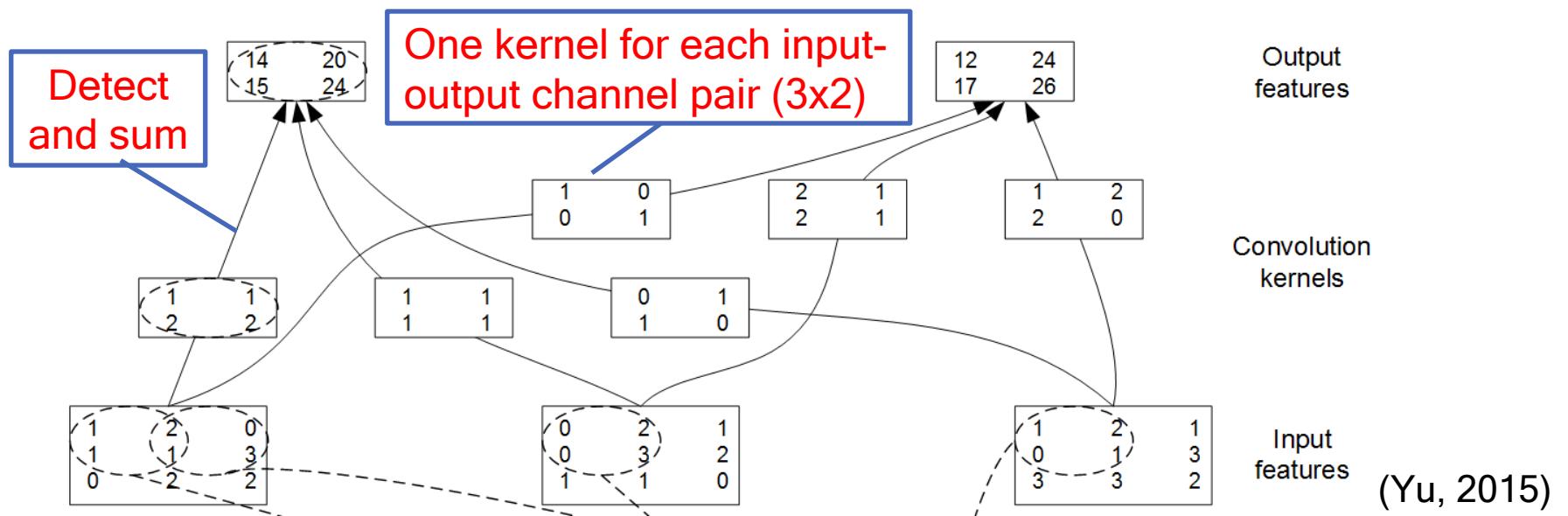


Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. Left: In this example, the input volume of size  $[224 \times 224 \times 64]$  is pooled with filter size 2, stride 2 into output volume of size  $[112 \times 112 \times 64]$ . Notice that the volume depth is preserved. Right: The most common downsampling operation is max, giving rise to max pooling, here shown with a stride of 2. That is, each max is taken over 4 numbers (little  $2 \times 2$  square).

<https://cs231n.github.io/convolutional-networks/>

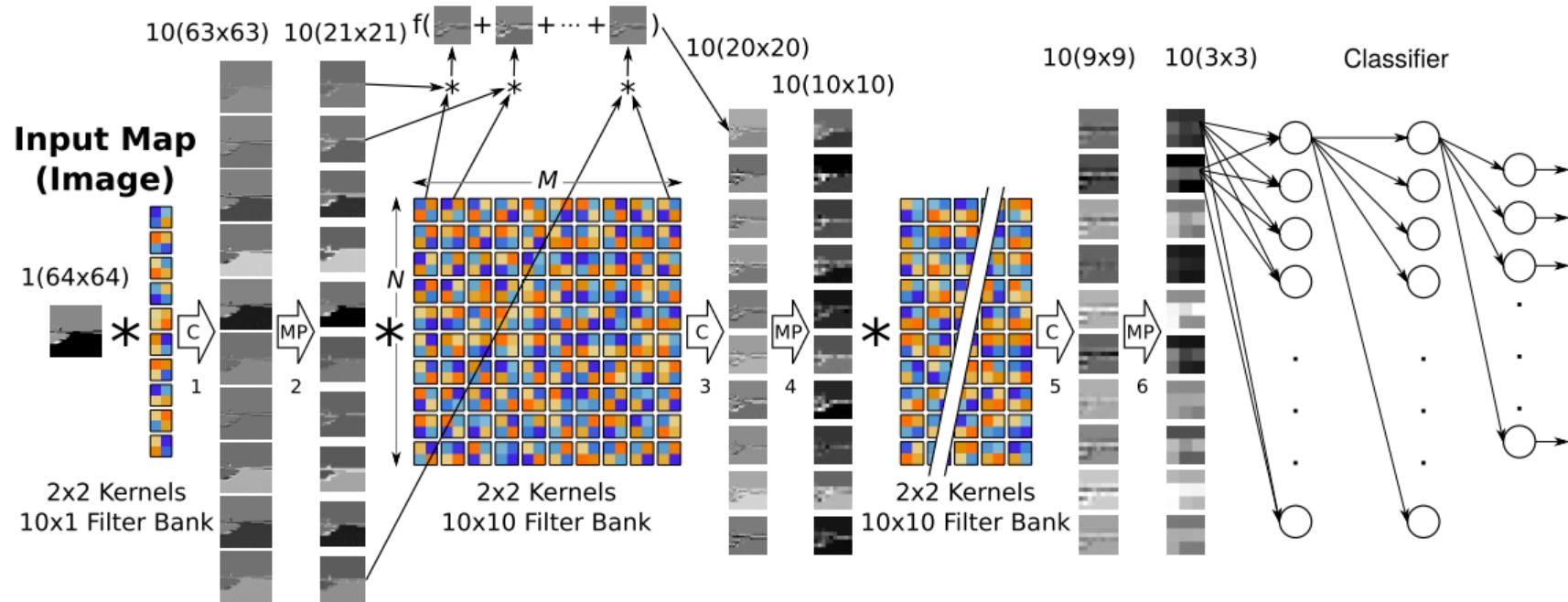
# Convolutional Neural Networks

- Explicitly models translational variability and enables shift invariance
  - Shared local filters (weights) tiled across image to detect the same pattern at different locations
  - Sub-sampling through pooling (max, average, or other) to reduce variability



# Deep CNN for Classification

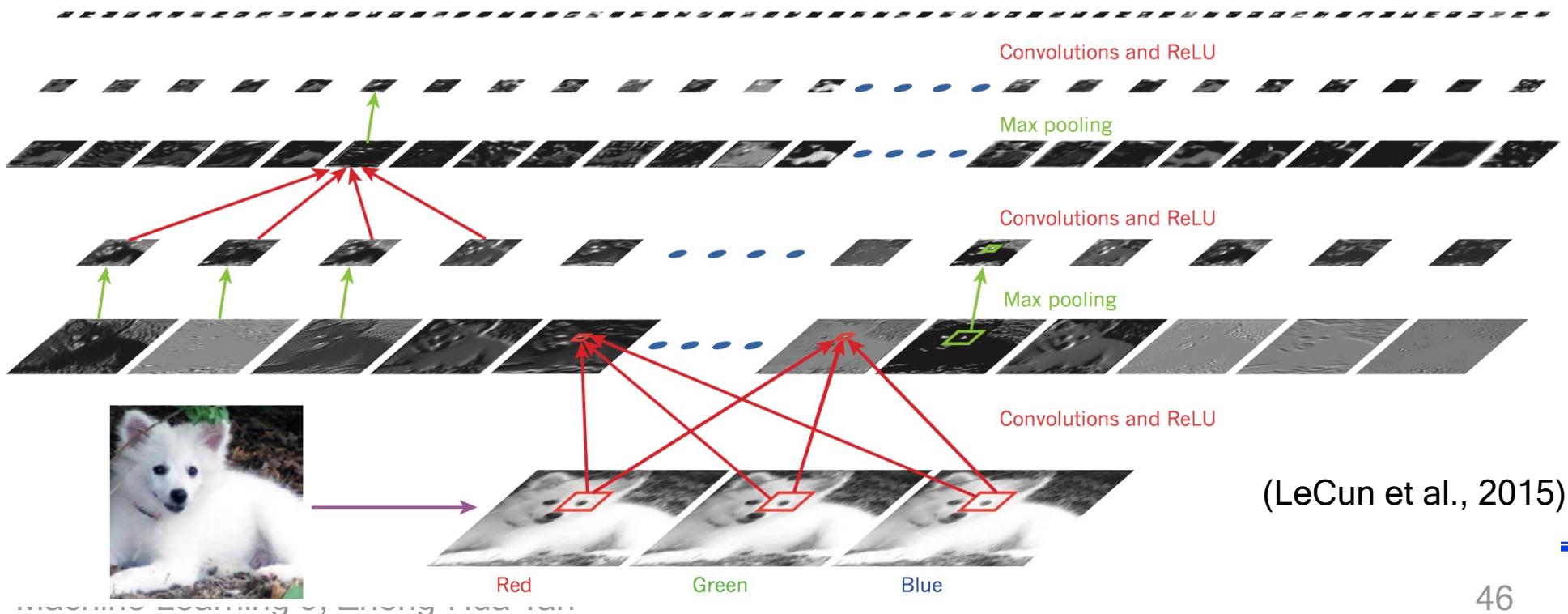
- Alternate between convolutional and pooling layers
- Convolutional layers: translation equivariance
- Pooling layers: translation invariance
- Stack a DNN on top of the last pooling layer for classification



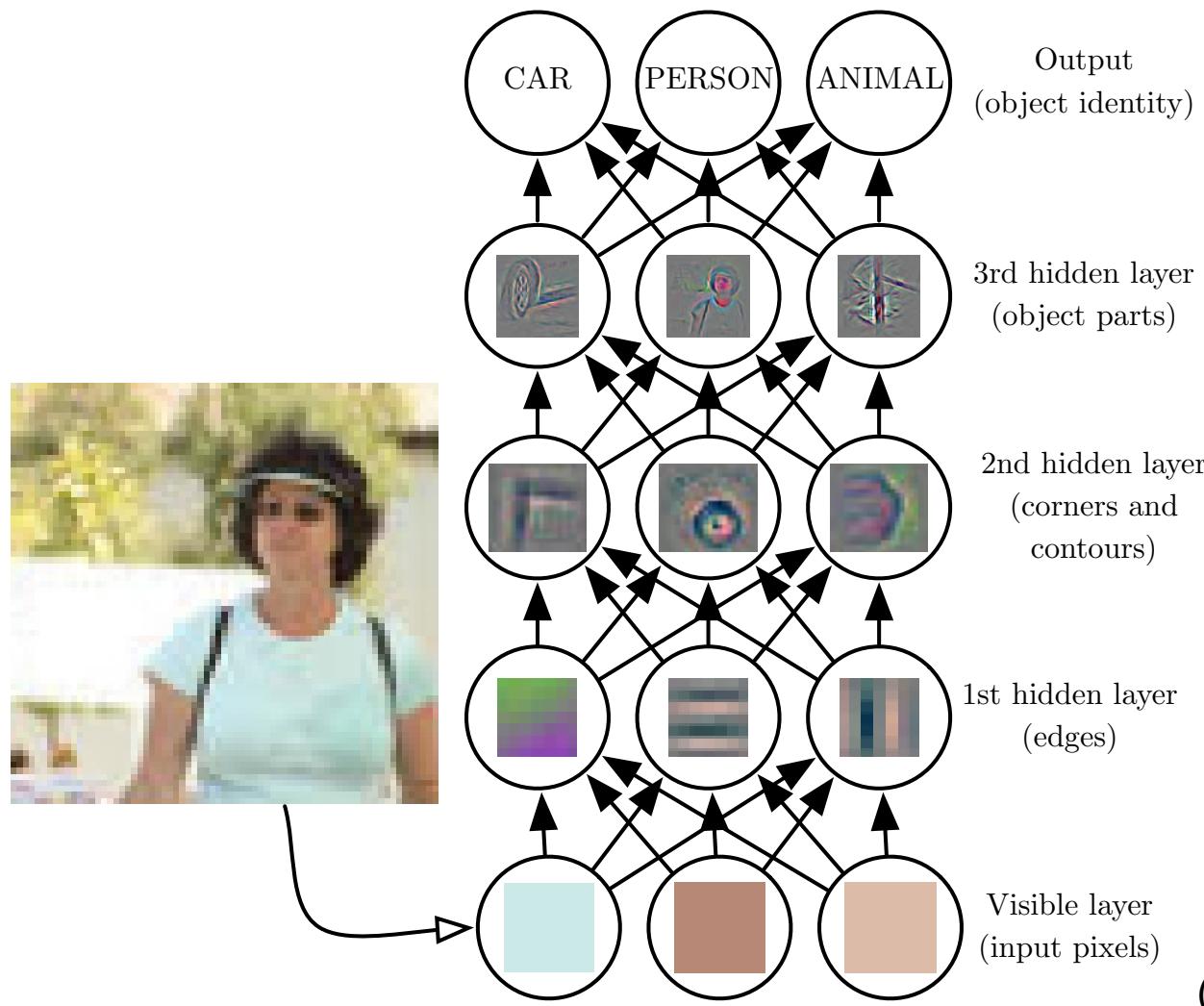
(Koutnik, 2015)

# Inside a Convolutional Network

- The outputs (not the filters) of each layer of a typical convolutional network architecture applied to the image of a Samoyed dog (bottom left) with RGB inputs (bottom right).
- Each rectangular image is a feature map

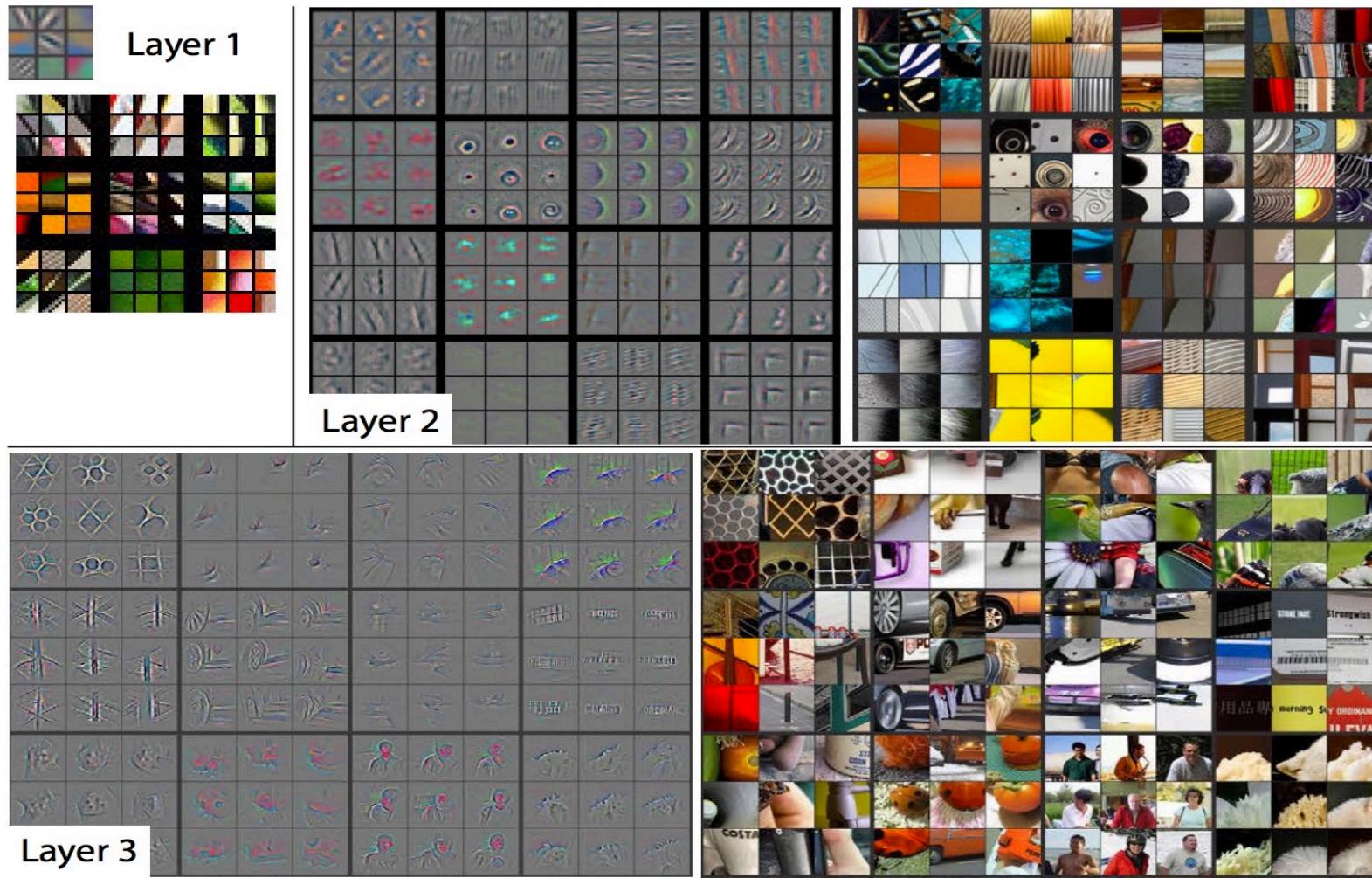


# Depth (and it matters)



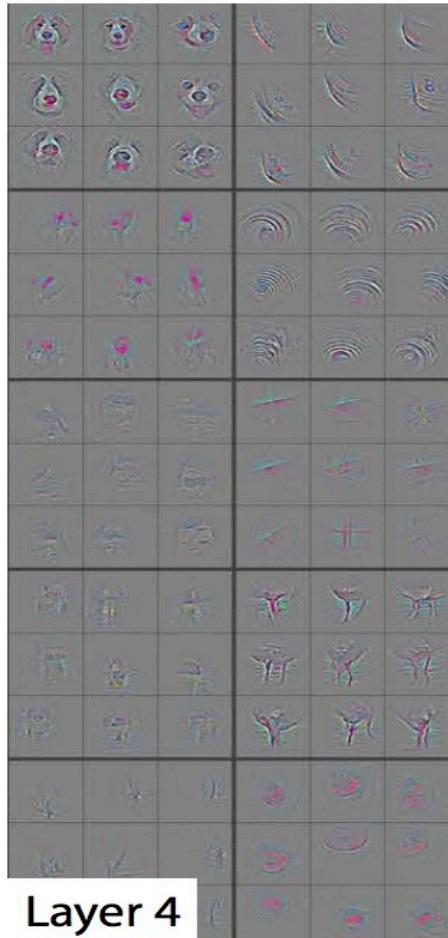
# Feature Learning - Examples

(Zeiler and Fergus, 2014)



# Feature Learning - Examples

(Zeiler and Fergus, 2014)



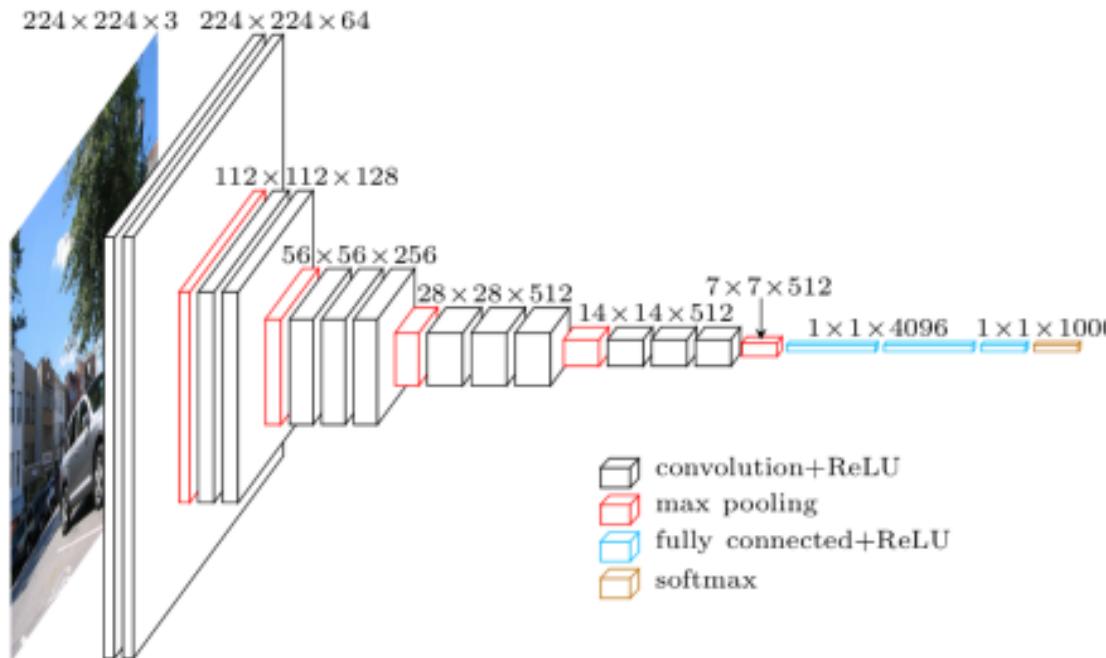
Layer 4



Layer 5



# Popular CNNs: VGG



Stack of convolutional layers (followed by relu activation) and maxpooling layers of different sizes.

# Residual Networks

- Vanishing gradient problem:
  - As you backpropagate, gradient becomes smaller
  - You can't just keep increasing depth, esp with range-squashing activation functions (like sigmoid and tanh)
  - ReLU and other piecewise linear activation functions help: Function is basically  $\max(0, x)$
- Enter Residual Networks

# Residual Networks: Deep Residual Network for Image Recognition

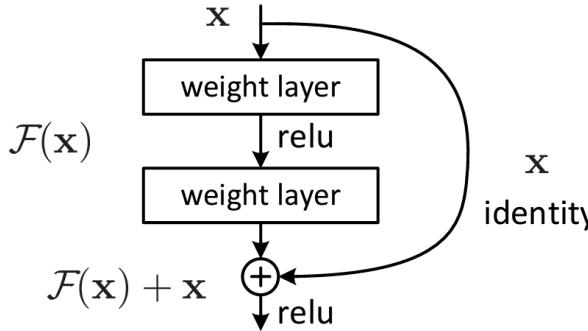
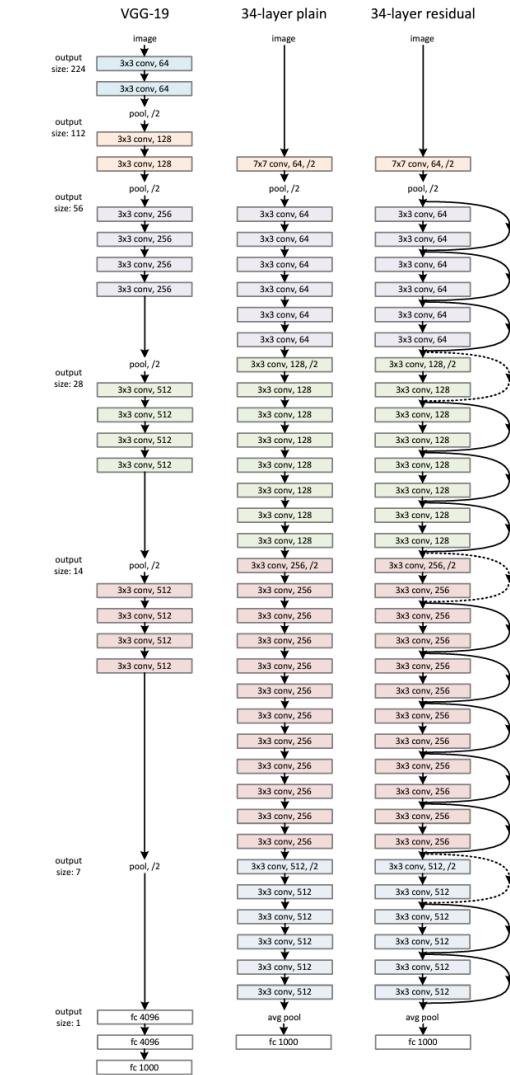


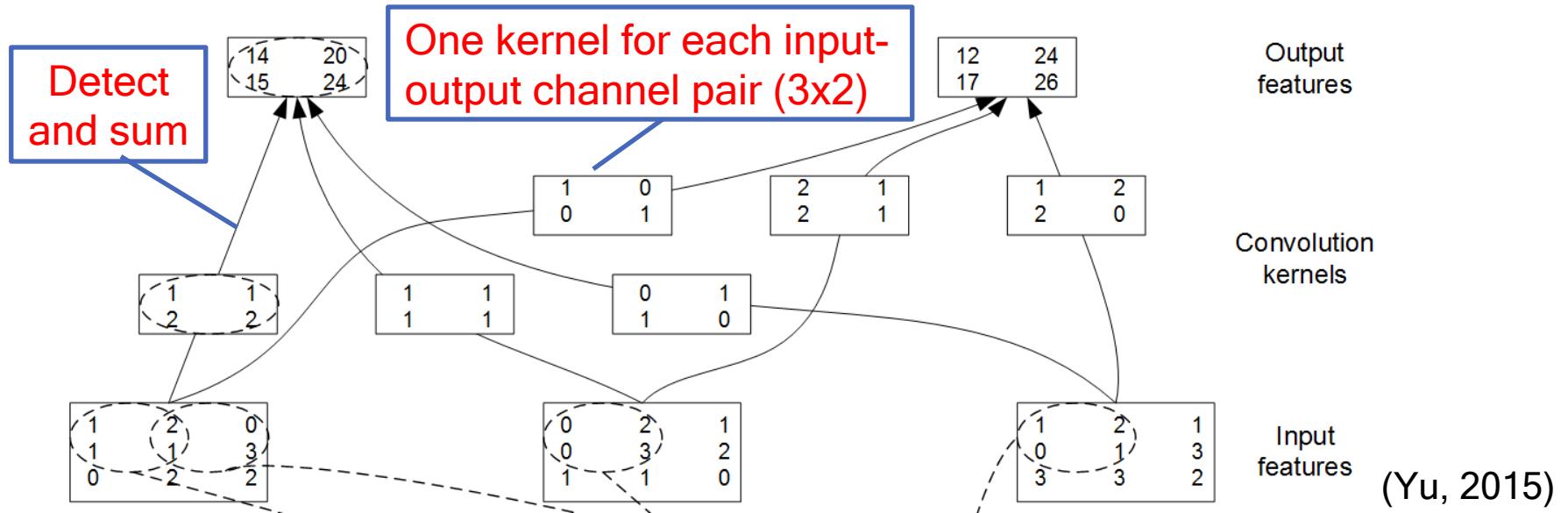
Figure 2. Residual learning: a building block

- Solution: learn a “skip connection” basically adding a passthrough branch that learns an identity mapping
  - Might not look like much, but this paper played a crucial role in giving us truly deep modern neural networks
  - 99% of modern architectures have a skip connection.



# CNNs: why they're good?

- Model translational invariance globally in the network (when pooling layers are used) and equivariance (locally in the convolutional layers)
  - Shared local filters (weights) tiled across image to detect the same pattern at different locations
  - Sub-sampling through pooling (max, average, or other) to reduce variability



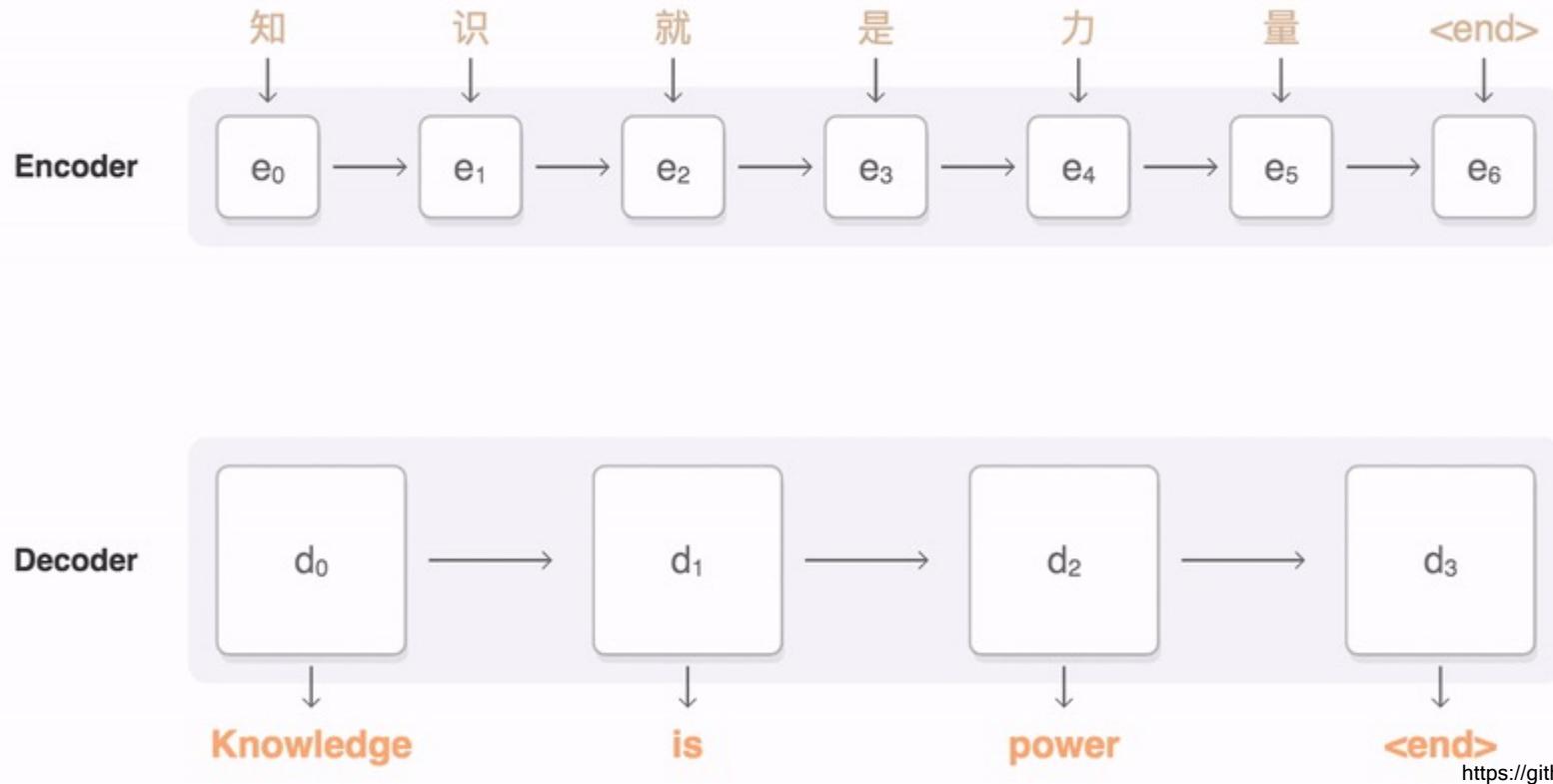
# Limitations of CNNs

- CNNs cannot take advantage of dependencies and correlations between samples (and labels) in a sequence
- No inherent concept of “memory”
- Recurrent neural networks (RNNs) are designed for this
- Many tasks are inherently sequential
  - Speech, video, natural language, DNA

(Yu, 2015)

# RECURRENT NEURAL NETWORKS

# Sequence modelling



In sequence modelling we have a structured sequence of data  $X = [x_0, x_1, x_2, \dots, x_n]$  which we wish to extract information from using a model  $f$ .

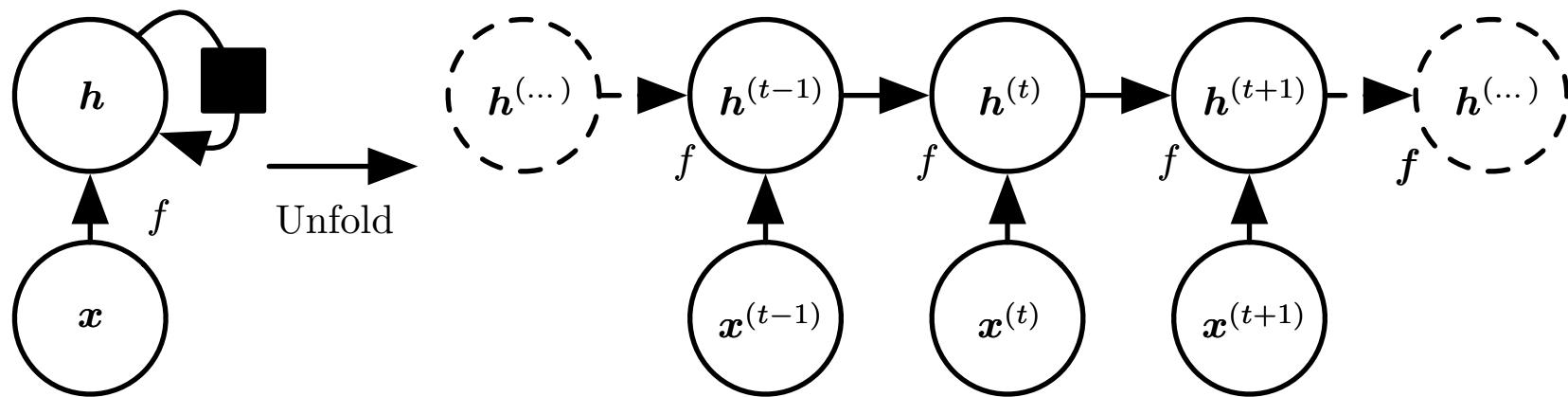
The task is to find a function  $f(X)$  which can extract information  $Y = [y_0, y_1, y_2, \dots, y_k]$  *from the input sequence X*.

# Recurrent Neural Network

- In sequential problems the memory is important in decision making.
- If the connectivity has directed cycles, the network can do much more than just computing a fixed sequence of non-linear transforms:
  - It can remember things for a long time.  
The network has internal state. It can decide to ignore the input for a while if it wants to.
  - It can model sequential data in a natural way.  
No need to use delay taps to spatialize time.

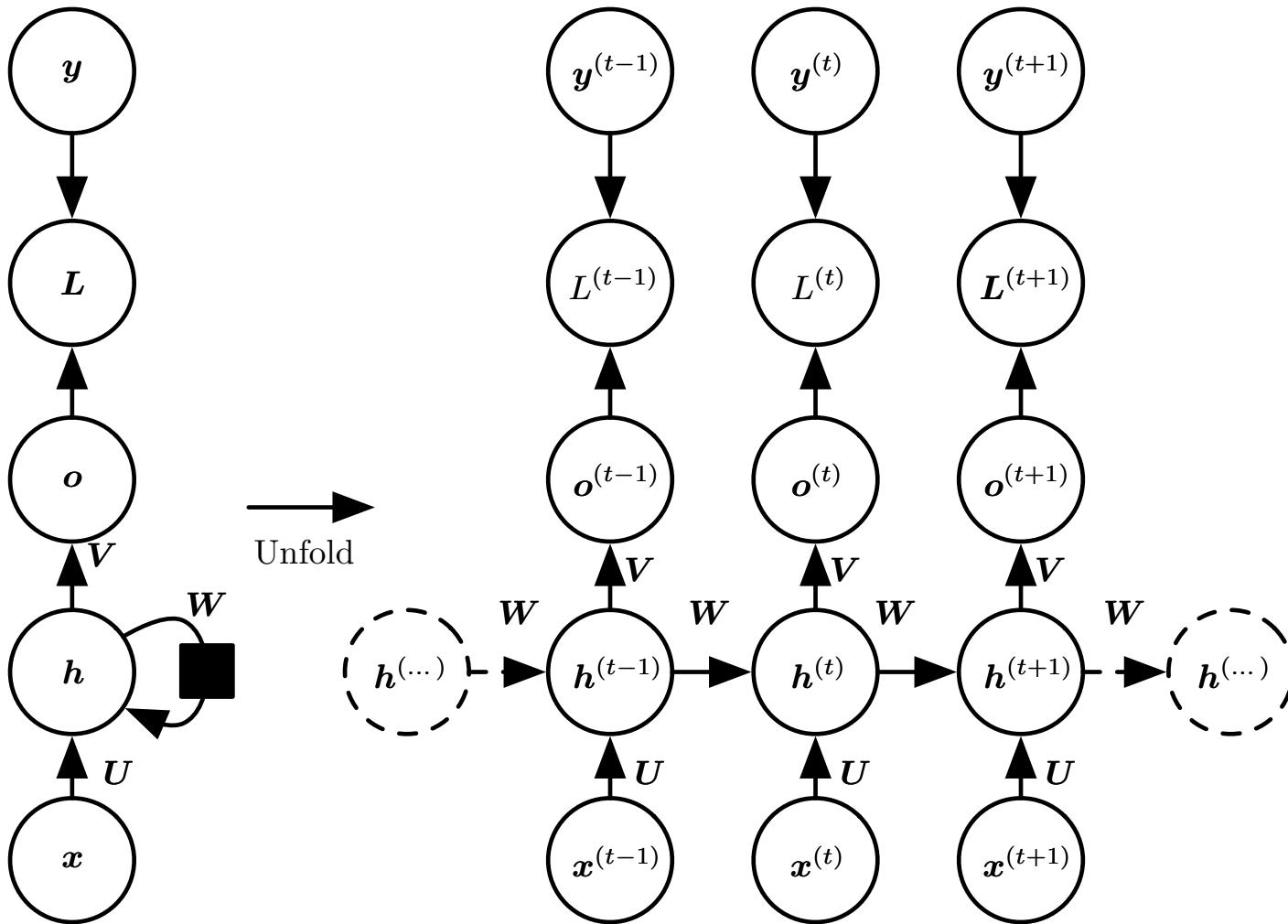
(Yu, 2015)

# Unfolding Computation Graphs



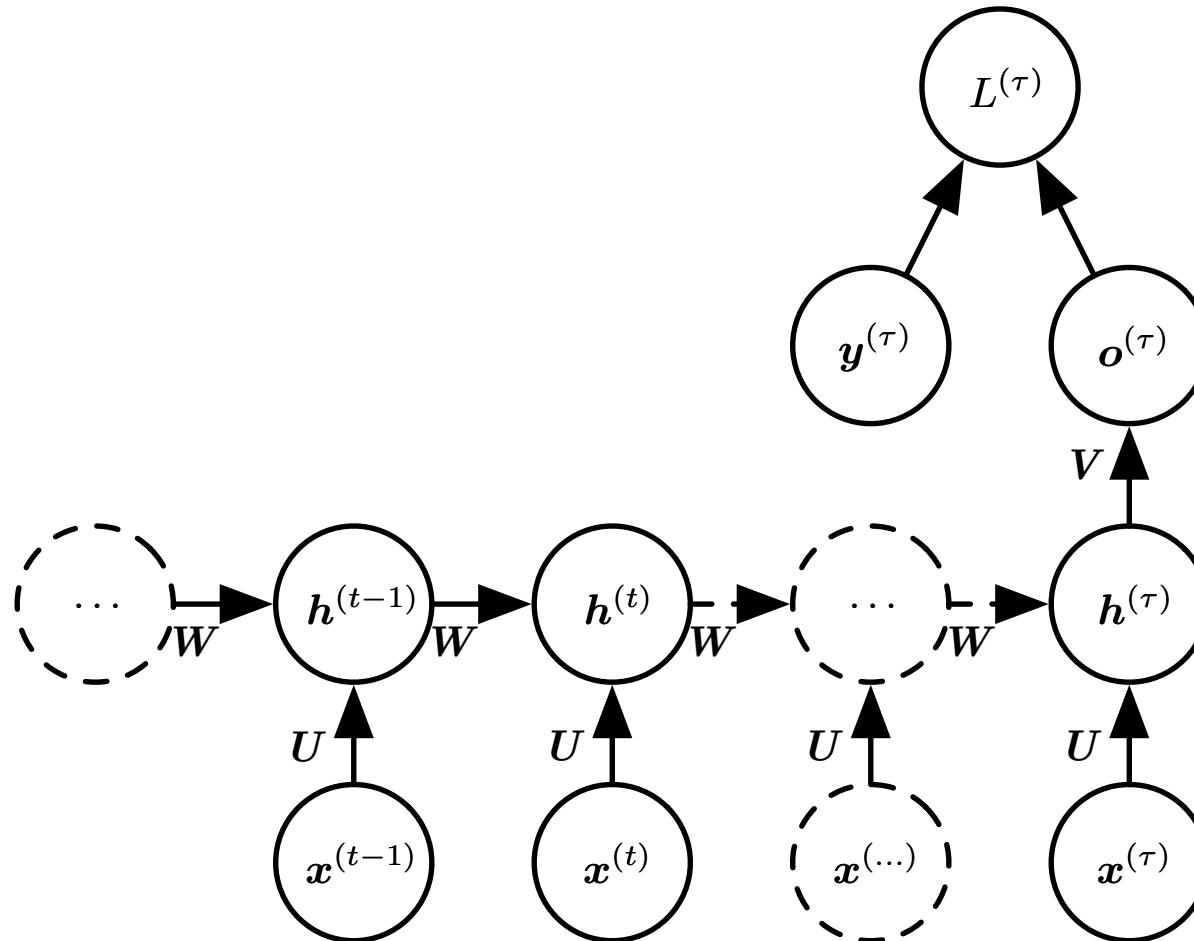
(Goodfellow, 2016)

# Recurrent Hidden Units



(Goodfellow, 2016)

# Sequence Input, Single Output



(Goodfellow, 2016)

# Model Learning

- Possible targets in RNN
  - Desired final activities of all the units
  - Desired activities of all units for the last few steps
  - Desired activity of a subset of the units.  
The other units are “hidden”
- Backpropagation through time
  - forward pass: builds up a stack of the activities of all the units at each time step.
  - backward pass: peels activities off the stack to compute the error derivatives at each time step.
  - Summation pass: add together the derivatives at all the different times for each weight.

(Yu, 2015)

# Limitations of Simple RNNs

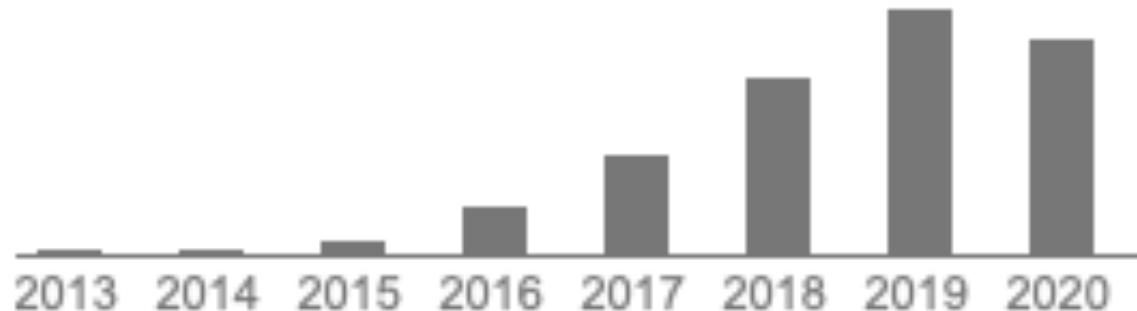
- Simple RNNs are difficult to train due to diminishing and explosion of gradients over time
  - Can be partially alleviated with gradient thresholding
- Simple RNNs have difficulty modeling long-range dependencies
  - The effect of information from past samples decreases exponentially
- Is it possible to solve the gradient diminishing problem so that we can model long-range dependencies
- Yes, with carefully designed recurrent structures such as long short-term memory (LSTM) RNNs.

(Yu, 2015)

# Long Short-Term Memory RNNs

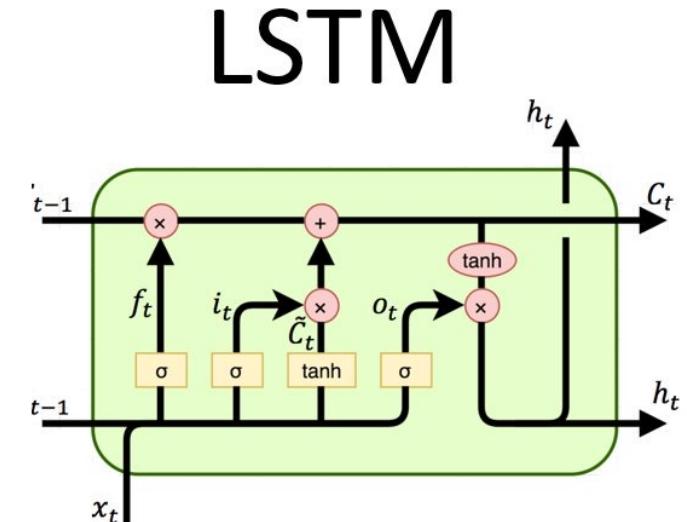
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

Cited by 38972



# Long Short-Term Memory RNNs

- An extension of RNN that addresses vanishing gradient problem



$$\text{Input gate } \mathbf{i}_t = \sigma \left( \mathbf{W}^{(xi)} \mathbf{x}_t + \mathbf{W}^{(hi)} \mathbf{h}_{t-1} + \mathbf{W}^{(ci)} \mathbf{c}_{t-1} + \mathbf{b}^{(i)} \right)$$

$$\text{forget gate } \mathbf{f}_t = \sigma \left( \mathbf{W}^{(xf)} \mathbf{x}_t + \mathbf{W}^{(hf)} \mathbf{h}_{t-1} + \mathbf{W}^{(cf)} \mathbf{c}_{t-1} + \mathbf{b}^{(f)} \right)$$

$$\text{Cell state } \mathbf{c}_t = \mathbf{f}_t \bullet \mathbf{c}_{t-1} + \mathbf{i}_t \bullet \tanh \left( \mathbf{W}^{(xc)} \mathbf{x}_t + \mathbf{W}^{(hc)} \mathbf{h}_{t-1} + \mathbf{b}^{(c)} \right)$$

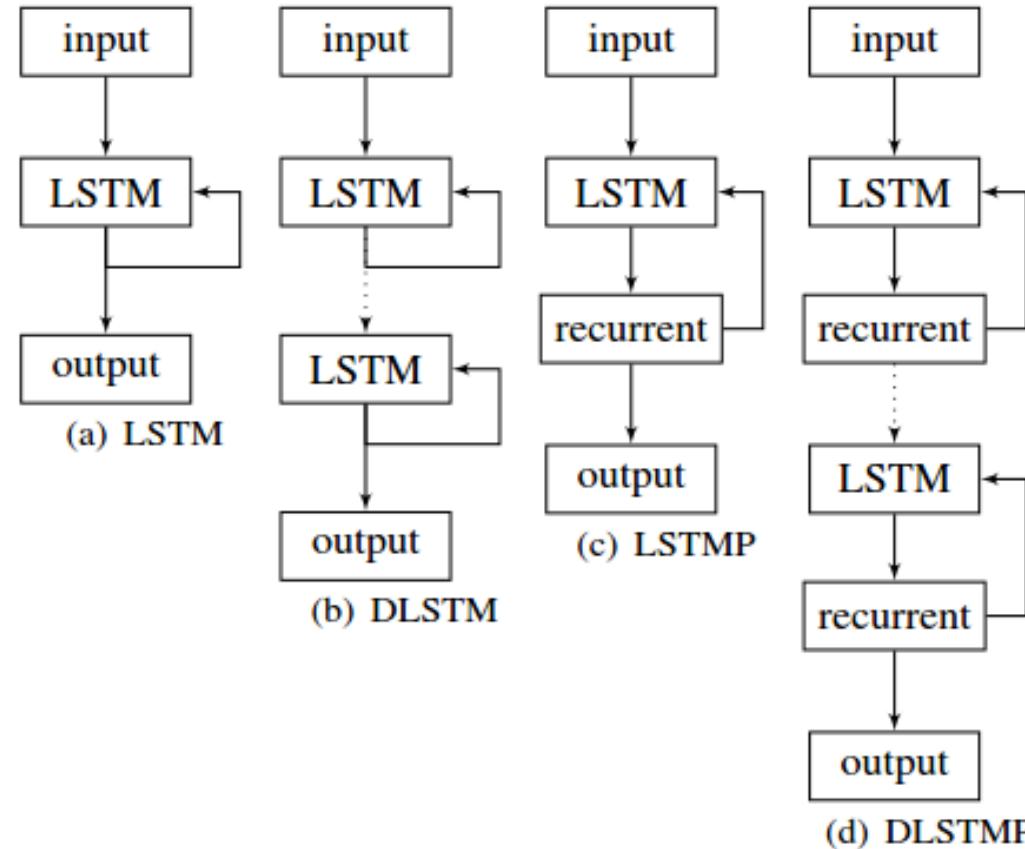
$$\text{Output gate } \mathbf{o}_t = \sigma \left( \mathbf{W}^{(xo)} \mathbf{x}_t + \mathbf{W}^{(ho)} \mathbf{h}_{t-1} + \mathbf{W}^{(co)} \mathbf{c}_t + \mathbf{b}^{(o)} \right)$$

$$\text{output } \mathbf{h}_t = \mathbf{o}_t \bullet \tanh(\mathbf{c}_t),$$

(Graves, 2013)

# Deep LSTMs

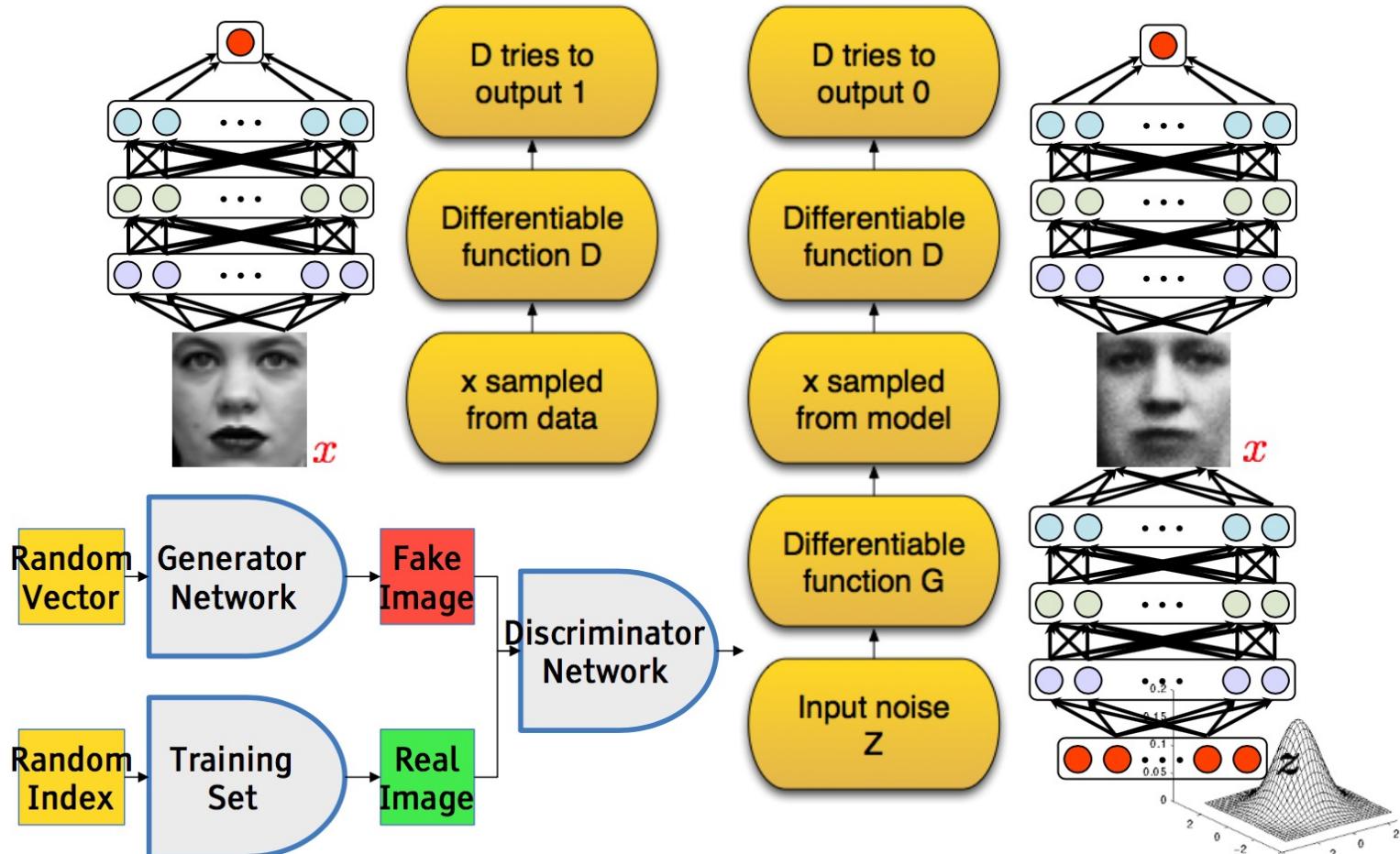
- Stack LSTM layers



(Sak et al., 2014)

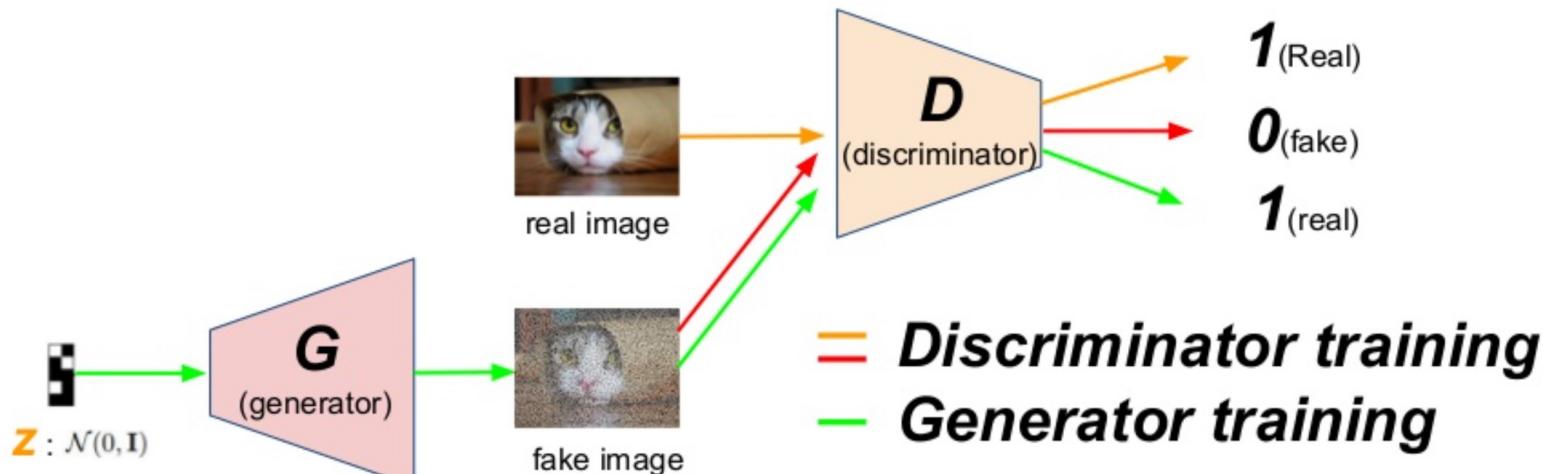
# ADVERSARIAL NEURAL NETWORKS

# Adversarial nets framework



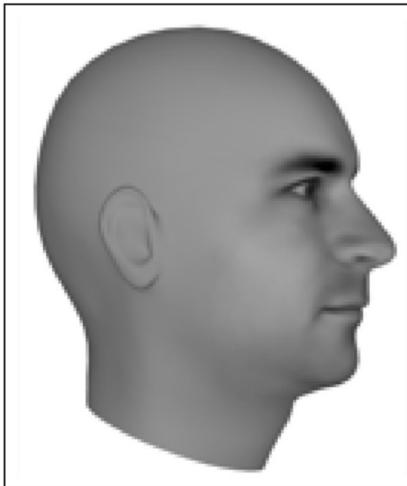
(Goodfellow, 2016)

# Alternate training

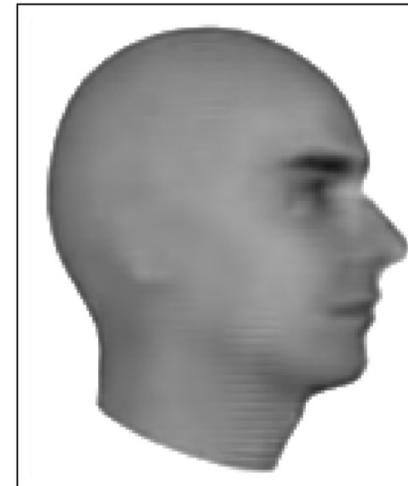


(KakaoBrain)

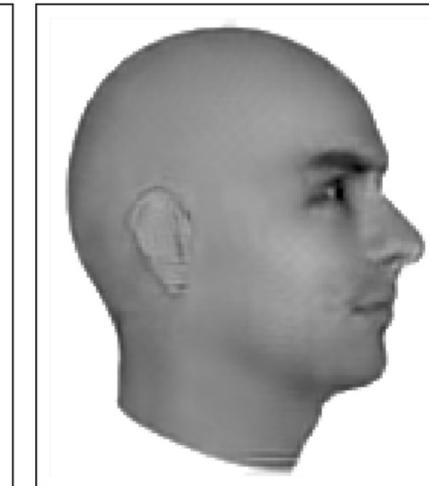
Ground Truth



MSE

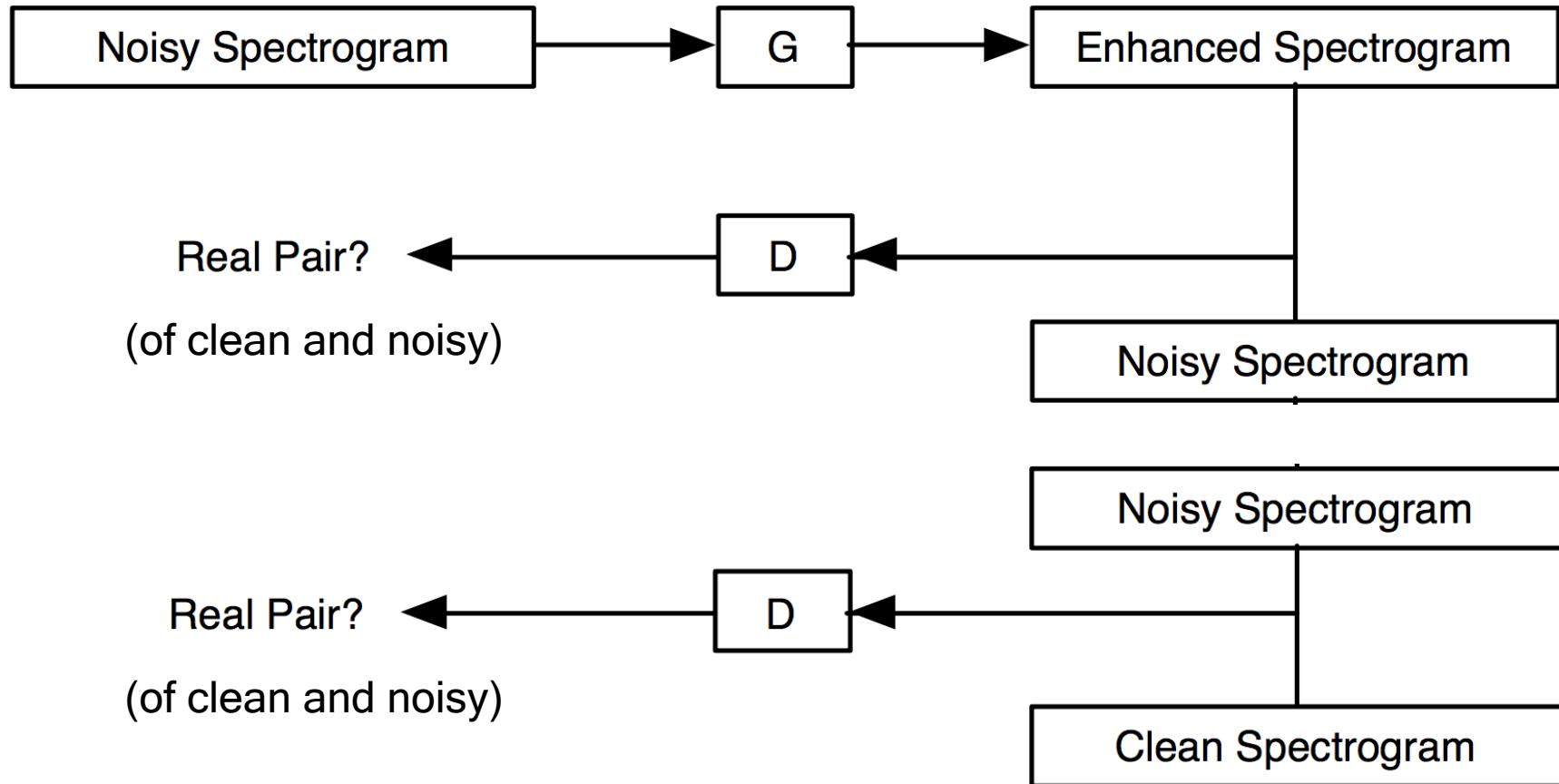


Adversarial



(Goodfellow, 2016)

# Conditional GAN for Speech Enhancement



(Michelsanti and Tan, 2017)

# Generative adversarial net (GAN) for image generation



1024 × 1024 images generated using the CELEBA-HQ dataset.



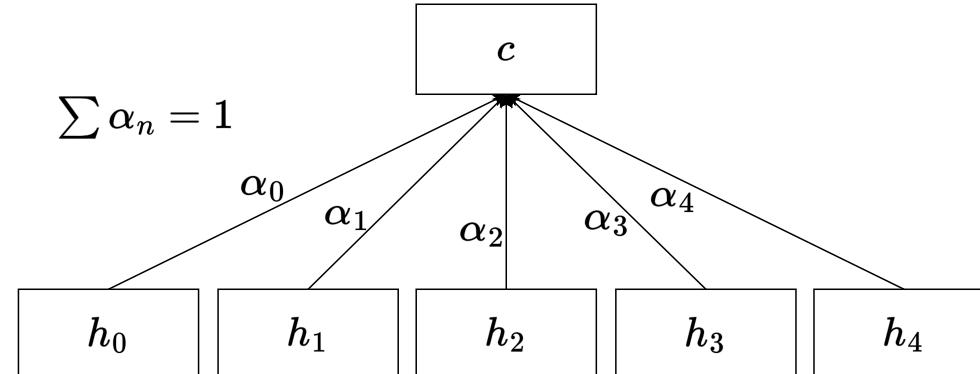
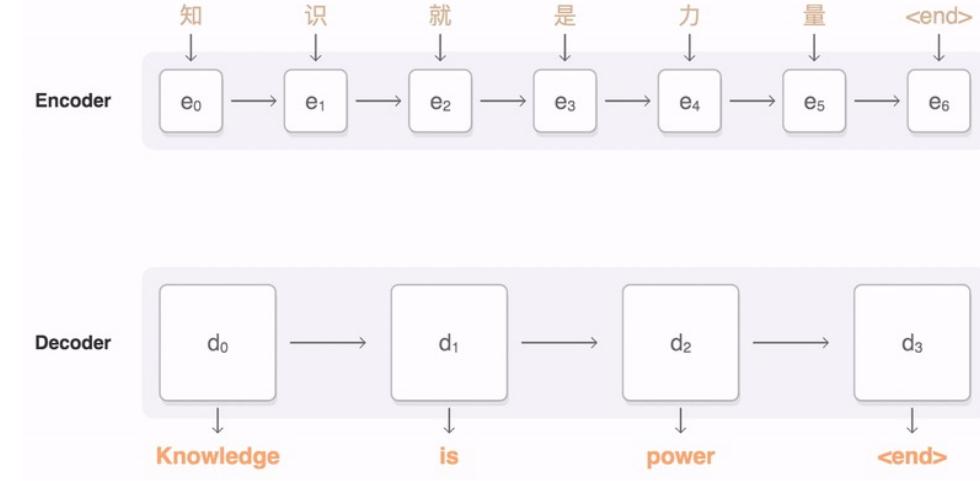
The nearest neighbors found from the training data ( $L_1$  distance in pixel space).

# TRANSFORMERS

# Attention mechanism

## Attention in simple terms

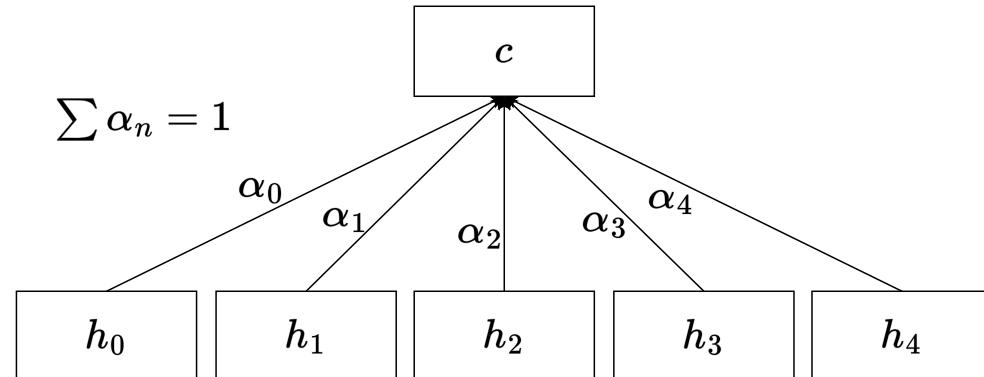
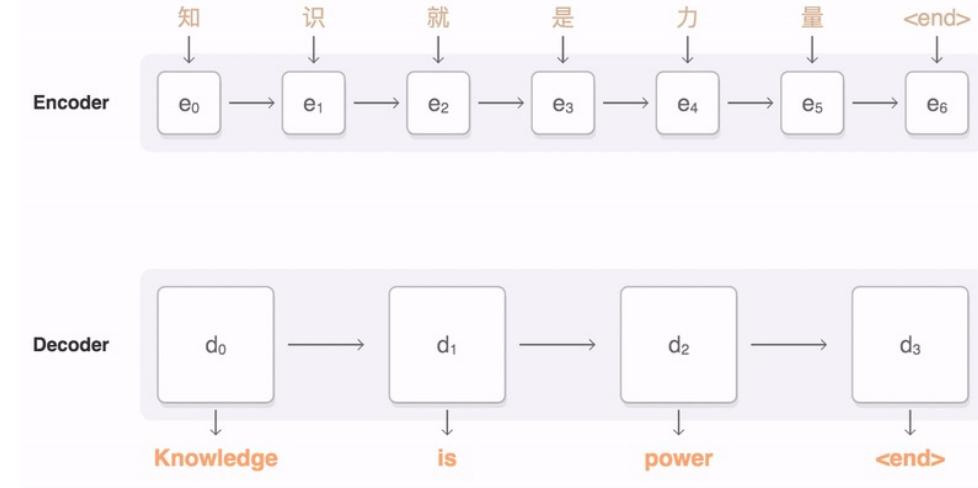
- Given a sequence, the relationship between some elements of the sequence can be more or less important
- Despite this, in RNNs, only the last state encodes all the information in a fixed size context vector.
- By adding an attention mechanism, information from each individual hidden state can be summarized into a context vector.
- While attention mechanisms in RNNs solves some of the long-term dependency issues, they are computationally inefficient as each hidden state relies on the previous hidden state.



# Attention mechanism

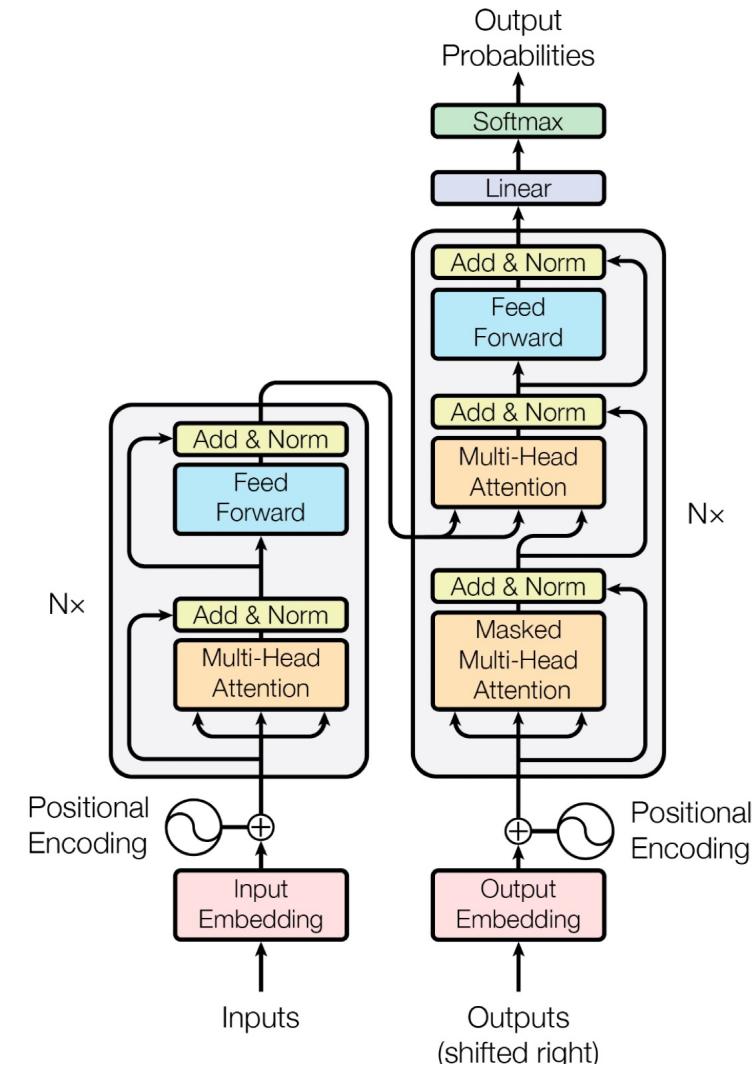
BUT WHAT IF

We build a model based only on Attention



# Transformer: Attention is all you need

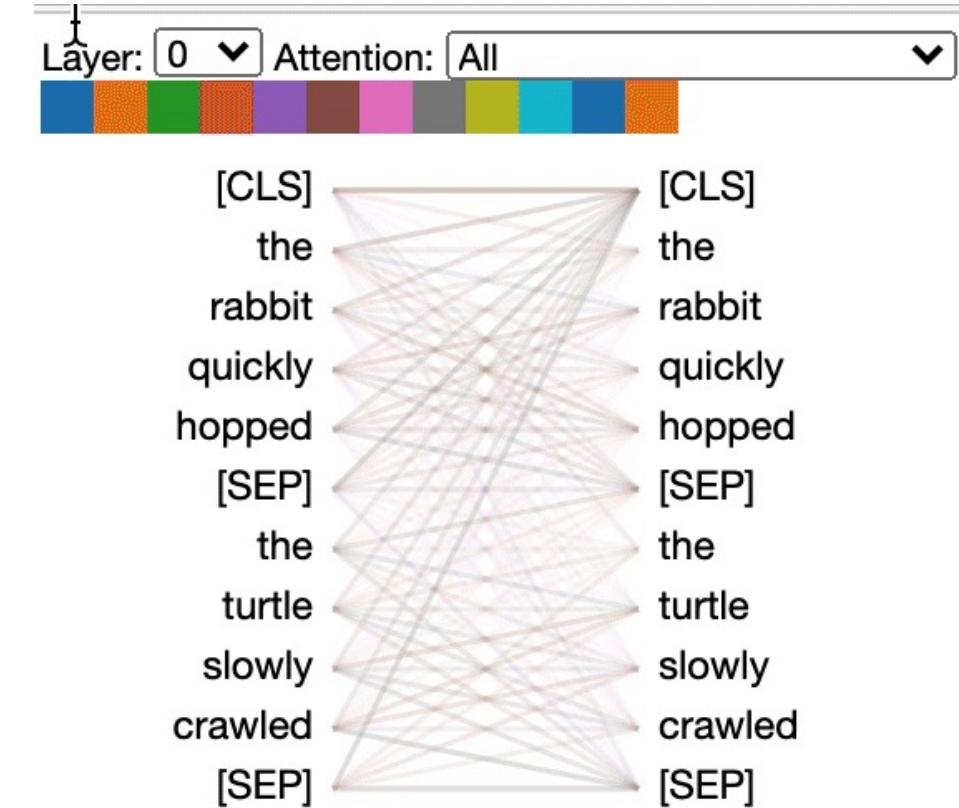
- Encoder maps input sequence  $(x_1, x_2, \dots, x_n)$  to representation  $z = (z_1, z_2, \dots, z_n)$
- Decoder predicts output probabilities of the next sequence in an “auto-regressive manner” based on encoder representations and shifted output sequence.



# Building blocks: Scaled Dot-product attention

## In simple words

- Compares each element of the input sequence with all other elements to determine their relative importances
- Focuses on contextually relevant information
- Easily vectorized, so highly parallelizable



# Building blocks: Scaled Dot-product attention

## Scaled Dot-Product Attention

### Mathematically

Inputs: queries  $q \in \mathbb{R}^{d_k}$ , keys  $k \in \mathbb{R}^{d_k}$  and values  $v \in \mathbb{R}^{d_v}$  are vectors of dimensions  $k, k$  and  $v$  respectively.

$$\text{Attn}: (\mathbb{R}^{d_k}, \mathbb{R}^{d_k}, \mathbb{R}^{d_v}) \rightarrow \mathbb{R}^{d_v}$$

1. Dot product:  $q \cdot k$
2. Scale the dot product:  $w_v = \frac{q \cdot k}{\sqrt{d_k}}$
3. Apply softmax:  $\text{softmax}\left(\frac{q \cdot k}{\sqrt{d_k}}\right)$
4. Get weighted values:  $\text{softmax}\left(\frac{q \cdot k}{\sqrt{d_k}}\right) \cdot v$

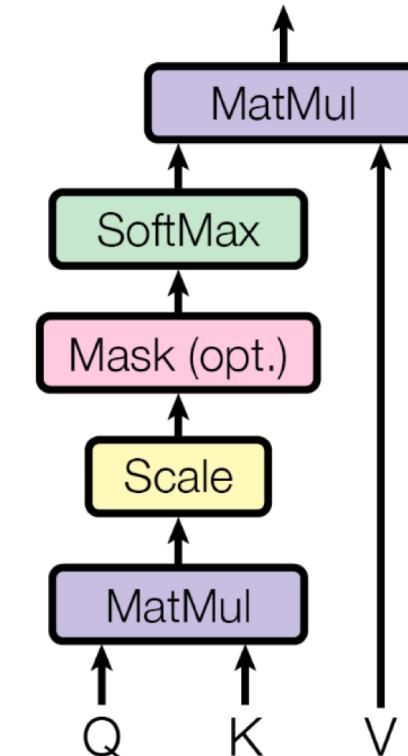


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS 2017

# Building blocks: Scaled Dot-product attention

## Scaled Dot-Product Attention

Finally, matrix notation in all it's glory

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{Q \cdot K^T}{\sqrt{d_k}} \right) V$$

Where,

$Q \in \mathbb{R}^{n \times d_k}$  is the query matrix,

$K \in \mathbb{R}^{n \times d_k}$  is the query matrix,

$V \in \mathbb{R}^{n \times d_v}$  is the query matrix,

And  $n$  is the sequence length/number of tokens

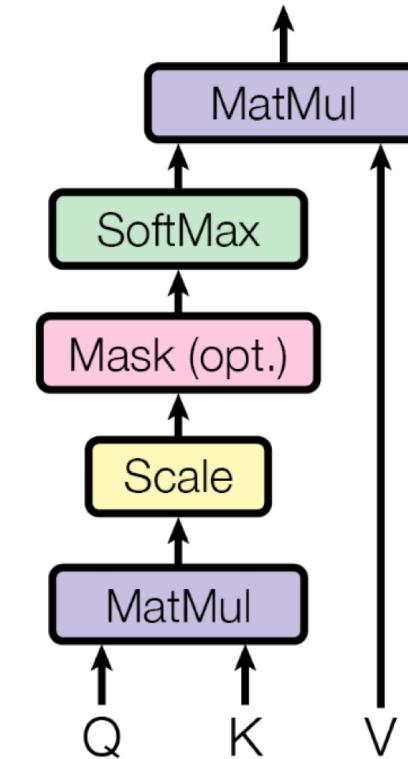


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS 2017

# Building blocks: Scaled Dot-product attention

## Scaled Dot-Product Attention

- Why (and where) is it called self-attention?

*When queries, keys and values are linear projections of the same input (as we'll see soon)*

- Wait, where's the learning?

*Yep, there's no learning here. We just described the Scaled dot-product attention operation. To make it dynamic, we simply learn weights for queries, keys and values and compute attention on their linear projections.*

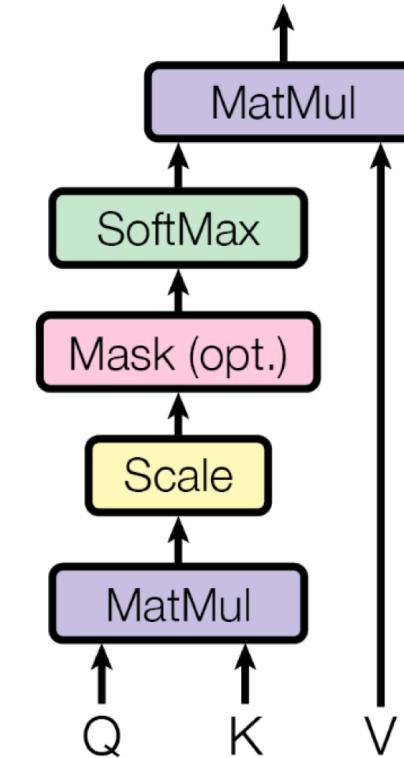


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS 2017

# Building blocks: Scaled Dot-product attention

## Scaled Dot-Product Attention

➤ Why divide by  $\sqrt{d_k}$ ?

*Hint: has some to do with backpropagation*

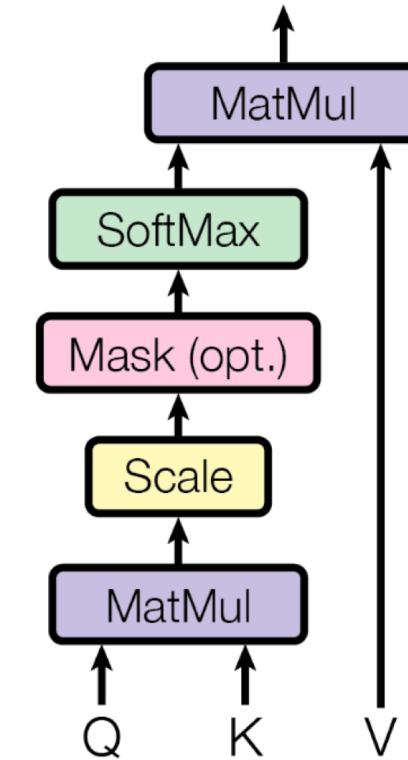


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS 2017

# Building blocks: Scaled Dot-product attention

## Scaled Dot-Product Attention

- Why divide by  $\sqrt{d_k}$ ?
- To stop the magnitude of the dot product from growing too large of course. But why?
- Softmax!! Softmax of large values  $\rightarrow$  pushes gradients to be very small.
- Small but key difference v/s previous additive and dot product attention

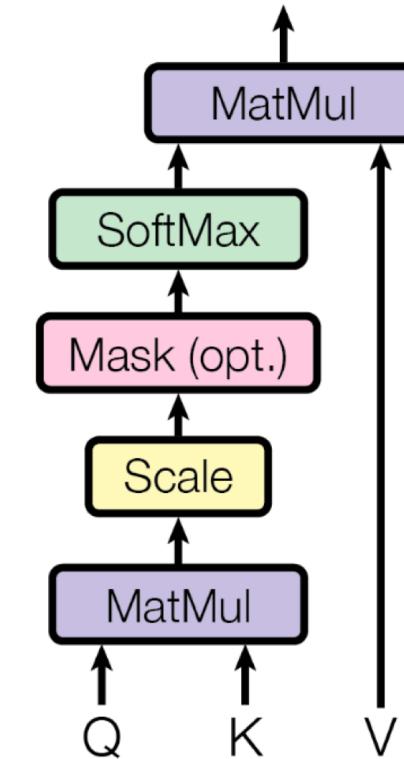


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS 2017

# Building blocks: Multi-Head Attention

## In simple words

Instead of computing attention once, how about we divide the workload across multiple parallel attention computations

## In other words

- Let  $d_m$  be the desired dimensionality of the feature space in which we want to compute attention.
- Instead of a single attention computation, how about we compute attention independently across  $h$  heads of dimensionality  $\frac{d_m}{h}$

Multi-Head Attention

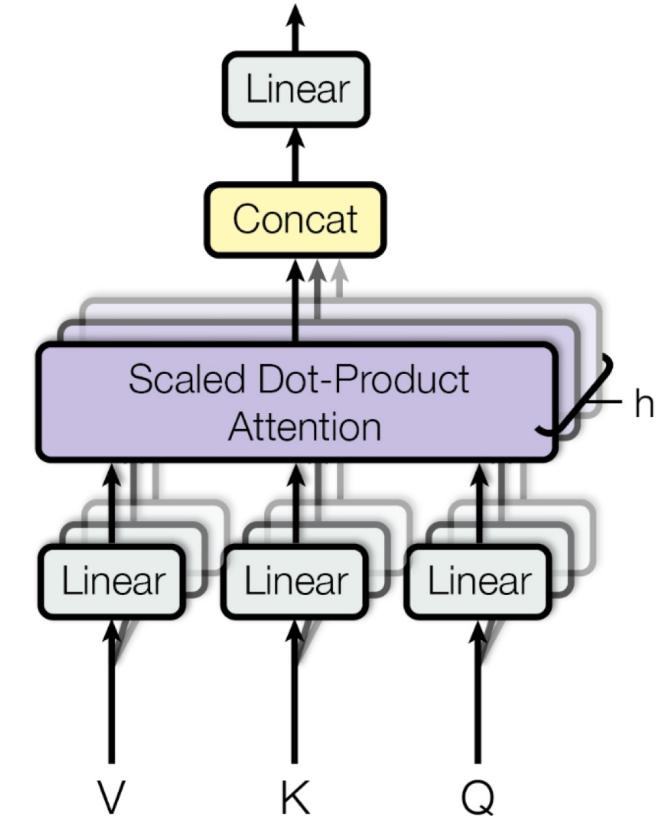


Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS2017

# Building blocks: Multi-Head Attention

## More specifically

- So, get  $h, \frac{d_m}{h}$  dimensional linear projections of  $k, q$  and  $v$
- Compute self-attentions of these projections, in parallel
- Concatenate them to get a  $d_m$  dimensional aggregated output.
- Apply a final linear projection!

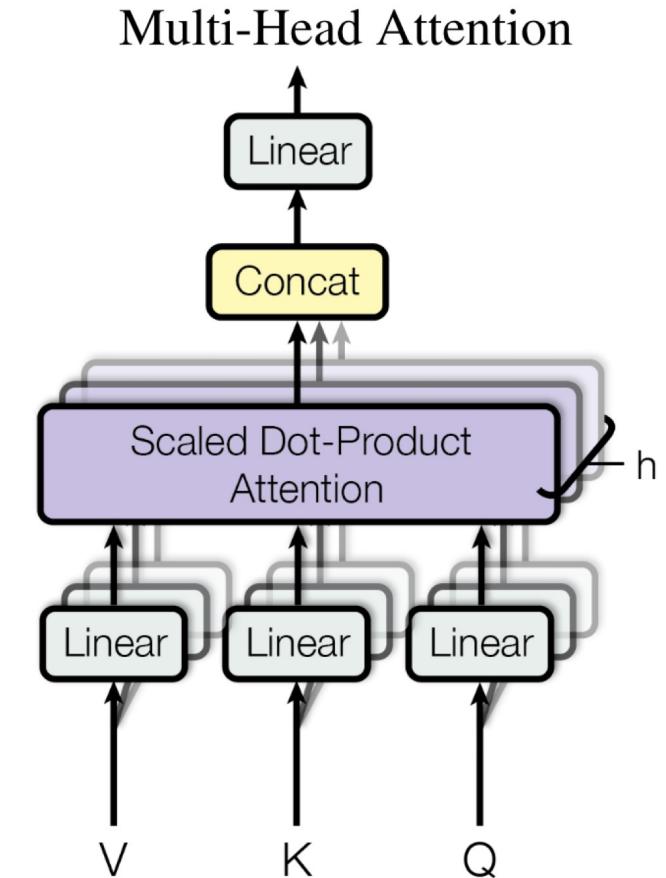
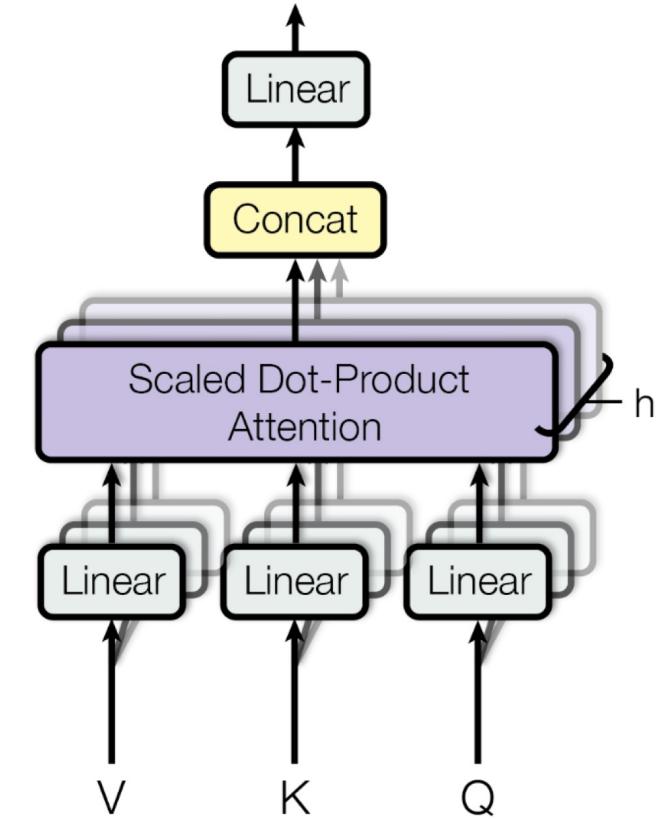


Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Building blocks: Multi-Head Attention

Multi-Head Attention



## Why though?

- Empirically worked better
- Total computation is the same, so why not
- The different attention heads capture different information. (We'll touch on this later)

Figure adapted from "Attention is all you Need", Vaswani et al, NeurIPS2017

# Building blocks: Multi-Head Attention

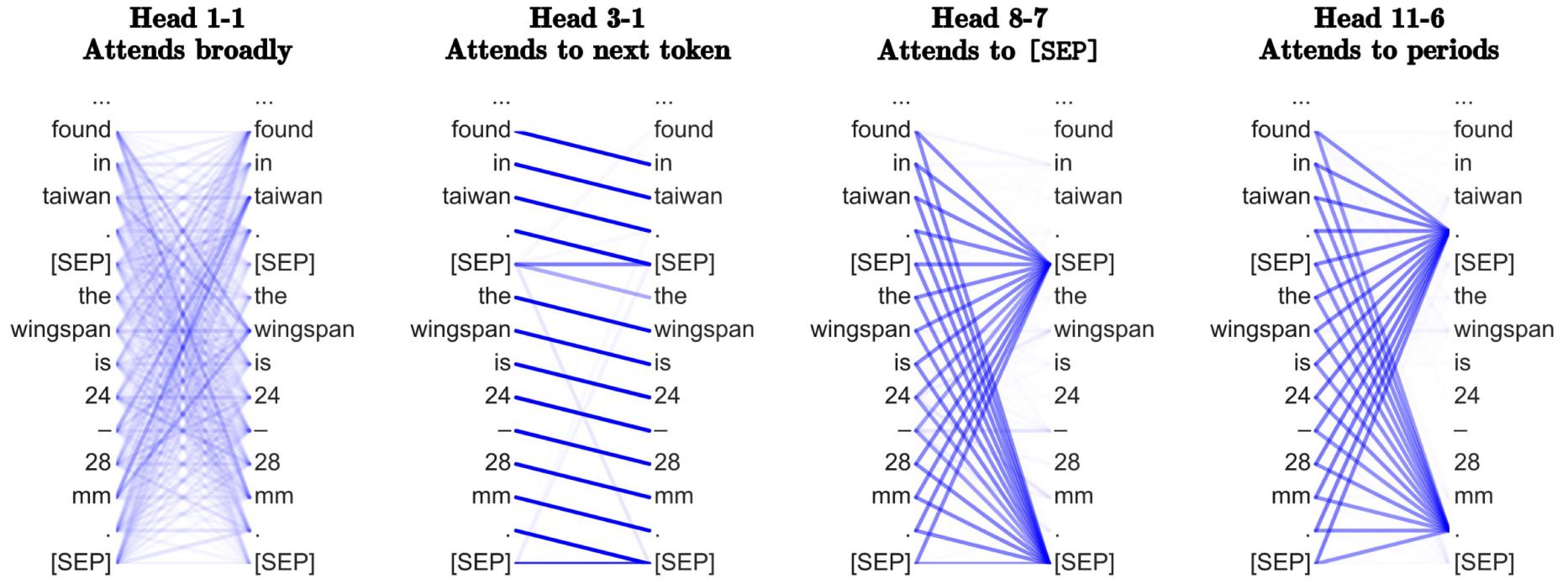


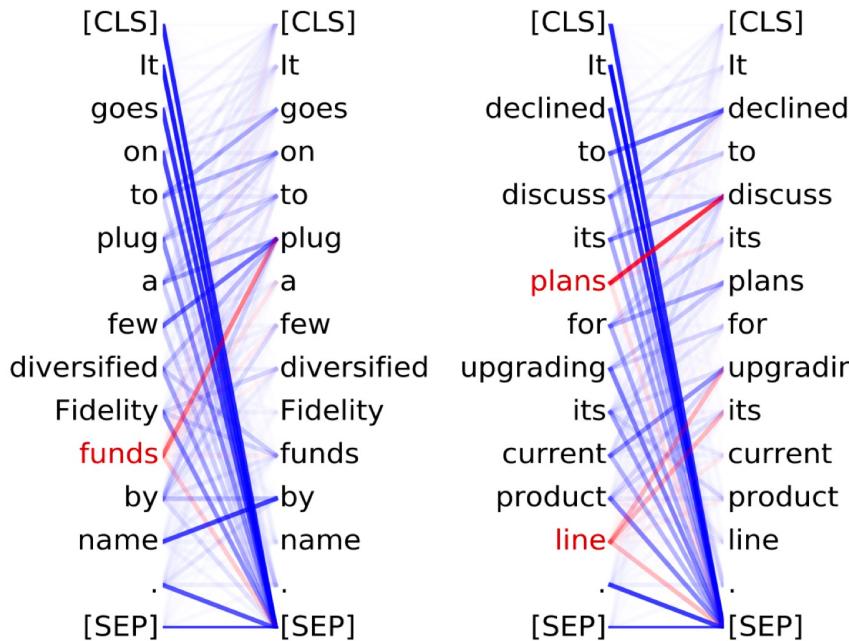
Figure 1: Examples of heads exhibiting the patterns discussed in Section 3. The darkness of a line indicates the strength of the attention weight (some attention weights are so low they are invisible).

Figure adapted from "What does BERT look at? An analysis of BERT's Attention" by Clark et al., BlackBoxNLP workshop, ACL 2019

# Building blocks: Multi-Head Attention

## Head 8-10

- Direct objects attend to their verbs
- 86.8% accuracy at the dobj relation



## Head 8-11

- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation

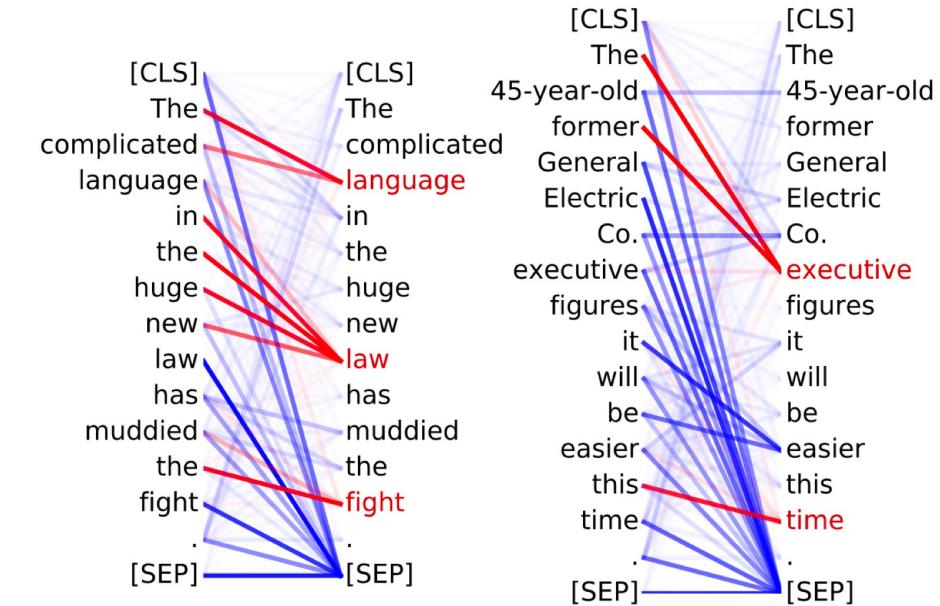
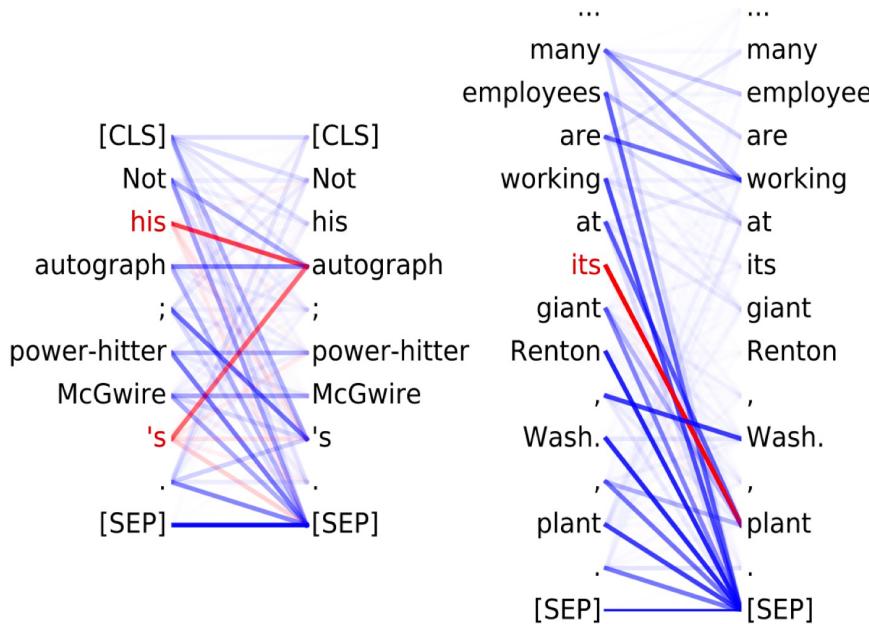


Figure adapted from "What does BERT look at? An analysis of BERT's Attention" by Clark et al., BlackBoxNLP workshop, ACL 2019

# Building blocks: Multi-Head Attention

## Head 7-6

- Possessive pronouns and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the poss relation



## Head 4-10

- Passive auxiliary verbs attend to the verb they modify
- 82.5% accuracy at the auxpass relation

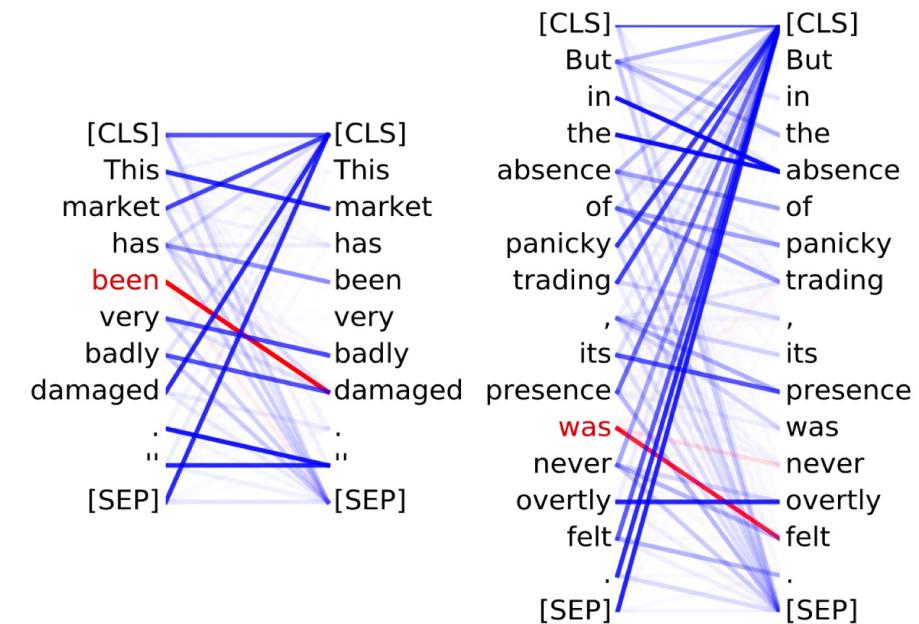


Figure adapted from "What does BERT look at? An analysis of BERT's Attention" by Clark et al., BlackBoxNLP workshop, ACL 2019

# Building blocks: Feed forward Network

- Each layer has a Feed-forward block after the Multi-Head attention block, which is quite simple:

$$FFN(x) = \max(0, xW_1 + b_1) W_2 + b_2$$

- i.e. a linear transformation  $\rightarrow$  ReLU activation  $\rightarrow$  linear transformation.

Here,

$$x \in \mathbb{R}^{n \times d_m}, W_1 \in \mathbb{R}^{d_m \times d_h}, W_2 \in \mathbb{R}^{d_h \times d_m}$$

$$d_m = 512, d_h = 2048$$

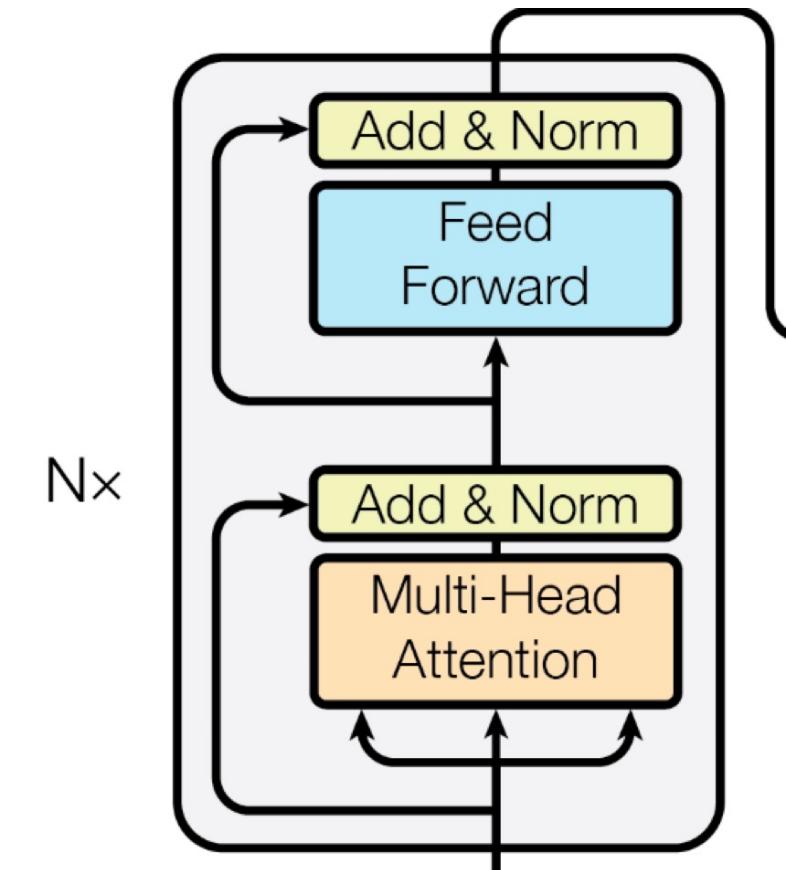


Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Building blocks

Question: What's this curve thingy going around across the different sub blocks?

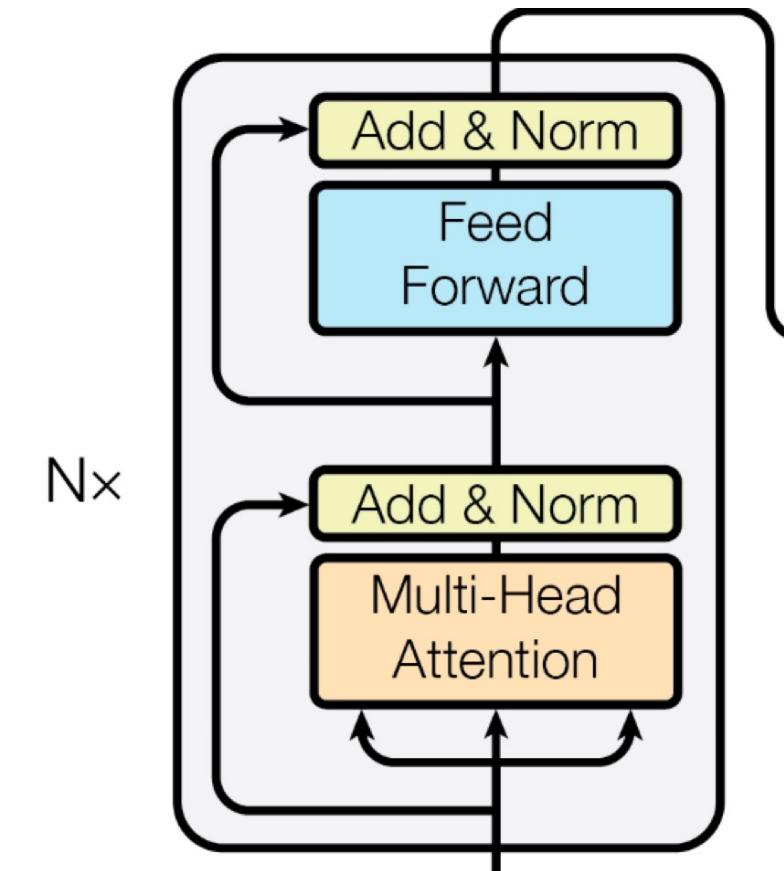


Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Building blocks: Residual Connections

Residual/skip connections!

(Told you. They're everywhere. Y'all didn't believe me)

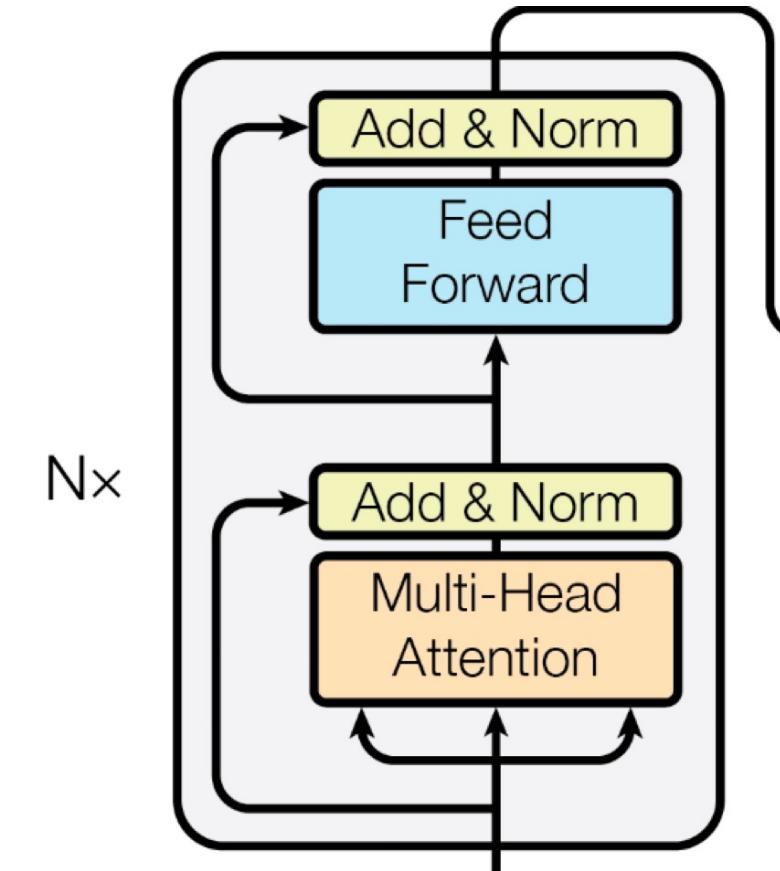


Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Building blocks: Positional Encoding

- Inputs to transformers are tokens
- No Convolution or RNN is done, and everything is computed in parallel
- Model has no positional information. We need to explicitly encode positional information into the input sequences.
- Positional Encoding to the rescue.

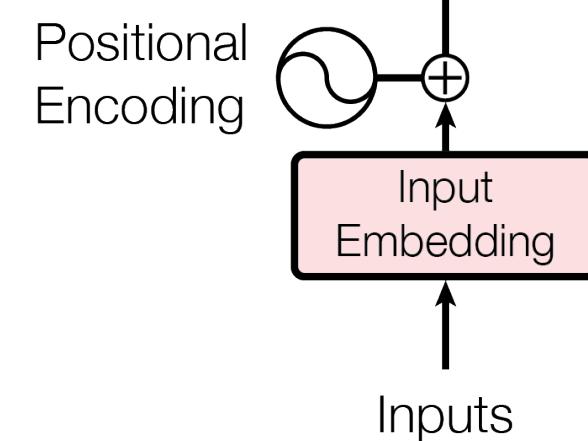


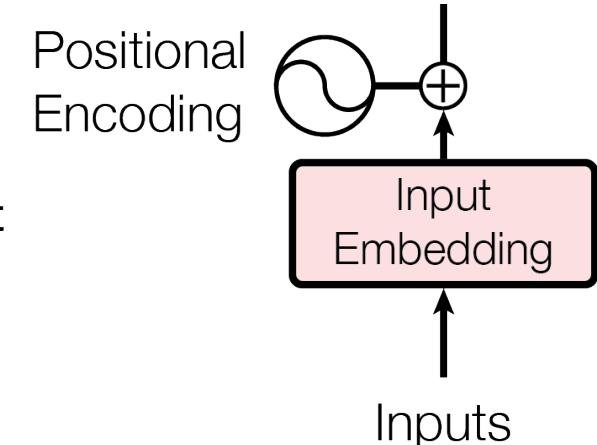
Figure adapted from “Attention is all you Need”, Vaswani et al,  
NeurIPS2017

# Building blocks: Positional Encoding

- Sinusoidal positional embedding: a vector  $p \in \mathbb{R}^{d_m}$  such that

$$p[i](t) = \begin{cases} \sin(\omega_k \cdot t) & i = 2k \\ \cos(\omega_k \cdot t) & i = 2k + 1 \end{cases}$$

Where  $\omega_k = \frac{1}{10000^{\frac{2k}{d_m}}}$



In other words, positional encoding is a vector with alternating sines and cosines

$$p_t = [\sin(\omega_1 t) \cos(\omega_1 t) \sin(\omega_2 t) \cos(\omega_2 t) \dots \sin(\omega_{d_m/2} t) \cos(\omega_{d_m/2} t)]^T$$

➤  $p_t$  is added to the input and output embeddings

Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Building blocks: Positional Encoding

**Why not use a learnable embedding instead?**

- Sure, though for most applications, they end up performing almost the same as fixed embeddings
- Sinusoidal embeddings allow extrapolation to longer sequence lengths than those encountered at training time.

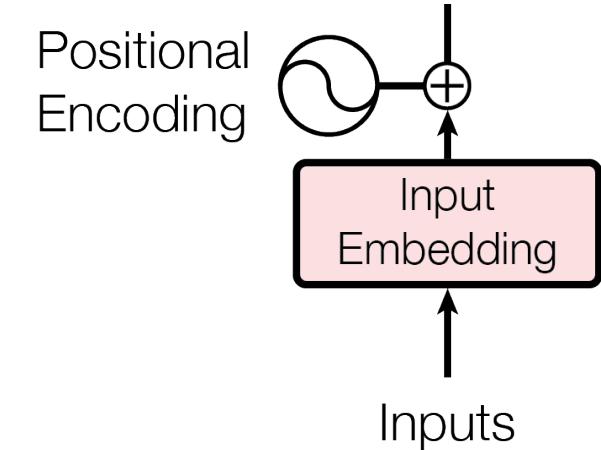


Figure adapted from "Attention is all you Need", Vaswani et al,  
NeurIPS2017

# Transformers: result and outlook

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [13]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		<b><math>3.3 \cdot 10^{18}</math></b>
Transformer (big)	<b>28.4</b>	<b>41.0</b>		$2.3 \cdot 10^{19}$

Table adopted from “Attention is all you Need”, Vaswani et al, NeurIPS 2017

At the time of release, single Transformer models outperformed/matched performance of ensemble of CNNs/RNNs at an order of magnitude lower training cost.

# Transformers: Why they got so popular

- Modality agnostic: unified architecture across types of input data
  - Images, text, videos, audio, signals
- Great performance
- Their treatment of the input as a set of tokens: a lot of flexibility
  - Led to resurgence of unsupervised learning

# An Image is Worth 16x16 words: Transformers for Image Recognition at Scale (ViTs)

## Intuition

- Transformers treat inputs as a set of tokens.
- Positional information has to be explicitly embedded.
- So how about we present images as a set of tokens, but a bit smartly?

## Enter Vision Transformers (ViTs)

# An Image is Worth 16x16 words: Transformers for Image Recognition at Scale (ViTs)

## The how

- Take non-overlapping patches of fixed size input images
- Project the patches to a  $d_m$  dimensional input space linearly
- Designate a special [CLS] token
- Feed the patches to the a (slightly modified) Transformer Encoder (no decoder here!)
- Add a MLP head on top to train a plain old classifier on the [CLS] token

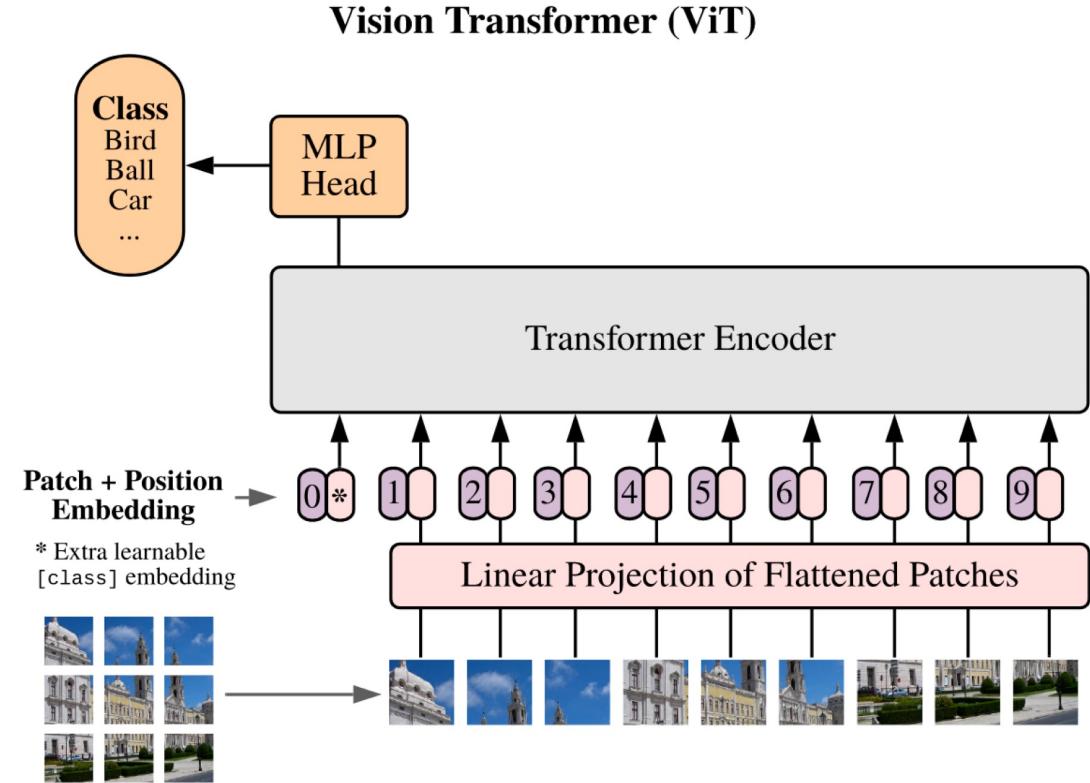


Figure adapted from "An Image is Worth 16x16 words: Transformers for Image Recognition at Scale", Dosovitskiy et al., ICLR 2021

# An Image is Worth 16x16 words: Transformers for Image Recognition at Scale (ViTs)

## The how

- Given input image  $x$ , make no-overlapping square patches of shape  $(P, P)$   
 $x \in \mathbb{R}^{H \times W \times C} \rightarrow x' \in \mathbb{R}^{N \times (P^2 \cdot C)}$
- Number of patches  $N = \frac{HW}{P^2}$

*Naive Question: A single NN layer to do patching + linear projection?*

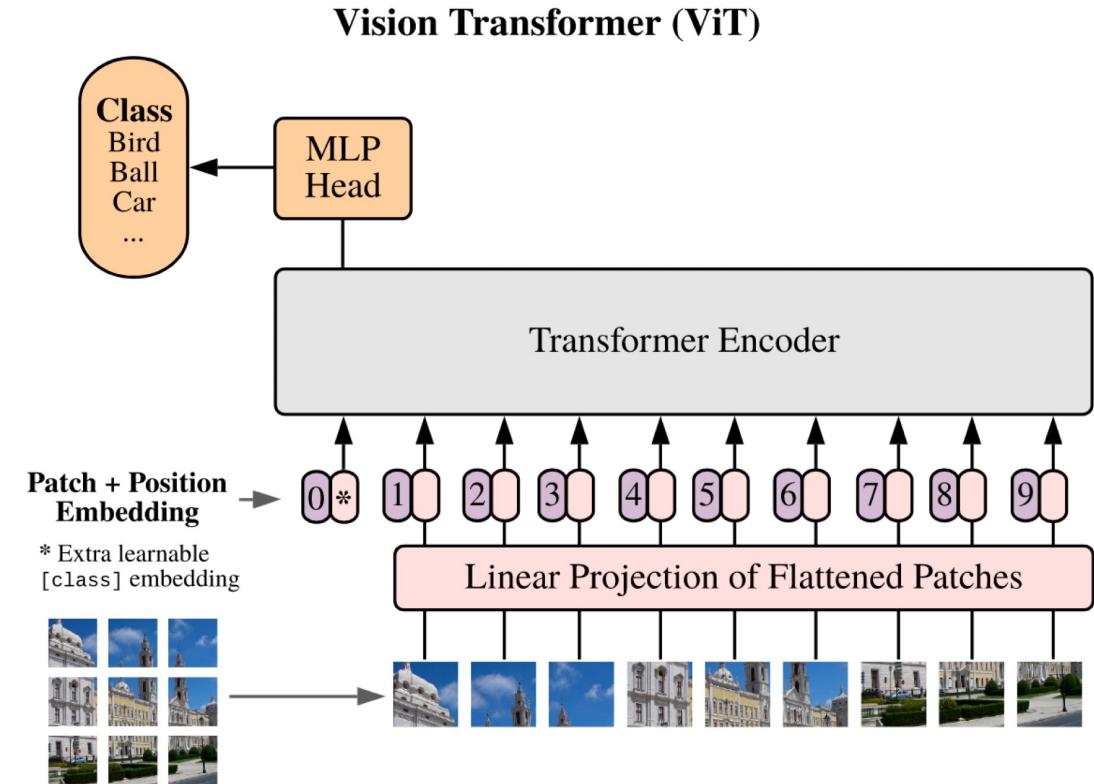


Figure adapted from "An Image is Worth 16x16 words: Transformers for Image Recognition at Scale", Dosovitskiy et al., ICLR 2021

# An Image is Worth 16x16 words: Transformers for Image Recognition at Scale (ViTs)

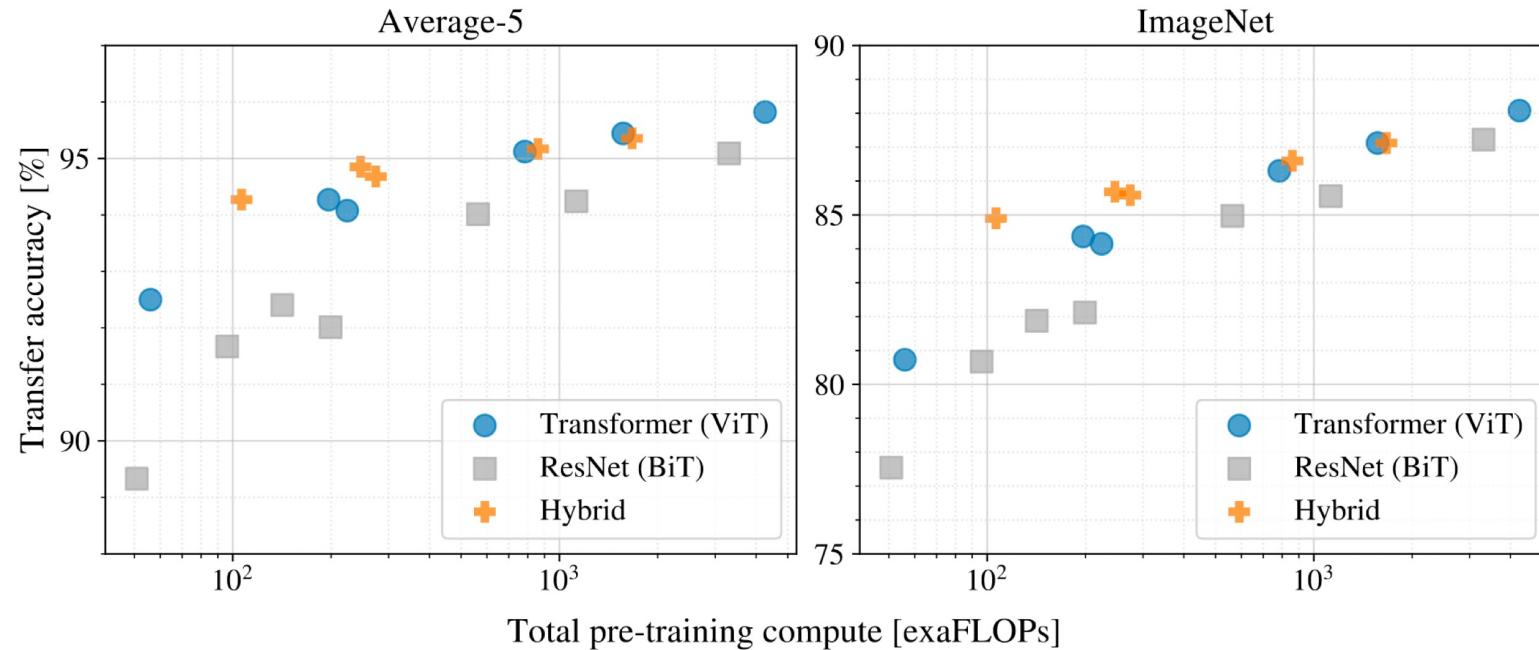
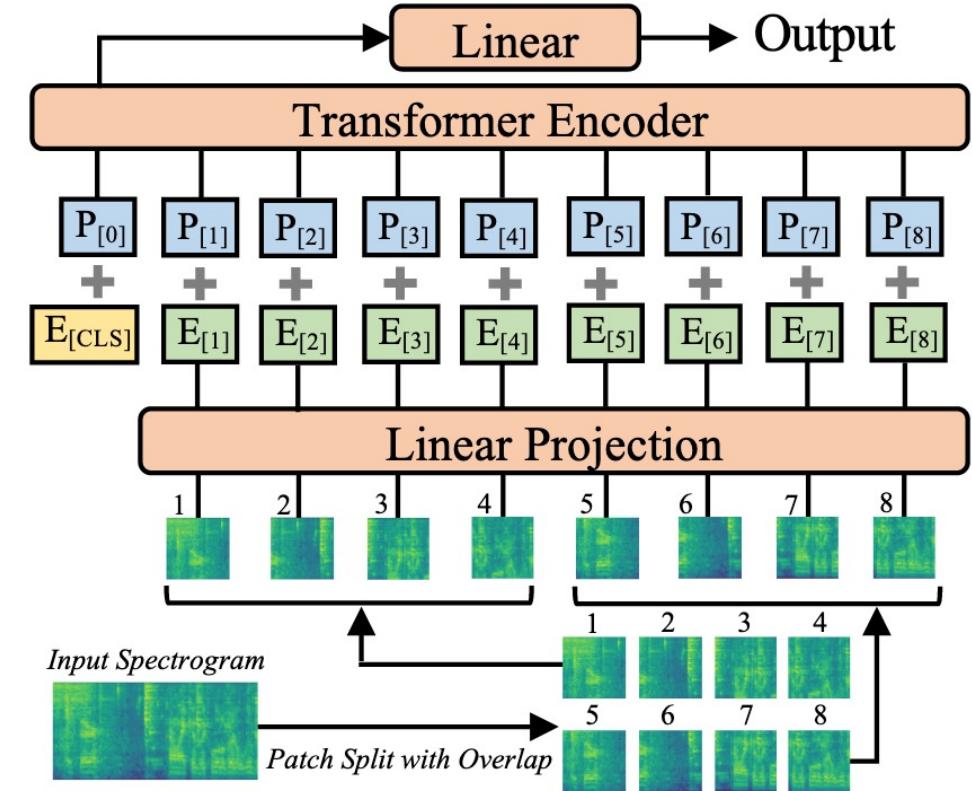


Figure adapted from "An Image is Worth 16x16 words: Transformers for Image Recognition at Scale", Dosovitskiy et al., ICLR 2021

- Better compute efficiency than similarly parameterised ResNet CNNs
- Go-to approach for comparing image and even audio classification these days

# Audio Spectrogram Transformer

- Designed to repurpose ViT models for audio
- Input features are time/frequency spectrograms which are split into patches
- Each patch is linearly projected to fit the input dimension of the transformer encoder



Source: <https://arxiv.org/pdf/2104.01778.pdf>

# Shortcomings of Transformers

Anecdotal evidence (twitter & r/machinelearning will concur)

Transformers can be a pain to train

- Overfit easily
- Much more sensitive to setting up a good optimization hyperparameters.
- Very sensitive to initial learning rate and learning rate schedule

# Shortcomings of Transformers

## Transformers are not like CNNs

- Self-attention captures **global information**. Convolutional Neural Networks are both **local** (in a layer) and **global** (in the full feature stack)
- Convolutional Neural Networks learn **hierarchical features**: simple edge-detector like features in the earlier layers and more abstract concepts in deeper layers. Standard Transformers do not\*

# Summary

## 1. Brief history and basic concepts

- Intro
- Brief History
- Notable Events
- Basic Concepts

## 2. Key models

- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Long Short-Term Memory (LSTM) RNNs
- Generative Adversarial Networks (GANs)
- Transformers

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. **Time series models**
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Quiz

- What are the key deep neural network models?
- Discuss the key components of them.

# Machine Learning

## Lecture 10: Time Series Models

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <http://kom.aau.dk/~zt>

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. **Time series models**
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Introduction

- Modeling dependencies in input; **no longer iid** (Independent and identically-distributed)
- Sequences:
  - Temporal
    - In speech: words in a sentence (syntax, semantics of the language), phonemes in a word (dictionary).
      - this is speech
      - th-ih-s-ih-z-s-p-ih-ch
    - In handwriting: pen movements
  - Spatial
    - In a DNA sequence: base pairs

# A three-state Markov model

- The weather on day  $t$  is characterised by a single one of the three states

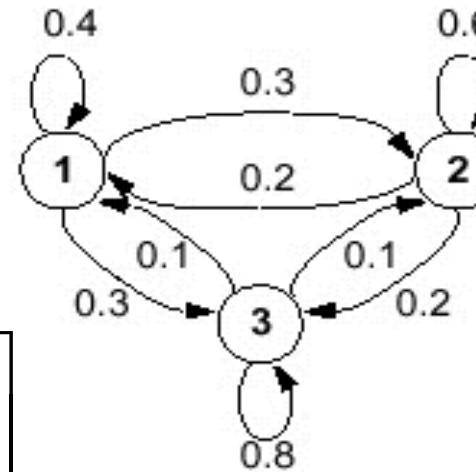
Example: A three-state model of the weather

State 1: precipitation (rain, snow, hail, etc.)

State 2: cloudy

State 3: sunny

$$A = \{a_{ij}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}$$



- The above stochastic process is considered an **observable Markov model** since the output process is a set of states at each instant of time, where each state corresponds to an observable event.

# Basic calculations

Example: What is the probability that the weather for eight consecutive days is "sun-sun-sun-rain-rain-sun-cloudy-sun"?

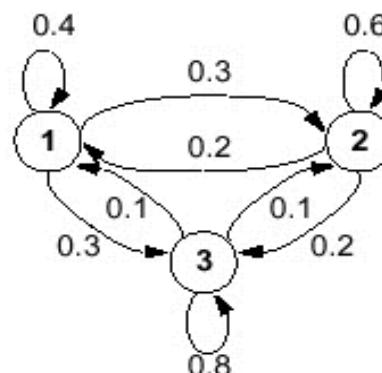
Solution:

$$\mathbf{O} = \begin{matrix} \text{sun} & \text{sun} & \text{sun} & \text{rain} & \text{rain} & \text{sun} & \text{cloudy} & \text{sun} \\ 3 & 3 & 3 & 1 & 1 & 3 & 2 & 3 \end{matrix}$$

$$\begin{aligned} P(\bar{O} | \text{Model}) &= P[3]P[3|3]P[3|3]P[1|3]P[1|1]P[3|1]P[2|3]P[3|2] \\ &= \pi_3 a_{33} a_{33} a_{31} a_{11} a_{13} a_{32} a_{23} \\ &= 1.536 \times 10^{-4} \end{aligned}$$

Example: A three-state model of the weather

- State 1: precipitation (rain, snow, hail, etc.)
- State 2: cloudy
- State 3: sunny



(After Joseph Picone)

# The Markov chain

- Consider a system described at any time  $t$  as being in one of a set of  $N$  states indexed by  $\{1, 2, \dots, N\}$

- Denote the actual state at time  $t$  as  $q_t$
- So, the first-order Markov chain is

$$P[q_t = j | q_{t-1} = i, q_{t-2} = k, \dots] = P[q_t = j | q_{t-1} = i]$$

- We consider those processes in which the right-side of the equation above is independent of time, leading to state-transition probabilities

$$a_{ij} = P[q_t = j | q_{t-1} = i], \quad 1 \leq i, j \leq N$$

with constraints:

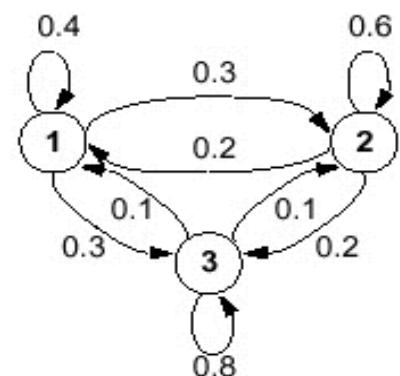
$$a_{ij} \geq 0 \quad \forall j, i$$

$$\sum_{j=1}^N a_{ij} = 1 \quad \forall i$$

Example: A three-state model of the weather

State 1: precipitation (rain, snow, hail, etc.)  
State 2: cloudy  
State 3: sunny

$N=3$

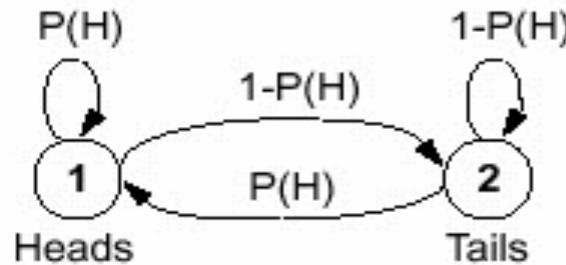


# “Hidden” Markov model

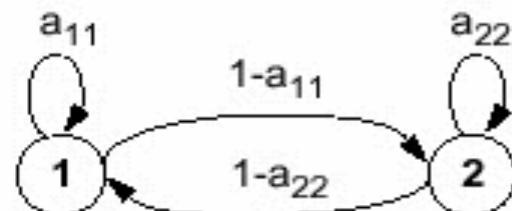
Consider the problem of predicting the outcome of a coin toss experiment.  
You observe the following sequence:

$$\bar{O} = (HHTTTHTTTH...H)$$

What is a reasonable model of the system?



1-Coin Model  
(Observable Markov Model)  
 $O = H \ H \ T \ T \ H \ T \ H \ H \ T \ T \ H \dots$   
 $S = 1 \ 1 \ 2 \ 2 \ 1 \ 2 \ 1 \ 1 \ 2 \ 2 \ 1 \dots$



2-Coins Model  
(Hidden Markov Model)  
 $O = H \ H \ T \ T \ H \ T \ H \ H \ T \ T \ H \dots$   
 $S = 2 \ 1 \ 1 \ 2 \ 2 \ 2 \ 1 \ 2 \ 2 \ 1 \ 2 \dots$

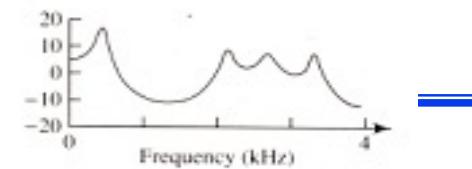
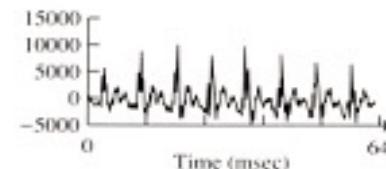
$$P(H) = P_1$$

$$P(T) = 1-P_1$$

$$P(H) = P_2$$

$$P(T) = 1-P_2$$

i (eve)



# Elements of a discrete HMM

- $N$ : the number of states
  - states,  $s = \{s_1, s_2, \dots, s_N\}$
  - state at time  $t$ ,  $q_t \in s$
- $M$ : the number of observation symbols
  - observation symbols,  $v = \{v_1, v_2, \dots, v_M\}$
  - observation at time  $t$ ,  $o_t \in v$
- $A = \{a_{ij}\}$ : state transition probability distribution
  - $a_{ij} = P(q_{t+1} = s_j | q_t = s_i), 1 \leq i, j \leq N$
- $B = \{b_j(k)\}$ : observation probability distribution in state  $j$ 
  - $b_j(k) = P(O_t = v_k | q_t = s_j), 1 \leq j \leq N, 1 \leq k \leq M$
- $\pi = \{\pi_i\}, 1 \leq i \leq N$ : initial state distribution
- For convenience, we use the notation:  $\lambda = (A, B, \pi)$

# Three basic HMM problems

1. **Scoring:** Given an observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  and a model  $\lambda = \{A, B, \pi\}$ , how to compute  $P(O | \lambda)$ , the probability of the observation sequence? → **The Forward-Backward Algorithm**
2. **Matching:** Given an observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  and a model  $\lambda = \{A, B, \pi\}$ , how to choose a state sequence  $q = \{q_1, q_2, \dots, q_T\}$  which is optimum in some sense? → **The Viterbi Algorithm**
3. **Training:** Given an observation sequence  $O = \{o_1, o_2, \dots, o_T\}$  and an (untrained) model  $\lambda = \{A, B, \pi\}$ , how to adjust the model parameters  $\lambda = \{A, B, \pi\}$  to maximize  $P(O | \lambda)$ ? → **The Baum-Welch Re-estimation Procedures**

# Problem 1: Scoring

- Given  $O = \{o_1, o_2, \dots, o_T\}$  and  $\lambda = \{A, B, \pi\}$ , how to compute  $P(O | \lambda)$ , the probability of the observation sequence? (probability evaluation)
  - Consider all possible state sequences ( $N^T$ ) of length  $T$ :

$$\begin{aligned}
 P(O | \lambda) &= \sum_{\text{all } q} P(O | q, \lambda) P(q | \lambda) \\
 &= \sum_{q_1, q_2, \dots, q_T} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \dots a_{q_{T-1} q_T} b_{q_T}(O_T)
 \end{aligned}$$

- Calculation required  $\approx N^T \cdot 2T$ 
  - For a toy example,  $N = 6, T = 100$ , the number of computations is  $2 \cdot 100 \cdot 6^{100} \approx 1.3 \times 10^{80}$ !

The number of atoms in the universe?

# The forward algorithm

- Consider the forward variable  $\alpha_t(i)$  defined as

$$\alpha_t(i) = P(o_1 o_2 \dots o_t, q_t = i | \lambda)$$

i.e., the probability of the partial observation sequence until time  $t$  and state at time  $t$  being  $i$ , given the model  $\lambda$

- We can solve for  $\alpha_t(i)$  inductively as follows:

1. Initialisation  $\alpha_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$

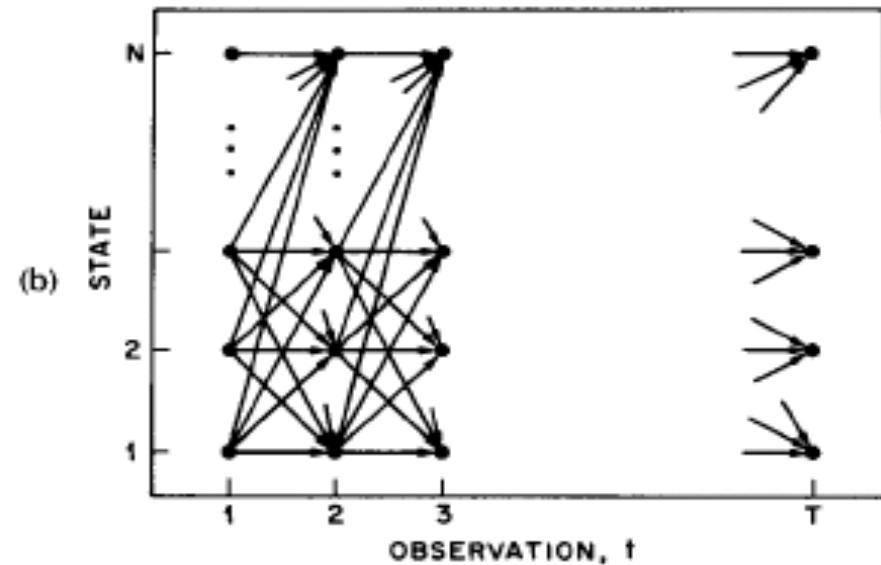
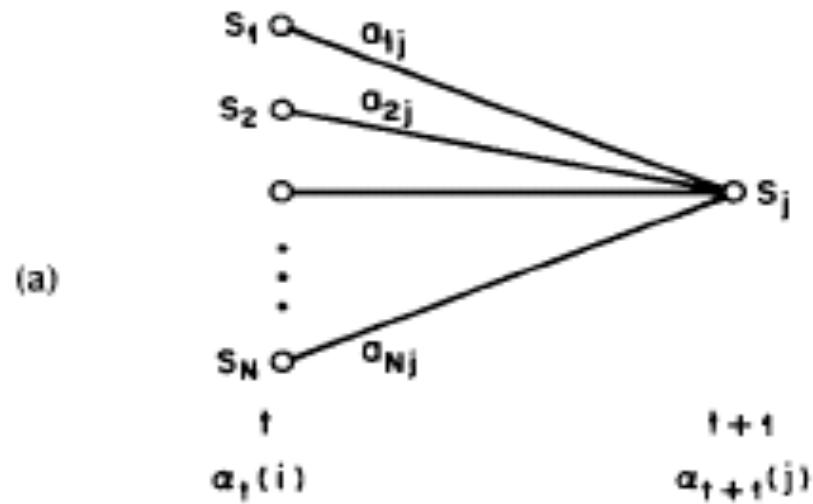
2. Induction  $\alpha_{t+1}(j) = \left[ \sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(o_{t+1}), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N$

3. Termination  $P(O | \lambda) = \sum_{i=1}^N \alpha_T(i)$

Calculation  $\approx T \cdot N^2$ . For  $N=6, T=100, 3600$ , instead of  $10^{80}$

# Illustration of forward algorithm

(Rabiner, 1989)



**Fig. 4.** (a) Illustration of the sequence of operations required for the computation of the forward variable  $\alpha_{t+1}(j)$ .  
 (b) Implementation of the computation of  $\alpha_t(i)$  in terms of a lattice of observations  $t$ , and states  $i$ .

# The backward algorithm

- Similarly, consider the backward variable  $\beta_t(i)$  defined as

$$\beta_t(i) = P(o_{t+1} o_{t+2} \dots o_T \mid q_t = i, \lambda)$$

i.e., the probability of the partial observation sequence from time  $t+1$  to the end, given state  $i$  at time  $t$  and model  $\lambda$

- We can solve for  $\beta_t(i)$  inductively as follows:

- Initialisation  $\beta_T(i) = 1, \quad 1 \leq i \leq N$

- Induction

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j),$$

- Termination

$$P(O \mid \lambda) = \sum_{i=1}^N \pi_i b_i(o_1) \beta_1($$

- Again, calculation  $\approx T \cdot N^2$ .

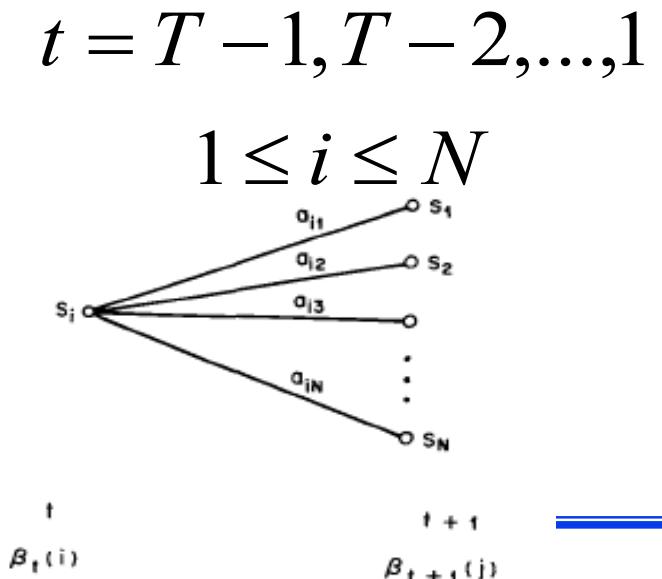
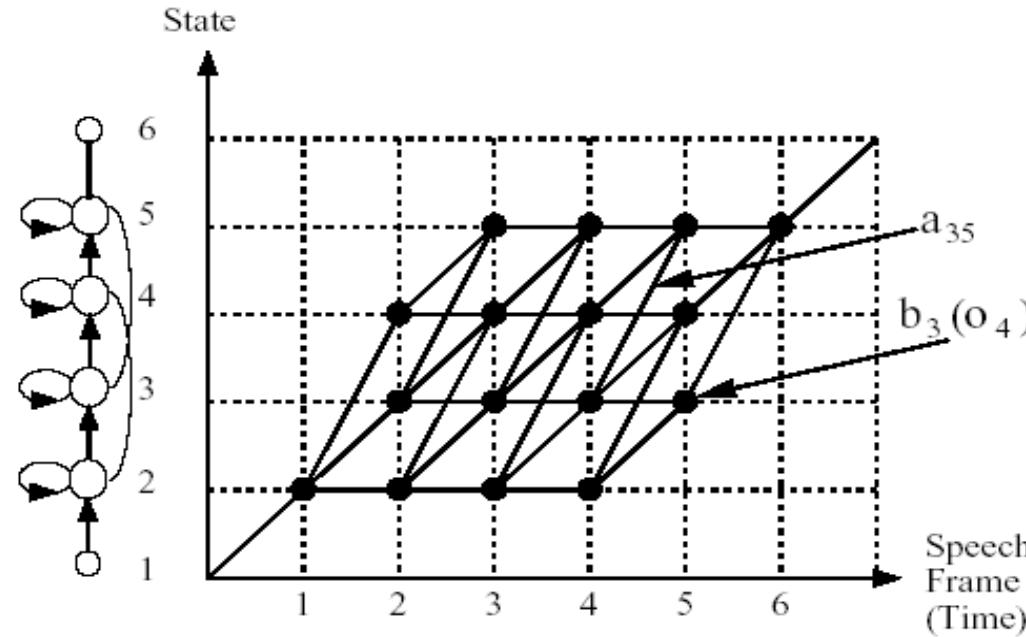


Fig. 5. Illustration of the sequence of operations required for the computation of the backward variable  $\beta_t(i)$ .

## Problem 2: Matching

- Given  $O = \{o_1, o_2, \dots, o_T\}$  and a model  $\lambda = \{A, B, n\}$ , how to choose a state sequence  $q = \{q_1, q_2, \dots, q_T\}$  which is optimum in some sense? (“Optimal” state sequence)



Trellis diagram for an Isolated Word Recognition task.

# Finding optimal state sequence

- One optimality criterion is to choose the states  $q_t$  that are **individually** most likely at each time  $t$ 
  - Define the probability of being in state  $i$  at time  $t$ , given the observation sequence  $O$ , and the model  $\lambda$

$$\gamma_t(i) = P(q_t = i | O, \lambda) = \frac{P(O, q_t = i | \lambda)}{P(O | \lambda)} = \frac{P(O, q_t = i | \lambda)}{\sum_{i=1}^N P(O, q_t = i | \lambda)}$$

Since  $P(O, q_t = i | \lambda) = \alpha_t(i)\beta_t(i)$

$$\text{We have } \gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^N \alpha_t(i)\beta_t(i)}$$

- The individually most likely state  $q_t^*$  at time  $t$  is

$$q_t^* = \arg \max_{1 \leq i \leq N} [\gamma_t(i)]$$

# Finding optimal state sequence (cont'd)

- The **individual** optimality criterion has the problem that the optimum state sequence may not obey state transition constraints →  
The “optimal” state sequence may not even be a valid sequence ( $a_{ij}=0$  for some  $i$  and  $j$ )
- Another optimality criterion is to find the single best state sequence (path), i.e., to maximize  $P(q, O|\lambda)$  →
  - The **Viterbi** algorithm - a method based on dynamic programming

# The Viterbi algorithm

- To find the best path  $q = \{q_1, q_2, \dots, q_T\}$ , for given  $O = \{o_1, o_2, \dots, o_T\}$ , we define the best score (highest probability) along a single path, at time  $t$ ,

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1 q_2 \dots q_{t-1}, q_t = i, o_1 o_2 \dots o_t \mid \lambda)$$

which accounts for the first  $t$  observations and ends in state  $i$ .

Then

$$\delta_{t+1}(j) = \max_i [\delta_t(i) a_{ij}] \cdot b_j(o_{t+1})$$

# The Viterbi algorithm (cont'd)

**1. Initialisation**  $\delta_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N$

$$\psi_1(i) = 0$$

**2. Recursion**

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(o_t), \quad 2 \leq t \leq T \quad 1 \leq j \leq N$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], \quad 2 \leq t \leq T \quad 1 \leq j \leq N$$

Find best  $i$  for each  $j$

**3. Termination**  $P^* = \max_{1 \leq i \leq N} [\delta_T(i)]$

$$q_T^* = \arg \max_{1 \leq i \leq N} [\delta_T(i)]$$

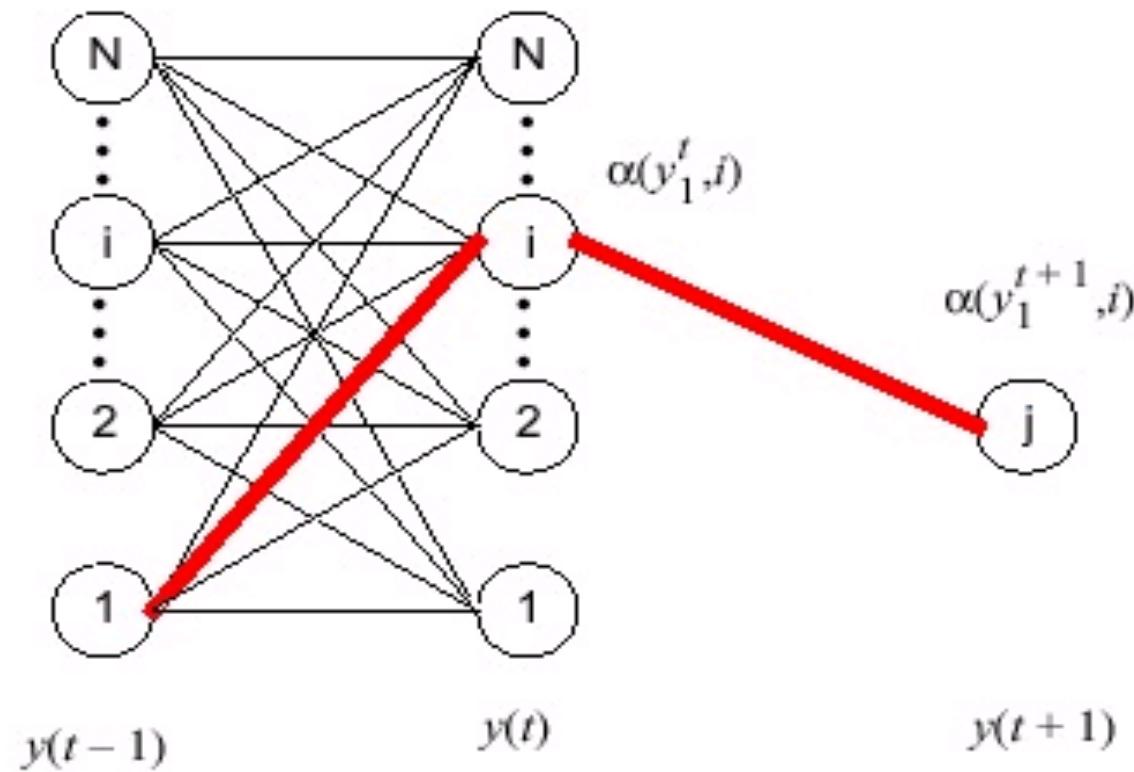
**4. Path (state sequence) backtracking**

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \dots, 1$$

# The Viterbi algorithm (cont'd)

(Joseph Picone)

The *Viterbi algorithm* can also be used to find the best state sequence. Note that the principal difference is that we model the overall sequence probability by the probability of the single best path:



# The power of recursive equation

Computing factorials  $n!$

Method 1. simply calculate  $n!$  for each  $n$

Method 2. use  $n! = n(n-1)!$

if  $F(n) = n!$  then

$F(n) = nF(n-1)$  for  $n \geq 1$

(Recursive Equation)

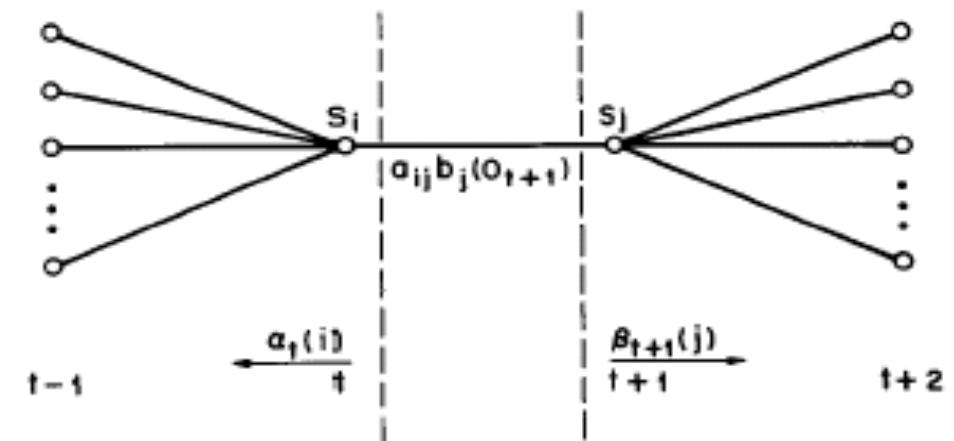
# Problem 3: Training

- How to tune the model parameters  $\lambda = \{A, B, \pi\}$  to maximize  $P(O | \lambda)$ ? - a learning problem
  - No efficient algorithm for global optimisation
  - Effective iterative algorithm for local optimisation: **the Baum-Welch re-estimation**
- **Baum-Welch**
  - = forward-backward algorithm (Baum, 1972)
  - is a special case of **EM** (expectation-maximization) algorithm
  - computes probabilities using current model  $\lambda$ ;
  - refines  $\lambda$  to  $\bar{\lambda}$  such that  $P(O | \lambda)$  is locally maximised
  - uses  $\alpha$  and  $\beta$  from forward-backward algorithm

# Baum-Welch re-estimation

Define  $\xi_t(i, j)$ , the probability of being in state  $i$  at time  $t$ , and state  $j$  at time  $t+1$ , given  $\lambda$  and  $O$ , i.e.

$$\begin{aligned}\xi_t(i, j) &= P(q_t = i, q_{t+1} = j | \mathbf{O}, \lambda) \\ &= \frac{P(q_t = i, q_{t+1} = j, \mathbf{O} | \lambda)}{P(\mathbf{O} | \lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{P(\mathbf{O} | \lambda)} \\ &= \frac{\alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^N \sum_{j=1}^N \alpha_t(i) a_{ij} b_j(\mathbf{o}_{t+1}) \beta_{t+1}(j)}\end{aligned}$$



**Fig. 6.** Illustration of the sequence of operations required for the computation of the joint event that the system is in state  $S_i$  at time  $t$  and state  $S_j$  at time  $t + 1$ .

# Baum-Welch Re-estimation (cont'd)

- Recall that  $\gamma_t(i)$  is defined as the probability of being in state  $i$  at time  $t$ , given the entire observation sequence and the model, so

$$\gamma_t(i) = P(q_t = i \mid O, \lambda) = \sum_{j=1}^N P(q_t = i, q_{t+1} = j \mid O, \lambda) = \sum_{j=1}^N \xi_t(i, j)$$

- Sum  $\gamma_t(i)$  and  $\xi_t(i, j)$  over  $t$ , we have

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from state } i \text{ in } \mathbf{O}$$

$$(\sum_{t=1}^T \gamma_t(i)) = \text{the expected number of times that state } i \text{ is visited.}$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transitions from state } i \text{ to state } j \text{ in } \mathbf{O}$$

# Baum-Welch re-estimation formulas

$\bar{\pi}_i$  = expected frequency (number of times) in state  $i$

at time ( $t = 1$ ) =  $\gamma_1(i)$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

$$= \frac{\sum_{t=1}^T \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} = \frac{\sum_{t=1}^T \gamma_t(j) \cdot \mathbf{1}(o_t = v_k)}{\sum_{t=1}^T \gamma_t(j)}$$

# Continuous density HMMs

- Replaces the discrete observation probabilities,  $b_j(k)$ , by a continuous **PDF** (probability density function)  $b_j(x)$
- The PDF  $b_j(x)$  is often represented as a mixture of Gaussians:

$c_{jk}$  is the mixture weight,  $c_{jk} \geq 0$ , and  $\sum_{k=1}^M c_{jk} = 1$

$$b_j(\mathbf{x}) = \sum_{k=1}^M c_{jk} N[\mathbf{x}, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}] \quad 1 \leq j \leq N$$

$N$  is the normal density

The mean and covariance matrix associated with state  $j$  and mixture  $k$

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. **Graphical models**
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 11a: Graphical Models

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <http://kom.aau.dk/~zt>

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. **Graphical models**
12. Algorithm-independent machine learning
13. Reinforcement learning

# Probabilistic graphical models

- A graph: nodes - random variables; links - probabilistic relationships.
- Joint distribution over all of the random variables is decomposed into a product of factors each depending only on a subset of the variables.
- Classes of graph
  - Bayesian networks, aka directed graphical models (GM), expressing causal relationships.
  - Markov random fields, aka undirected graphical models, expressing soft constraints.
  - Factor graph, converted from directed GM and undirected GM to solve inference problems.

# Probabilistic graphical models

- An intuitive way of representing and **visualising** the relationships between many variables
- Allow to abstract out the **conditional independence** relationships between the variables.
  - Thus, we can answer questions like: “Is A dependent on B given that we know the value of C?” just by looking at the graph.
  - Graphical models allow us to define general message-passing algorithms that implement probabilistic **inference** efficiently.
    - Thus, we can answer queries like “What is  $p(A|C = c)$ ?” without enumerating all settings of all variables in the model.

Graphical models = statistics  $\times$  graph theory  $\times$  computer science[/EE]. [Ghahramani]

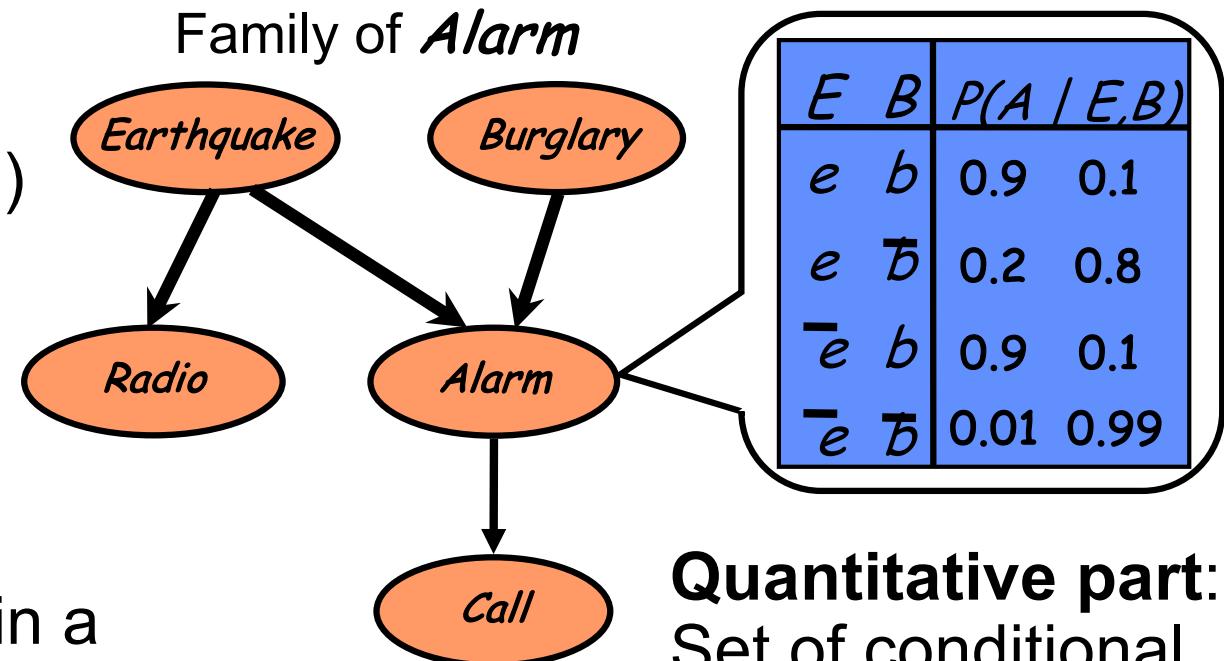
# A Bayesian network (Bayes net)

Compact representation of joint probability distributions via **conditional** independence.

**Qualitative part:**

Directed acyclic graph (DAG)

- Nodes - random vars.
- Edges - direct influence; statistical dependencies.



**Together:**

Define a unique distribution in a factored form

$$P(B, E, A, C, R) = P(B)P(E)P(A | B, E)P(R | E)P(C | A)$$

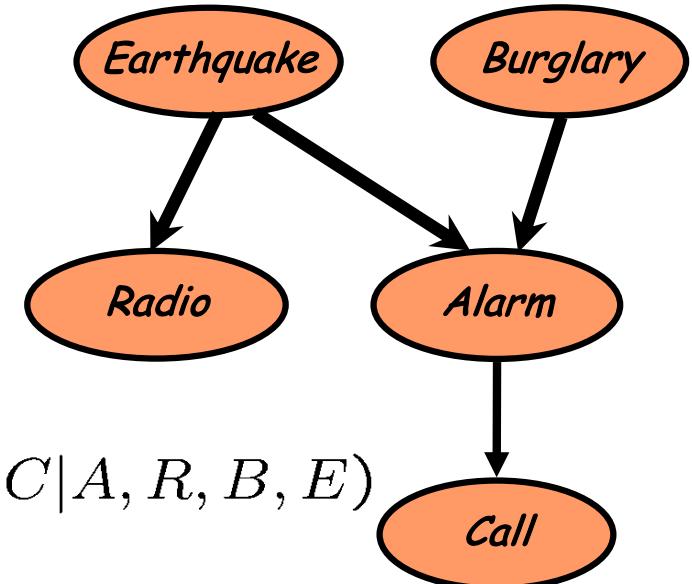
**Quantitative part:**  
Set of conditional probability distributions

[Murphy]

# A Bayesian network - cont.

A node is **conditionally independent** of its ancestors given its parents

$$\begin{aligned}
 P(E, B, R, A, C) &= P(E)P(B|E)P(R|B, E)P(A|R, B, E)P(C|A, R, B, E) \\
 &= P(E)P(B)P(R|E)P(A|B, E)P(C|A)
 \end{aligned}$$



From  $2^5 - 1 = 31$  parameters to  $1+1+2+4+2=10$

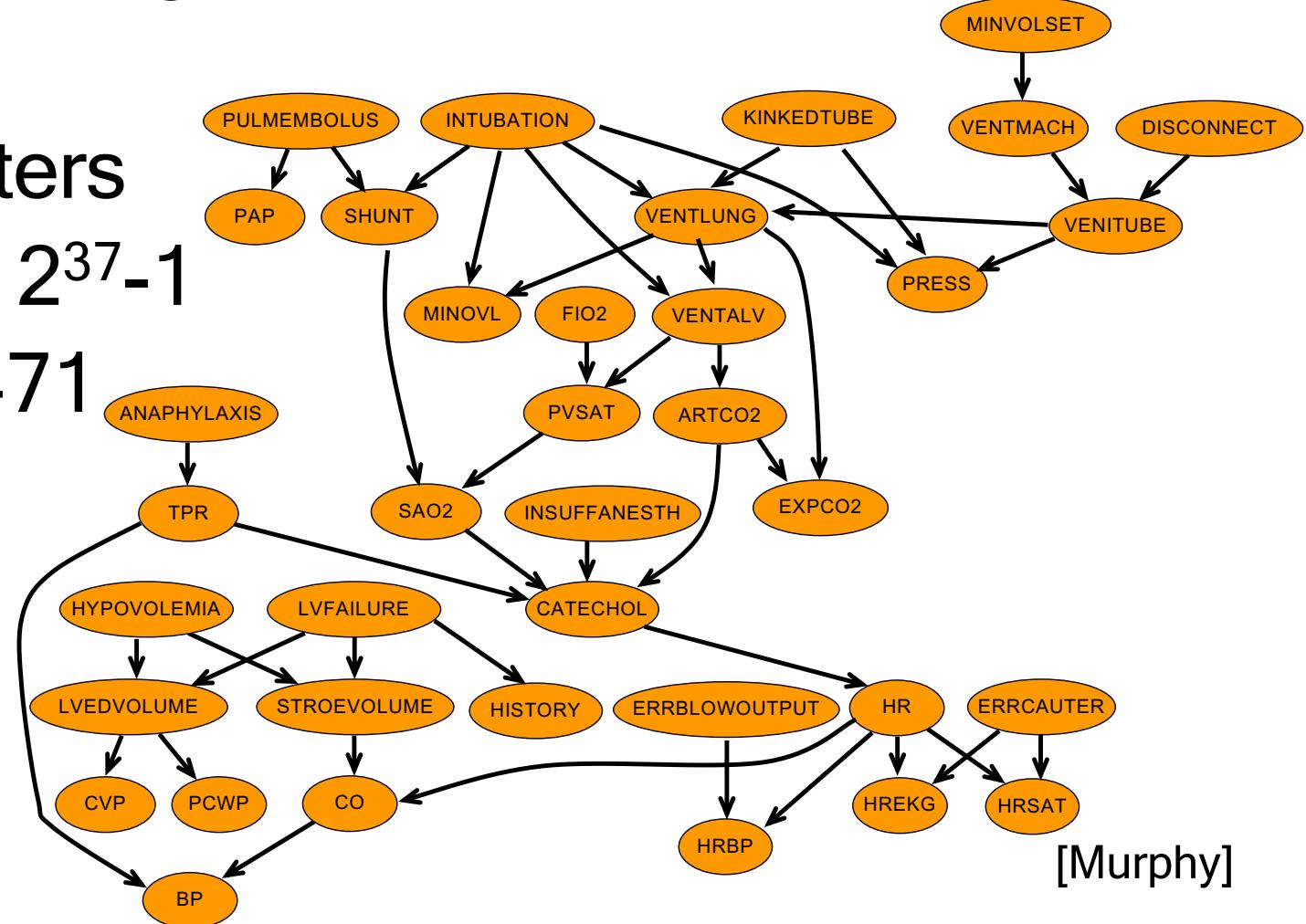
(Binary case.)

[Murphy]

# A real Bayesian network: Alarm

# Domain: Monitoring Intensive-Care Patients

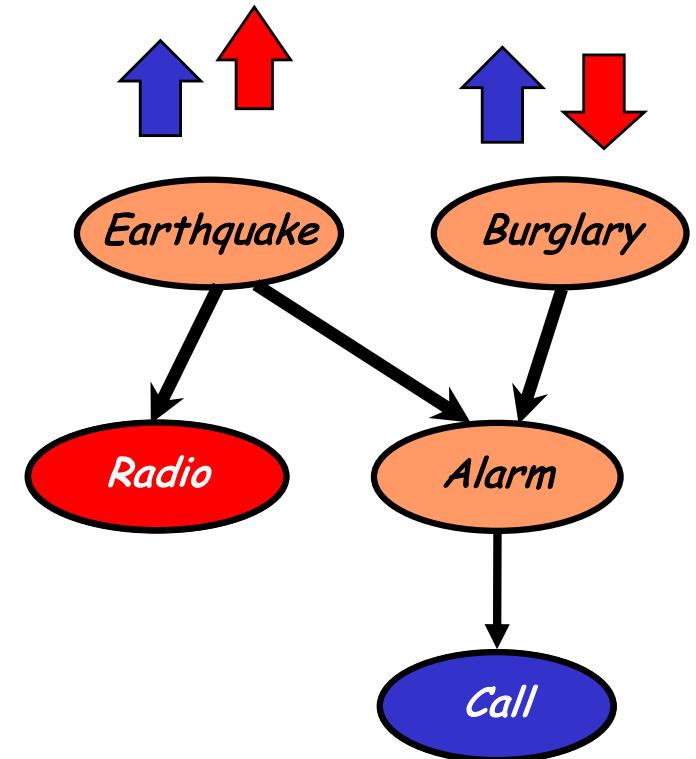
- 37 variables
  - 509 parameters  
...instead of  $2^{37}-1$



# Bayesian networks - Inference

- Posterior probabilities
  - Probability of any event given any evidence
- Most likely explanation
  - Scenario that explains evidence

Explaining away effect



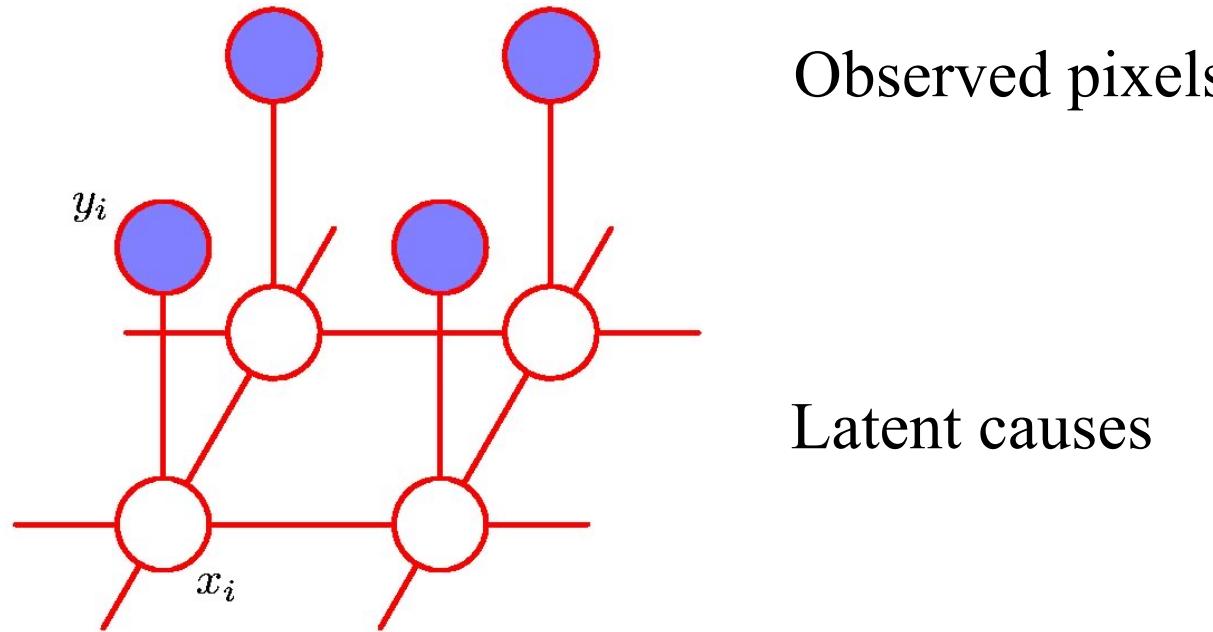
[Murphy]

# Why are Bayesian networks useful?

- Graph structure supports
  - Modular representation of knowledge
  - Local, distributed algorithms for inference and learning
  - Intuitive (possibly causal) interpretation.
- Factored representation may have exponentially fewer parameters than full joint  $P(X_1, \dots, X_n) \Rightarrow$ 
  - lower sample complexity (less data for learning)
  - lower time complexity (less time for inference)

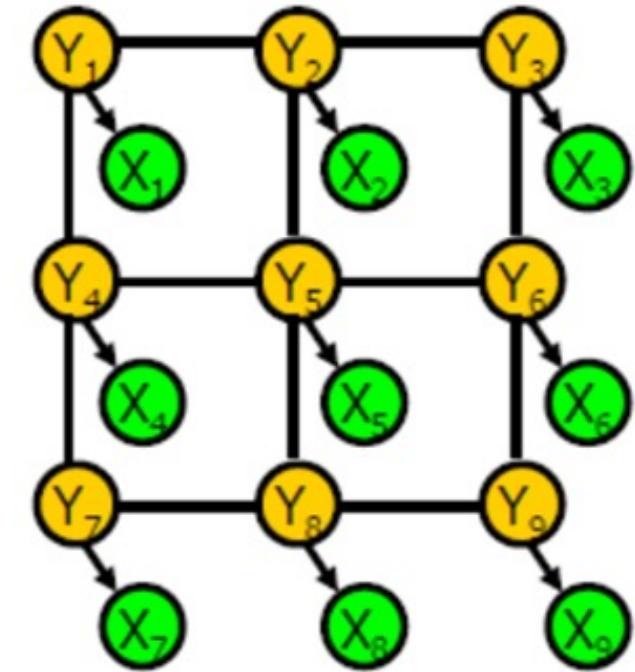
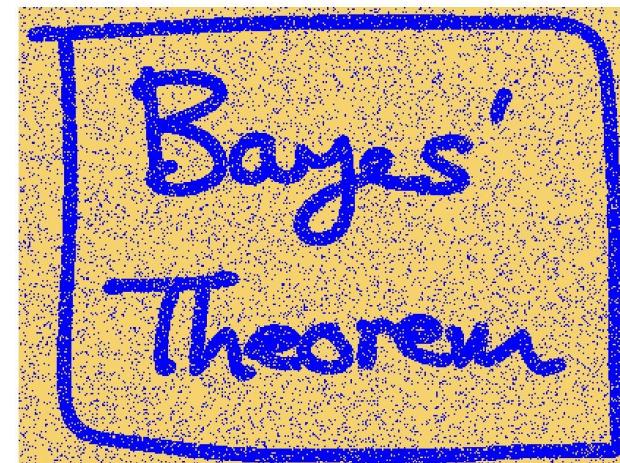
[Murphy]

# Markov random field (Markov net)



Estimate  $P(x_1, \dots, x_n | y_1, \dots, y_n)$

# Markov random field - cont.



$X_i$ : noisy pixels  
 $Y_i$ : “true” pixels

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. **Graphical models**
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 11

---

### PATTERN RECOGNITION AND MACHINE LEARNING

### CHAPTER 8: GRAPHICAL MODELS

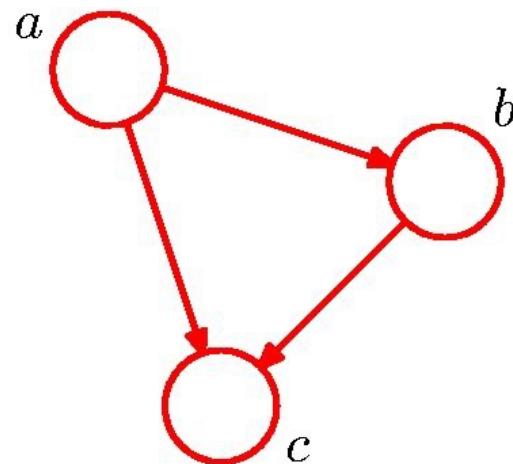
---

# Outline

- Bayesian networks (BNs)
- Conditional independence
- Markov random fields (MRFs)

# Bayesian Networks

## Directed Acyclic Graph (DAG)



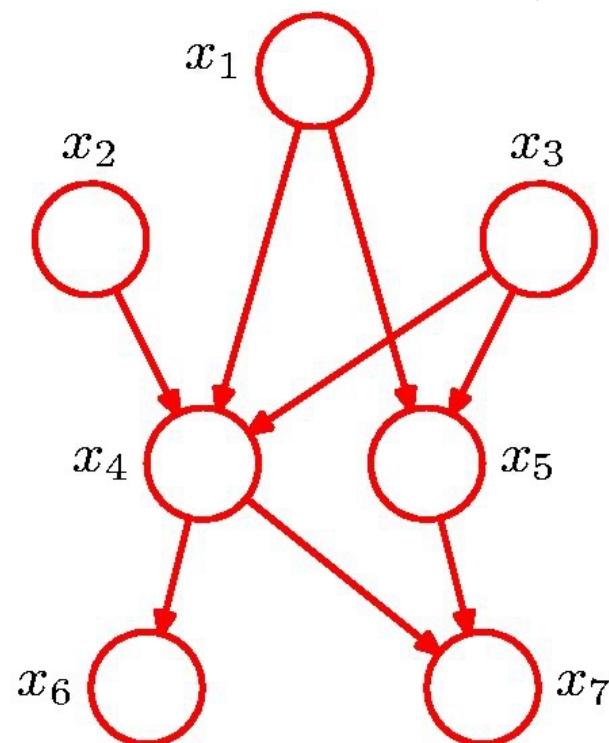
By applying product rule of probability:

$$p(a, b, c) = p(c|a, b)p(a, b) = p(c|a, b)p(b|a)p(a)$$

In general,  $p(x_1, \dots, x_K) = p(x_K|x_1, \dots, x_{K-1}) \dots p(x_2|x_1)p(x_1)$

# Bayesian Networks – cont.

$$p(x_1, \dots, x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3) \\ p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5)$$



General Factorization

$$p(\mathbf{x}) = \prod_{k=1}^K p(x_k | \text{pa}_k) \quad (8.5)$$

$\text{pa}_k$  is the set of parents of  $x_k$

# Outline

- Bayesian networks (BNs)
- Conditional independence
- Markov random fields (MRFs)

# Conditional Independence

---

$a$  is independent of  $b$  given  $c$

$$p(a|b, c) = p(a|c)$$

Equivalently

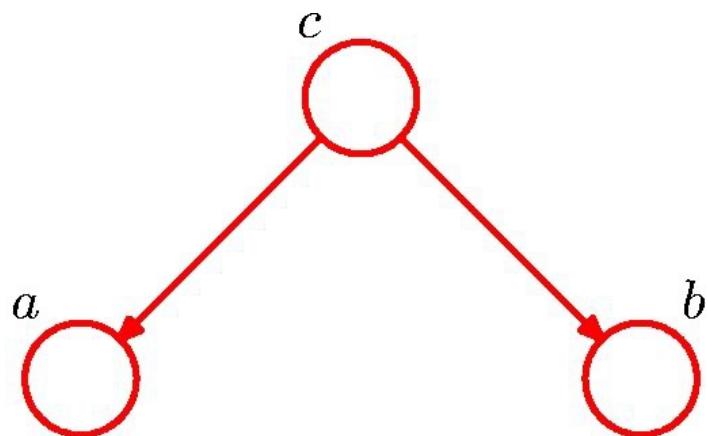
$$\begin{aligned} p(a, b|c) &= p(a|b, c)p(b|c) \\ &= p(a|c)p(b|c) \end{aligned}$$

Notation

$$a \perp\!\!\!\perp b \mid c$$

# Conditional Independence: Example 1

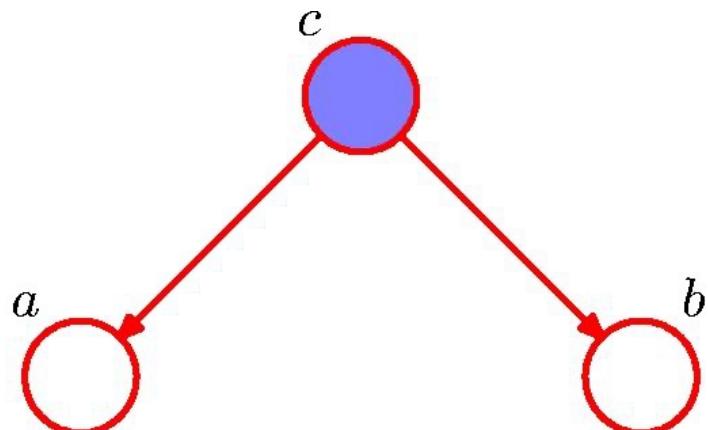
---



$$p(a, b, c) = p(a|c)p(b|c)p(c)$$

$$p(a, b) = \sum_c p(a|c)p(b|c)p(c)$$

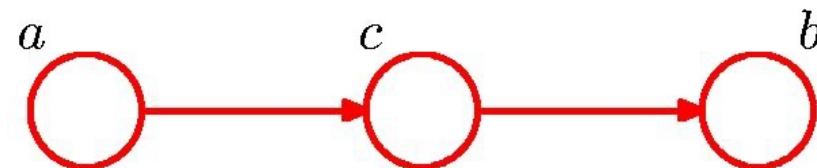
$$a \not\perp\!\!\!\perp b \mid \emptyset$$



$$\begin{aligned} p(a, b|c) &= \frac{p(a, b, c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

$$a \perp\!\!\!\perp b \mid c$$

# Conditional Independence: Example 2

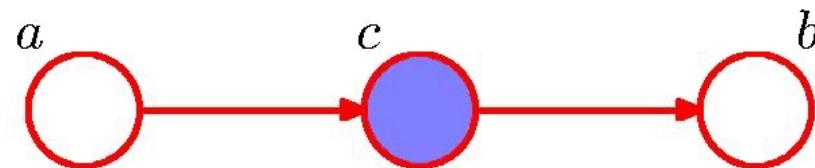


$$p(a, b, c) = p(a)p(c|a)p(b|c)$$

$$p(a, b) = p(a) \sum_c p(c|a)p(b|c) = p(a)p(b|a)$$

$$a \not\perp\!\!\!\perp b \mid \emptyset$$

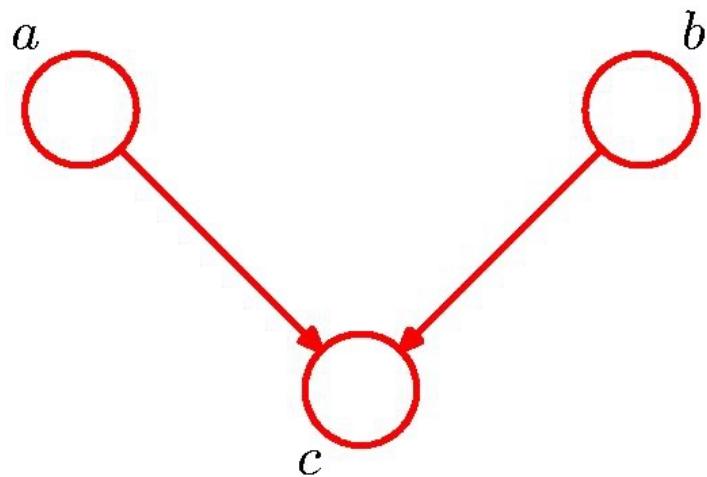
# Conditional Independence: Example 2



$$\begin{aligned} p(a, b | c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(a)p(c|a)p(b|c)}{p(c)} \\ &= p(a|c)p(b|c) \end{aligned}$$

$$a \perp\!\!\!\perp b \mid c$$

# Conditional Independence: Example 3



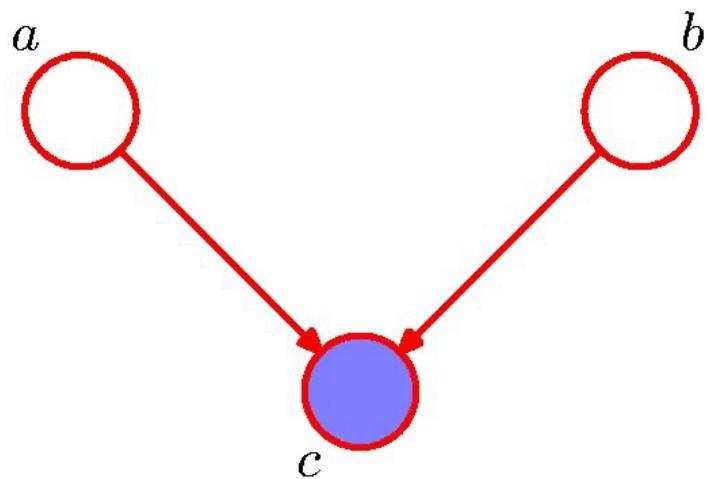
$$p(a, b, c) = p(a)p(b)p(c|a, b)$$

$$p(a, b) = p(a)p(b)$$

$$a \perp\!\!\!\perp b \mid \emptyset$$

Note: this is the opposite of Example 1, with c unobserved.

# Conditional Independence: Example 3



$$\begin{aligned} p(a, b | c) &= \frac{p(a, b, c)}{p(c)} \\ &= \frac{p(a)p(b)p(c|a, b)}{p(c)} \end{aligned}$$

$$a \not\perp\!\!\!\perp b \mid c$$

Note: this is the opposite of Example 1, with  $c$  observed.

# Outline

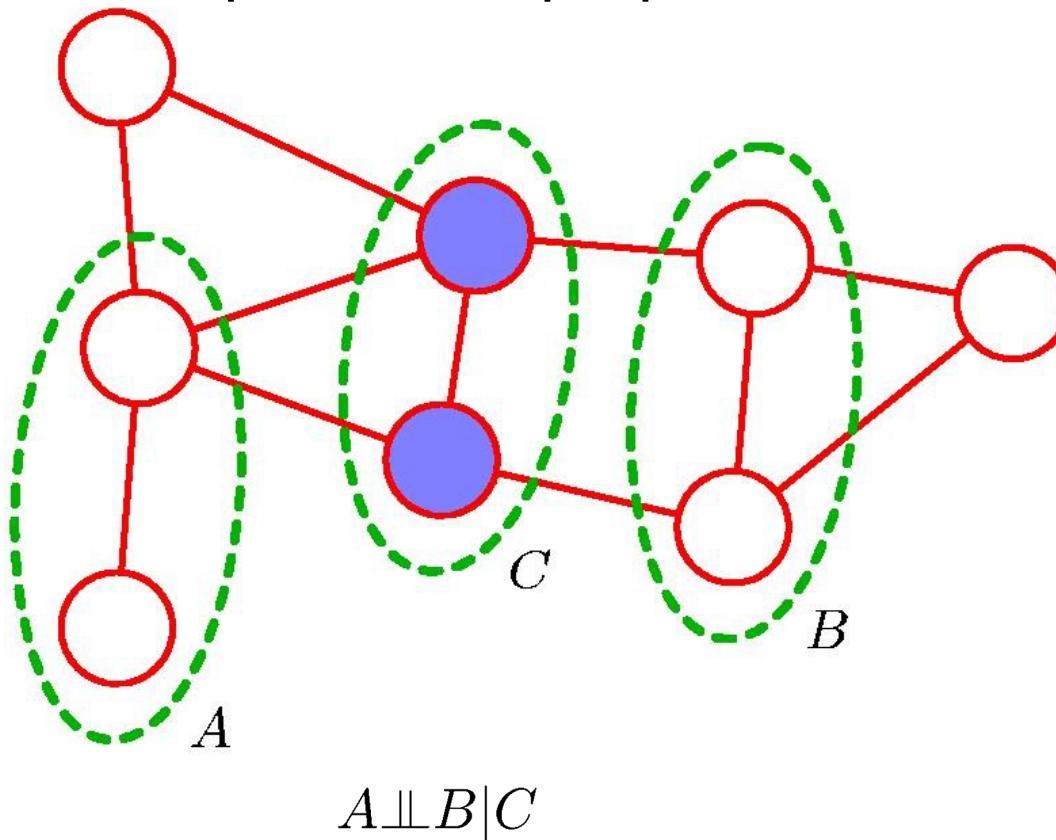
- Bayesian networks (BNs)
- Conditional independence
- Markov random fields (MRFs)

# Markov Random Fields

---

Undirected graph model of joint probability distribution.

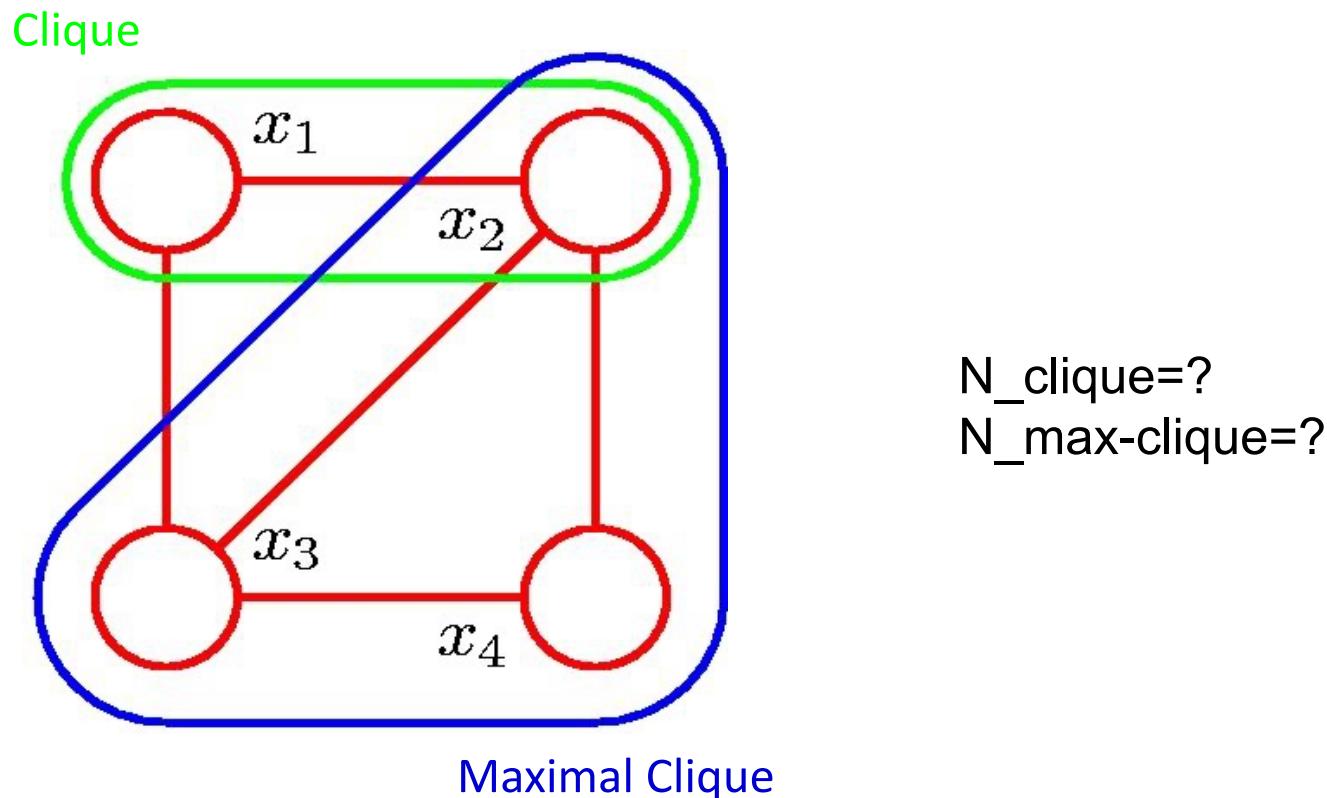
Conditional independence properties:



# Cliques and Maximal Cliques

---

Factorization properties: The factors in the decomposition of the joint distribution are functions of the variables in the cliques.



# Joint Distribution

---

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

$$p(\mathbf{x}) = \prod_{k=1}^K p(x_k | \text{pa}_k)$$

where  $\psi_C(\mathbf{x}_C)$  is the potential over clique C and

$$Z = \sum_{\mathbf{x}} \prod_C \psi_C(\mathbf{x}_C)$$

is the normalization coefficient.

Potential functions (positive) are usually given in terms of exponentials, more specifically Gibbs/Boltzmann distributions.

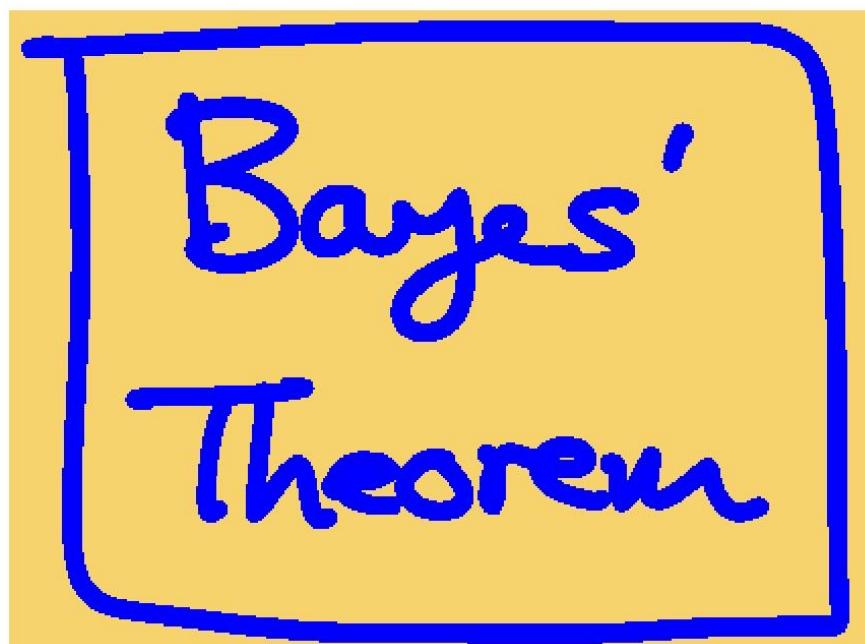
The Boltzmann distribution as a function of Energy E

$$\psi_C(\mathbf{x}_C) = \exp \{-E(\mathbf{x}_C)\}$$

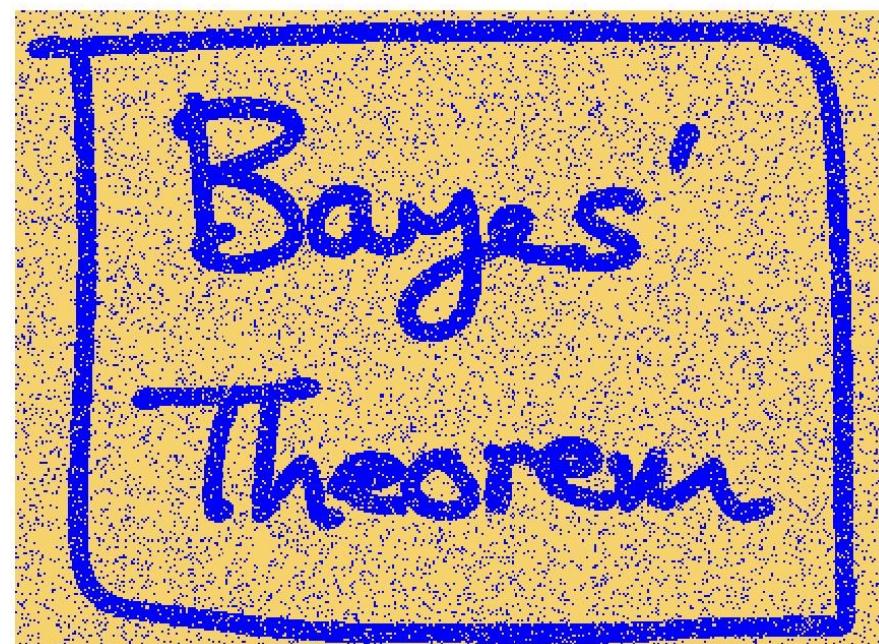
large energy means low probability

# Illustration: Image De-Noising (1)

---



Original Image

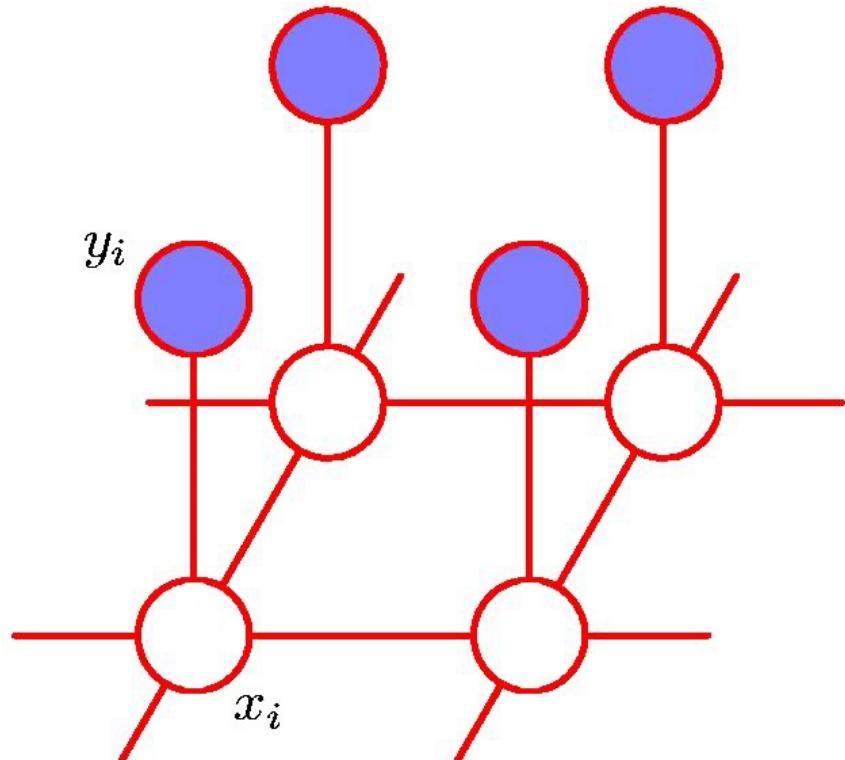


Noisy Image

# Illustration: Image De-Noising (2)

---

Observed image



Noise-free image

$$E(\mathbf{x}, \mathbf{y}) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j - \eta \sum_i x_i y_i$$

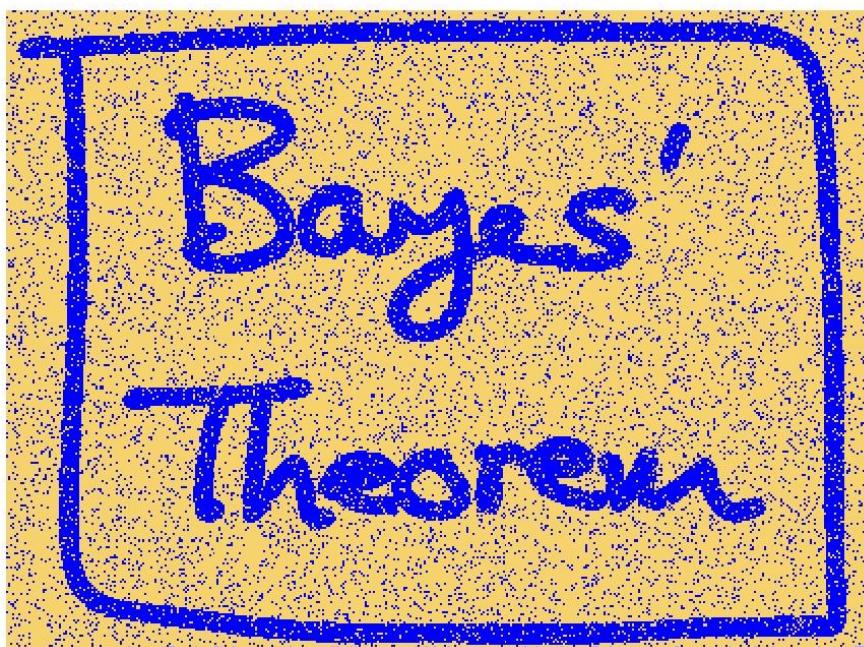
where i and j are indices of neighbouring pixels

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{y})\}$$

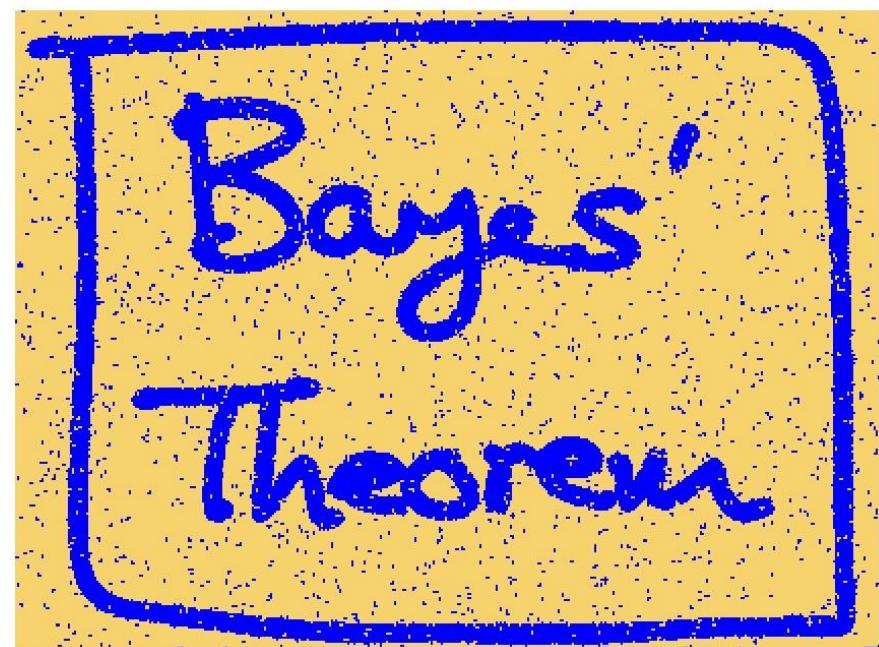
- large energy means low probability
- small energy means large probability

# Illustration: Image De-Noising (3)

---



Noisy Image



Restored Image by using  
Iterated conditional Modes (ICM)

# Outline

- Bayesian networks (BN's)
- Conditional independence
- Markov random fields (MRFs)

# Machine Learning

## Lecture 12: Algorithm Independent Machine Learning

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <http://kom.aau.dk/~zt>

Primarily based on Alpaydin, *Introduction to Machine Learning*, Bishop, *Pattern Recognition and Machine Learning*, Duda, Hart, Stork, *Pattern Classification*

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. **Algorithm-independent machine learning**
13. Reinforcement learning

# Algorithm-independent machine learning

- Lack of Inherent Superiority of Any Classifier
  - No Free Lunch Theorem
- Bias and Variance
- Resampling for
  - classifier design (ensemble learners: Boosting, AdaBoost)
  - validating models/classifiers for model selection (Cross-Validation)
- Combining Classifiers

# Algorithm-independent machine learning

- **Mathematical foundations** that do not depend upon any particular classifier or learning algorithm used, e.g.,
    - bias and variance dilemma
    - *no free lunch theorem*
  - **Techniques** that can be used in conjunction with different learning algorithms or providing guidelines in their use, e.g.,
    - *cross validation* for estimating and comparing classifiers
    - resampling for estimating statistics (Bootstrap) and for classifier design (Bagging, Boosting, *AdaBoost*)
    - combining classifiers (*decision tree*)
- (Duda et al. 2001)

# Algorithm-independent machine learning

- A hypothesis (no free lunch theorem)
  - No pattern recognition method is inherently superior to any other, or even to random guessing; it is the type of problem, prior distribution, and other information that determine which form of classifier should provide the best performance.
- How to estimate, compare and adjust the ‘match’ between a learning algorithm and the problem it addresses (e.g., cross validation)
- How to combine models (e.g., adaboost, decision trees)

# No free lunch theorem (NFLT)

- Questions
  - Without prior assumptions, can we expect any classification method to be superior or inferior overall?
  - Can we even find an algorithm that is overall superior to (or inferior to) random guessing?
- Answer is NO! This is according to NFLT.
  - <http://www.no-free-lunch.org/>
  - It's a proof that, averaged over all problems, no search/optimization algorithm is expected to outperform any other algorithm.

# No free lunch for binary data

- $h_1$ : assign all test data to class1
- $h_2$ : assign all test data to class2
- Test set error rates:
  - $E_1=40\%, E_2=60\%$ , leading to
  - $h_1$  superior to  $h_2$
- However, to compare the algorithms overall, we must average over all possible target functions  $F(x)$  consistent with the training data, incl.  $F(x)$  for all test data being -1 or +1. Then, there will be no difference in test set error rates.

Training	$x$	$F(x)$	$h_1$	$h_2$
	000	1	1	1
$D$	001	-1	-1	-1
	010	1	1 33%	1 66%
Test	011	-1	1	-1
	100	1	1	-1
	101	-1	1	-1
	110	1	1	-1
	111	1	1 40%	-1 60%

all +1 ?	0%	100%
all -1 ?	100%	0%

# No free lunch theorem - cont'd

- This is because almost all machine learning algorithms make some assumptions (known as inductive or learning bias) and accordingly introduce bias into the model.
- The assumptions made by algorithms mean that some algorithms will fit certain data sets better than others.
- How effective a model will be is directly dependent on how well the assumptions made by the model fit the true nature of the data.
- Most important aspects:
  - Prior information, data distribution, amount of training data, and cost or reward functions.

# No free lunch theorem - cont'd

- This justifies a healthy skepticism regarding studies that claim to demonstrate the overall superiority of a particular learning algorithm.
- Experience with a broad range of techniques is the best insurance for solving arbitrary new classification problems.
- No-Free Lunch: In the absence of assumptions we should not prefer any learning or classification algorithm over another.
- Ugly Duckling Theorem: In the absence of assumptions there is no “best” feature representation.

# Algorithm-independent machine learning

- Lack of Inherent Superiority of Any Classifier
  - No Free Lunch Theorem
- Bias and Variance
- Resampling for
  - classifier design (ensemble learners: Boosting, AdaBoost)
  - validating models/classifiers for model selection (Cross-Validation)
- Combining Classifiers

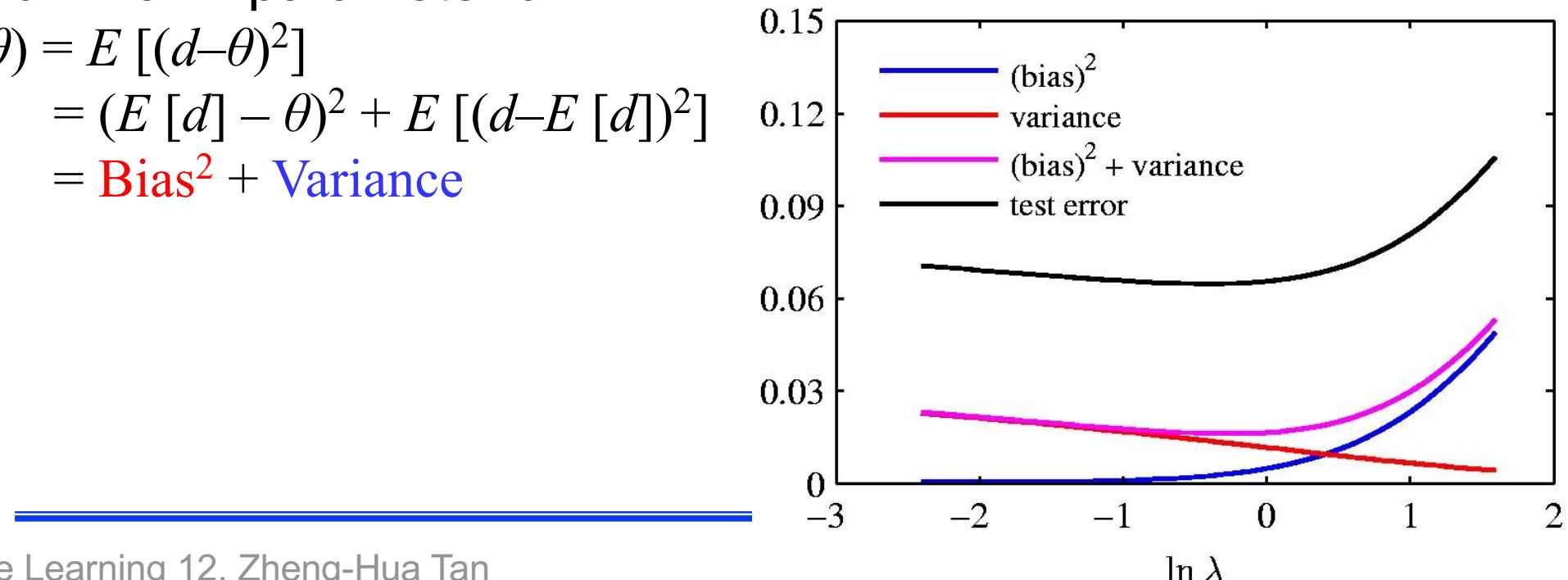
# Bias-variance trade-off

- There is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance.

Mean square error of the estimator

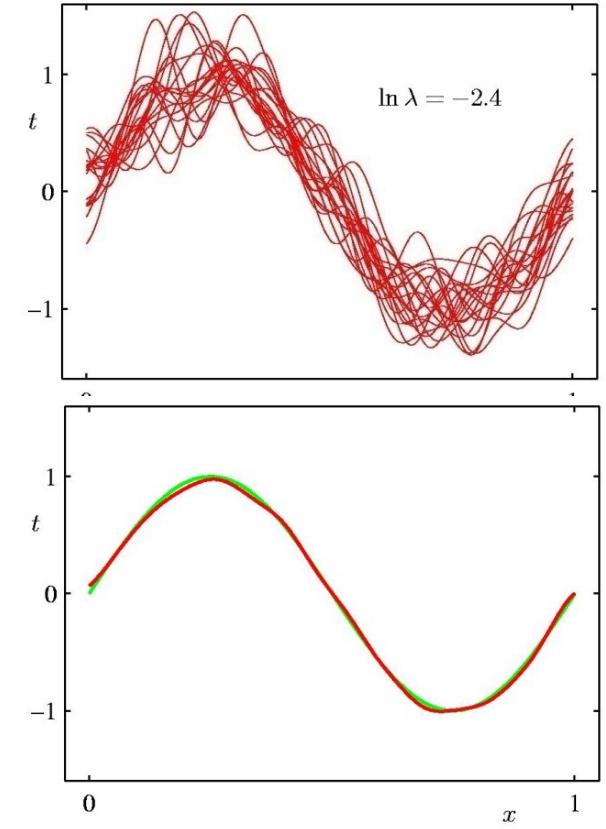
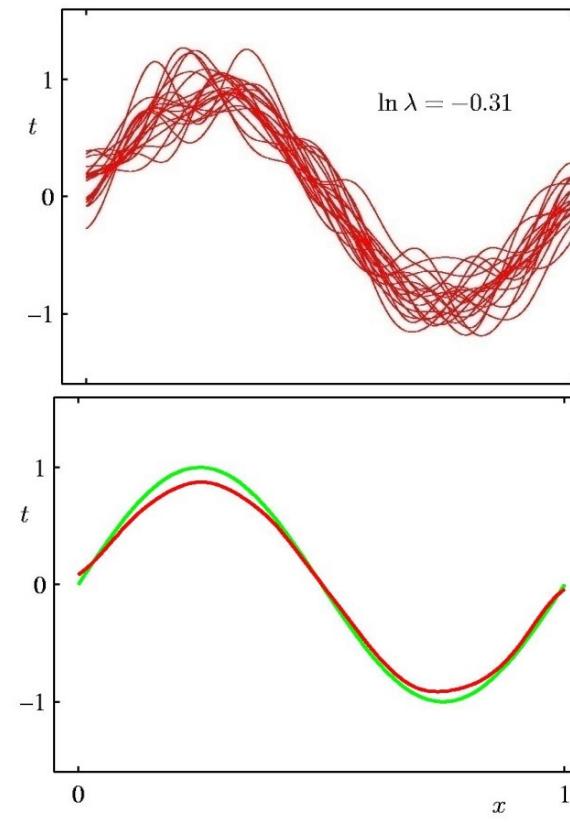
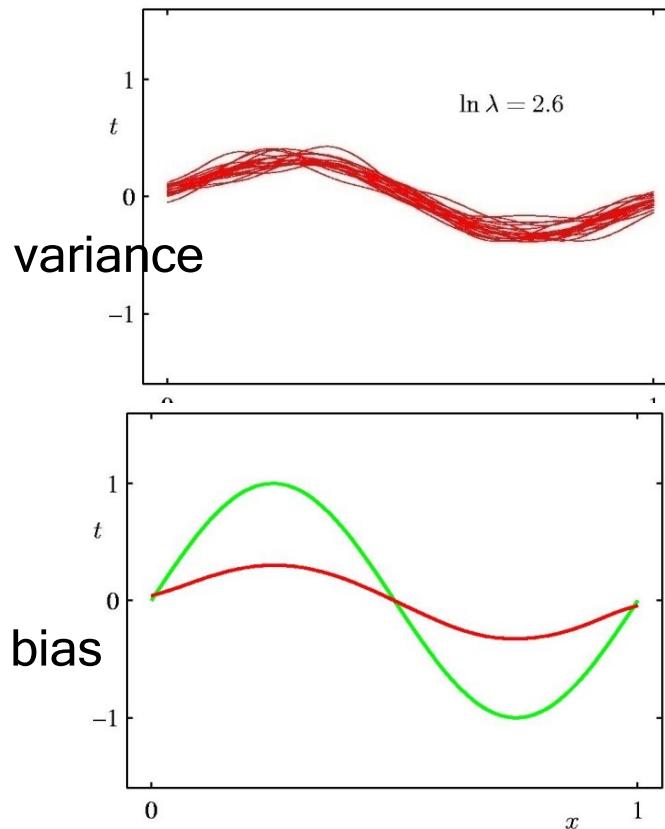
$d$  for unknown parameter  $\theta$ :

$$\begin{aligned}
 r(d, \theta) &= E[(d - \theta)^2] \\
 &= (E[d] - \theta)^2 + E[(d - E[d])^2] \\
 &= \text{Bias}^2 + \text{Variance}
 \end{aligned}$$



# Number of data sets for 9<sup>th</sup> order

- 100 data sets; training multiple polynomials and then averaging them, the contribution from the variance term tended to cancel, leading to improved predictions. from Bishop
- The dependence of bias and variance on model complexity

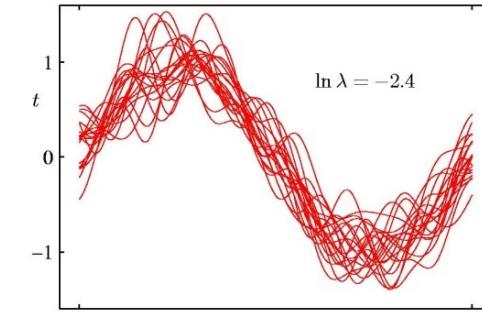


# Bias-variance dilemma

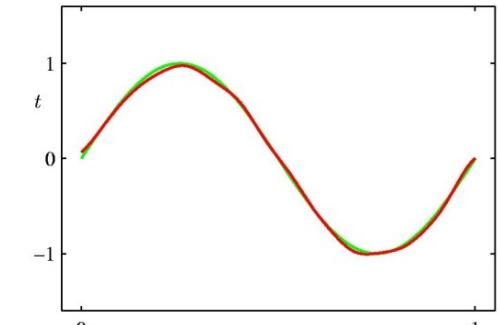
- Procedures with increased flexibility to adapt to the training data (e.g., having more free parameters) tend to have lower bias but higher variance.
- The best way to get low bias and low variance is to have **prior information** about the target function.
- These considerations of bias and variance help to clarify why
  - We seek to have accurate prior information about the solution form, and large training sets;
  - The match of the algorithm to the problem is crucial.

# Averaging solutions wrt multiple datasets

- If we have 100 data sets?
- Training multiple polynomials and then averaging them, the contribution from the variance term tended to cancel, leading to improved predictions.
- When averaging a set of low-bias models -> accurate predictions.
- You can reduce the variance but not the bias.
- In practice, we have only a single data set -> **resampling** to beat the bias-variance dilemma.



only 20 of the 100 fits shown



Averaging 100 fits

# Algorithm-independent machine learning

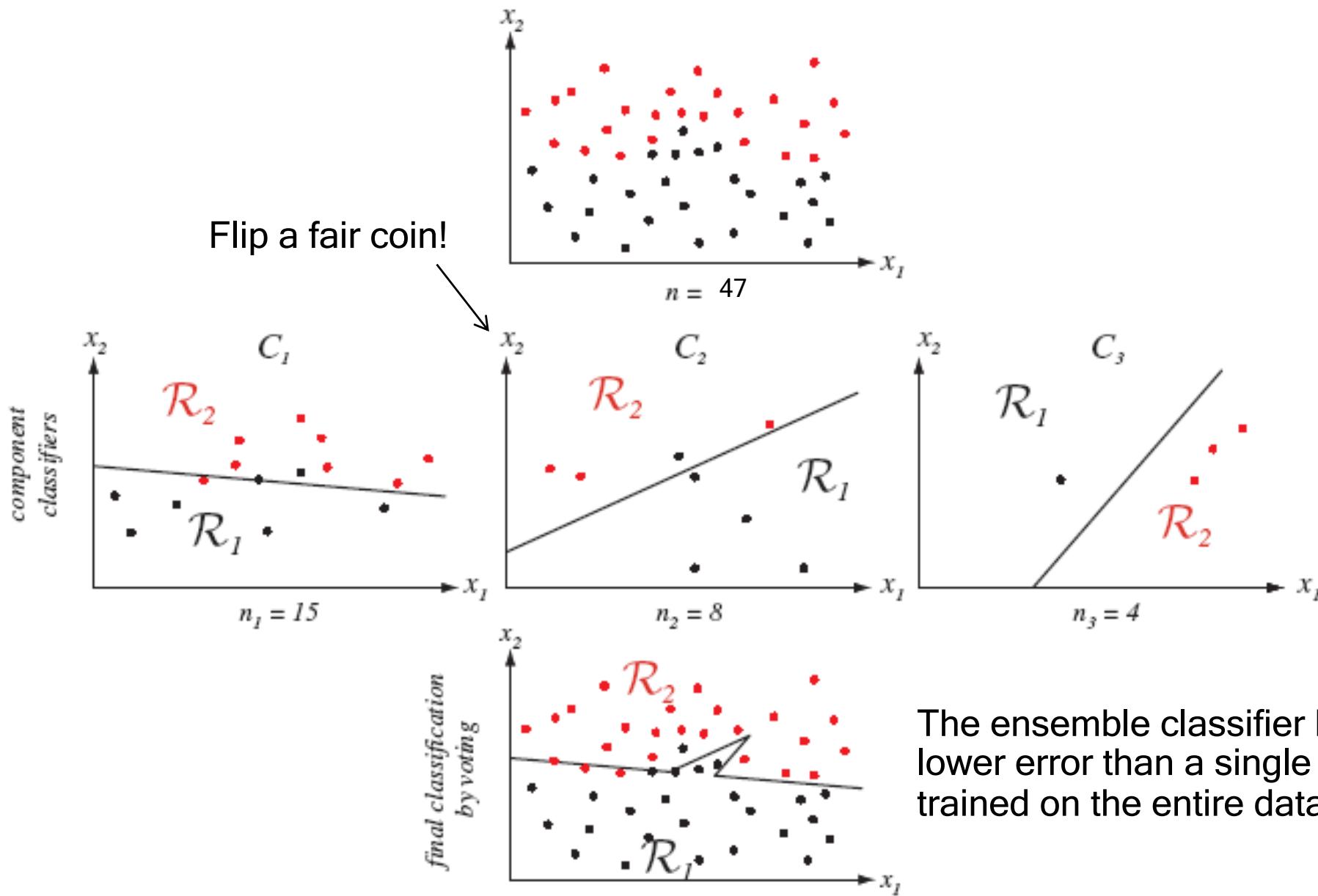
- Lack of Inherent Superiority of Any Classifier
  - No Free Lunch Theorem
- Bias and Variance
- Resampling for
  - classifier design (ensemble learners: Boosting, AdaBoost)
  - validating models/classifiers (Cross-Validation)
- Combining Classifiers

# Basic boosting

- Goal is to "boost" the performance and accuracy of a learning algorithm.
  - Randomly select  $n_1 < n$  instances (**without replacement**),  $X_1$ , and train the 1<sup>st</sup> classifier  $d_1$  with  $X_1$ .  $d_1$  only need be a weak learner - slightly better than chance, or benefit of boosting will be small.
  - Construct a 2<sup>nd</sup> training set  $X_2$ : half correctly classified by  $d_1$ , half incorrectly, from the remaining instances of  $X$ . Train  $d_2$  with  $X_2$ .
  - Construct a 3<sup>rd</sup> training set  $X_3$ : instances with which  $d_1$  and  $d_2$  disagree, from the remaining instances of  $X$ . Train  $d_3$  with  $X_3$ .
  - Test: if  $d_1$  and  $d_2$  agree, that's it; otherwise,  $d_3$ 's output.
- It builds multiple "experts" that specialize on different types of patterns, by training new machines on samples that the previously trained machines found difficult.

# Basic boosting for 2D two-category problem

Flip a fair coin!



The ensemble classifier has lower error than a single classifier trained on the entire dataset.

# AdaBoost

AdaBoost (adaptive boosting) - a most popular variation.

- Allows to keep adding component learners until some target training error is reached, not limited to 3.
- Each training instance receives a weight that determines its probability of being selected for a component classifier.
- Initially uniform weights. For correctly classified instance, its chance of being used again is reduced; otherwise, chance raised. In this way, it focuses on informative or difficult patterns.
- On each iteration  $k$ , draw a training set randomly according to these weights.
- Uses over and over the same training set.



AdaBoost is an algorithm for constructing a "strong" classifier as linear combination of "simple" "weak" classifiers

# AdaBoost (a variant of it)

Training:

For all  $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$ , initialize  $p_1^t = 1/N$

For all base-learners  $j = 1, \dots, L$

Randomly draw  $\mathcal{X}_j$  from  $\mathcal{X}$  with probabilities  $p_j^t$

Train  $d_j$  using  $\mathcal{X}_j$

For each  $(x^t, r^t)$ , calculate  $y_j^t \leftarrow d_j(x^t)$

Calculate error rate:  $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If  $\epsilon_j > 1/2$ , then  $L \leftarrow j - 1$ ; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each  $(x^t, r^t)$ , decrease probabilities if correct:

If  $y_j^t = r^t$ ,  $p_{j+1}^t \leftarrow \beta_j p_j^t$  Else  $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t$ ;  $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

Testing:

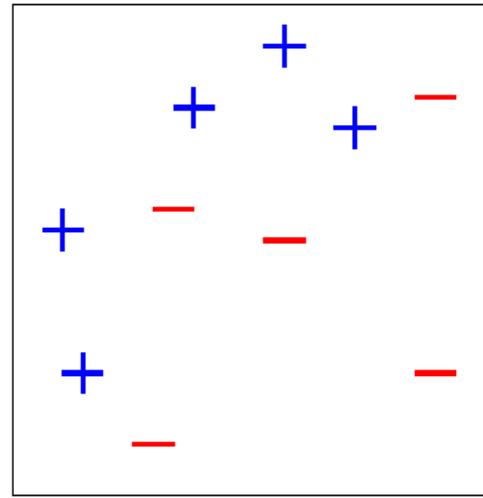
Given  $x$ , calculate  $d_j(x), j = 1, \dots, L$

Calculate class outputs,  $i = 1, \dots, K$ :

$$y_i = \sum_{j=1}^L \left( \log \frac{1}{\beta_j} \right) d_{ji}(x)$$

# AdaBoost toy example 2

Round 1



from Freund &  
Schapire, A tutorial  
on Boosting

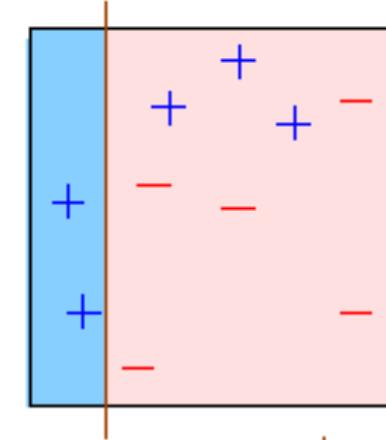
$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

X2

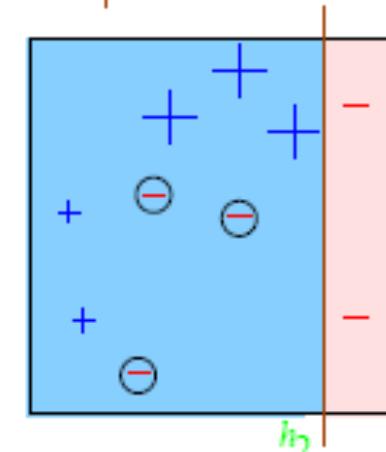
Machine Learning 12, Zhe

Round 2

$$\begin{aligned}\epsilon_1 &= 0.30 \\ \alpha_1 &= 0.42\end{aligned}$$



$$\begin{aligned}\epsilon_2 &= 0.21 \\ \alpha_2 &= 0.65\end{aligned}$$

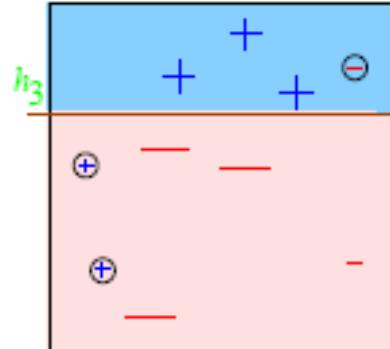
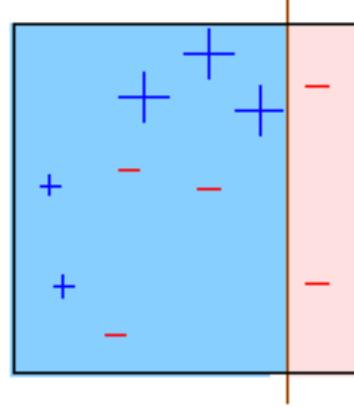
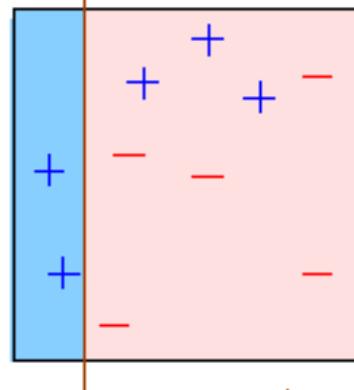


X3

—

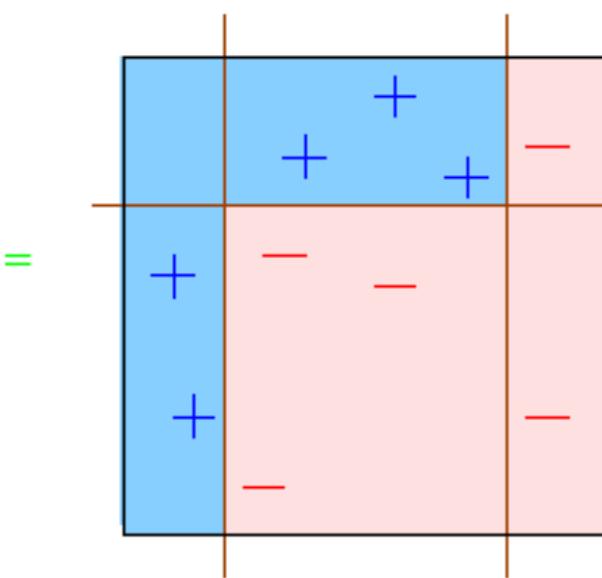
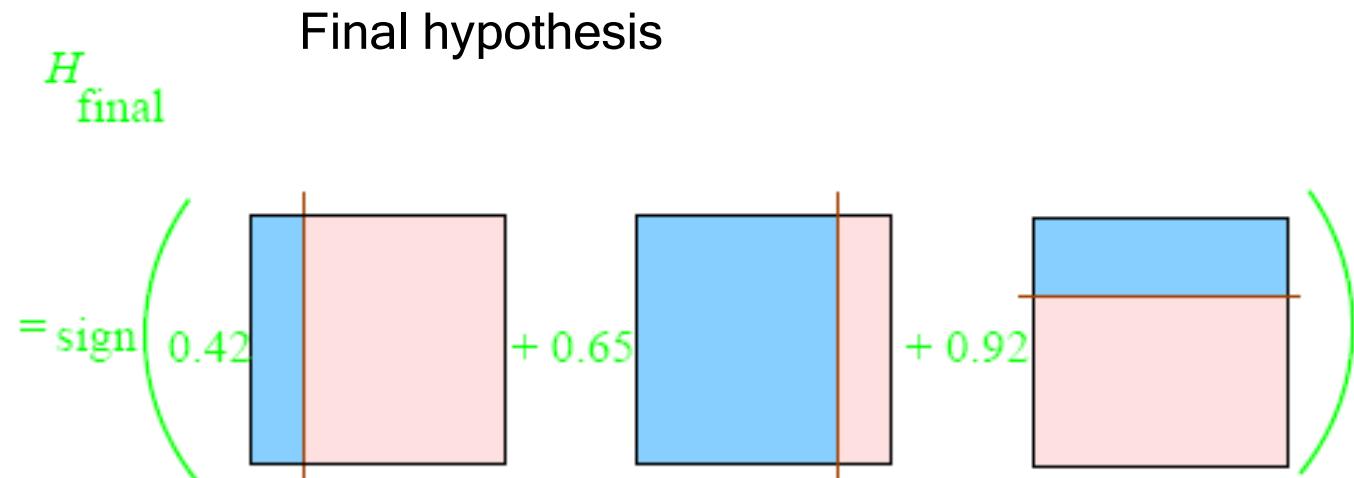
# AdaBoost toy example 2

Round 3



$H_{\text{final}}$

Final hypothesis

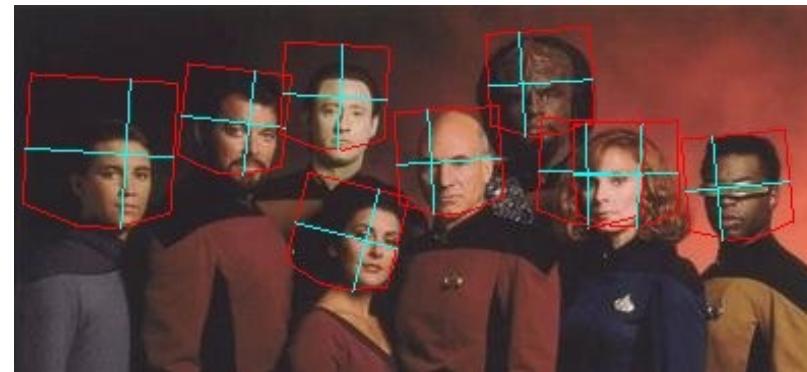


$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

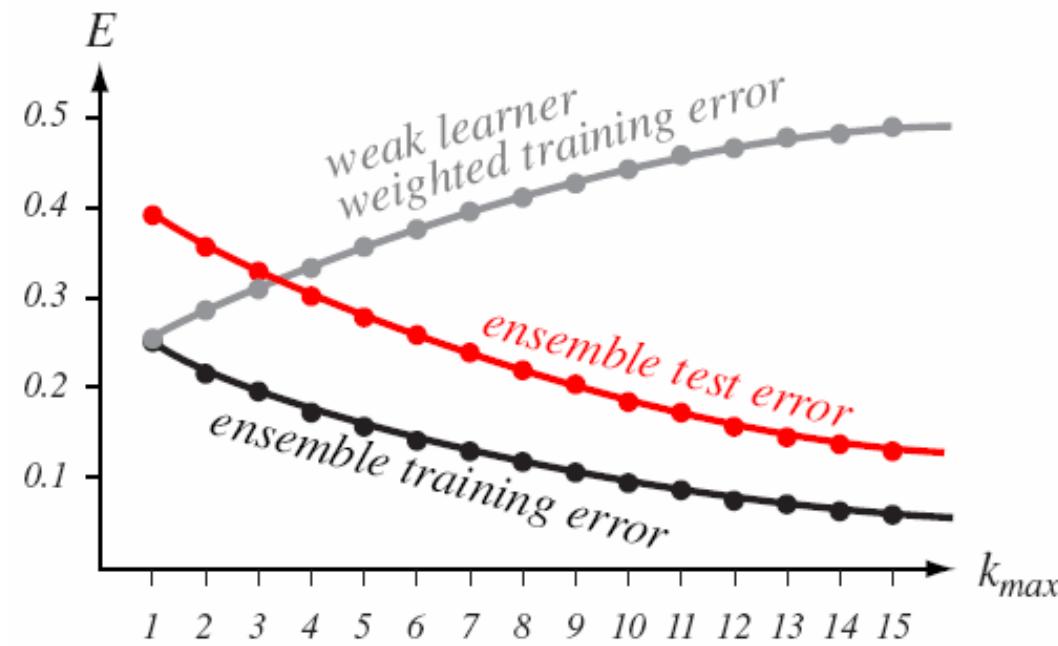
# Real example - face detection

- Viola & Jones face detection - 3 major steps:
  - Feature extraction: rectangular features are used and efficient to compute.
  - Classification: classifier training and feature selection using **AdaBoost** - works very well.
  - Multi-scale detection algorithm



# AdaBoost error rates

- Ensemble error rate keeps decreasing.
- Individual classifiers perform worse on weighted training sets



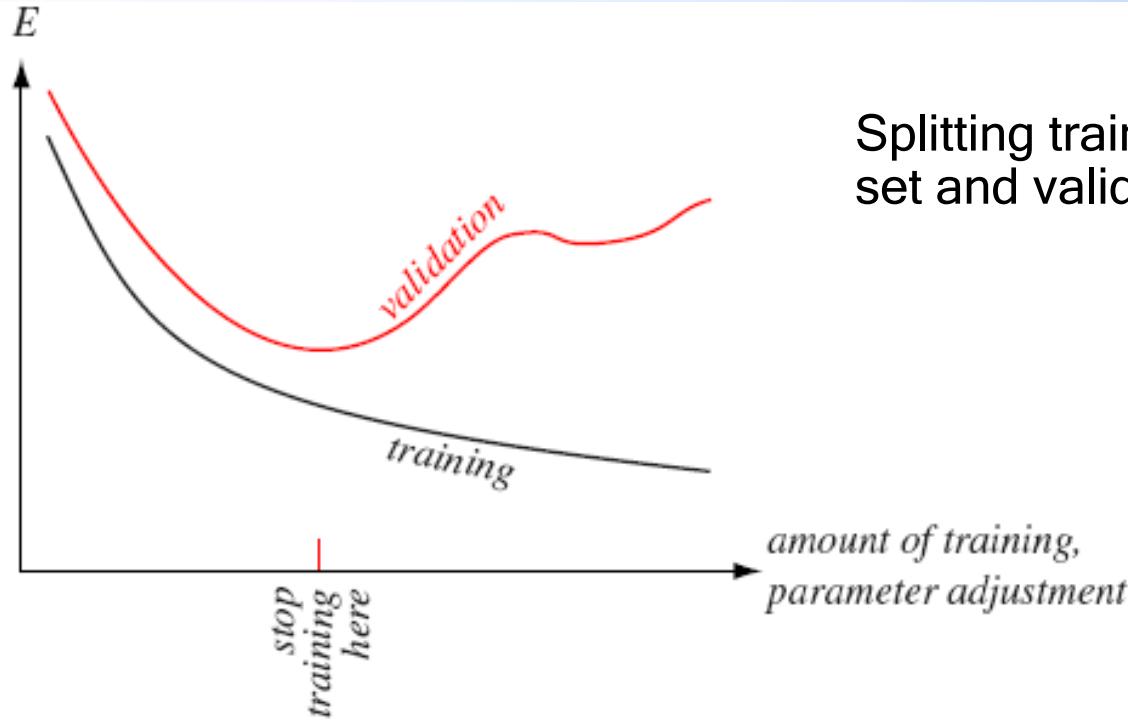
# Algorithm-independent machine learning

- Lack of Inherent Superiority of Any Classifier
  - No Free Lunch Theorem
- Bias and Variance
- Resampling for
  - classifier design (ensemble learners: Boosting, AdaBoost)
  - validating models/classifiers for model selection (Cross-Validation)
- Combining Classifiers

# Model comparison and selection

- Given a training set  $X$ , an algorithm, using empirical risk minimization for model selection, is
  - 1. Train each model  $M_i$  on  $X$ , to get some hypotheses  $h_i$ .
  - 2. Pick the hypothesis with the smallest training error.
- This algorithm does not work.
  - Consider choosing the order of a polynomial. The higher the order of the polynomial, the better it will fit the training set  $X$ , and thus the lower the training error.
  - Hence, this method will always select a high-variance, high-degree polynomial model, which we saw previously is often a poor choice.

# Estimating and comparing classifiers



Splitting training data into training set and validation set.

**FIGURE 9.9.** In validation, the data set  $\mathcal{D}$  is split into two parts. The first (e.g., 90% of the patterns) is used as a standard training set for setting free parameters in the classifier model; the other (e.g., 10%) is the validation set and is meant to represent the full generalization task. For most problems, the training error decreases monotonically during training, as shown in black. Typically, the error on the validation set decreases, but then increases, an indication that the classifier may be overfitting the training data. In validation, training or parameter adjustment is stopped at the first minimum of the validation error. In the more general method of cross-validation, the performance is based on multiple independently formed validation sets. From: Richard O. Duda, Peter E. Hart, and

Machin David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

# Splitting data set

- To avoid overfitting and predict performance, data is divided into 3 sets: training data, validation data, test data
- Training data: a set of examples used for learning, i.e. to fit the parameters with
  - E.g. determining weights in neural networks
- Validation data: a set of examples used to tune the parameters, to avoid overfitting
  - E.g. determining number of iterations in neural network
- Final testing is done on test data. After assessing the final model with the test set, you must not further tune the model.

# Small sample size issues

- Wish to use as much of data for training as possible.
- If validation set is small, one will get noisy predictive performance.
- One solution to this dilemma is to use cross validation.

# k-fold cross validation

- Data ( $N$  instances) is partitioned into  $k$  groups (here  $k=4$ )
  - $k-1$  groups are used to train a set of models that are evaluated on the remaining group
  - Repeat for all  $k$  possible choices for the held-out group
  - Performance for the  $k$  runs are averaged
  - A typical choice for  $k$  is 10. When data is scarce, with  $k=N$  we get the Leave-one-out method
  - Choose the best performing one from the set of models trained.
  - Cross validation can also be used more simply to evaluate a single model or algorithm.
  - Computationally expensive: Number of training runs to be performed increases by a factor of  $k$
-

# Algorithm-independent machine learning

- Lack of Inherent Superiority of Any Classifier
  - No Free Lunch Theorem
- Bias and Variance
- Resampling for
  - estimating statistics (Bootstrap)
  - classifier design (ensemble learners: Bagging, Boosting, AdaBoost)
  - validating models/classifiers (Cross-Validation)
- Combining Classifiers

# Combining Models

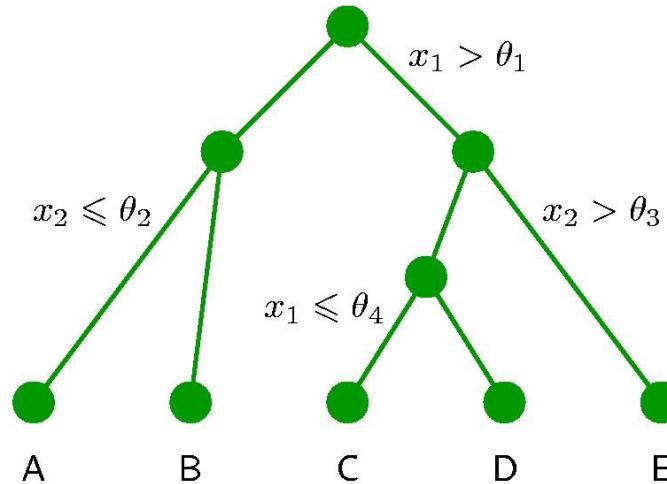
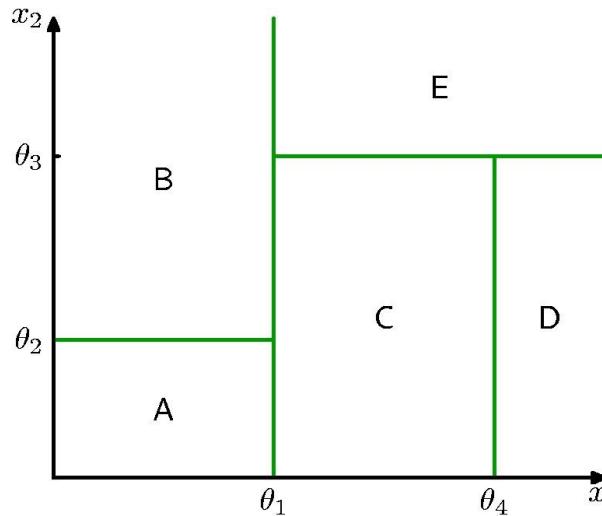
- Aka, mixture of experts, ensemble classifiers
  - Boosting, AdaBoost
  - Useful if each of its component classifiers is highly trained (i.e., an “expert”) in a different region in the feature space.

# Ensemble methods

- Democracy applied to decision making:
  - 1,000,000 learning machines can't be wrong.
- Generate many learning machines that are slightly different from each other and make them vote.
- The key idea of ensemble methods: find a way to make each replica of the learning machine different from the others, yet useful.

# Tree-based models

- Partition the input space into regions, and then assigning a simple model to each region.
- A model combination method where only one model is responsible at any given point in input space.
- Decision tree recursively conducts binary partitioning of input space, along with the corresponding tree structure.



# Decision tree

- A hierarchical model for supervised learning
- Implementing the divide-and-conquer strategy
- An efficient nonparametric method
- For classification and regression
- Composed of internal decision nodes and terminal leaves.
- The tree can be converted to a set of IF-THEN rules that are easily understandable.

# Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute,  $x_i$ 
    - Numeric  $x_i$  : for binary split , check if  $x_i > w_m$
    - Discrete  $x_i$  :  $n$ -way split for  $n$  possible values
  - Multivariate: Uses all attributes,  $x$
- Leaves
  - Classification: Class labels
  - Regression: Numeric
- Learning is **greedy**; find the best split recursively.

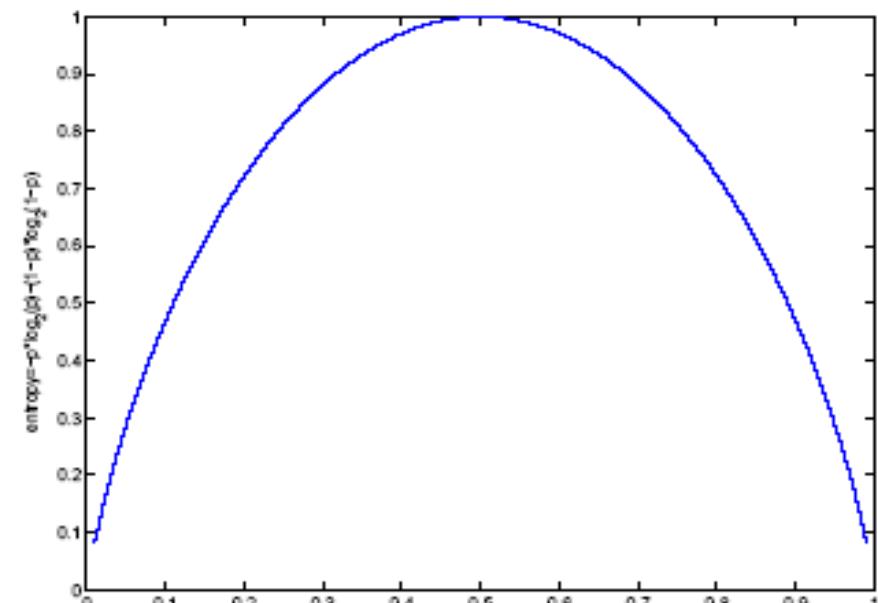
# Classification trees (ID3, CART, C4.5)

- The goodness of a split is quantified by impurity measure.
- For node  $m$ ,  $N_m$  instances reach the node  $m$ , and  $N_m^i$  instances belong to  $C_i$ . If an instance reaches  $m$ , the estimate of probability of class  $C_i$  is

$$\hat{P}(C_i | \mathbf{x}, m) \equiv p_m^i = \frac{N_m^i}{N_m}$$

- Node  $m$  is pure if  $p_m^i$  is 0 or 1
- Measure of **impurity** is **entropy**.

$$J_m = -\sum_{i=1}^K p_m^i \log_2 p_m^i$$



Entropy function for a two-class problem

# Best split

- If node  $m$  is pure, generate a leaf and stop, otherwise split and continue recursively.
- Impurity after split:  $N_{mj}$  of  $N_m$  take branch  $j$ .  $N_{mj}^i$  belong to  $C_i$

$$\hat{P}(C_i \mid \mathbf{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}} \quad I'_m = -\sum_{j=1}^n \frac{N_{mj}}{N_m} \sum_{i=1}^K p_{mj}^i \log_2 p_{mj}^i$$

- For all attributes, calculate the impurity and choose the one that has the minimum entropy. Tree construction continues recursively and in parallel for all branches that are not pure, until all are pure.

# Regression trees

- The goodness of a split is measured by the mean square error from the estimated value. Error at node  $m$ :

$$b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_m : \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

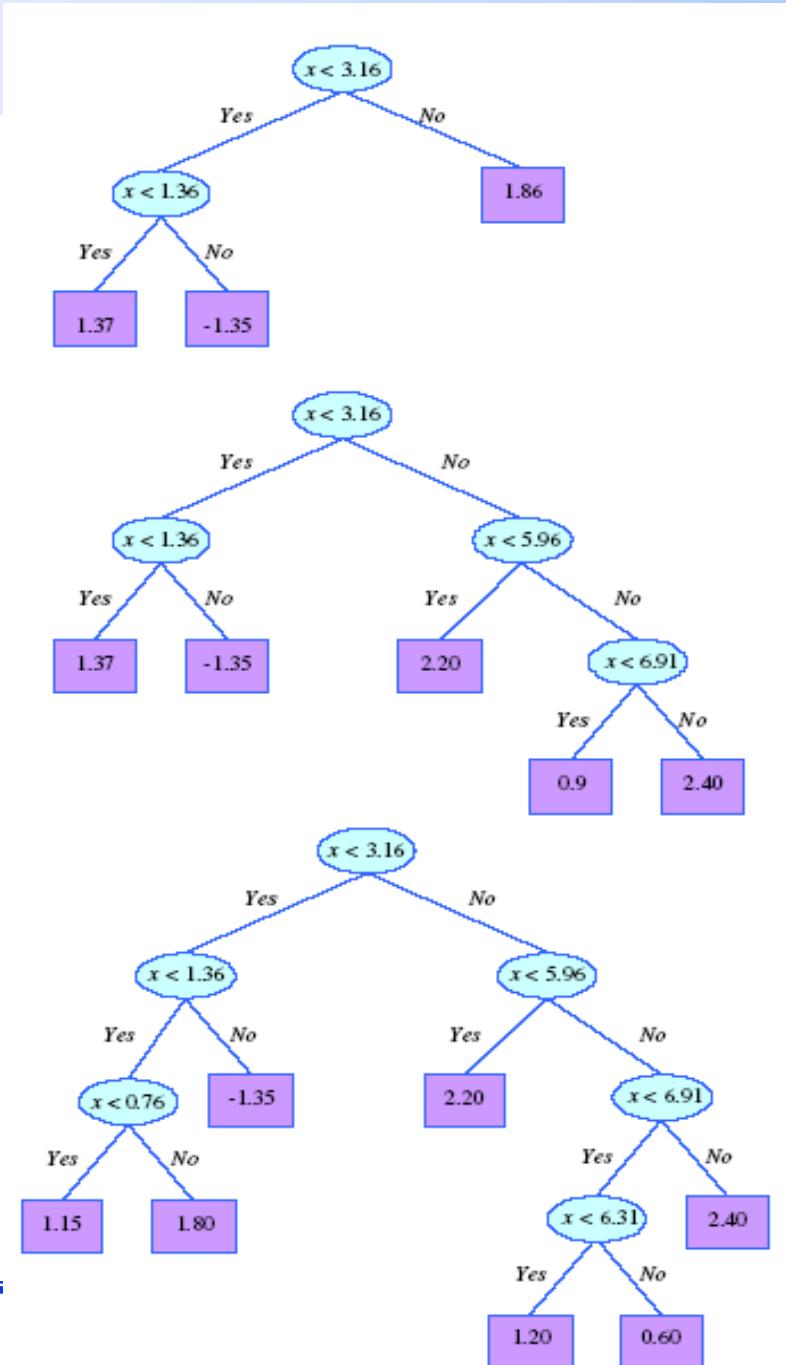
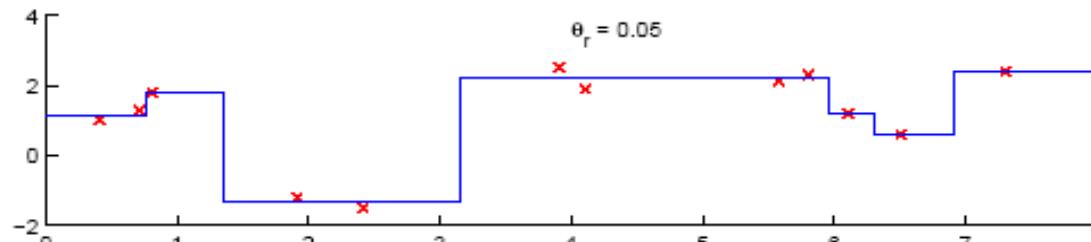
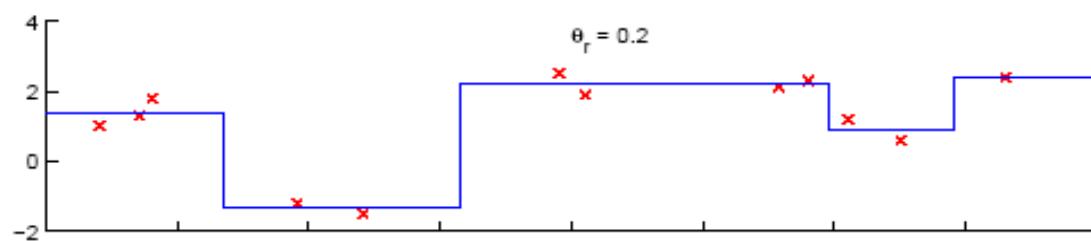
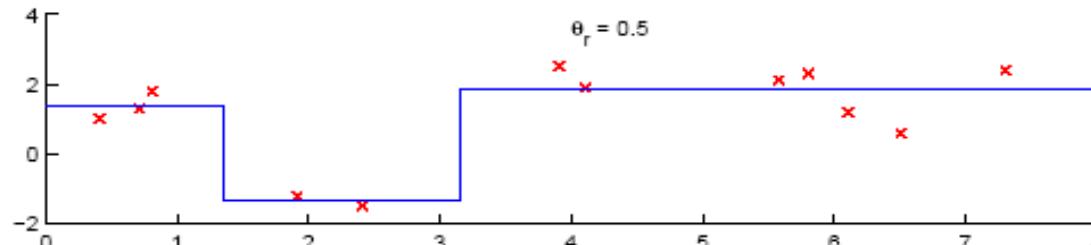
$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\mathbf{x}^t) \quad g_m = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{\sum_t b_m(\mathbf{x}^t)} = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{N_m}$$

- After splitting:

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_{mj} : \mathbf{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

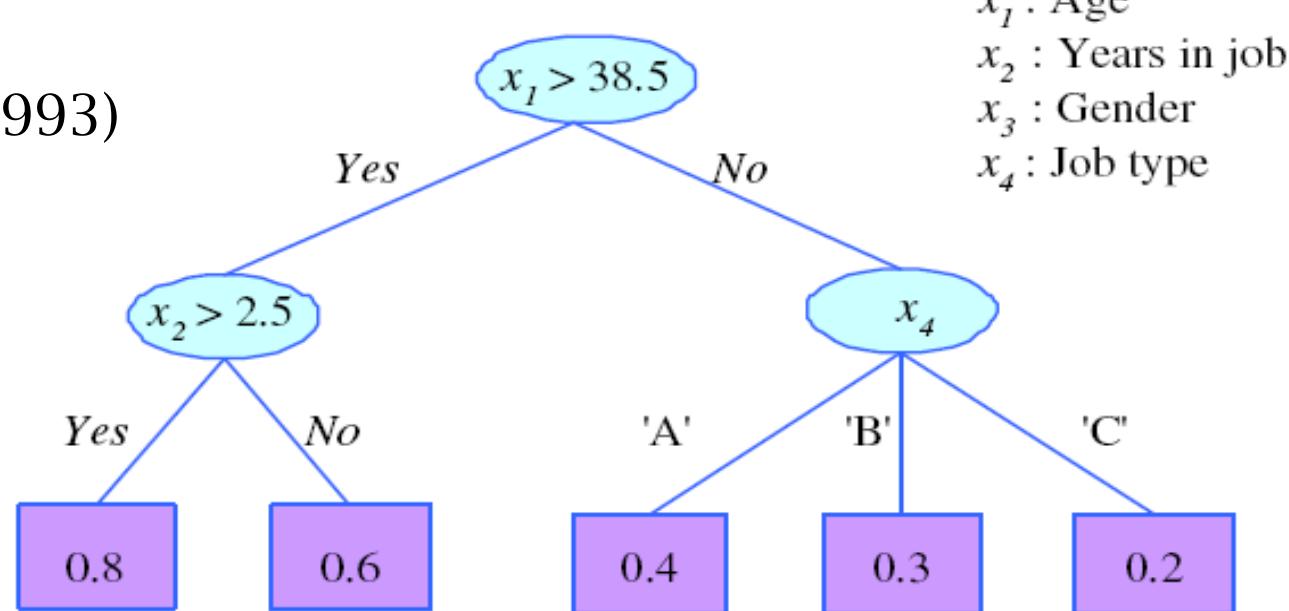
$$E'_m = \frac{1}{N_m} \sum_j \sum_t (r^t - g_{mj})^2 b_{mj}(\mathbf{x}^t) \quad g_{mj} = \frac{\sum_t b_{mj}(\mathbf{x}^t) r^t}{\sum_t b_{mj}(\mathbf{x}^t)}$$

## Model Selection in Trees:



# Rule Extraction from Trees

C4.5Rules  
(Quinlan, 1993)

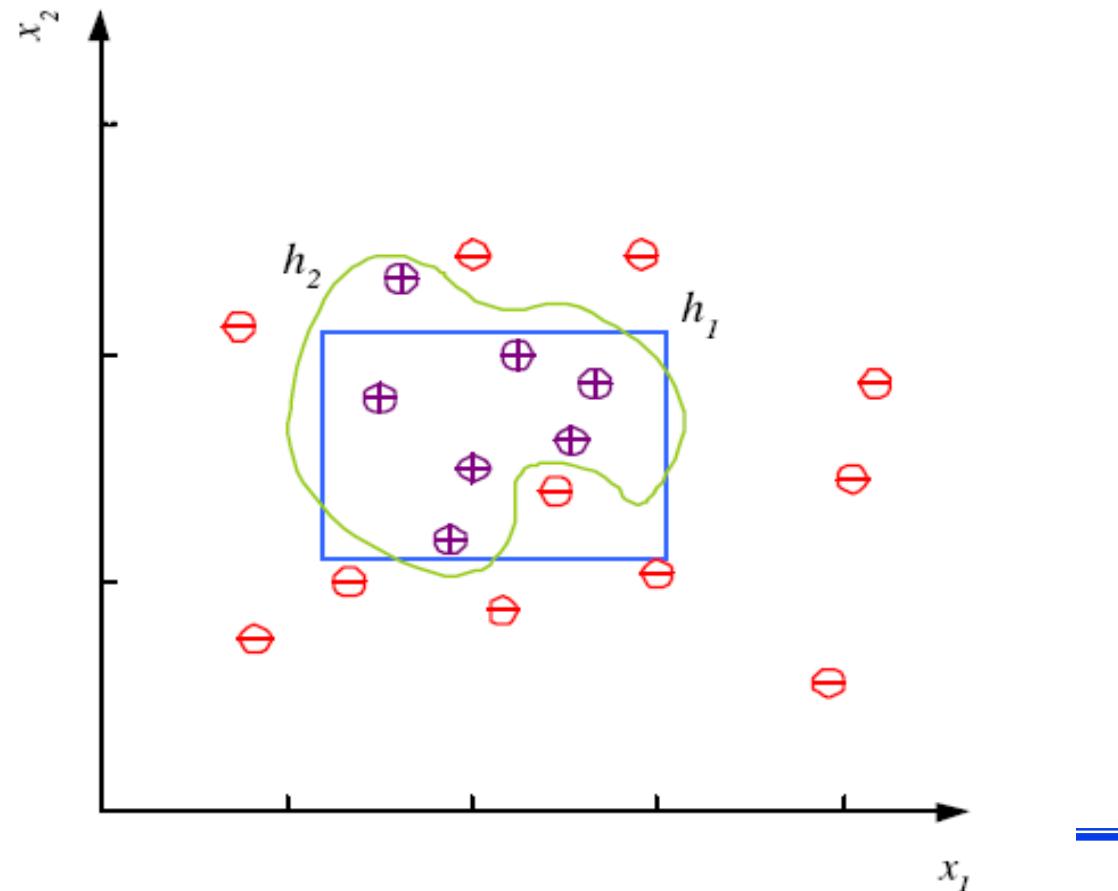


- R1: IF (age>38.5) AND (years-in-job>2.5) THEN  $y = 0.8$
- R2: IF (age>38.5) AND (years-in-job≤2.5) THEN  $y = 0.6$
- R3: IF (age≤38.5) AND (job-type='A') THEN  $y = 0.4$
- R4: IF (age≤38.5) AND (job-type='B') THEN  $y = 0.3$
- R5: IF (age≤38.5) AND (job-type='C') THEN  $y = 0.2$

# Noise and Model Complexity

Use the simpler one because

- Simpler to use  
(lower computational complexity)
- Easier to train (lower space complexity)
- Easier to explain  
(more interpretable)
- Generalizes better (lower variance - Occam's razor)



# Model Selection & Generalization

- Learning is an ill-posed problem, and data by itself is not sufficient to find the solution (i.e. the function underlying the data,  $f$ ).
- So we make assumptions to have a unique solution with the data we have. The set of assumptions is called *inductive bias* of the learning algorithm.
  - To introduce inductive bias, we assume a hypothesis class,  $\mathcal{H}$ , e.g. a linear function.
- Generalization: How well a learned model performs on new data
  - Overfitting:  $\mathcal{H}$  more complex than  $f$
  - Underfitting:  $\mathcal{H}$  less complex than  $f$

# Triple trade-off

There is a trade-off between three factors (Dietterich, 2003):

1. Complexity of  $\mathcal{H}$ ,  $c(\mathcal{H})$ ,
2. Training set size,  $N$ ,
3. Generalization error,  $E$ , on new data

- As  $N$  increases,  $E$  decreases.
- As  $c(\mathcal{H})$  increases, first  $E$  decreases and then  $E$  increases.

# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

# Machine Learning

## Lecture 13: Reinforcement Learning

Zheng-Hua Tan

Dept. of Electronic Systems, Aalborg Univ., Denmark

[zt@es.aau.dk](mailto:zt@es.aau.dk), <https://zhenghuatan.es.aau.dk/>

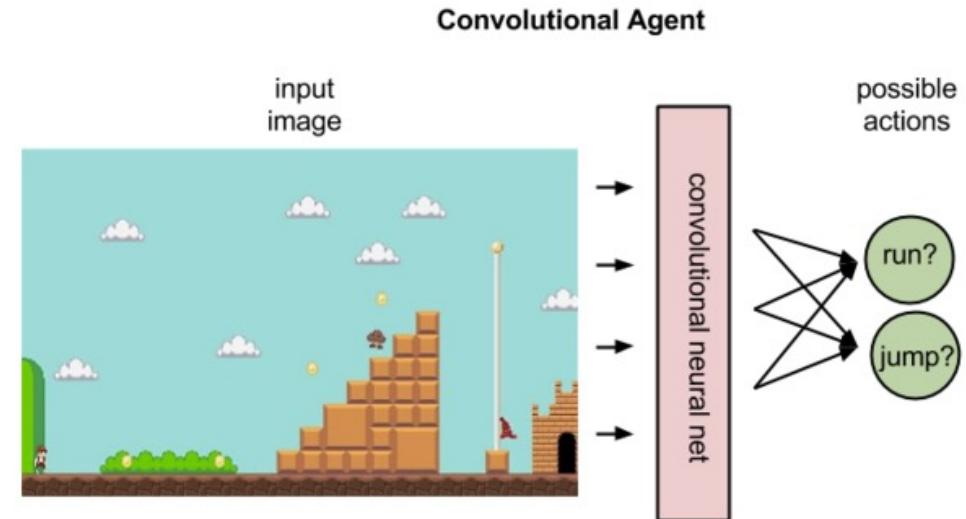
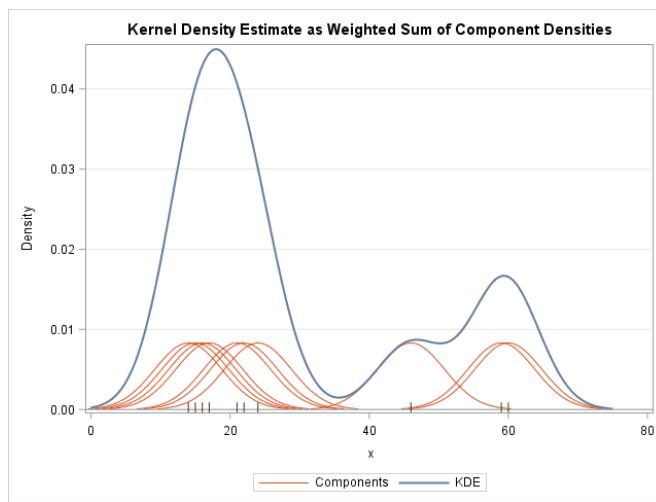
# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning

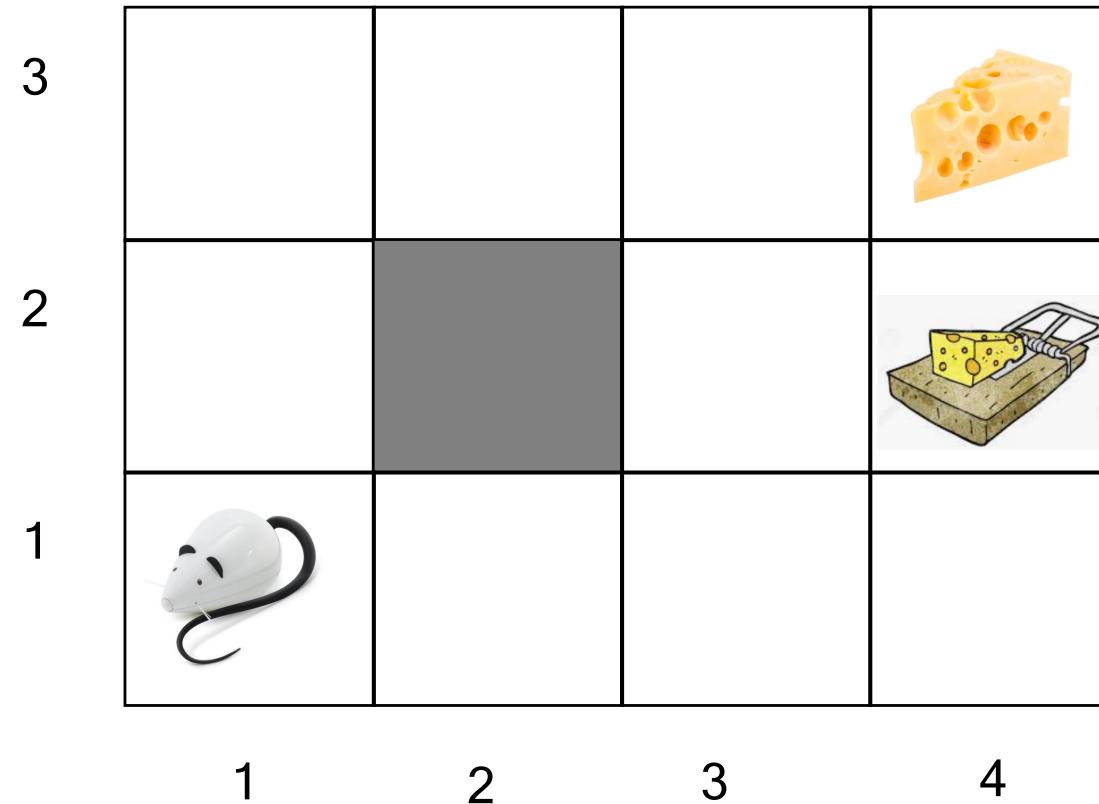
# Machine learning problems

- Machine learning: the development of algorithms that allow computers to learn from examples or experiences.
- Machine learning problems:

# Machine learning problems - cont.

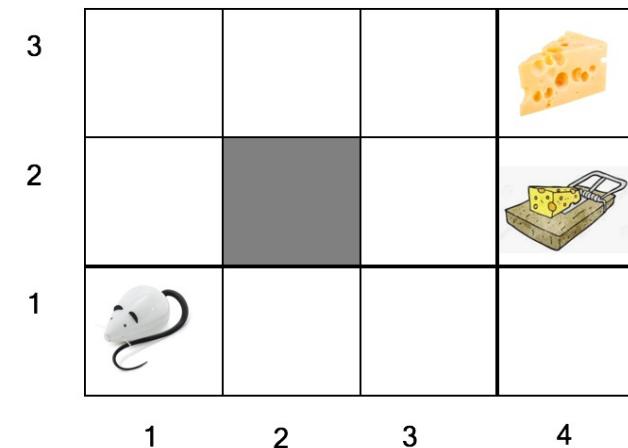
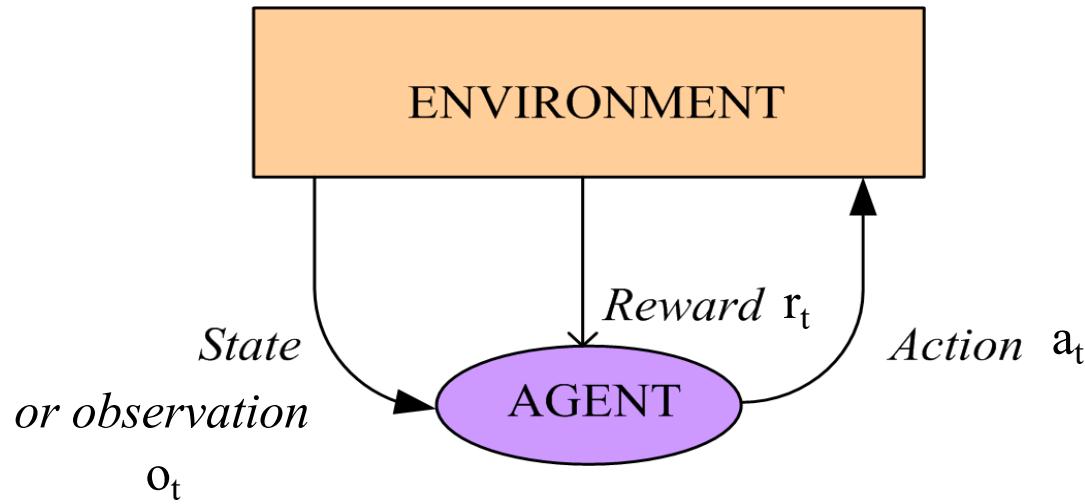


# Reach a goal / maximize award



# Reinforcement learning $\{a_t, o_t, r_t\}$

- Reinforcement learning: given inputs from the environment, take actions that affect the environment and produce action sequences that maximize the expected scalar reward/punishment. This is similar to animal learning.
- Optimal outputs are discovered by a process of *trial and error*. Examples: game-playing; grid world



# Reinforcement learning $\{a_t, o_t, r_t\}$ - cont.

- Concerned with how **agents** take **actions**  $a_t$  in an **environment**  $s_t$ , based on history and **observation**  $o_t$ , so as to maximize cumulative **reward**  $r_t$
- The environment is typically formulated as a **Markov decision process** (MDP).
- Two classes of methods: **dynamic programming** (DP) and **reinforcement learning** (RL)
- The main difference is that RL does not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible.
  - Need to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge)

# RL problems

- No hard-code knowledge about the environment and neither about the best actions to take in every specific situation
- To gain such knowledge is too expansive or infeasible, e.g. Go game:
  - The number of possible board positions for Go game is around  $10^{172}$
  - This number for chess is around  $2.0 \times 10^{52}$
  - The number of atoms in the universe is about  $2.4 \times 10^{78}$
- Knowledge may become useless even with a slight change to the environment, e.g. new games, exploration on Mars.

# Reinforcement learning

- Optimization
- Generalization
- Make a sequence of decisions
- Trial and error to obtain data
  - only get a reward (label) for decision made; no reward information is known if a different decision had been taken
  - decisions impact what the agent learns about
- Tradeoff between exploration and exploitation
- Delayed consequences / rewards
- Examples: play games, choose education and career with one example being your joining Aalborg University

# RL applications

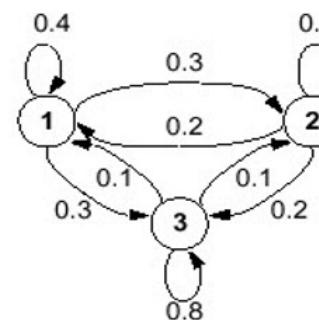
Examples:

- Chess, Go game, computer games
- Natural language processing (NLP),  
e.g. ChatGPT
- Neural architecture search (NAS)
- Control



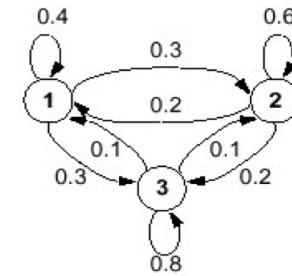
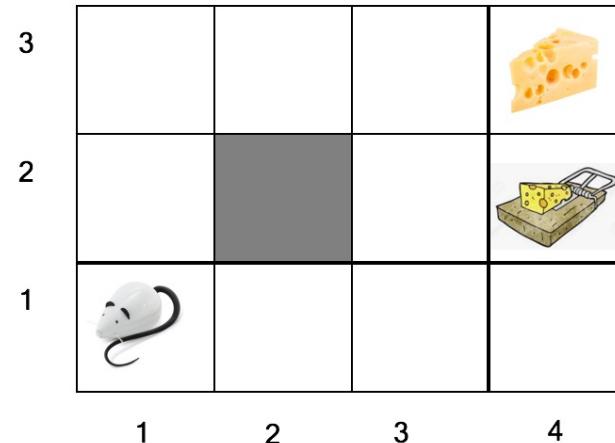
# Stochastic process

- Stochastic process: A random process that is defined as a collection of random variables that are indexed by a set of numbers (points in time)
- A Markov process is a stochastic process that satisfies the Markov property
  - Meaning that the probability of state  $s_{t+1}$  having any given value depends only upon state  $s_t$ :  $s_{t+1} = f(s_t)$
- A Markov chain is discrete-time stochastic process that satisfies the Markov property, e.g.,



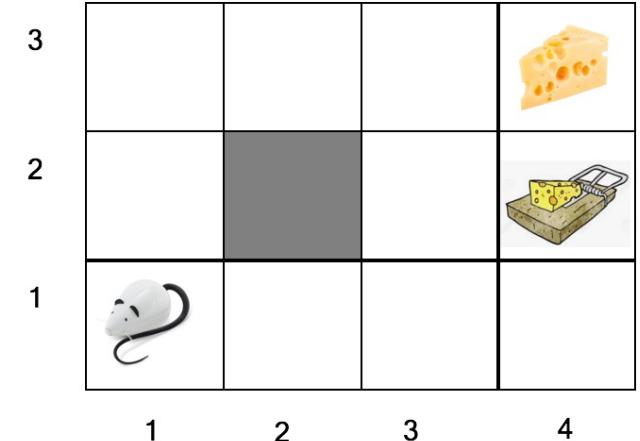
# Markov decision process

- A Markov decision process is an extension of Markov chains with the addition of
    - actions (of an agent, external to the process /environment) and
    - rewards
  - If the actions are fixed, an MDP reduces to a Markov chain



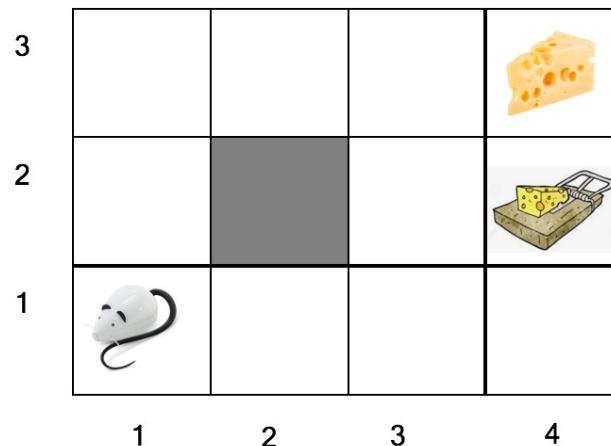
# Deterministic MDP

- A deterministic MDP is defined by
  - The state space of the process,  $S$ 
    - $s_t$ : State of agent at time  $t$
  - The action space of the agent,  $A$ 
    - $a_t$ : Action taken at time  $t$
  - The transition function of the process, describing how the state changes as a result of agent actions (deterministic)
    - $s_{t+1} = f(s_t, a_t)$
  - The reward function, evaluating the immediate action performance
    - Reward prob:  $p(r_{t+1} | s_t, a_t)$



# Stochastic MDP

- The next state is a random variable, whose probability density is given by the current state and action
- The deterministic transition function is replaced by a transition probability function
  - Next state prob:  $P(s_{t+1} | s_t, a_t)$  (vs. deterministic MDP  $s_{t+1} = f(s_t, a_t)$ ).



# Value functions

- Almost all RL algorithms involve estimating **value functions**
  - **Value functions of states**,  $V(s_t)$ , that estimate *how good* (i.e. expected return) it is for the agent to be in a given state
  - **Value functions of state-action pairs**,  $Q(s_t, a_t)$ , that estimate *how good* (i.e. expected return) it is to perform a given action in a given state.
  - Value functions are defined with respect to *particular ways of actions*, called **policies**
    - A policy is a mapping from states to probabilities of selecting each possible action:  $\pi(a | s)$  is the probability that  $A_t = a$  if  $S_t = s$ .

# Policy and cumulative reward

- A policy is a mapping from states to probabilities of selecting each possible action:  $\pi(a | s)$ .
- Value of a policy, the expected cumulative reward while following the policy:  $V^\pi(s_t)$
- In finite-horizon model, the agent maximizes the expected reward for the next T steps:

$$V^\pi(s_t) = E[r_{t+1} + r_{t+2} + \dots + r_{t+T}] = E\left[\sum_{i=1}^T r_{t+i}\right]$$

- In infinite horizon model, it maximizes the expected discounted returns:

$$V^\pi(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

$0 \leq \gamma < 1$  is the discount rate

# Policy and cumulative reward - cont.

Optimal policy  $\pi^*(s_t)$ : choose  $a_t^*$  that maximizes the state value  $V(s_t)$ :

$$V^*(s_t) = \max_{\pi} V^\pi(s_t), \forall s_t$$

$$= \max_{a_t} E \left[ \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} \right]$$

$$= \max_{a_t} E \left[ r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1} \right]$$

$$= \max_{a_t} E \left[ r_{t+1} + \gamma \frac{V^*(s_{t+1})}{E[V^*(S_{t+1})]} \right]$$

$$V^*(s_t) = \max_{a_t} \left( E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \frac{V^*(s_{t+1})}{E[V^*(S_{t+1})]} \right)$$

$$V^*(s_t) = \max_{a_t} \frac{Q^*(s_t, a_t)}{\text{Value of } a_t \text{ in } s_t}$$

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

where the value of state-action pair is

# Algorithms for solving MDPs

- Two classes:
  - Model-based algorithms: dynamic programming
  - Model-free algorithms: reinforcement learning
- Three subclasses of DP and RL algorithms
  - Value iteration: search for the optimal value function (maximal returns from every state or from every state-action pair)
  - Policy iteration: evaluate policies by constructing their value functions and use these value functions to find new, improved policies
  - Policy search: directly search for an optimal policy

# Elements of DP and RL

- Stochastic Markov decision processes (MDP)
  - State of agent at time  $t$ :  $s_t$
  - Action taken at time  $t$ :  $a_t$
  - The world / environment (model):
    - State transition probability:  $P(s_{t+1} | s_t, a_t)$
    - Reward probability:  $p(r_{t+1} | s_t, a_t)$
- In  $s_t$ , action  $a_t$  is taken, clock ticks and reward  $r_{t+1}$  is received and state changes to  $s_{t+1}$
- Initial state(s), goal state(s)
- Episode (trial): the sequence of actions from initial state to goal

HMM

State at time t,  $s_t$

State transition prob

$p(s_{t+1}|s_t)$

Observation prob:

$p(o_t|s_t)$

# DP and RL

- The key idea of DP, and RL in general, is to use value functions for searching good policies
- We can easily obtain optimal policies once we have found the optimal value functions (value of state under the optimal policy or value of taking action  $a$  under state  $s$  under the optimal policy), which satisfy the Bellman optimality equations

# Dynamic programming

- A collection of algorithms for computing optimal policies given a perfect model of the environment as an MDP
  - $P(s_{t+1} | s_t, a_t)$  and  $p(r_{t+1} | s_t, a_t)$  are completely known, and thus there is no need for any exploration and we can directly solve for the optimal value function and policy.

# Dynamic programming - cont.

- Model-based learning
- Environment / model,  $P(s_{t+1} | s_t, a_t)$ ,  $p(r_{t+1} | s_t, a_t)$ , is known
- There is no need for exploration
- Solve for Bellman's equation:

$$V^*(s_t) = \max_{a_t} \left( E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

- Optimal policy

$$\pi^*(s_t) = \arg \max_{a_t} \left( E[r_{t+1} | s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

# Value iteration

Initialize  $V(s)$  to arbitrary values

Repeat

For all  $s \in \mathcal{S}$

For all  $a \in \mathcal{A}$

$$Q(s, a) \leftarrow E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a)V(s')$$

$$V(s) \leftarrow \max_a Q(s, a)$$

Until  $V(s)$  converge

$$V^*(s_t) = \max_{a_t} \left( E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t) \quad \text{Value of } a_t \text{ in } s_t$$

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

# Dynamic programming

- Provides foundation for other methods
  - Providing an essential foundation for understanding RL methods
  - RL methods can be viewed as attempts to achieve much the same as DP, only with less computation and without assuming a perfect model of the environment
- This classical view is not so useful in practice since we rarely have a perfect environment model and DP involves a great computational expanse
- Not practical for large problems

# Temporal difference learning

- Environment,  $P(s_{t+1} | s_t, a_t)$ ,  $p(r_{t+1} | s_t, a_t)$ , is not known but stationary; **model-free learning**; more interesting.
- There is need for exploration to sample from  $P(s_{t+1} | s_t, a_t)$  and  $p(r_{t+1} | s_t, a_t)$
- Use the reward received in the next time step to update the value of current state (action).
- These algorithms are called **temporal difference** algorithms since we look at the difference between the current estimate of the value of a state (or a state-action pair) and the discounted value of the next state and the reward received

# Temporal difference learning - cont'd

- Temporal difference error

$$TD_{error} = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

where  $\gamma$  is the discount factor

- Update rule

$$V(S_t) \leftarrow V(S_t) + \alpha TD_{error}$$

where  $\alpha$  is the learning rate

# Exploration strategies

- Epsilon-greedy (exploit, but once in a while do something random):
  - choose one action at random uniformly with probability  $\epsilon/(N-1)$ , i.e., explore;
  - choose the best action with probability  $1-\epsilon$ , i.e., exploit.
  - start with large  $\epsilon$  and gradually decrease it.
- Boltzmann exploration (choose probabilistically relative to expected rewards) - use softmax to convert values to probabilities; move smoothly from exploration to exploitation: start with large  $T$  and decrease it gradually (annealing); large  $T$  - exploration, small  $T$  - exploitation. The probability of choosing action  $a$  as

$$P(a | s) = \frac{\exp Q(s, a)}{\sum_{b=1}^{\mathcal{A}} \exp Q(s, b)}$$

$$P(a | s) = \frac{\exp[Q(s, a)/T]}{\sum_{b=1}^{\mathcal{A}} \exp[Q(s, b)/T]}$$

# Deterministic rewards and actions

- Bellman's equation for value of state-action pair

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

- Deterministic: single possible reward and next state, so

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

which is used as an update rule, i.e.

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

# Nondeterministic rewards and actions

- When next states and rewards are nondeterministic (there is an opponent or randomness in the environment), we keep averages (expected values) instead as assignments
- Bellman's equation for value of state-action pair

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} | s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

- We cannot do a direct assignment here as for same  $s_t$  and  $a_t$ , we have nondeterministic rewards and actions. So, we keep a running average, which is known as Q-learning:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta \left( r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t) \right)$$

# Q-learning

Initialize all  $Q(s, a)$  arbitrarily

For all episodes

    Initialize  $s$

    Repeat

        Choose  $a$  using policy derived from  $Q$ , e.g.,  $\epsilon$ -greedy

        Take action  $a$ , observe  $r$  and  $s'$

        Update  $Q(s, a)$ :

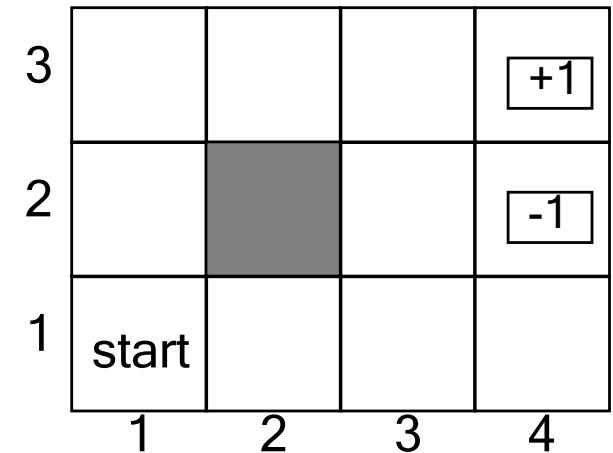
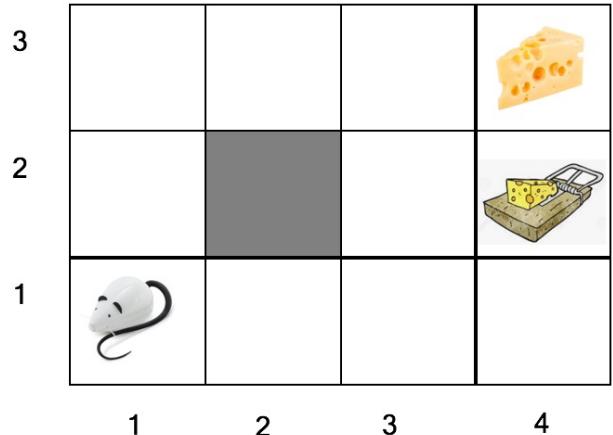
$$Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

$s \leftarrow s'$

    Until  $s$  is terminal state

# Grid world

- States: 4x3 – 1 possible states
- Actions: UP, DOWN, LEFT, RIGHT
- Transitions from state to state
  - Deterministic transitions
  - Stochastic transitions
- Reward prob:  $p(r_{t+1} | s_t, a_t)$ 
  - Reward +1 at [4,3], -1 at [4,2]
  - Reward -0.04 for each step
- What's the strategy to achieve max reward?

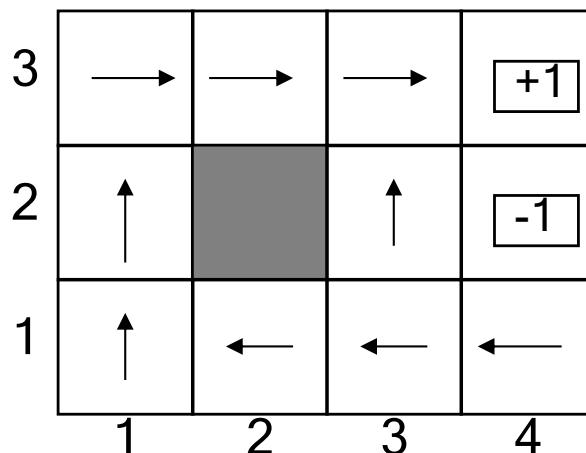


# Grid world - cont.

$M = 0.8$  in direction you want to go;  $0.2$  in perpendicular

0.1 left  
0.1 right

An optimal policy for the stochastic environment is as follows (to avoid [-1]):



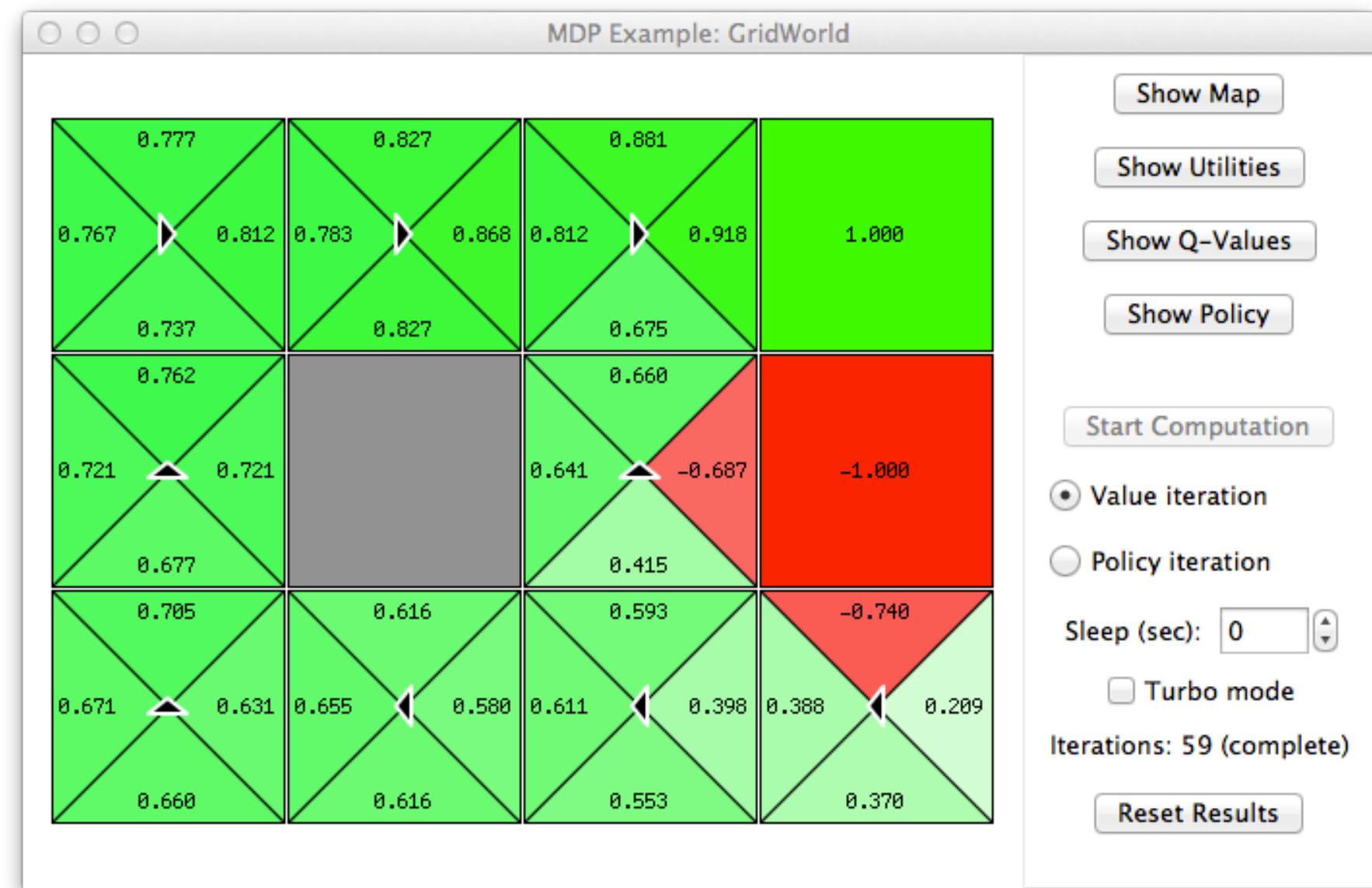
Utilities of states:

3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388

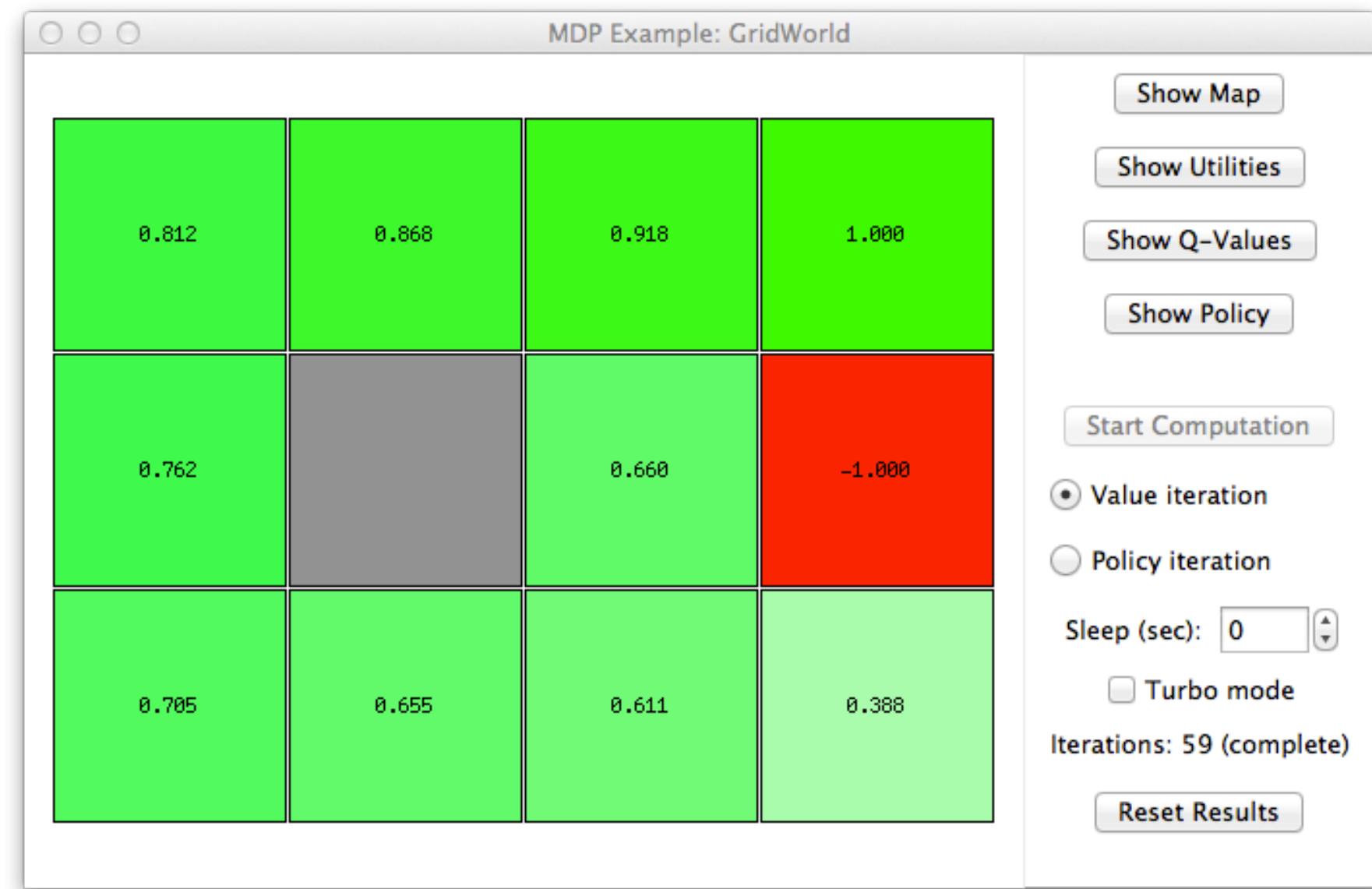
known utilities

Calculated in appr. values, that match the policy shown in the left figure

# Q values (of actions)



# Utilities (of states)



# Course Outline

1. Introduction
2. Bayesian decision theory (incl. probability theory)
3. Parametric and nonparametric methods
4. Dimensionality reduction
5. Clustering
6. Linear discrimination
7. Support vector machines
8. Multilayer perceptrons
9. Deep learning
10. Time series models
11. Graphical models
12. Algorithm-independent machine learning
13. Reinforcement learning