

Data Science Certification

Module 4 - Machine Learning

Veerasak Kritsanapraphan - IMC Institution



Introduction

Data Science Certification

Definitions of Machine Learning

Machine learning is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

A computer program that can learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, so that its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

-Mitchell

Traditional Programming



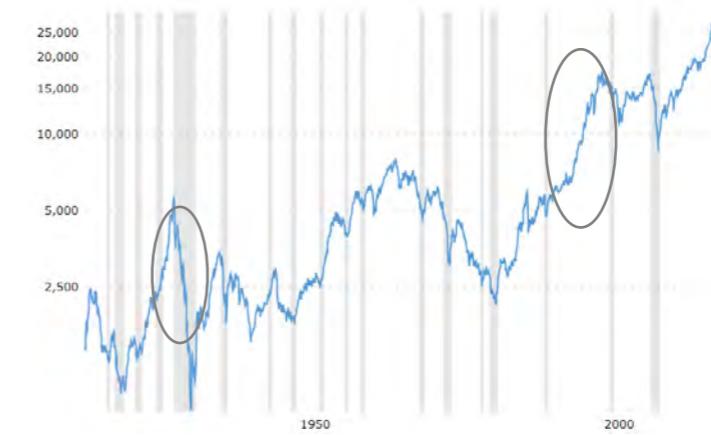
Machine Learning



Why Machine Learning

- Big problems
 - ~100 million songs
 - ~130 million books
 - ~644 million websites
- Open ended problems
 - 8.6k tweets per second
 - 2.4 billion active Facebook users
 - ~66k new web pages per day

Time changing problems

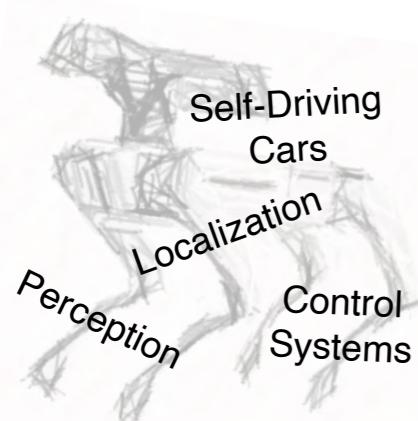


Intrinsically hard problems
Human Level Perception
First speech recognition 1952
Complex Open-ended games
First chess program 1957
Deep Blue 1997 beat Kasparov

Successes of Machine Learning



Web Search



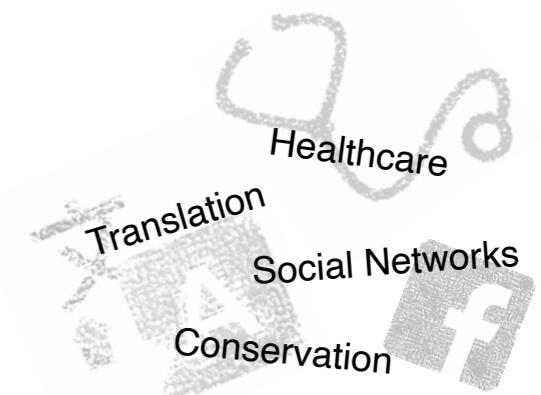
Robotics



Digital Assistants



Games



Many Others

Types of Machine Learning Algorithms

Thousands of machine learning algorithms, hundreds new every year

Supervised (inductive) learning

Training data includes desired outputs

Unsupervised learning

Training data does not include desired outputs

Semi-supervised learning

Training data includes a few desired outputs

Reinforcement learning

Rewards from sequence of actions

Machine Learning

Tens of thousands of machine learning algorithms

Hundreds new every year

Decades of ML research oversimplified:

All of Machine Learning:

Learn a mapping from input to output $f: X \rightarrow Y$

X : emails, Y : {spam, notspam}

Machine Learning

Input: x (images, text, emails...)

Output: y (spam or non-spam...)

(Unknown) Target Function

$f: X \rightarrow Y$ (the “true” mapping / reality)

Data

$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$

Machine Learning

- Every machine learning algorithm has three components:
 - Representation / Model Class
 - Evaluation / Objective Function
 - Optimization

Making Working Systems

Traditional Software

- Programming Languages
- Algorithms
- Data Structures
- Networking
- Etc.

- Statistics
- Data Science
- Machine Learning
- Computer Vision
- Natural Language Processing

Software Engineering

Machine Learning Engineering

Summary

Reasons for ML

Big problems

Open-ended problems

Time-changing problems

Hard problems

Types of Learning

Supervised, unsupervised, semi-supervised, reinforcement learning

- Supervised Learning
 - Training data $\langle x, f(x) \rangle$
 - Goal to learn $f(x)$ that matches training data
- Components of ML algorithm
 - Model Structure
 - Loss Function
 - Optimization

Exciting time for ML – many challenges & many ways to get involved



Asking Question in Machine Learning

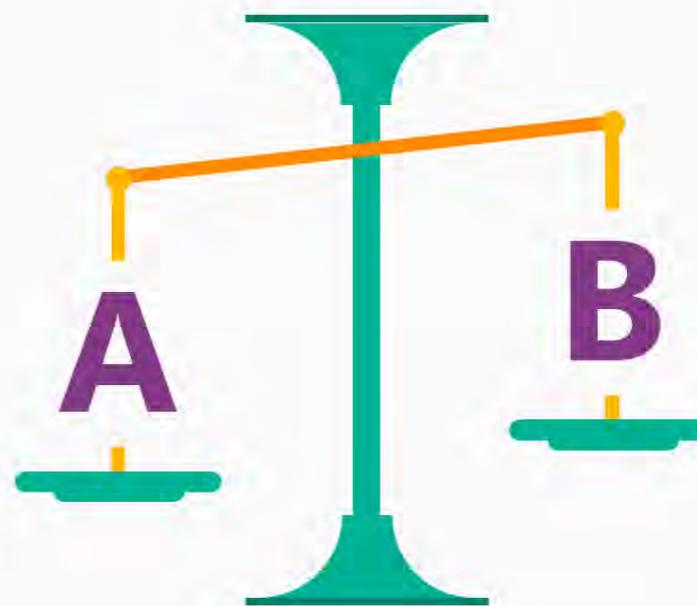
Fundamental Data Science for Data Scientist

What **questions** can data science **answer**?

1. Is this **A** or **B**?
2. Is this **weird**?
3. How **much** or how **many**?
4. How is this **organized**?
5. What should I **do next**?

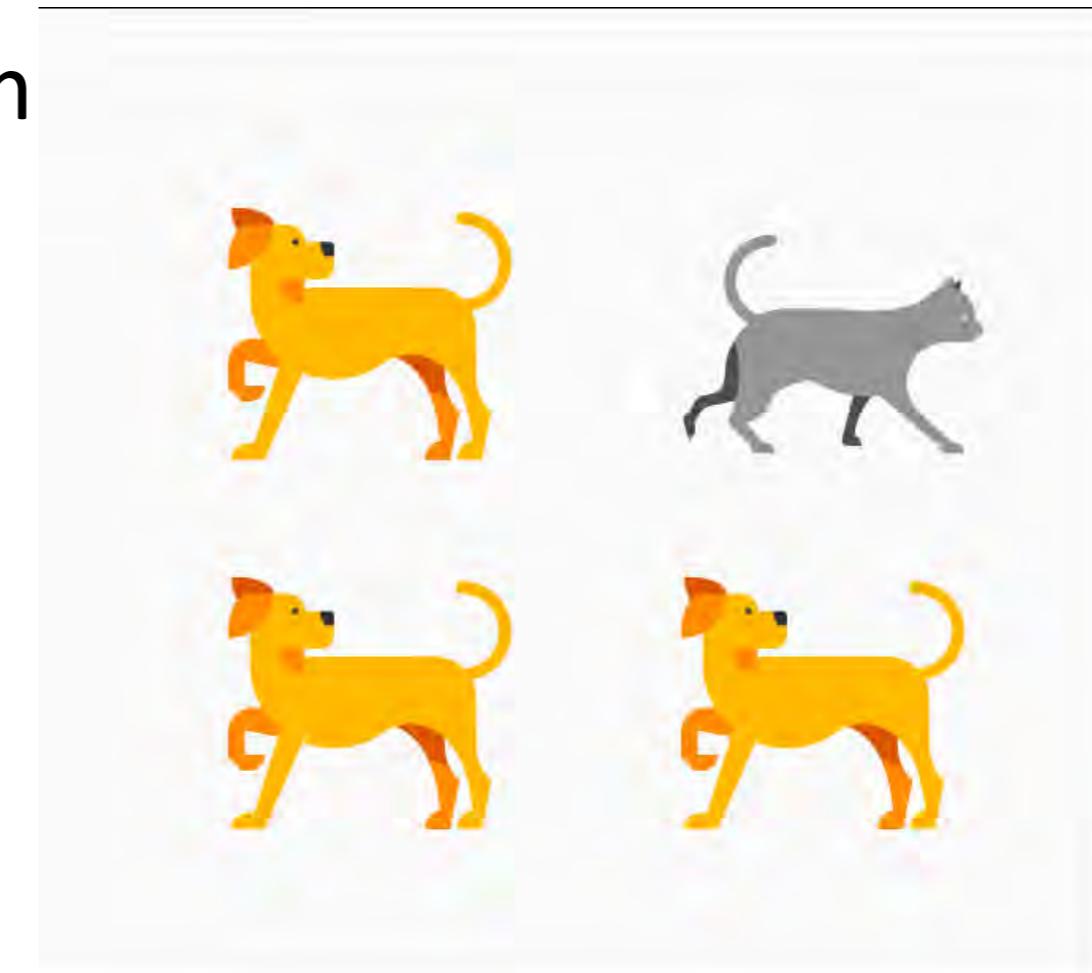
Is this A or B?

Classification algorithm



Is this weird?

Anomaly detection algorithm



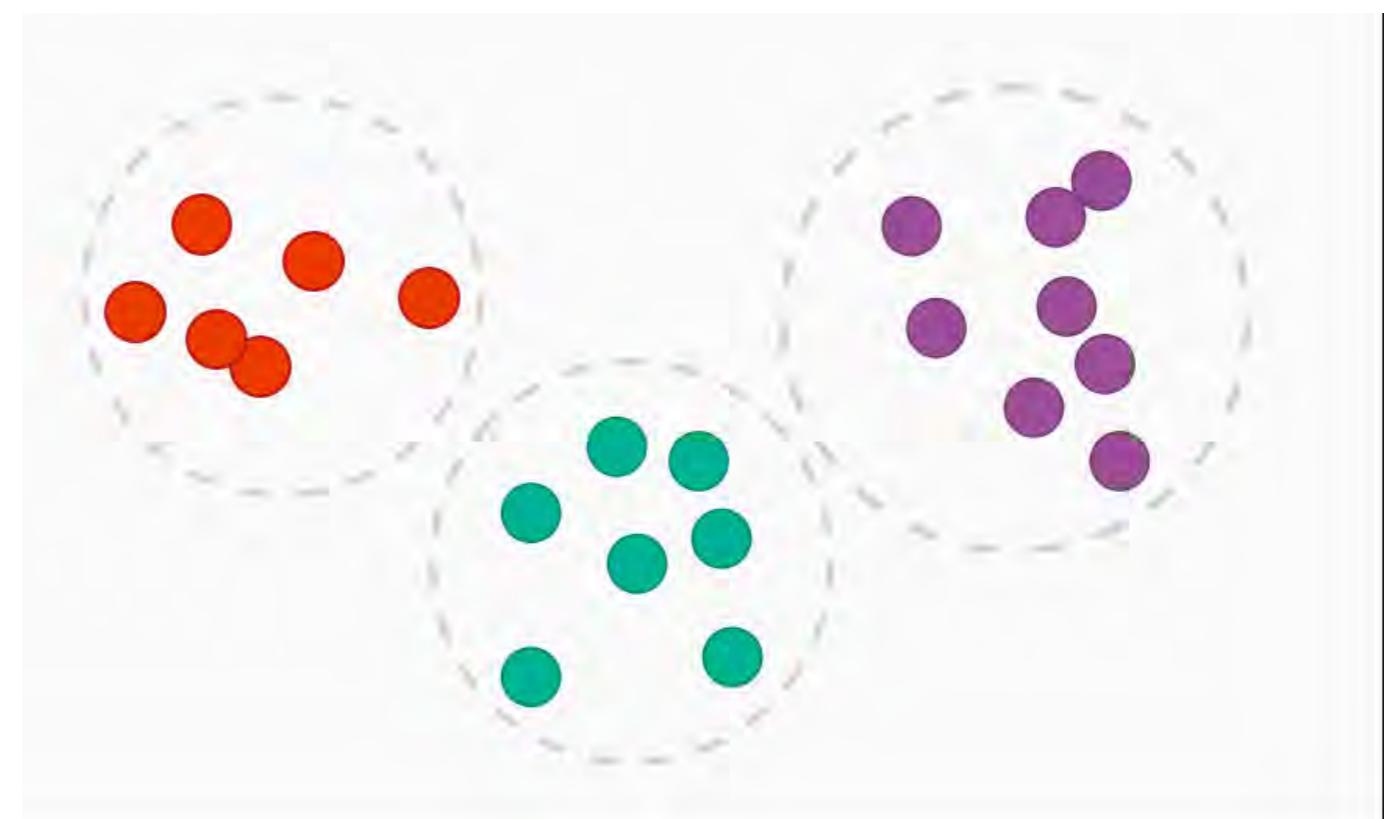
How much and How many?

Regression algorithm



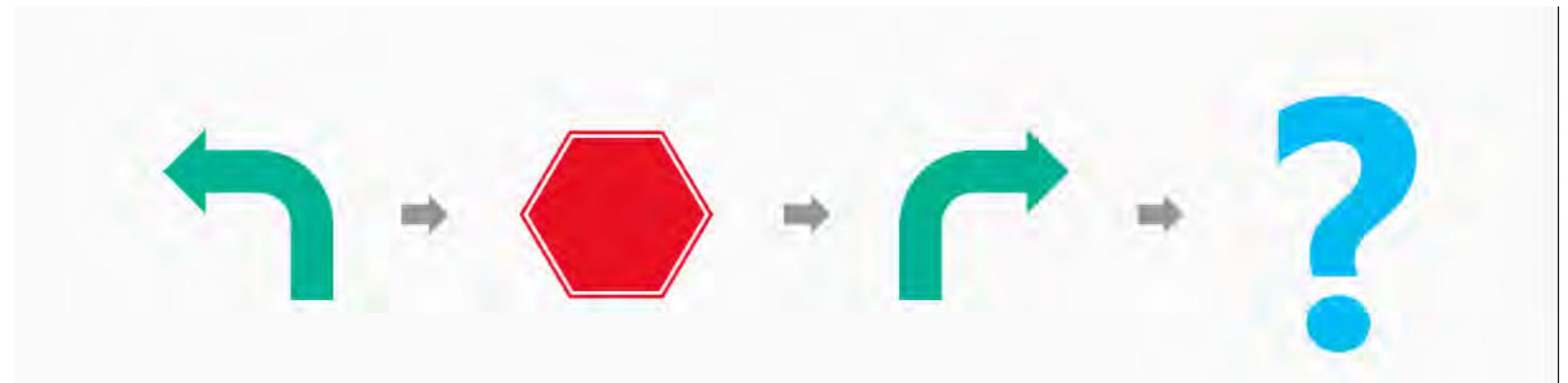
How is this organized?

Clustering algorithm

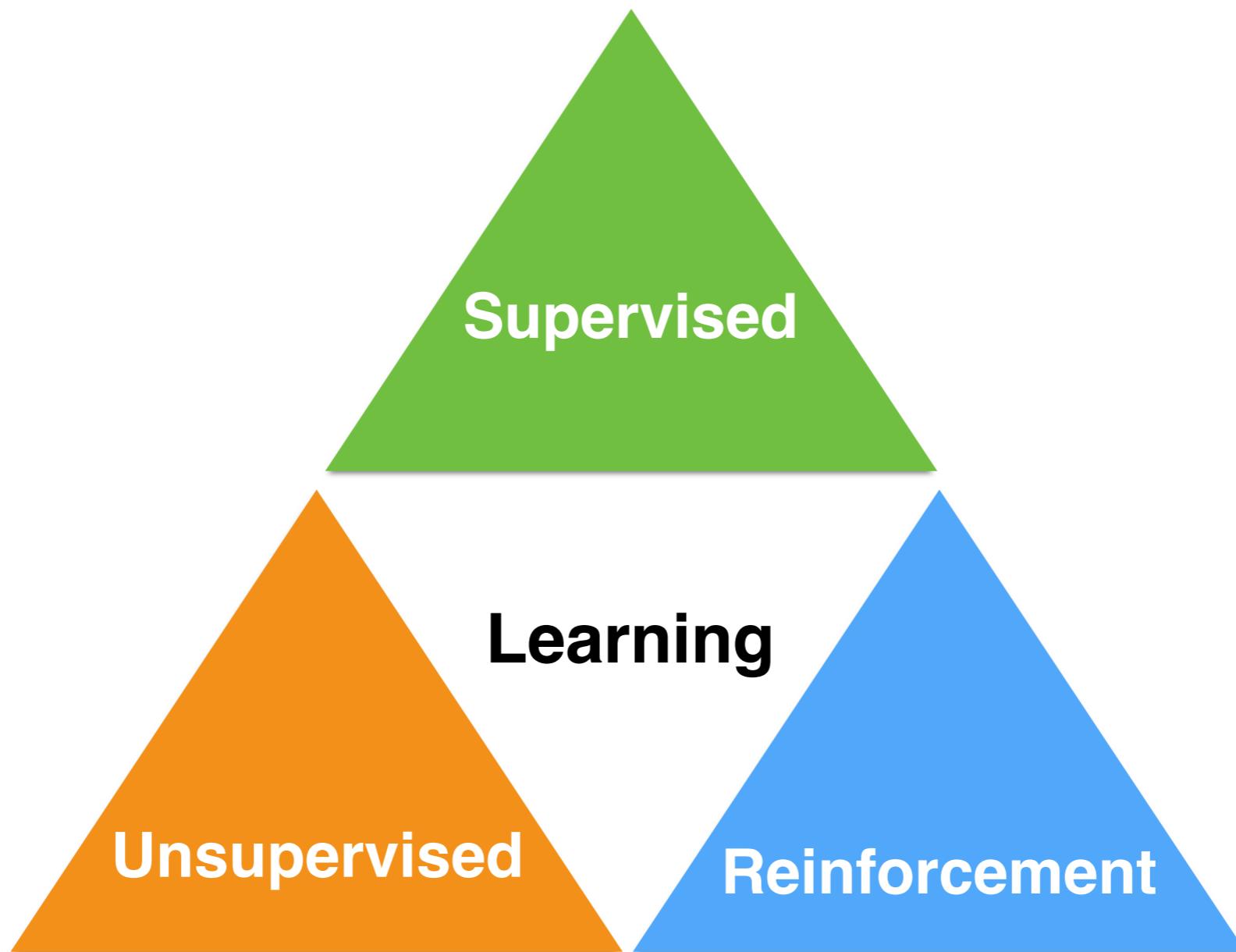


What should I do next?

Reinforcement Learning algorithm



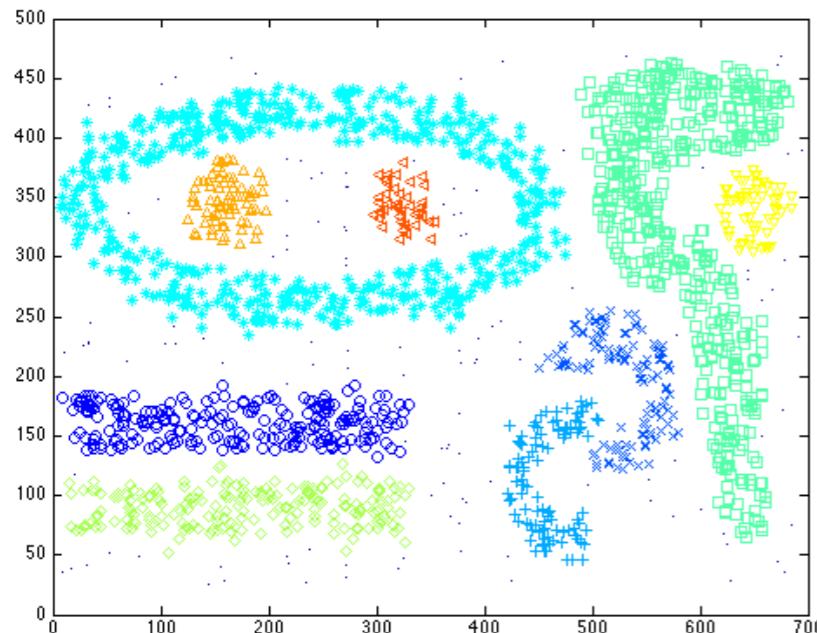
- Labeled data
- Direct feedback
- Predict outcome/future



- No labels
- No feedback
- “Find hidden structure”

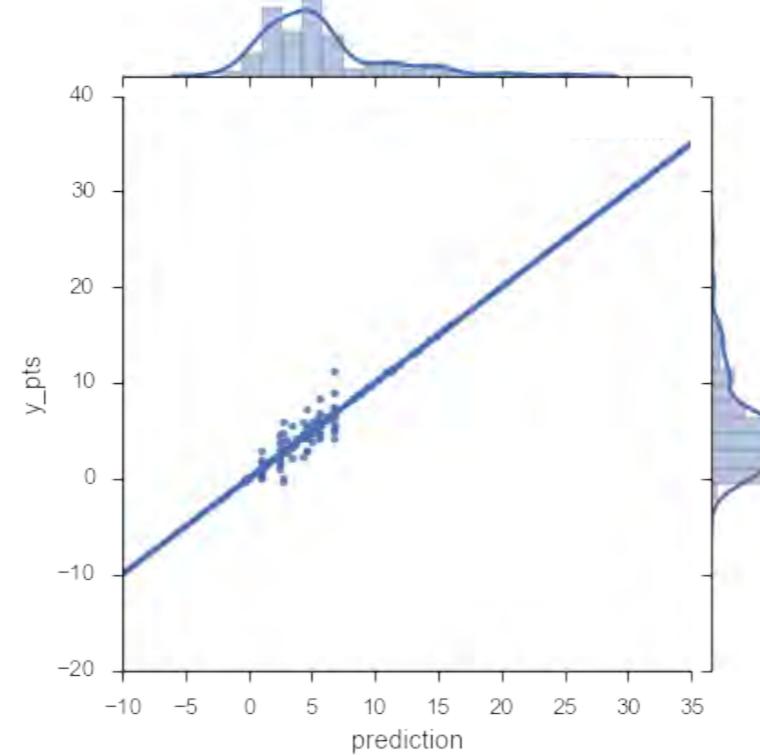
- Decision process
- Reward system
- Learn series of actions

Unsupervised Learning

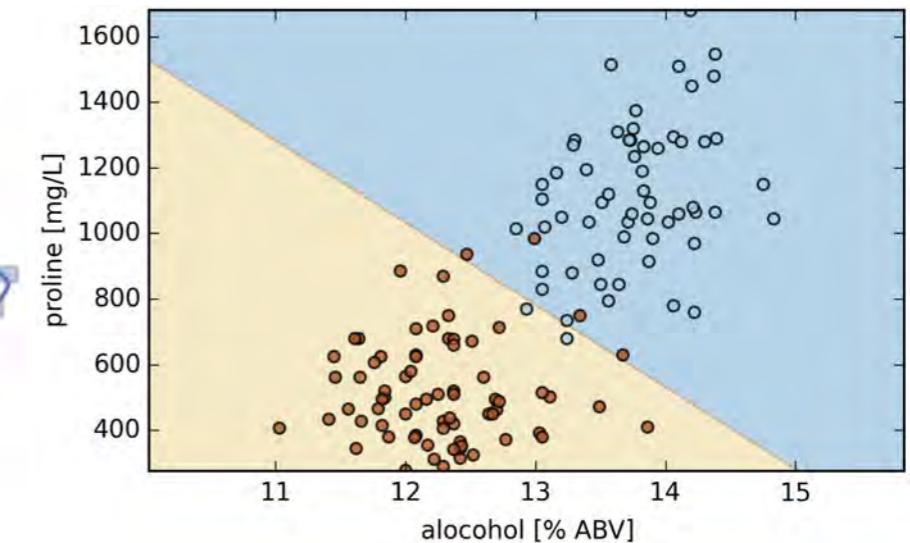


Clustering:
[DBSCAN on a toy dataset]

Supervised Learning



Regression:
[Soccer Fantasy Score prediction]



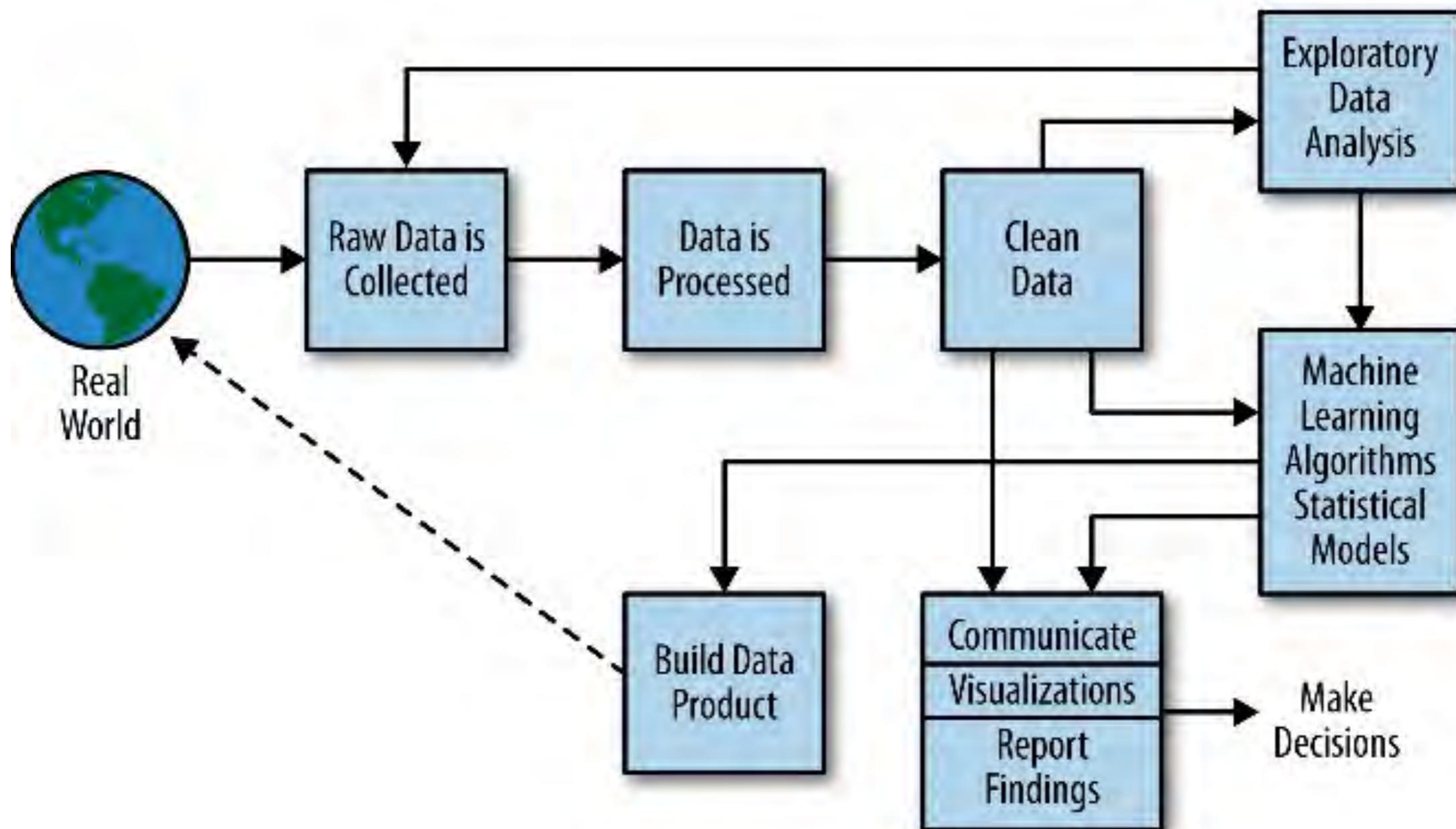
Classification:
[SVM on 2 classes of the Wine dataset]



Data Science Life Cycle

Fundamental Data Science for Data Scientist

Data Science Process





Data and Getting Data

Data Science Certification

Recap import data in R

```
# first row contains variable names, comma is separator  
# assign the variable id to row names  
# note the / instead of \ on mswindows systems
```

```
mydata <- read.table("c:/mydata.csv", header=TRUE,  
                      sep=",", row.names="id")
```

Data Frame

Function

Field Separation

Keep first line as a header

Filename to import

Getting Data

Iris Data Set from UCI Machine Learning Repository (<https://archive.ics.uci.edu/ml/datasets/Iris>)

Attribute Information:

1. Sepal Length in cm

2. Sepal width in cm

3. Petal length in cm

4. Petal length in cm

5. Classes:

- Iris Setosa

- Iris Versicolour

- Iris Virginica



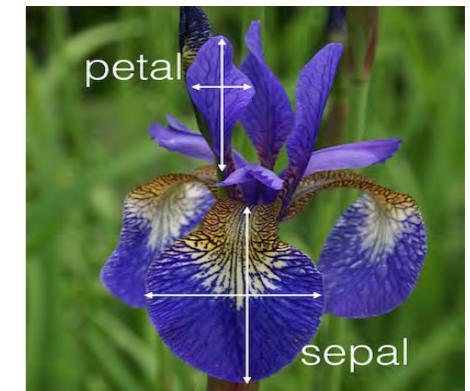
Nomenclature

IRIS

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

Instances (samples, observations)

	sepal_length	sepal_width	petal_length	petal_width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
...
50	6.4	3.2	4.5	1.5	veriscolor
...
150	5.9	3.0	5.1	1.8	virginica



Features (attributes, dimensions, variables)

Classes (targets)

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1D} \\ x_{21} & x_{22} & \cdots & x_{2D} \\ x_{31} & x_{32} & \cdots & x_{3D} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{ND} \end{bmatrix}$$

$$\mathbf{y} = [y_1, y_2, y_3, \dots, y_N]$$

Iris setosa



Iris virginica



Iris versicolor



Getting Data

```
> iris <- read.csv("iris.data.csv", header=TRUE)

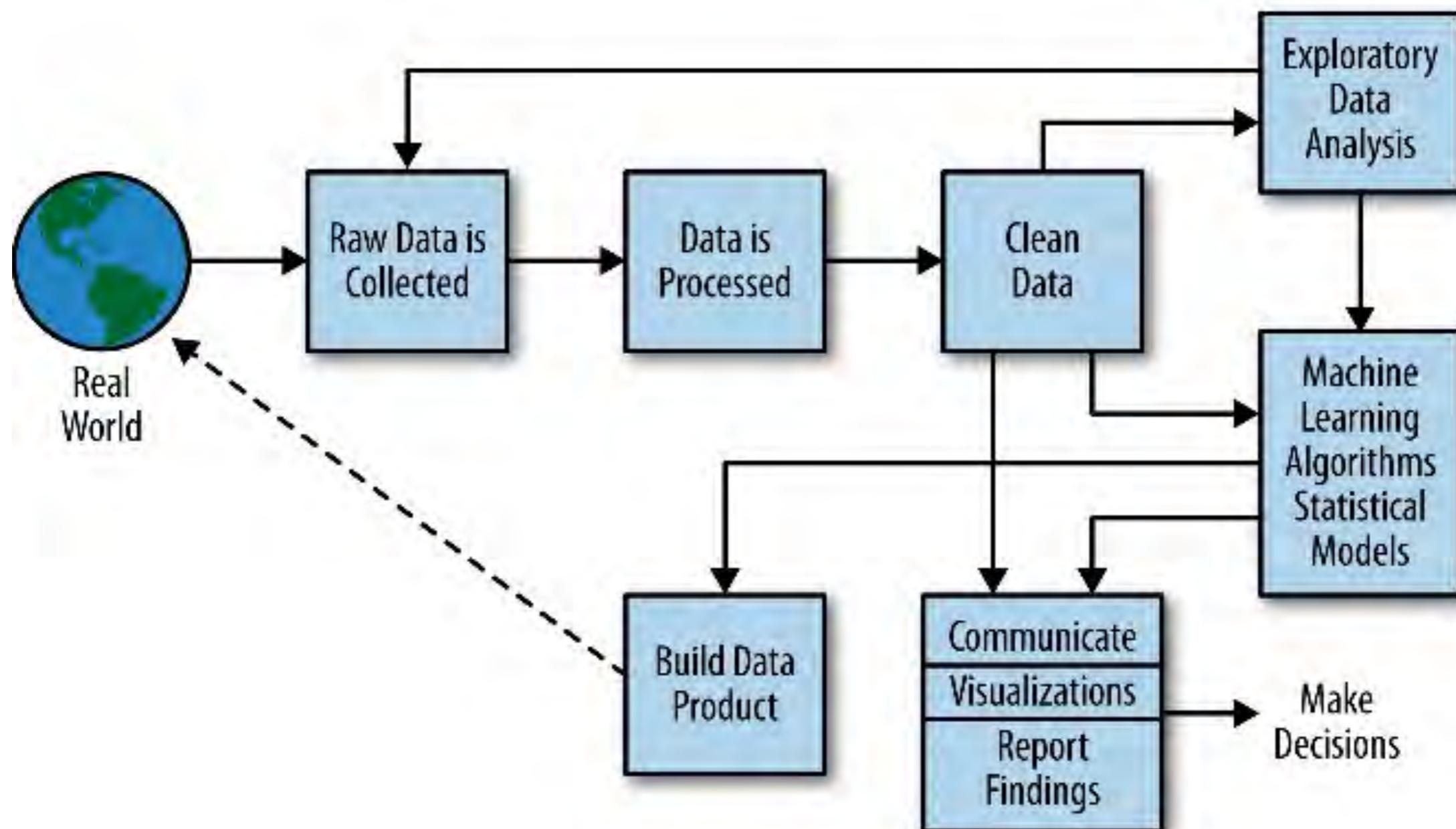
> library(datasets)

> iris

> colnames(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length",
  "Petal.Width", "Species")
```

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> nrow(iris)
[1] 150
> table(iris$Species)

setosa versicolor virginica
      50          50          50
>
```



Exploratory Data Analysis (EDA)

- The goal during EDA is to develop an understanding of your data.
- The easiest way to do this is to use questions as tools to guide your investigation.
- When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.
- Two types of questions will always be useful for making discoveries within your data.
 - 1.What type of variation occurs within my variables?
 - 2.What type of covariation occurs between my variables?

Variation

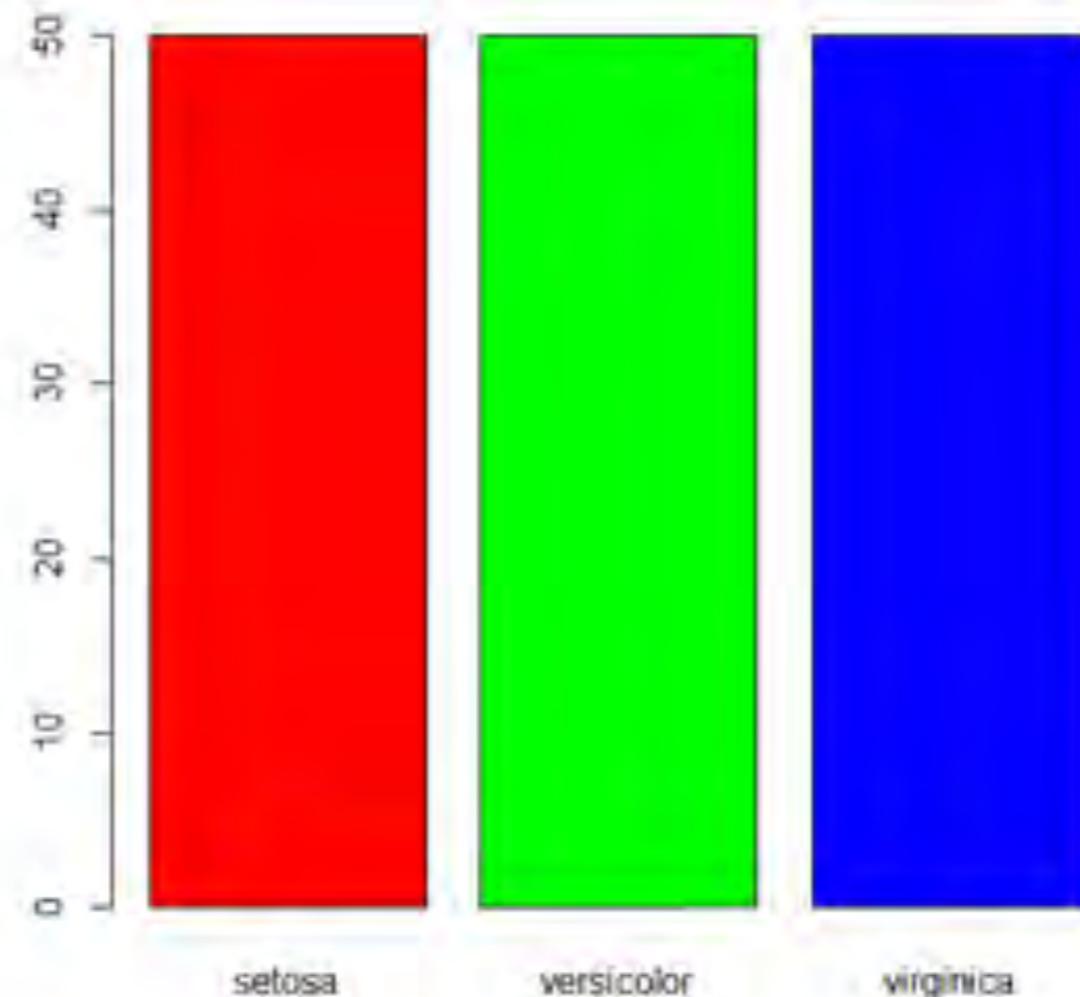
- The tendency of the values of a variable to change from measurement to measurement. You can see variation easily in real life.
- If you measure any continuous variable twice, you will get two different results. This is true even if you measure quantities that are constant, like the speed of light. Categorical variables can also vary if you measure across different subjects or different times.
- Every variable has its own pattern of variation, which can reveal interesting information. The best way to understand that pattern is to visualise the distribution of the variable's values.

Visualizing distributions

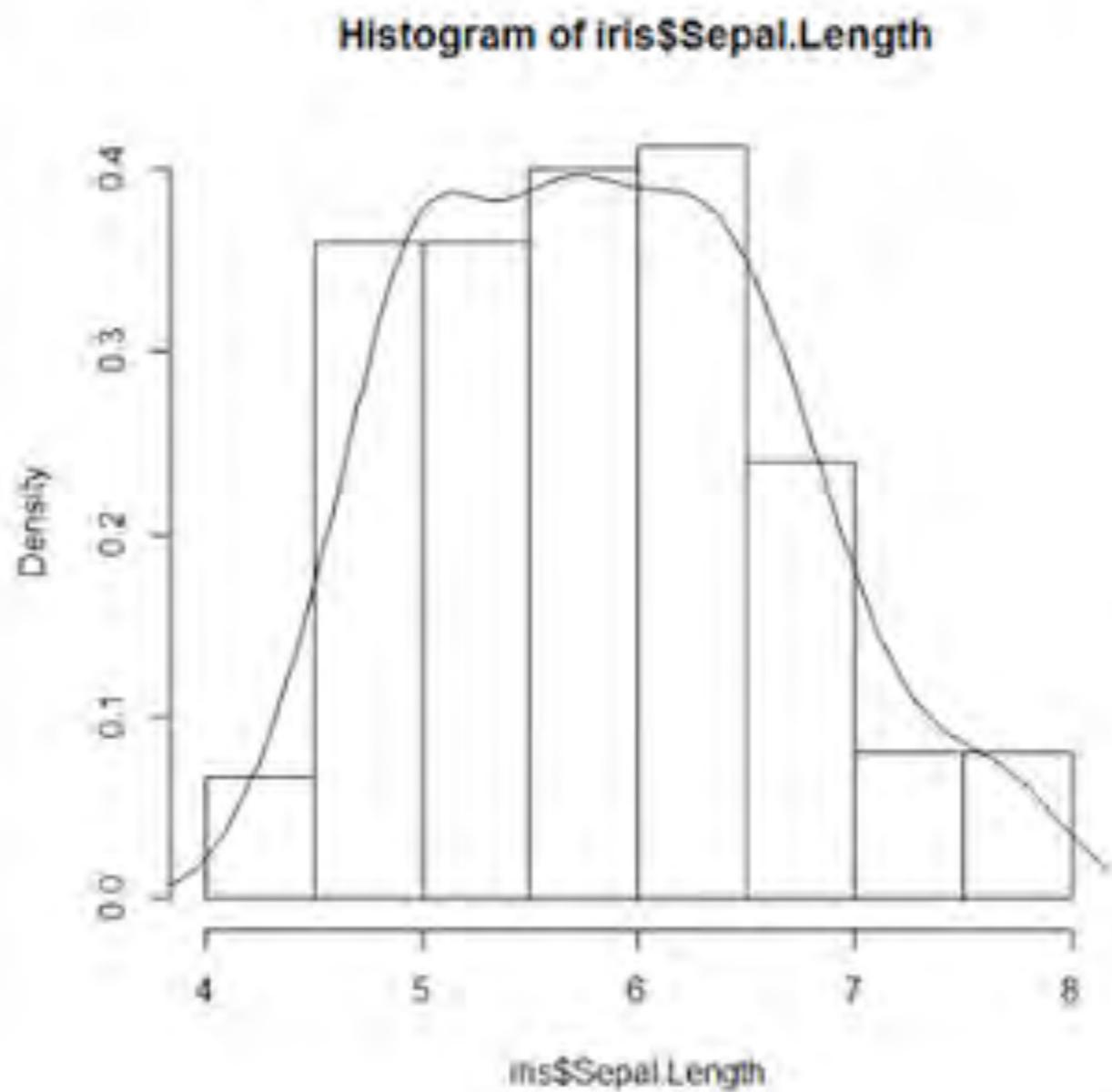
How you visualize the distribution of a variable will depend on whether the variable is categorical or continuous.

- A variable is **categorical** if it can only take one of a small set of values. In R, **categorical variables** are usually saved as factors or character vectors. To examine the distribution of a categorical variable, **use a bar chart**.
- A variable is **continuous** if it can take any of an infinite set of ordered values. Numbers and date-times are two examples of continuous variables. To examine the distribution of a **continuous** variable, **use a histogram**:

```
> categories <- table(iris$Species)
> barplot(categories, col=c('red', 'green', 'blue'))
>
```

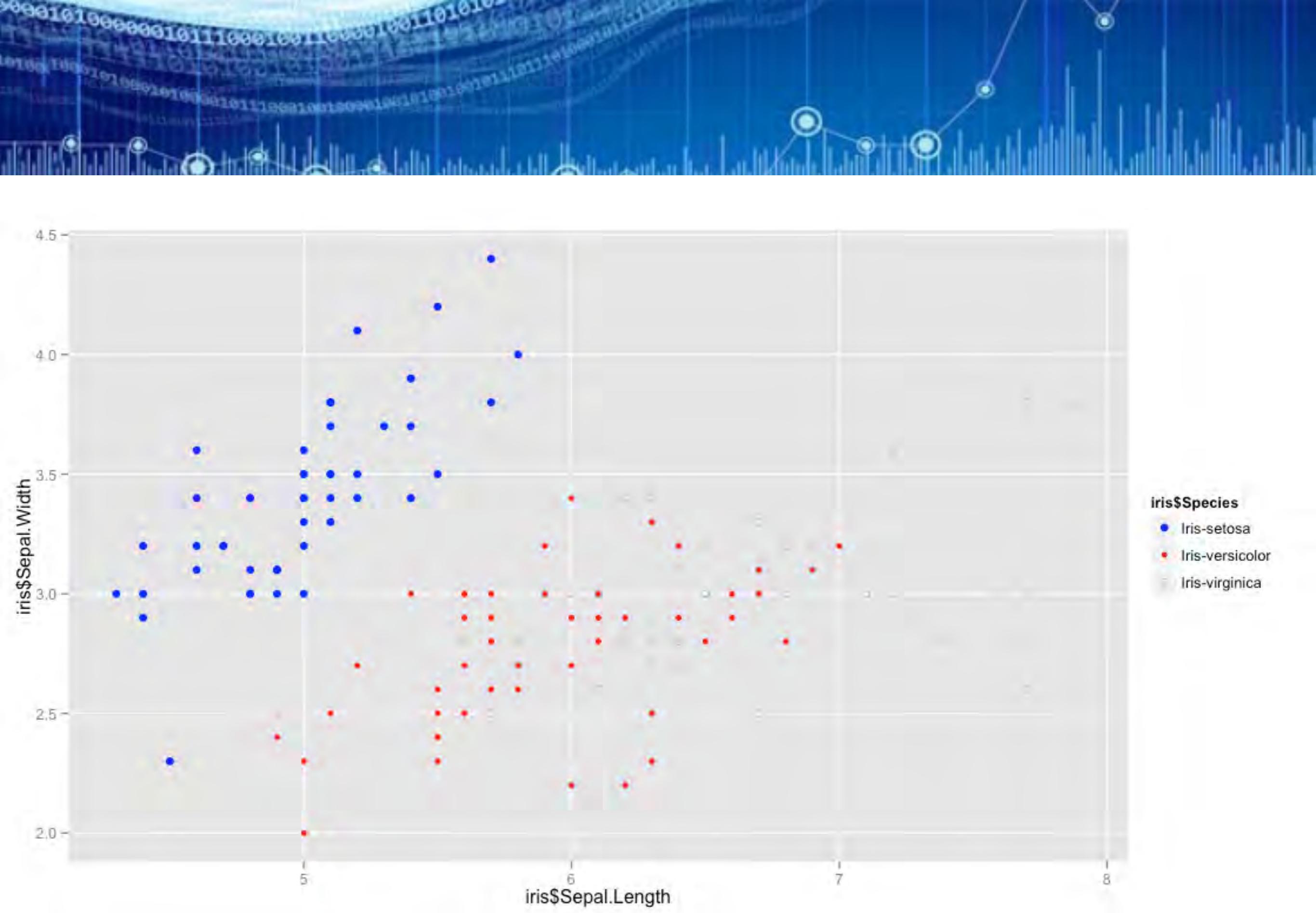


```
> # Plot the histogram  
> hist(iris$Sepal.Length, breaks=10, prob=T)  
> # Plot the density curve  
> lines(density(iris$Sepal.Length))  
>
```





```
ggplot(iris, aes(x=iris$SepalLength, y=iris$SepalWidth,  
group=iris$Species, shape=iris$Species)) +  
  geom_point(aes(colour=iris$Species)) +  
  scale_shape_manual(values=c(19,20,21))+  
  scale_colour_manual(values=c("blue", "red","gray"))
```

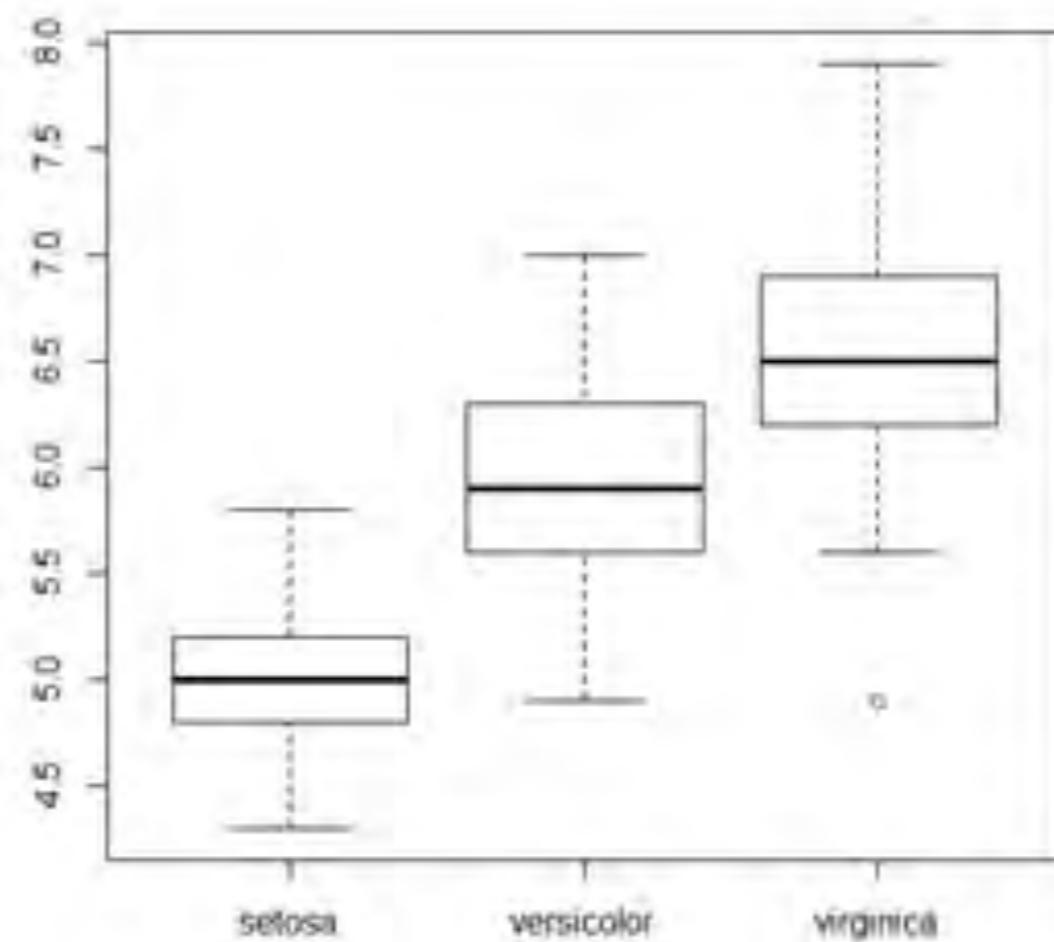


Covariation

If variation describes the behavior within a variable, **covariation describes the behavior between variables. Covariation is the tendency for the values of two or more variables to vary together in a related way.** The best way to spot covariation is to visualize the relationship between two or more variables. How you do that should again depend on the type of variables involved.

1) **A categorical and continuous variable :** We want to explore the distribution of a continuous variable broken down by a categorical variable. Boxplot can help us

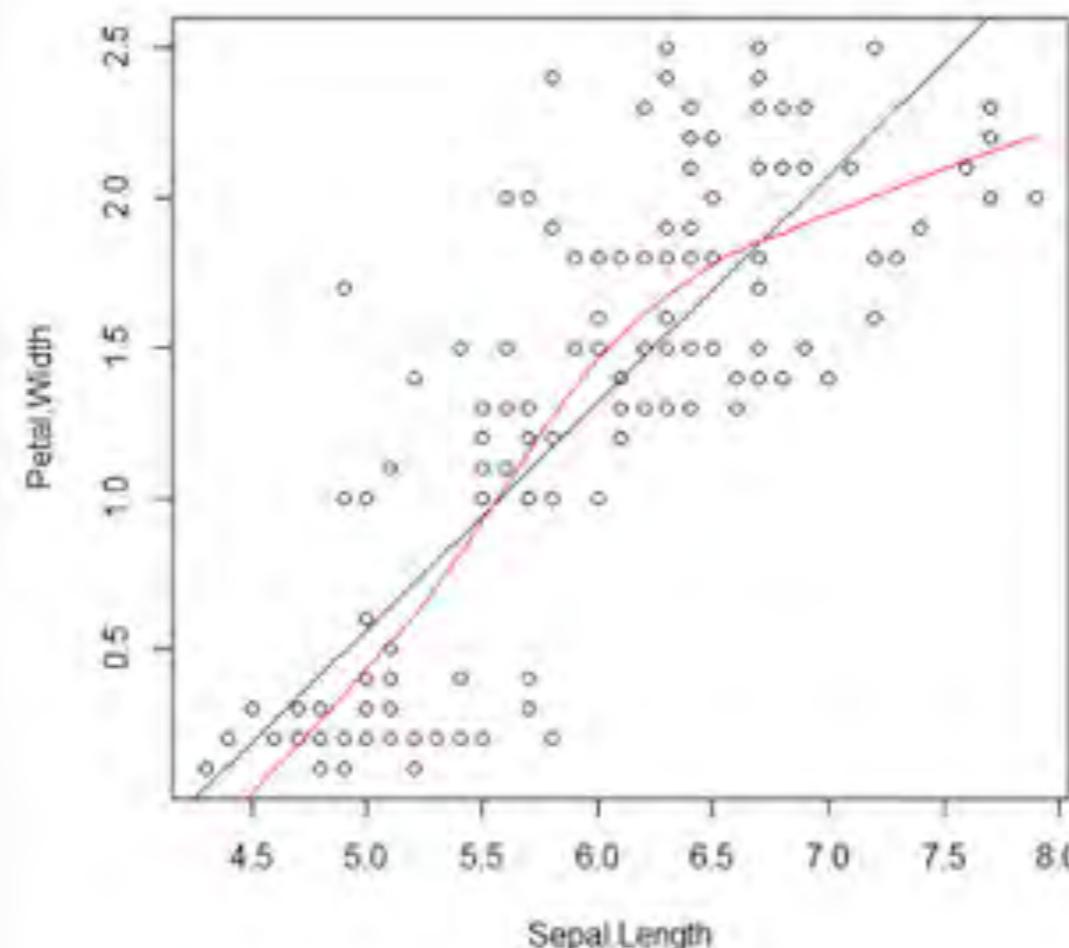
```
> boxplot(Sepal.Length~Species, data=iris)
>
```





2) Two continuous variables : One great way to visualize the covariation between two continuous variables: draw a **scatterplot**.

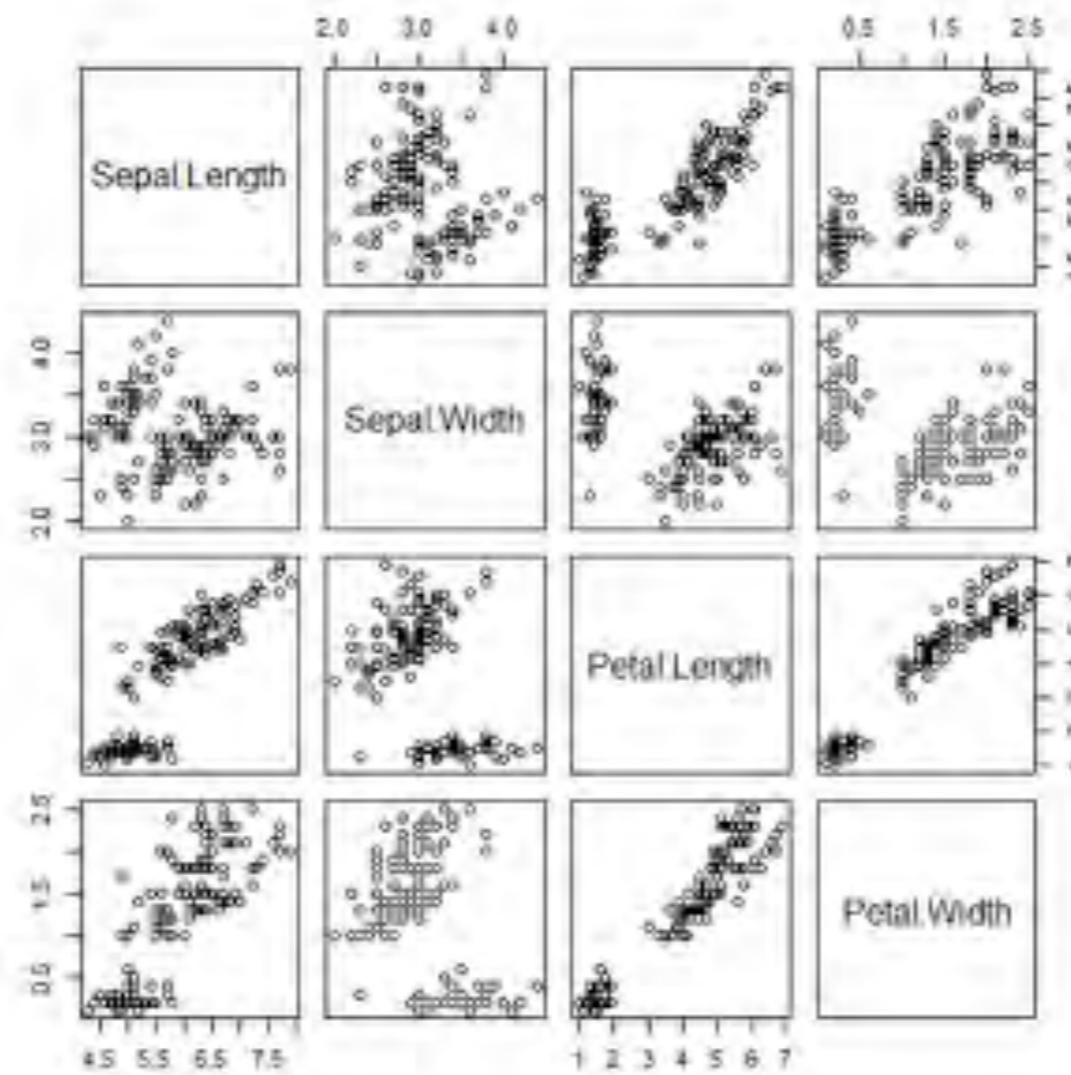
```
> # First plot the 2 variables  
> plot(Petal.Width~Sepal.Length, data=iris)  
> # Learn the regression model  
> model <- lm(Petal.Width~Sepal.Length, data=iris)  
> # Plot the regression line  
> abline(model)  
> # Now learn the local linear model  
> model2 <- lowess(iris$Petal.Width~iris$Sepal.Length)  
> lines(model2, col="red")  
>
```



```

> # Scatter plot for all pairs
> pairs(iris[,c(1,2,3,4)])
> # Compute the correlation matrix
> cor(iris[,c(1,2,3,4)])
      Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length     1.0000000 -0.1170695   0.8716902  0.8179410
Sepal.Width      -0.1170695  1.0000000  -0.4284401 -0.3661259
Petal.Length     0.8716902 -0.4284401   1.0000000  0.9628654
Petal.Width      0.8179410 -0.3661259   0.9628654  1.0000000
>

```



Workshop 4.1 - Getting Data

- Choose data from dataset below.
 - Sales Win or Loss Dataset
 - HR Employee Attrition Dataset
 - Telco Customer Churn Dataset
 - Titanic Survival Dataset
- Import data
- Practice data visualization
- Do step on page 42-51



Data Preparation

Data Science Certification

Preparing Training Data

At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.

- Data sampling
- Data validation and handle missing data
- Normalize numeric value into a uniform range
- Compute aggregated value (a special case is to compute frequency counts)
- Expand categorical field to binary fields
- Discretize numeric value into categories
- Create derived fields from existing fields
- Reduce dimensionality
- Power and Log transformation

Data Sampling

```
> # select 10 records out from iris with replacement
> index <- sample(1:nrow(iris), 10, replace=T)
> index
[1] 133 36 107 140 66 67 36 3 97 37
> irissample <- iris[index,]
> irissample
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
133          6.4        2.8         5.6        2.2 virginica
36           5.0        3.2         1.2        0.2   setosa
107          4.9        2.5         4.5        1.7 virginica
140          6.9        3.1         5.4        2.1 virginica
66           6.7        3.1         4.4        1.4 versicolor
67           5.6        3.0         4.5        1.5 versicolor
36.1         5.0        3.2         1.2        0.2   setosa
3            4.7        3.2         1.3        0.2   setosa
97           5.7        2.9         4.2        1.3 versicolor
37           5.5        3.5         1.3        0.2   setosa
>
```

Impute missing data

- Discard the whole record
- Infer missing value based on the data of other record. Approach is to fill the missing data with the average or the median.

```
> # Create some missing data
> irissample[10, 1] <- NA
> irissample
   Sepal.Length Sepal.Width Petal.Length Petal.Width     Species
133          6.4         2.8          5.6          2.2 virginica
 36          5.0         3.2          1.2          0.2    setosa
107          4.9         2.5          4.5          1.7 virginica
140          6.9         3.1          5.4          2.1 virginica
 66          6.7         3.1          4.4          1.4 versicolor
 67          5.6         3.0          4.5          1.5 versicolor
36.1          5.0         3.2          1.2          0.2    setosa
 3          4.7         3.2          1.3          0.2    setosa
 97          5.7         2.9          4.2          1.3 versicolor
 37           NA         3.5          1.3          0.2    setosa
```

```
> library(e1071)
Loading required package: class
Warning message:
package 'e1071' was built under R version 2.14.2
> fixIris1 <- impute(irissample[,1:4], what='mean')
> fixIris1
  Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.400000     2.8          5.6         2.2
36       5.000000     3.2          1.2         0.2
107      4.900000     2.5          4.5         1.7
140      6.900000     3.1          5.4         2.1
66       6.700000     3.1          4.4         1.4
67       5.600000     3.0          4.5         1.5
36.1     5.000000     3.2          1.2         0.2
3        4.700000     3.2          1.3         0.2
97       5.700000     2.9          4.2         1.3
37      5.655556     3.5          1.3         0.2
> fixIris2 <- impute(irissample[,1:4], what='median')
> fixIris2
  Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.4          2.8          5.6         2.2
36       5.0          3.2          1.2         0.2
107      4.9          2.5          4.5         1.7
140      6.9          3.1          5.4         2.1
66       6.7          3.1          4.4         1.4
67       5.6          3.0          4.5         1.5
36.1     5.0          3.2          1.2         0.2
3        4.7          3.2          1.3         0.2
97       5.7          2.9          4.2         1.3
37      5.6          3.5          1.3         0.2
>
```

Normalize numeric value

```
> library(BBmisc)
> scaleiris <- normalize(iris[,1:4], method = "standardize", range = c(0, 1), margin = 1L, on.constant = "quiet")
> head(scaleiris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1   -0.8976739  1.01560199   -1.335752   -1.311052
2   -1.1392005 -0.13153881   -1.335752   -1.311052
3   -1.3807271  0.32731751   -1.392399   -1.311052
4   -1.5014904  0.09788935   -1.279104   -1.311052
5   -1.0184372  1.24503015   -1.335752   -1.311052
6   -0.5353840  1.93331463   -1.165809   -1.048667
> |
```

Reduce dimensionality

There are many ways to reduce the number of input attributes.

1. Removing irrelevant input variables.
2. Removing redundant input variables.
3. PCA

Motivation

Clustering

One way to summarize a complex real-valued data point
with a single categorical variable

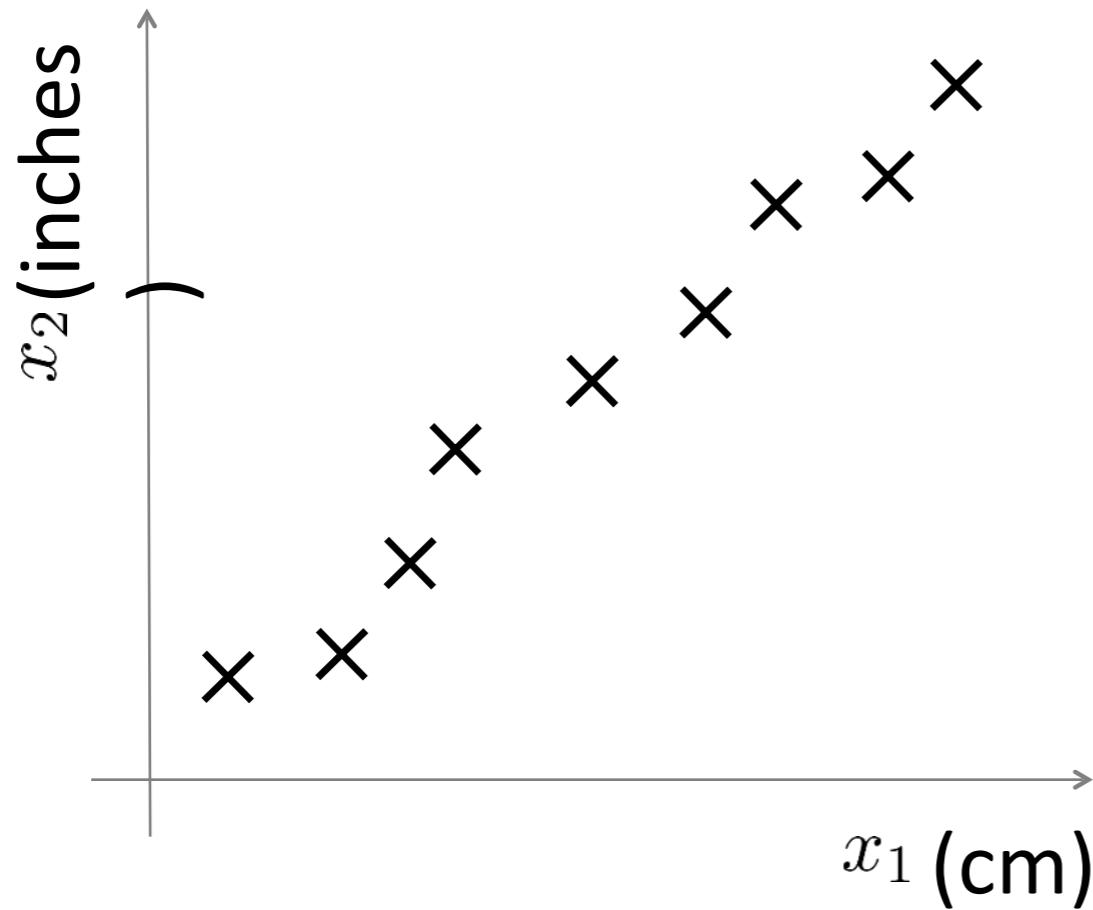
Dimensionality reduction

Another way to simplify complex high-dimensional data
Summarize data with a lower dimensional real valued
vector



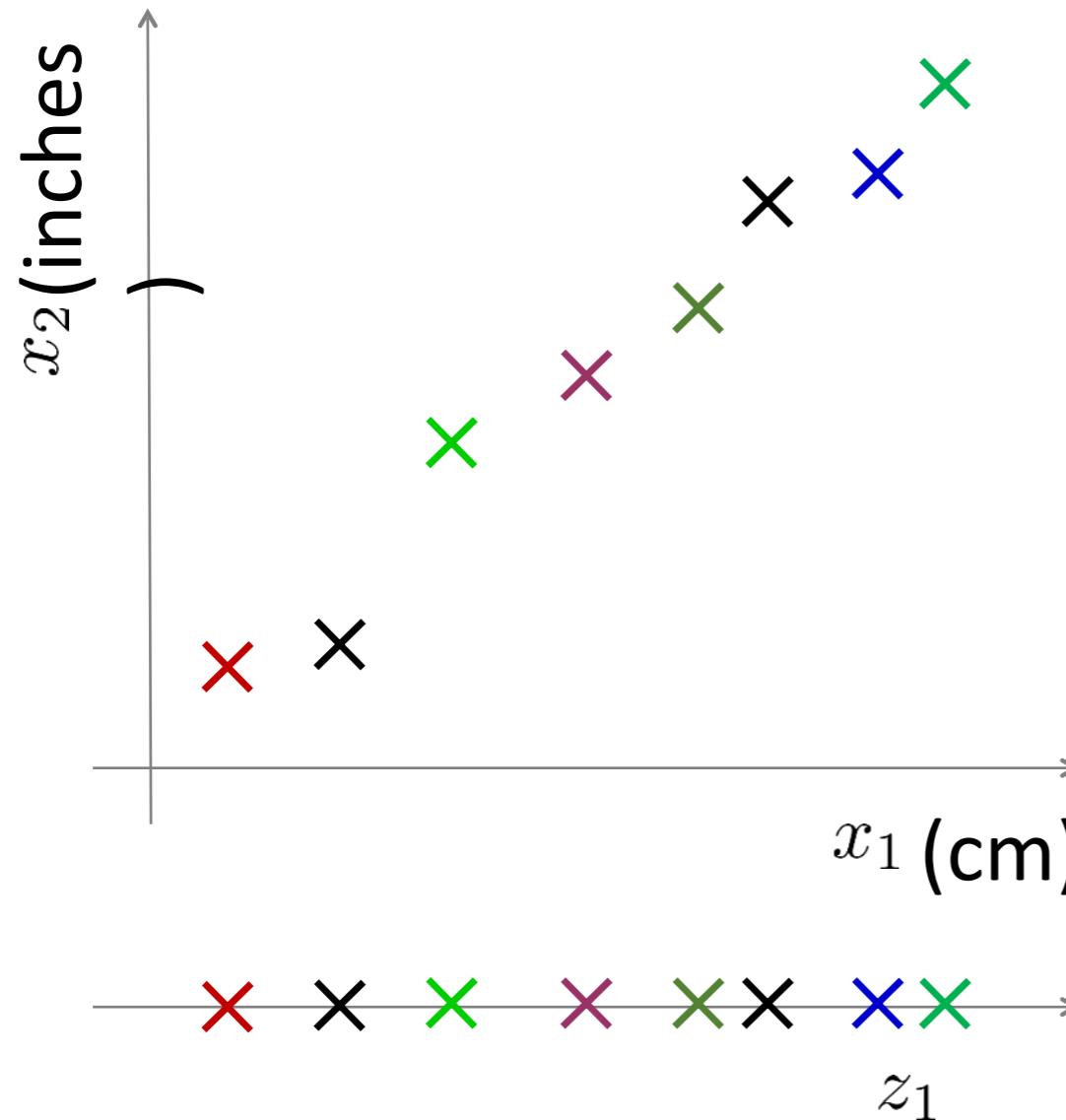
- Given data points in d dimensions
- Convert them to data points in $r < d$ dimensions
- With minimal loss of information

Data Compression



Reduce data from
2D to 1D

Data Compression



Reduce data from
2D to 1D

$$x^{(1)} \rightarrow z^{(1)}$$

$$x^{(2)} \rightarrow z^{(2)}$$

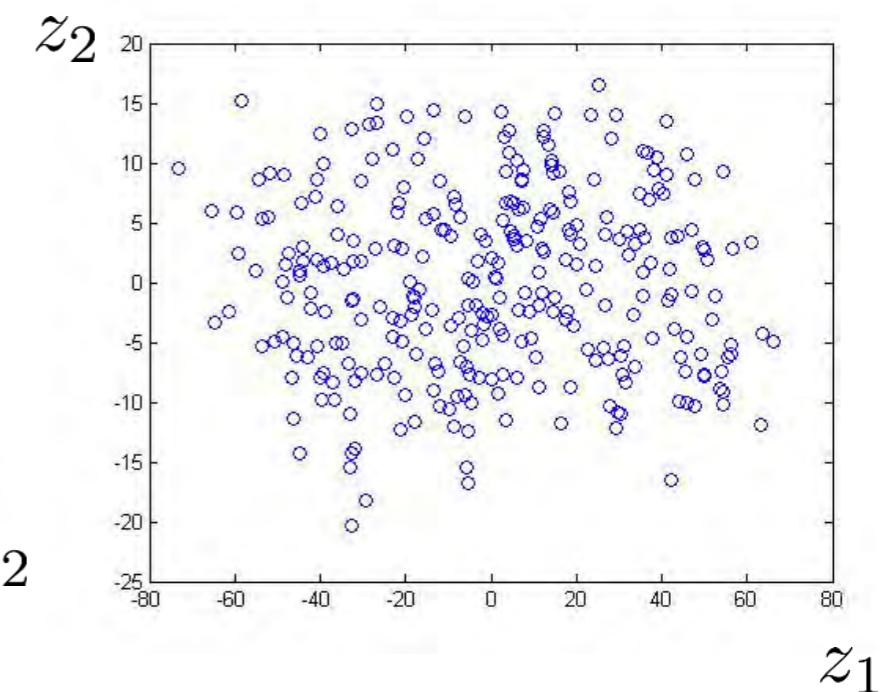
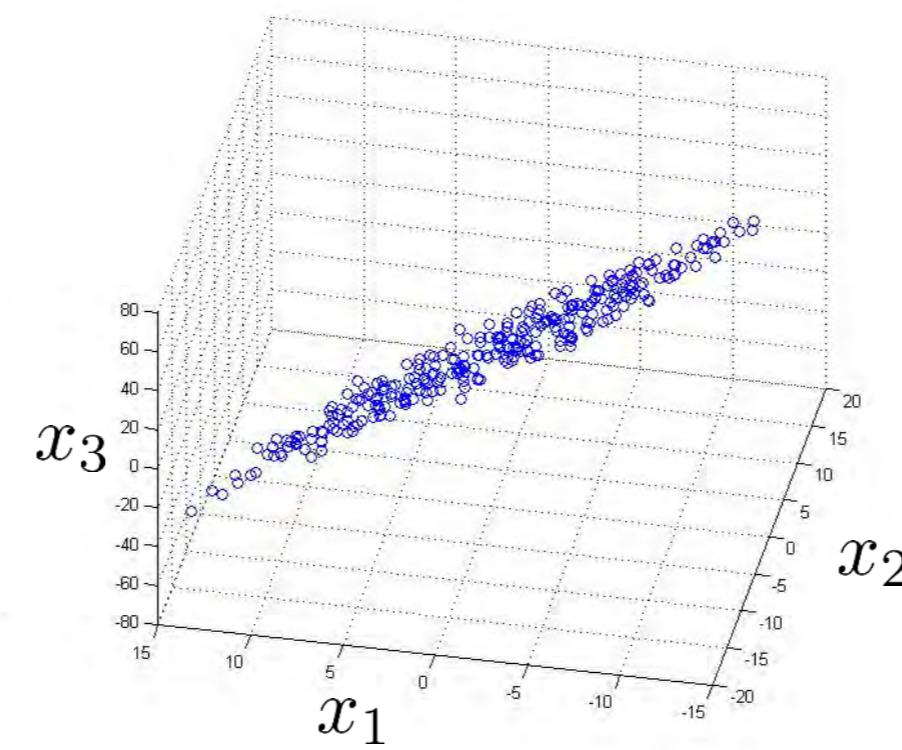
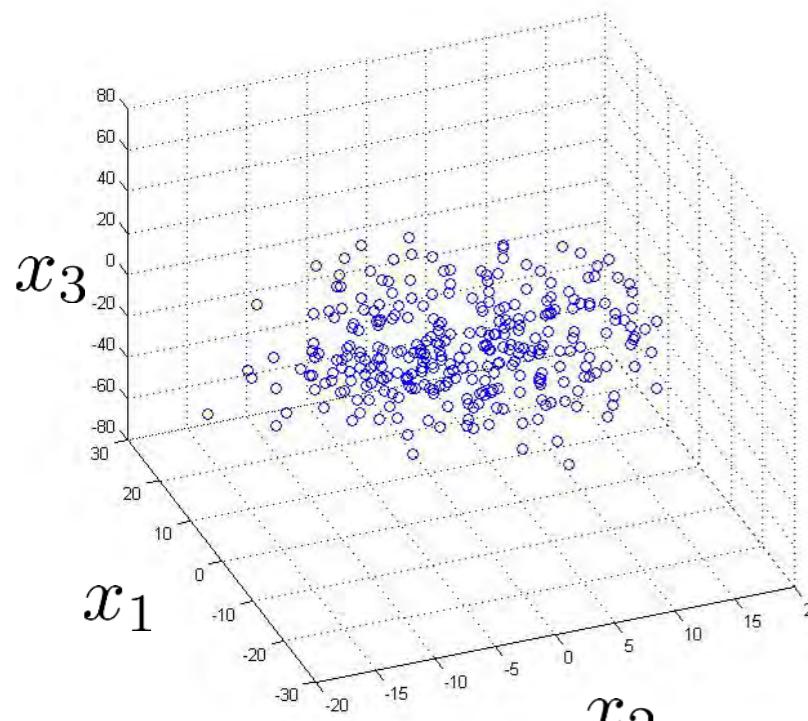
⋮

$$x^{(m)} \rightarrow z^{(m)}$$

Andrew N

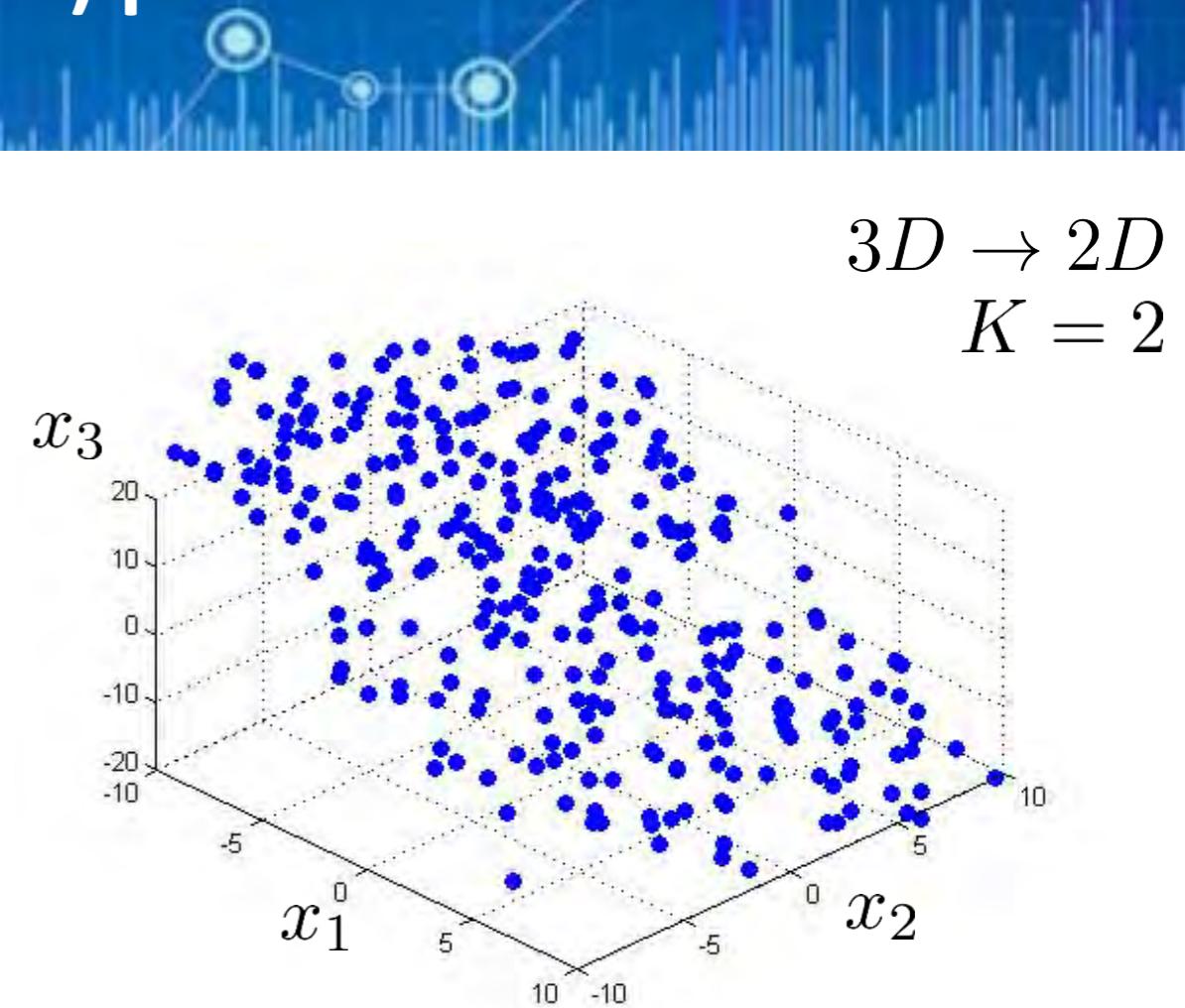
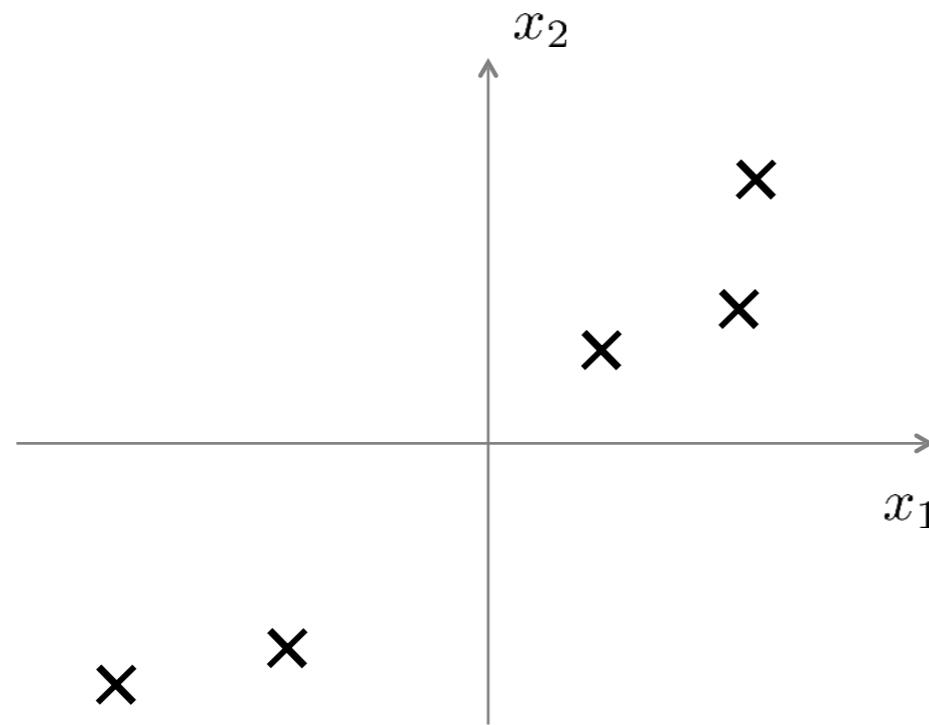
Data Compression

Reduce data from 3D to 2D



Andrew N

Principal Component Analysis (PCA) problem formulation



Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Principal Component Analysis

Goal: Find r -dim projection that best preserves variance

1. Compute mean vector μ and covariance matrix Σ of original points
2. Compute eigenvectors and eigenvalues of Σ
3. Select top r eigenvectors
4. Project points onto subspace spanned by them:

$$y = A(x - \mu)$$

where y is the new point, x is the old one, and the rows of A are the eigenvectors

Covariance

Variance and Covariance:

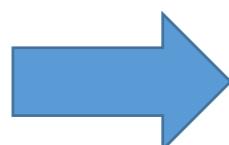
Measure of the “spread” of a set of points around their center of mass(mean)

Variance:

Measure of the deviation from the mean for points in one dimension

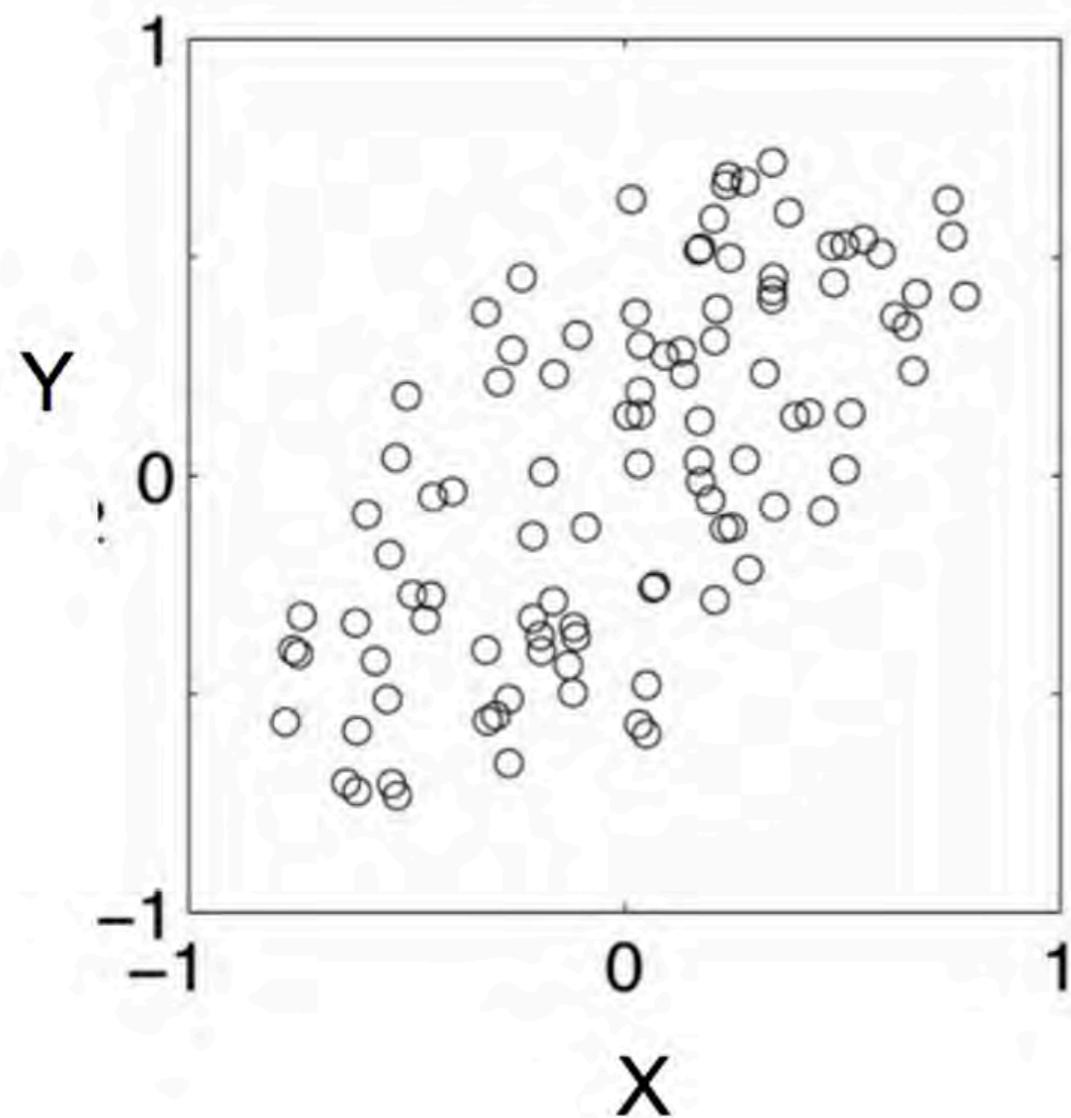
Covariance:

Measure of how much each of the dimensions vary from the mean with **respect to each other**

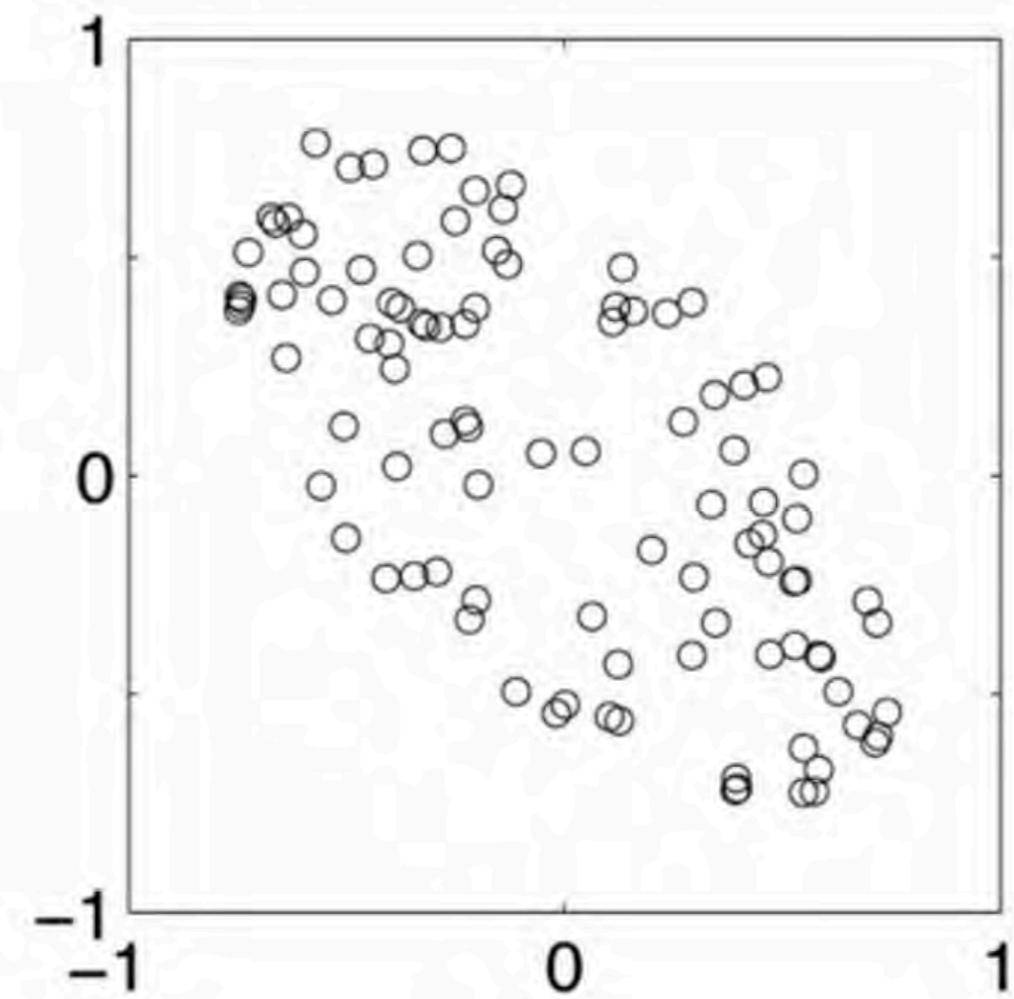


- Covariance is measured between two dimensions
- Covariance sees if there is a relation between two dimensions
- Covariance between one dimension is the variance

positive covariance



negative covariance



Positive: Both dimensions increase or decrease together

Negative: While one increase the other decrease

Covariance

Used to find relationships between dimensions in high dimensional data sets

$$q_{jk} = \frac{1}{N} \sum_{i=1}^N (X_{ij} - E(X_j))(X_{ik} - E(X_k))$$

↓
The Sample mean

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ : Eigenvector or characteristic vector

X: Eigenvalue or characteristic value



- *The zero vector can not be an eigenvector*
- *The value zero can be eigenvalue*

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ : Eigenvector or characteristic vector

X: Eigenvalue or characteristic value

Example

Show $x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ is an eigenvector for $A = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix}$

Solution : $Ax = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

But for $\lambda = 0$, $\lambda x = 0 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Thus, x is an eigenvector of A , and $\lambda = 0$ is an eigenvalue.

The space of all face images

- When viewed as vectors of pixel values, face images are extremely high-dimensional
 - 100×100 image = 10,000 dimensions
 - Slow and lots of storage
- But very few 10,000-dimensional vectors are valid face images
- We want to effectively model the subspace of face images

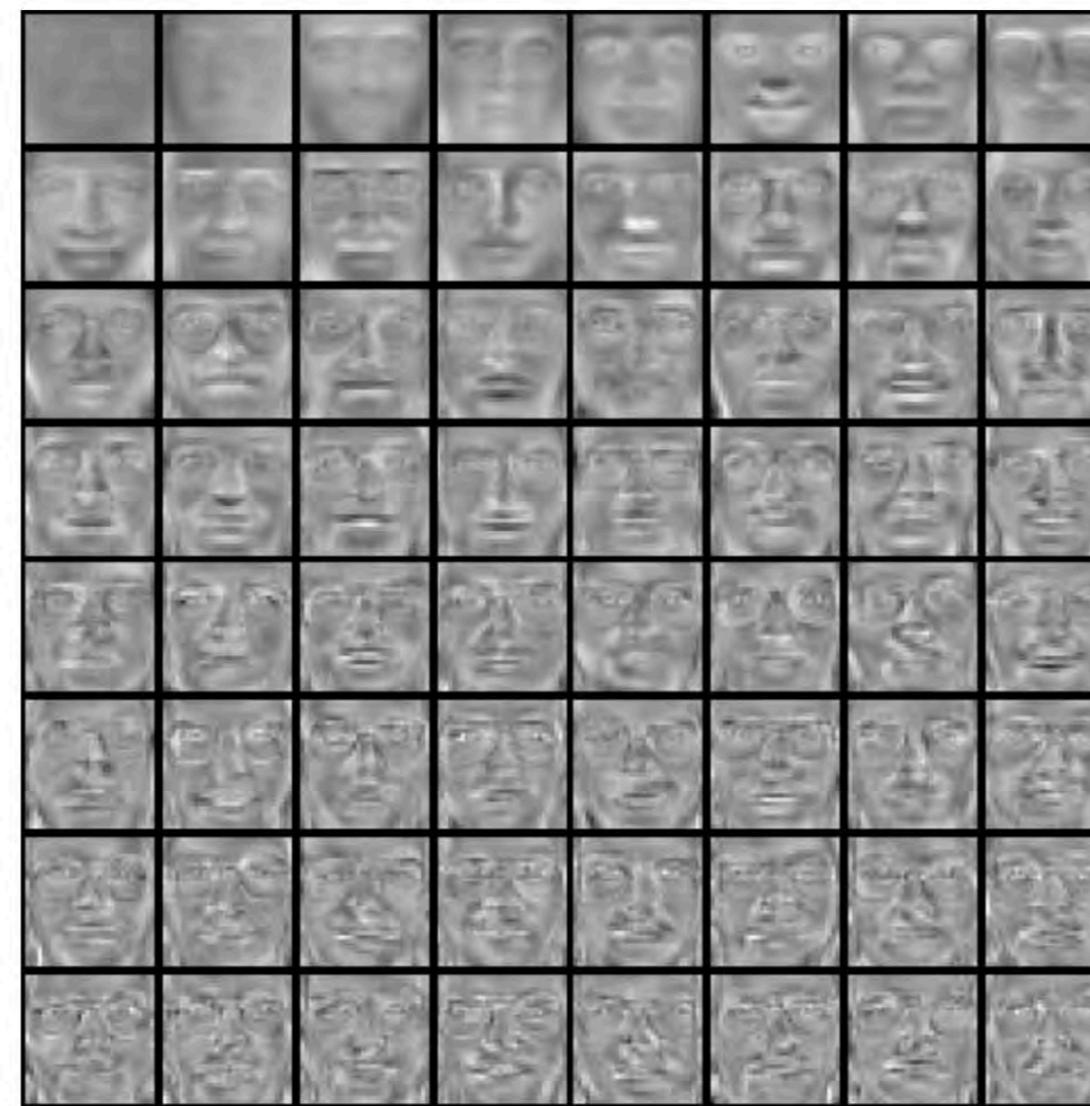


slide by Derek Hoiem

Eigenfaces example

Top eigenvectors: u_1, \dots, u_k

Mean: μ



slide by Derek Hoiem

Representation and reconstruction

- Face \mathbf{x} in “face space” coordinates:



$$\begin{aligned}\mathbf{x} &\rightarrow [\mathbf{u}_1^T(\mathbf{x} - \mu), \dots, \mathbf{u}_k^T(\mathbf{x} - \mu)] \\ &= w_1, \dots, w_k\end{aligned}$$

- Reconstruction:

$$\begin{aligned}\hat{\mathbf{x}} &= \hat{\mu} + \begin{matrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \mathbf{u}_4 \\ \vdots \end{matrix} \\ \hat{\mathbf{x}} &= \mu + w_1\mathbf{u}_1 + w_2\mathbf{u}_2 + w_3\mathbf{u}_3 + w_4\mathbf{u}_4 + \dots\end{aligned}$$

Reconstruction

$P = 4$



$P = 200$

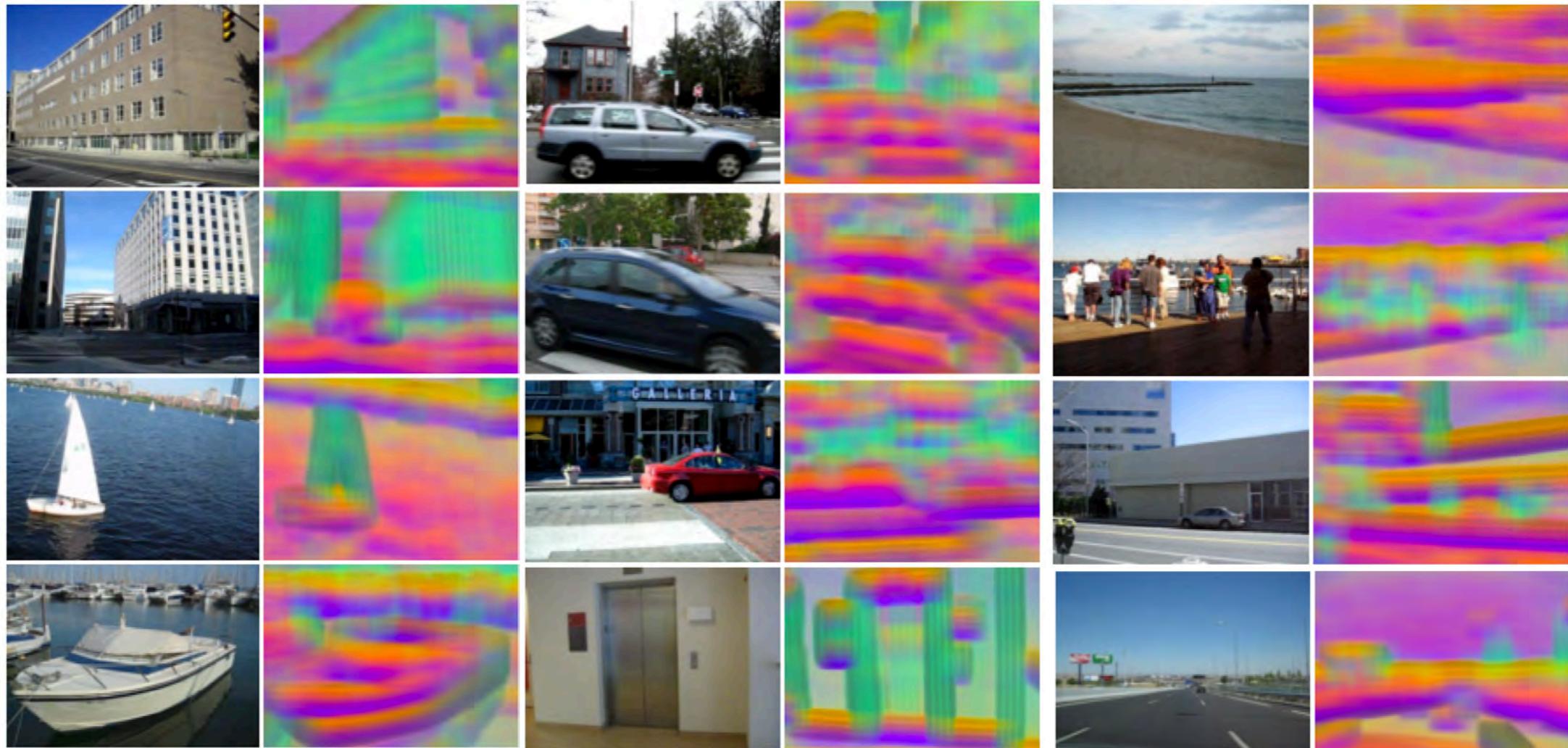


$P = 400$



After computing eigenfaces using 400 face images from ORL face database

SIFT feature visualization



- The top three principal components of SIFT descriptors from a set of images are computed
- Map these principal components to the principal components of the RGB space
- pixels with similar colors share similar structures

Application: Image compression



- Divide the original 372x492 image into patches:
 - Each patch is an instance that contains 12x12 pixels on a grid
 - View each as a 144-D vector

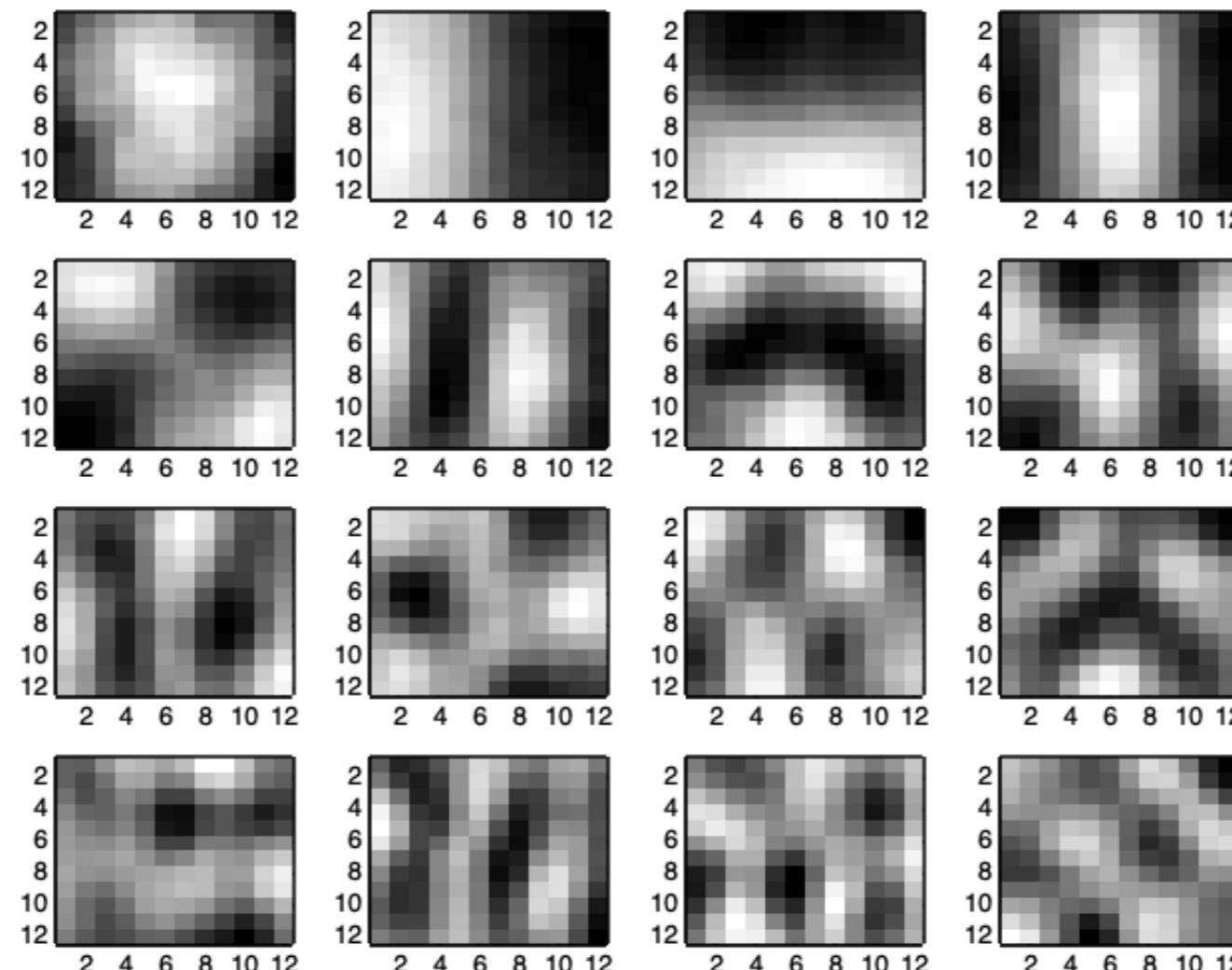
PCA compression: 144D → 60D



PCA compression: 144D \rightarrow 16D



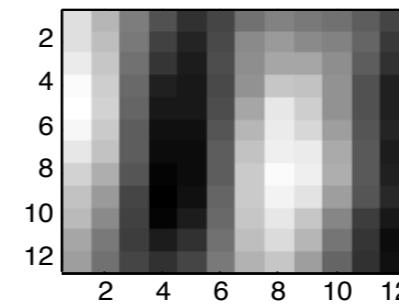
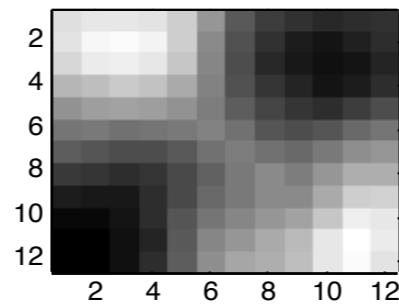
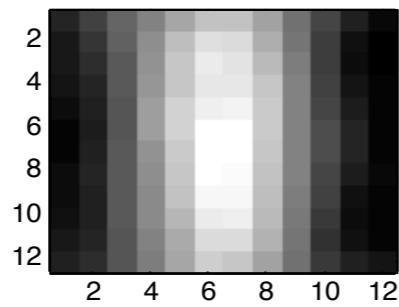
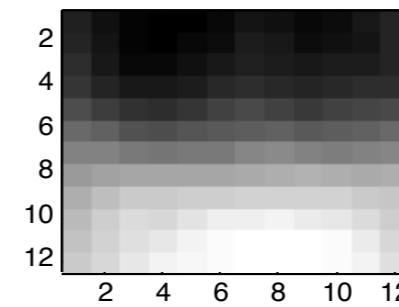
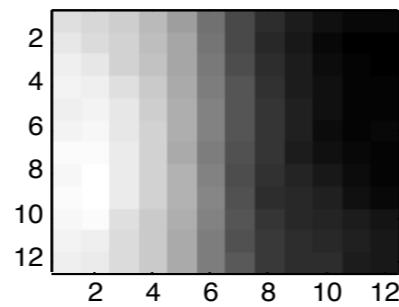
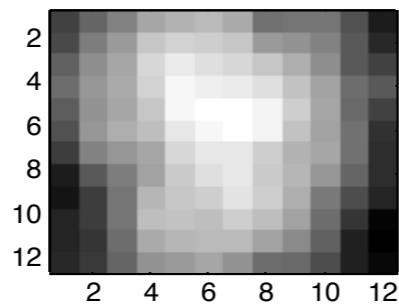
16 most important eigenvectors



PCA compression: 144D → 6D



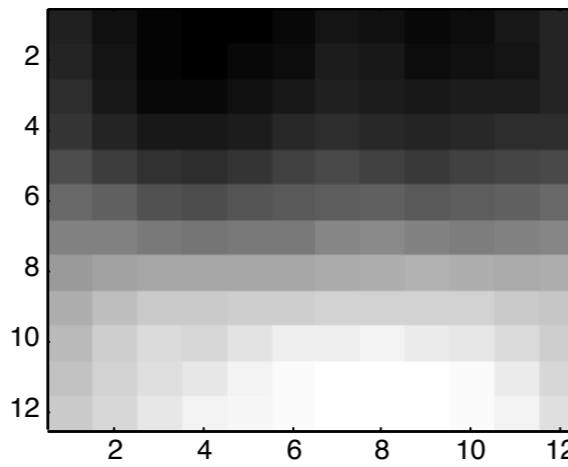
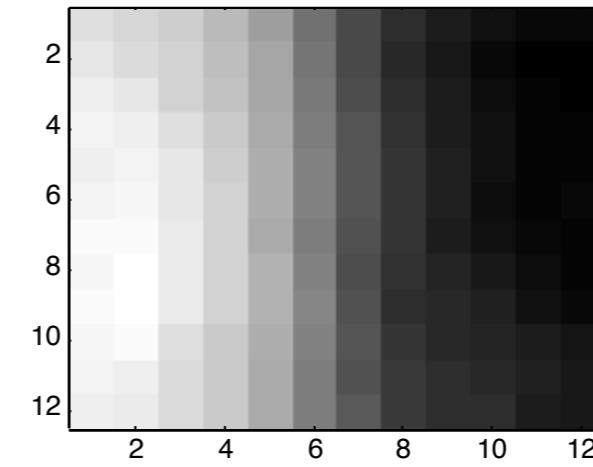
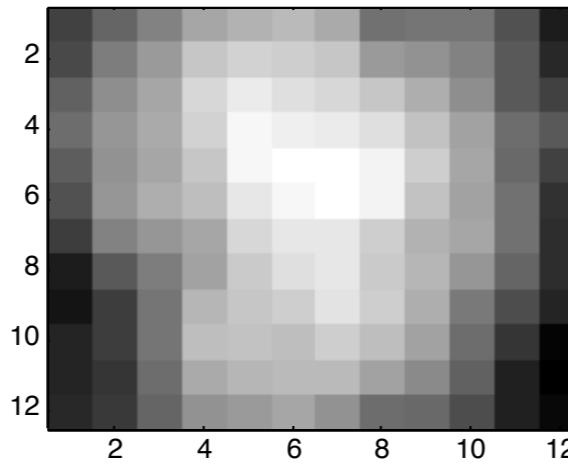
6 most important eigenvectors



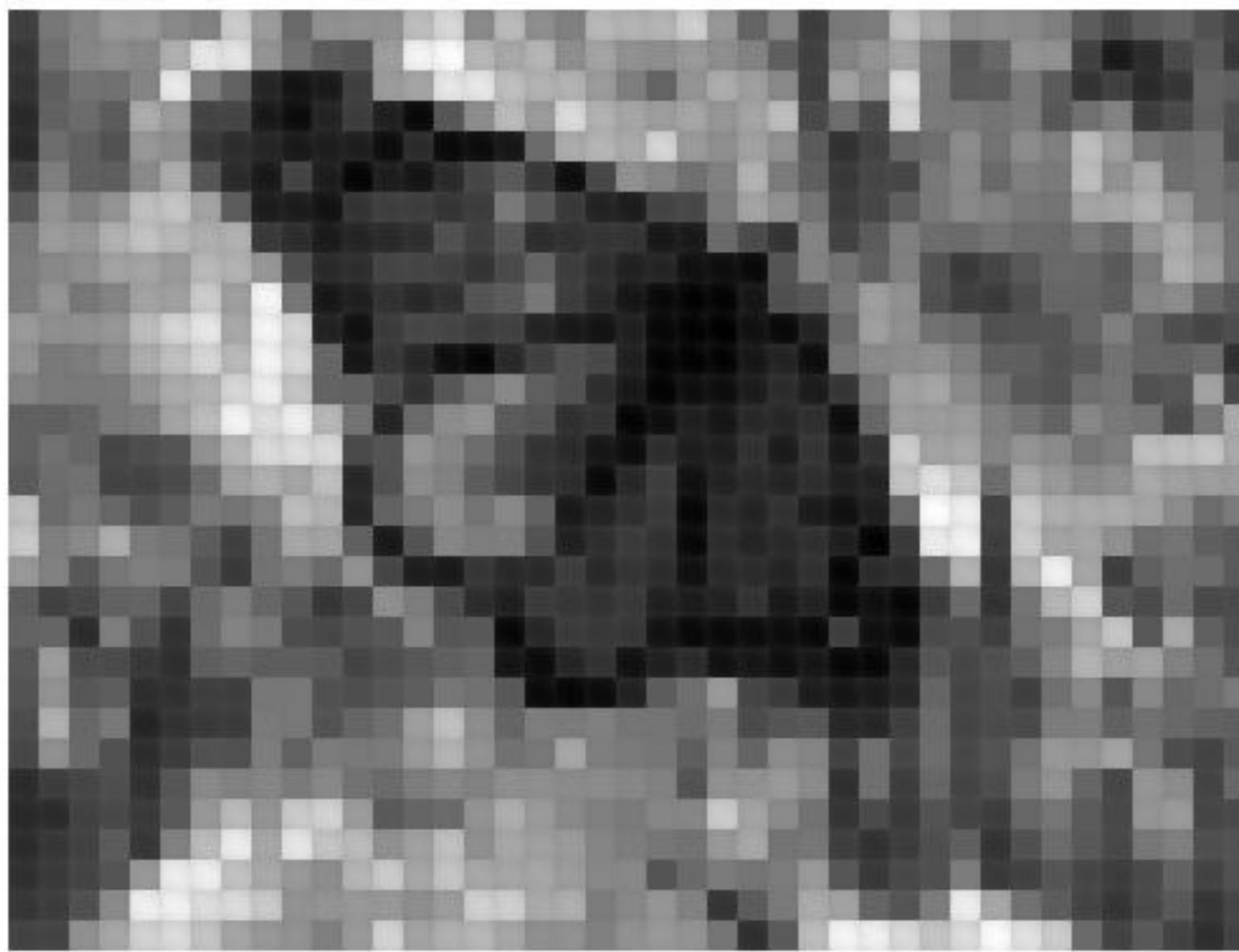
PCA compression: 144D \rightarrow 3D



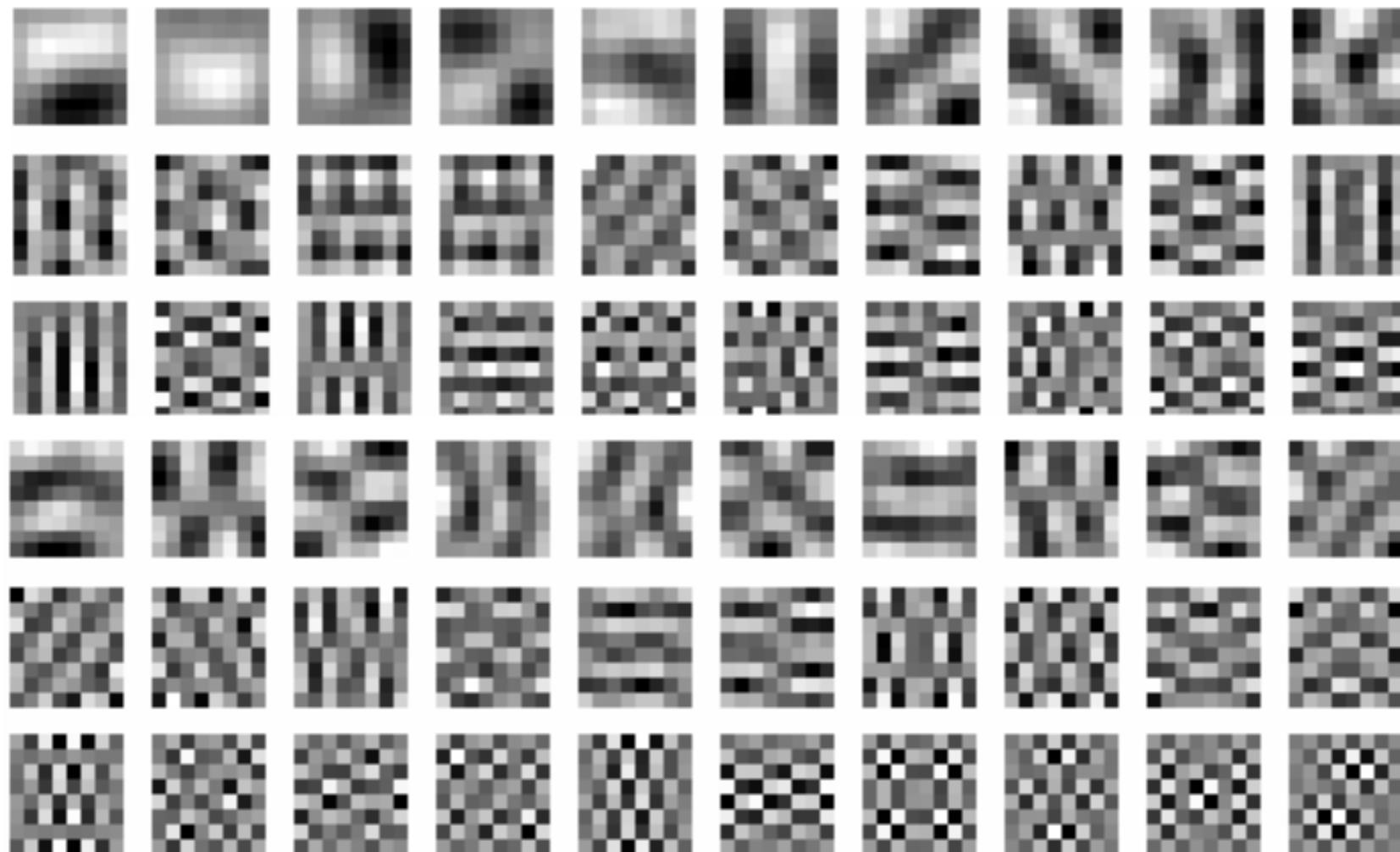
3 most important eigenvectors



PCA compression: 144D \rightarrow 1D

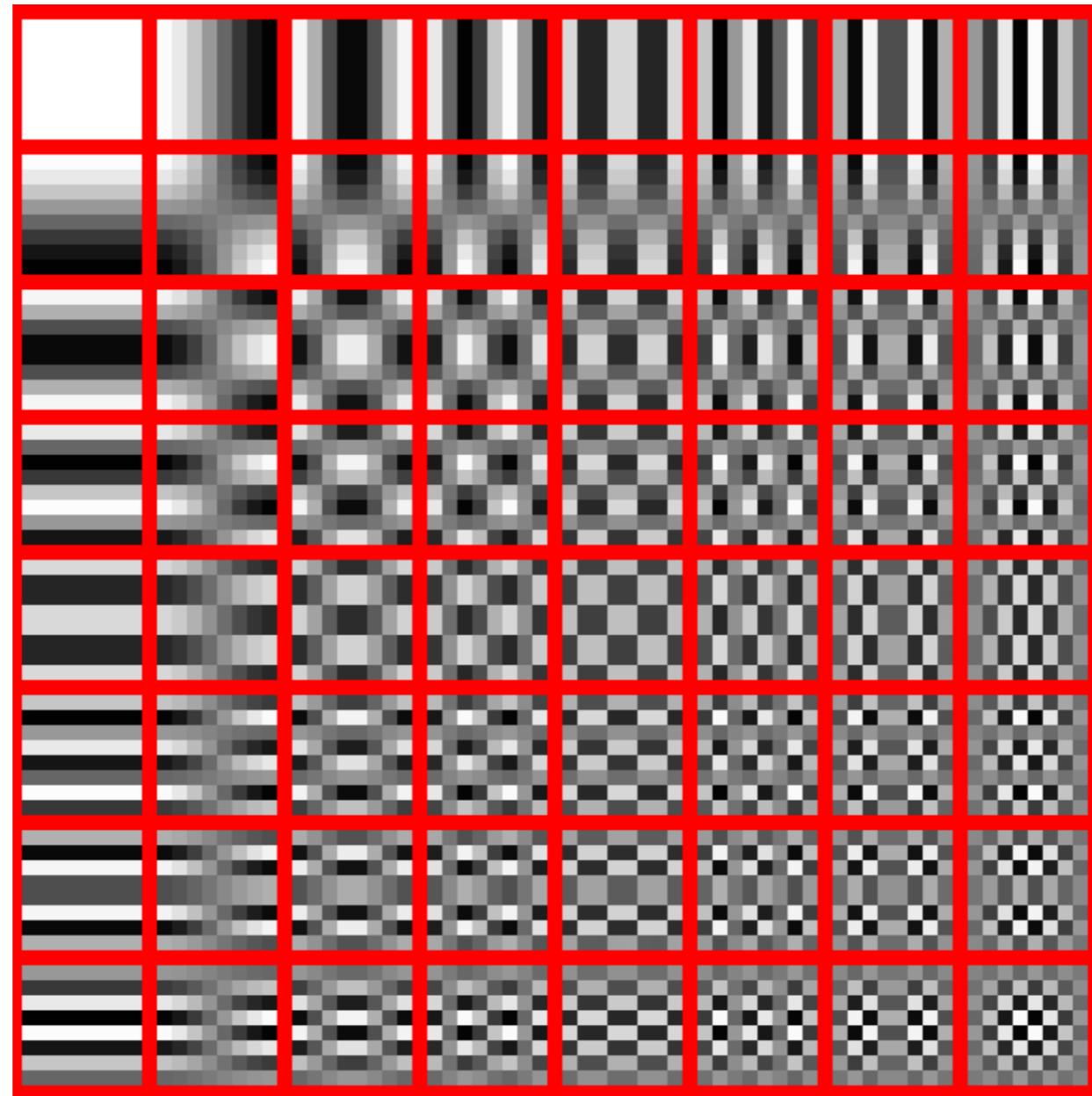


60 most important eigenvectors



Looks like the discrete cosine bases of JPG!...

2D Discrete Cosine Basis



http://en.wikipedia.org/wiki/Discrete_cosine_transform

Dimensionality reduction

PCA (Principal Component Analysis):

Find projection that maximize the variance

ICA (Independent Component Analysis):

Very similar to PCA except that it assumes non-Gaussian features

Multidimensional Scaling:

Find projection that best preserves inter-point distances

LDA(Linear Discriminant Analysis):

Maximizing the component axes for class-separation

...

Principle Component Analysis

```
# Load data
data(iris)
head(iris, 3)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2   setosa
2          4.9         3.0          1.4         0.2   setosa
3          4.7         3.2          1.3         0.2   setosa

# log transform
log.ir <- log(iris[, 1:4])
ir.species <- iris[, 5]

# apply PCA - scale. = TRUE is highly
# advisable, but default is FALSE.
ir.pca <- prcomp(log.ir,
                  center = TRUE,
                  scale. = TRUE)

# print method
print(ir.pca)

Standard deviations:
[1] 1.7124583 0.9523797 0.3647029 0.1656840

Rotation:
PC1        PC2        PC3        PC4
Sepal.Length 0.5038236 -0.45499872 0.7088547 0.19147575
Sepal.Width -0.3023682 -0.88914419 -0.3311628 -0.09125405
Petal.Length 0.5767881 -0.03378802 -0.2192793 -0.78618732
Petal.Width  0.5674952 -0.03545628 -0.5829003 0.58044745
```

```
# summary method  
summary(ir.pca)
```

Importance of components:

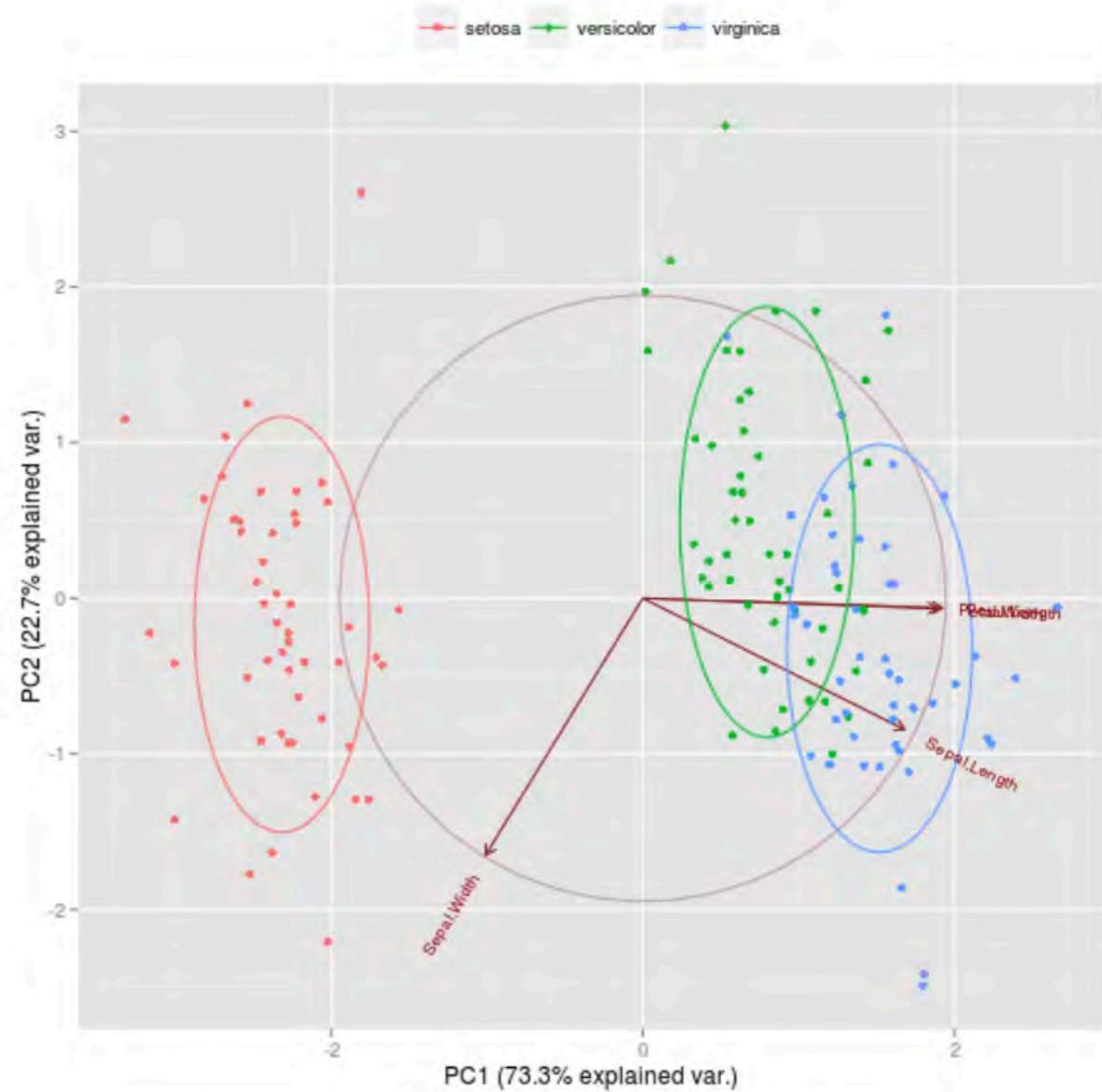
	PC1	PC2	PC3	PC4
Standard deviation	1.7125	0.9524	0.36470	0.16568
Proportion of Variance	0.7331	0.2268	0.03325	0.00686
Cumulative Proportion	0.7331	0.9599	0.99314	1.00000

```
# Predict PCs  
predict(ir.pca,  
        newdata=tail(log.ir, 2))
```

	PC1	PC2	PC3	PC4
149	1.0809930	-1.01155751	-0.7082289	-0.06811063
150	0.9712116	-0.06158655	-0.5008674	-0.12411524

```
library(devtools)
install_github("ggbioplot", "vqv")           install_github("vqv/ggbioplot")

library(ggbioplot)
g <- ggbioplot(ir.pca, obs.scale = 1, var.scale = 1,
               groups = ir.species, ellipse = TRUE,
               circle = TRUE)
g <- g + scale_color_discrete(name = '')
g <- g + theme(legend.direction = 'horizontal',
               legend.position = 'top')
print(g)
```



Add derived attributes

```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width, 2))
> head(iris2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio
1          5.1        3.5         1.4       0.2   setosa 1.46
2          4.9        3.0         1.4       0.2   setosa 1.63
3          4.7        3.2         1.3       0.2   setosa 1.47
4          4.6        3.1         1.5       0.2   setosa 1.48
5          5.0        3.6         1.4       0.2   setosa 1.39
6          5.4        3.9         1.7       0.4   setosa 1.38
```

Discretize numeric value into categories

```
library(arules)

data(iris)
x <- iris[,1]

### look at the distribution before discretizing
hist(x, breaks = 20, main = "Data")

def.par <- par(no.readonly = TRUE) # save default
layout(mat = rbind(1:2,3:4))

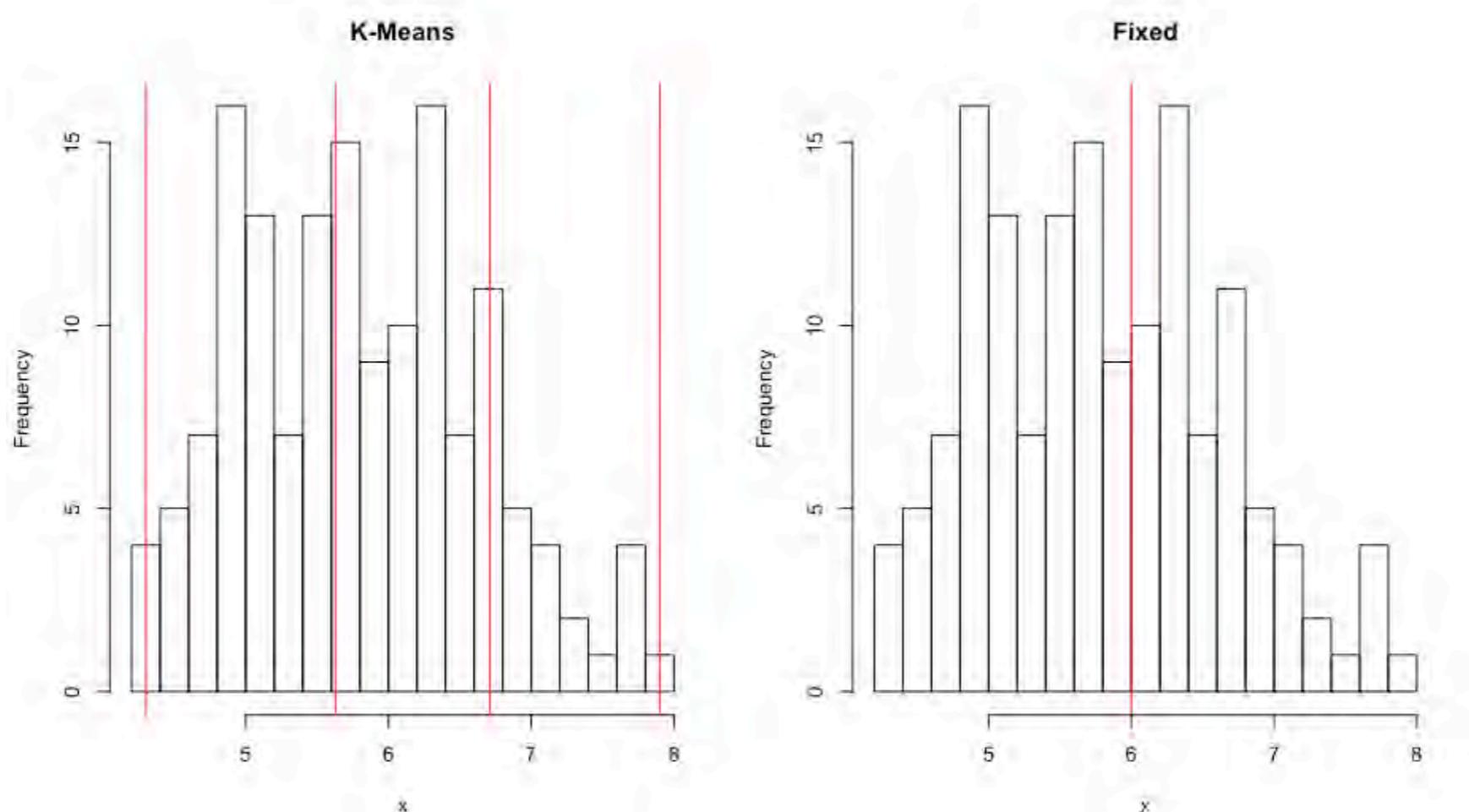
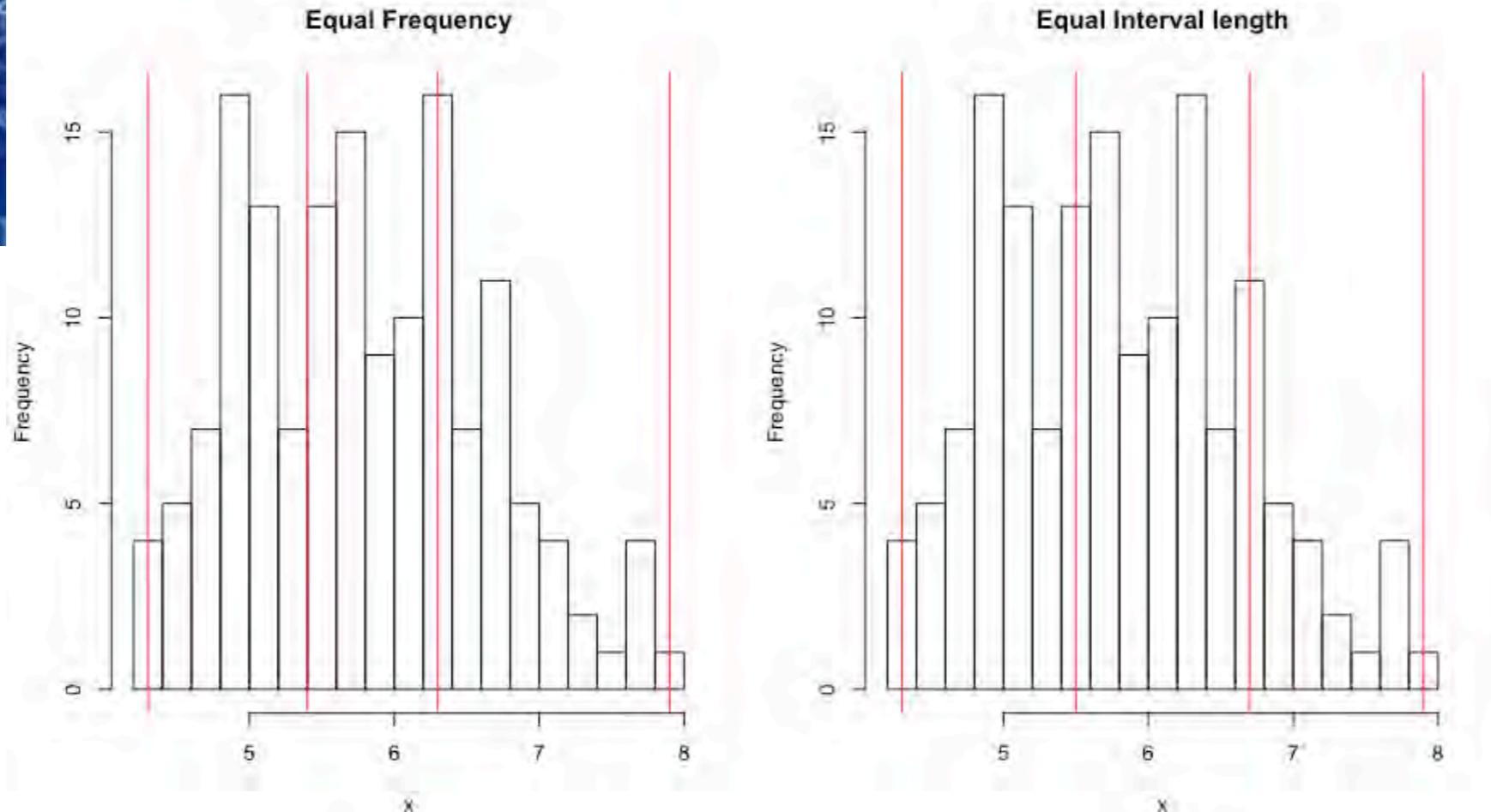
### convert continuous variables into categories (there are 3 types of flowers)
### the default method is equal frequency
table(discretize(x, breaks = 3))
hist(x, breaks = 20, main = "Equal Frequency")
abline(v = discretize(x, breaks = 3,
  onlycuts = TRUE), col = "red")
# Note: the frequencies are not exactly equal because of ties in the data

### equal interval width
table(discretize(x, method = "interval", breaks = 3))
hist(x, breaks = 20, main = "Equal Interval length")
abline(v = discretize(x, method = "interval", breaks = 3,
  onlycuts = TRUE), col = "red")

### k-means clustering
table(discretize(x, method = "cluster", breaks = 3))
hist(x, breaks = 20, main = "K-Means")
abline(v = discretize(x, method = "cluster", breaks = 3,
  onlycuts = TRUE), col = "red")

### user-specified (with labels)
table(discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
  labels = c("small", "large")))
hist(x, breaks = 20, main = "Fixed")
abline(v = discretize(x, method = "fixed", breaks = c(-Inf, 6, Inf),
  onlycuts = TRUE), col = "red")

par(def.par) # reset to default
```



Binarize categorical attributes

```
> library(fastDummies)
> results <- dummy_cols(iris)
> results[45:55,]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species Species_setosa Species_versicolor Species_virginica
45          5.1       3.8        1.9       0.4    setosa           1                  0                  0
46          4.8       3.0        1.4       0.3    setosa           1                  0                  0
47          5.1       3.8        1.6       0.2    setosa           1                  0                  0
48          4.6       3.2        1.4       0.2    setosa           1                  0                  0
49          5.3       3.7        1.5       0.2    setosa           1                  0                  0
50          5.0       3.3        1.4       0.2    setosa           1                  0                  0
51          7.0       3.2        4.7      1.4 versicolor         0                  1                  0
52          6.4       3.2        4.5      1.5 versicolor         0                  1                  0
53          6.9       3.1        4.9      1.5 versicolor         0                  1                  0
54          5.5       2.3        4.0      1.3 versicolor         0                  1                  0
55          6.5       2.8        4.6      1.5 versicolor         0                  1                  0
>
```

Workshop 4.2 - Data Preparation

- Use data from Workshop 4.1
- Practice data preparation using step from Page 53-85



Supervised Learning

Data Science Certification

Topic

- **Basic concepts**

- Decision tree induction
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

An example application

An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.

A decision is needed: whether to put a new patient in an intensive-care unit.

Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.

Problem: to predict **high-risk patients** and discriminate them from **low-risk patients**.

Another application

A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,

- age
 - Marital status
 - annual salary
 - outstanding debts
 - credit rating
- etc.

Problem: to decide whether an application should approved, or to classify applications into two categories, **approved** and **not approved**.

Machine learning and our focus

- Like human learning from past experiences.
- A computer does not have “experiences”.
- **A computer system learns from data**, which represent some “past experiences” of an application domain.
- **Our focus:** learn **a target function** that can be used to predict the values of a discrete class attribute, e.g., **approve** or **not-approved**, and **high-risk** or **low risk**.
- The task is commonly called: **Supervised learning, classification**, or **inductive learning**.

The data and the goal

Data: A set of data records (also called examples, instances or cases) described by

k attributes: A_1, A_2, \dots, A_k .

a class: Each example is labelled with a pre-defined class.

Goal: To learn a **classification model** from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

An example: the learning task

- Learn a classification model from the data
- Use the model to classify future loan applications into
 - Yes (approved) and
 - No (not approved)
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

Supervised vs. unsupervised Learning

Supervised learning: classification is seen as supervised learning from examples.

- **Supervision:** The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a “teacher” gives the classes (**supervision**).
- Test data are classified into these classes too.

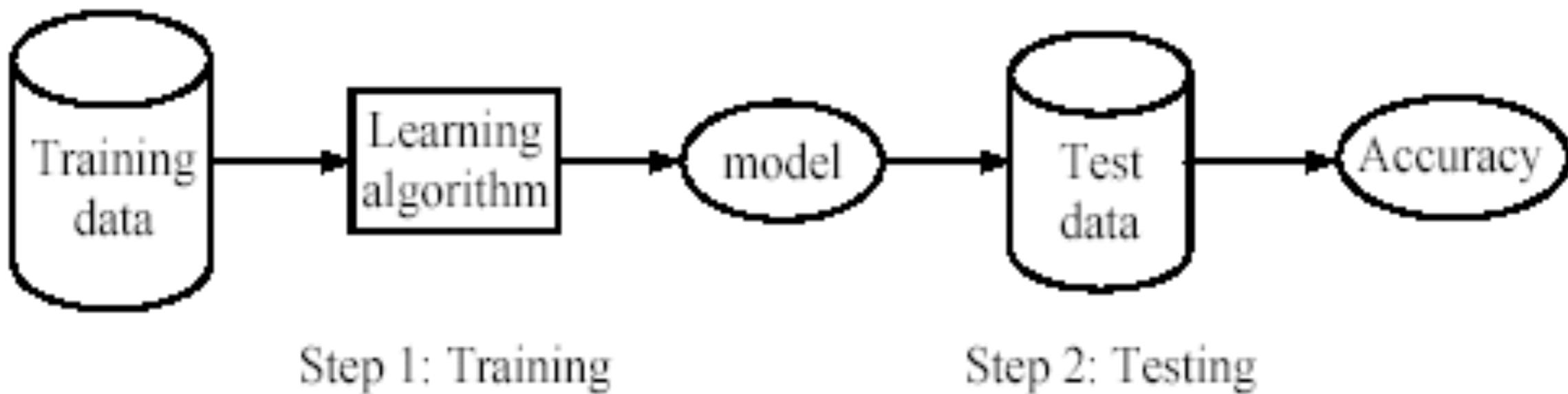
Unsupervised learning (clustering)

- **Class labels of the data are unknown**
- Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- **Learning (training)**: Learn a model using the training data
- **Testing**: Test the model using **unseen test data** to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



What do we mean by learning?

Given

a data set D ,

a task T , and

a performance measure M ,

a computer system is said to **learn** from D to perform the task T if after learning the system's performance on T improves as measured by M .

In other words, the learned model helps the system to perform T better as compared to no learning.

An example

Data: Loan application data

Task: Predict whether a loan should be approved or not.

Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., **Yes**):

$$\text{Accuracy} = 9/15 = 60\%.$$

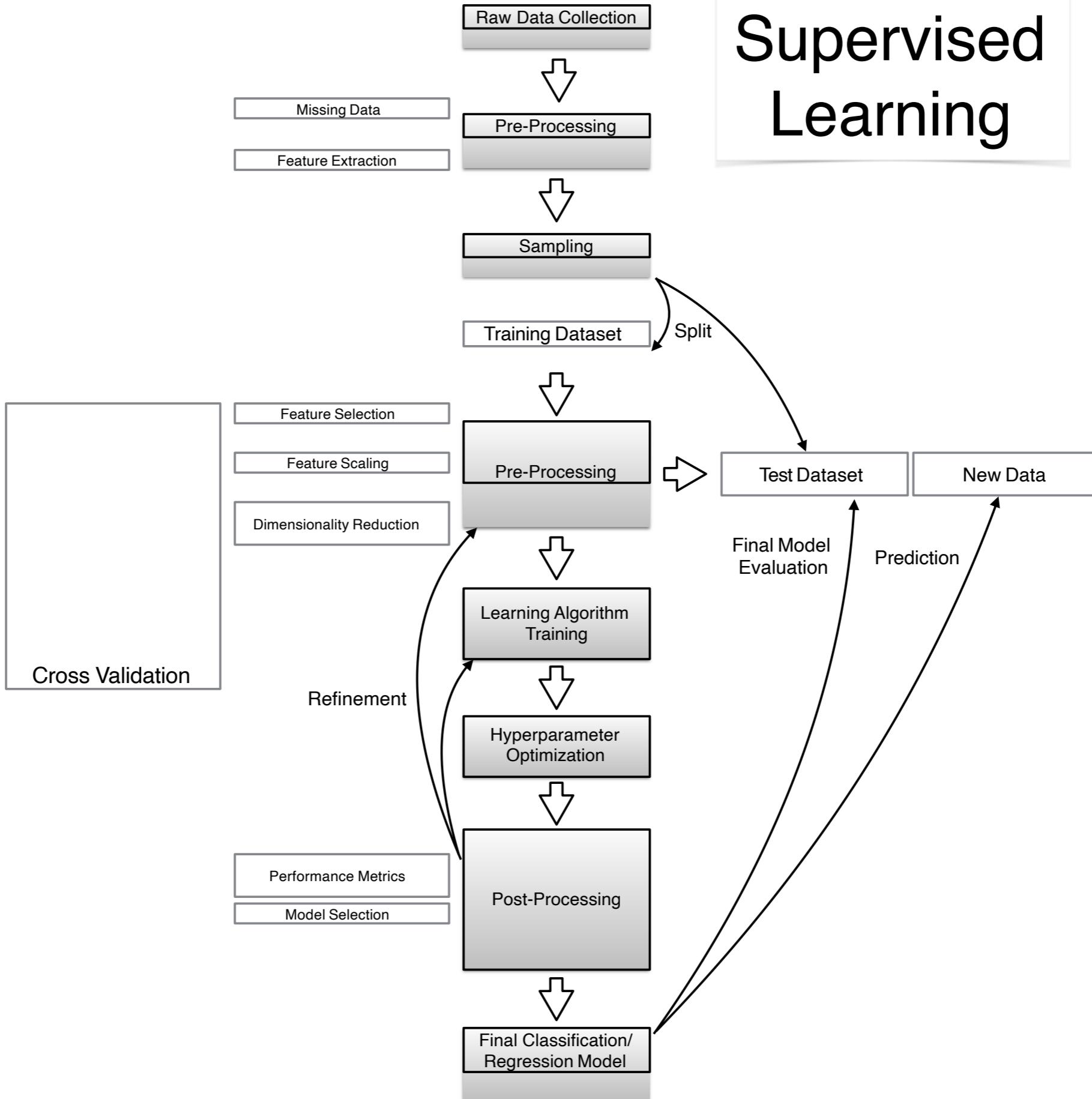
We can do better than 60% with learning.

Fundamental assumption of learning

Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Supervised Learning



Topic

- Basic concepts
- **Decision tree induction**
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Decision tree learning is one of the most widely used techniques for classification.
 - Its classification accuracy is competitive with other methods, and
 - it is very efficient.
- The classification model is a tree, called **decision tree**.
- [C4.5](#) by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

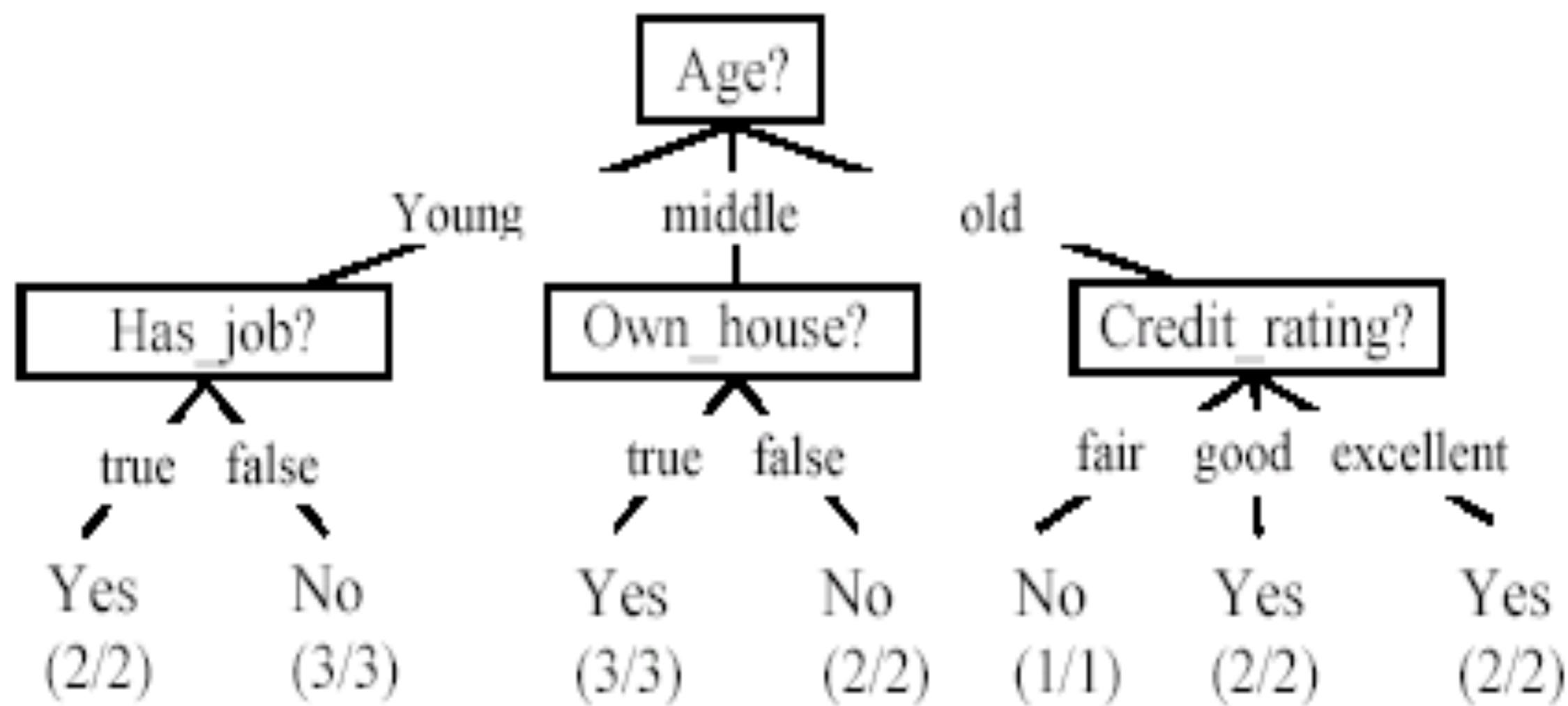
The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

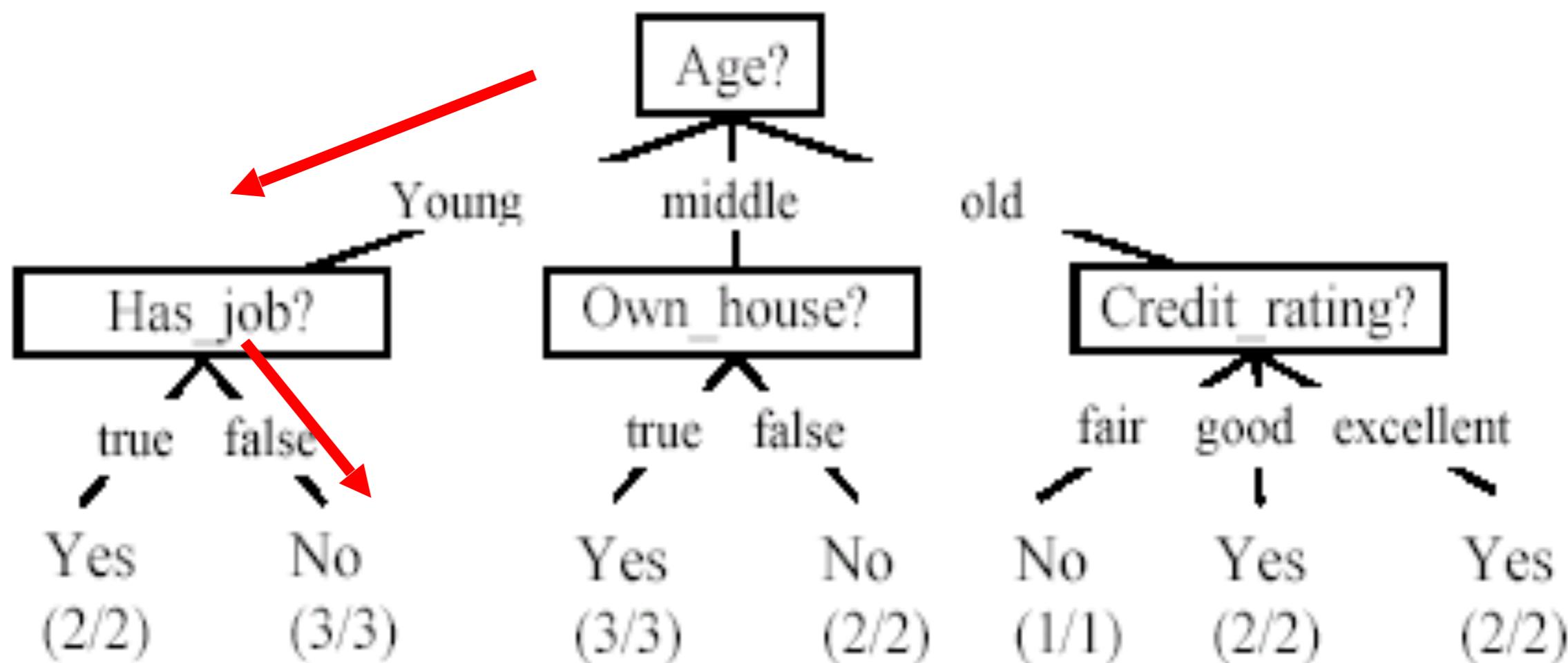
A decision tree from the loan data

- Decision nodes and leaf nodes (classes)



Use the decision tree

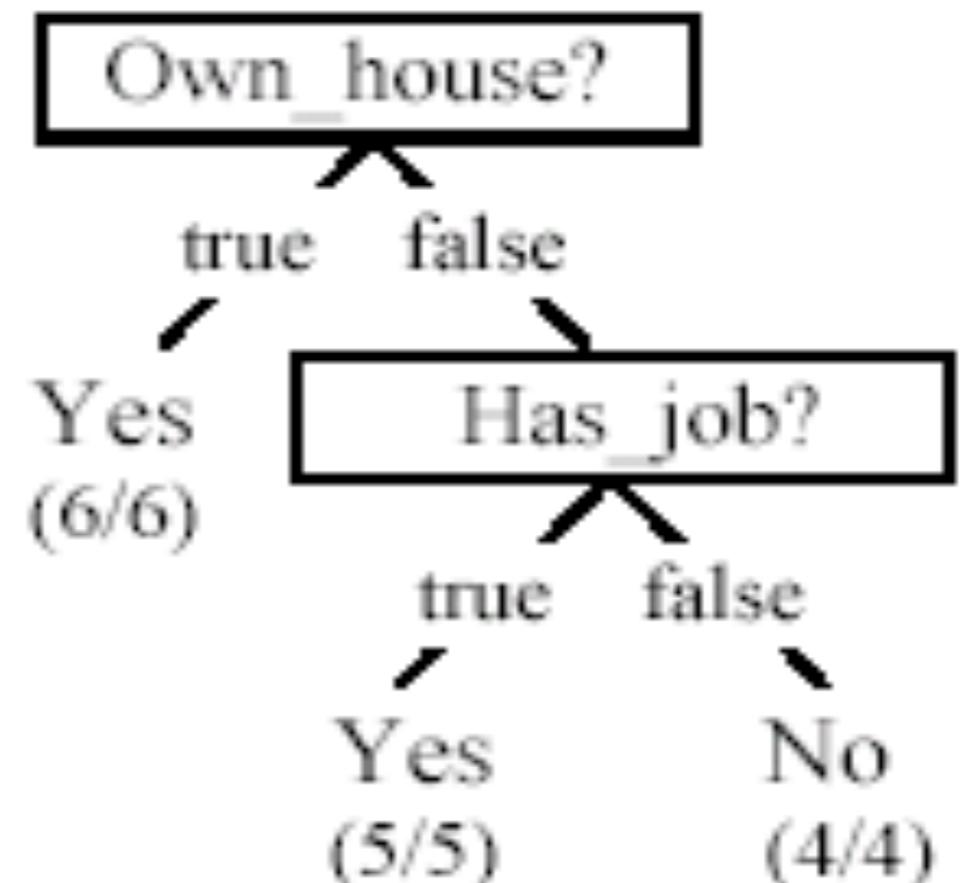
Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	No



Is the decision tree unique?

- No. Here is a simpler tree.
- We want smaller tree and accurate tree.
 - Easy to understand and perform better.

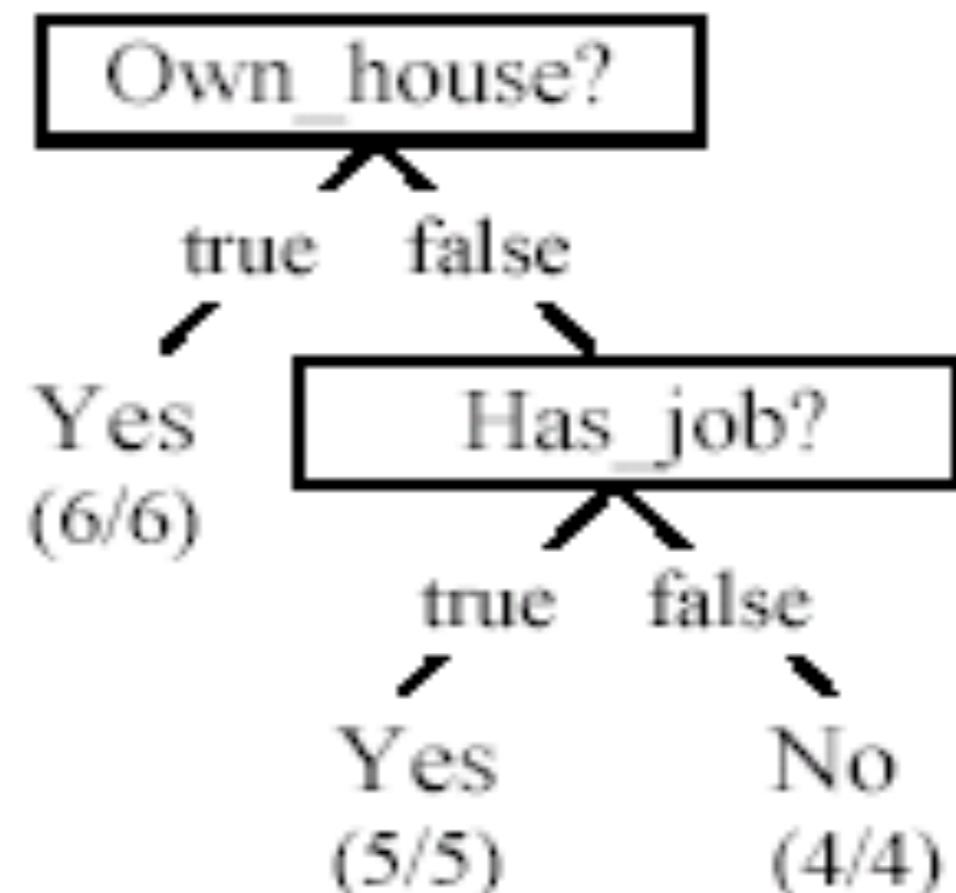
- Finding the best tree is NP-hard.
- All current tree building algorithms are heuristic algorithms



NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP"

From a decision tree to a set of rules

- A decision tree can be converted to a set of rules
- Each path from the root to a leaf is a rule.



Own_house = true → Class = Yes [sup=6/15, conf=6/6]

Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]

Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithm for decision tree learning

Basic algorithm (a greedy **divide-and-conquer** algorithm)

- Assume attributes are categorical now (continuous attributes can be handled too)
 - Tree is constructed in a **top-down recursive manner**
 - At start, all the training examples are at the root
 - Examples are partitioned recursively based on selected attributes
 - Attributes are selected on the basis of an impurity function (e.g., **information gain**)

Conditions for stopping partitioning

- All examples for a given node belong to the same class
- There are no remaining attributes for further partitioning – majority class is the leaf
- There are no examples left

Decision tree learning algorithm

```
. Algorithm decisionTree( $D, A, T$ )
1   if  $D$  contains only training examples of the same class  $c_j \in C$  then
2       make  $T$  a leaf node labeled with class  $c_j$ ;
3   elseif  $A = \emptyset$  then
4       make  $T$  a leaf node labeled with  $c_j$ , which is the most frequent class in  $D$ 
5   else //  $D$  contains examples belonging to a mixture of classes. We select a single
6       // attribute to partition  $D$  into subsets so that each subset is purer
7        $p_0 = \text{impurityEval-1}(D)$ ;
8       for each attribute  $A_i \in \{A_1, A_2, \dots, A_k\}$  do
9            $p_i = \text{impurityEval-2}(A_i, D)$ 
10      end
11      Select  $A_g \in \{A_1, A_2, \dots, A_k\}$  that gives the biggest impurity reduction,
12          computed using  $p_0 - p_g$ ;
13      if  $p_0 - p_g < \text{threshold}$  then //  $A_g$  does not significantly reduce impurity  $p_0$ 
14          make  $T$  a leaf node labeled with  $c_j$ , the most frequent class in  $D$ .
15      else //  $A_g$  is able to reduce impurity  $p_0$ 
16          Make  $T$  a decision node on  $A_g$ ;
17          Let the possible values of  $A_g$  be  $v_1, v_2, \dots, v_m$ . Partition  $D$  into  $m$ 
18          disjoint subsets  $D_1, D_2, \dots, D_m$  based on the  $m$  values of  $A_g$ .
19          for each  $D_j$  in  $\{D_1, D_2, \dots, D_m\}$  do
20              if  $D_j \neq \emptyset$  then
21                  create a branch (edge) node  $T_j$  for  $v_j$  as a child node of  $T$ ;
22                  decisionTree( $D_j, A - \{A_g\}, T_j$ ) //  $A_g$  is removed
23              end
24          end
25      end
26  end
```

Choose an attribute to partition data

- The *key* to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.

A subset of data is **pure** if all instances belong to the same class.

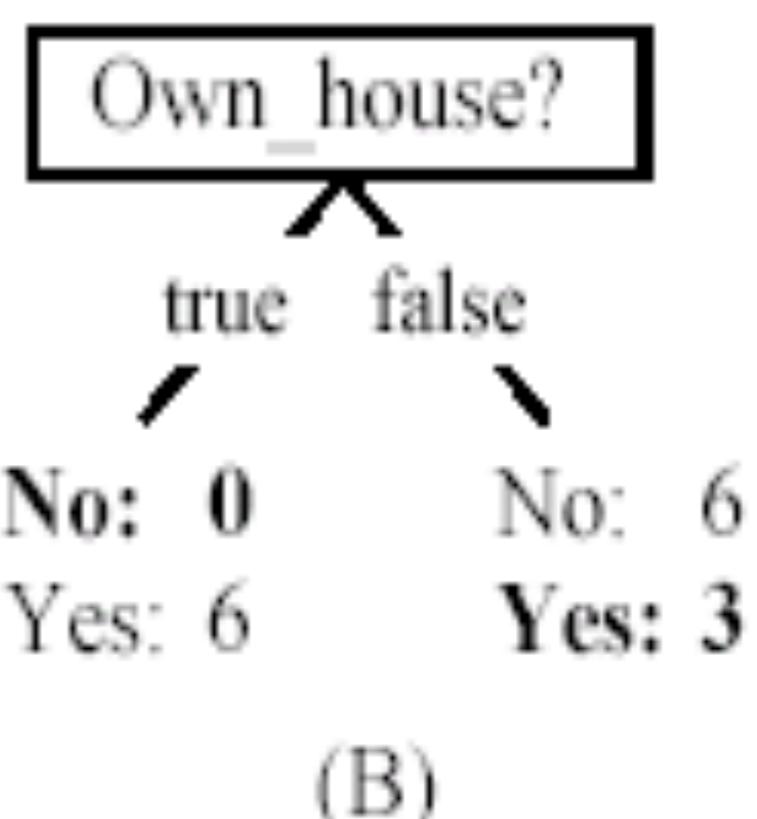
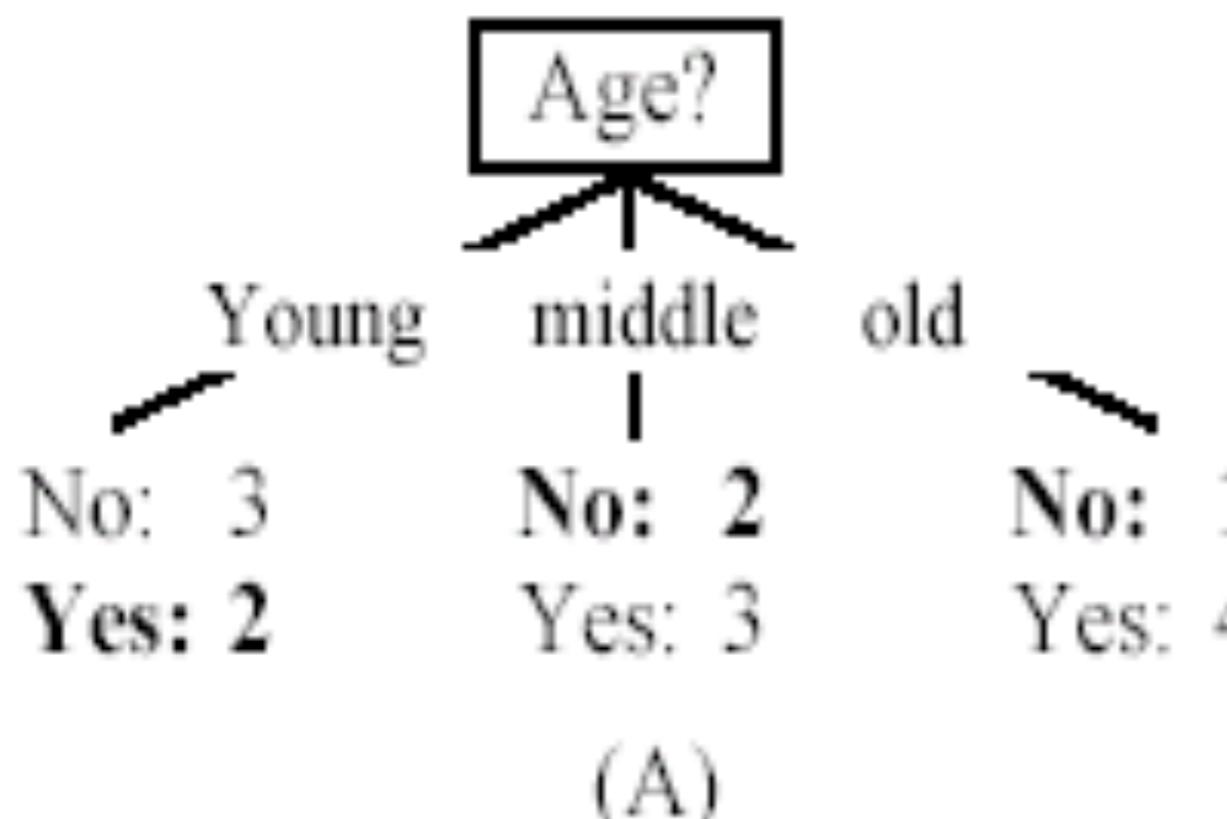
- The *heuristic* in C4.5 is to choose the attribute with the maximum **Information Gain** or **Gain Ratio** based on information theory.

The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Two possible roots, which is better?

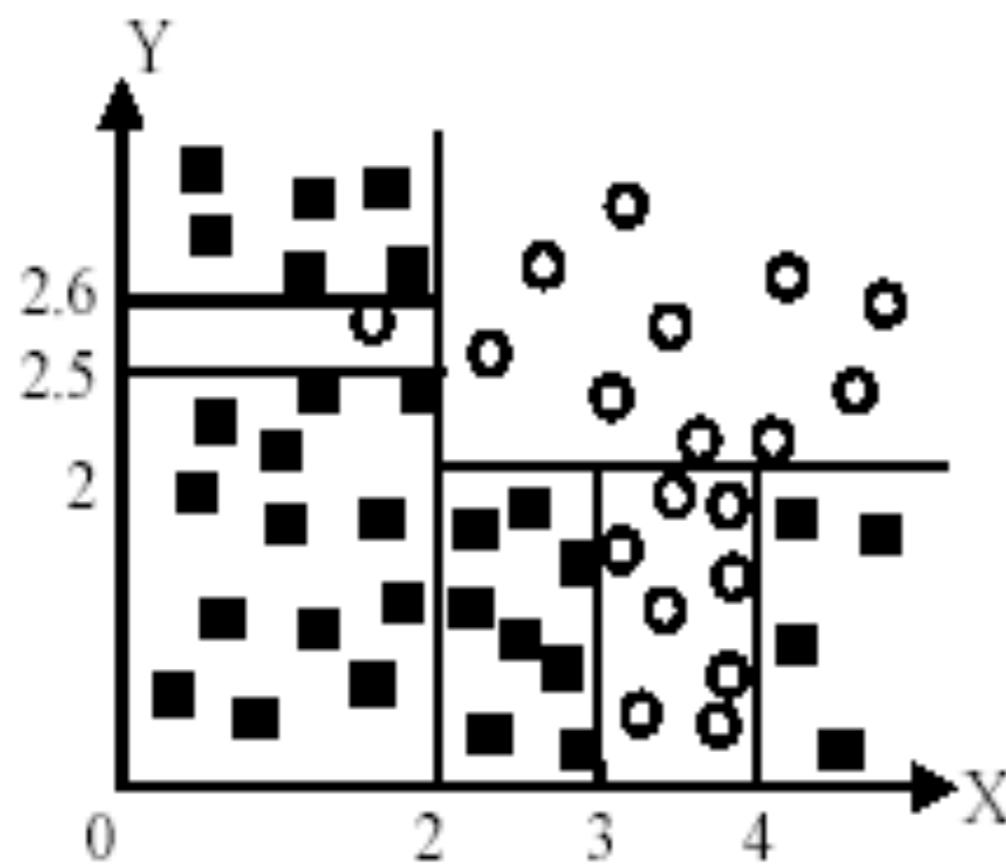


- Fig. (B) seems to be better.

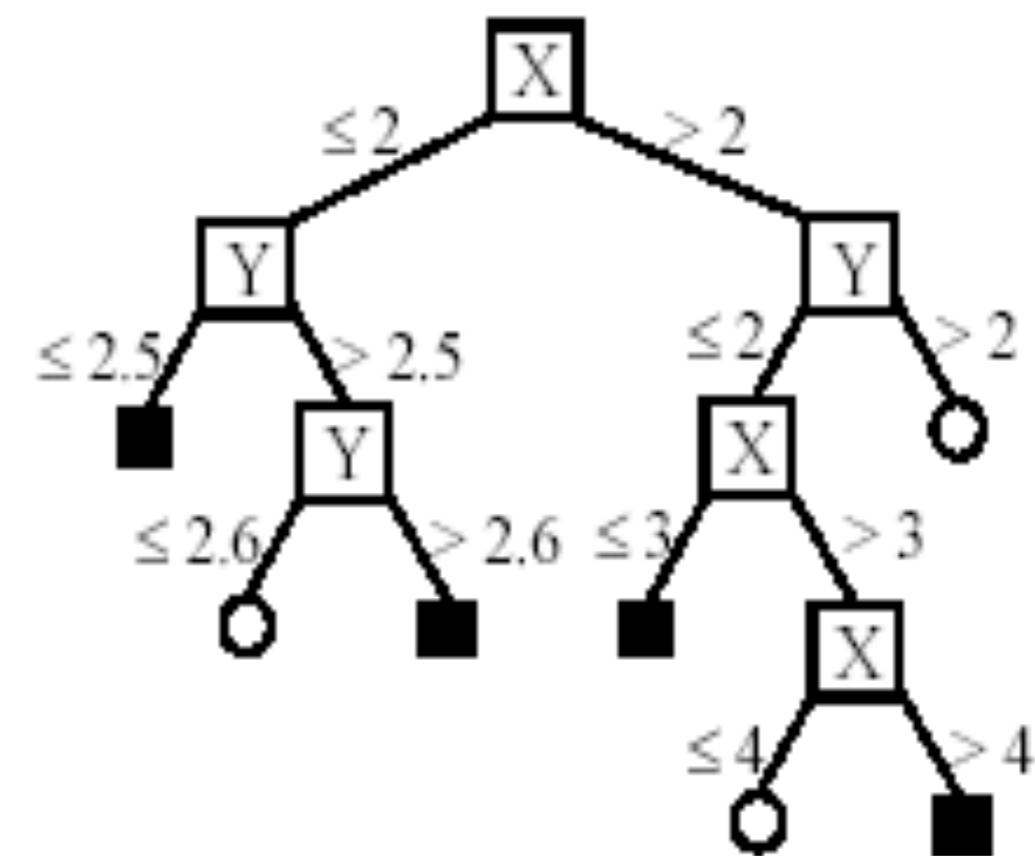
Handling continuous attributes

- Handle continuous attribute by splitting into two intervals (can be more) at each node.
- How to find the best threshold to divide?
 - Use information gain or gain ratio again
 - Sort all the values of an continuous attribute in increasing order $\{v_1, v_2, \dots, v_r\}$,
 - One possible threshold between two adjacent values v_i and v_{i+1} . Try all possible thresholds and find the one that maximizes the gain (or gain ratio).

An example in a continuous space



(A) A partition of the data space



(B). The decision tree

Avoid overfitting in classification

Overfitting: A tree may overfit the training data

Good accuracy on training data but poor on test data

Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers

Two approaches to avoid overfitting

Pre-pruning: Halt tree construction early

Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.

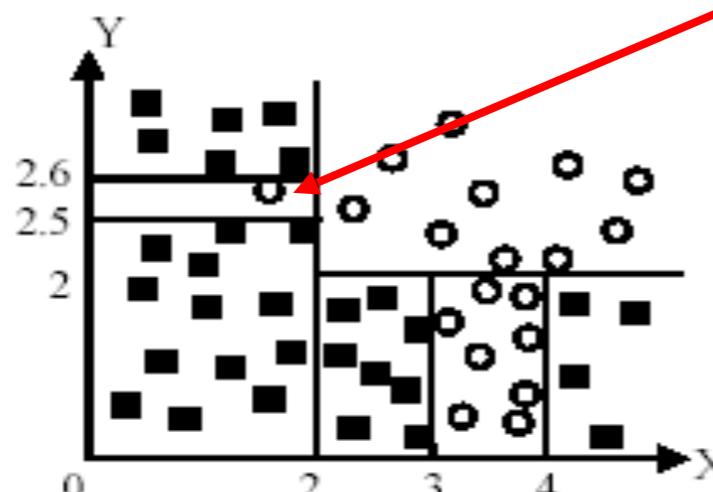
Post-pruning: Remove branches or sub-trees from a “fully grown” tree.

This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.

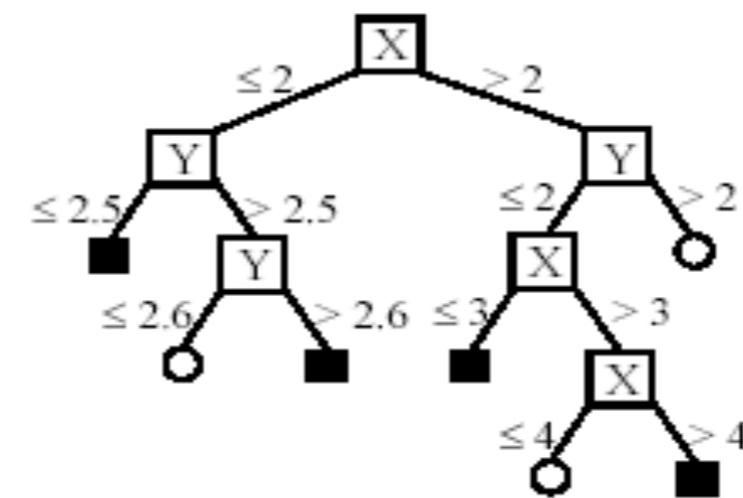
A validation set may be used for pruning as well.

An example

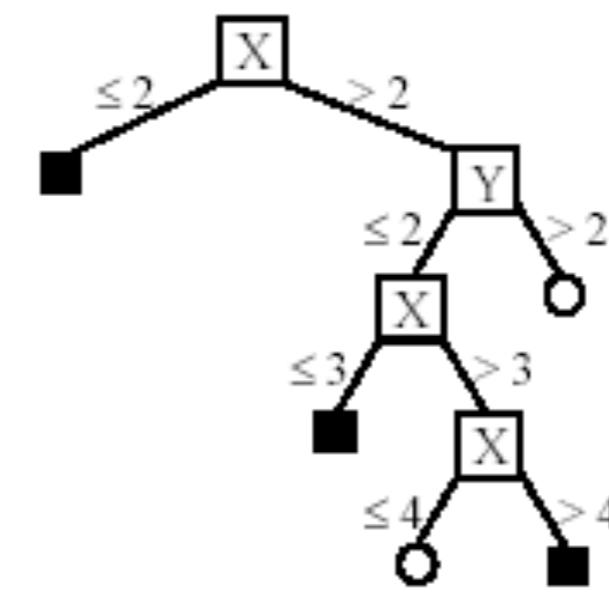
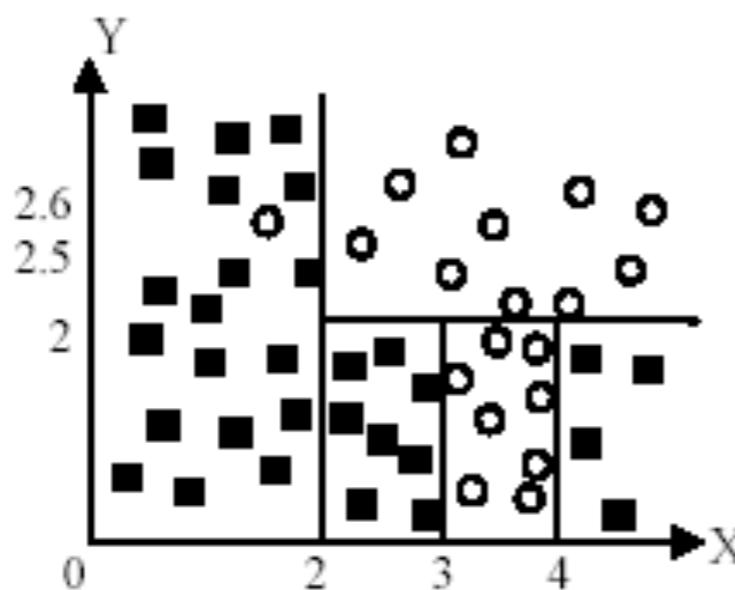
Likely to overfit the data



(A) A partition of the data space



(B). The decision tree



Other issues in decision tree learning

- From tree to rules, and rule pruning
- Handling of miss values
- Handing skewed distributions
- Handling attributes and classes with different costs.
- Attribute construction
- Etc.



Decision Tree in R

Fundamental Data Science for Data Scientist

Iris Data Preparation

```
> set.seed(1234)  
  
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))  
  
> trainData <- iris[ind==1,]  
  
> testData <- iris[ind==2,]
```

Decision Tree

Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

```
ctree(formula, data, subset = NULL, weights = NULL,  
      controls = ctree_control(), xtrafo = ptrtrafo, ytrafo = ptrtrafo,  
      scores = NULL)
```

Arguments

- formula** a symbolic description of the model to be fit. Note that symbols like : and - will not work and the tree will make use of all variables listed on the rhs of **formula**.
- data** a data frame containing the variables in the model.
- subset** an optional vector specifying a subset of observations to be used in the fitting process.
- weights** an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
- controls** an object of class [TreeControl](#), which can be obtained using [ctree_control](#).
- xtrafo** a function to be applied to all input variables. By default, the [ptrtrafo](#) function is applied.
- ytrafo** a function to be applied to all response variables. By default, the [ptrtrafo](#) function is applied.
- scores** an optional named list of scores to be attached to ordered factors.

Decision Tree - Create Model

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

	setosa	versicolor	virginica
setosa	40	0	0
versicolor	0	37	3
virginica	0	1	31

```
> print(iris_ctree)
```

Conditional inference tree with 4 terminal nodes

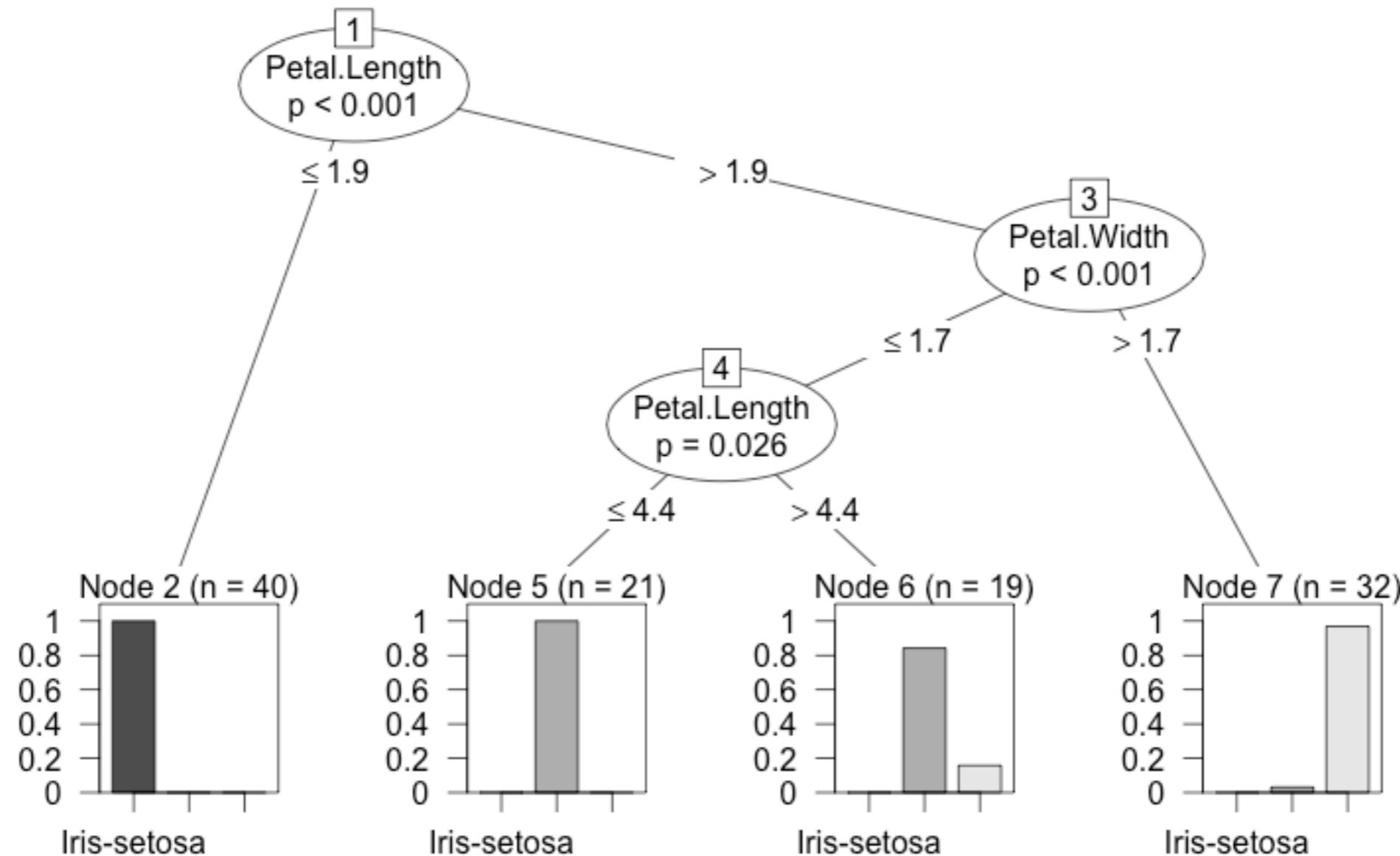
Response: Species

Inputs: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width

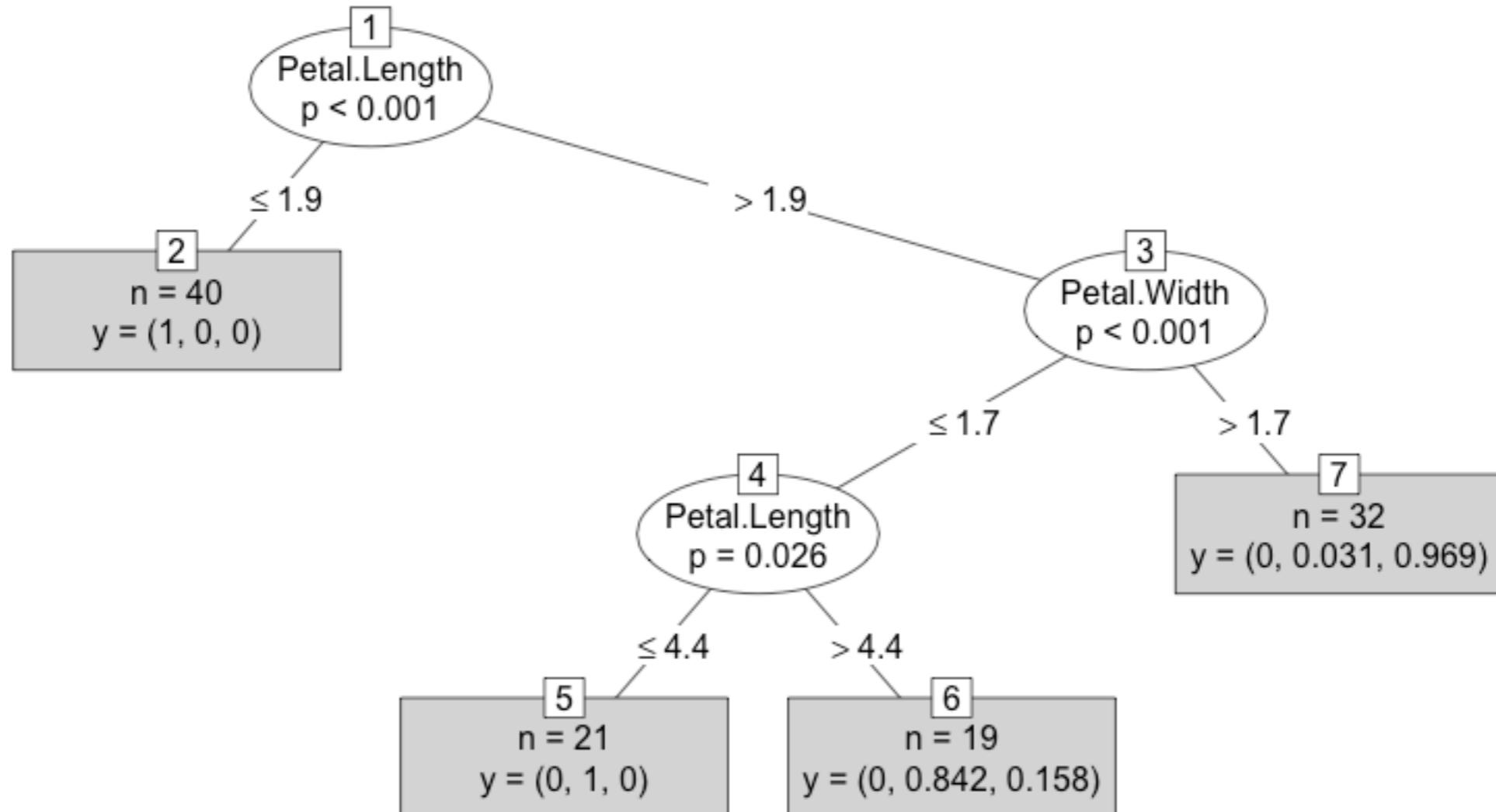
Number of observations: 112

- 1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
2)* weights = 40
- 1) Petal.Length > 1.9
 - 3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
 - 4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
5)* weights = 21
 - 4) Petal.Length > 4.4
6)* weights = 19
 - 3) Petal.Width > 1.7
7)* weights = 32

```
> plot(iris_ctree)
```



```
> plot(iris_ctree, type="simple")
```



Decision Tree - Prediction

```
> # predict on test data  
> testPred <- predict(iris_ctree, newdata = testData)  
> table(testPred, testData$Species)
```

testPred	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	12	2
virginica	0	0	14

Workshop 4.3 - Decision Tree

1. Use data from Workshop 2
2. Perform Decision Tree Algorithm to predict outcomes

Topic

- Basic concepts
- Decision Tree
- **Logistic Regression**
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Overview of Logistic Regression

A linear model for classification and probability estimation.

Can be very effective when:

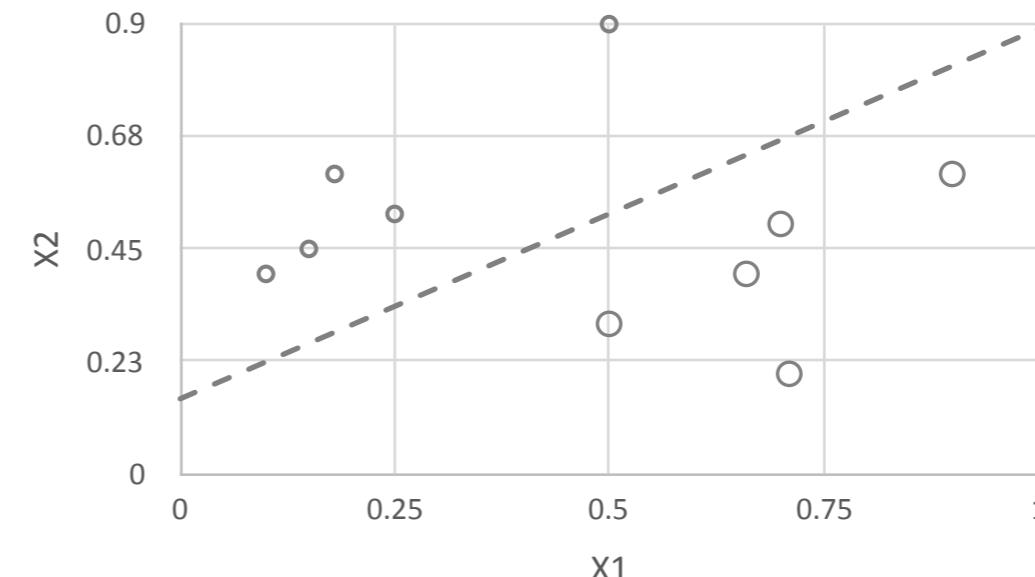
- The problem is linearly separable

- Or there are a lot of relevant features (10s - 100s of thousands can work)

- You need something simple and efficient as a baseline

- Efficient runtime

Logistic regression will generally not be the most accurate option.



Components of Learning Algorithm:

Model Structure –

Linear model with sigmoid activation

Loss Function –

Log Loss

Optimization Method –

Gradient Descent

Structure of Logistic Regression

Bias Weight Weight per Feature

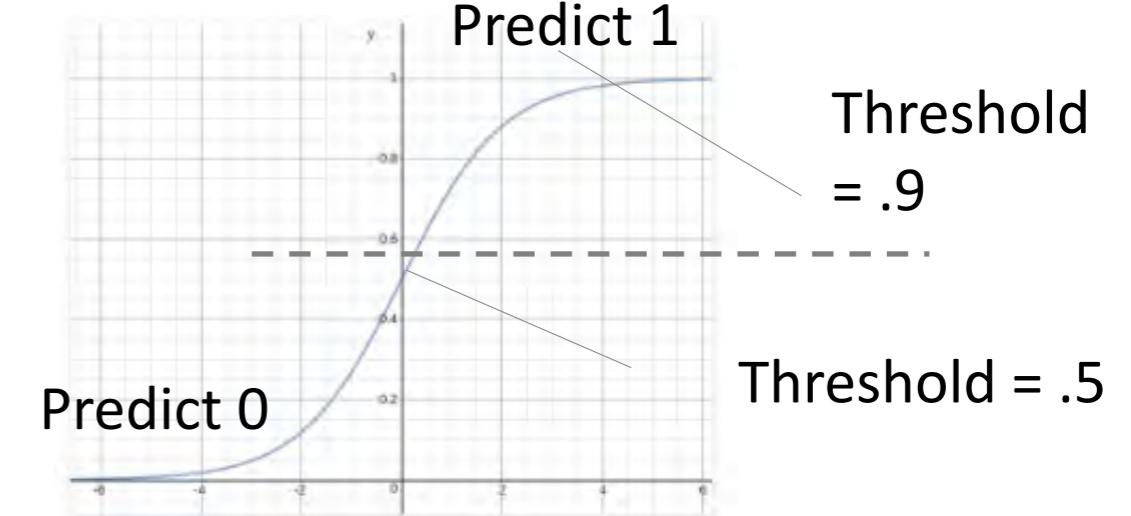
Linear Model: $w_0 + \sum_i^n w_i * x_i$

$$\hat{y} = sigmoid(w_0 + \sum_i^n w_i * x_i)$$

Example:

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

.25	-1	1



1	1
1	0
0	1
0	0

Intuition about additional dimensions

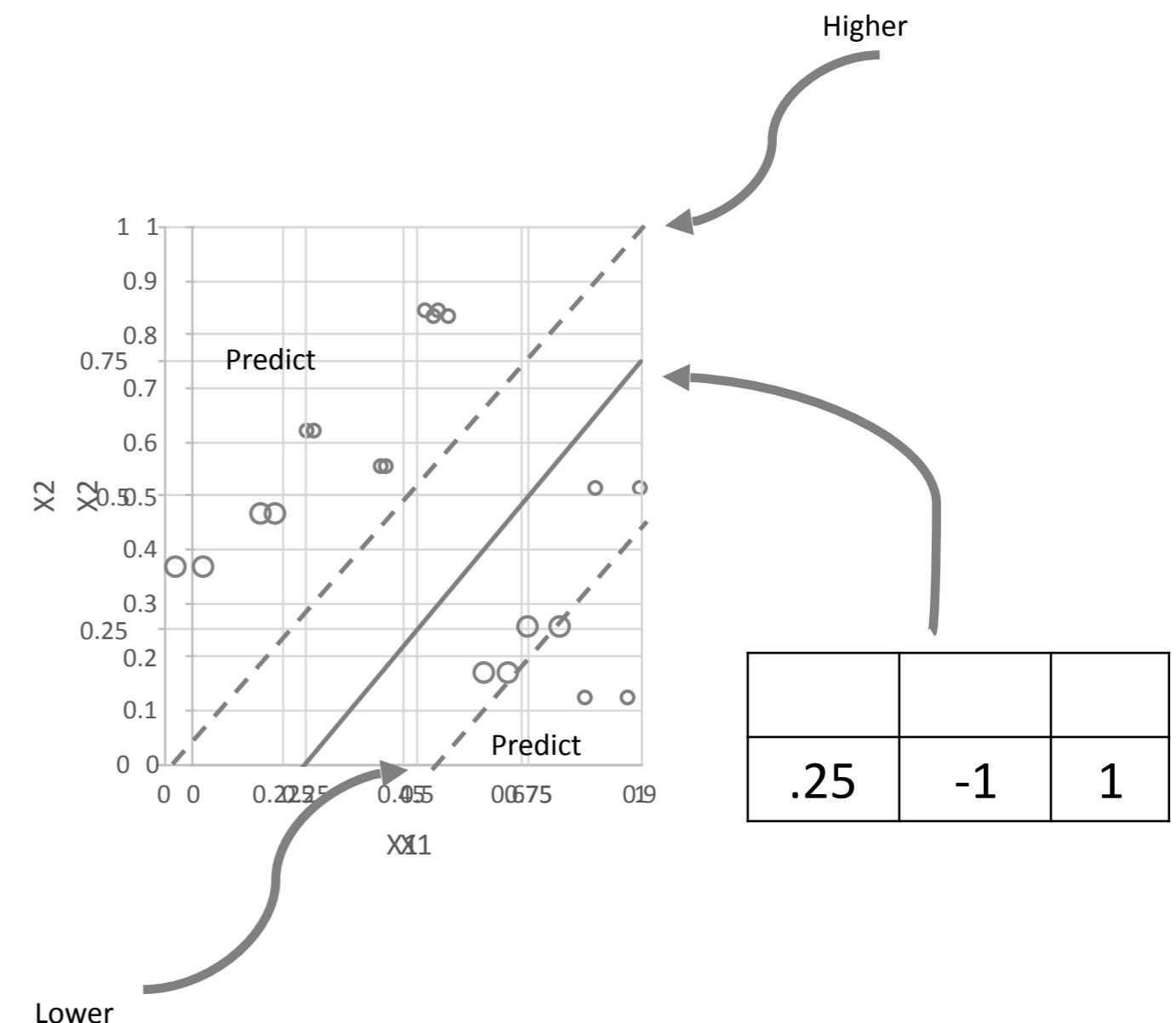
3 Dimensions

Decision surface is plane

N-Dimensions

Decision surface is n-dimensional hyper-plane

High-dimensions are weird
High-dimensional hyper-planes can represent quite a lot



Loss Function: Log Loss

y^{\wedge} -- The predicted y (pre-threshold)

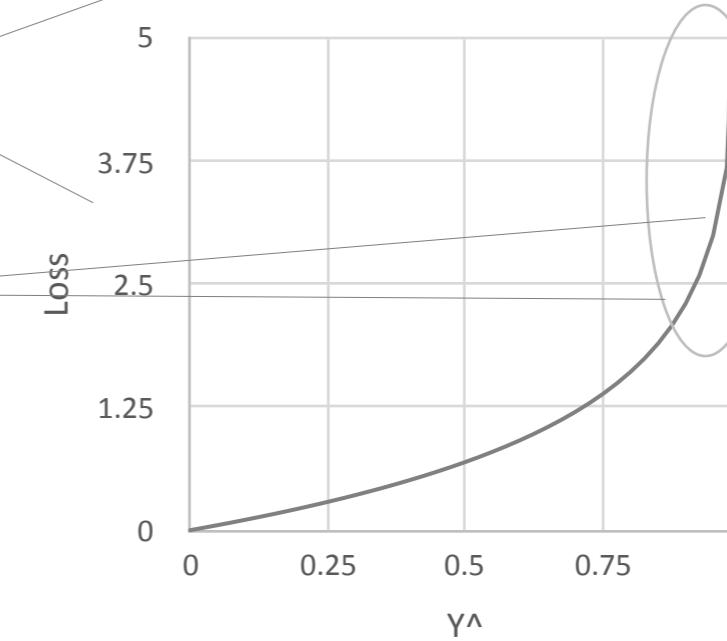
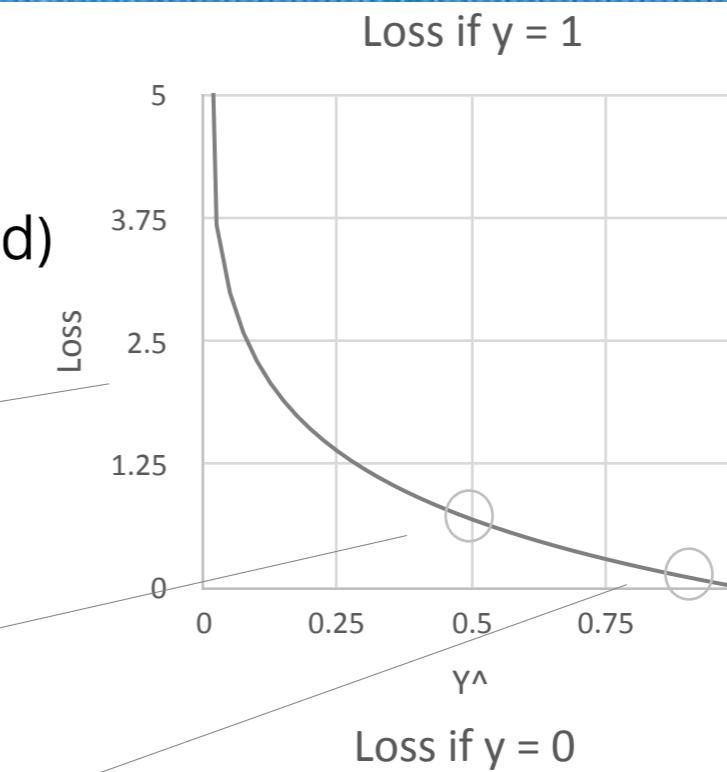
Log Loss:

Examples If y is 1: $-\log(y^{\wedge})$

If y is 0: $-\log(1 - y^{\wedge})$

		Loss
.9	1	.105
.5	1	.693
.9	0	2.3
.95	0	2.99

Use Natural Log (base e)



Logistic Regression Optimization

Initialize model weights to 0

Do 'numIterations' steps of gradient descent (thousands of steps)

- Find the gradient for each weight by averaging across the training set

- Update each weight by taking a step of size α opposite the gradient

Parameters

- α – size of the step to take in each iteration

- numIterations – number of iterations of gradient descent to perform

- Or use a convergence criteria...

- Threshold – value between 0-1 to convert \hat{y} into a classification



Logistic using R

```
library(nnet)  
model_multi = multinom(Species ~ ., data = iris_trn, trace = FALSE)  
summary(model_multi)$coefficients
```

```
##          (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width  
## versicolor    26.81602     -6.983313   -16.24574    20.35750    3.218787  
## virginica     -34.24228     -8.398869   -17.03985    31.94659   11.594518
```

```
head(predict(model_multi, newdata = iris_trn))
```

```
## [1] setosa      virginica  setosa      setosa      virginica  setosa  
## Levels: setosa versicolor virginica
```

```
head(predict(model_multi, newdata = iris_trn, type = "prob"))
```

```
##           setosa    versicolor    virginica  
## 23 1.000000e+00 2.607782e-19 3.891079e-44  
## 106 1.451651e-38 2.328295e-09 1.000000e+00
```

Workshop 4.4 - Logistic Regression

1. Use data from Workshop 4.2
2. Perform Logistic Regression Algorithm to predict outcomes

Evaluating classification methods

Predictive accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

Efficiency

time to construct the model

time to use the model

Robustness: handling noise and missing values

Scalability: efficiency in disk-resident databases

Interpretability:

understandable and insight provided by the model

Compactness of the model: size of the tree, or the number of rules.

Evaluation methods

- **Holdout set:** The available data set D is divided into two disjoint subsets,
 - the *training set* D_{train} (for learning a model)
 - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
 - Unseen test set provides a unbiased estimate of accuracy.
 - The test set is also called the **holdout set**. (the examples in the original data set D are all labeled with classes.)
 - This method is mainly used when the data set D is large.

Evaluation methods (cont...)

- **n-fold cross-validation**: The available data is partitioned into n equal-size disjoint subsets.
- Use each subset as the test set and combine the rest $n-1$ subsets as the training set to learn a classifier.
- The procedure is run n times, which give n accuracies.
- The final estimated accuracy of learning is the average of the n accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont...)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only **a single test example** and all the rest of the data is used in training.
- If the original data has m examples, this is **m -fold cross-validation**

Evaluation methods (cont...)

- **Validation set:** the available data is divided into three subsets,
 - a training set,
 - a validation set and
 - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure ($\text{error} = 1 - \text{accuracy}$).
- **Accuracy is not suitable in some applications.**
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, **we are interested only in the minority class.**
 - High accuracy does not mean any intrusion is detected.
 - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Precision and recall measures (cont...)

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP}.$$

$$r = \frac{TP}{TP + FN}.$$

- **Precision p** is the number of **correctly classified positive examples** divided by the total number of examples that are classified as positive.
- **Recall r** is the number of **correctly classified positive examples** divided by the total number of actual positive examples in the test set.

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- ▶ This confusion matrix gives
 - precision $p = 100\%$ and
 - recall $r = 1\%$
- ▶ because we only classified one positive example correctly and no negative examples wrongly.
- ▶ Note: precision and recall only measure classification on the positive class.

F_1 -value (also called F_1 -score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p+r}$$

F_1 -score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1 -value to be large, both p and r must be large.

Receive operating characteristics curve

- It is commonly called the **ROC curve**.
- It is a plot of the **true positive rate (TPR)** against the **false positive rate (FPR)**.
- **True positive rate:**

$$TPR = \frac{TP}{TP + FN}$$

- **False positive rate:**

$$FPR = \frac{FP}{TN + FP}$$

Sensitivity and Specificity

- In statistics, there are two other evaluation measures:

- **Sensitivity**: Same as TPR
- **Specificity**: Also called **True Negative Rate (TNR)**

$$TNR = \frac{TN}{TN + FP}$$

- Then we have

$$FPR = 1 - specificity$$

Example ROC curves

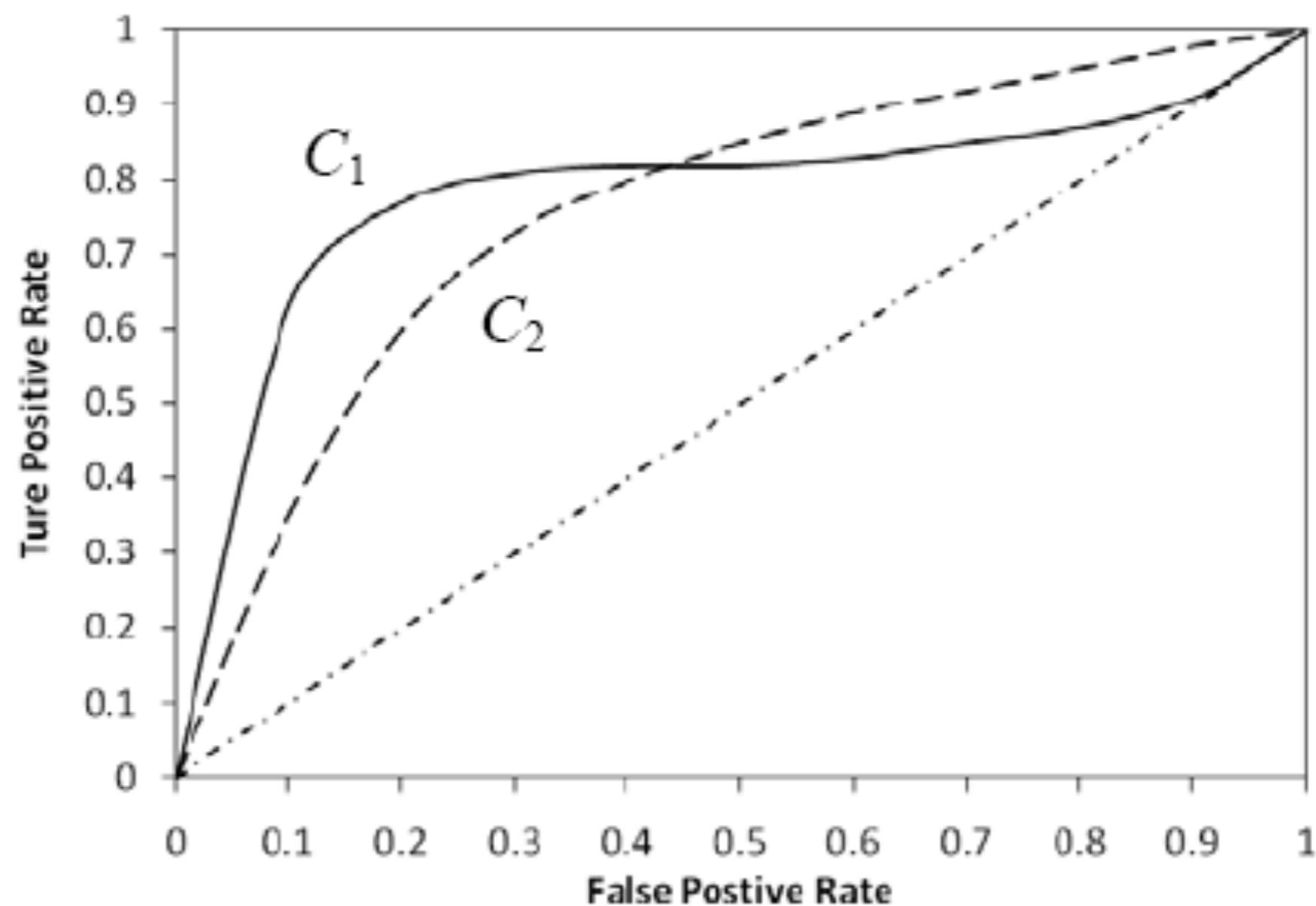


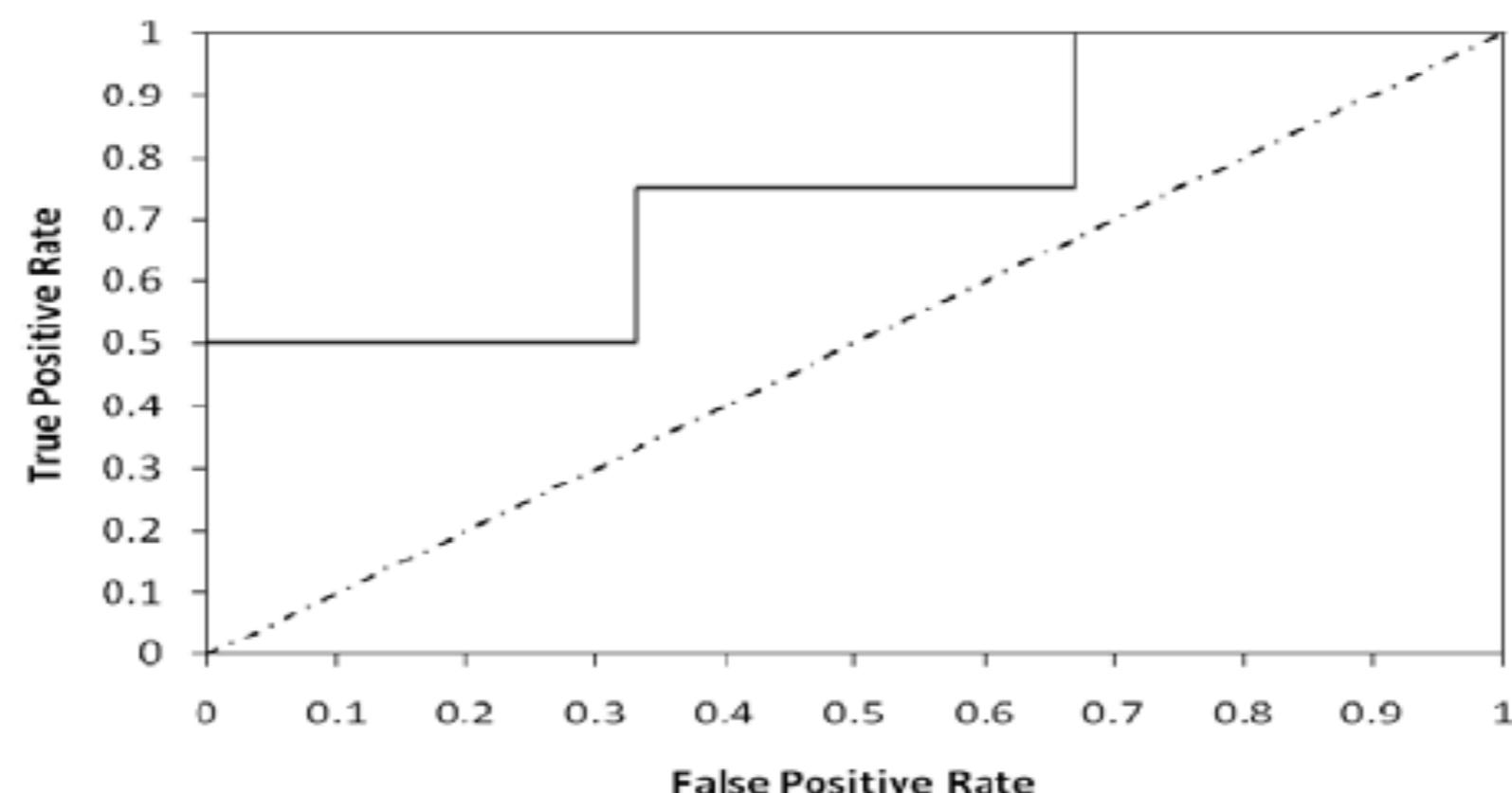
Fig. 3.8. ROC curves for two classifiers (C_1 and C_2) on the same data

Area under the curve (AUC)

- Which classifier is better, C_1 or C_2 ?
 - It depends on which region you talk about.
- Can we have one measure?
 - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j , it is said that C_i is better than C_j .
 - If a classifier is perfect, its AUC value is 1
 - If a classifier makes all random guesses, its AUC value is 0.5.

Drawing an ROC curve

Rank	1	2	3	4	5	6	7	8	9	10
Actual class	+	+	-	-	+	-	-	+	-	-
TP	1	2	2	2	3	3	3	4	4	4
FP	0	0	0	1	2	2	3	4	4	5
TN	6	6	6	5	4	4	3	2	1	0
FN	4	3	2	2	2	1	1	0	0	0
TPR	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83



Another evaluation method: Scoring and ranking

- **Scoring** is related to classification.
- We are interested in a single class (**positive class**), e.g., buyers class in a marketing database.
- Instead of assigning each test instance a definite class, scoring assigns a probability estimate (PE) to indicate the likelihood that the example belongs to the positive class.

Ranking and lift analysis

- After each example is given a PE score, we can rank all examples according to their PEs.
- We then divide the data into n (say 10) bins. A lift curve can be drawn according how many positive examples are in each bin. This is called **lift analysis**.
- Classification systems can be used for scoring. Need to produce a probability estimate.
 - E.g., in decision trees, we can use the confidence value at each leaf node as the score.

An example

- We want to send promotion materials to potential customers to sell a watch.
- Each package cost \$0.50 to send (material and postage).
- If a watch is sold, we make \$5 profit.
- Suppose we have a large amount of past data for building a predictive/classification model. We also have a large list of potential customers.
- How many packages should we send and who should we send to?

An example

Assume that the test set has 10000 instances. Out of this, 500 are positive cases.

After the classifier is built, we score each test instance. We then rank the test set, and divide the ranked test set into 10 bins.

Each bin has 1000 test instances.

Bin 1 has 210 actual positive instances

Bin 2 has 120 actual positive instances

Bin 3 has 60 actual positive instances

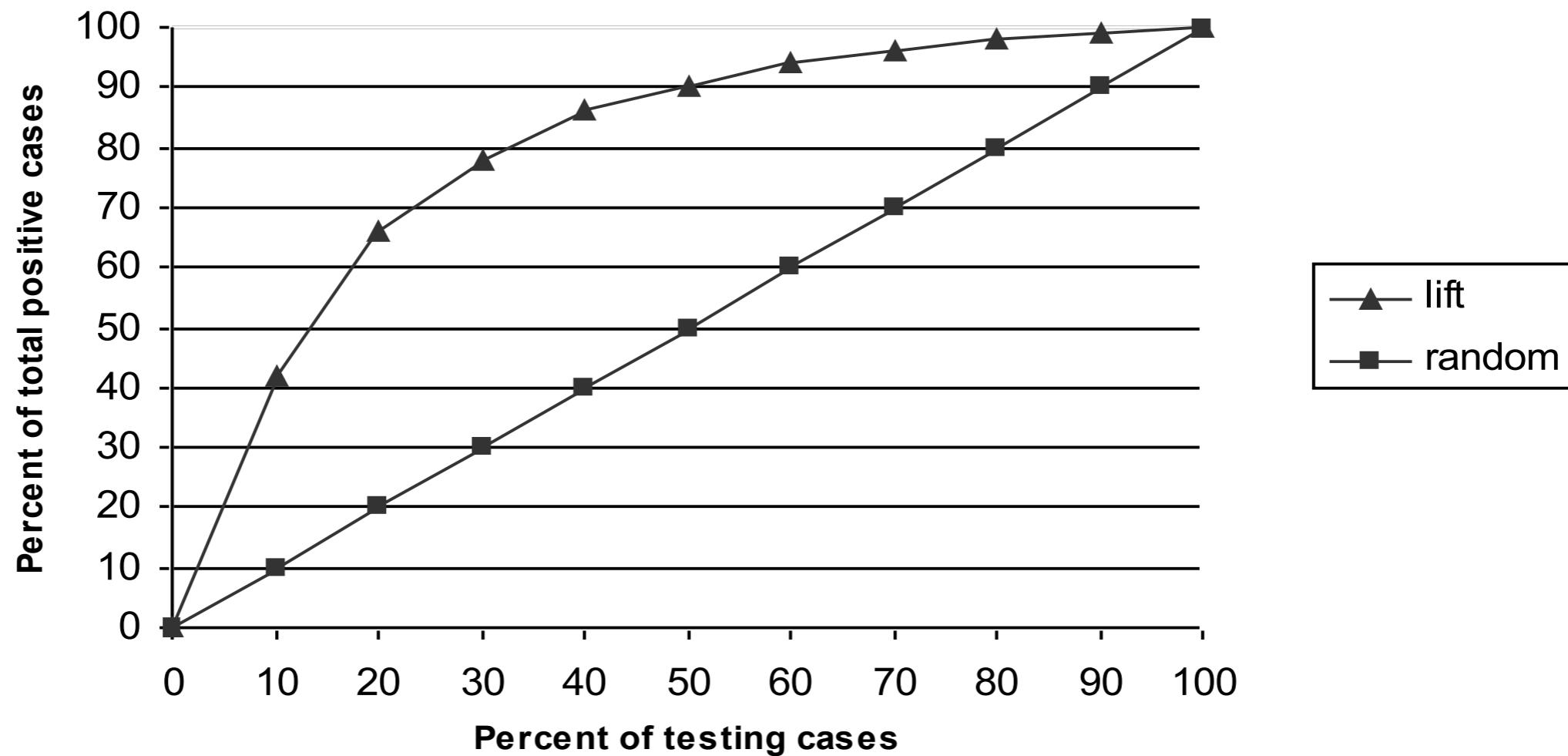
...

Bin 10 has 5 actual positive instances

Lift curve

Bin 1 2 3 4 5 6 7 8 9 10

210	120	60	40	22	18	12	7	6	5
42%	24%	12%	8%	4.40%	3.60%	2.40%	1.40%	1.20%	1%
42%	66%	78%	86%	90.40%	94%	96.40%	97.80%	99%	100%



Evaluation of Classifier in R

Fundamental Data Science for Data Scientist

Confusion Matrix

```
install.packages("caret")  
library(caret)  
confusionMatrix(testPred, testdata$Species)
```

Confusion Matrix and Statistics			
Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	2
Iris-virginica	0	0	13
Overall Statistics			
Accuracy : 0.9535			
95% CI : (0.8419, 0.9943)			
No Information Rate : 0.3488			
P-Value [Acc > NIR] : < 2.2e-16			
Kappa : 0.9303			
McNemar's Test P-Value : NA			

Statistics by Class:

	Class: Iris-setosa	Class: Iris-versicolor	Class: Iris-virginica
Sensitivity	1.0000	1.0000	0.8667
Specificity	1.0000	0.9333	1.0000
Pos Pred Value	1.0000	0.8667	1.0000
Neg Pred Value	1.0000	1.0000	0.9333
Prevalence	0.3488	0.3023	0.3488
Detection Rate	0.3488	0.3023	0.3023
Detection Prevalence	0.3488	0.3488	0.3023
Balanced Accuracy	1.0000	0.9667	0.9333

Creating ROC

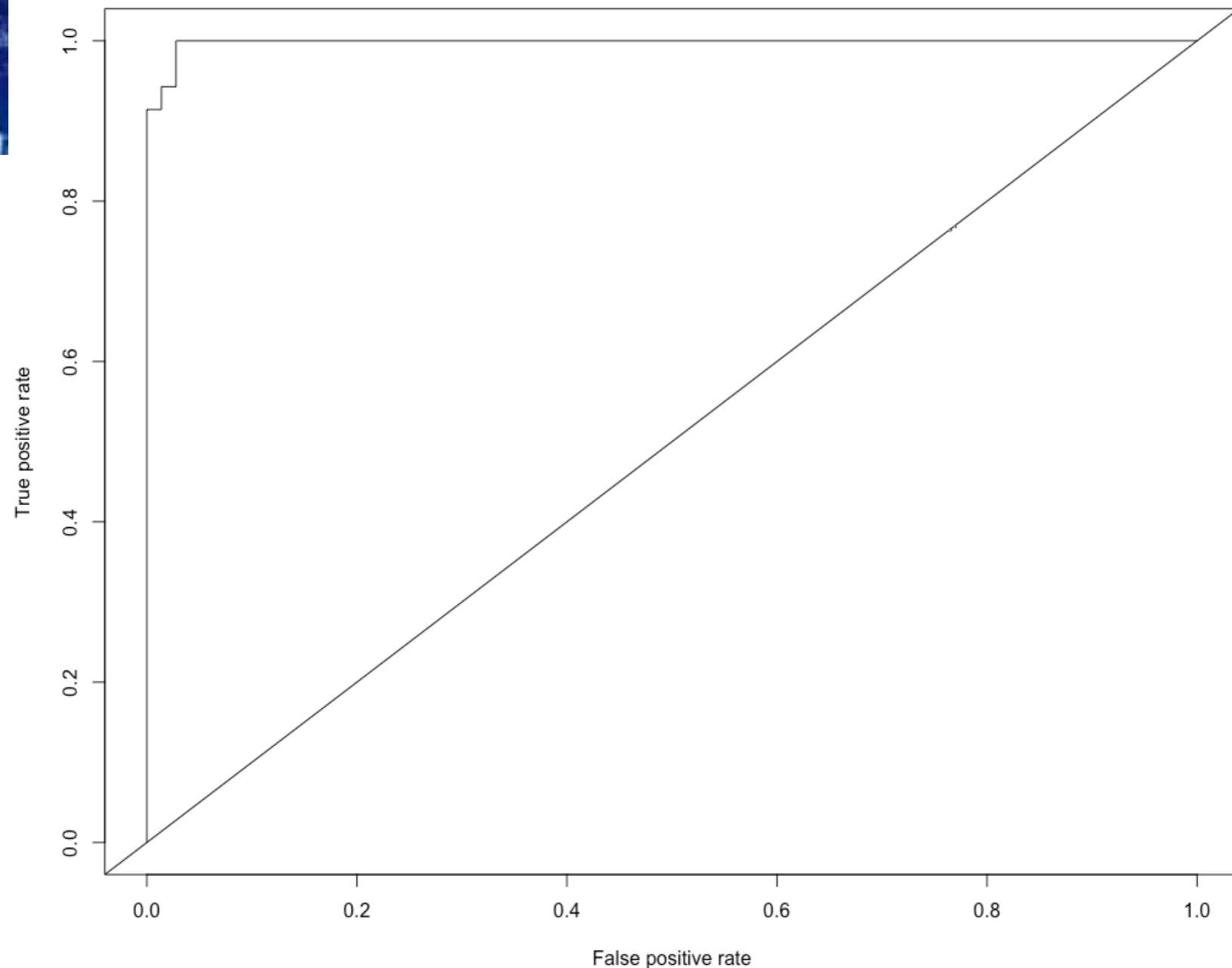
```
library(ROCR)
df <- trainData[,1:4]
df$virgi <- as.numeric(trainData$Species=="Iris-virginica")
svmMod <- svm(virgi ~ ., data = df)

svmPred <- predict(svmMod, df[,-5])

svmPredictiction <- prediction(svmPred, df$virgi)
svmPerf <- performance(svmPredictiction, "tpr", "fpr")

plot(svmPerf)
abline(a=0, b= 1)

library(pROC)
predictions <- as.numeric(predict(svmfit, testData, type = "response"))
multiclass.roc(testData$Species,predictions)
```



```
> multiclass.roc(testData$Species,predictions)
```

Call:

```
multiclass.roc.default(response = testData$Species, predictor = predictions)
```

Data: predictions with 3 levels of testData\$Species: Iris-setosa, Iris-versicolor, Iris-virginica.

Multi-class area under the curve: 1

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- **Naïve Bayesian classification**
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Bayesian classification

Probabilistic view: Supervised learning can naturally be studied from a probabilistic point of view.

Let A_1 through A_k be attributes with discrete values. The class is C .

Given a test example d with observed attribute values a_1 through a_k .

Classification is basically to compute the following posteriori probability.

The prediction is the class c_j such that

$$\Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|})$$

is maximal

Apply Bayes' Rule

$$\Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|})$$
$$= \frac{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_j) \Pr(C = c_j)}{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|})}$$
$$= \frac{\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_j) \Pr(C = c_j)}{\sum_{r=1}^{|C|} \Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_r) \Pr(C = c_r)}$$

- $\Pr(C=c_j)$ is the class prior probability: easy to estimate from the training data.

Computing probabilities

The denominator $P(A_1=a_1, \dots, A_k=a_k)$ is irrelevant for decision making since it is the same for every class.

We only need $P(A_1=a_1, \dots, A_k=a_k | C=c_i)$, which can be written as

$$Pr(A_1=a_1 | A_2=a_2, \dots, A_k=a_k, C=c_j) * Pr(A_2=a_2, \dots, A_k=a_k | C=c_j)$$

Recursively, the second factor above can be written in the same way, and so on.

Now an assumption is needed.

Conditional independence assumption

All attributes are conditionally independent given the class $C = c_j$.

Formally, we assume,

$$\Pr(A_1=a_1 \mid A_2=a_2, \dots, A_{|A|}=a_{|A|}, C=c_j) = \Pr(A_1=a_1 \mid C=c_j)$$

and so on for A_2 through $A_{|A|}$. I.e.,

$$\Pr(A_1 = a_1, \dots, A_{|A|} = a_{|A|} \mid C = c_i) = \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

Final naïve Bayesian classifier

$$\Pr(C = c_j \mid A_1 = a_1, \dots, A_{|A|} = a_{|A|}) \\ = \frac{\Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)}{\sum_{r=1}^{|C|} \Pr(C = c_r) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_r)}$$

Classify a test instance

If we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class. Thus, given a test example, we compute the following to decide the most probable class for the test instance

$$c = \arg \max_{c_j} \Pr(c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

An example

- Compute all probabilities required for classification

A	B	C
m	b	t
m	s	t
g	q	t
h	s	t
g	q	t
g	q	f
g	s	f
h	b	f
h	q	f
m	b	f

$$\Pr(C = t) = 1/2,$$

$$\Pr(C = f) = 1/2$$

$$\Pr(A=m \mid C=t) = 2/5$$

$$\Pr(A=g \mid C=t) = 2/5$$

$$\Pr(A=h \mid C=t) = 1/5$$

$$\Pr(A=m \mid C=f) = 1/5$$

$$\Pr(A=g \mid C=f) = 2/5$$

$$\Pr(A=h \mid C=f) = 2/5$$

$$\Pr(B=b \mid C=t) = 1/5$$

$$\Pr(B=s \mid C=t) = 2/5$$

$$\Pr(B=q \mid C=t) = 2/5$$

$$\Pr(B=b \mid C=f) = 2/5$$

$$\Pr(B=s \mid C=f) = 1/5$$

$$\Pr(B=q \mid C=f) = 2/5$$

Now we have a test example:

$$A = m \quad B = q \quad C = ?$$

An Example (cont ...)

For $C = t$, we have

$$\Pr(C = t) \prod_{j=1}^2 \Pr(A_j = a_j | C = t) = \frac{1}{2} \times \frac{2}{5} \times \frac{2}{5} = \frac{2}{25}$$

For class $C = f$, we have

$$\Pr(C = f) \prod_{j=1}^2 \Pr(A_j = a_j | C = f) = \frac{1}{2} \times \frac{1}{5} \times \frac{2}{5} = \frac{1}{25}$$

$C = t$ is more probable. t is the final class.

Additional issues

Numeric attributes: Naïve Bayesian learning assumes that all attributes are categorical. Numeric attributes need to be discretized.

Zero counts: An particular attribute value never occurs together with a class in the training set. We need smoothing.

Missing values: Ignored

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda n_i}$$

On naïve Bayesian classifier

Advantages:

Easy to implement

Very efficient

Good results obtained in many applications

Disadvantages

Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)



naïve Bayesian classifier in R

Data Science Certification

Naive Bayes

Naive Bayes Classifier

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula'  
naiveBayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)  
## Default S3 method:  
naiveBayes(x, y, laplace = 0, ...)  
  
## S3 method for class 'naiveBayes'  
predict(object, newdata,  
       type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```

Naive Bayes

Arguments

- x** A numeric matrix, or a data frame of categorical and/or numeric variables.
- y** Class vector.
- formula** A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
- data** Either a data frame of predictors (categorical and/or numeric) or a contingency table.
- laplace** positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
- ...** Currently not used.
- subset** For data given in a data frame, an index vector specifying the cases to be used in the training sample.
(NOTE: If given, this argument must be named.)
- na.action** A function to specify the action to be taken if NAs are found. The default action is not to count them for the computation of the probability factors. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
- object** An object of class "naiveBayes".
- newdata** A data frame with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
- type** If "raw", the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
- threshold** Value replacing cells with probabilities within `eps` range.
- eps** double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `threshold`.)

Naive Bayes

```
> library(e1071)
> # Can handle both categorical and numeric input,
> # but output must be categorical
> model <- naiveBayes(Species~., data=iristrain)
> prediction <- predict(model, iristest[, -5])
> table(prediction, iristest[, 5])
```

prediction	setosa	versicolor	virginica
setosa	10	0	0
versicolor	0	10	2
virginica	0	0	8

```
> confusionMatrix(prediction, testdata$Species)
```

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	2
Iris-virginica	0	0	13

Overall Statistics

Accuracy : 0.9535

95% CI : (0.8419, 0.9943)

No Information Rate : 0.3488

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9303

Mcnemar's Test P-Value : NA

Statistics by Class:

	Class: Iris-setosa	Class: Iris-versicolor	Class: Iris-virginica
Sensitivity	1.0000	1.0000	0.8667
Specificity	1.0000	0.9333	1.0000
Pos Pred Value	1.0000	0.8667	1.0000
Neg Pred Value	1.0000	1.0000	0.9333
Prevalence	0.3488	0.3023	0.3488
Detection Rate	0.3488	0.3023	0.3023
Detection Prevalence	0.3488	0.3488	0.3023
Balanced Accuracy	1.0000	0.9667	0.9333

Workshop 4.5 - Naive Bayes Classifier

1. Use data from Workshop 4.2
2. Perform Naive Bayes Classifier Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- **K-nearest neighbor**
- Support Vector Machine
- Neural Net
- Ensemble methods: Bagging and Boosting

k-Nearest Neighbor Classification (kNN)

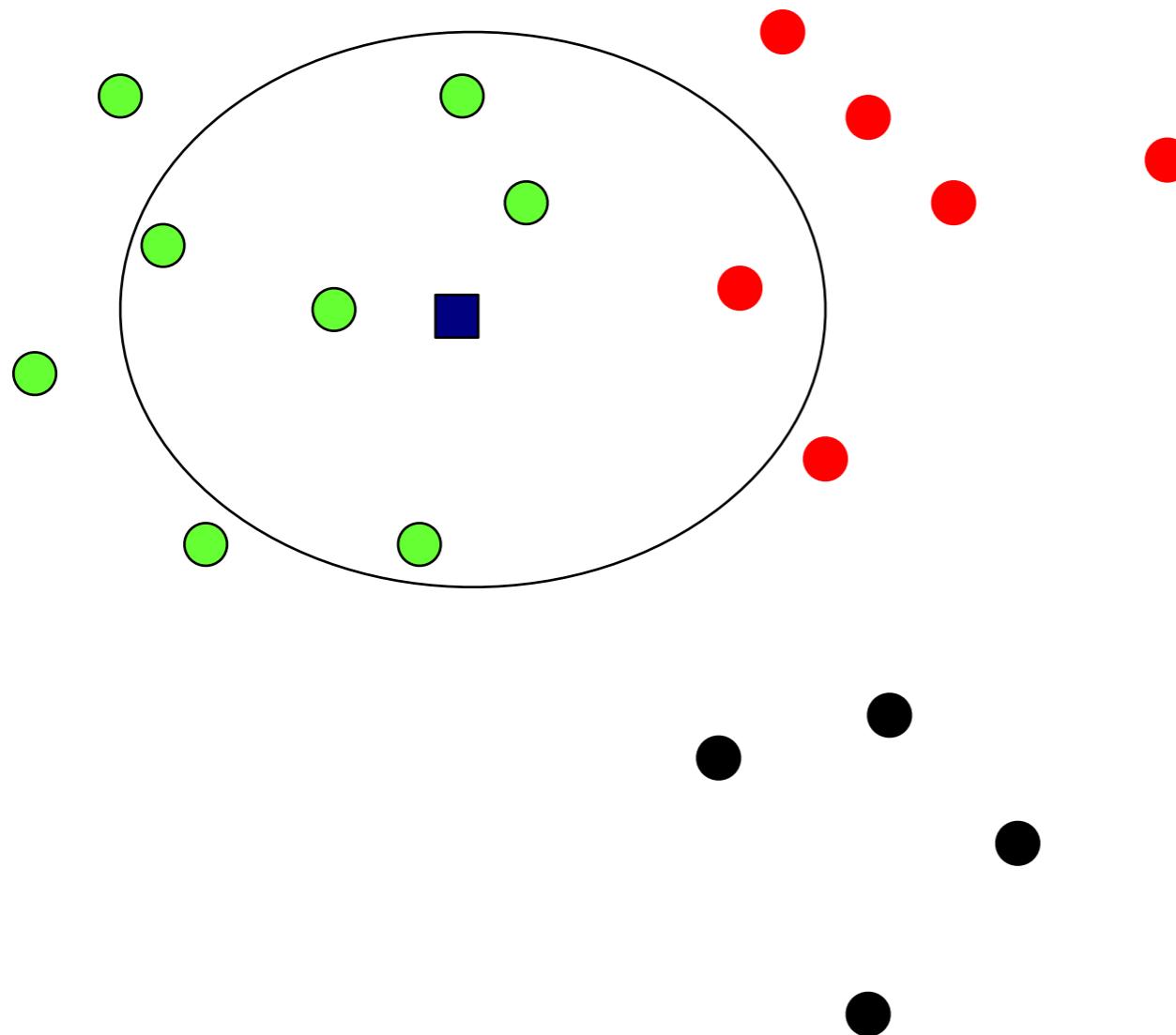
- Unlike all the previous learning methods, kNN does not build model from the training data.
- To classify a test instance d , define k -neighborhood P as k nearest neighbors of d
- Count number n of training instances in P that belong to class c_j
- Estimate $\Pr(c_j|d)$ as n/k
- No training is needed. Classification time is linear in training set size for each test case.

kNNAlgorithm

Algorithm $\text{kNN}(D, d, k)$

- 1 Compute the distance between d and every example in D ;
 - 2 Choose the k examples in D that are nearest to d , denote the set by P ($\subseteq D$);
 - 3 Assign d the class that is the most frequent class in P (or the majority class);
- k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.
 - *Distance function* is crucial, but depends on applications.

Example: k=6 (6NN)



- Government
- Science
- Arts

A new point ■
Pr(science) ■?

Discussions

- kNN can deal with complex and arbitrary decision boundaries.
- Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong and in many cases as accurate as those elaborated methods.
- kNN is slow at the classification time
- kNN does not produce an understandable model



k-NN in R

Data Science Certification

K-NN

k-Nearest Neighbour Classification

Description

k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

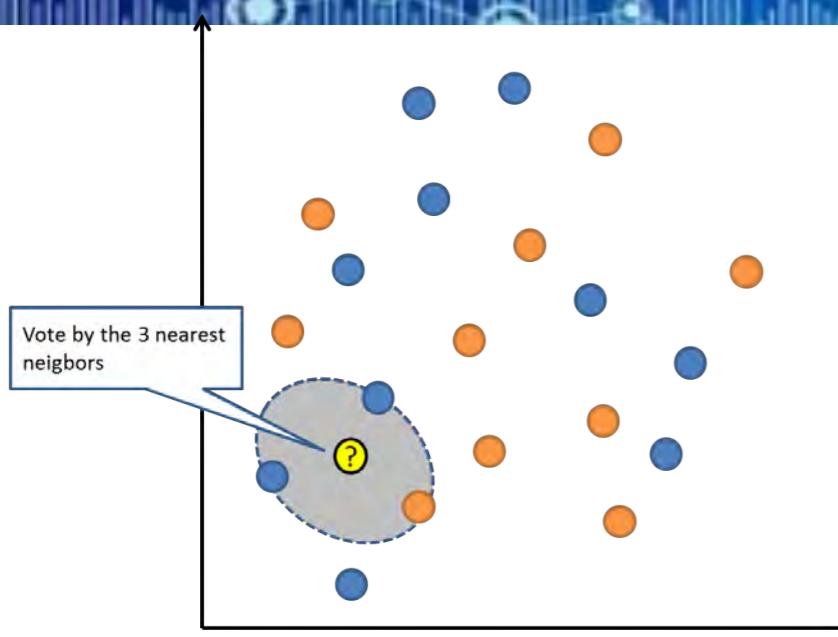
Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

- train** matrix or data frame of training set cases.
- test** matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
- cl** factor of true classifications of training set
- k** number of neighbours considered.
- l** minimum vote for definite decision, otherwise doubt. (More precisely, less than k-1 dissenting votes are allowed, even if k is increased by ties.)
- prob** If this is true, the proportion of the votes for the winning class are returned as attribute **prob**.
- use.all** controls handling of ties. If true, all distances equal to the kth largest are included. If false, a random selection of distances equal to the kth is chosen to use exactly k neighbours.

K-NN



```
> library(class)
> train_input <- as.matrix(iristrain[,-5])
> train_output <- as.vector(irisstrain[,5])
> test_input <- as.matrix(iristest[,-5])
> prediction <- knn(train_input, test_input,
+                      train_output, k=5)
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa        10         0         0
  versicolor     0        10         1
  virginica      0         0         9
>
```

```
> confusionMatrix(prediction, testdata$Species)
```

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	12	0
Iris-virginica	0	1	15

Overall Statistics

Accuracy : 0.9767
95% CI : (0.8771, 0.9994)

No Information Rate : 0.3488

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.965

McNemar's Test P-Value : NA

Statistics by Class:

	myFormula		
	Class: Iris-setosa	Class: Iris-versicolor	Class: Iris-virginica
Sensitivity	1.0000	0.9231	1.0000
Specificity	1.0000	1.0000	0.9643
Pos Pred Value	1.0000	1.0000	0.9375
Neg Pred Value	1.0000	0.9677	1.0000
Prevalence	0.3488	0.3023	0.3488
Detection Rate	0.3488	0.2791	0.3488
Detection Prevalence	0.3488	0.2791	0.3721
Balanced Accuracy	1.0000	0.9615	0.9821



Workshop 4.6 - k-NN

1. Use data from Workshop 4.2
2. Perform k-NN Algorithm to predict outcomes
3. Perform performance evaluation by creating confusion matrix

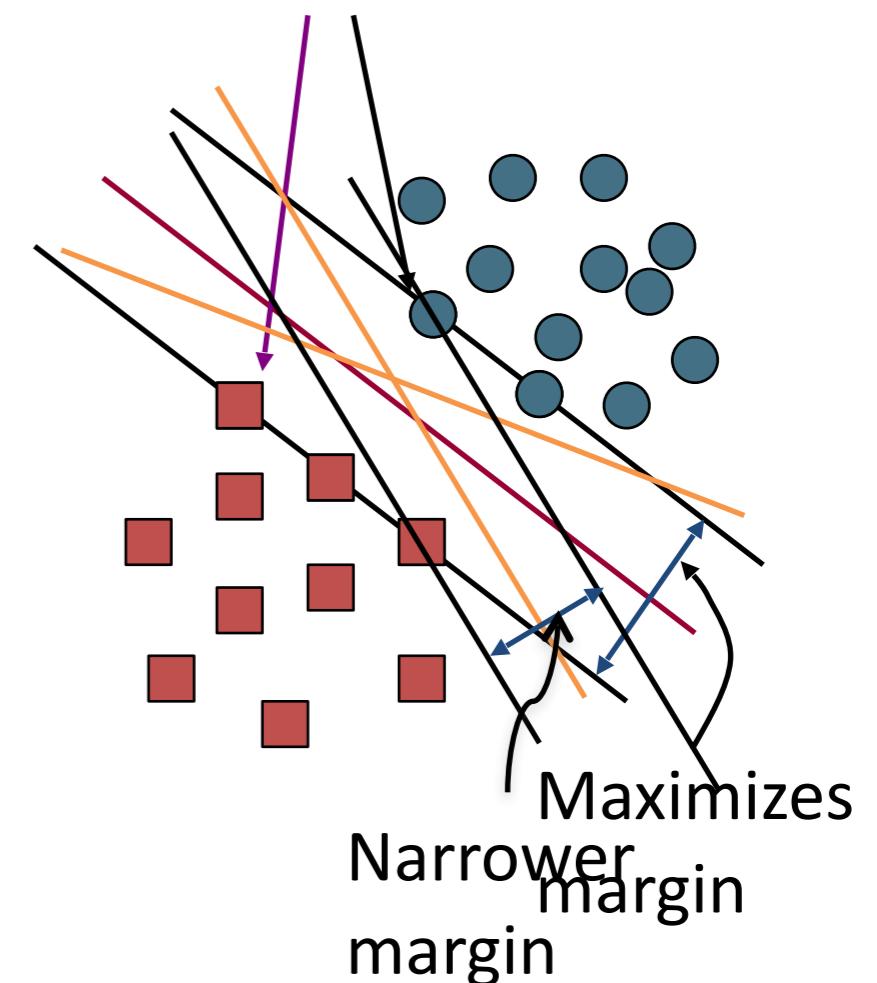
Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- **Support Vector Machine**
- Neural Net
- Ensemble methods: Bagging and Boosting
- Summary

Support Vector Machine (SVM)

- SVMs maximize the *margin* around the separating hyperplane.
 - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem
- Seen by many as the most successful current text classification method*

Support vectors



Maximum Margin: Formalization

\mathbf{w} : decision hyperplane normal vector

\mathbf{x}_i : data point i

y_i : class of data point i (+1 or -1) NB: Not 1/0

Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^\top \mathbf{x}_i + b)$

Functional margin of \mathbf{x}_i is: $y_i (\mathbf{w}^\top \mathbf{x}_i + b)$

But note that we can increase this margin simply by scaling \mathbf{w}, \mathbf{b}

Functional margin of dataset is twice the minimum functional margin for any point

The factor of 2 comes from measuring the whole width of the margin

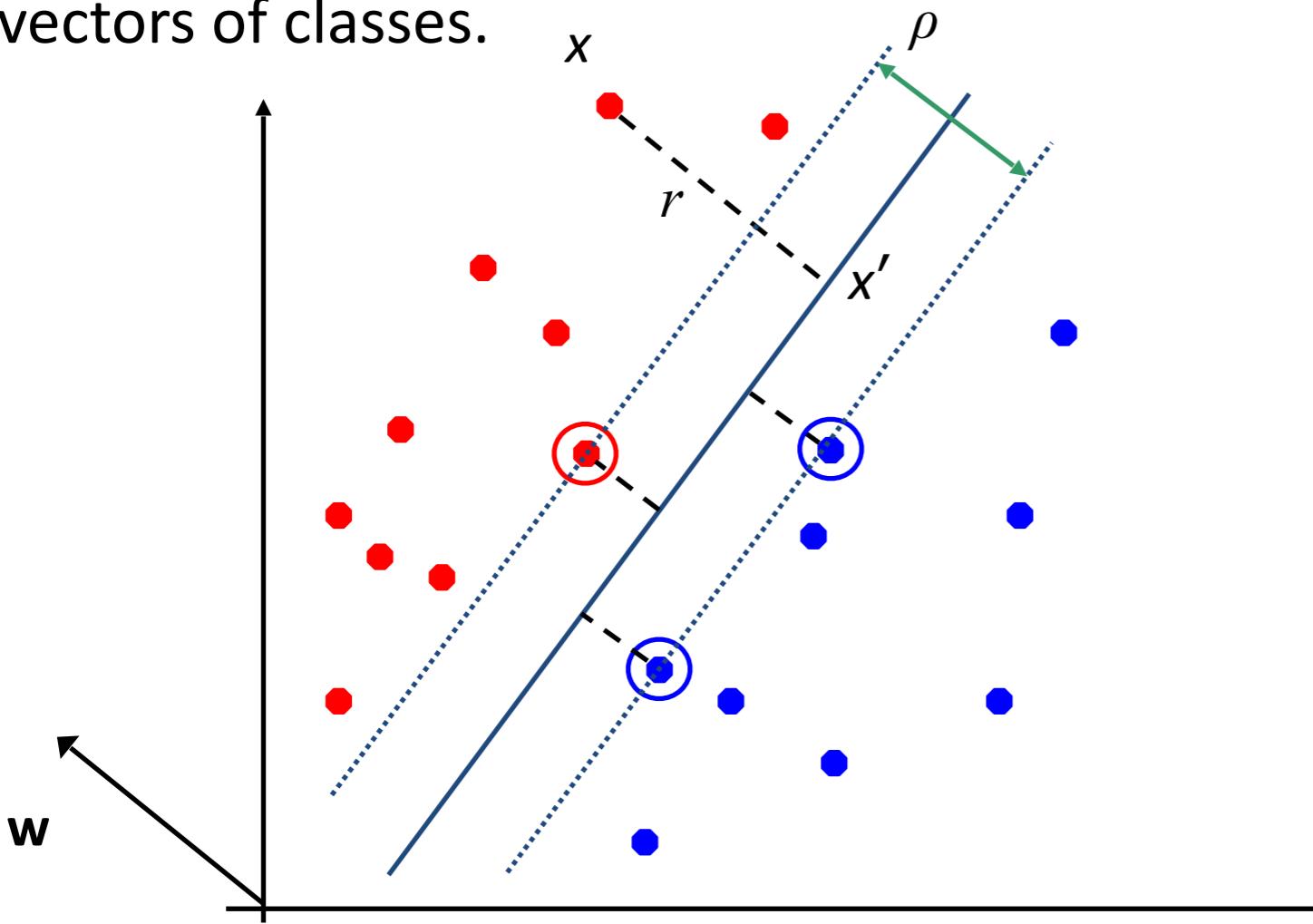
Geometric Margin

Distance from example to the separator is

$$r = \frac{y \mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|}$$

Examples closest to the hyperplane are **support vectors**.

Margin ρ of the separator is the width of separation between support vectors of classes.



Derivation of finding r :

Dotted line $x' - x$ is perpendicular to decision boundary so parallel to w . Unit vector is $w / \|w\|$, so line is $rw / \|w\|$.

$$x' = x - yrw / \|w\|.$$

$$x' \text{ satisfies } \mathbf{w}^T \mathbf{x}' + b = 0.$$

$$\text{So } \mathbf{w}^T(x - yrw / \|w\|) + b = 0$$

$$\text{Recall that } \|w\| = \sqrt{\mathbf{w}^T \mathbf{w}}.$$

$$\text{So } \mathbf{w}^T x - yr \|\mathbf{w}\| + b = 0$$

So, solving for r gives:

$$r = y(\mathbf{w}^T \mathbf{x} + b) / \|\mathbf{w}\|$$

Linear SVM Mathematically

The linearly separable case

Assume that all data is at least distance 1 from the hyperplane, then the following two constraints follow for a training set $\{(\mathbf{x}_i, y_i)\}$

$$\mathbf{w}^T \mathbf{x}_i + b \geq 1 \quad \text{if } y_i = 1$$

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1 \quad \text{if } y_i = -1$$

For support vectors, the inequality becomes an equality

Then, since each example's distance from the hyperplane is

$$r = y \frac{\mathbf{w}^T \mathbf{x} + b}{\|\mathbf{w}\|}$$

The margin is:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

Linear Support Vector Machine (SVM)

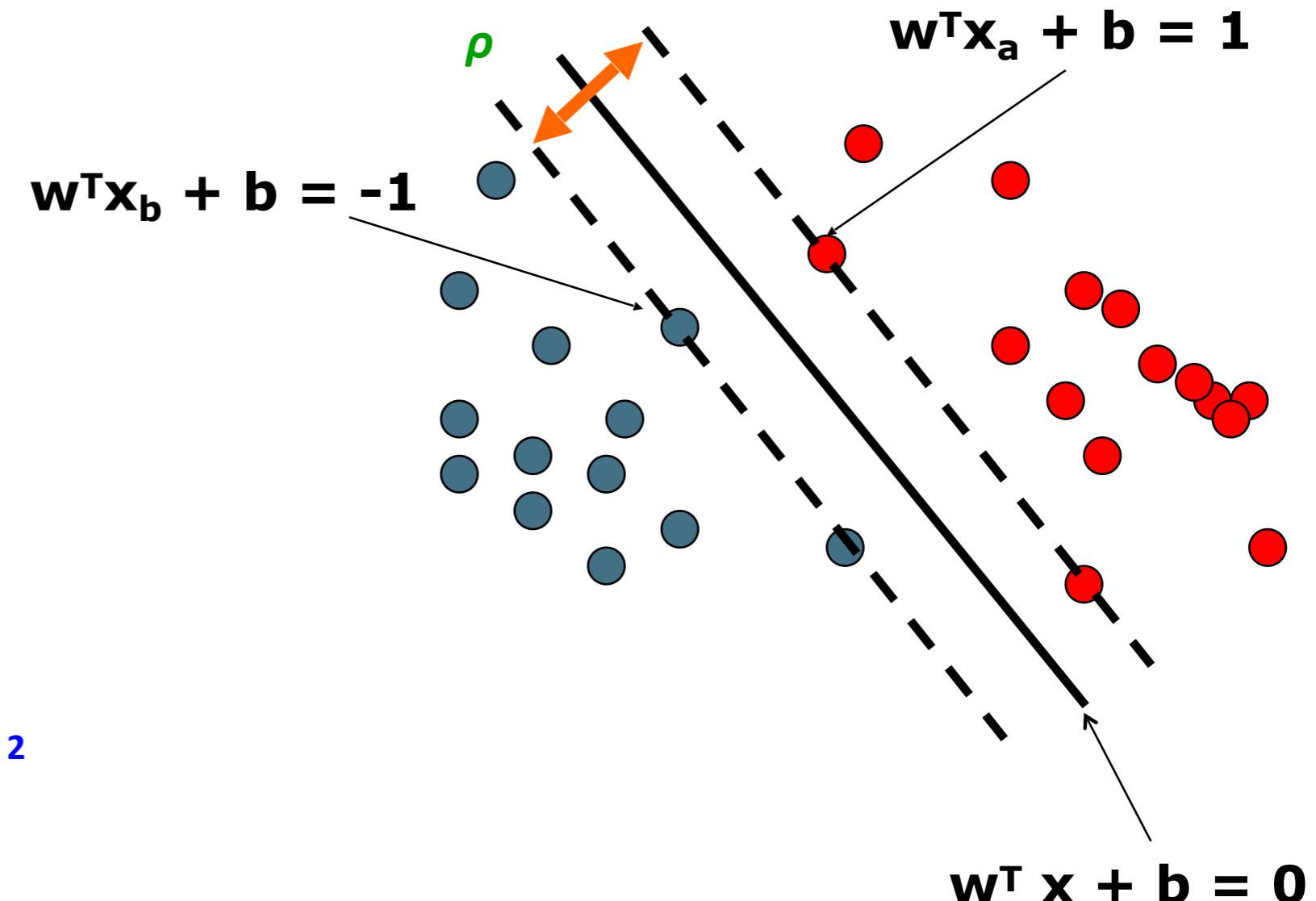
Hyperplane
 $\mathbf{w}^T \mathbf{x} + b = 0$

Extra scale constraint:
 $\min_{i=1,\dots,n} |\mathbf{w}^T \mathbf{x}_i + b| = 1$

This implies:

$$\mathbf{w}^T (\mathbf{x}_a - \mathbf{x}_b) = 2$$

$$\rho = \|\mathbf{x}_a - \mathbf{x}_b\|_2 = 2/\|\mathbf{w}\|_2$$





Support Vector Machine in R

Data Science Certification

Support Vector Machine

svm {e1071}

R Documentation

Support Vector Machines

Description

`svm` is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'  
svm(formula, data = NULL, ..., subset, na.action =  
na.omit, scale = TRUE)  
## Default S3 method:  
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =  
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),  
coef0 = 0, cost = 1, nu = 0.5,  
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,  
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,  
..., subset, na.action = na.omit)
```

Arguments

formula

a symbolic description of the model to be fit.

data

an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.

x

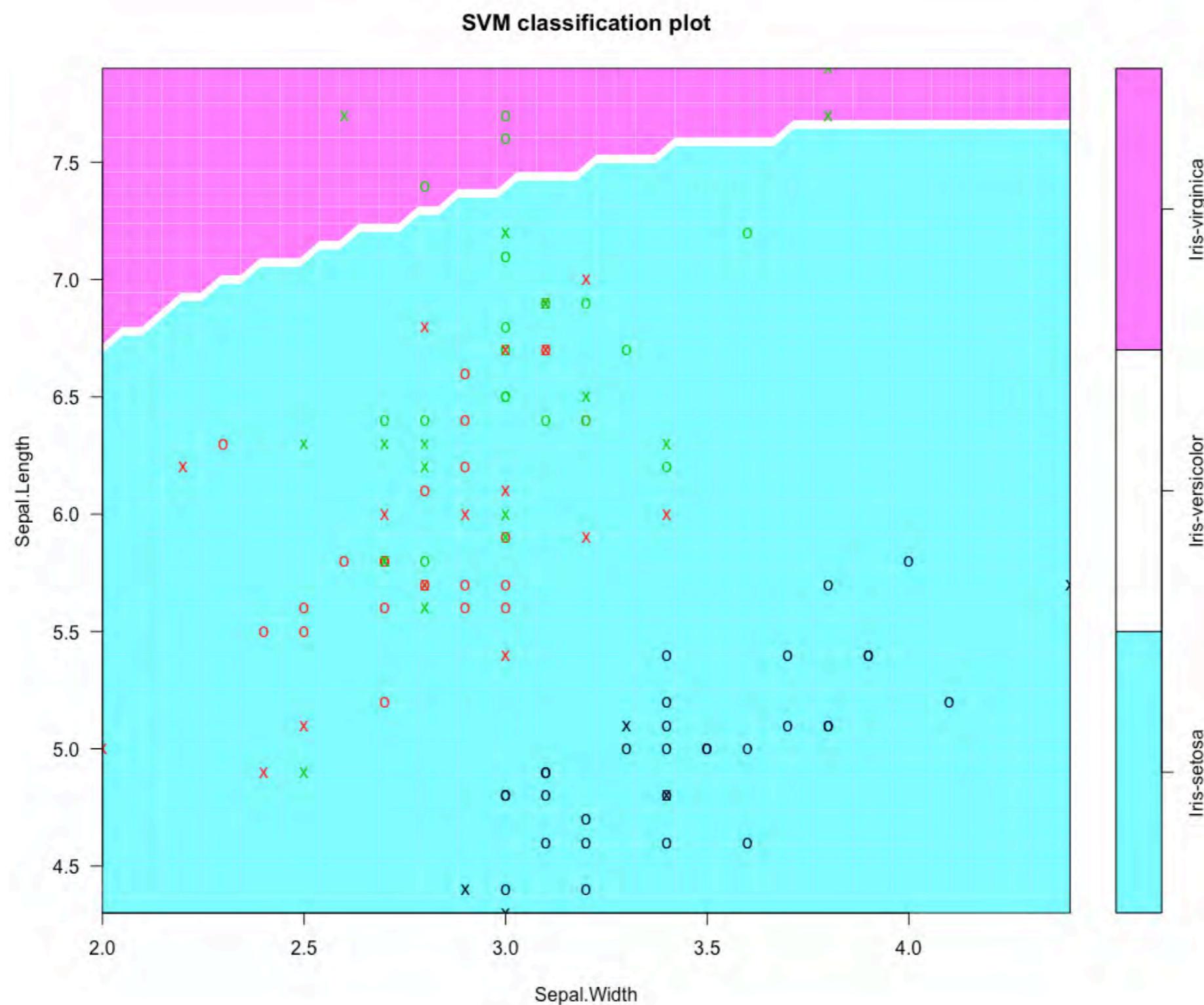
a data matrix, a vector, or a sparse matrix (object of class [Matrix](#) provided by the **Matrix** package, or of class [matrix.csr](#) provided by the **SparseM** package, or of class [simple_triplet_matrix](#) provided by the **slam** package).

y

a response vector with one label for each row/component of x. Can be either a factor (for classification tasks) or a numeric vector (for regression).

```
1
2 library(tidyverse)      # data manipulation and visualization
3 library(e1071)          # SVM methodology
4
5 # Fit Support Vector Machine model to data set
6 svmfit <- svm(Species~., data = trainData, method="C-classification",
7                 kernel = "radial", scale = FALSE, probability=T)
8 # Plot Results
9 plot(svmfit, data=trainData, formula=Sepal.Length~Sepal.Width)
10 prediction <- predict(svmfit, testData, probability=T)
11 table(testData$Species, prediction)
12 |
```

Support Vector Machine



```
> library(caret)
> confusionMatrix(testData$Species, prediction)
```

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	0
Iris-virginica	0	0	15

Overall Statistics

Accuracy : 1
95% CI : (0.9178, 1)

No Information Rate : 0.3488

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 1

McNemar's Test P-Value : NA

Statistics by Class:

	Class: Iris-setosa	Class: Iris-versicolor	Class: Iris-virginica
Sensitivity	1.0000	1.0000	1.0000
Specificity	1.0000	1.0000	1.0000
Pos Pred Value	1.0000	1.0000	1.0000
Neg Pred Value	1.0000	1.0000	1.0000
Prevalence	0.3488	0.3023	0.3488
Detection Rate	0.3488	0.3023	0.3488
Detection Prevalence	0.3488	0.3023	0.3488
Balanced Accuracy	1.0000	1.0000	1.0000

Workshop 4.7 - SVM

1. Use data from Workshop 4.2
2. Perform SVM Algorithm to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

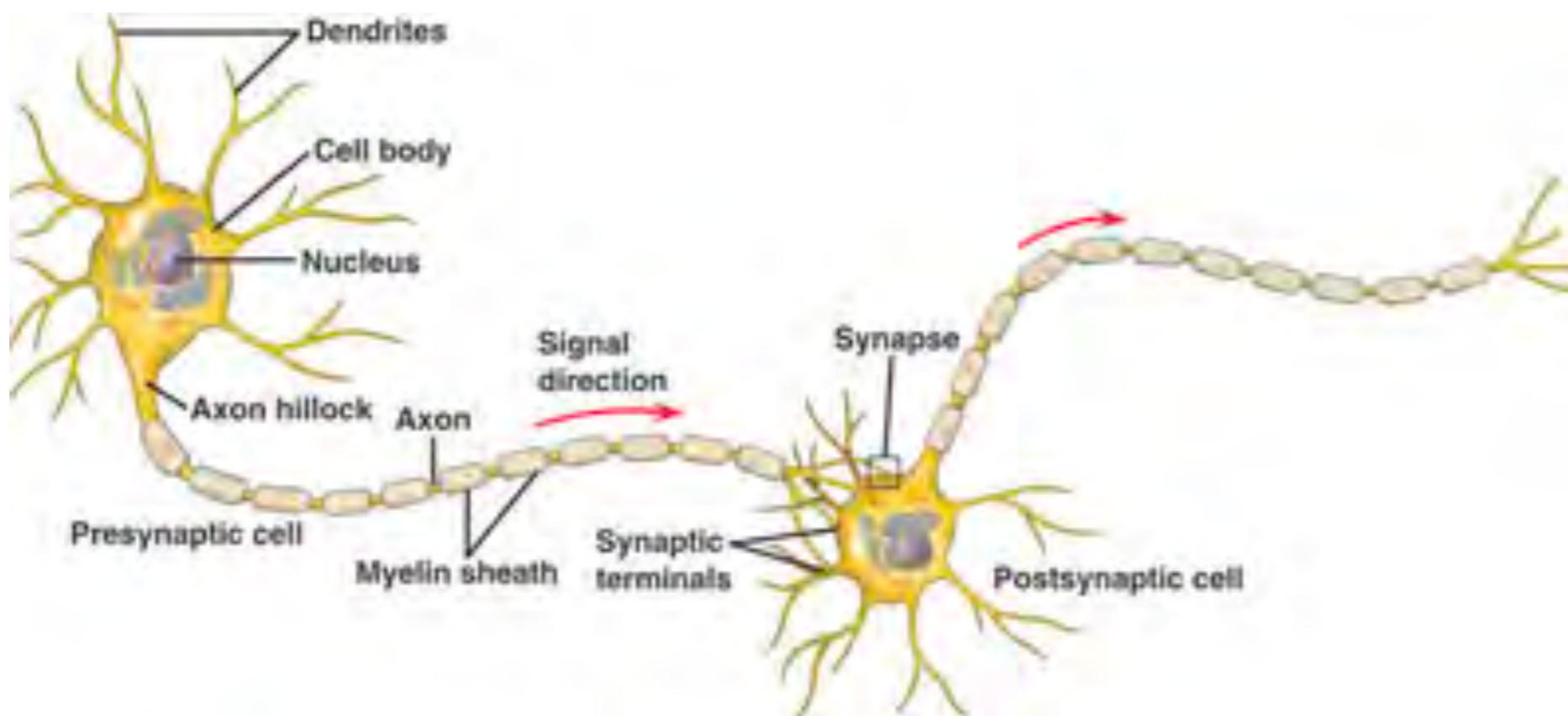
- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- **Neural Net**
- Ensemble methods: Bagging and Boosting
- Summary

Neural Network

- Neural networks learning methods provide a robust approach to approximating real-valued, discrete valued and vector valued target functions.
- Often used in problems such as handwriting, voice, or image recognition
- **Feed-Forward Neural Networks, Convolutional Neural Networks, Deep Learning**

Biological Motivation

- Inspired by biological learning systems that are built from very complex webs of interconnected neurons

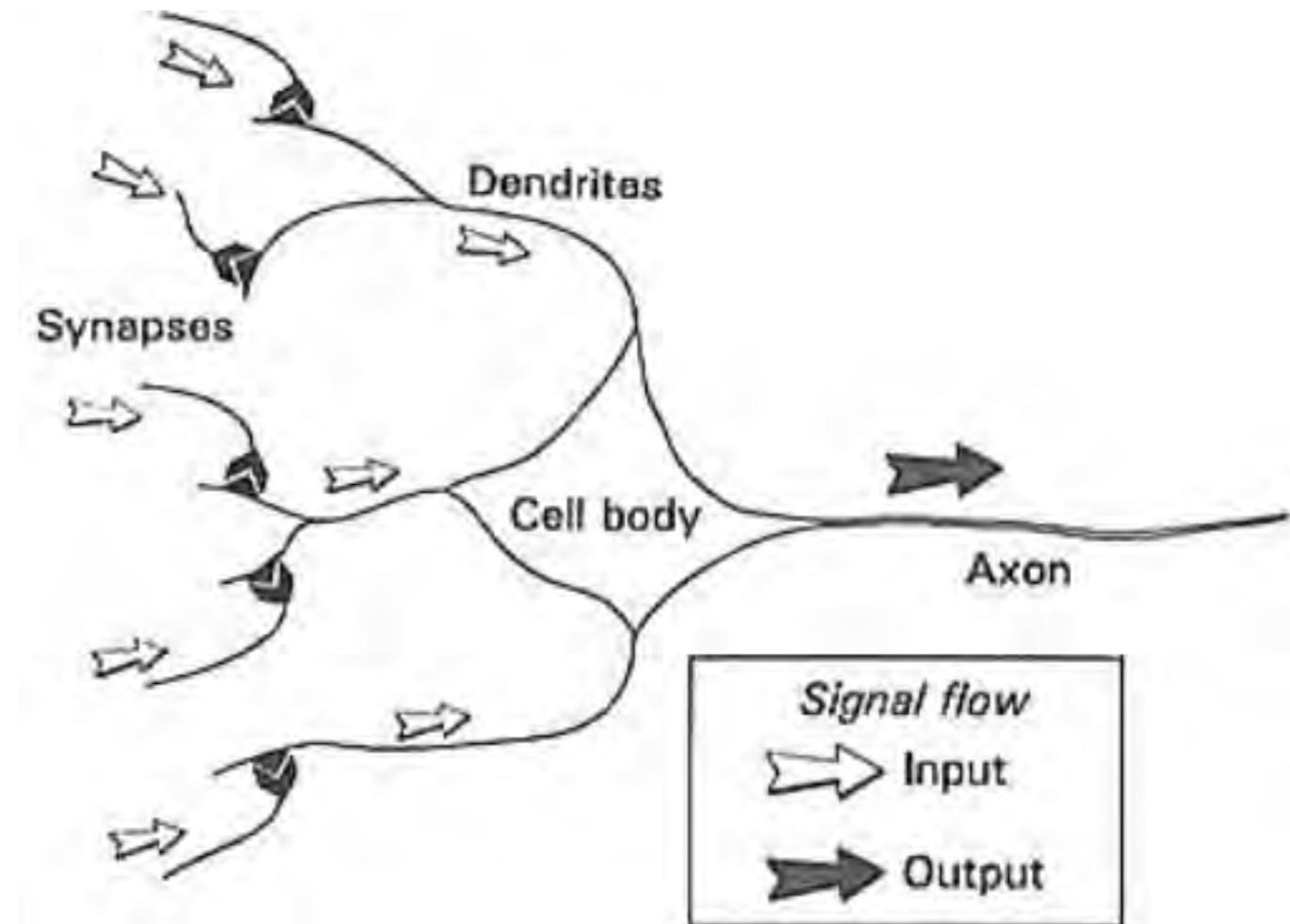


Facts About Neuro-biology

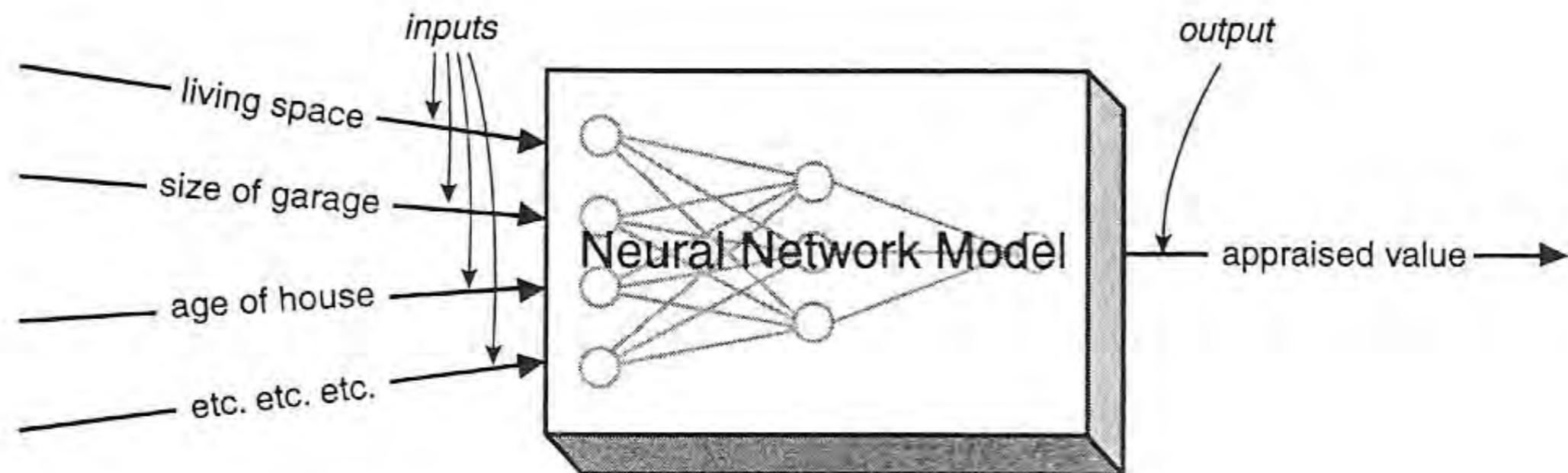
- Human brain is estimated to contain a densely interconnected network of approximately 10^{11} neurons.
- Each connected, on average, to 10^4 other neurons.
- Neuron activity is typically excited through connections to other neurons.
- The fastest neuron switching times are in the order of 10^{-3} seconds.
- It requires 10^{-1} seconds to visibly recognize your mother.

Analogy of Biological Learning Systems

- Artificial neural networks are built out of a interconnected set of simple units, where each unit takes a number of real-valued inputs (can be the outputs of other units) and produces a single real-valued output (which may become the input of other units).



Example

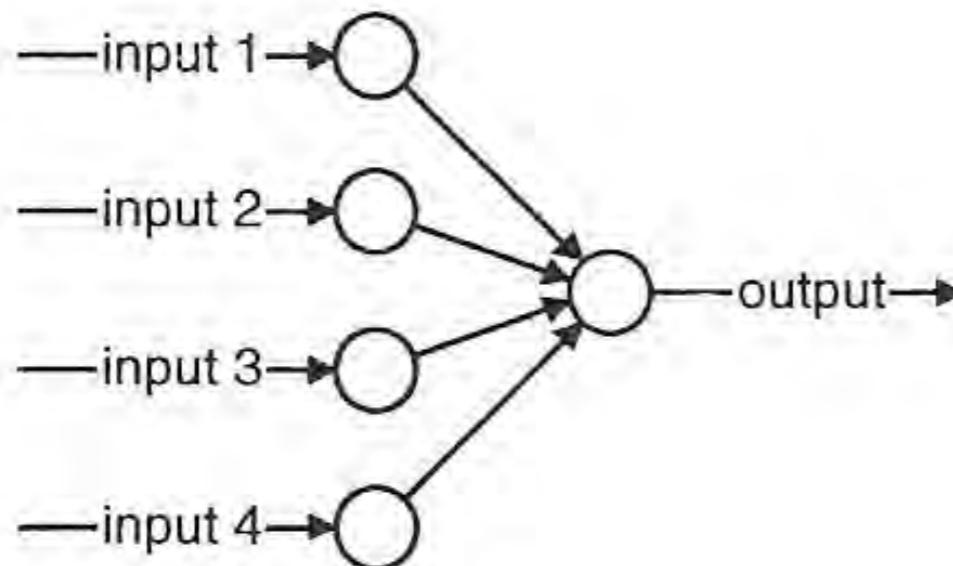


A neural network is like a black box that know how to process input to create an output. The calculation is quite complex and difficult to understand.

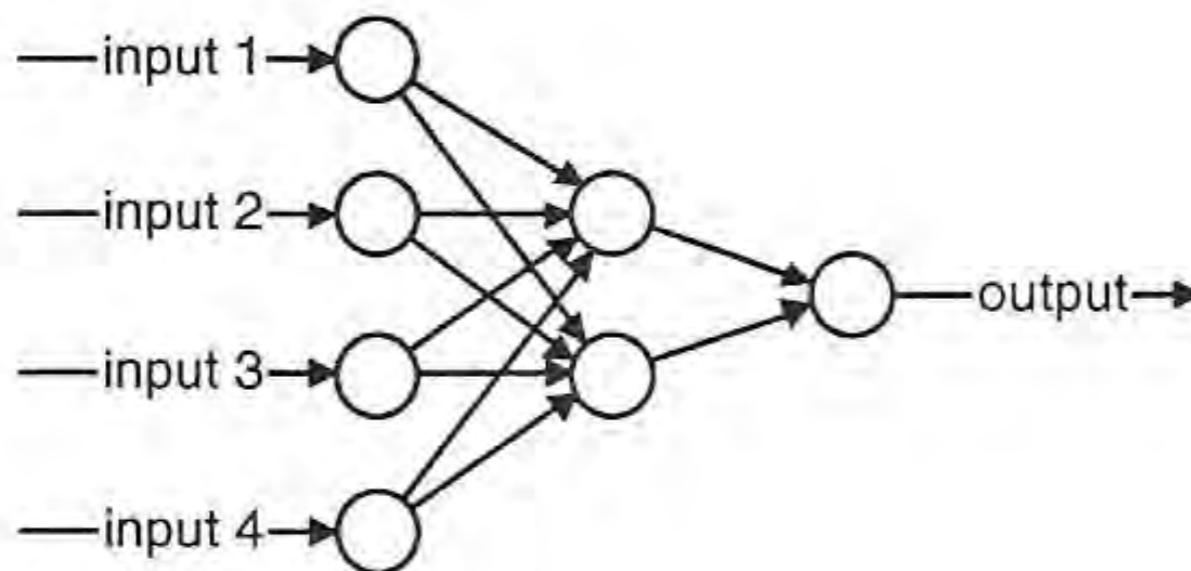
Neural Networks Process

1. Identify the input and output features
2. Transform the input and output so that they are in a small range
(-1 to 1)
3. Set up a network with an appropriate topology
4. Train the network on a representative set of training examples
5. Use the validation set to choose the set of weights that minimizes the error
6. Evaluate the network using the test set to see how well it performs
7. Apply the model generated by the network to predict outcomes for unknown input

What is a Neural Network? (1)

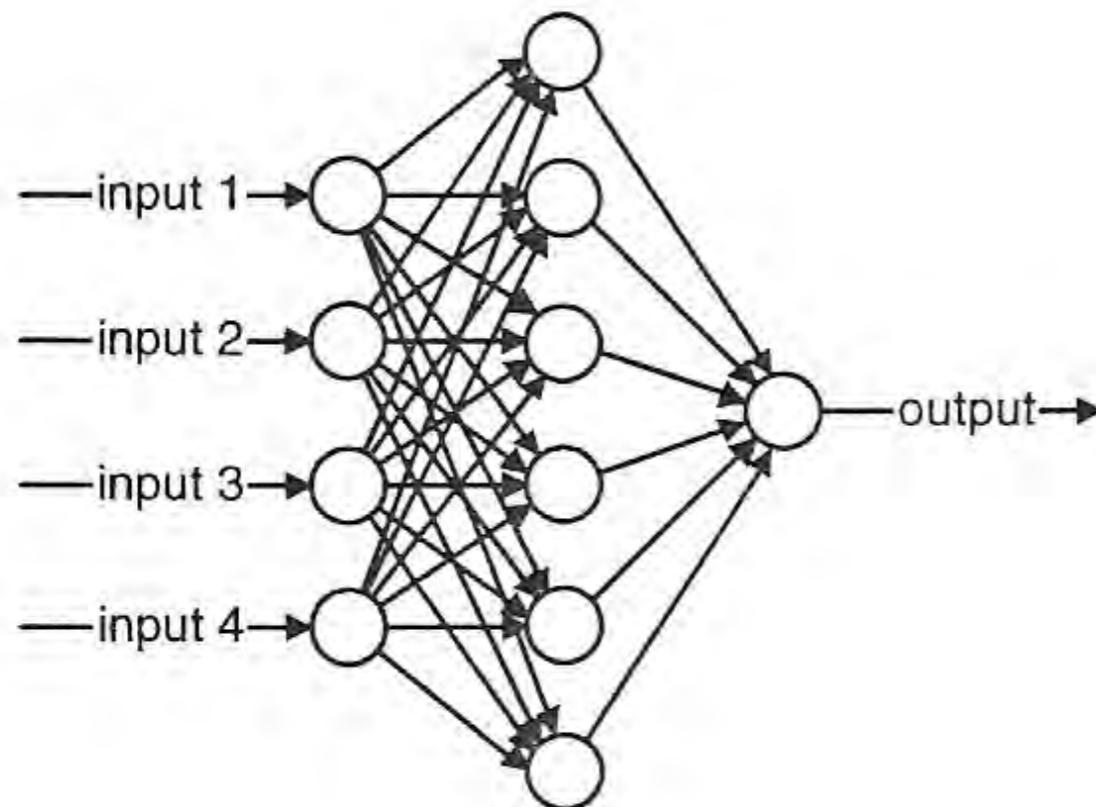


This simple neural network takes four inputs and produces an output. This result of training this network is equivalent to the statistical technique called logistic regression.

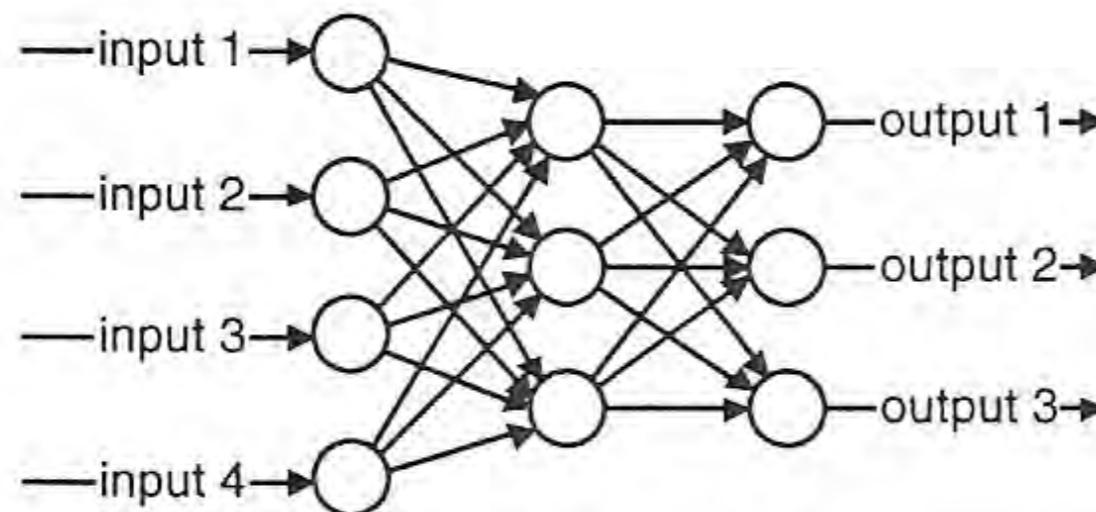


This network has a middle layer called the *hidden layer*, which makes the network more powerful by enabling it to recognize more patterns.

What is a Neural Network? (2)



Increasing the size of the hidden layer makes the network more powerful but introduces the risk of overfitting. Usually, only one hidden layer is needed.



A neural network can produce multiple output values.

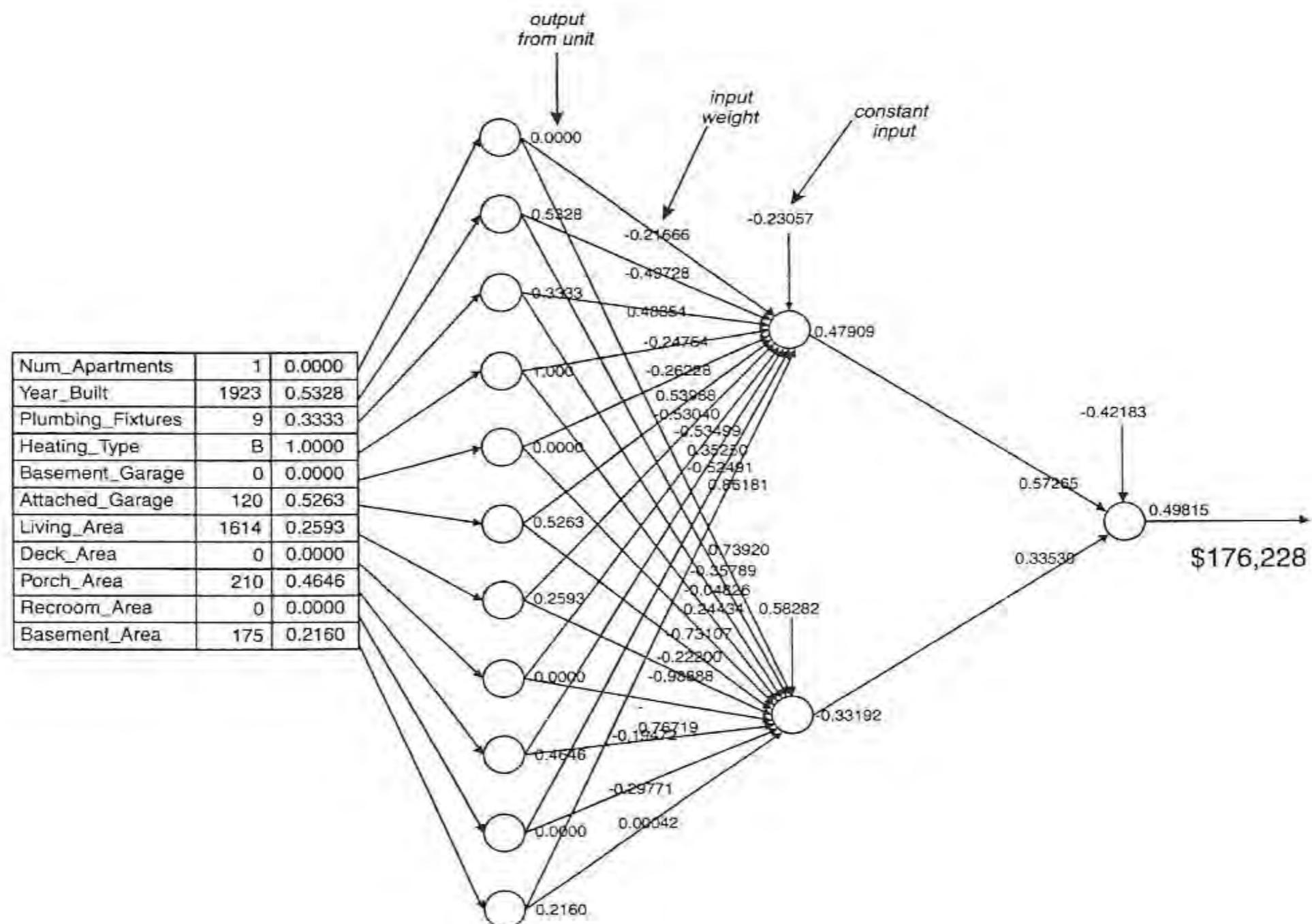
Three Main Components of a Neural Network

- Activation function
- Network topology
- How the network is trained?
 - Feed forward
 - Back propagation

Activation Function (1)

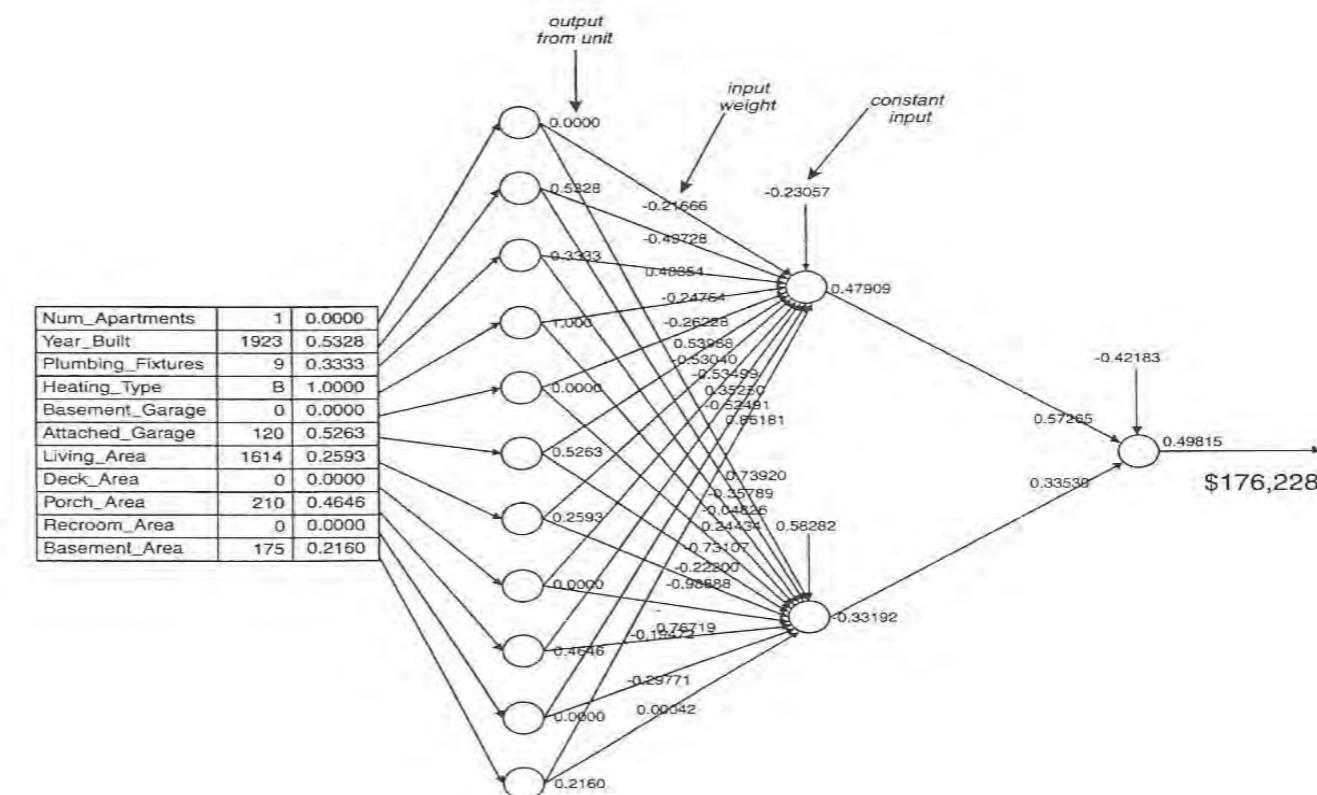
- **Combination function**
 - The function that merges all the input into a single value.
 - The most common combination function is the weighted sum.
- **Transfer function**
 - The function that transfers the value of the combination function to the output of the unit.

Feed-Forward Neural Networks (1)



Feed-Forward Neural Networks: Input Layer

- **Input layer**
 - Each unit in the input layer is connected to exactly one source field.



Feed-Forward Neural Networks: Hidden layer (1)

- **Hidden layer**
 - Each unit in the hidden layer is typically connected to all the units in the input layer.
 - The units in the hidden layer calculate their output by multiplying the value of each input by its corresponding weight, adding them up, and applying the transfer function.

Feed-Forward Neural Networks: Hidden layer (2)

- **Hidden layer**
 - A neural network can have any number of hidden layers, but in general **one** hidden layer is sufficient.
 - The wider the layer (the more units it contains), the greater the capacity of the network to recognize patterns.

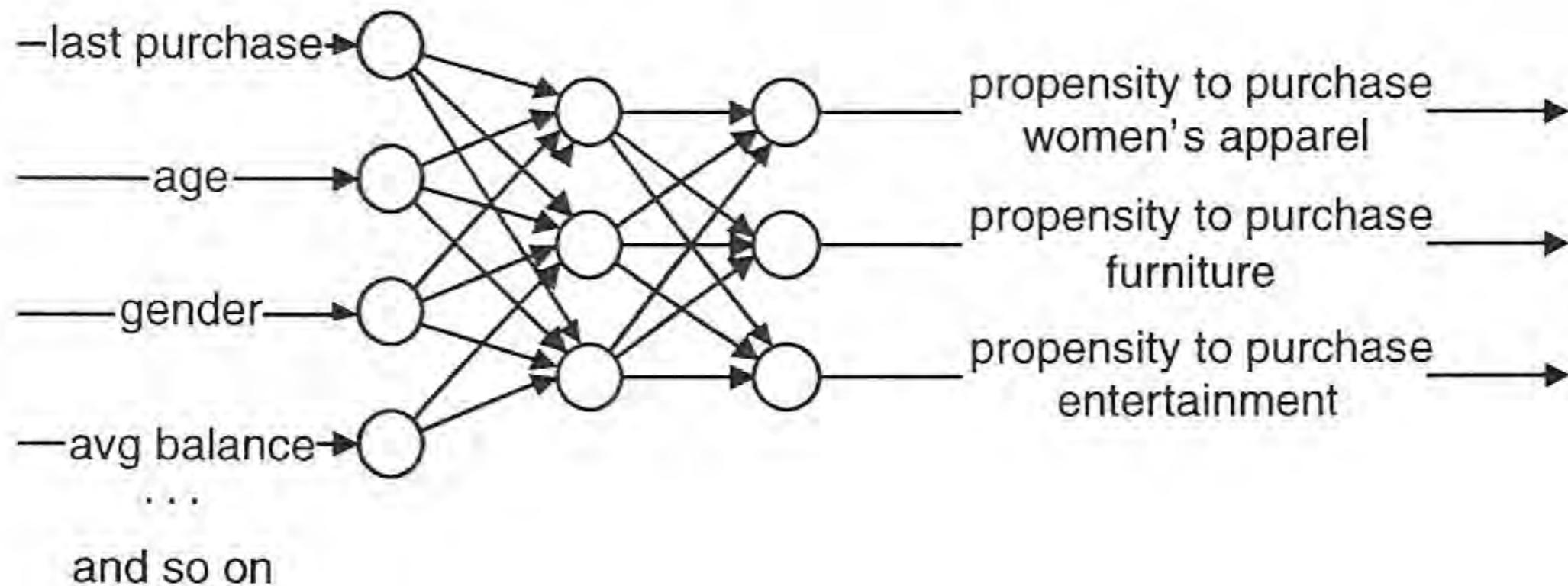
Feed-Forward Neural Networks : Output Layer (1)

- **Output layer**
 - It is fully connected to all the units in the hidden layer.
 - Most of the time, the neural network is being used to calculate a single value, so there is only one unit in the output layer and the value.
 - **We must map this value back to understand the output.**

Feed-Forward Neural Networks: Output Layer (2)

- **Output layer**
 - It is possible for the output layer to have more than one unit.
 - Eg. A department store chain wants to predict the likelihood that customers will be purchasing products from various departments, such as women's apparel, furniture, and entertainments.

Feed-Forward Neural Networks: Output Layer (3)



- After feeding the inputs for a customer into the network, the network calculates three values.

Back-Propagation Algorithm

- Assume the network is a fixed structure that corresponds to a directed graph.
- Learning corresponds to choosing a weight value for each edge in the graph.
- It attempts to minimize the squared error between the network output values and the target values for these outputs.
- It is the ANN learning technique that is most commonly used.

Back Propagation Neural Networks: Steps (1)

1. The network gets a training example and, using the existing weights in the network, it calculates the output.
2. Back propagation calculates the error by taking the difference between the calculated result and the expected (actual result).

Back Propagation Neural Networks: Steps (2)

3. The error is fed back through the network and the weights are adjusted to minimize the error.
4. After being shown enough training examples, the weights on the network no longer change significantly and the error no longer decreases. This is the point where training stops.

Back-Propagation Algorithm: Input

- Training example is a pair of the form $\langle x, t \rangle$, where x is the vector of network input values and t is the vector of target network output values
 - n is the learning rate
 - n_{in} is the number of network inputs
 - n_{hidden} is the number of units in the hidden layer
 - n_{out} is the number of output units
- The input from unit i into unit j is denoted x_{ji}
- The weight from unit i to unit j is denoted w_{ji}

Back-Propagation Algorithm: process (1)

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units
- Initialize all network weight to small random numbers
- Until the termination condition is met, Do repeat the steps

Back-Propagation Algorithm: process (2)

Propagate the input forward through the network:

1. Input the instance x to the network and compute the output o_u of every unit u in the network

Propagate the errors backward through the network:

2. For each network output unit k , calculate its error term e_k

$$e_k \leftarrow o_k (1 - o_k) (t_k - o_k)$$

Back-Propagation Algorithm: process

3. For each hidden unit h , calculate its error term e_h

$$e_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh} e_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = n e_j x_{ji}$$

Termination conditions

- Stop after a fixed number of iterations
- Stop when the error on the training examples falls below some threshold
- Stop when the error on a separate validation set of examples meets some criterion

Hidden Layer Representations

- Training examples constrain the network inputs and outputs.
- The weight tuning procedure is free to set weights that define whatever hidden unit representation is most effective in minimizing the squared error.
- The ability of multilayer networks to automatically discover useful representation **at the hidden layers** is a key feature of ANN learning.

Heuristics for Using Feed-Forward & Back Propagation Networks

- One important decision is the **number of units in the hidden layer**. The more unit, the more patterns the network can recognize.
- We generally **do not use hidden layers larger than the number of inputs**.
- A good place to start for many problems is to experiment with one, two, and three nodes in the hidden layer.

Choosing the Training Set: Coverage of values for all Features

- In general, it is good to have several examples in the training set for each value of a categorical feature and for values throughout the ranges of ordered discrete and continuous features.

Choosing the Training Set: Number of Features

- The more features used as inputs into the networks, the larger the networks needs to be, increasing the risk of overfitting and increasing the size of the training set.
- The more features, the longer it takes the network to converge to a set of weights.
- In many cases, it is useful to calculate new variables that represent particular aspects of the business problems.

Choosing the Training Set: Size of Training Set

- The more features there are, the more training examples that are needed to get a good coverage of patterns in the data.
- Overfitting is guaranteed to happen when there are fewer training examples than there are weights in the network.

Choosing the Training Set: Number of Output

- The number of training examples for each possible output should be about the same.

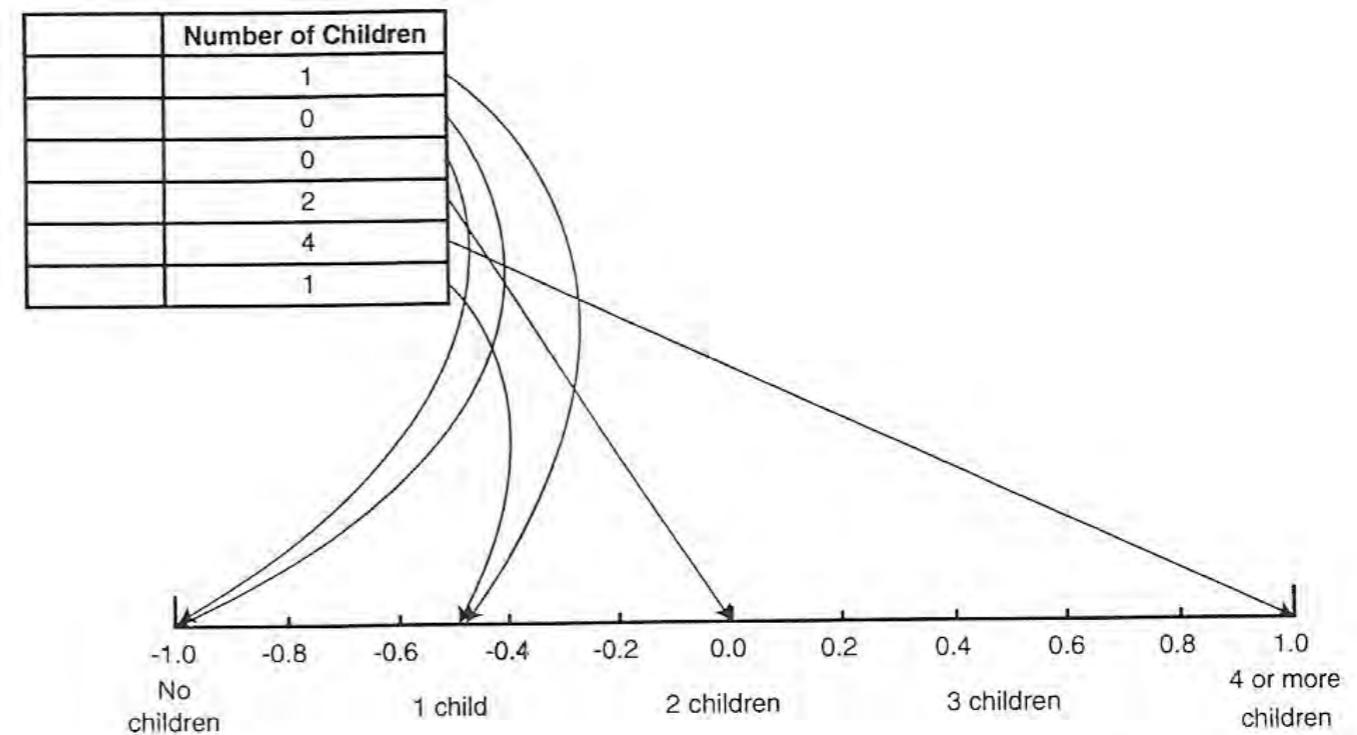
Preparing the Data: Features with Continuous Values

- The value should be scaled to be in a reasonable range.
- Plan for a larger range
- Reject out-of-range values
- Peg values lower than the minimum to the minimum and higher than the maximum to the maximum
- Map the minimum value to -0.9 and the maximum value to 0.9 instead of -1 and 1

Preparing the Data: Features with Ordered Discrete Values (1)

- First, count the number of different values and assign each a proportional fraction into some range.
- eg. If there are five distinct values, these get mapped to 0, 0.25, 0.50, 0.75, and 1.0

$$\begin{array}{rcl} 0 & \rightarrow & 0\ 0\ 0\ 0 = 0/16 = 0.0000 \\ 1 & \rightarrow & 1\ 0\ 0\ 0 = 8/16 = 0.5000 \\ 2 & \rightarrow & 1\ 1\ 0\ 0 = 12/16 = 0.7500 \\ 3 & \rightarrow & 1\ 1\ 1\ 0 = 14/16 = 0.8750 \end{array}$$



Preparing the Data: Features with Ordered Discrete Values (2)

- It is also possible to break a range into unequal parts.
- One example is called thermometer codes

$$\textcircled{m} \ 0 \rightarrow 0\ 0\ 0 = 0 / 16 = 0.0000$$

$$\textcircled{m} \ 1 \rightarrow 1\ 0\ 0 = 8 / 16 = 0.5000$$

$$\textcircled{m} \ 2 \rightarrow 1\ 1\ 0 = 12 / 16 = 0.7500$$

$$\textcircled{m} \ 3 \rightarrow 1\ 1\ 1 = 14 / 16 = 0.8750$$

It shows that the difference on one end of the scale is less significant than differences on the other end.

Preparing the Data: Features with Categorical Values (1)

- When working with categorical variables in neural networks, the mapping of variables into numbers **introduces an ordering** of the variables, which the neural network takes into account.
- The second way of handling categorical features is to break the categories into flags, one for each value.

Preparing the Data: Features with Categorical Values (2)

Gender	N coding			N-1 coding	
	Gender Male Flag	Gender Female Flag	Gender Unknown Flag	Gender Male Flag	Gender Female Flag
Male	+1.0	-1.0	-1.0	+1.0	-1.0
Female	-1.0	+1.0	-1.0	-1.0	+1.0
Unknown	-1.0	-1.0	+1.0	-1.0	-1.0

Interpreting the Results (1)

- When estimating a continuous value, often the output needs to be scaled back to the correct range.
- For binary or categorical output variables, the approach is still to take the inverse of the translation used for training the network.
- eg. If “churn” is given the value of 1 and “no churn” a value of -1 , the values near 1 represent churn and those near -1 represent no churn.

Interpreting the Results (2)

- When there are two outcomes, the meaning of the output depends on the training set used to train the network.
- The average value produced by the network during training is usually going to be close to the average value in the training set.

Interpreting the Results (3)

- One way to handle is, eg.
- If the training set had 50% churn and 50% no churn, the average value the network will produce from the training examples is going to be close to 0.0.
- Values higher than 0.0 are more like churn and those less than 0.0, less like churn.
- If the original training set had 10% churn, then the cutoff would more reasonable be –
- 0.8 rather than 0.0 (0.8 is 10% of the way from -1 to 1).

Interpreting the Results (4)

- For binary values, it is also possible to create a network that produces two output, one for each value.
- In this case, each output represents the strength of evidence that that category is the correct one. The chosen category would then be the one with the higher value.

How to Know What is Going on Inside a Neural Network

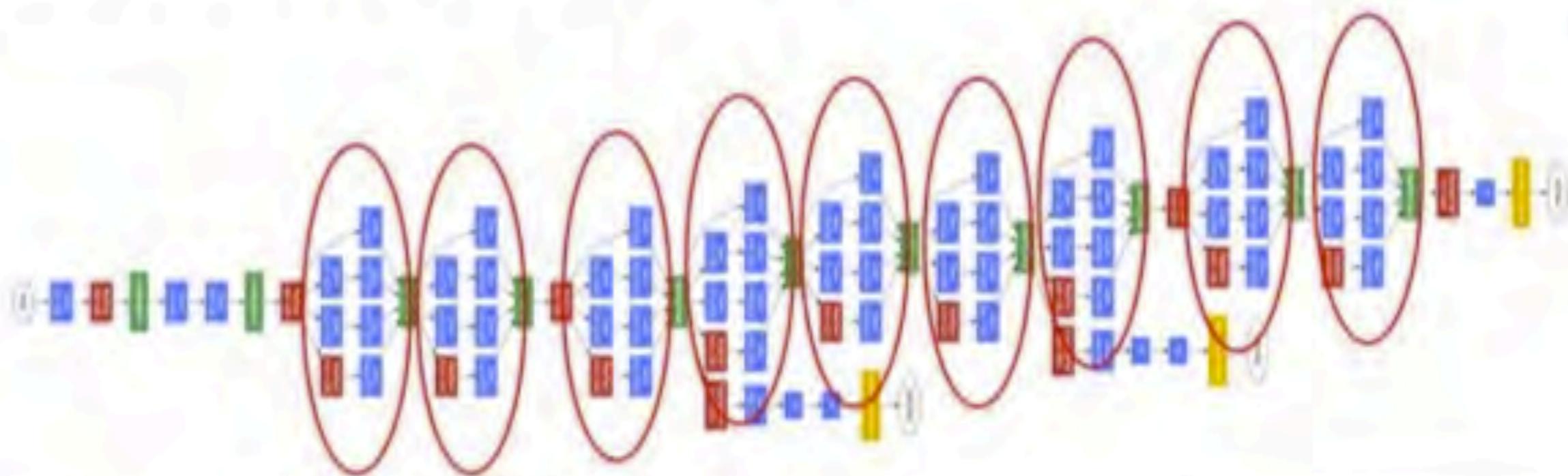
Sensitivity analysis uses the test set to determine how sensitive the output of the network is to each input :

1. Find the average value for each input.
2. Measure the output of the network when all inputs are at their average value.
3. Measure the output of the network when each input is modified, one at a time, to be at **its minimum and maximum values**.

Summary

- Neural networks can be used for both categorical and continuous inputs.
- Neural networks learn best when input fields have been mapped to the range between -1 and +1.
- Neural networks work best when there are only a few variables.
- When training a network, there is no guarantee that the resulting set of weights is optimal.
- A neural network cannot explain what it is doing.

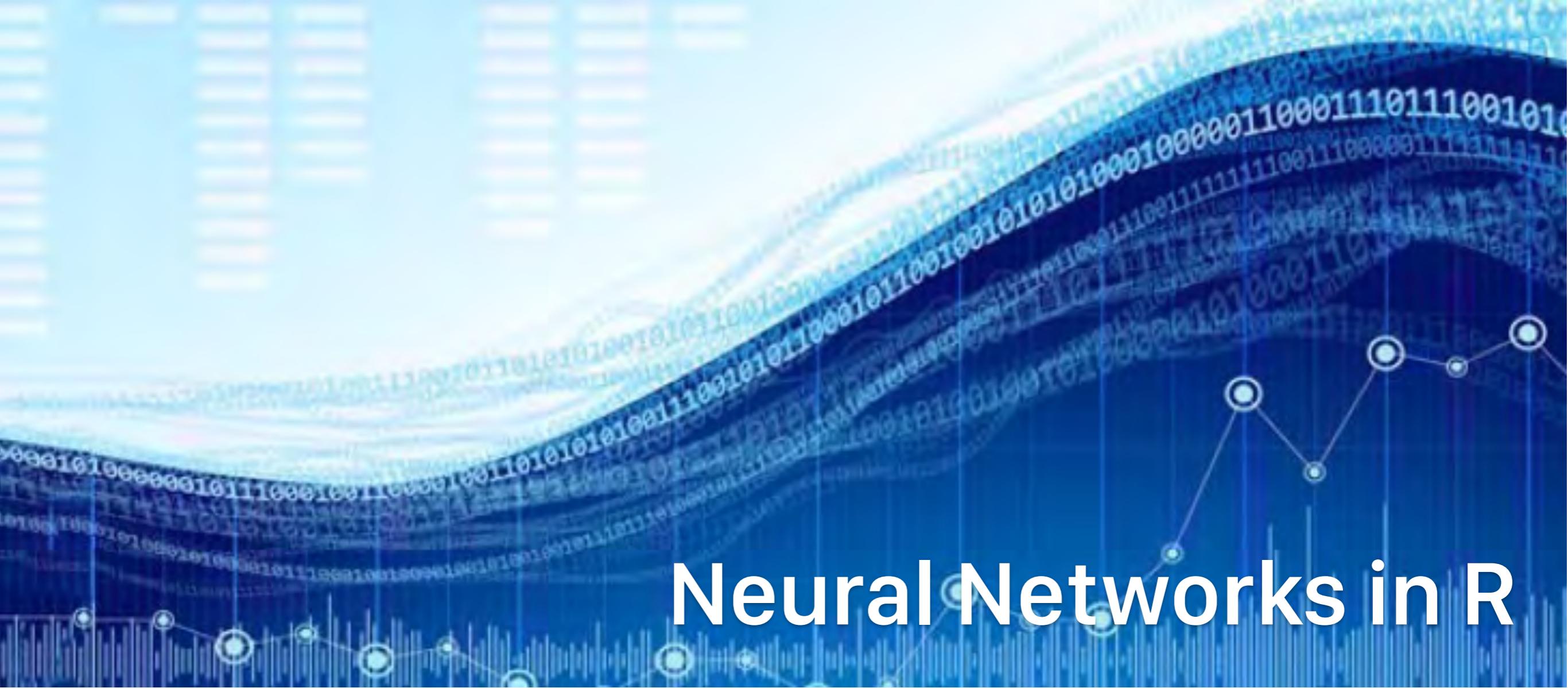
GoogLeNet (Inception)



9 **Inception** modules

Network in a network in a network...

Convolution
Pooling
Softmax
Other



Neural Networks in R

Data Science Certification

Neural Network

Training of neural networks

Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,  
          stepmax = 1e+05, rep = 1, startweights = NULL,  
          learningrate.limit = NULL,  
          learningrate.factor = list(minus = 0.5, plus = 1.2),  
          learningrate=NULL, lifesign = "none",  
          lifesign.step = 1000, algorithm = "rprop+",  
          err.fct = "sse", act.fct = "logistic",  
          linear.output = TRUE, exclude = NULL,  
          constant.weights = NULL, likelihood = FALSE)
```

Neural Network

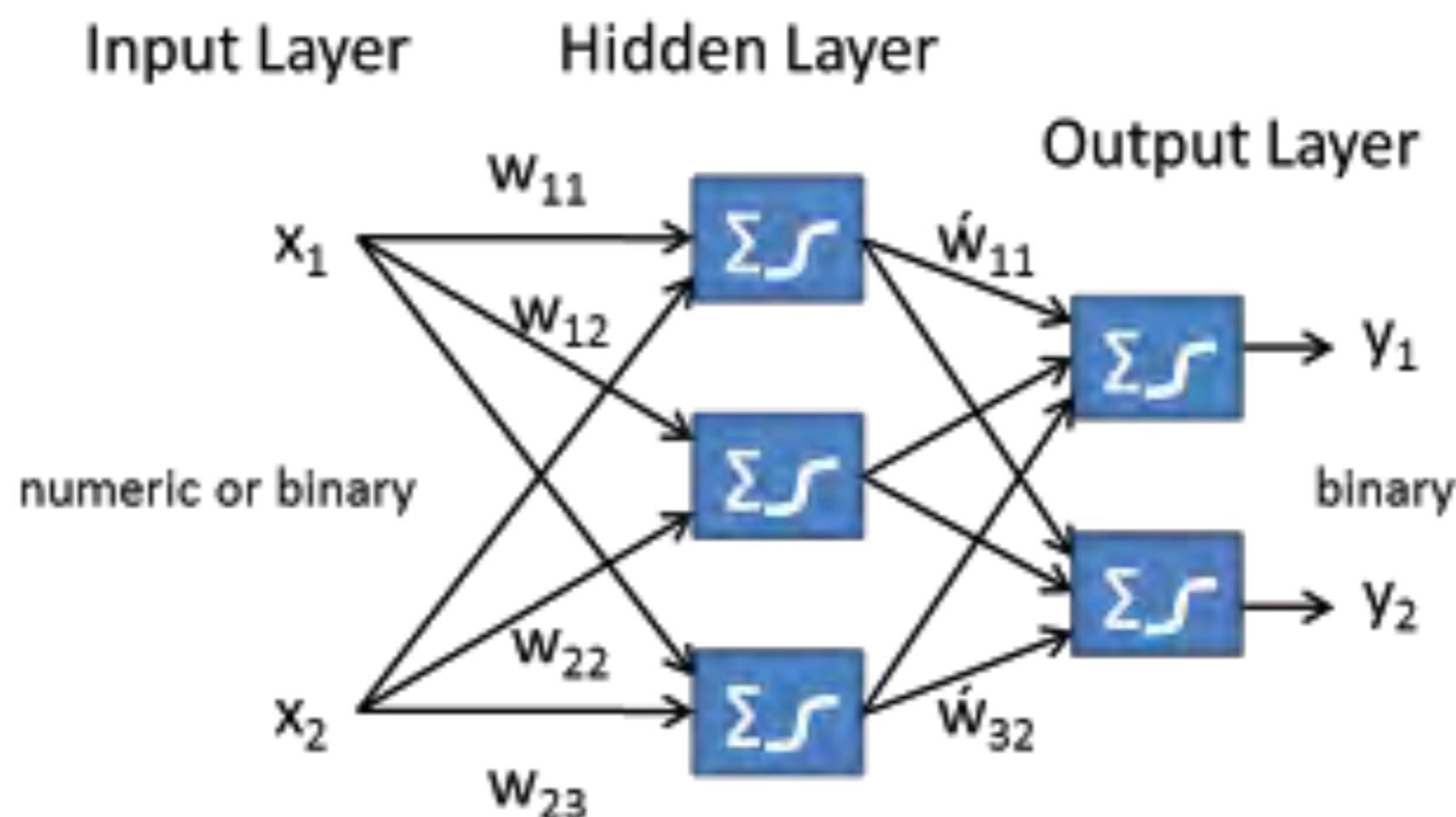
Arguments

<code>formula</code>	a symbolic description of the model to be fitted.
<code>data</code>	a data frame containing the variables specified in <code>formula</code> .
<code>hidden</code>	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
<code>threshold</code>	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
<code>stepmax</code>	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
<code>rep</code>	the number of repetitions for the neural network's training.
<code>startweights</code>	a vector containing starting values for the weights. The weights will not be randomly initialized.
<code>learningrate.limit</code>	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
<code>learningrate.factor</code>	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
<code>learningrate</code>	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
<code>lifesign</code>	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

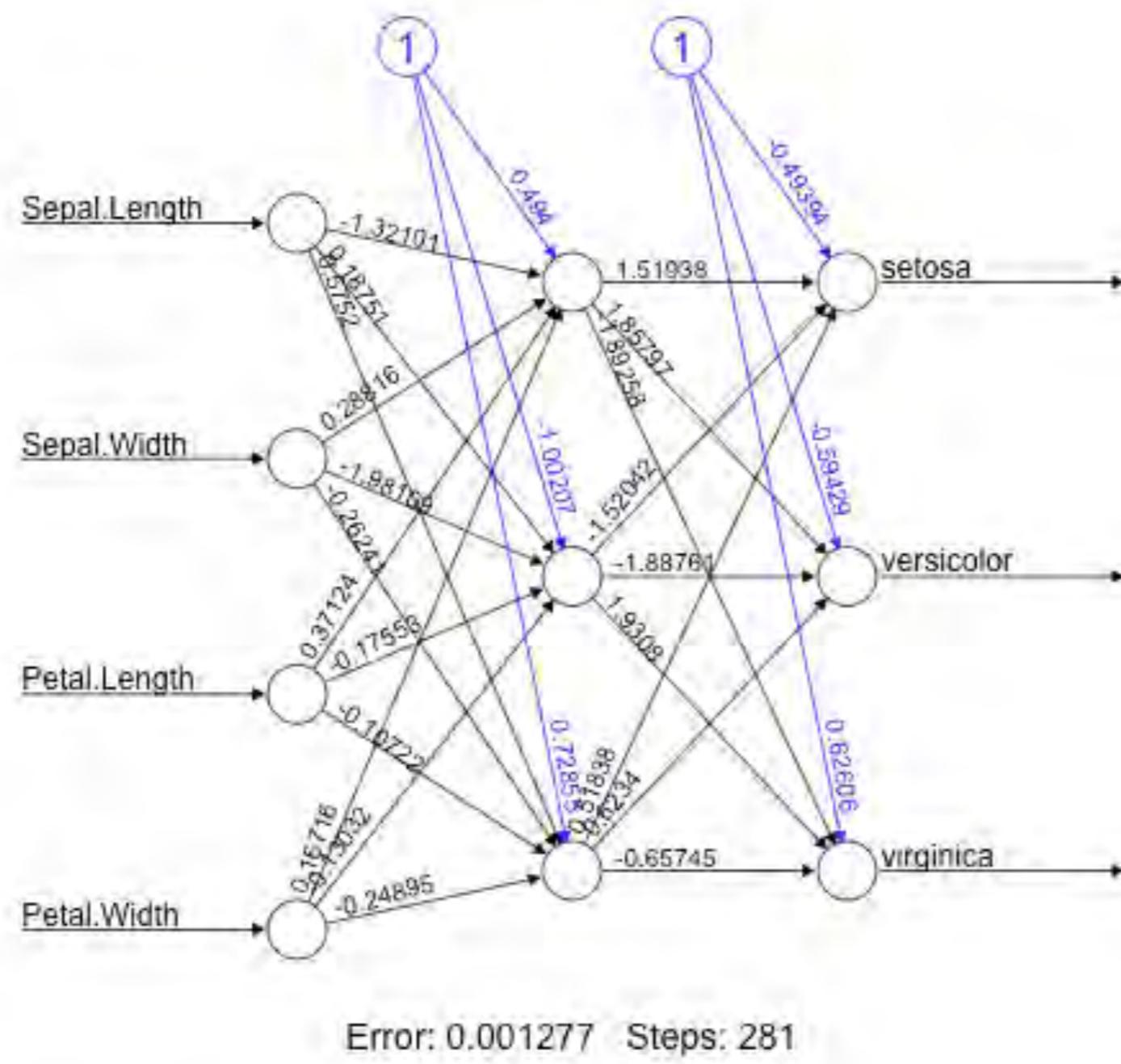
Neural Network

<code>lifesign.step</code>	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
<code>algorithm</code>	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
<code>err.fct</code>	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
<code>act.fct</code>	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
<code>linear.output</code>	logical. If <code>act.fct</code> should not be applied to the output neurons set <code>linear.output</code> to TRUE, otherwise to FALSE.
<code>exclude</code>	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
<code>constant.weights</code>	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
<code>likelihood</code>	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of <code>confidence.interval</code> is meaningful.

Neural Network



```
2
3 library(neuralnet)
4 iris <- read.csv("iris.data.csv", header=TRUE)
5 # Prepare iris
6 set.seed(567)
7 ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
8 trainData <- iris[ind==1,]
9 testData <- iris[ind==2,]
10 nnet_iristrain <- trainData
11
12 #Binarize the categorical output
13 nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-setosa')
14 nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-versicolor')
15 nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-virginica')
16 names(nnet_iristrain)[6] <- 'Setosa'
17 names(nnet_iristrain)[7] <- 'Versicolor'
18 names(nnet_iristrain)[8] <- 'Virginica'
19
20 nn <- neuralnet(Setosa+Versicolor+Virginica ~ Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,
21                   data=nnet_iristrain, hidden=c(3))
22
23 plot(nn)
24 mypredict <- compute(nn, testData[-5])$net.result
25 # Put multiple binary output to categorical output
26 maxidx <- function(arr) {
27   return(which(arr == max(arr)))
28 }
29 idx <- apply(mypredict, c(1), maxidx)
30 prediction <- c('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')[idx]
31 table(prediction, testData$Species)
32
33
```



```
> table(prediction, testData$Species)
```

prediction	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	0
Iris-virginica	0	0	15

Accuracy

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	12	0
Iris-virginica	0	1	15

Overall Statistics

Accuracy : 0.9767442

95% CI : (0.8771095, 0.9994114)

No Information Rate : 0.3488372

P-Value [Acc > NIR] : < 0.000000000000022204

Kappa : 0.9649837

Mcnemar's Test P-Value : NA



Statistics by Class:

	Class: Iris-setosa	Class: Iris-versicolor	Class: Iris-virginica
Sensitivity	1.0000000	0.9230769	1.0000000
Specificity	1.0000000	1.0000000	0.9642857
Pos Pred Value	1.0000000	1.0000000	0.9375000
Neg Pred Value	1.0000000	0.9677419	1.0000000
Prevalence	0.3488372	0.3023256	0.3488372
Detection Rate	0.3488372	0.2790698	0.3488372
Detection Prevalence	0.3488372	0.2790698	0.3720930
Balanced Accuracy	1.0000000	0.9615385	0.9821429

Workshop 4.8 - Neural Networks

1. Use data from Workshop 4.2
2. Build Neural Networks Model to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Topic

- Basic concepts
- Decision tree induction
- Logistic Regression
- Evaluation of classifiers
- Naïve Bayesian classification
- K-nearest neighbor
- Support Vector Machine
- Neural Net
- **Ensemble methods: Bagging and Boosting**
- Summary

Combining classifiers

- So far, we have only discussed individual classifiers, i.e., how to build them and use them.
- Can we combine multiple classifiers to produce a better classifier?
- Yes, sometimes
- We discuss two main algorithms:
 - **Bagging**
 - **Boosting**

Bagging

- Breiman, 1996
- Bootstrap Aggregating = Bagging
 - Application of bootstrap sampling
 - Given: set D containing m training examples
 - Create a sample $S[i]$ of D by drawing m examples at random with replacement from D
 - $S[i]$ of size m: expected to leave out 0.37 of examples from D

Bagging (cont...)

- **Training**
 - Create k bootstrap samples $S[1], S[2], \dots, S[k]$
 - Build a distinct classifier on each $S[i]$ to produce k classifiers, using the same learning algorithm.
- **Testing**
 - Classify each new instance by voting of the k classifiers (equal weights)

Bagging Example

Original	1	2	3	4	5	6	7	8
Training set 1	2	7	8	3	7	6	3	1
Training set 2	7	8	5	6	4	2	7	1
Training set 3	3	6	2	7	5	6	2	2
Training set 4	4	5	1	4	6	4	3	8

Bagging (cont ...)

When does it help?

When learner is unstable

Small change to training set causes large change in the output classifier

True for decision trees, neural networks; not true for k -nearest neighbor, naïve Bayesian, class association rules

Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

Boosting

- A family of methods:
 - We only study **AdaBoost** (Freund & Schapire, 1996)
- Training
 - Produce a sequence of classifiers (the same base learner)
 - Each classifier is dependent on the previous one, and focuses on the previous one's errors
 - Examples that are incorrectly predicted in previous classifiers are given higher weights
- Testing
 - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

AdaBoost

Weighted training set

(x_1, y_1, w_1)

(x_2, y_2, w_2)

...

(x_n, y_n, w_n)

Non-negative weights
sum to 1

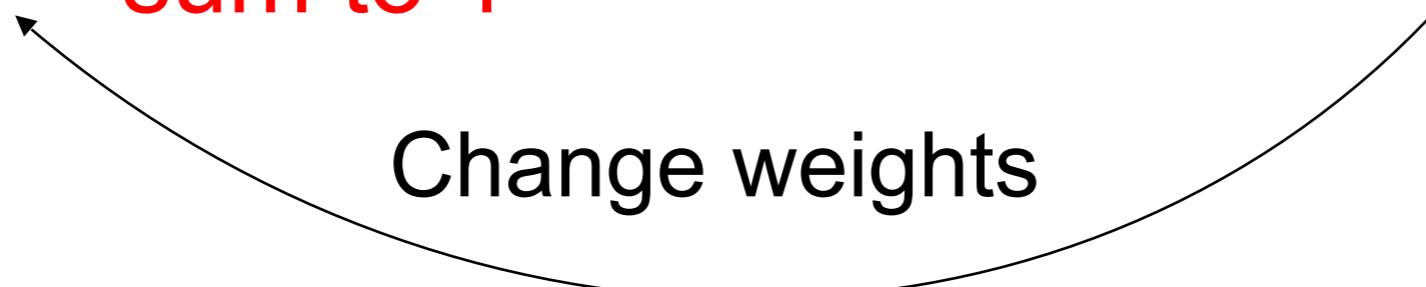


called a weaker classifier



- Build a classifier h_t whose accuracy on training set $> \frac{1}{2}$ (better than random)

Change weights



AdaBoost algorithm

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle(x_1, y_1), \dots, (x_m, y_m)\rangle$
with labels $y_i \in Y = \{1, \dots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i .

Do for $t = 1, 2, \dots, T$:

1. Call **WeakLearn**, providing it with the distribution D_t .
2. Get back a hypothesis $h_t : X \rightarrow Y$.

3. Calculate the error of h_t : $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.

If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.

4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t :

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$h_{fin}(x) = \arg \max_{y \in Y} \sum_{t: h_t(x) = y} \log \frac{1}{\beta_t}.$$

Does AdaBoost always work?

- The actual performance of boosting depends on the data and the base learner.
 - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
 - When the number of outliers is very large, the emphasis placed on the hard examples can hurt the performance.

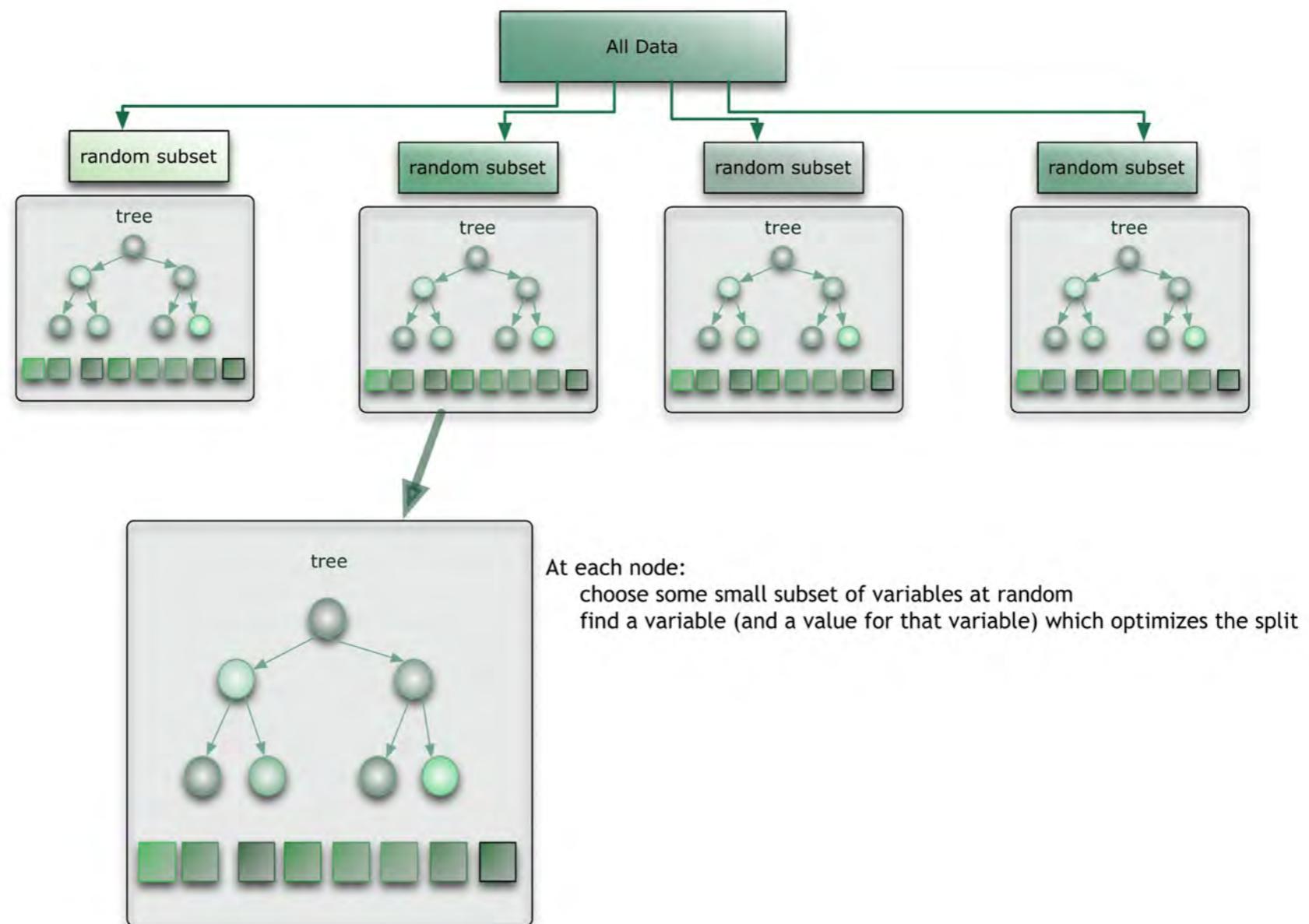


Ensemble in R

Fundamental Data Science for Data Scientist

Ensemble : Bagging

Random Forest



Random Forest

Here is how such a system is trained; for some number of trees T :

- 1) Sample N cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
- 2) At each node:
 - a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.
 - b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
 - c) At the next node, choose another m variables at random from all predictor variables and do the same.

Bagging

```
> library(randomForest)
#Train 100 trees, random selected attributes
> model <- randomForest(Species~, data=iristrain, nTree=500)
#Predict using the forest
> prediction <- predict(model, newdata=iristest, type='class')
> table(prediction, iristest$Species)
> importance(model)
      MeanDecreaseGini
Sepal.Length      7.807602
Sepal.Width       1.677239
Petal.Length     31.145822
Petal.Width      38.617223
```

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	2
Iris-virginica	0	0	13

Overall Statistics

Accuracy : 0.9535

95% CI : (0.8419, 0.9943)

No Information Rate : 0.3488

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9303

Mcnemar's Test P-Value : NA

Statistics by Class:

	Class: Iris-setosa	Class: Iris-versicolor
Sensitivity	1.0000	1.0000
Specificity	1.0000	0.9333
Pos Pred Value	1.0000	0.8667
Neg Pred Value	1.0000	1.0000
Prevalence	0.3488	0.3023
Detection Rate	0.3488	0.3023
Detection Prevalence	0.3488	0.3488
Balanced Accuracy	1.0000	0.9667

	Class: Iris-virginica
Sensitivity	0.8667
Specificity	1.0000
Pos Pred Value	1.0000
Neg Pred Value	0.9333
Prevalence	0.3488
Detection Rate	0.3023
Detection Prevalence	0.3023
Balanced Accuracy	0.9333

Boosting

```
> library(adabag)  
  
> iris.adaboost <- boosting(Species~, data=trainData, boost=TRUE, mfinal=5)  
  
> iris.adaboost$class  
  
> table(iris.adaboost$class, trainData$Species)  
  
> prediction <- predict(iris.adaboost, newdata=testData)  
  
> table(prediction$class, testData$Species)  
  
> confusionMatrix(as.factor(prediction$class), testData$Species)
```

Confusion Matrix and Statistics

Prediction	Reference		
	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	15	0	0
Iris-versicolor	0	13	1
Iris-virginica	0	0	14

Overall Statistics

Accuracy : 0.9767

95% CI : (0.8771, 0.9994)

No Information Rate : 0.3488

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9651

McNemar's Test P-Value : NA

Statistics by Class:

Class: Iris-setosa Class: Iris-versicolor

Sensitivity	1.0000	1.0000
Specificity	1.0000	0.9667
Pos Pred Value	1.0000	0.9286
Neg Pred Value	1.0000	1.0000
Prevalence	0.3488	0.3023
Detection Rate	0.3488	0.3023
Detection Prevalence	0.3488	0.3256
Balanced Accuracy	1.0000	0.9833

Class: Iris-virginica

Sensitivity	0.9333
Specificity	1.0000
Pos Pred Value	1.0000
Neg Pred Value	0.9655
Prevalence	0.3488
Detection Rate	0.3256
Detection Prevalence	0.3256
Balanced Accuracy	0.9667

Workshop 4.9 - Ensemble Method

1. Use data from Workshop 4.2
2. Train your model using either Bagging (Random Forest) or Boosting (AdaBoost) to predict outcomes.
3. Perform performance evaluation by creating confusion matrix

Machine Learning Summary

- Applications of supervised learning are in almost any field or domain.
- We studied 8 classification techniques.
- There are still many other methods, e.g.,
 - Bayesian networks
 - Neural networks
 - Genetic algorithms
 - Fuzzy classification
 - This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.



Regression Model

Data Science Certification

Workshop 4.10 - Regression Prediction

1. Use Workshop 3.6, convert from Statistic Regression to Predictive Regression

Hint:

- Use Split data into Train and Test

- Do regression model
- Predict Test data and
- Find Accuracy



Unsupervised Learning

Data Science Certification

Supervised learning vs. unsupervised learning

Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute.

These patterns are then utilized to predict the values of the target attribute in future data instances.

Unsupervised learning: The data have no target attribute.

We want to explore the data to find some intrinsic structures in them.

Clustering

- Clustering is a technique for finding **similarity groups** in data, called **clusters**. i.e.,
 - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.
- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.
- Due to historical reasons, clustering is often considered synonymous with unsupervised learning.
 - In fact, association rule mining is also unsupervised
- This chapter focuses on clustering.

An illustration

The data set has three natural groups of data points, i.e., 3 natural clusters.



What is clustering for?

- Let us see some real-life examples
- Example 1: groups people of similar sizes together to make “small”, “medium” and “large” T-Shirts.
 - Tailor-made for each person: too expensive
 - One-size-fits-all: does not fit all.
- Example 2: In marketing, segment customers according to their similarities
 - To do targeted marketing.

What is clustering for? (cont...)

Example 3: Given a collection of text documents, we want to organize them according to their content similarities,

To produce a topic hierarchy

In fact, clustering is one of the most utilized data mining techniques.

It has a long history, and used in almost every field, e.g., medicine, psychology, botany, sociology, biology, archeology, marketing, insurance, libraries, etc.

In recent years, due to the rapid increase of online documents, text clustering becomes important.

Aspects of clustering

- A clustering algorithm
 - Partitional clustering
 - Hierarchical clustering
 - ...
- A distance (similarity, or dissimilarity) function
- Clustering quality
 - Inter-clusters distance \Rightarrow maximized
 - Intra-clusters distance \Rightarrow minimized
- The **quality** of a clustering result depends on the algorithm, the

K-means clustering

K-means is a **partitional clustering** algorithm

Let the set of data points (or instances) D be

$$\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\},$$

where $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ir})$ is a **vector** in a real-valued space $X \subseteq \mathbb{R}^r$, and r

is the number of attributes (dimensions) in the data.

The k-means algorithm partitions the given data into k clusters.

Each cluster has a cluster **center**, called **centroid**.

k is specified by the user

K-means algorithm

Given k , the k-means algorithm works as follows:

- 1) Randomly choose k data points (**seeds**) to be the initial **centroids**, cluster centers
- 2) Assign each data point to the closest **centroid**
- 3) Re-compute the **centroids** using the current cluster memberships.
- 4) If a convergence criterion is not met, go to 2).

K-means algorithm – (cont ...)

```
Algorithm k-means( $k$ ,  $D$ )
1   Choose  $k$  data points as the initial centroids (cluster centers)
2   repeat
3       for each data point  $\mathbf{x} \in D$  do
4           compute the distance from  $\mathbf{x}$  to each centroid;
5           assign  $\mathbf{x}$  to the closest centroid      // a centroid represents a cluster
6       endfor
7       re-compute the centroids using the current cluster memberships
8   until the stopping criterion is met
```

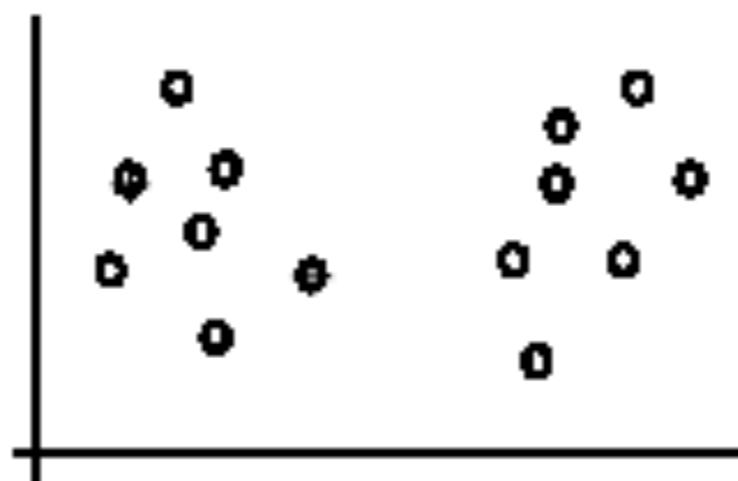
Stopping/convergence criterion

1. no (or minimum) re-assignments of data points to different clusters,
2. no (or minimum) change of centroids, or
3. minimum decrease in the **sum of squared error** (SSE),

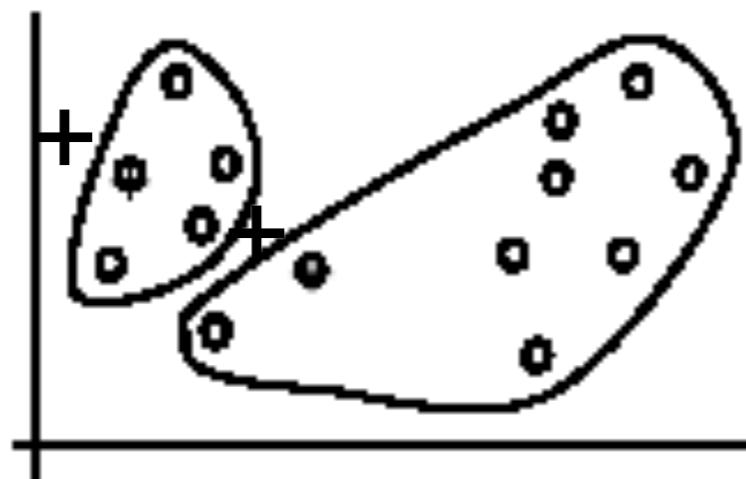
C_i is the j th cluster, \mathbf{m}_j is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point \mathbf{x} and centroid \mathbf{m}_j .

$$SSE = \sum_{j=1}^k \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \quad (1)$$

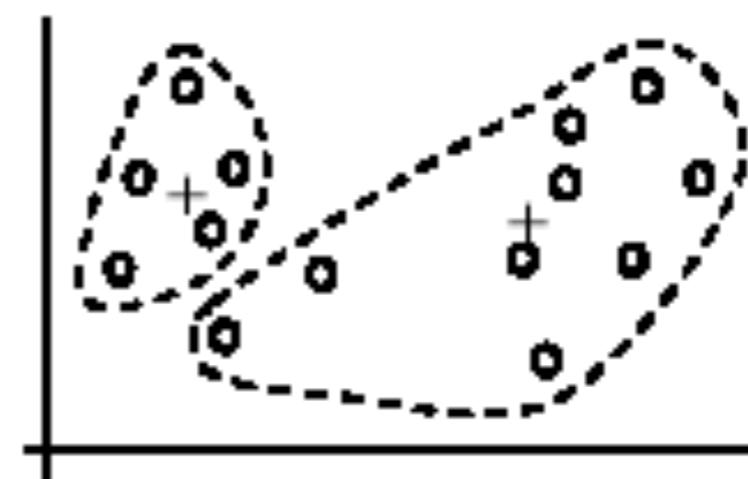
An example



(A). Random selection of k centers

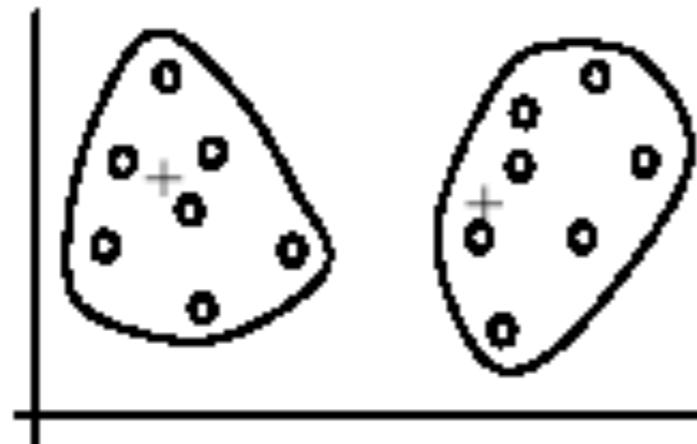


Iteration 1: (B). Cluster assignment

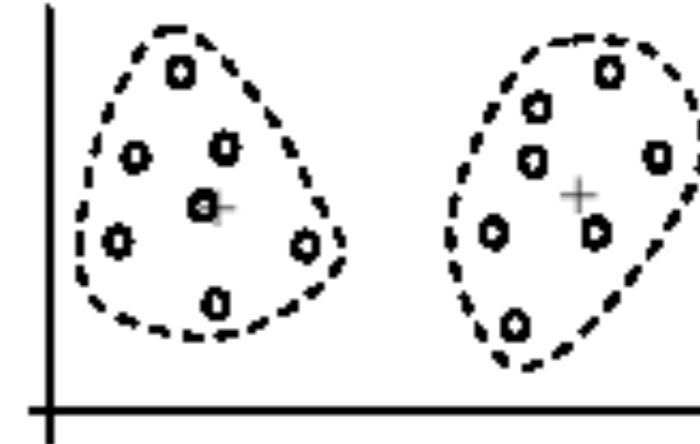


(C). Re-compute centroids

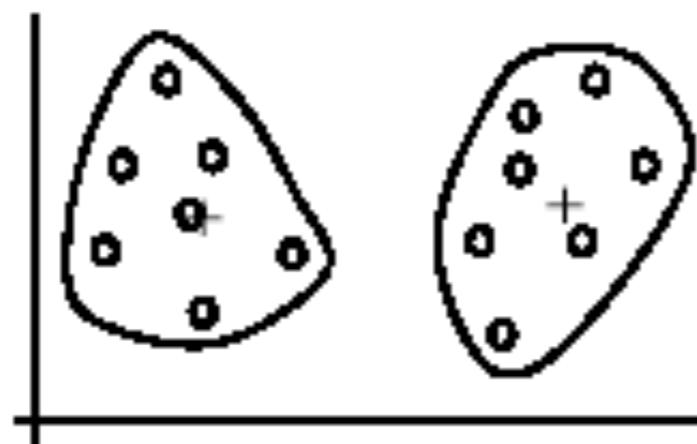
An example (cont ...)



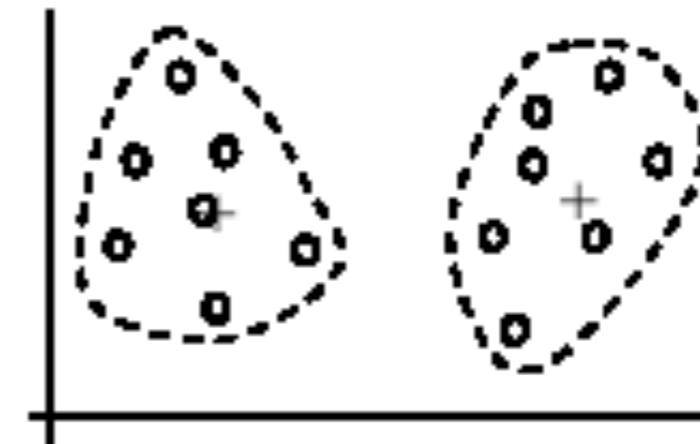
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

An example distance function

The k -means algorithm can be used for any application data set where the **mean** can be defined and computed. In the **Euclidean space**, the mean of a cluster is computed with:

$$\mathbf{m}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \quad (2)$$

where $|C_j|$ is the number of data points in cluster C_j . The distance from one data point \mathbf{x}_i to a mean (centroid) \mathbf{m}_j is computed with

$$\begin{aligned} dist(\mathbf{x}_i, \mathbf{m}_j) &= \| \mathbf{x}_i - \mathbf{m}_j \| \\ &= \sqrt{(x_{i1} - m_{j1})^2 + (x_{i2} - m_{j2})^2 + \dots + (x_{ir} - m_{jr})^2} \end{aligned} \quad (3)$$

A disk version of k -means

- K-means can be implemented with data on disk
 - In each iteration, it scans the data once.
 - as the centroids can be computed incrementally
- It can be used to cluster large datasets that do not fit in main memory
- We need to control the number of iterations
 - In practice, a limit is set (< 50).
- Not the best method. There are other scale-up algorithms, e.g., BIRCH (balanced iterative reducing and clustering using hierarchies).

A disk version of k-means (cont...)

Algorithm disk- k -means(k, D)

- 1 Choose k data points as the initial centroids $\mathbf{m}_j, j = 1, \dots, k,$
- 2 **repeat**
- 3 initialize $\mathbf{s}_j = \mathbf{0}, j = 1, \dots, k,$ // $\mathbf{0}$ is a vector with all 0's
- 4 initialize $n_j = 0, j = 1, \dots, k;$ // n_j is the number points in cluster j
- 5 **for** each data point $\mathbf{x} \in D$ **do**
- 6 $j = \arg \min_j \text{dist}(\mathbf{x}, \mathbf{m}_j);$
- 7 assign \mathbf{x} to the cluster $j;$
- 8 $\mathbf{s}_j = \mathbf{s}_j + \mathbf{x};$
- 9 $n_j = n_j + 1;$
- 10 **endfor**
- 11 $\mathbf{m}_i = \mathbf{s}_j / n_j, i = 1, \dots, k,$
- 12 **until** the stopping criterion is met

Strengths of k-means

Strengths:

Simple: easy to understand and to implement

Efficient: Time complexity: $O(tkn)$,

where n is the number of data points,

k is the number of clusters, and

t is the number of iterations.

Since both k and t are small, k-means is considered a linear algorithm.

K-means is the most popular clustering algorithm.

Note that: it terminates at a **local optimum** if SSE is used. The **global optimum** is hard to find due to complexity.

Weaknesses of k-means

The algorithm is only applicable if the **mean** is defined.

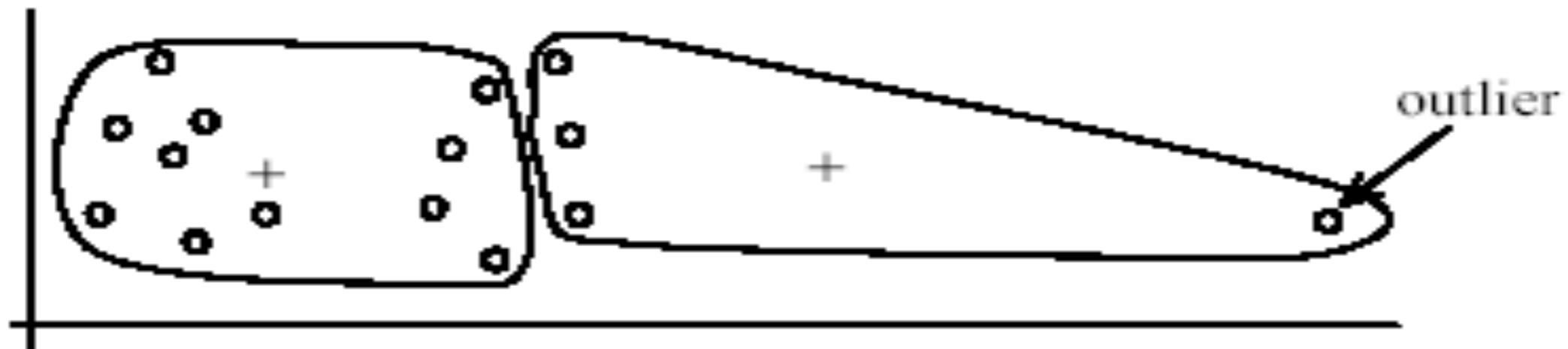
For categorical data, **k-mode** - the centroid is represented by most frequent values.

The user needs to specify **k**.

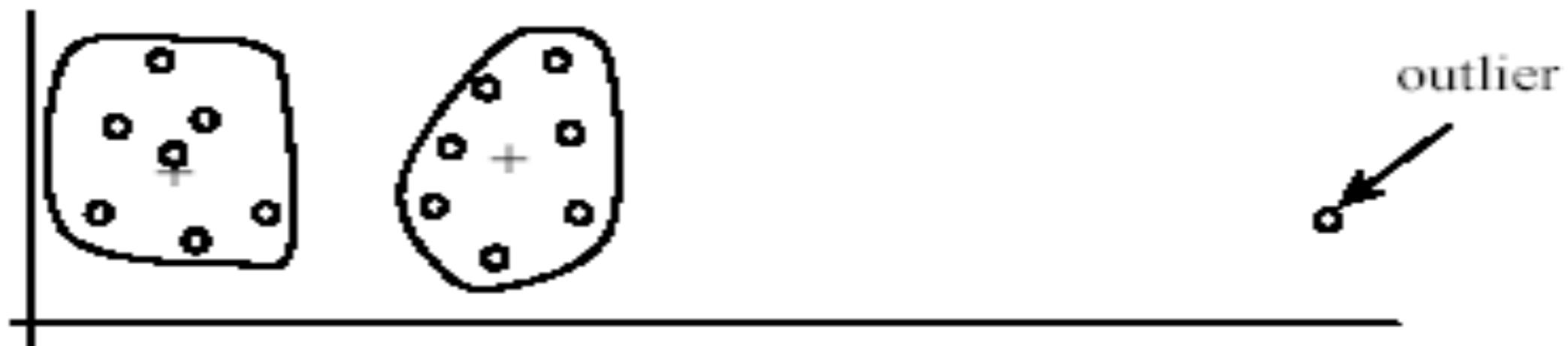
The algorithm is sensitive to **outliers**

- Outliers are data points that are very far away from other data points.
- Outliers could be errors in the data recording or some special data points with very different values.

Weaknesses of k-means: Problems with outliers



(A): Undesirable clusters



(B): Ideal clusters

Weaknesses of k-means: To deal with outliers

One method is to **remove some data points** in the clustering process that are much further away from the centroids than other data points.

To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.

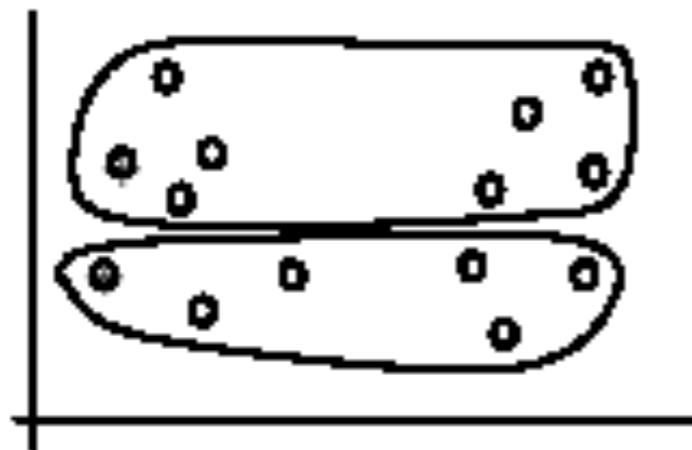
Another method is to **perform random sampling**. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.

- Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

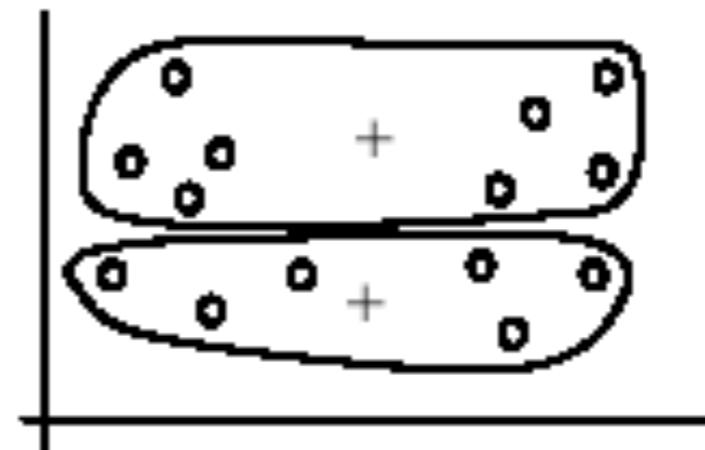
Weaknesses of k-means (cont ...)



(A). Random selection of seeds (centroids)



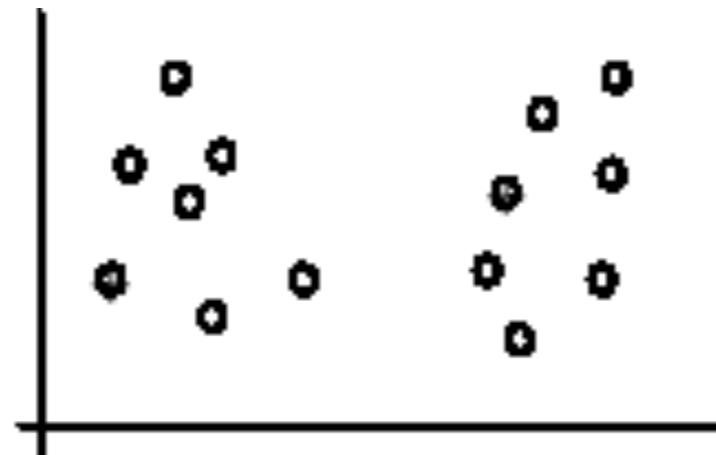
(B). Iteration 1



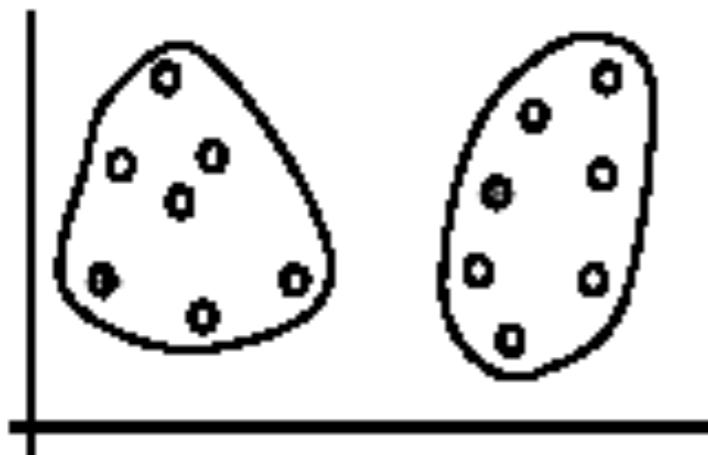
(C). Iteration 2

Weaknesses of k-means (cont ...)

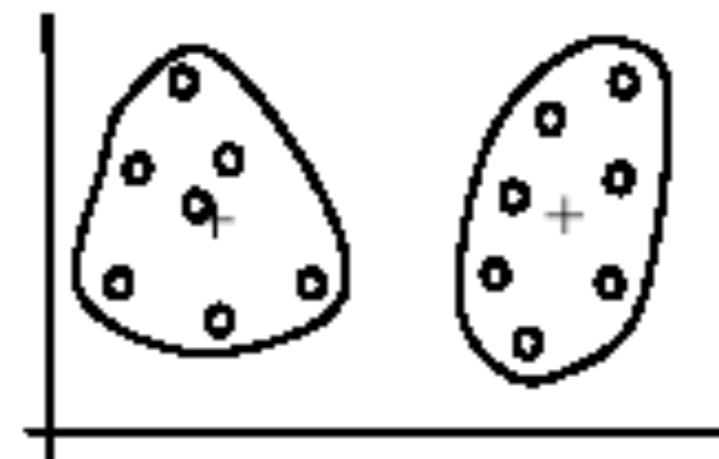
- There are some methods to help choose good seeds



(A). Random selection of k seeds (centroids)



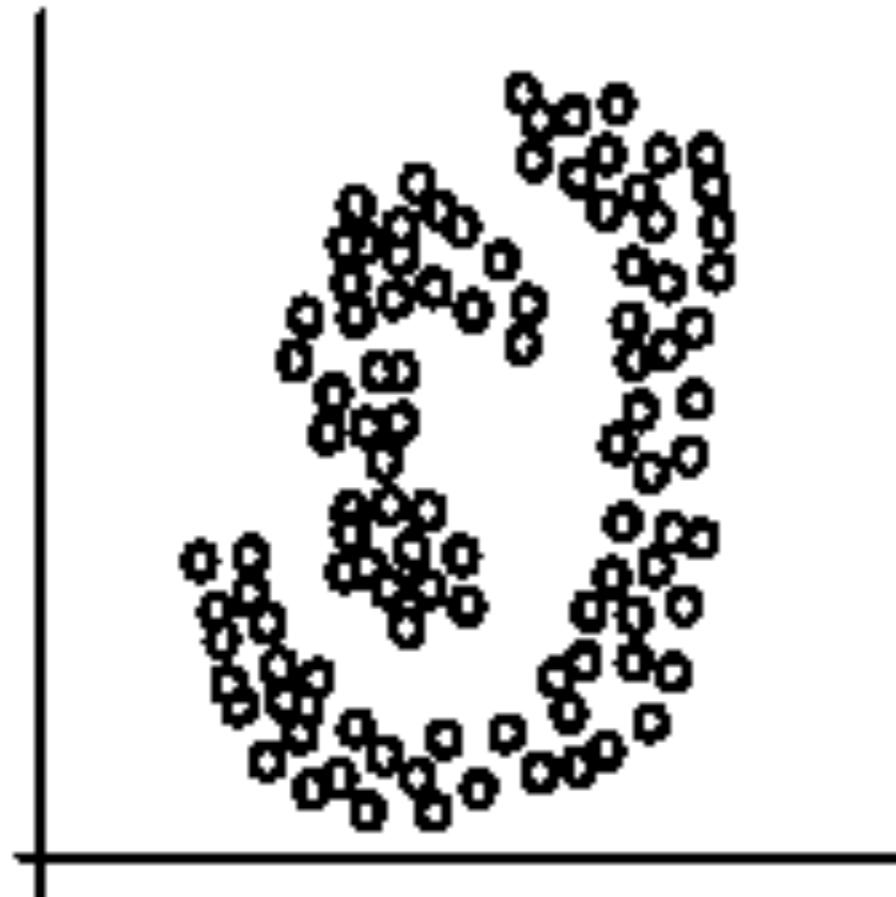
(B). Iteration 1



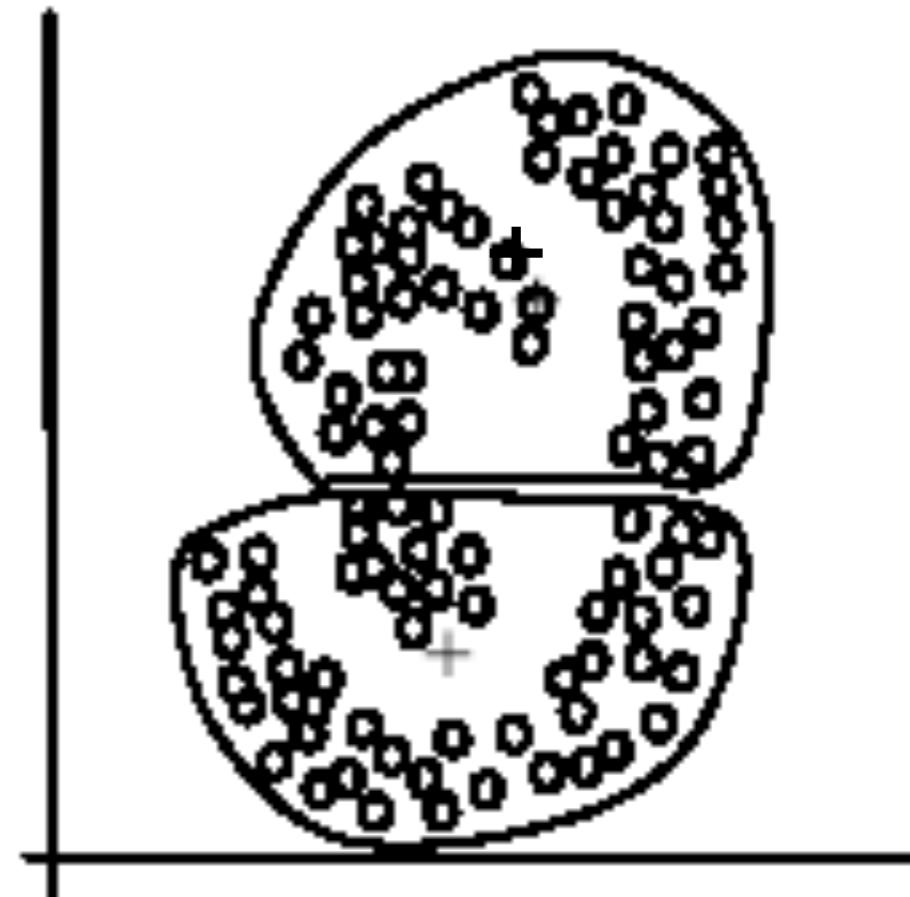
(C). Iteration 2

Weaknesses of k-means (cont ...)

The k -means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).



(A): Two natural clusters



(B): k -means clusters

K-means summary

- Despite weaknesses, k-means is still the most popular algorithm due to its simplicity, efficiency and
 - other clustering algorithms have their own lists of weaknesses.
- No clear evidence that any other clustering algorithm performs better in general
 - although they may be more suitable for some specific types of data or applications.
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!

Common ways to represent clusters

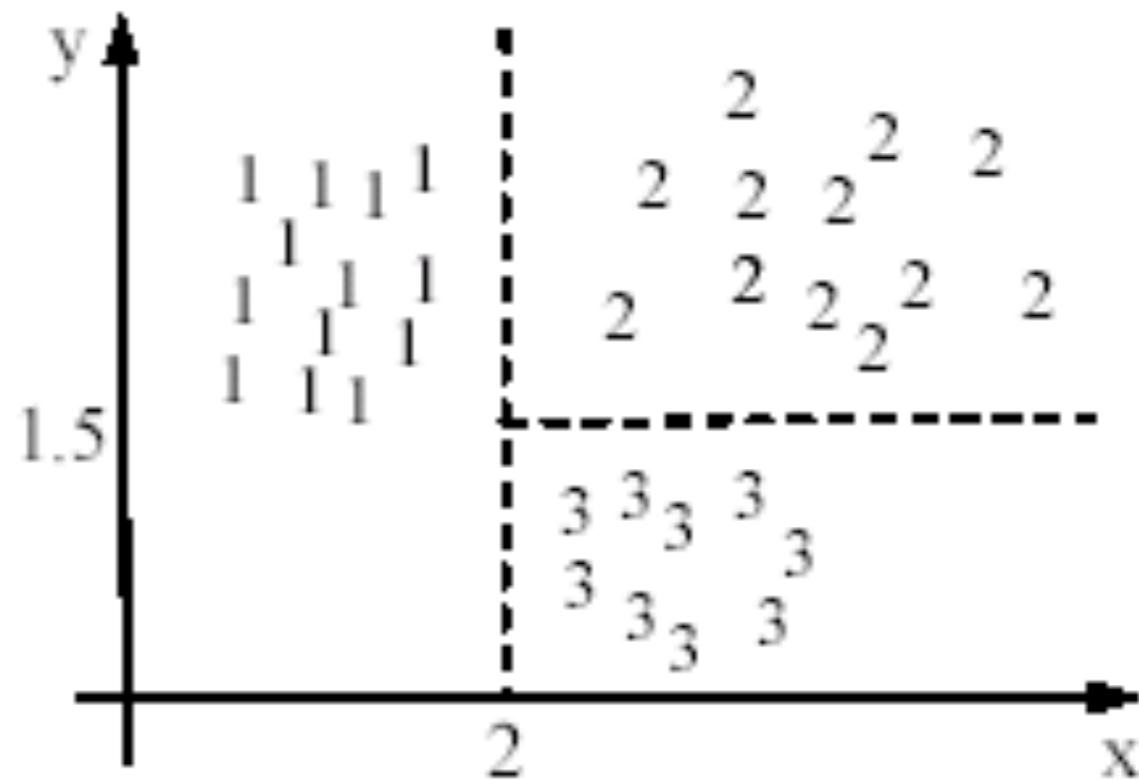
Use the centroid of each cluster to represent the cluster.

- compute the radius and
- standard deviation of the cluster to determine its spread in each dimension
- The centroid representation alone works well if the clusters are of the hyper-spherical shape.
- If clusters are elongated or are of other shapes, centroids are not sufficient

Using classification model

All the data points in a cluster are regarded to have the same class label, e.g., the cluster ID.

run a supervised learning algorithm on the data to find a classification model.



$x \leq 2 \rightarrow$ cluster 1

$x > 2, y > 1.5 \rightarrow$ cluster 2

$x > 2, y \leq 1.5 \rightarrow$ cluster 3

Use frequent values to represent cluster

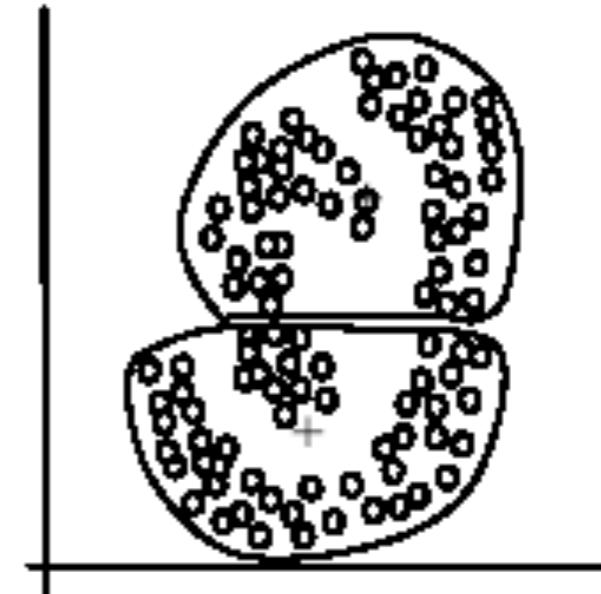
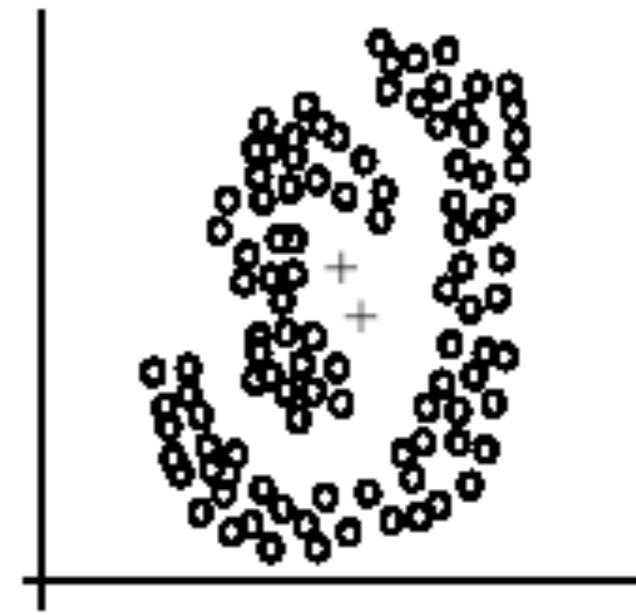
- This method is mainly for clustering of categorical data (e.g., k -modes clustering).
- Main method used in text clustering, where a small set of frequent words in each cluster is selected to represent the cluster.

Clusters of arbitrary shapes

Hyper-elliptical and hyper-spherical clusters are usually easy to represent, using their centroid together with spreads.

Irregular shape clusters are hard to represent. They may not be useful in some applications.

- Using centroids are not suitable (upper figure) in general
- K-means clusters may be more useful (lower figure), e.g., for making 2 size T-shirts.



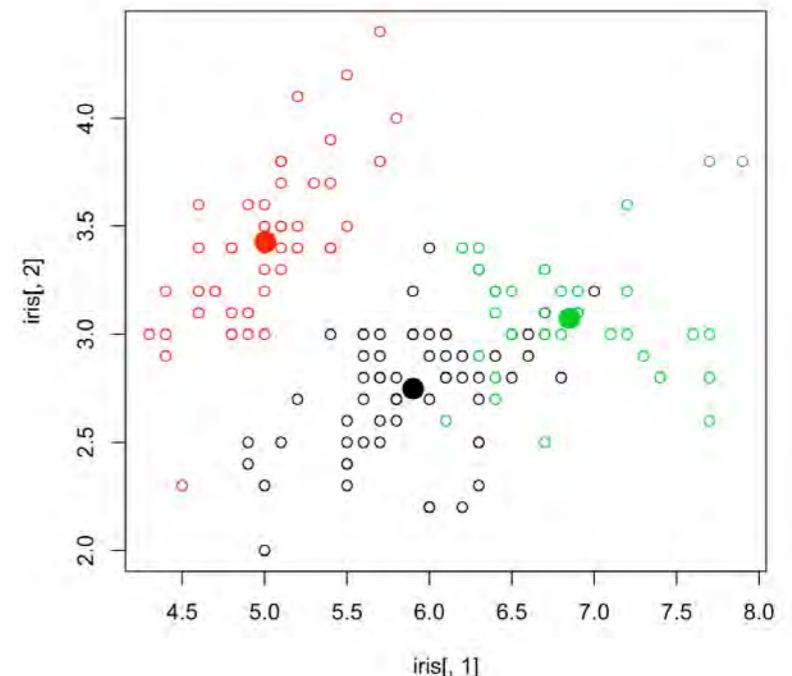
k-Mean in R

Fundamental Data Science for Data Scientist

K-Means Clustering

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

```
library(stats)  
set.seed(101)  
km <- kmeans(iris[,1:4], 3)  
plot(iris[,1], iris[,2], col=km$cluster)  
points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)
```



```
table(km$cluster, iris$Species)
```

	setosa	versicolor	virginica
1	0	48	14
2	50	0	0
3	0	2	36

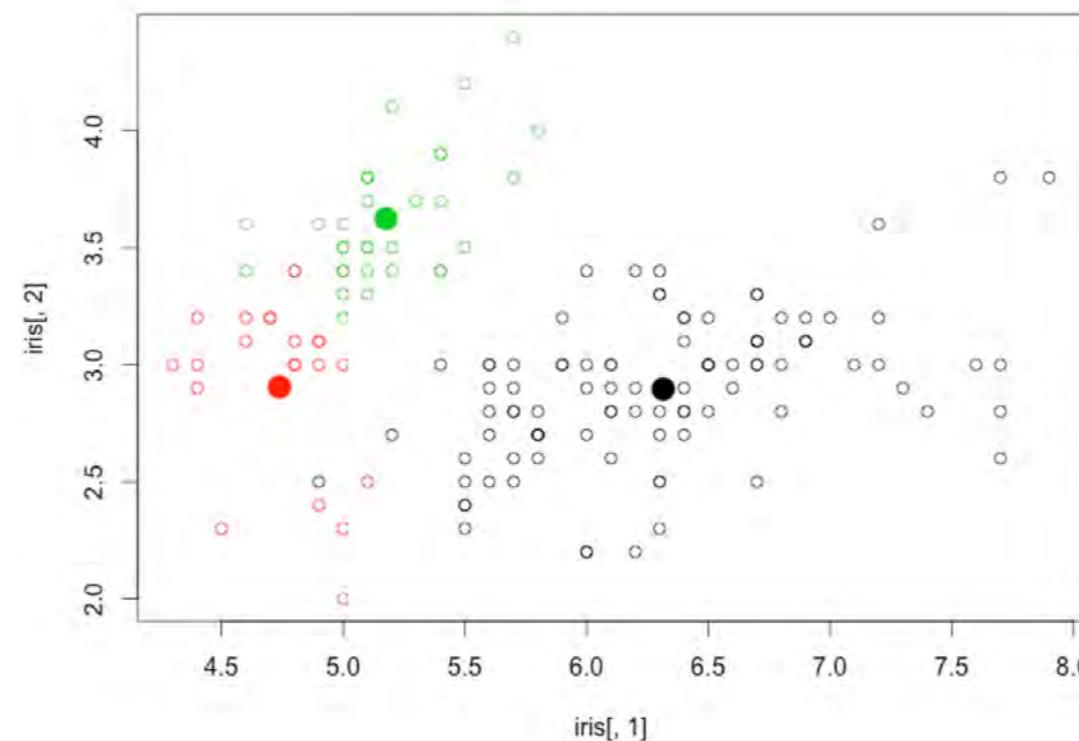
Another round

```
set.seed(900)
```

```
km <- kmeans(iris[,1:4], 3)
```

```
plot(iris[,1], iris[,2], col=km$cluster)
```

```
points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)
```



```
table(km$cluster, iris$Species)
```

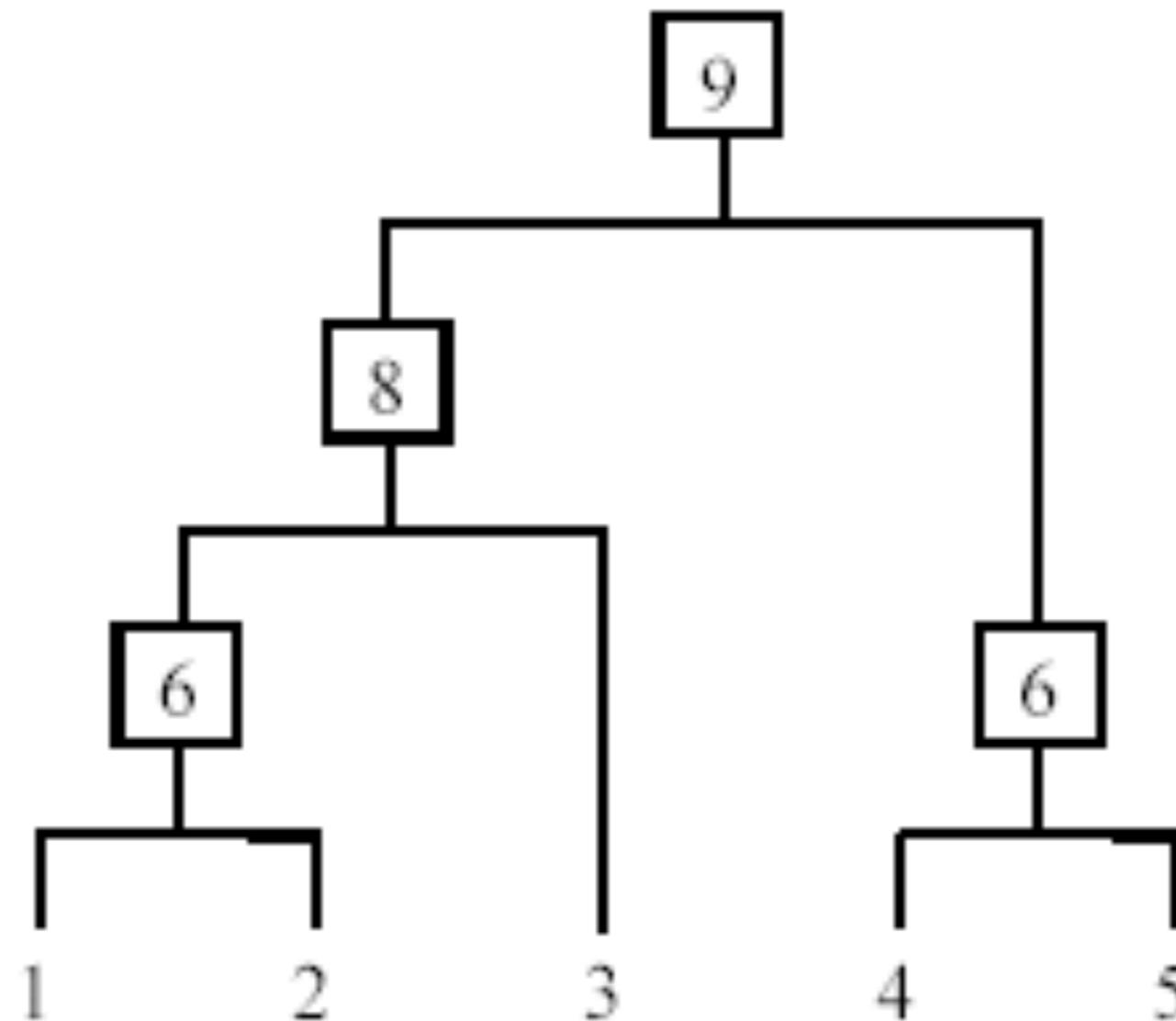
	setosa	versicolor	virginica
1	0	46	50
2	17	4	0
3	33	0	0
>			

Workshop 4.11 - k-Mean Clustering

1. Use data from Workshop 4.2
2. Perform k-Mean clustering to divide data into k groups

Hierarchical Clustering

Produce a nested sequence of clusters, a **tree**, also called **Dendrogram**.



Types of hierarchical clustering

Agglomerative (bottom up) clustering: It builds the dendrogram (tree) from the bottom level, and

- merges the most similar (or nearest) pair of clusters
- stops when all the data points are merged into a single cluster (i.e., the root cluster).

Divisive (top down) clustering: It starts with all data points in one cluster, the root.

- Splits the root into a set of child clusters. Each child cluster is recursively divided further
- stops when only singleton clusters of individual data points remain, i.e., each cluster with only a single point

Agglomerative clustering

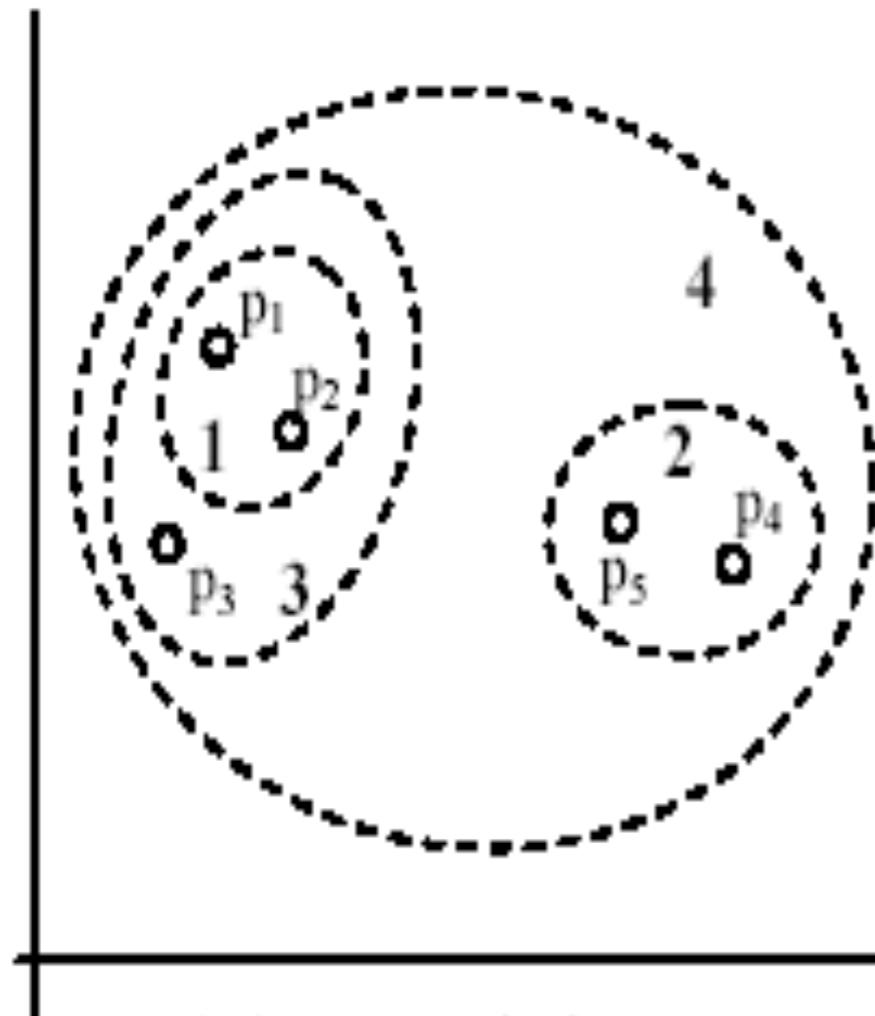
- It is more popular than divisive methods.
- At the beginning, each data point forms a cluster (also called a node).
- Merge nodes/clusters that have the least distance.
- Go on merging
- Eventually all nodes belong to one cluster

Agglomerative clustering algorithm

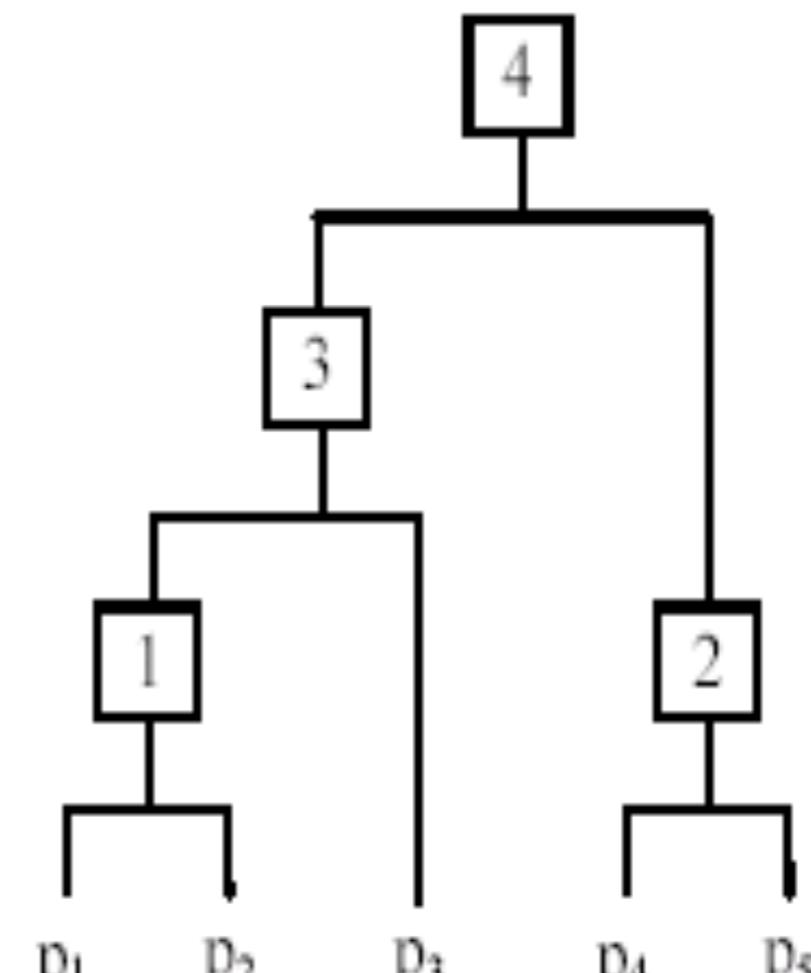
Algorithm Agglomerative(D)

- 1 Make each data point in the data set D a cluster,
- 2 Compute all pair-wise distances of $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in D$;
- 2 **repeat**
- 3 find two clusters that are nearest to each other;
- 4 merge the two clusters form a new cluster c ;
- 5 compute the distance from c to all other clusters;
- 12 **until** there is only one cluster left

An example: working of the algorithm



(A). Nested clusters



(B) Dendrogram

Measuring the distance of two clusters

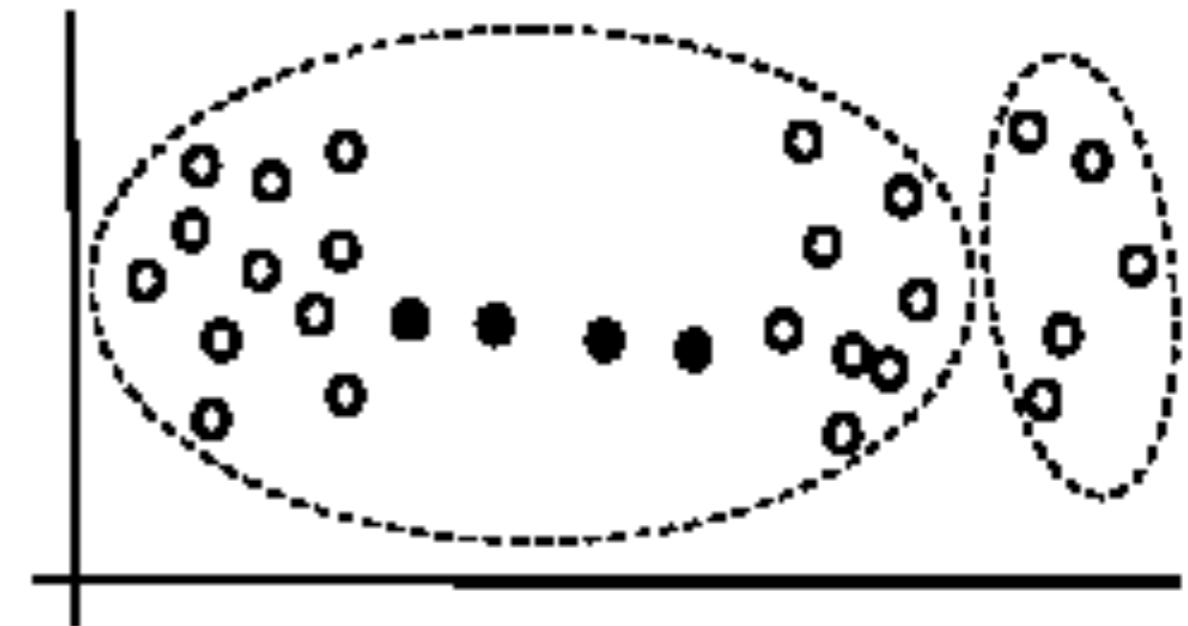
- A few ways to measure distances of two clusters.
- Results in different variations of the algorithm.
 - Single link
 - Complete link
 - Average link
 - Centroids
 - ...

Single link method

The distance between two clusters is the distance between two **closest data points** in the two clusters, one data point from each cluster.

It can find arbitrarily shaped clusters, but

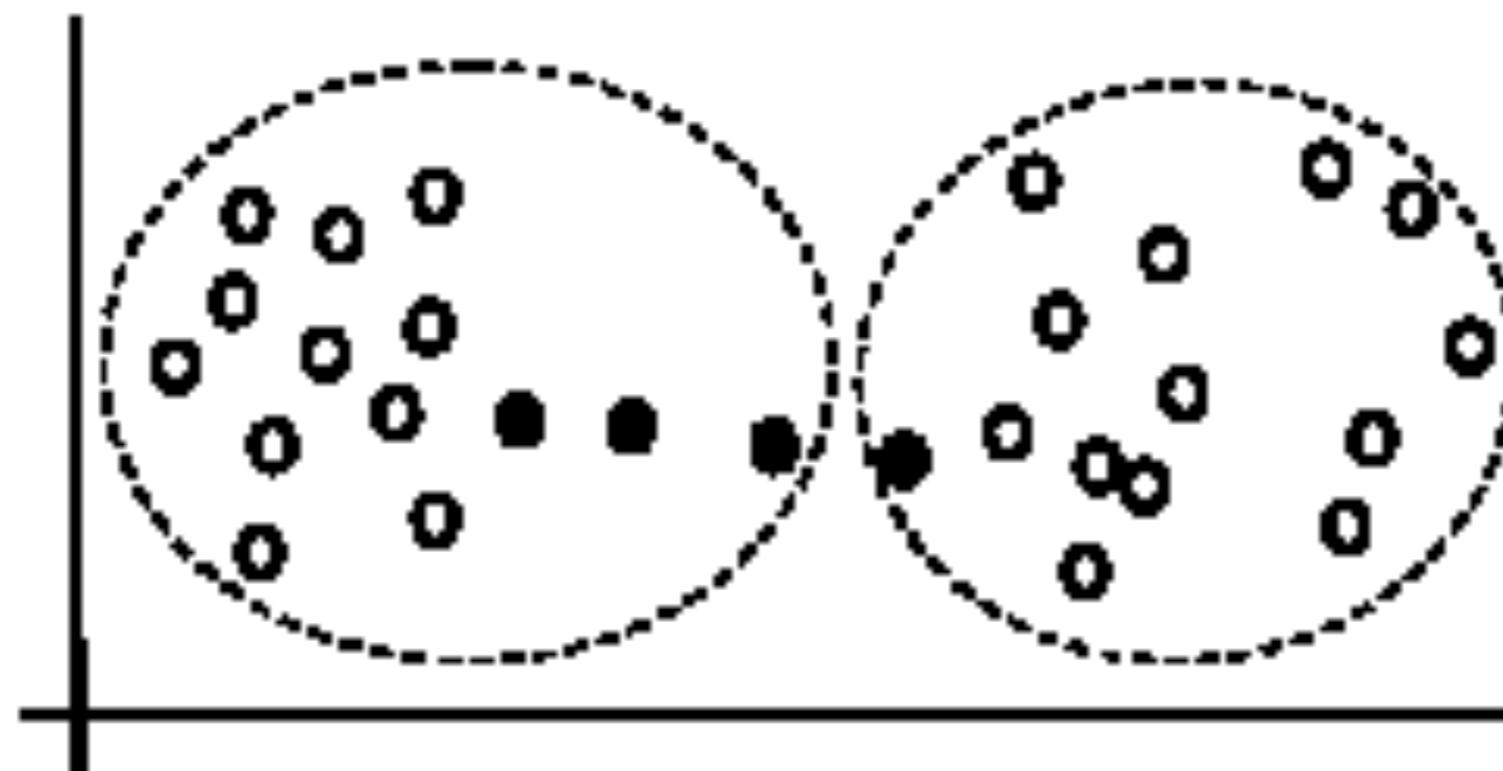
It may cause the undesirable “**chain effect**” by noisy points



Two natural clusters are split into two

Complete link method

- The distance between two clusters is the distance of two **furthest** data points in the two clusters.
- It is sensitive to outliers because they are far away



Average link and centroid methods

Average link: A compromise between

- the sensitivity of complete-link clustering to outliers and
- the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.

In this method, the distance between two clusters is the average distance of all pair-wise distances between the data points in two clusters.

Centroid method: In this method, the distance between two clusters is the distance between their centroids

The complexity

- All the algorithms are at least $O(n^2)$. n is the number of data points.
- Single link can be done in $O(n^2)$.
- Complete and average links can be done in $O(n^2\log n)$.
- Due to the complexity, hard to use for large data sets.
 - Sampling
 - Scale-up methods (e.g., BIRCH).

Distance functions

- Key to clustering. “**similarity**” and “**dissimilarity**” can also commonly used terms.
- There are numerous distance functions for
 - Different types of data
 - Numeric data
 - Nominal data
 - Different specific applications

Distance functions for numeric attributes

- Most commonly used functions are
 - **Euclidean distance** and
 - **Manhattan (city block) distance**
- We denote distance with: $dist(\mathbf{x}_i, \mathbf{x}_j)$, where \mathbf{x}_i and \mathbf{x}_j are data points (vectors)
- They are special cases of **Minkowski distance**. h is positive integer.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \left((x_{i1} - x_{j1})^h + (x_{i2} - x_{j2})^h + \dots + (x_{ir} - x_{jr})^h \right)^{\frac{1}{h}}$$

Euclidean distance and Manhattan distance

If $h = 2$, it is the **Euclidean distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2}$$

If $h = 1$, it is the **Manhattan distance**

$$dist(\mathbf{x}_i, \mathbf{x}_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ir} - x_{jr}|$$

Weighted Euclidean distance

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \dots + w_r(x_{ir} - x_{jr})^2}$$

Squared distance and Chebychev distance

- **Squared Euclidean distance:** to place progressively greater weight on data points that are further apart.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \dots + (x_{ir} - x_{jr})^2$$

- **Chebychev distance:** one wants to define two data points as "different" if they are different on any one of the attributes.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \max(|x_{i1} - x_{j1}|, |x_{i2} - x_{j2}|, \dots, |x_{ir} - x_{jr}|)$$

Distance functions for binary and nominal attributes

- **Binary attribute**: has two values or states but no ordering relationships,
e.g.,
 - Gender: male and female.
- We use a confusion matrix to introduce the distance functions/measures.
- Let the i^{th} and j^{th} data points be \mathbf{x}_i and \mathbf{x}_j (vectors)

Confusion matrix

		Data point j		(10)
		1	0	
Data point i	1	a	b	$a+b$
	0	c	d	$c+d$
		$a+c$	$b+d$	$a+b+c+d$

- a : the number of attributes with the value of 1 for both data points.
- b : the number of attributes for which $x_{if}=1$ and $x_{jf}=0$, where x_{if} (x_{jf}) is the value of the f th attribute of the data point \mathbf{x}_i (\mathbf{x}_j).
- c : the number of attributes for which $x_{if}=0$ and $x_{jf}=1$.
- d : the number of attributes with the value of 0 for both data points.

Symmetric binary attributes

- A binary attribute is **symmetric** if both of its states (0 and 1) have equal importance, and carry the same weights, e.g., male and female of the attribute Gender
- Distance function: [Simple Matching Coefficient](#), proportion of mismatches of their values

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b + c}{a + b + c + d}$$

Symmetric binary attributes: example

\mathbf{x}_1	1	1	1	0	1	0	0
\mathbf{x}_2	0	1	1	0	0	1	0

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{2+1}{2+2+1+2} = \frac{3}{7} = 0.429$$

Asymmetric binary attributes

- **Asymmetric:** if one of the states is more important or more valuable than the other.
 - By convention, state 1 represents the more important state, which is typically the rare or infrequent state.
 - **Jaccard coefficient** is a popular measure

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{b + c}{a + b + c}$$

- We can have some variations, adding weights

Nominal attributes

- **Nominal attributes:** with more than two states or values.
 - the commonly used distance measure is also based on the [simple matching method](#).
 - Given two data points \mathbf{x}_i and \mathbf{x}_j , let the number of attributes be r , and the number of values that match in \mathbf{x}_i and \mathbf{x}_j be q .

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{r - q}{r}$$

Distance function for text documents

- A text document consists of a sequence of sentences and each sentence consists of a sequence of words.
- To simplify: a document is usually considered a “bag” of words in document clustering.
 - Sequence and position of words are ignored.
- A document is represented with a vector just like a normal data point.
- It is common to use similarity to compare two documents rather than distance.
 - The most commonly used similarity function is the **cosine similarity**.

Data standardization

- In the Euclidean space, standardization of attributes is recommended so that all attributes can have equal impact on the computation of distances.
- Consider the following pair of data points
 - $\mathbf{x}_i: (0.1, 20)$ and $\mathbf{x}_j: (0.9, 720)$.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(0.9 - 0.1)^2 + (720 - 20)^2} = 700.000457,$$

- The distance is almost completely dominated by $(720-20) = 700$.
- **Standardize attributes:** to force the attributes to have a common value range

Interval-scaled attributes

- Their values are real numbers following a linear scale.
 - The difference in Age between 10 and 20 is the same as that between 40 and 50.
 - The key idea is that intervals keep the same importance through out the scale
- Two main approaches to standardize interval scaled attributes, **range** and **z-score**. f is an attribute

$$range(x_{if}) = \frac{x_{if} - \min(f)}{\max(f) - \min(f)},$$

Interval-scaled attributes (cont ...)

Z-score: transforms the attribute values so that they have a mean of zero and a **mean absolute deviation** of 1. The mean absolute deviation of attribute f , denoted by s_f , is computed as follows

$$s_f = \frac{1}{n} (|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|)$$

$$m_f = \frac{1}{n} (x_{1f} + x_{2f} + \dots + x_{nf})$$

Z-score:
$$z(x_{if}) = \frac{x_{if} - m_f}{s_f}.$$

Ratio-scaled attributes

- Numeric attributes, but unlike interval-scaled attributes, their scales are exponential,
- For example, the total amount of microorganisms that evolve in a time t is approximately given by
 - Ae^{Bt} ,
 - where A and B are some positive constants.
- Do log transform:
$$\log(x_{if})$$
- Then treat it as an interval-scaled attribute

Nominal attributes

- Sometime, we need to transform nominal attributes to numeric attributes.
- Transform nominal attributes to binary attributes.
 - The number of values of a nominal attribute is v .
 - Create v binary attributes to represent them.
 - If a data instance for the nominal attribute takes a particular value, the value of its binary attribute is set to 1, otherwise it is set to 0.
- The resulting binary attributes can be used as numeric attributes, with two values, 0 and 1.

Nominal attributes: an example

- Nominal attribute *fruit*: has three values,
 - Apple, Orange, and Pear
- We create three binary attributes called, Apple, Orange, and Pear in the new data.
- If a particular data instance in the original data has Apple as the value for *fruit*,
 - then in the transformed data, we set the value of the attribute Apple to 1, and
 - the values of attributes Orange and Pear to 0

Ordinal attributes

- Ordinal attribute: an ordinal attribute is like a nominal attribute, but its values have a numerical ordering. E.g.,
 - Age attribute with values: Young, MiddleAge and Old. They are ordered.
 - Common approach to standardization: treat as an interval-scaled attribute.

Mixed attributes

- Our distance functions given are for data with all numeric attributes, or all nominal attributes, etc.
- Practical data has different types:
 - Any subset of the 6 types of attributes,
 - **interval-scaled**,
 - **symmetric binary**,
 - **asymmetric binary**,
 - **ratio-scaled**,
 - **ordinal** and
 - **nominal**

Convert to a single type

- One common way of dealing with mixed attributes is to
 - Decide the dominant attribute type, and
 - Convert the other types to this type.
- E.g, if most attributes in a data set are interval-scaled,
 - we convert ordinal attributes and ratio-scaled attributes to interval-scaled attributes.
 - It is also appropriate to treat symmetric binary attributes as interval-scaled attributes.

Convert to a single type (cont ...)

- It does not make much sense to convert a **nominal attribute** or an **asymmetric binary** attribute to an interval-scaled attribute, but it is still frequently done in practice by assigning some numbers to them according to some hidden ordering, e.g., prices of the fruits
- Alternatively, a nominal attribute can be converted to a set of (symmetric) binary attributes, which are then treated as numeric attributes.

Combining individual distances

This approach computes individual attribute distances and then combine them.

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{f=1}^r \delta_{ij}^f d_{ij}^f}{\sum_{f=1}^r \delta_{ij}^f}$$

This distance value is between 0 and 1. r is the number of attributes in the data set. The indicator δ_{ij}^f is 1 when both values x_{if} and x_{jf} for attribute f are non-missing, and it is set to 0 otherwise. It is also set to 0 if attribute f is asymmetric and the match is 0-0. Equation (25) cannot be computed if all δ_{ij}^f 's are 0. In such a case, some default value may be used or one of the data points is removed.

d_{ij}^f is the distance contributed by attribute f , and it is in the 0-1 range.



Hierarchical Clustering in R

Data Science Certification

Hierarchical Clustering

Compute distance between every pairs of point/cluster.

- (a) Distance between point is just using the distance function.
- (b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).
- (c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.

Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains

```
set.seed(101)

sampleiris <- iris[sample(1:150, 40),] # get samples from iris dataset

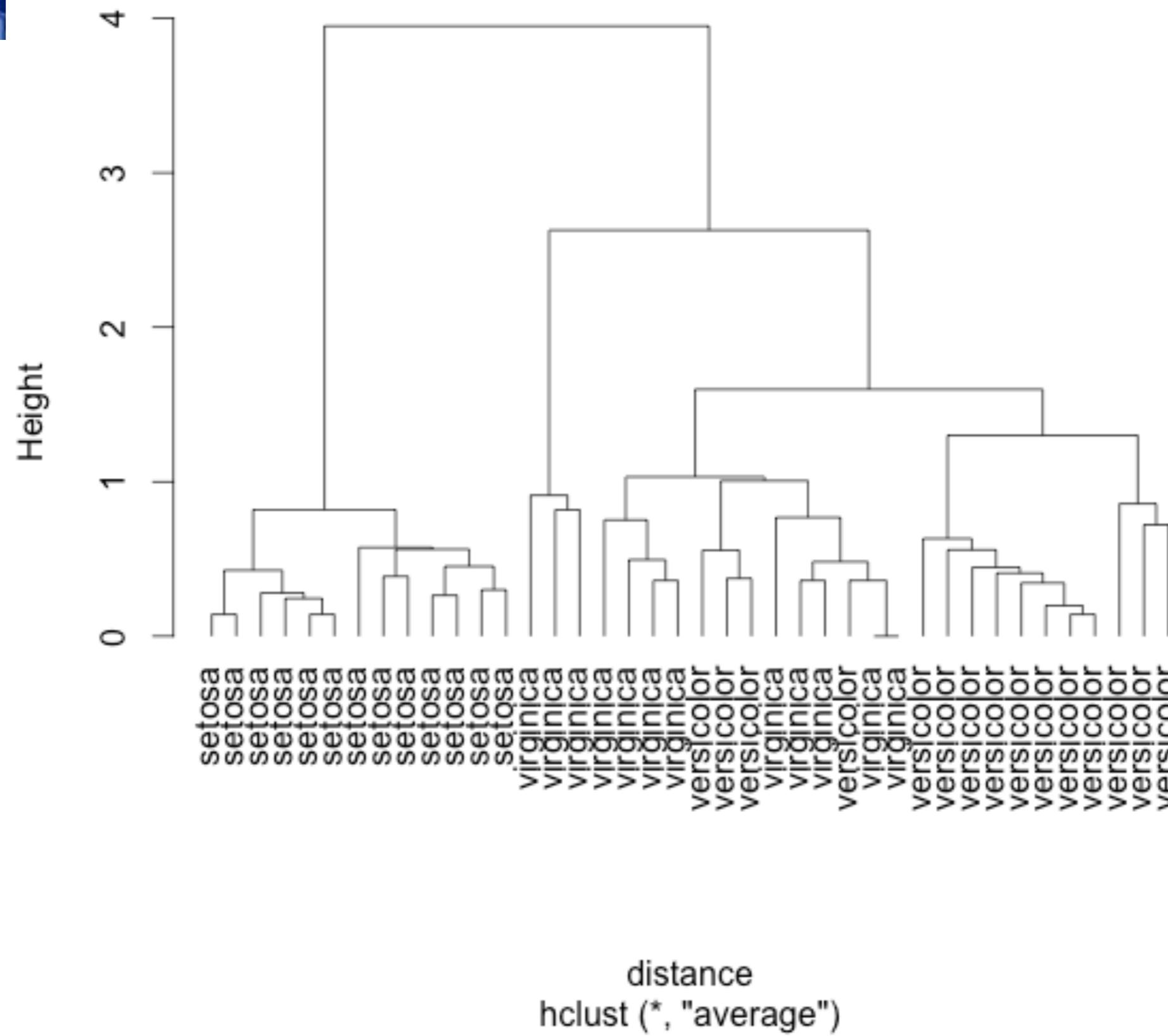
# each observation has 4 variables, ie, they are interpreted as 4-D points

distance <- dist(sampleiris[,-5], method="euclidean")

cluster <- hclust(distance, method="average")

plot(cluster, hang=-1, label=sampleiris$Species)
```

Cluster Dendrogram



It's possible to prune the result tree.

```
par(mfrow=c(1,2))

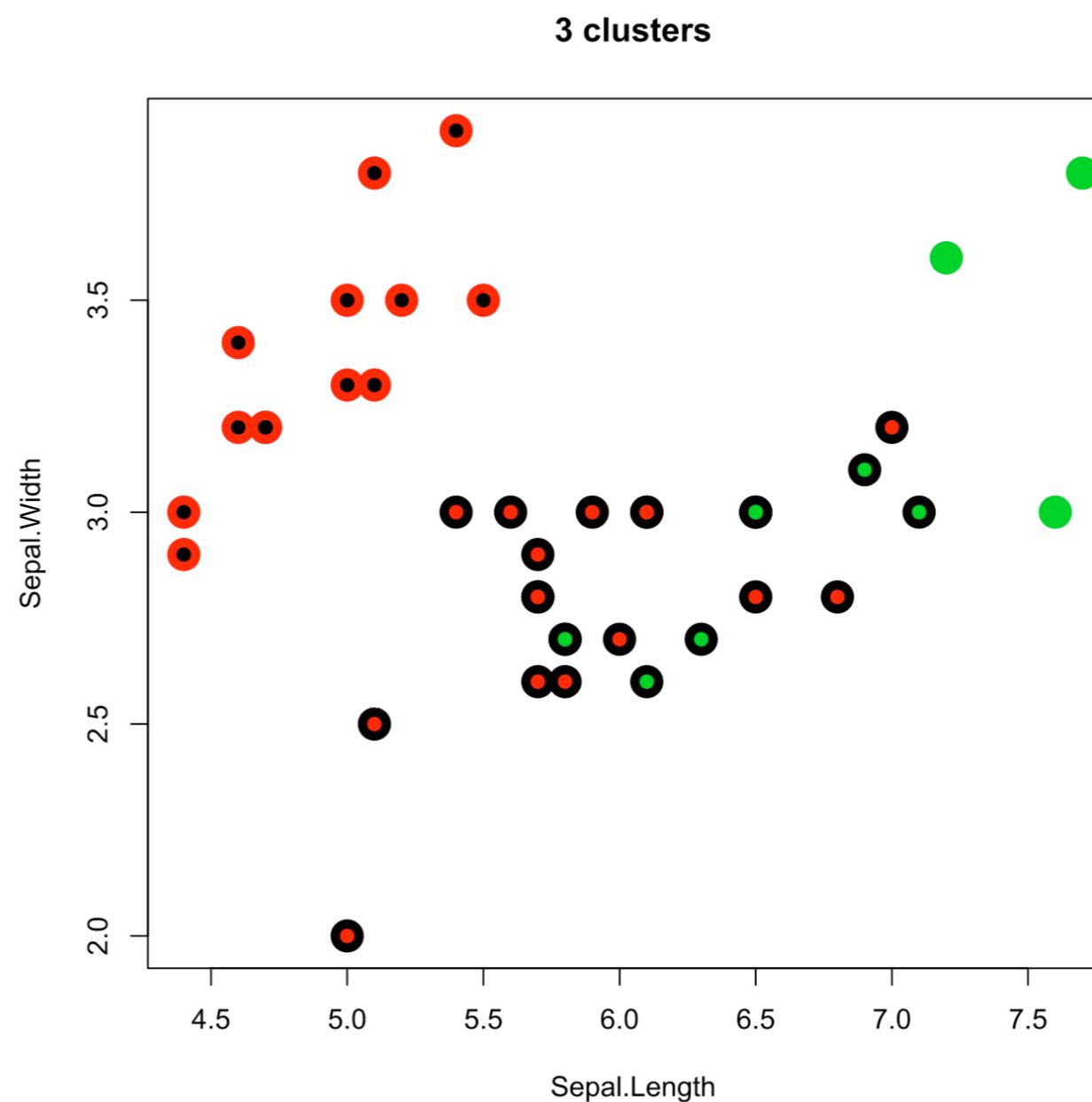
group.3 <- cutree(cluster, k = 3) # prune the tree by 3 clusters

table(group.3, sampleiris$Species) # compare with known classes
```

group.3	setosa	versicolor	virginica
1	0	15	9
2	13	0	0
3	0	0	3

>

```
plot(sampleiris[,c(1,2)], col=group.3, pch=19, cex=2.5, main="3 clusters")
points(sampleiris[,c(1,2)], col=sampleiris$Species, pch=19, cex=1)
```



Workshop 4.12 - Hierarchical Clustering

1. Use data from Workshop 4.2
2. Perform Hierarchical clustering to divide data into 4 groups

How to choose a clustering algorithm

- Clustering research has a long history. A vast collection of algorithms are available.
 - We only introduced several main algorithms.
- Choosing the “best” algorithm is a challenge.
 - Every algorithm has limitations and works well with certain data distributions.
 - It is very hard, if not impossible, to know what distribution the application data follow. The data may not fully follow any “ideal” structure or distribution required by the algorithms.
 - One also needs to decide how to standardize the data, to choose a suitable distance function and to select other parameter values.

Choose a clustering algorithm (cont ...)

- Due to these complexities, the common practice is to
 - run several algorithms using different distance functions and parameter settings, and
 - then carefully analyze and compare the results.
- The interpretation of the results must be based on insight into the meaning of the original data together with knowledge of the algorithms used.
- Clustering is highly **application dependent** and to certain extent **subjective** (personal preferences).

Cluster Evaluation: hard problem

- The quality of a clustering is very hard to evaluate because
 - We do not know the correct clusters
- Some methods are used:
 - User inspection
 - Study centroids, and spreads
 - Rules from a decision tree.
 - For text documents, one can read some documents in clusters.

Cluster evaluation: ground truth

- We use some labeled data (for classification)
- **Assumption:** Each class is a cluster.
- After clustering, a confusion matrix is constructed. From the matrix, we compute various measurements, **entropy, purity, precision, recall and F-score.**
- Let the classes in the data D be $C = (c_1, c_2, \dots, c_k)$. The clustering method produces k clusters, which divides D into k disjoint subsets, D_1, D_2, \dots, D_k .

Evaluation measures: Entropy

Entropy: For each cluster, we can measure its entropy as follows:

$$\text{entropy}(D_i) = -\sum_{j=1}^k \Pr_i(c_j) \log_2 \Pr_i(c_j), \quad (29)$$

where $\Pr_i(c_j)$ is the proportion of class c_j data points in cluster i or D_i . The total entropy of the whole clustering (which considers all clusters) is

$$\text{entropy}_{\text{total}}(D) = \sum_{i=1}^k \frac{|D_i|}{|D|} \times \text{entropy}(D_i) \quad (30)$$

Evaluation measures: purity

Purity: This again measures the extent that a cluster contains only one class of data. The purity of each cluster is computed with

$$purity(D_i) = \max_j(\Pr_i(c_j)) \quad (31)$$

The total purity of the whole clustering (considering all clusters) is

$$purity_{total}(D) = \sum_{i=1}^k \frac{|D_i|}{|D|} \times purity(D_i) \quad (32)$$

An example

Example 14: Assume we have a text collection D of 900 documents from three topics (or three classes), Science, Sports, and Politics. Each class has 300 documents. Each document in D is labeled with one of the topics (classes). We use this collection to perform clustering to find three clusters. Note that class/topic labels are not used in clustering. After clustering, we want to measure the effectiveness of the clustering algorithm.

Cluster	Science	Sports	Politics		Entropy	Purity
1	250	20	10		0.589	0.893
2	20	180	80		1.198	0.643
3	30	100	210		1.257	0.617
Total	300	300	300		1.031	0.711

A remark about ground truth evaluation

- Commonly used to compare different clustering algorithms.
- A real-life data set for clustering has no class labels.
 - Thus although an algorithm may perform very well on some labeled data sets, no guarantee that it will perform well on the actual application data at hand.
- The fact that it performs well on some label data sets does give us some confidence of the quality of the algorithm.
- This evaluation method is said to be based on **external data** or information.

Evaluation based on internal information

- **Intra-cluster cohesion** (compactness):
 - Cohesion measures how near the data points in a cluster are to the cluster centroid.
 - Sum of squared error (SSE) is a commonly used measure.
- **Inter-cluster separation** (isolation):
 - Separation means that different cluster centroids should be far away from one another.
 - In most applications, expert judgments are still the key.

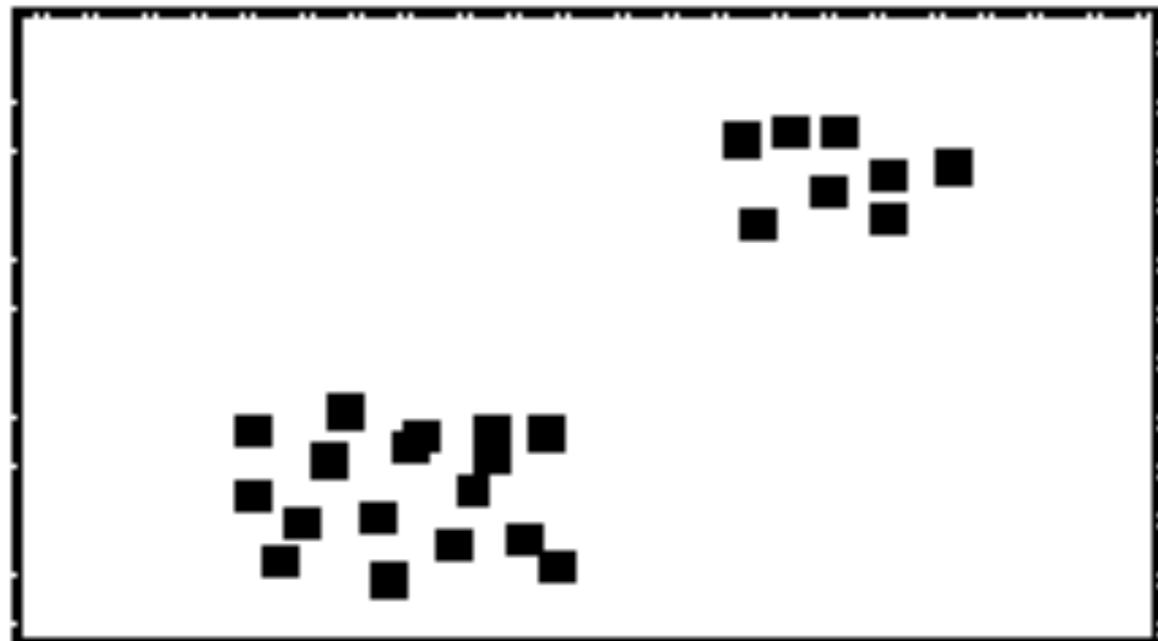
Indirect evaluation

- In some applications, clustering is **not the primary task**, but used to help perform another task.
- We can use the performance on the primary task to compare clustering methods.
- For instance, in an application, the primary task is to provide recommendations on book purchasing to online shoppers.
 - If we can cluster books according to their features, we might be able to provide better recommendations.
 - We can evaluate different clustering algorithms based on how well they help with the recommendation task.
 - Here, we assume that the recommendation can be reliably evaluated.

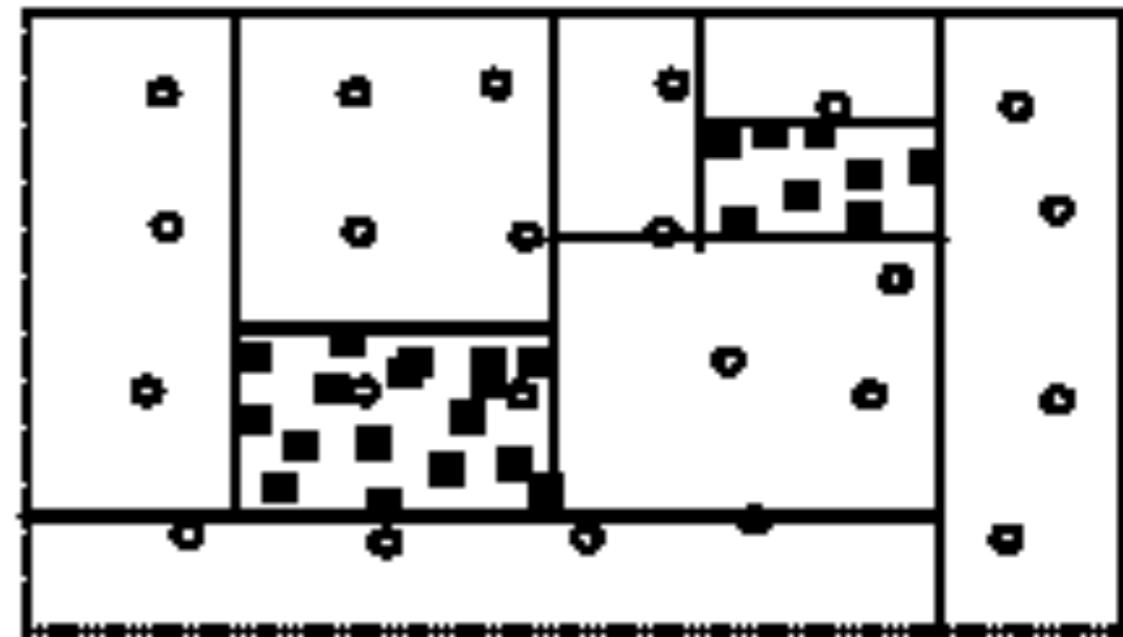
Supervised learning for unsupervised learning

- Decision tree algorithm is not directly applicable.
 - it needs at least two classes of data.
 - A clustering data set has no class label for each data point.
- The problem can be dealt with by a simple idea.
 - Regard each point in the data set to have a class label Y .
 - Assume that the data space is uniformly distributed with another type of points, called **non-existing points**. We give them the class, N .
- With the N points added, the problem of partitioning the data space into data and empty regions becomes a supervised classification problem.

An example



(A): The original data space



(B). Partitioning with added
 N points

- A decision tree method is used for partitioning in (B).

Can it done without adding N points?

Yes.

Physically adding N points increases the size of the data and thus the running time.

More importantly: it is unlikely that we can have points truly uniformly distributed in a high dimensional space as we would need an exponential number of points.

Fortunately, no need to physically add any N points.

We can compute them when needed

Characteristics of the approach

It provides representations of the resulting data and empty regions in terms of **hyper-rectangles**, or **rules**.

It detects outliers automatically. Outliers are data points in an empty region.

It may not use all attributes in the data just as in a normal decision tree for supervised learning.

It can automatically determine what attributes are useful. Subspace clustering ...

Drawback: data regions of irregular shapes are hard to handle since decision tree learning only generates hyper-rectangles (formed by axis-parallel hyper-planes), which are rules.

Building the Tree

The main computation in decision tree building is to evaluate **entropy** (for **information gain**):

$$\text{entropy}(D) = - \sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

Can it be evaluated without adding **N** points? **Yes**.

$\Pr(c_j)$ is the probability of class c_j in data set D , and $|C|$ is the number of classes, Y and N (2 classes).

To compute $\Pr(c_j)$, we only need the number of Y (data) points and the number of N (non-existing) points.

We already have Y (or data) points, and we can compute the number of N points on the fly. Simple: as we assume that the N points are uniformly distributed in the space.

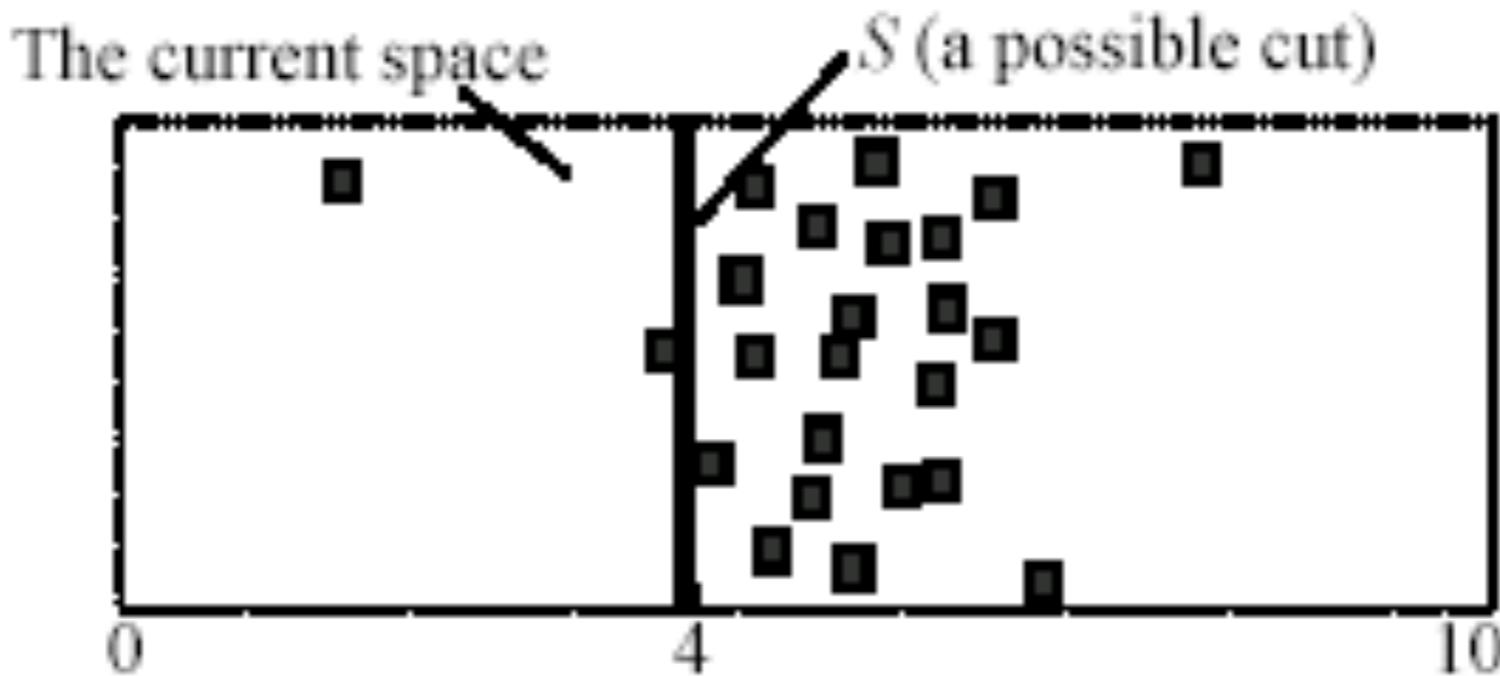
An example

The space has 25 data (Y) points and 25 N points. Assume the system is evaluating a possible cut S .

N points on the left of S is $25 * 4/10 = 10$. The number of Y points is 3.

Likewise, # N points on the right of S is 15 (= $25 - 10$). The number of Y points is 22.

With these numbers, entropy can be computed.



How many N points to add?

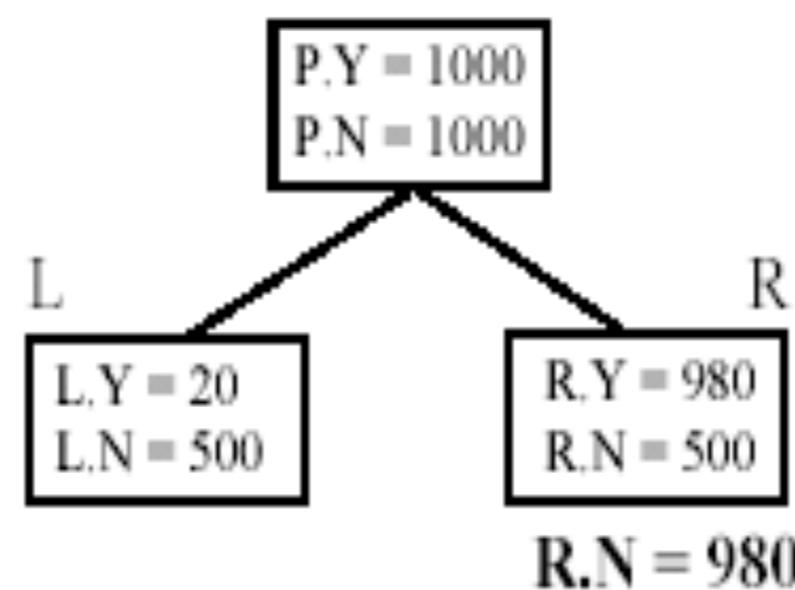
We add a different number of N points at each different node.

The number of N points for the current node E is determined by the following rule (note that at the root node, the number of inherited N points is 0):

- 1 If the number of N points inherited from the parent node of E is less than the number of Y points in E **then**
- 2 the number of N points for E is increased to the number of Y points in E
- 3 **else** the number of inherited N points is used for E

An example

Example 17: Fig. 20 gives an example. The (parent) node P has two children nodes L and R . Assume P has 1000 Y points and thus 1000 N points, stored in $P.Y$ and $P.N$ respectively. Assume after splitting, L has 20 Y points and 500 N points, and R has 980 Y points and 500 N points. According to the above rule, for subsequent partitioning, we increase the number of N points at R to 980. The number of N points at L is unchanged.



How many N points to add? (cont...)

Basically, for a Y node (which has more data points), we increase N points so that

$$\#Y = \#N$$

The number of N points is not reduced if the current node is an N node (an N node has more N points than Y points).

A reduction may cause outlier Y points to form Y nodes (a Y node has an equal number of Y points as N points or more).

Then data regions and empty regions may not be separated well.

Building the decision tree

Using the above ideas, a decision tree can be built to separate data regions and empty regions.

The actual method is more sophisticated as a few other tricky issues need to be handled in

tree building and

tree pruning.

Summary

- Clustering is has along history and still active
 - There are a huge number of clustering algorithms
 - More are still coming every year.
- We only introduced several main algorithms. There are many others, e.g.,
 - density based algorithm, sub-space clustering, scale-up methods, neural networks based methods, fuzzy clustering, co-clustering, etc.
- Clustering is hard to evaluate, but very useful in practice. This partially explains why there are still a large number of clustering algorithms being devised every year.
- Clustering is highly application dependent and to some extent subjective.

Recommendation System

- Concept
- Building Recommendation System



You may also like

14% OFF

SKU: 77372



COTTO

ฟลัชวาล์วโถชายก่อตรง COTTO
CT475(SS)

1,690 **1,450 บาท**

16% OFF

SKU: 77016



COTTO

ฟลัชวาล์วโถชายก่อโค้ง COTTO
CT474NS

1,790 **1,490 บาท**

Let's Start with Terminology of recommendation

- **Users** – People that use the system and receive recommendations. Users also provide the ratings for items.
- **Items** – The items that are being recommended (e.g. movies, products, hotels, etc.).
- **Ratings** – Ratings refer to the choices of users in relation to items. Ratings can be explicit, by e.g. tagging a product, or implicit (e.g. opening a document, buying a product). Ratings can be Boolean, ordinal or numeric in nature, requiring different algorithms in implementation.
- **Content** – The data from within the items that can be analyzed for recommendation. When documents are recommended, typically the

Let's Start with Terminology of

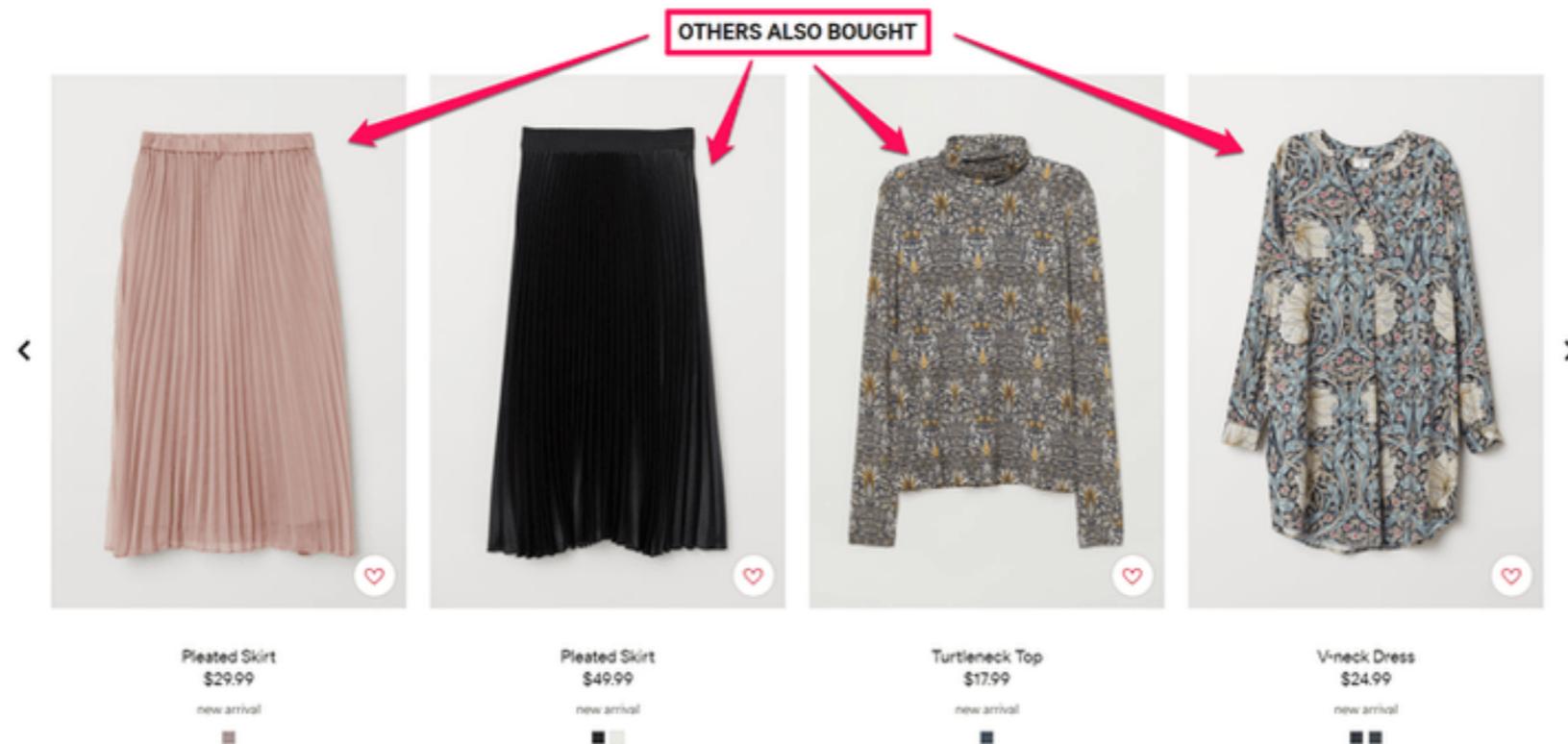
- **Context** – The sum of all contextual factors that influence the use and evaluation of the recommender system and their interactions.
- **Sparse Matrix** – A matrix that contains mostly the value 0. In user-item matrices, we often have many users and many items and only few ratings for individual users and items.
- **Cold start problem** – When a new item or new user enters the system, we have very little information on the user to base recommendations on.
- **Coverage** – Coverage refers to the criterion that addresses, whether all items in the database are getting recommended [6]. Recommenders that only recommend the most bought items, reach low coverage.

What is a recommendation system?

- A **recommendation system** is a type of **information filtering system**. By drawing from huge data sets, the system's algorithm can pinpoint accurate user preferences. Once you know what your users like, you can recommend them new, relevant content. And that's true for everything from movies and music, to romantic partners.
- Netflix, YouTube, Tinder, Amazon use recommendation systems to entice users with relevant suggestions.

What is a recommendation system?

Examples





5 Types of Recommenders

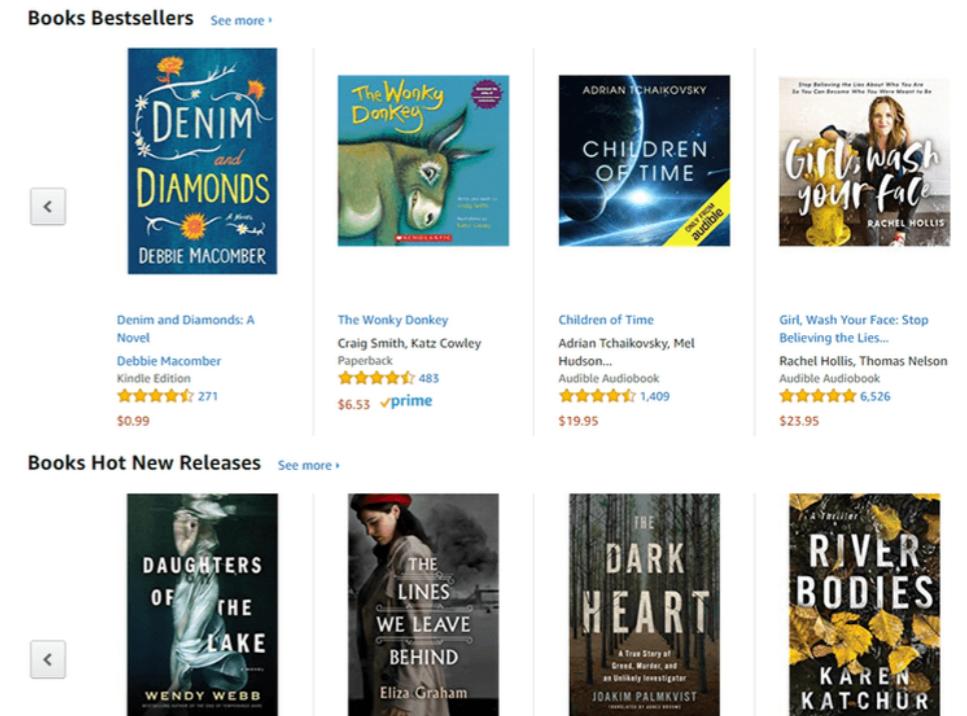
1. Most Popular Items
2. Association and Market Basket Analysis
3. Content Filtering
4. Collaborative Filtering
5. Hybrid Models

Most Popularity Model

- a common base line approach is the Popularity model. This is not actually personalized. It simply recommends to a user the most popular items that the user has not previously consumed.
- The main objective of a recommender system is to leverage the long-tail items to the users with very specific interests, which goes far beyond this simple technique.

Most Popularity Model

- a common base line approach is the Popularity model. This is not actually personalized. It simply recommends to a user the most popular items that the user has not previously consumed.



- The main objective of a recommender system is to leverage the long-tail items to the users with very specific interests, which goes far beyond this simple technique.

Most Popularity Model

- a common base line approach is the Popularity model. This is not actually personalized.
- It simply recommends to a user the most popular items that the user has not previously consumed.
- The main objective of a recommender system is to leverage the long-tail items to the users with very specific interests, which goes far beyond this simple technique.
- Home Depot and GAP in US use 'best sellers' to increase revenue.
- Use it as a supplementary strategy

Association or Market Basket Analysis

- Use content exclusively as input for recommendation.
- Association and Market Basket Analysis use the same algorithm.
- Association conducts analysis at customer level MBA conducts at transaction level (what's in their basket).



Association or Market Basket Analysis - Steps

- Evaluate the strength of relationship between each of products and every other products using the algorithms of Association math.
- Identify pairings that have strong affinity score.
- Create a personalized offering for customer who have one product of a strongly associated pair.

Association or Market Basket Analysis - Advantage

- Extremely simple and fast
- Work with small customer bases
- Knowledge of customer beyond what product they have is not necessary
- Data preparation is minimal.
- We often found it under “Customer who brought this also brought these” or “Items bought together” from Amazon.

Frequently bought together



Total price: \$29.15
[Add all three to Cart](#)
[Add all three to List](#)

Best Value

Buy [Philips Norelco BG2020 Men's Bodygroom](#) and get [MANGROOMER Do-It-Yourself Electric Back Hair Shaver](#) \$5.00 off Amazon.com's everyday low price.

Total List Price: \$79.98
Buy Together Today: \$69.94
You Save: \$10.04
[Buy both and save](#)

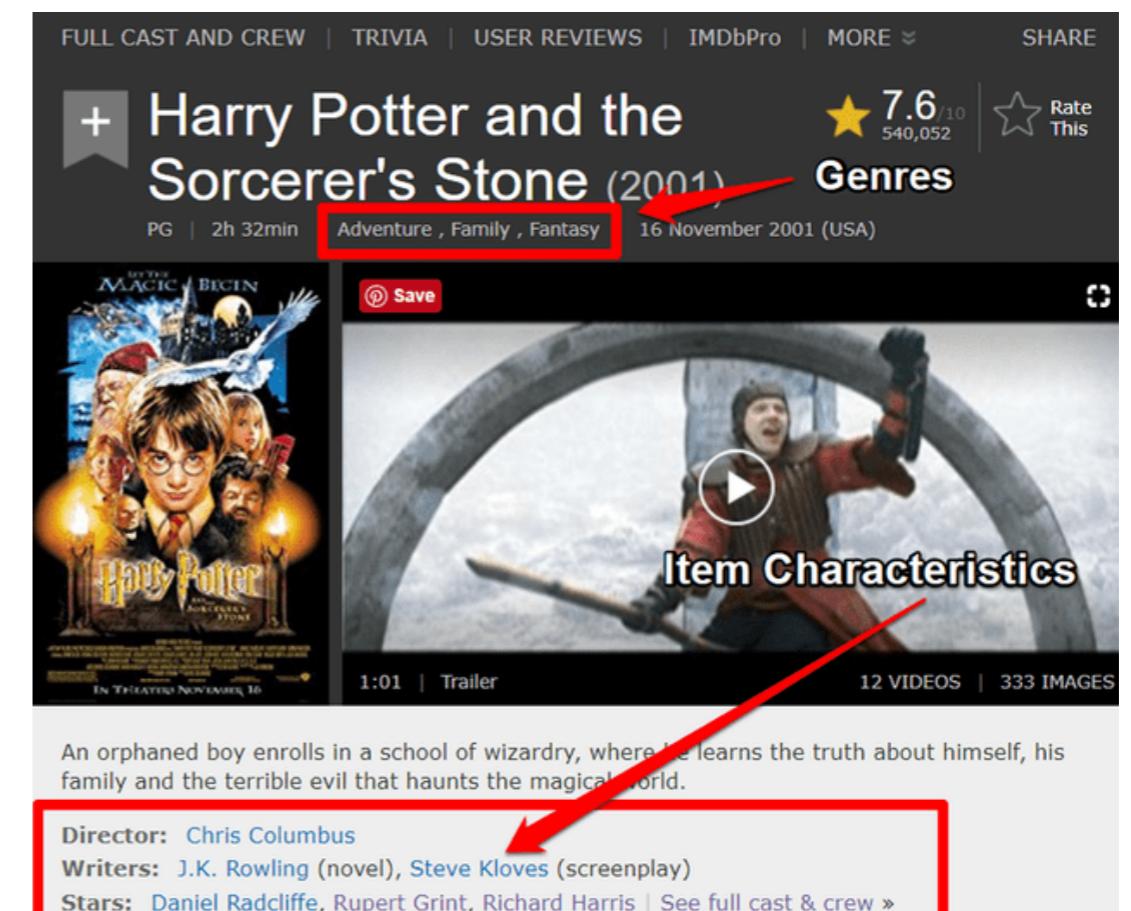
Customers who bought this item also bought

 MANGROOMER Do-It-Yourself Electric Back Hair Shaver ★★★★★ \$39.95	 Panasonic ER421KC Nose and Ear Hair Trimmer, Wet/Dry, Lighted ★★★★★ \$14.99	 Tweezerman Deluxe Men's Grooming Kit \$19.99	 Tweezerman Stainless Steel Ingrown Hair Splintertweeze ★★★★★ \$8.97
--	--	--	--

▶ [Explore similar items : Health & Personal Care \(21\) DVD \(18\) Music \(3\) Books \(2\)](#)

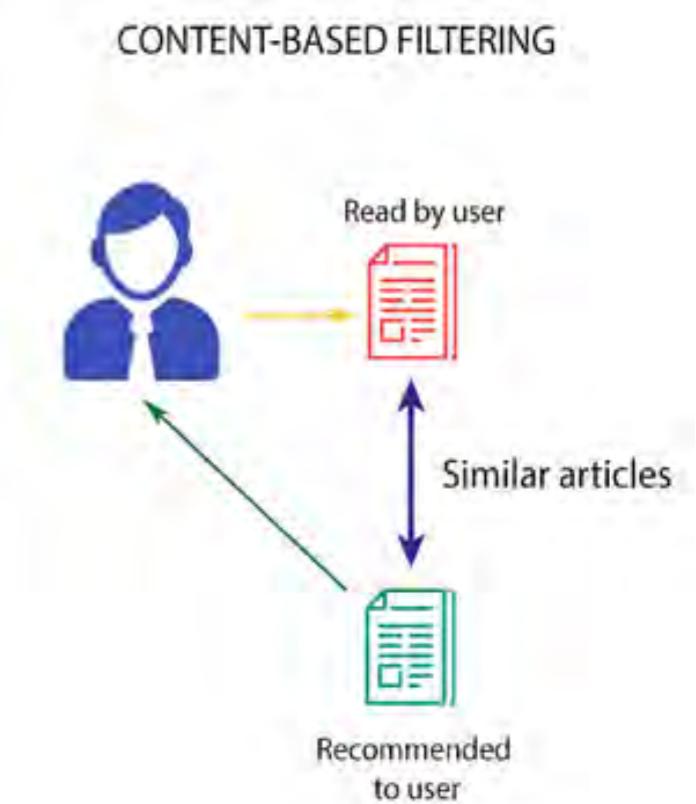
Content Filtering (CF)

- CF looks for similarities between items the customer has consumed or browsed in the past to present options in the future.
- CF are user-specific classifiers that learn to positively or negatively categorize alternatives based on the user's likes or dislikes (the user profile)



Content Filtering (CF) - Step

- Algorithm create a user-specific content-based profile using discrete attributes.
- User's history of consumption or browsing is used to create weighted vector of item features.
- Weights are learned or assigned to vary the importance of attributes for the particular user.
- That weight is used to compare to the vector weight of different items that might be recommended.



Content Filtering (CF) - Item Attributes

- Requirement is to provide a reasonably large number of content descriptors to use in the classification.
- These can be **Boolean** (the movie is “**animation**”, the book author is “**Michael Crichton**”, the shirt material is “**cotton**”)
- Or **Numeric** (The “**rating**” received by the product or rating source, “**average star rating**” of other customers or percentage of number of minutes in the movie considered “**action**” or “**comedy**”)

Content Filtering (CF) - Item Attributes

- In large-value high-turnover environments like movies, music and news, maintaining those attributes could be challenging.
- Pandora uses external data sources to populate over 4000 attributes for songs and artists provided by Music Genome Project.
- Rotten Tomatoes use this algorithm as well.

Content Filtering (CF) - How to enhance performance

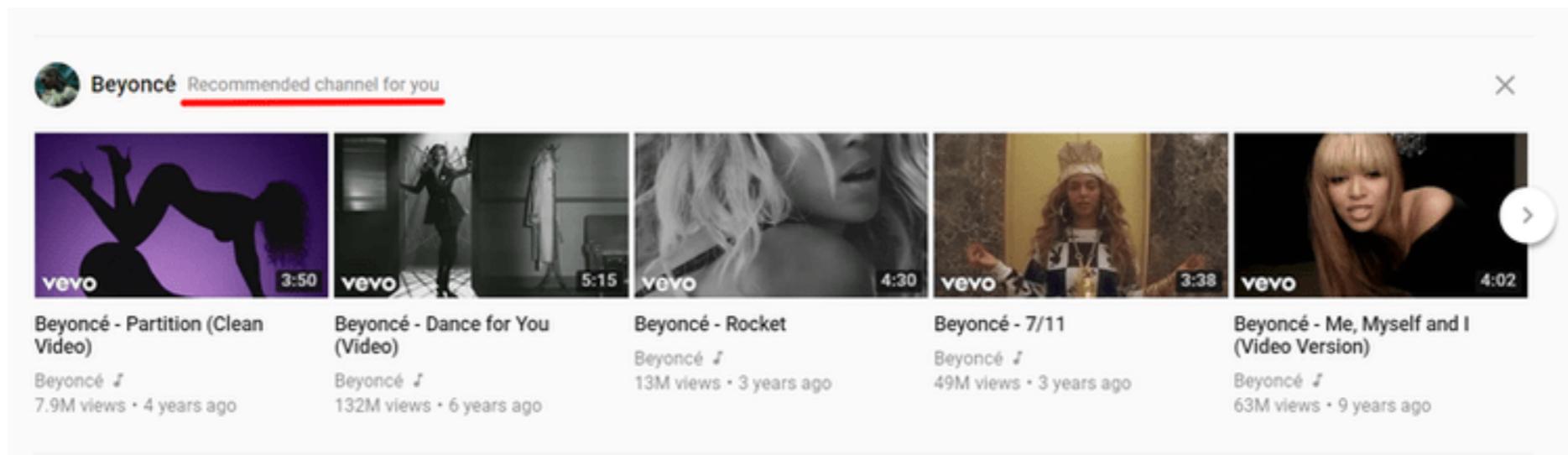
- If there is no attributes other than purchased/not purchased, the recommendation result would not be accurate.
- Solutions to increase accuracy are
 - **Post Suggestion Rating** : The user is given the option of using a like/dislike button even if they are not purchased or selected.
 - **Post purchase rating** : Typical 1-5 regarding to their satisfaction with item.
 - **Pre-consumption voluntary profile** : ask the user in advance to provide some profile information regarding preference aids
 - **User profiling**: The user profile can be enhanced with segmentation or demographic.

Content Filtering (CF) - Strength and Weaknesses

- Solve the “cold start” problem
- Enhanced profiles may not be equal for all users
- Only one single source is considered.
- Without expanded user profiling or feed back, CF has little to use for calculation.

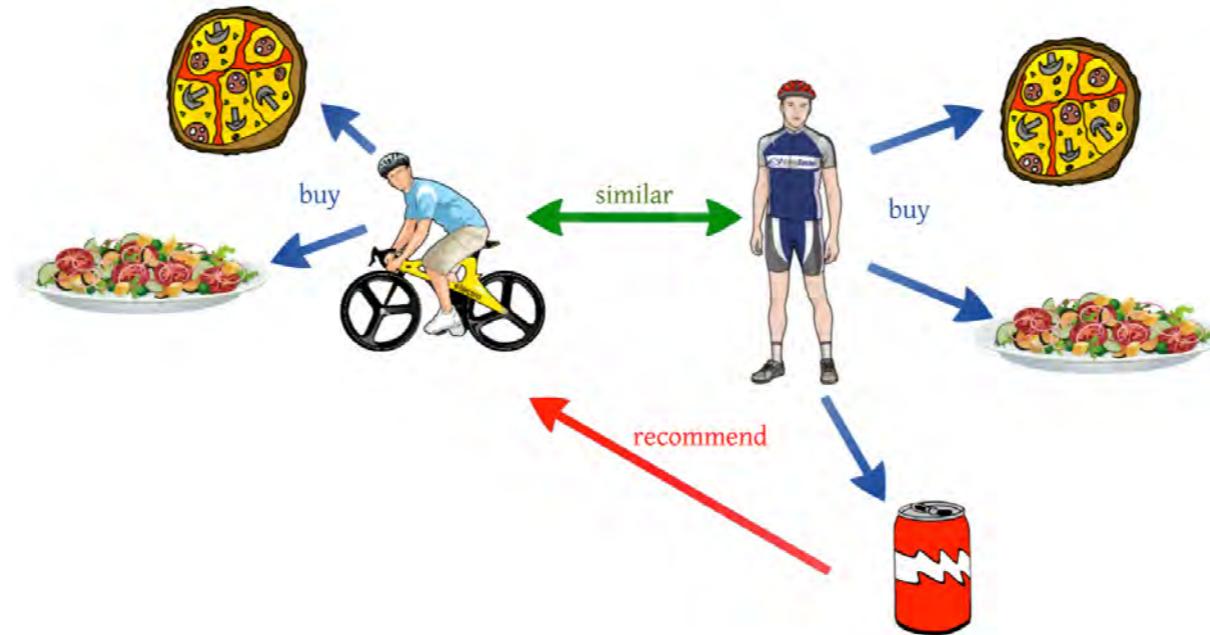
Collaborative Filtering (CB)

- Focus on the user and other users found to be similar to the user.
- No specific attributes are required for the content.
- Algorithm looks if 2 users have a strong similarity of likes and dislikes in the past that they will continue to have strong similarity in the future.
- CB can also predict post-selection rating, like/dislike if there is.



Collaborative Filtering (CB)

- Focus on the user and other users found to be similar to the user.
- No specific attributes are required for the content.
- Algorithm looks if 2 users have a strong similarity of likes and dislikes in the past that they will continue to have strong similarity in the future.
- CB can also predict post-selection rating, like/dislike if there is.



Collaborative Filtering (CB) - The algorithm

- Begin by creating a feature vector describing the user (products and features identified as interesting, size and frequency of prior purchases etc)
- Cosine similarity calculations are made against the feature vectors to identify similar customers and similar products.
- Recommendation can be made based on either the **similarity of the customers** or the **similarity of products** the customer has purchased or browsed.
- Customer based factorization use linear algebra. Product based factorization is based on **clustering** or Pearson's **correlation**

Collaborative Filtering (CB) - User Attributes

- CB systems rely on two types of data. The first is by user including:
 - Rating an item on a five point scale or using like/dislike button
 - Create a profile of likes and dislikes by different factors such as genre, author or similar factors.
 - Presenting two alternatives and asking the user which is better (conjoint analysis)
- The secondary user's actual on-line behavior including:
 - Observing items viewed including time spent viewing.
 - Actual purchases.
 - Using outside data source like Social Network to add like/dislike for this user.

CB - Strengths and Weaknesses

- CBs do not rely on predefined content attributes and can make recommendations for different categories of content without requiring a human-designated understanding of the item.
- CBs require large data to analyze.
- Item that are new and have never been rated cannot be recommended since recommendation relies on prior rating.
- CBs tend to recommend items that are popular.
- Items can be processed separately.

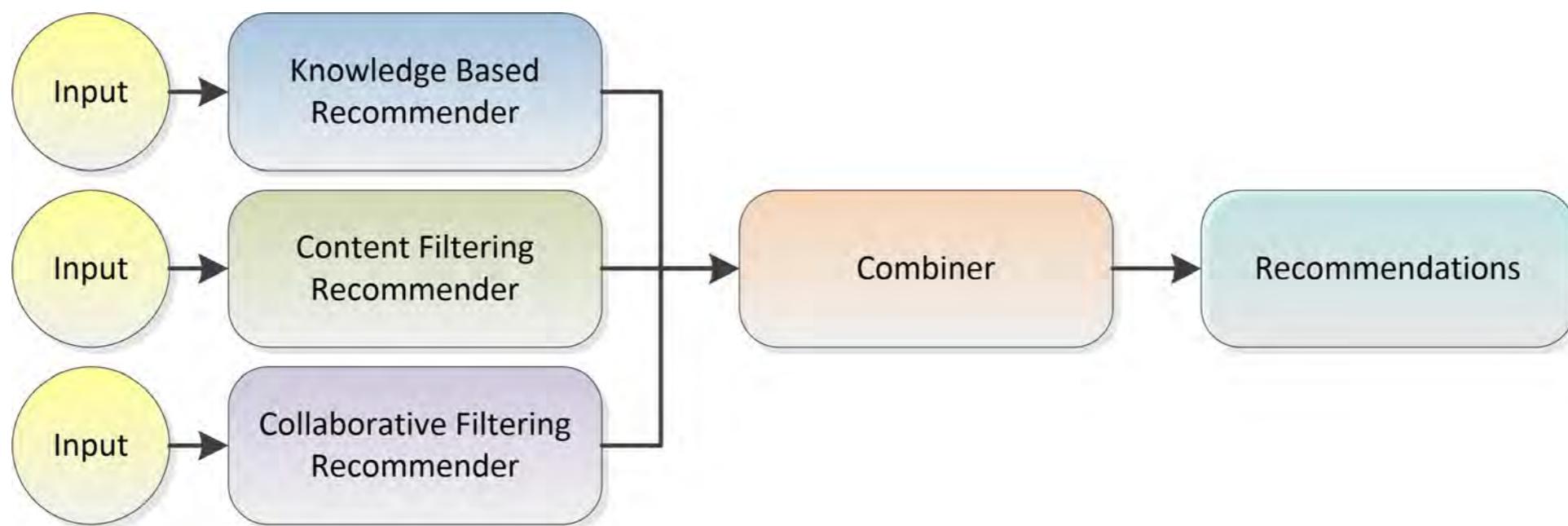
Hybrid Model

- Two type of hybrid recommendation
 - 1. Brute Force or Knowledge Based
 - 2. Combined CF/CB systems

Brute Force or Knowledge Based

- Easy to understand since it involves the addition of rules by human subject matter experts.
- Product manager can define what product do and do not go together and may be complementary versus supplementary

Combined CF/CB system



- Easy to understand since it involves the addition of rules by human subject matter experts.
- Product manager can define what product do and do not go together and may be complementary versus supplementary

CF/CB Combiner

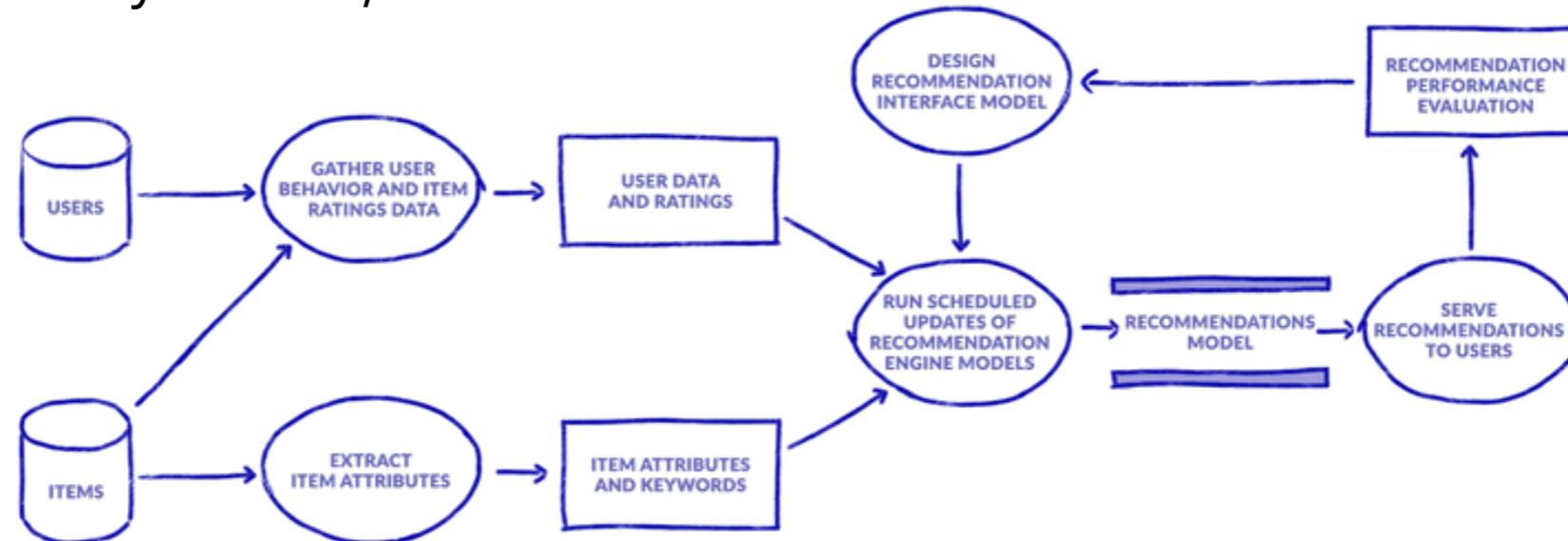
- No specific universal best practices for hybrid. Some strategies might include:
 - Weighting : Voting strategy
 - Rules base selection: Based on rules you devise
 - Combination: Recommend if both are present
 - Attribution Integration: Take mata data from both to create a third recommender
- Netflix use hybrid CB/CF recommender

3 Activities to implement Recommendation System

- Designing and evaluating a recommendation model.
- Scheduling system updates and piping data into the model and out to the user.
- Integrating the recommendation system with the company's business system.

3 Activities to implement Recommendation System

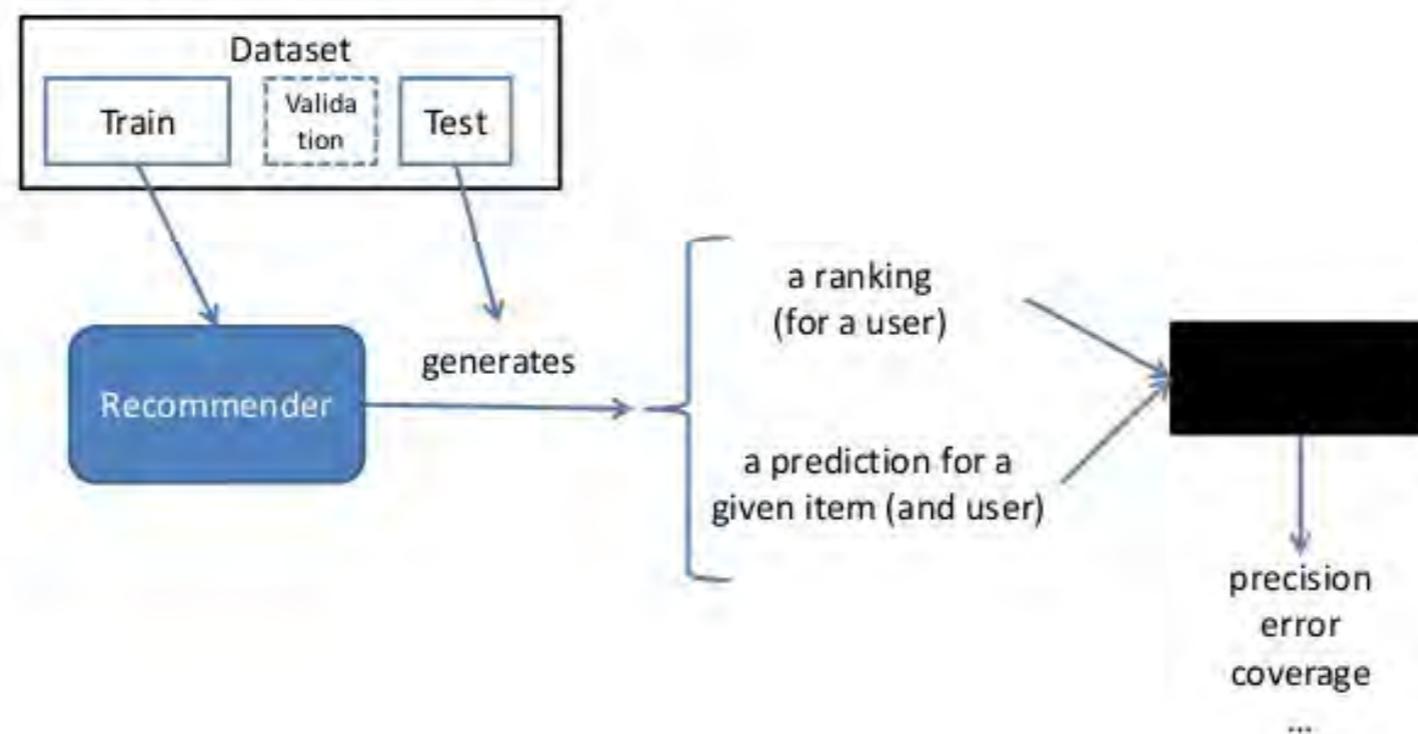
- No specific universal best practices for hybrid. Some strategies might include:
- Weighting : Voting strategy
- Rules base selection: Based on rules you devise
- Combination: Recommend if both are present
- Attribution Integration: Take meta data from both to create a third recommender
- Netflix use hybrid CB/CF recommender



Evaluating Recommendation System

Recommender Systems Evaluation

- Typically: as a black box



Evaluation Metrics

- A major obstacle while designing recommendation systems is choosing what metrics to optimize.
- There are a few metrics that we could use:
 - Statistical Accuracy Metrics
 - Decision Support Accuracy Metrics

Statistical Accuracy Metrics

- This metric used to evaluate accuracy of a filtering technique by comparing the predicted rating directly with actual user rating.
 - Mean Absolute Error (MAE)
 - Root Mean Square Error (RMSE)
 - Correlation

Statistical Accuracy Metrics

- This metric used to evaluate accuracy of a filtering technique by comparing the predicted rating directly with actual user rating.
 - Mean Absolute Error (MAE)
 - Root Mean Square Error (RMSE)
 - Correlation

Statistical Accuracy Metrics

- This metric used to evaluate accuracy of a filtering technique by comparing the predicted rating directly with actual user rating.
 - Mean Absolute Error (MAE)
 - Root Mean Square Error (RMSE)
 - Correlation

$$MAE = \frac{1}{N} \sum |predicted - actual|$$

$$RMSE = \sqrt{\frac{1}{N} \sum (predicted - actual)^2}$$

Rule strength measures

Support: The rule holds with support sup in T (the transaction data set) if $sup\%$ of transactions contain $X \cup Y$.

$$sup = \Pr(X \cup Y).$$

Confidence: The rule holds in T with confidence $conf$ if $conf\%$ of transactions that contain X also contain Y .

$$conf = \Pr(Y | X)$$

An association rule is a pattern that states when X occurs, Y occurs with certain probability.

Support and Confidence

Support count: The support count of an itemset X , denoted by $X.count$, in a data set T is the number of transactions in T that contain X . Assume T has n transactions.

Then,

$$support = \frac{(X \cup Y).count}{n}$$

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Decision Support Accuracy Metrics

- Precision and Recall often help users select items that are more similar among available set of items.

The metrics view prediction procedure as a binary operation which distinguishes good items from those items that are not good.

- Recall@k and Precision@k

Recall@k and Precision@k

Can be used for most recommendation systems.

recommender system precision:

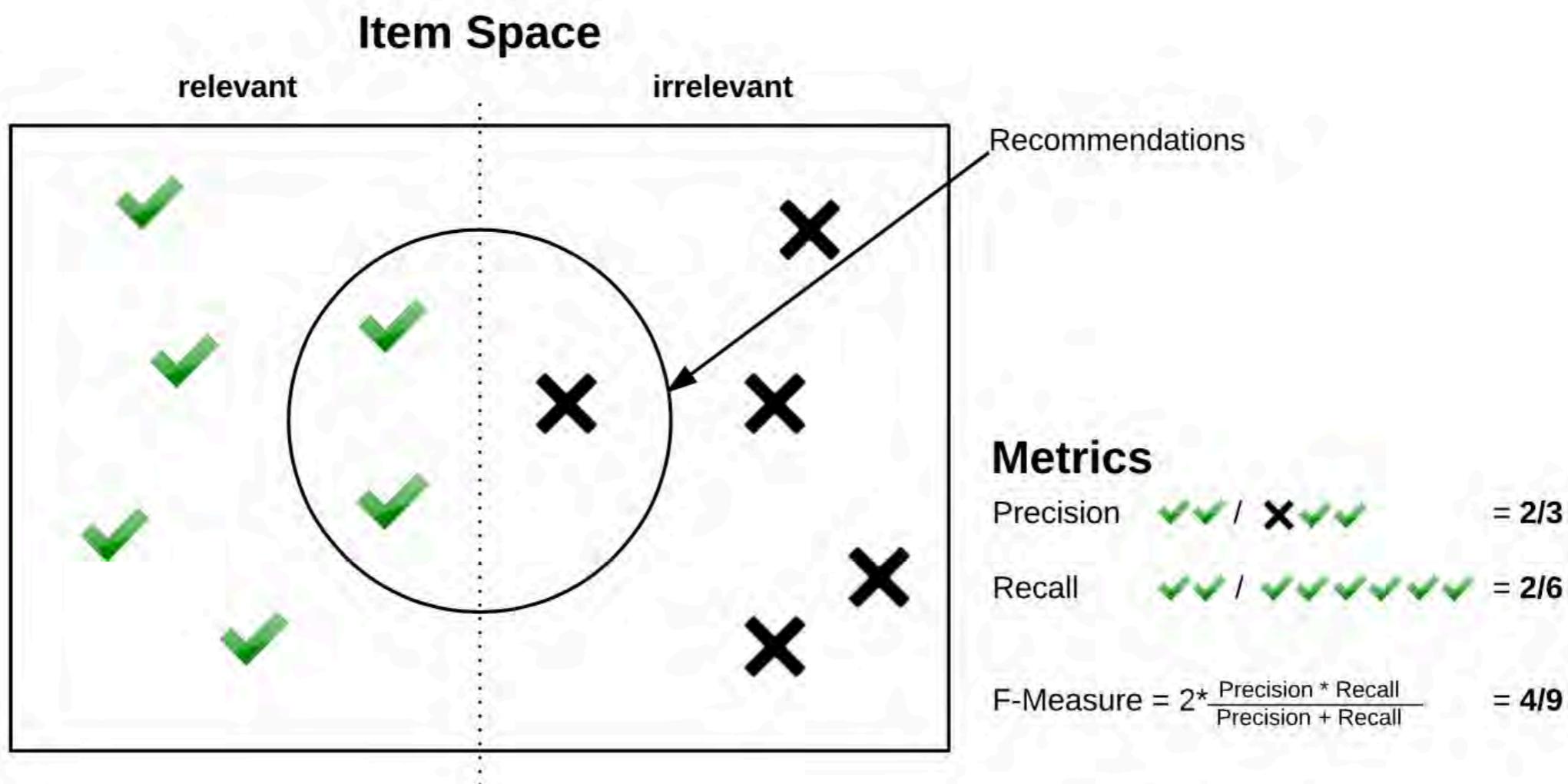
$$P = \frac{\text{\# of our recommendations that are relevant}}{\text{\# of items we recommended}}$$

recommender system recall:

$$r = \frac{\text{\# of our recommendations that are relevant}}{\text{\# of all the possible relevant items}}$$

Precision and Recall

- Can be used for most recommendation systems.



Precision and Recall at cutoff k, P@k, and r@k

- Precision and Recall at cutoff k P@k and r@k are simply the precision and recall calculated by considering only the subset of your recommendations from rank 1 through k.

$P@3 = 1/3$

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
4	Auto loan	False positive
5	HELOC	Correct Positive
6	College Fund	Correct positive
7	Personal loan	False positive

$P@6 = 3/6$

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
4	Auto loan	False positive
5	HELOC	Correct Positive
6	College Fund	Correct positive
7	Personal loan	False positive

Average Precision

If we have to recommend N items and there are m relevant items in the full space of items, Average Precision AP@N is defined as:

$$AP@N = \frac{1}{m} \sum_{k=1}^N (P(k) \text{ if } k^{\text{th}} \text{ item was relevant}) = \frac{1}{m} \sum_{k=1}^N P(k) \cdot rel(k)$$

$$AP@N = \sum_{k=1}^N (\text{precision at } k) \cdot (\text{change in recall at } k) = \sum_{k=1}^N P(k) \Delta r(k)$$

Mean Average Precision

MAP@N just average the AP across all users.

$$\text{MAP@N} = \frac{1}{|U|} \sum_{u=1}^{|U|} U |(\text{AP@N})_u|$$



Building Simple Popularity Recommendation System in R

Data Science Certification

```
library("recommenderlab")

data("MovieLense")
### use only users with more than 100 ratings
MovieLense100 <- MovieLense[rowCounts(MovieLense) >100,]
MovieLense100

index <- sample(2, nrow(MovieLense100), replace = TRUE, prob = c(0.7, 0.3))
train <- MovieLense100[index==1]
test <- MovieLense100[index==2]
recommender <- Recommender(train, method = "POPULAR")
recommender
```

```
pred <- predict(recommender, MovieLense100[101:102], n = 10)
pred
as(pred, "list")  
```
$`291`
[1] "Godfather, The (1972)"
[2] "Schindler's List (1993)"
[3] "Casablanca (1942)"
[4] "One Flew Over the Cuckoo's Nest (1975)"
[5] "Amadeus (1984)"
[6] "Titanic (1997)"
[7] "Rear Window (1954)"
[8] "Alien (1979)"
[9] "Citizen Kane (1941)"
[10] "Psycho (1960)"

$`292`
[1] "Schindler's List (1993)"
[2] "Empire Strikes Back, The (1980)"
[3] "Usual Suspects, The (1995)"
[4] "Braveheart (1995)"
[5] "One Flew Over the Cuckoo's Nest (1975)"
[6] "Amadeus (1984)"
[7] "Blade Runner (1982)"
[8] "Titanic (1997)"
[9] "Citizen Kane (1941)"
[10] "Psycho (1960)"
```

## Workshop 4.13 - Build popularity recommender system

```
retail <- readRDS("retail.rds")
```

build most popularity recommender system using recommenderlab package



# Association Rule Recommendation System

Data Science Certification

# Topics

- Basic concepts
- Apriori algorithm
- Summary

# Association rule mining

- Proposed by **Agrawal et al in 1993.**
- It is an important data mining model studied extensively by the database and data mining community.
- Assume all data are categorical.
- No good algorithm for numeric data.
- Initially used for **Market Basket Analysis** to find how items purchased by customers are related.

Bread → Milk [sup = 5%, conf = 100%]

# The model: data

$I = \{i_1, i_2, \dots, i_m\}$ : a set of *items*.

Transaction  $t$ :

$t$  a set of items, and  $t \subseteq I$ .

Transaction Database  $T$ : a set of transactions  $T = \{t_1, t_2, \dots, t_n\}$ .

# Transaction data: supermarket data

Market basket transactions:

t1: {bread, cheese, milk}

t2: {apple, eggs, salt, yogurt}

...

...

tn: {biscuit, eggs, milk}

Concepts:

An *item*: an item/article in a basket

*I*: the set of all items sold in the store

A *transaction*: items purchased in a basket; it may have TID (transaction ID)

A *transactional dataset*: A set of transactions

# Transaction data: a set of documents

**A text document data set. Each document is treated as a “bag” of keywords**

doc1: Student, Teach, School

doc2: Student, School

doc3: Teach, School, City, Game

doc4: Baseball, Basketball

doc5: Basketball, Player, Spectator

doc6: Baseball, Coach, Game, Team

doc7: Basketball, Team, City, Game

# The model: rules

A transaction  $t$  contains  $X$ , a set of items (itemset) in  $I$ , if  $X \subseteq t$ .

An association rule is an implication of the form:

$X \rightarrow Y$ , where  $X, Y \subset I$ , and  $X \cap Y = \emptyset$

An itemset is a set of items.

E.g.,  $X = \{\text{milk, bread, cereal}\}$  is an itemset.

A  $k$ -itemset is an itemset with  $k$  items.

E.g.,  $\{\text{milk, bread, cereal}\}$  is a 3-itemset

# Goal and key features

**Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf).

## Key Features

Completeness: find all rules.

No target item(s) on the right-hand-side

Mining with data on **hard disk** (not in memory)

# An example

- Transaction data
- Assume:
  - $\text{minsup} = 30\%$
  - $\text{minconf} = 80\%$
- An example frequent *itemset*:
  - {Chicken, Clothes, Milk} [sup = 3/7]
  - Association rules from the itemset:
    - Clothes  $\rightarrow$  Milk, Chicken [sup = 3/7, conf = 3/3]
    - ...
    - Clothes, Chicken  $\rightarrow$  Milk, [sup = 3/7, conf = 3/3]

|     |                                      |
|-----|--------------------------------------|
| t1: | Beef, Chicken, Milk                  |
| t2: | Beef, Cheese                         |
| t3: | Cheese, Boots                        |
| t4: | Beef, Chicken, Cheese                |
| t5: | Beef, Chicken, Clothes, Cheese, Milk |
| t6: | Chicken, Clothes, Milk               |
| t7: | Chicken, Milk, Clothes               |

# Transaction data representation

- A simplistic view of shopping baskets,
- Some important information not considered. E.g,
  - the quantity of each item purchased and
  - the price paid.

# Many mining algorithms

- **There are a large number of them!!**
- They use different strategies and data structures.
- Their resulting sets of rules are all the same.
- Given a transaction data set  $T$ , and a minimum support and a minimum confident, the set of association rules existing in  $T$  is uniquely determined.
- Any algorithm should find the same set of rules although their computational efficiencies and memory requirements may be different.
- We study only one: **the Apriori Algorithm**

# Topics

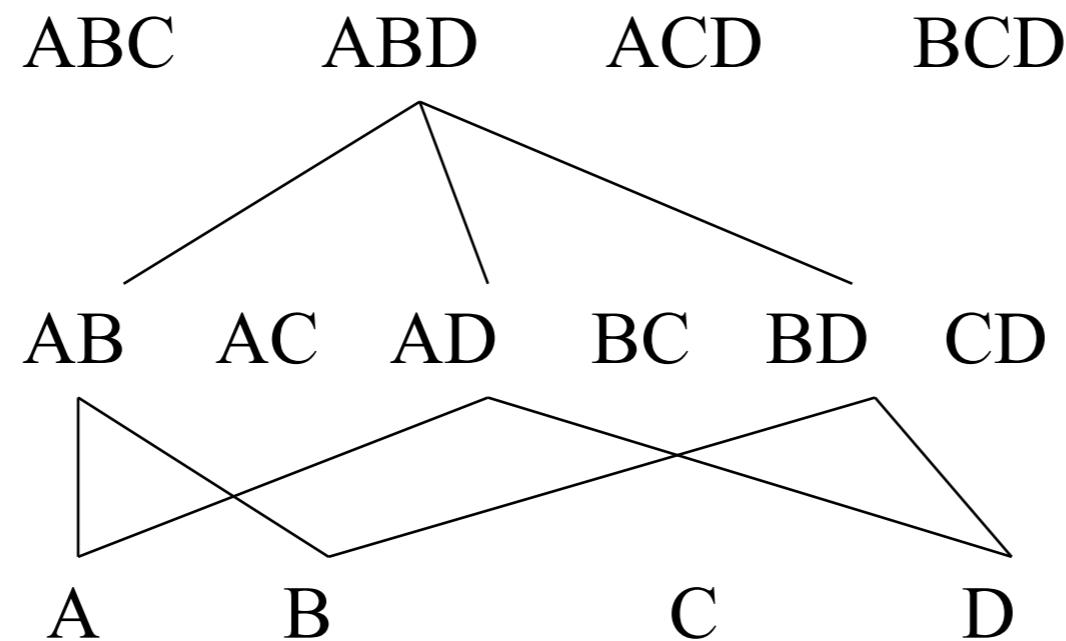
- Basic concepts
- Apriori algorithm
- Summary

# The Apriori algorithm

- **Probably the best known algorithm**
- **Two steps:**
  - Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets).
  - Use frequent itemsets to **generate rules**.
- E.g., a frequent itemset
  - {Chicken, Clothes, Milk} [sup = 3/7]
  - and one rule from the frequent itemset
    - Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

# Step 1: Mining all frequent itemsets

- A **frequent itemset** is an itemset whose support is  $\geq \text{minsup}$ .
- **Key idea: The apriori property (downward closure property)**: any subsets of a frequent itemset are also frequent itemsets



# The Algorithm

- **Iterative algo.** (also called **level-wise search**): Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on.
  - In each iteration  $k$ , only consider itemsets that contain some  $k-1$  frequent itemset.
- Find frequent itemsets of size 1:  $F_1$
- From  $k = 2$ 
  - $C_k$  = candidates of size  $k$ : those itemsets of size  $k$  that could be frequent, given  $F_{k-1}$
  - $F_k$  = those itemsets that are actually frequent,  $F_k \subseteq C_k$  (need to scan the database once).

# Example – Finding frequent itemsets

Dataset T  
minsup=0.5

| TID  | Items      |
|------|------------|
| T100 | 1, 3, 4    |
| T200 | 2, 3, 5    |
| T300 | 1, 2, 3, 5 |
| T400 | 2, 5       |

itemset:count

1. scan T → C<sub>1</sub>: {1}:2, {2}:3, {3}:3, {4}:1, {5}:3

→ F<sub>1</sub>: {1}:2, {2}:3, {3}:3, {5}:3

→ C<sub>2</sub>: {1,2}, {1,3}, {1,5}, {2,3}, {2,5}, {3,5}

2. scan T → C<sub>2</sub>: {1,2}:1, {1,3}:2, {1,5}:1, {2,3}:2, {2,5}:3, {3,5}:2

→ F<sub>2</sub>: {1,3}:2, {2,3}:2, {2,5}:3, {3,5}:2

→ C<sub>3</sub>: {2, 3, 5}

3. scan T → C<sub>3</sub>: {2, 3, 5}:2 → F<sub>3</sub>: {2, 3, 5}

# Details: ordering of items

- The items in  $I$  are sorted in **lexicographic order** (which is a total order).
- The order is used throughout the algorithm in each itemset.
- $\{w[1], w[2], \dots, w[k]\}$  represents a  $k$ -itemset  $w$  consisting of items  $w[1]$ ,  $w[2], \dots, w[k]$ , where  $w[1] < w[2] < \dots < w[k]$  according to the total order.

# Details: the algorithm

**Algorithm Apriori( $T$ )**

```
 $C_1 \leftarrow \text{init-pass}(T);$
 $F_1 \leftarrow \{f \mid f \in C_1, f.\text{count}/n \geq \text{minsup}\}; \quad // n: \text{no. of transactions in } T$
for ($k = 2; F_{k-1} \neq \emptyset; k++$) do
 $C_k \leftarrow \text{candidate-gen}(F_{k-1});$
 for each transaction $t \in T$ do
 for each candidate $c \in C_k$ do
 if c is contained in t then
 $c.\text{count}++;$
 end
 end
 end
 $F_k \leftarrow \{c \in C_k \mid c.\text{count}/n \geq \text{minsup}\}$
end
return $F \leftarrow \bigcup_k F_k;$
```

# Apriori candidate generation

- The **candidate-gen** function takes  $F_{k-1}$  and returns a **superset** (called the candidates) of the set of all **frequent  $k$ -itemsets**. It has two steps
  - **join step**: Generate all possible candidate itemsets  $C_k$  of length  $k$
  - **prune step**: Remove those candidates in  $C_k$  that cannot be frequent.

# Candidate-gen function

**Function** candidate-gen( $F_{k-1}$ )

```
 $C_k \leftarrow \emptyset;$
forall $f_1, f_2 \in F_{k-1}$
 with $f_1 = \{i_1, \dots, i_{k-2}, i_{k-1}\}$
 and $f_2 = \{i_1, \dots, i_{k-2}, i'_{k-1}\}$
 and $i_{k-1} < i'_{k-1}$ do
 $c \leftarrow \{i_1, \dots, i_{k-1}, i'_{k-1}\};$ // join f_1 and f_2
 $C_k \leftarrow C_k \cup \{c\};$
 for each ($k-1$)-subset s of c do
 if ($s \notin F_{k-1}$) then
 delete c from $C_k;$ // prune
 end
 end
 return $C_k;$
```

# An example

- $F_3 = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}\}$
- After join
- $C_4 = \{\{1, 2, 3, 4\}, \{1, 3, 4, 5\}\}$
- After pruning:
- $C_4 = \{\{1, 2, 3, 4\}\}$
- because  $\{1, 4, 5\}$  is not in  $F_3$  ( $\{1, 3, 4, 5\}$  is removed)

## Step 2: Generating rules from frequent itemsets

- Frequent itemsets  $\neq$  association rules
- One more step is needed to generate association rules
- For each frequent itemset  $X$ ,
- For each proper nonempty subset  $A$  of  $X$ ,
  - Let  $B = X - A$
  - $A \rightarrow B$  is an association rule if
    - $\text{Confidence}(A \rightarrow B) \geq \text{minconf}$ ,
    - $\text{support}(A \rightarrow B) = \text{support}(A \cup B) = \text{support}(X)$
    - $\text{confidence}(A \rightarrow B) = \text{support}(A \cup B) / \text{support}(A)$

# Generating rules: an example

- Suppose  $\{2,3,4\}$  is frequent, with sup=50%
- Proper nonempty subsets:  $\{2,3\}$ ,  $\{2,4\}$ ,  $\{3,4\}$ ,  $\{2\}$ ,  $\{3\}$ ,  $\{4\}$ , with sup=50%, 50%, 75%, 75%, 75% respectively
- These generate these association rules:
  - $2,3 \rightarrow 4$ , confidence=100%
  - $2,4 \rightarrow 3$ , confidence=100%
  - $3,4 \rightarrow 2$ , confidence=67%
  - $2 \rightarrow 3,4$ , confidence=67%
  - $3 \rightarrow 2,4$ , confidence=67%
  - $4 \rightarrow 2,3$ , confidence=67%
  - All rules have support = 50%

# Generating rules: summary

- To recap, in order to obtain  $A \rightarrow B$ , we need to have  $\text{support}(A \cup B)$  and  $\text{support}(A)$
- All the required information for confidence computation has already been recorded in itemset generation. No need to see the data  $T$  any more.
- This step is not as time-consuming as frequent itemsets generation.

# On Apriori Algorithm

- Seems to be very expensive
- Level-wise search
- $K =$  the size of the largest itemset
- It makes at most  $K$  passes over data
- In practice,  $K$  is bounded (10).
- The algorithm is very fast. Under some conditions, all rules can be found in **linear time**.
- Scale up to large data sets

# More on association rule mining

- Clearly the space of all association rules is **exponential**,  $O(2^m)$ , where  $m$  is the number of items in  $I$ .
- The mining exploits **sparseness of data**, and **high minimum support** and **high minimum confidence** values.
- Still, it always produces a **huge number of rules**, thousands, tens of thousands, millions, ...

# Building Association Rule Based Recommendation System in R



Data Science Certification

# Support, Confidence and Lift

There are several measures used to understand various aspects of associated products.

Let's understand the measures with the help of an example.

- In a store, there are 1000 transactions overall.
- Item A appears in 80 transactions and
- Item B occurs in 100 transactions.
- Items A and B appear in 20 transactions together.



**Support** is the ratio of number of times two or more items occur together to the total number of transactions.

- **Support of A** =  $\Pr(A) = 80/1000 = 8\%$  and
- **Support of B** =  $\Pr(B) = 100/1000 = 10\%$ .

**Confidence** is a conditional probability that a randomly selected transaction will include Item A given Item B.

- **Confidence of A** =  $\Pr(A/B) = 20/100 = 20\%$ .

**Lift** can be expressed as the ratio of the probability of Items A and B occurring together to the multiple of the two individual probabilities for Item A and Item B.

- **Lift** =  $\Pr(A,B) / \Pr(A).\Pr(B) = (20/1000)/((80/1000)\times(100/1000)) = 2.5$ .

# How would you use Support, Confidence and Lift?

**Support** of a product or product bundle indicates the popularity of the product or product bundle in the transaction set. Higher the support, more popular is the product or product bundle. This measure can help in identifying driver of traffic to the store. Hence, if Barbie dolls have a higher support then they can be attractively priced to attract traffic to a store.

**Confidence** can be used for product placement strategy and increasing profitability. Place high-margin items with associated high selling (driver) items. If Market Basket Analysis indicates that customers who bought high selling Barbie dolls also bought high-margin candies, then candies should be placed near Barbie dolls.

**Lift** indicates the strength of an association rule over the random co-occurrence of Item A and Item B, given their individual support. Lift provides information about the change in probability of Item A in presence of Item B. Lift values greater than 1.0 indicate that transactions containing Item B tend to contain Item A more often than transactions that do not contain Item B.

# Apriori Algorithm

?apriori

## Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

## Arguments

**data**

object of class [transactions](#) or any data structure which can be coerced into [transactions](#) (e.g., a binary matrix or data.frame).

**parameter**

object of class [APparameter](#) or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.

**appearance**

object of class [APappearance](#) or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted.

**control**

object of class [APcontrol](#) or named list. Controls the performance of the mining algorithm (item sorting, etc.)

# Apriori Algorithm

So lets get started by loading up our libraries and data set.

```
Load the libraries
```

```
library(arules)
```

```
library(arulesViz)
```

```
library(datasets)
```

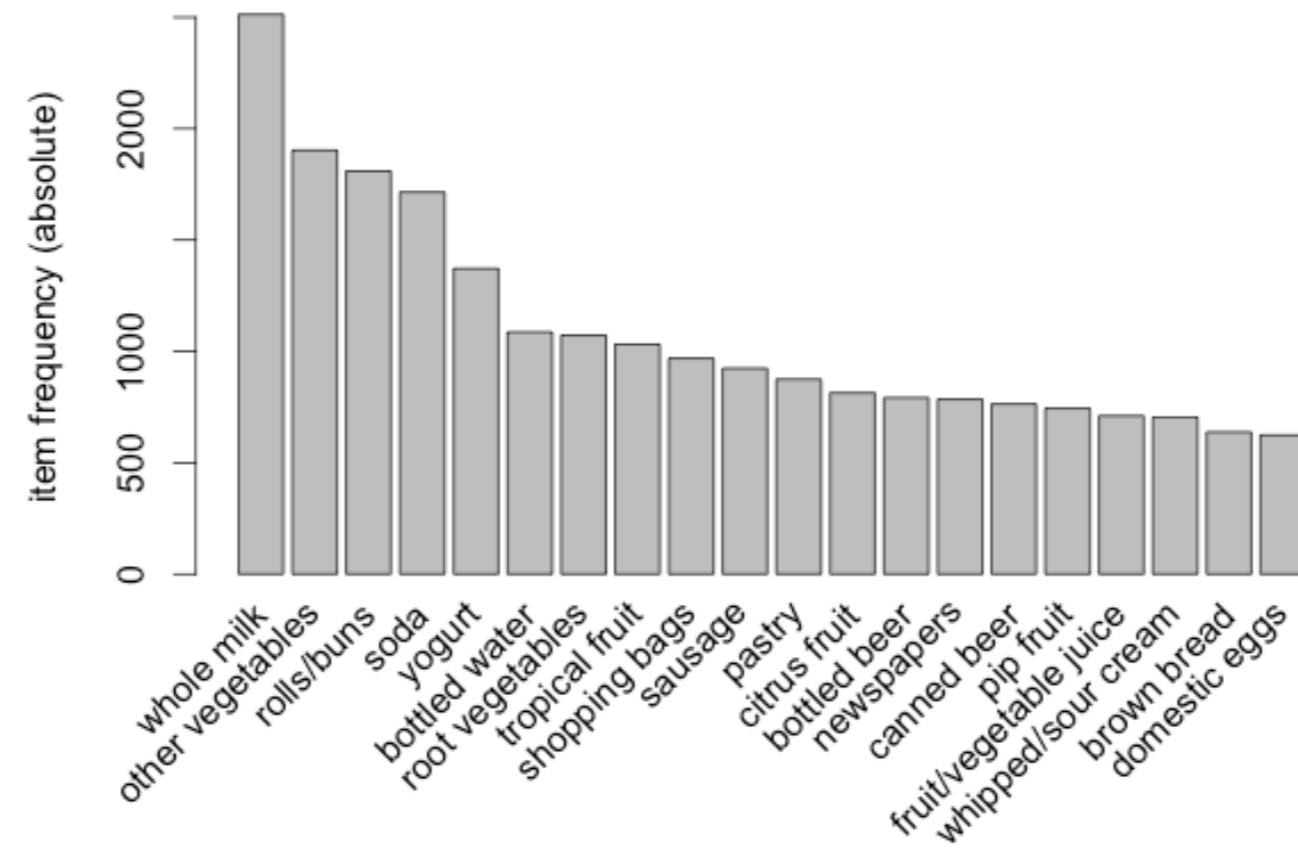
```
Load the data set
```

```
data(Groceries)
```

# Explore Data

# Create an item frequency plot for the top 20 items

```
itemFrequencyPlot(Groceries, topN=20,type="absolute")
```



```
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))
```

```
Show the top 5 rules, but only 2 digits
```

```
options(digits=2)
```

```
inspect(rules[1:5])
```

| lhs                                             | rhs             | support | confidence | lift |
|-------------------------------------------------|-----------------|---------|------------|------|
| 1 {liquor,<br>red/blush wine} => {bottled beer} | 0.0019          | 0.90    | 11.2       |      |
| 2 {curd,<br>cereals}                            | => {whole milk} | 0.0010  | 0.91       | 3.6  |
| 3 {yogurt,<br>cereals}                          | => {whole milk} | 0.0017  | 0.81       | 3.2  |
| 4 {butter,<br>jam}                              | => {whole milk} | 0.0010  | 0.83       | 3.3  |
| 5 { soups,<br>bottled beer}                     | => {whole milk} | 0.0011  | 0.92       | 3.6  |
| >                                               |                 |         |            |      |

```
summary(rules)
```

set of 410 rules

rule length distribution (lhs + rhs):sizes

| 3  | 4   | 5   | 6  |
|----|-----|-----|----|
| 29 | 229 | 140 | 12 |

| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
|------|---------|--------|------|---------|------|
| 3.0  | 4.0     | 4.0    | 4.3  | 5.0     | 6.0  |

summary of quality measures:

| support         | confidence   | lift         |
|-----------------|--------------|--------------|
| Min. :0.00102   | Min. :0.80   | Min. : 3.1   |
| 1st Qu.:0.00102 | 1st Qu.:0.83 | 1st Qu.: 3.3 |
| Median :0.00122 | Median :0.85 | Median : 3.6 |
| Mean   :0.00125 | Mean   :0.87 | Mean   : 4.0 |
| 3rd Qu.:0.00132 | 3rd Qu.:0.91 | 3rd Qu.: 4.3 |
| Max.   :0.00315 | Max.   :1.00 | Max.   :11.2 |

mining info:

| data      | ntransactions | support | confidence |
|-----------|---------------|---------|------------|
| Groceries | 9835          | 0.001   | 0.8        |

## # Sort Rules

```
rules<-sort(rules, by="confidence", decreasing=TRUE)
```

| lhs                                                  | rhs             | support | confidence | lift |
|------------------------------------------------------|-----------------|---------|------------|------|
| 1 {rice,<br>sugar}                                   | => {whole milk} | 0.0012  | 1          | 3.9  |
| 2 {canned fish,<br>hygiene articles}                 | => {whole milk} | 0.0011  | 1          | 3.9  |
| 3 {root vegetables,<br>butter,<br>rice}              | => {whole milk} | 0.0010  | 1          | 3.9  |
| 4 {root vegetables,<br>whipped/sour cream,<br>flour} | => {whole milk} | 0.0017  | 1          | 3.9  |
| 5 {butter,<br>soft cheese,<br>domestic eggs}         | => {whole milk} | 0.0010  | 1          | 3.9  |

# Change to have limit association in one

```
change to have maximum of 3
rules <- apriori(Groceries, parameter = list(supp = 0.001, conf =
0.8,maxlen=3))
inspect(rules[1:5])
```

|   | lhs                            | rhs             | support | confidence | lift |
|---|--------------------------------|-----------------|---------|------------|------|
| 1 | {liquor,<br>red/blush wine} => | {bottled beer}  | 0.0019  | 0.90       | 11.2 |
| 2 | {curd,<br>cereals}             | => {whole milk} | 0.0010  | 0.91       | 3.6  |
| 3 | {yogurt,<br>cereals}           | => {whole milk} | 0.0017  | 0.81       | 3.2  |
| 4 | {butter,<br>jam}               | => {whole milk} | 0.0010  | 0.83       | 3.3  |
| 5 | { soups,<br>bottled beer}      | => {whole milk} | 0.0011  | 0.92       | 3.6  |

## # Rules pruned

```
subset.matrix <- is.subset(rules, rules)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules[!redundant]
rules<-rules.pruned
summary(rules)
```

```
set of 330 rules

rule length distribution (lhs + rhs):sizes
 3 4 5 6
 29 216 84 1

 Min. 1st Qu. Median Mean 3rd Qu. Max.
 3.0 4.0 4.0 4.2 5.0 6.0

summary of quality measures:
 support confidence lift
 Min. :0.00102 Min. :0.80 Min. : 3.1
 1st Qu.:0.00102 1st Qu.:0.82 1st Qu.: 3.3
 Median :0.00122 Median :0.85 Median : 3.6
 Mean :0.00127 Mean :0.86 Mean : 3.8
 3rd Qu.:0.00132 3rd Qu.:0.91 3rd Qu.: 4.3
 Max. :0.00315 Max. :1.00 Max. :11.2

mining info:
 data ntransactions support confidence
 Groceries 9835 0.001 0.8
```

# Targeting Items

What are customers likely to buy before buying whole milk?

What are customers likely to buy if they purchase whole milk?

This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

# Find whole milk's antecedents

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf =
0.08), appearance = list(default="lhs",rhs="whole milk"), control =
list(verbose=F))
```

```
rules<-sort(rules, decreasing=TRUE,by="confidence")
```

```
inspect(rules[1:5])
```

|   | lhs                                                | rhs             | support | confidence | lift |
|---|----------------------------------------------------|-----------------|---------|------------|------|
| 1 | {rice,<br>sugar}                                   | => {whole milk} | 0.0012  | 1          | 3.9  |
| 2 | {canned fish,<br>hygiene articles}                 | => {whole milk} | 0.0011  | 1          | 3.9  |
| 3 | {root vegetables,<br>butter,<br>rice}              | => {whole milk} | 0.0010  | 1          | 3.9  |
| 4 | {root vegetables,<br>whipped/sour cream,<br>flour} | => {whole milk} | 0.0017  | 1          | 3.9  |
| 5 | {butter,<br>soft cheese,<br>domestic eggs}         | => {whole milk} | 0.0010  | 1          | 3.9  |
| > |                                                    |                 |         |            |      |

# Likely to buy after buy whole milk

```
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2),
appearance = list(default="rhs",lhs="whole milk"), control = list(verbose=F))
rules<-sort(rules, decreasing=TRUE,by="confidence")
inspect(rules[1:5])
```

| lhs                                  | rhs   | support | confidence | lift |
|--------------------------------------|-------|---------|------------|------|
| 1 {whole milk} => {other vegetables} | 0.075 | 0.29    | 1.5        |      |
| 2 {whole milk} => {rolls/buns}       | 0.057 | 0.22    | 1.2        |      |
| 3 {whole milk} => {yogurt}           | 0.056 | 0.22    | 1.6        |      |
| 4 {whole milk} => {root vegetables}  | 0.049 | 0.19    | 1.8        |      |
| 5 {whole milk} => {tropical fruit}   | 0.042 | 0.17    | 1.6        |      |
| >                                    |       |         |            |      |

# Workshop 4.14 - Build Recommendation System

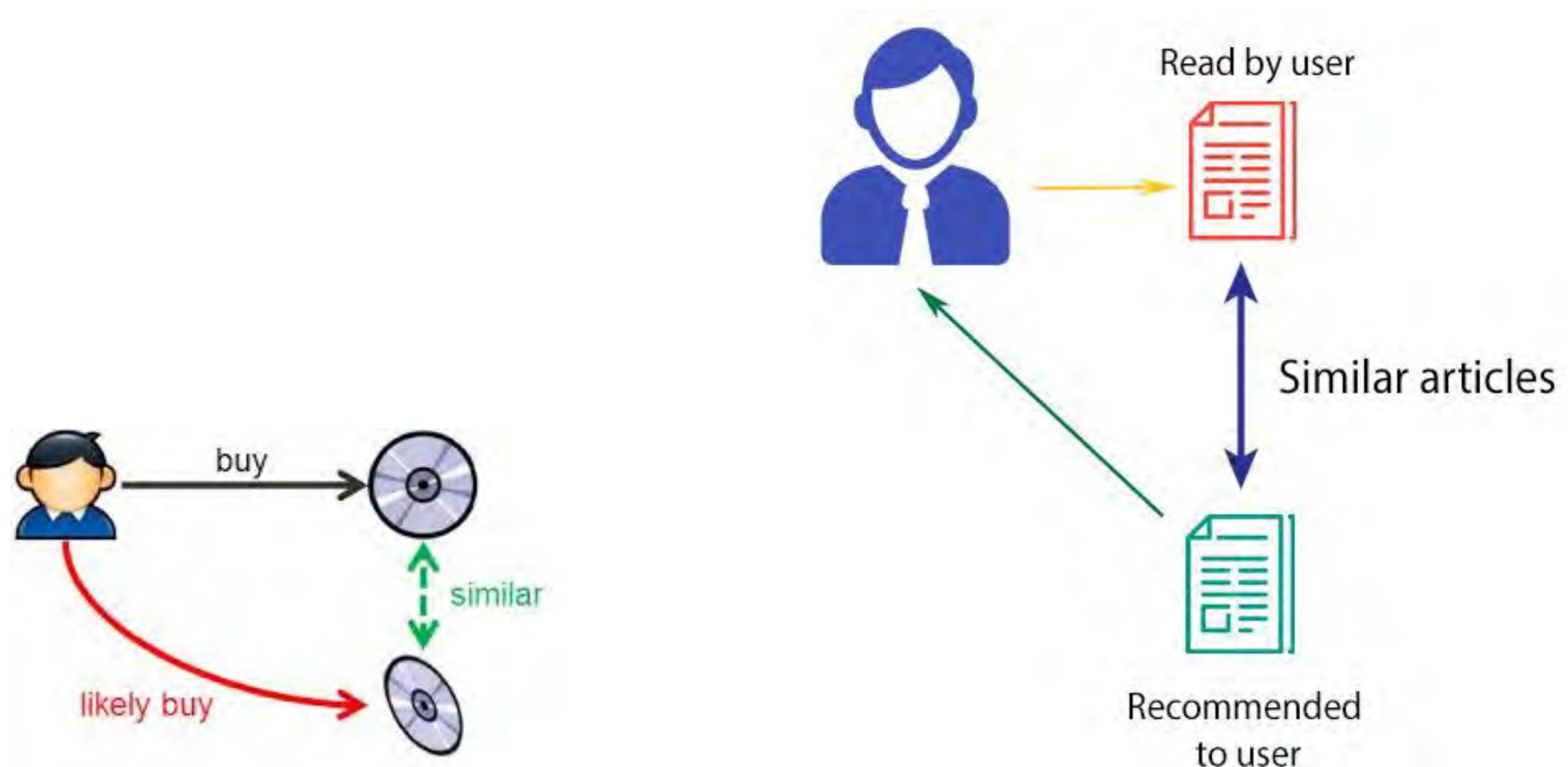
1. From data in workshop 4.13
2. Generate Association Rules



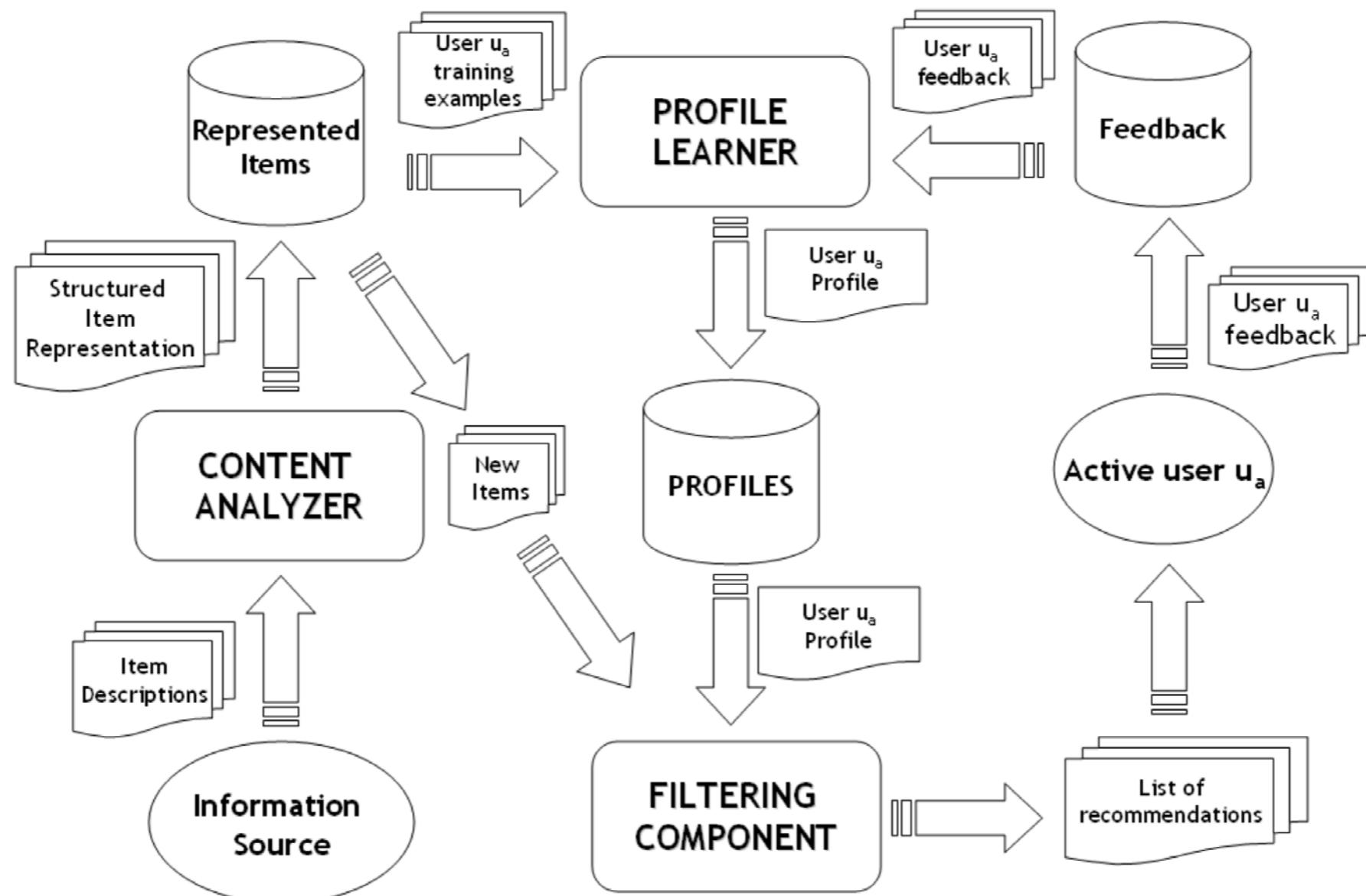
# Content-based Recommendation System

Data Science Certification

# How do content-based recommendation system work?



# High Level Process



# Creating a Item Profile

- For each item, create an item profile
- Profile is a set of features
  - movies: author, title, actor, director,...
  - text: set of “important” words in document
  - How to pick important words?
  - Usual heuristic is TF.IDF (Term Frequency times Inverse Doc Frequency)

# TF-IDF

$f_{ij}$  = frequency of term  $t_i$  in document  $d_j$

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

$n_i$  = number of docs that mention term  $i$

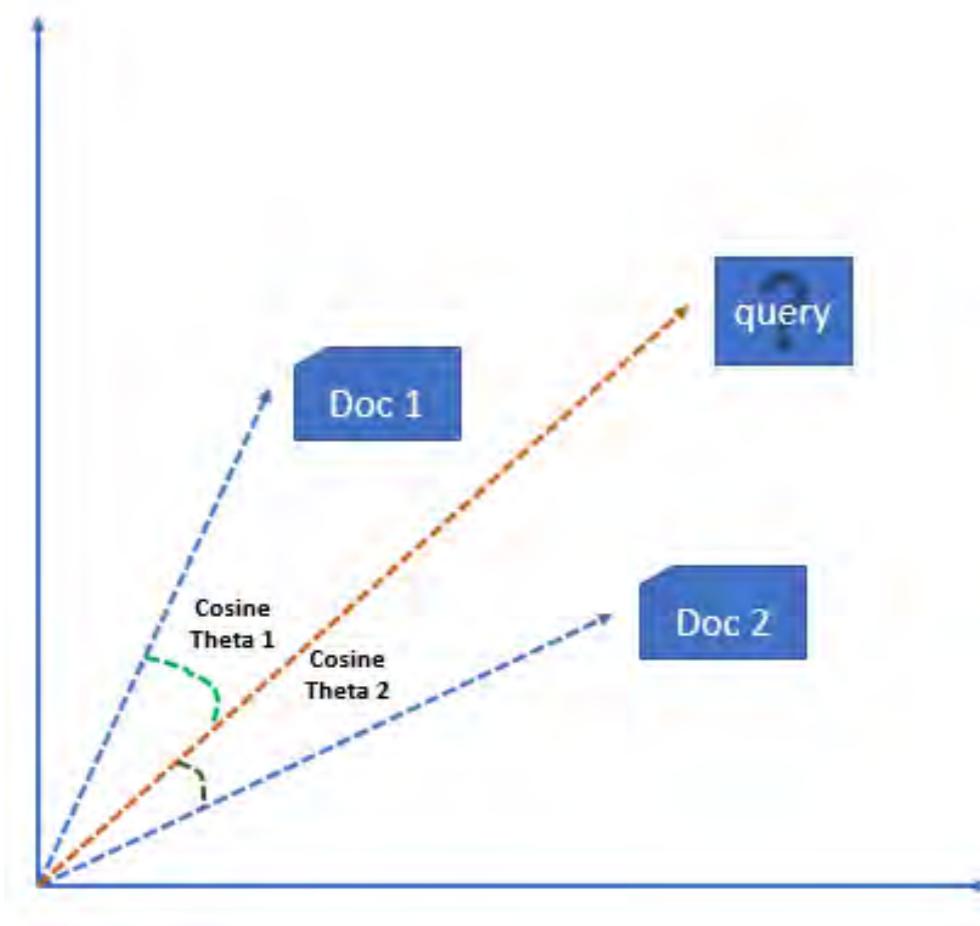
$N$  = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

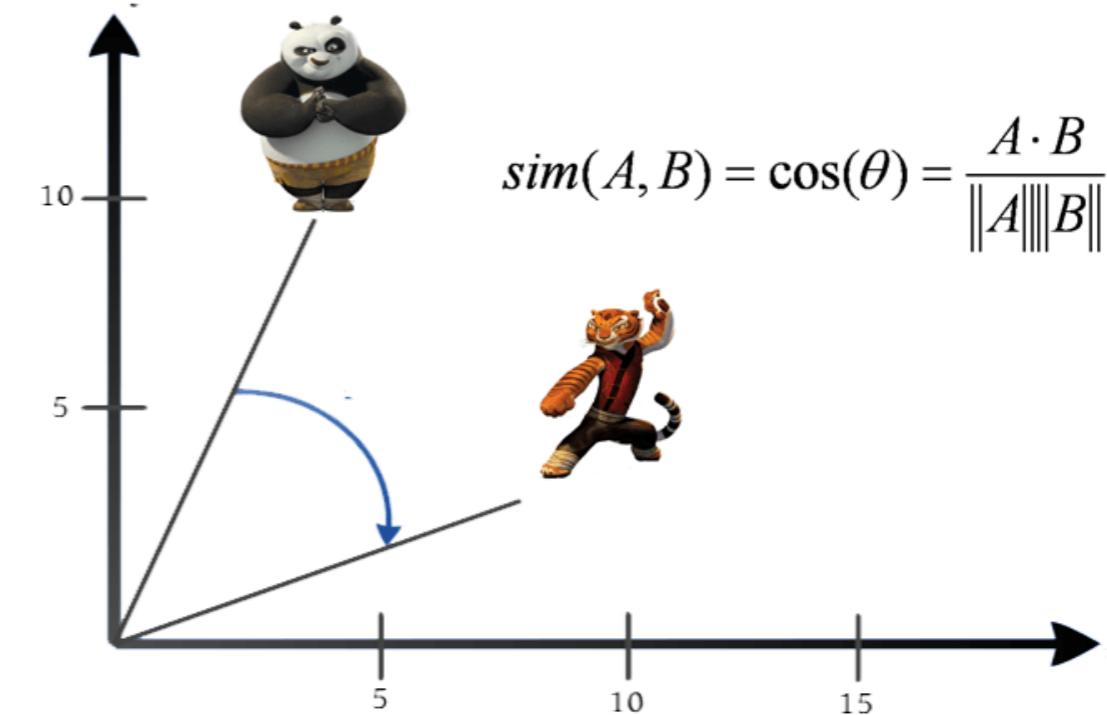
TF.IDF score  $w_{ij} = TF_{ij} \cdot IDF_i$

Doc profile = set of words with highest TF.IDF scores,  
together with their scores

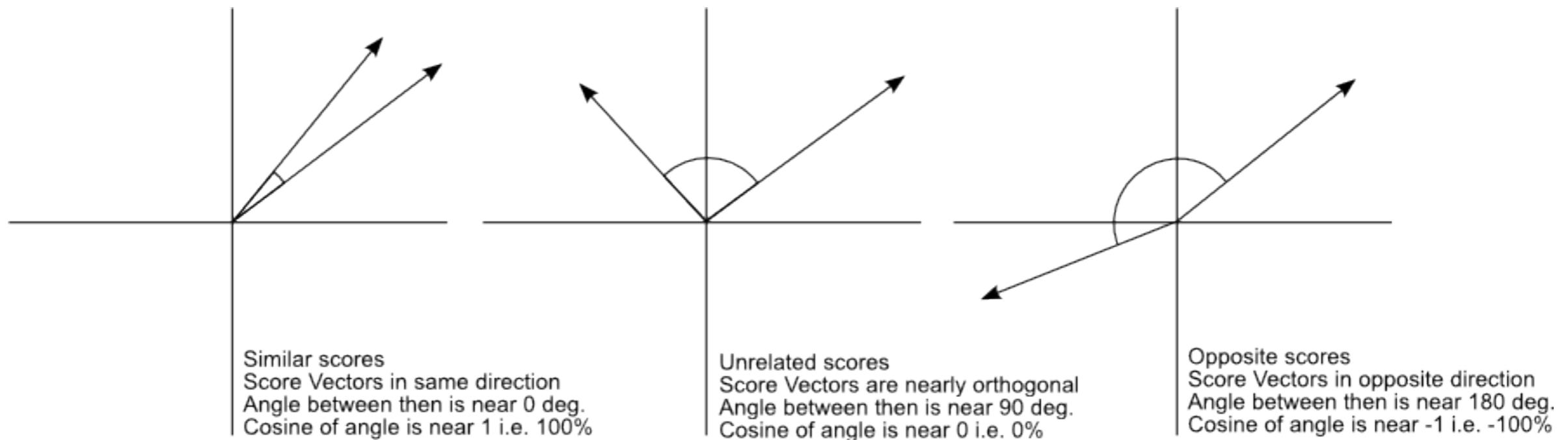
# Finding Similarity using Vector Model



Cosine Similarity



# Finding Similarity using Vector Model





# Content-based Recommendation System



# Workshop 4.15 Content-based recommender system

Using data in “Sample-data.csv”

Create Content-based recommendation system



# Collaborative Filtering

Data Science Certification

# Collaborative Filtering (CF)

- The most prominent approach to generate recommendations
  - used by large, commercial e-commerce sites
  - well-understood, various algorithms and variations exist
  - applicable in many domains (book, movies, DVDs, ..)
- Approach
  - use the "wisdom of the crowd" to recommend items
- Basic assumption and idea
  - Users give ratings to catalog items (implicitly or explicitly)
  - Customers who had similar tastes in the past, will have similar tastes in the future



# Pure CF Approaches

- Input
  - Only a matrix of given user–item ratings
- Output types
  - A (numerical) prediction indicating to what degree the current user will like or dislike a certain item
  - A top-N list of recommended items

# User-based nearest-neighbor collaborative filtering (1)

- The basic technique
  - Given an "active user" (Alice) and an item  $i$  not yet seen by Alice
    - find a set of users (peers/nearest neighbors) who liked the same items as Alice in the past **and** who have rated item  $i$
    - use, e.g. the average of their ratings to predict, if Alice will like item  $i$
    - do this for all items Alice has not seen and recommend the best-rated
  - Basic assumption and idea
    - If users had similar tastes in the past they will have similar tastes in the future
    - User preferences remain stable and consistent over time

# User-based nearest-neighbor collaborative filtering (2)

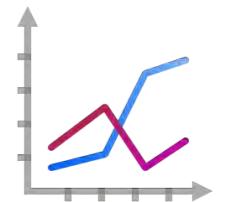
## Example

A database of ratings of the current user, Alice, and some other users is given:

|       | Item1 | Item2 | Item3 | Item4 | Item5 |
|-------|-------|-------|-------|-------|-------|
| Alice | 5     | 3     | 4     | 4     | ?     |
| User1 | 3     | 1     | 2     | 3     | 3     |
| User2 | 4     | 3     | 4     | 3     | 5     |
| User3 | 3     | 3     | 1     | 5     | 4     |
| User4 | 1     | 5     | 5     | 2     | 1     |

# User-based nearest-neighbor collaborative filtering (3)

- Some first questions
  - How do we measure similarity?
  - How many neighbors should we consider?
  - How do we generate a prediction from the neighbors' ratings?



|       | Item1 | Item2 | Item3 | Item4 | Item5 |
|-------|-------|-------|-------|-------|-------|
| Alice | 5     | 3     | 4     | 4     | ?     |
| User1 | 3     | 1     | 2     | 3     | 3     |
| User2 | 4     | 3     | 4     | 3     | 5     |
| User3 | 3     | 3     | 1     | 5     | 4     |
| User4 | 1     | 5     | 5     | 2     | 1     |

# Measuring user similarity (1)

A popular similarity measure in user-based CF: Pearson correlation

$a, b$  : users

$r_{a,p}$  : rating of user  $a$  for item  $p$

$P$  : set of items, rated both by  $a$  and  $b$

Possible similarity values between  $-1$  and  $1$

$$sim(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}}$$

# Measuring user similarity (2)

A popular similarity measure in user-based CF: Pearson correlation

$a, b$  : users

$r_{a,p}$  : rating of user  $a$  for item  $p$

$P$  : set of items, rated both by  $a$  and  $b$

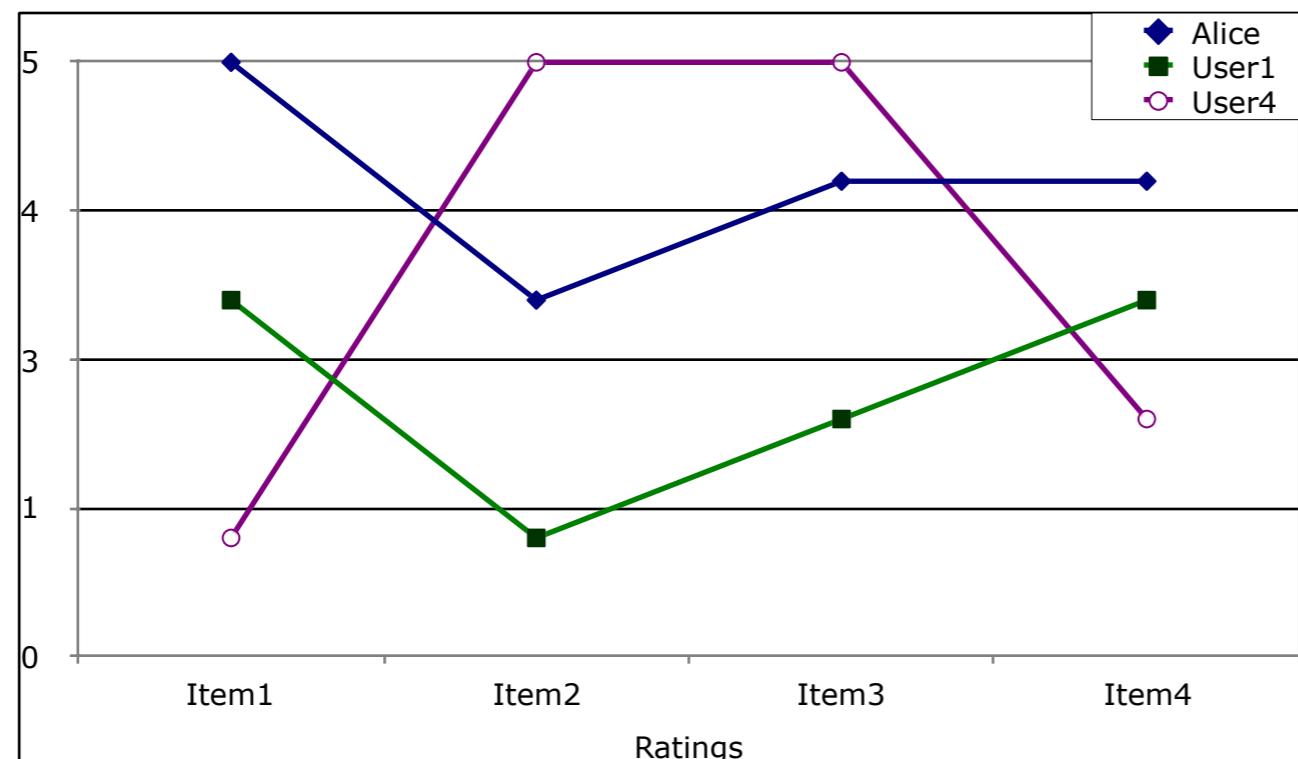
|       | Item1 | Item2 | Item3 | Item4 | Item5 |  |
|-------|-------|-------|-------|-------|-------|--|
| Alice | 5     | 3     | 4     | 4     | ?     |  |
| User1 | 3     | 1     | 2     | 3     | 3     |  |
| User2 | 4     | 3     | 4     | 3     | 5     |  |
| User3 | 3     | 3     | 1     | 5     | 4     |  |
| User4 | 1     | 5     | 5     | 2     | 1     |  |

sim = 0,85  
sim = 0,00  
sim = 0,70  
sim = -0,79



# Pearson correlation

Takes differences in rating behavior into account



Works well in usual domains, compared with alternative measures such as cosine similarity

# Making predictions

- A common prediction function:

$$pred(a, p) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b) * (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} sim(a, b)}$$



- Calculate, whether the neighbors' ratings for the unseen item  $i$  are higher or lower than their average
- Combine the rating differences – use the similarity with  $a$  as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

# Improving the metrics / prediction function

- Not all neighbor ratings might be equally "valuable"
  - Agreement on commonly liked items is not so informative as agreement on controversial items
  - Possible solution: Give more weight to items that have a higher variance
- Value of number of co-rated items
  - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
- Case amplification
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.

# Memory-based and model-based approaches

- User-based CF is said to be "memory-based"
  - the rating matrix is directly used to find neighbors / make predictions
  - does not scale for most real-world scenarios
  - large e-commerce sites have tens of millions of customers and millions of items
- Model-based approaches
  - based on an offline pre-processing or "model-learning" phase
  - at run-time, only the learned model is used to make predictions
  - models are updated / re-trained periodically
  - large variety of techniques used

# Item-based collaborative filtering

- Basic idea:
  - Use the similarity between items (and not users) to make predictions
- Example:
  - Look for items that are similar to Item5
  - Take Alice's ratings for these items to predict the rating for Item5

|       | Item1 | Item2 | Item3 | Item4 | Item5 |
|-------|-------|-------|-------|-------|-------|
| Alice | 5     | 3     | 4     | 4     | ?     |
| User1 | 3     | 1     | 2     | 3     | 3     |
| User2 | 4     | 3     | 4     | 3     | 5     |
| User3 | 3     | 3     | 1     | 5     | 4     |
| User4 | 1     | 5     | 5     | 2     | 1     |

# The cosine similarity measure

- Produces better results in item-to-item filtering
- Ratings are seen as vector in n-dimensional space
- Similarity is calculated based on the angle between the vectors

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| * |\vec{b}|}$$



- Adjusted cosine similarity
  - take average user ratings into account, transform the original ratings
  - $U$ : set of users who have rated both items  $a$  and  $b$

$$sim(\vec{a}, \vec{b}) = \frac{\sum_{u \in U} (r_{u,a} - \bar{r}_u)(r_{u,b} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,a} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,b} - \bar{r}_u)^2}}$$



# Making predictions

- A common prediction function:

$$pred(u, p) = \frac{\sum_{i \in ratedItem(u)} sim(i, p) * r_{u,i}}{\sum_{i \in ratedItem(u)} sim(i, p)}$$



- Neighborhood size is typically also limited to a specific size
- Not all neighbors are taken into account for the prediction
- An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002)

# Pre-processing for item-based filtering

- Item-based filtering does not solve the scalability problem itself
- Pre-processing approach by Amazon.com (in 2003)
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
  - Item similarities are supposed to be more stable than user similarities
- Memory requirements
  - Up to  $N^2$  pair-wise similarities to be memorized ( $N$  = number of items) in theory
  - In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible

# More on ratings – Explicit ratings

- Probably the most precise ratings
- Most commonly used (1 to 5, 1 to 7 Likert response scales)
- Research topics
  - Optimal granularity of scale; indication that 10-point scale is better accepted in movie dom.
  - An even more fine-grained scale was chosen in the joke recommender discussed by Goldberg et al. (2001), where a continuous scale (from  $-10$  to  $+10$ ) and a graphical input bar were used
    - No precision loss from the discretization
    - User preferences can be captured at a finer granularity
    - Users actually "like" the graphical interaction method

# More on ratings – Implicit ratings

- Typically collected by the web shop or application in which the recommender system is embedded
- When a customer buys an item, for instance, many recommender systems interpret this behavior as a positive rating
- Clicks, page views, time spent on some page, demo downloads ...
- Implicit ratings can be collected constantly and do not require additional efforts from the side of the user
- Main problem
  - One cannot be sure whether the user behavior is correctly interpreted
  - For example, a user might not like all the books he or she has bought; the user also might have bought a book for someone else

# Data sparsity problems

- Cold start problem
  - How to recommend new items? What to recommend to new users?
- Straightforward approaches
  - Ask/force users to rate a set of items
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase
  - Default voting: assign default values to items that only one of the two users to be compared has rated (Breese et al. 1998)
- Alternatives
  - Use better algorithms (beyond nearest-neighbor approaches)

# Example algorithms for sparse datasets

- Recursive CF (Zhang and Pu 2007)
  - Assume there is a very close neighbor  $n$  of  $u$  who however has not rated the target item  $i$  yet.
  - Idea:
    - Apply CF-method recursively and predict a rating for item  $i$  for the neighbor
    - Use this predicted rating instead of the rating of a more distant direct neighbor

|       | Item1 | Item2 | Item3 | Item4 | Item5 |
|-------|-------|-------|-------|-------|-------|
| Alice | 5     | 3     | 4     | 4     | ?     |
| User1 | 3     | 1     | 2     | 3     | ?     |
| User2 | 4     | 3     | 4     | 3     | 5     |
| User3 | 3     | 3     | 1     | 5     | 4     |
| User4 | 1     | 5     | 5     | 2     | 1     |

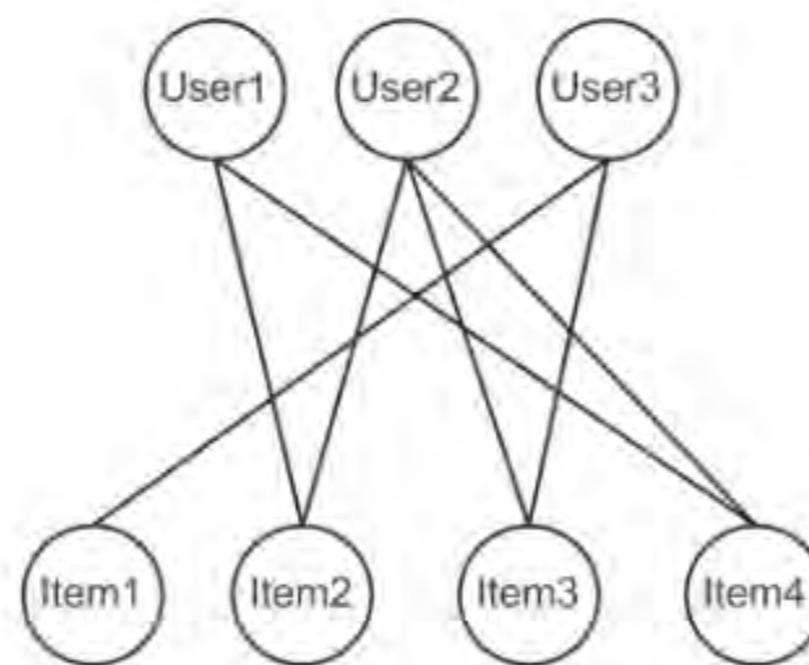
sim = 0.85

Predict rating for User1



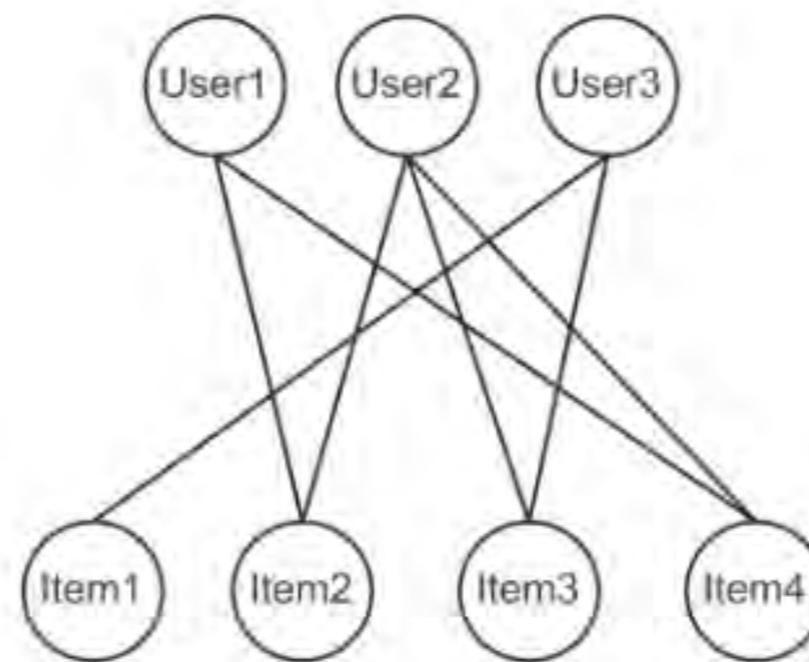
# Graph-based methods (1)

- "Spreading activation" (Huang et al. 2004)
  - Exploit the supposed "transitivity" of customer tastes and thereby augment the matrix with additional information
  - Assume that we are looking for a recommendation for *User1*
  - When using a standard CF approach, *User2* will be considered a peer for *User1* because they both bought *Item2* and *Item4*
  - Thus *Item3* will be recommended to *User1* because the nearest neighbor, *User2*, also bought or liked it



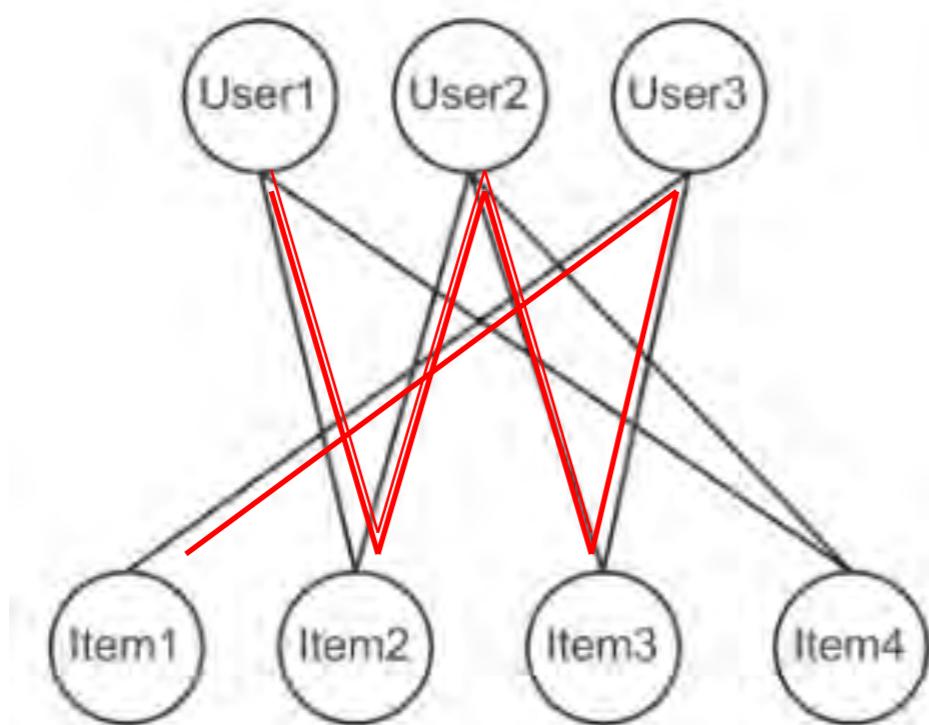
# Graph-based methods (2)

- "Spreading activation" (Huang et al. 2004)
  - In a standard user-based or item-based CF approach, paths of length 3 will be considered – that is, *Item3* is relevant for *User1* because there exists a three-step path (*User1–Item2–User2–Item3*) between them
  - Because the number of such paths of length 3 is small in sparse rating databases, the idea is to also consider longer paths (indirect associations) to compute recommendations
  - Using path length 5, for instance



# Graph-based methods (3)

- "Spreading activation" (Huang et al. 2004)
  - Idea: Use paths of lengths > 3 to recommend items
  - Length 3: Recommend Item3 to User1
  - Length 5: Item1 also recommendable



# More model-based approaches

- Plethora of different techniques proposed in the last years, e.g.,
  - Matrix factorization techniques, statistics
    - singular value decomposition, principal component analysis
  - Association rule mining
    - compare: shopping basket analysis
  - Probabilistic models
    - clustering models, Bayesian networks, probabilistic Latent Semantic Analysis
  - Various other machine learning approaches
- Costs of pre-processing
  - Usually not discussed
  - Incremental updates possible?

2000: *Application of Dimensionality Reduction in Recommender System*, B. Sarwar et al., WebKDD Workshop

- Basic idea: Trade more complex offline model building for faster online prediction generation
- Singular Value Decomposition for dimensionality reduction of rating matrices
  - Captures important factors/aspects and their weights in the data
  - factors can be genre, actors but also non-understandable ones
  - Assumption that  $k$  dimensions capture the signals and filter out noise ( $K = 20$  to 100)
- Constant time to make recommendations
- Approach also popular in IR (Latent Semantic Indexing), data compression,...

# Matrix factorization

- Informally, the SVD theorem (Golub and Kahan 1965) states that a given matrix  $M$  can be decomposed into a product of three matrices as follows

$$M = U \times \Sigma \times V^T$$

- where  $U$  and  $V$  are called *left* and *right singular vectors* and the values of the diagonal of  $\Sigma$  are called the *singular values*
- We can approximate the full matrix by observing only the most important features
  - those with the largest singular values
- In the example, we calculate  $U$ ,  $V$ , and  $\Sigma$  (with the help of some linear algebra software) but retain only the two most important features by taking only the first two columns of  $U$  and  $V^T$

# Example for SVD-based recommendation

- SVD:  $M_k = U_k \times \Sigma_k \times V_k^T$

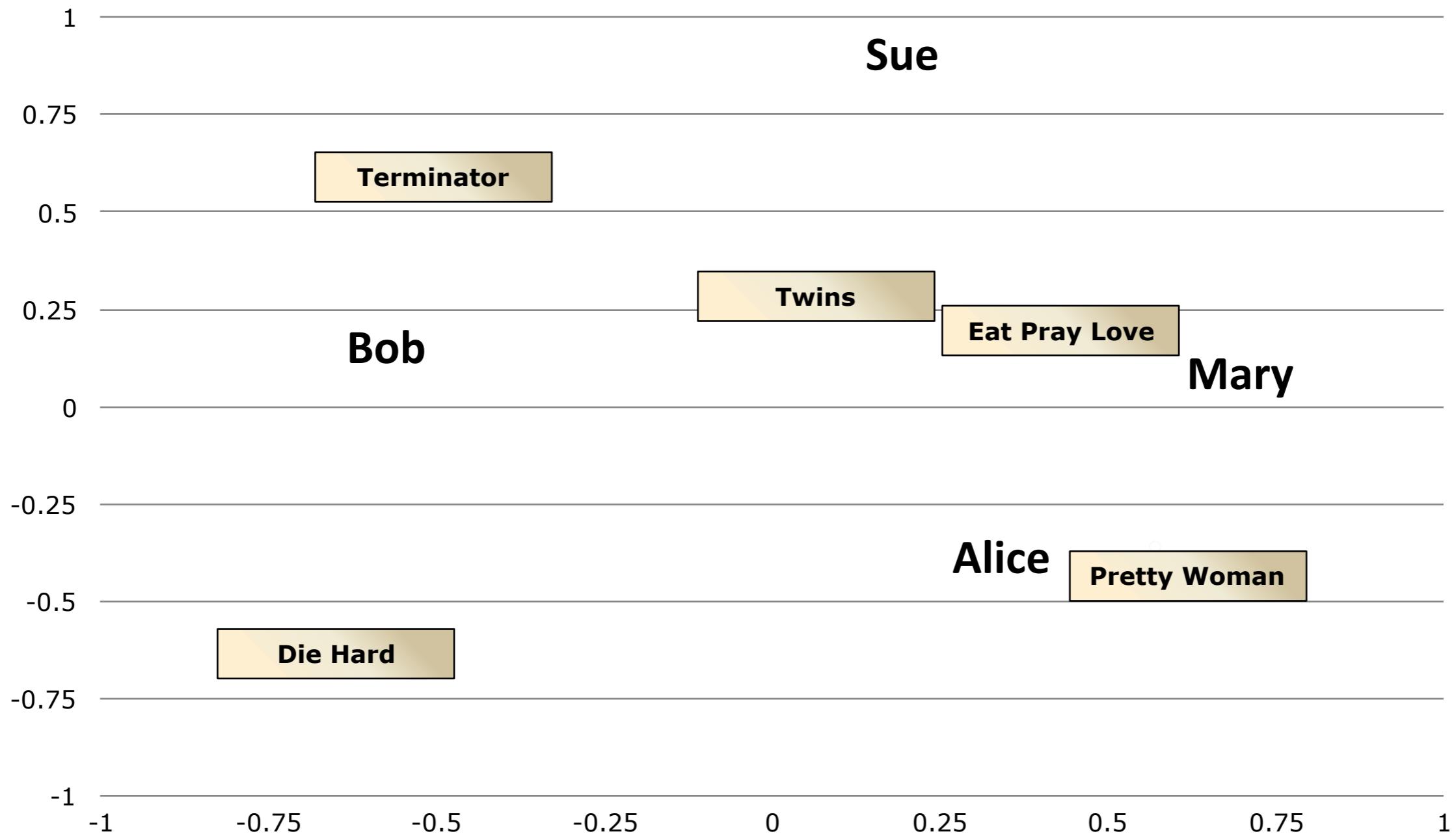
| $U_k$ | Dim1  | Dim2  |
|-------|-------|-------|
| Alice | 0.47  | -0.30 |
| Bob   | -0.44 | 0.23  |
| Mary  | 0.70  | -0.06 |
| Sue   | 0.31  | 0.93  |

| $V_k^T$ | Terminator | Die Hard | Twins | Eat Pray Love | Pretty Woman |
|---------|------------|----------|-------|---------------|--------------|
| Dim1    | -0.44      | -0.57    | 0.06  | 0.38          | 0.57         |
| Dim2    | 0.58       | -0.66    | 0.26  | 0.18          | -0.36        |

| $\Sigma_k$ | Dim1 | Dim2 |
|------------|------|------|
| Dim1       | 5.63 | 0    |
| Dim2       | 0    | 3.23 |

- Prediction:  $\hat{r}_{ui} = \bar{r}_u + U_k(Alice) \times \Sigma_k \times V_k^T(EPL)$   
 $= 3 + 0.84 = 3.84$

# The projection of $U$ and $V^T$ in the 2 dimensional space $(U_2, V_2^T)$



## Discussion about dimensionality reduction (Sarwar et al. 2000a)

- Matrix factorization
  - Generate low-rank approximation of matrix
  - Detection of latent factors
  - Projecting items and users in the same n-dimensional space
- Prediction quality can decrease because...
  - the original ratings are not taken into account
- Prediction quality can increase as a consequence of...
  - filtering out some "noise" in the data and
  - detecting nontrivial correlations in the data
- Depends on the right choice of the amount of data reduction

# Collaborative Filtering Issues

- Pros: 
- well-understood, works well in some domains, no knowledge engineering required
- Cons: 
- requires user community, sparsity problems, no integration of other knowledge sources, no explanation of results
- What is the best CF method?
  - In which situation and which domain? Inconsistent findings; always the same domains and data sets; differences between methods are often very small (1/100)
- How to evaluate the prediction quality?
  - MAE / RMSE: What does an MAE of 0.7 actually mean?
  - Serendipity (novelty and surprising effect of recommendations)
    - Not yet fully understood
- What about multi-dimensional ratings?



# Collaborative Filtering in R

Data Science Certification

# Workshop 4.16 Collaborative Filtering

Using the R markdown document and change to jester dataset and try collaborative filtering



# Hybrid Recommender System

Data Science Certification

# Most Common

- Combine Item Scores
- Combine Item Ranks
- Integrated Models
- Conditionally switch algorithms
- Deep Integration

# Combining Item Scores

Linear blends

$$s(i; u, q, x) = b + \beta_1 s_1(i; u, q, x) + \beta_2 s_2(i; u, q, x)$$

Feature-weighted linear stacking

$$s(i; \cdot) = b + f_1(u, i)s_1(i; \cdot) + f_2(u, i)s_2(i; \cdot)$$

# Combine Item Ranks

Combine Output of Recommender

| Model 1 | Model 2 | Score | Final Model |
|---------|---------|-------|-------------|
| A       | B       | 1.0   | B           |
| B       | D       | 0.8   | A           |
| C       | A       | 0.6   | D           |
| D       | C       | 0.4   | C           |
| E       | E       | 0.2   | E           |

# Integrated Models

- Matrix factorization techniques use both Rating and Content data
  - SVD-Feature
  - Factorization Machines
  - GPMF



# Hybrid Recommender System in R

Data Science Certification

```
data("MovieLense")
MovieLense100 <- MovieLense[rowCounts(MovieLense) >100,]
train <- MovieLense100[1:100]
test <- MovieLense100[101:103]

mix popular movies with a random recommendations for diversity and
rerecommend some movies the user liked.
recom <- HybridRecommender(
 Recommender(train, method = "POPULAR"),
 Recommender(train, method = "RANDOM"),
 Recommender(train, method = "RERECOMMEND"),
 weights = c(.6, .1, .3)
)

recom

getModel(recom)

as(predict(recom, test), "list")
```

```
Loading required package: registry
Recommender of type 'HYBRID' for 'ratingMatrix'
learned using NA users.
$recommender
$recommender[[1]]
Recommender of type 'POPULAR' for 'realRatingMatrix'
learned using 100 users.

$recommender[[2]]
Recommender of type 'RANDOM' for 'realRatingMatrix'
learned using 100 users.

$recommender[[3]]
Recommender of type 'RERECOMMEND' for 'ratingMatrix'
learned using 100 users.

$weights
[1] 0.6 0.1 0.3

[[1]]
[1] "Great Day in Harlem, A (1994)"
[2] "Dangerous Beauty (1998)"
[3] "Two or Three Things I Know About Her (1966)"
[4] "Hearts and Minds (1996)"
[5] "Brassed Off (1996)"
[6] "Santa with Muscles (1996)"
[7] "Boys, Les (1997)"
[8] "Tough and Deadly (1995)"
[9] "Crooklyn (1994)"
[10] "Saint of Fort Washington, The (1993)"

[[2]]
[1] "Great Day in Harlem, A (1994)"
[2] "Two or Three Things I Know About Her (1966)"
[3] "Dangerous Beauty (1998)"
[4] "Hearts and Minds (1996)"
[5] "Boys, Les (1997)"
[6] "Tough and Deadly (1995)"
[7] "Santa with Muscles (1996)"
[8] "Brassed Off (1996)"
[9] "Crooklyn (1994)"
[10] "Whole Wide World, The (1996)"

[[3]]
[1] "Dangerous Beauty (1998)"
[2] "Hearts and Minds (1996)"
[3] "Two or Three Things I Know About Her (1966)"
[4] "Boys, Les (1997)"
[5] "Crooklyn (1994)"
```

# Workshop 4.17 Hybrid Recommender

Try hybrid recommender system using other data



# Anomaly Detection

Data Science Certification

# Universality of IoT Technology

## Smart Transport & Logistics



Vehicle, asset, person & pet monitoring & controlling



Embedded Mobile



M2M & wireless sensor network

## Smart Factories

## Smart Agriculture



Agriculture automation

## Smart Energy



Energy consumption



Security & surveillance



Building management

## Smart Homes & Cities



Everyday things



Smart homes & cities

## Smart Retail



Telemedicine & healthcare

## Smart Health

- "From autonomous cars to intelligent personal assistants, **data is the lifeblood of a rapidly growing digital existence** – opening up opportunities previously unimagined by businesses" IDC

# Digital Transformation of the Physical World

McKinsey, GE, IBM, Cisco et al. estimate hundreds of billions dollar savings/efficiency improvements in the next 10 years

| Industry           | Past: Shelling a Product                   | Future: a Service                                                                                                 |
|--------------------|--------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Energy & utilities | Power networks/grids                       | On demand energy production/consumption                                                                           |
| Automotive         | Cars                                       | Transportation (assisted, autonomous driving)                                                                     |
| Agriculture        | Seeds                                      | Crop Yields                                                                                                       |
| Healthcare         | Diabetes pumps                             | Diabetes cares                                                                                                    |
| Food               | Packaged goods                             | Nutrition                                                                                                         |
| Cities             | Physical Urban infrastructure / Facilities | Smart city e-services (street lighting, urban noise/pollution/ traffic monitoring, parking/waste management etc.) |
| ....               | ....                                       | ....                                                                                                              |
| IT Industry        | Computers                                  | Computation                                                                                                       |



## INDUSTRY 1.0

**Mechanical production** using the power of water and steam



## INDUSTRY 2.0

Centralized electric power infrastructure; **mass production** by division of labor



## INDUSTRY 3.0

Digital computing & communication technology, **enhancing systems' intelligence**



## INDUSTRY 4.0

**Everybody & everything is networked** – networked information as a “huge brain”

1784

1870

1969

TODAY

# Types of IoT Data

Addresses/Unique Identifiers (nominal)

IP (IPv4, IPv6), Bluetooth, Zigbee, LoRa, RFID

Positional Data (continuous)

GPS (Longitude, Latitude, Altitude), WiFi (Longitude, Latitude)

Temporal Data (continuous)

Time and Date

Sensor Data (numerical)

Output of a device that detects and responds to some type of input from the physical environment (motion, position, environment, mass, biomarker)

Descriptive Data (categorical)

Objects, Processes, and Systems

# IoT Data Analytics

- Use Cases
  - Real-time monitoring and control
  - Failure detection and predictive maintenance
  - Malicious cyber activity detection
  - Product life-cycle management
  - Operational efficiency, optimization, self-adaptation
- Anomaly Detection

# Real Time Anomaly Detection



- Monitor Traffic Conditions



- Network and Cyber Security Detect Intrusion



- Monitor Biomarkers



- Predictive maintenance, Production safety Monitoring



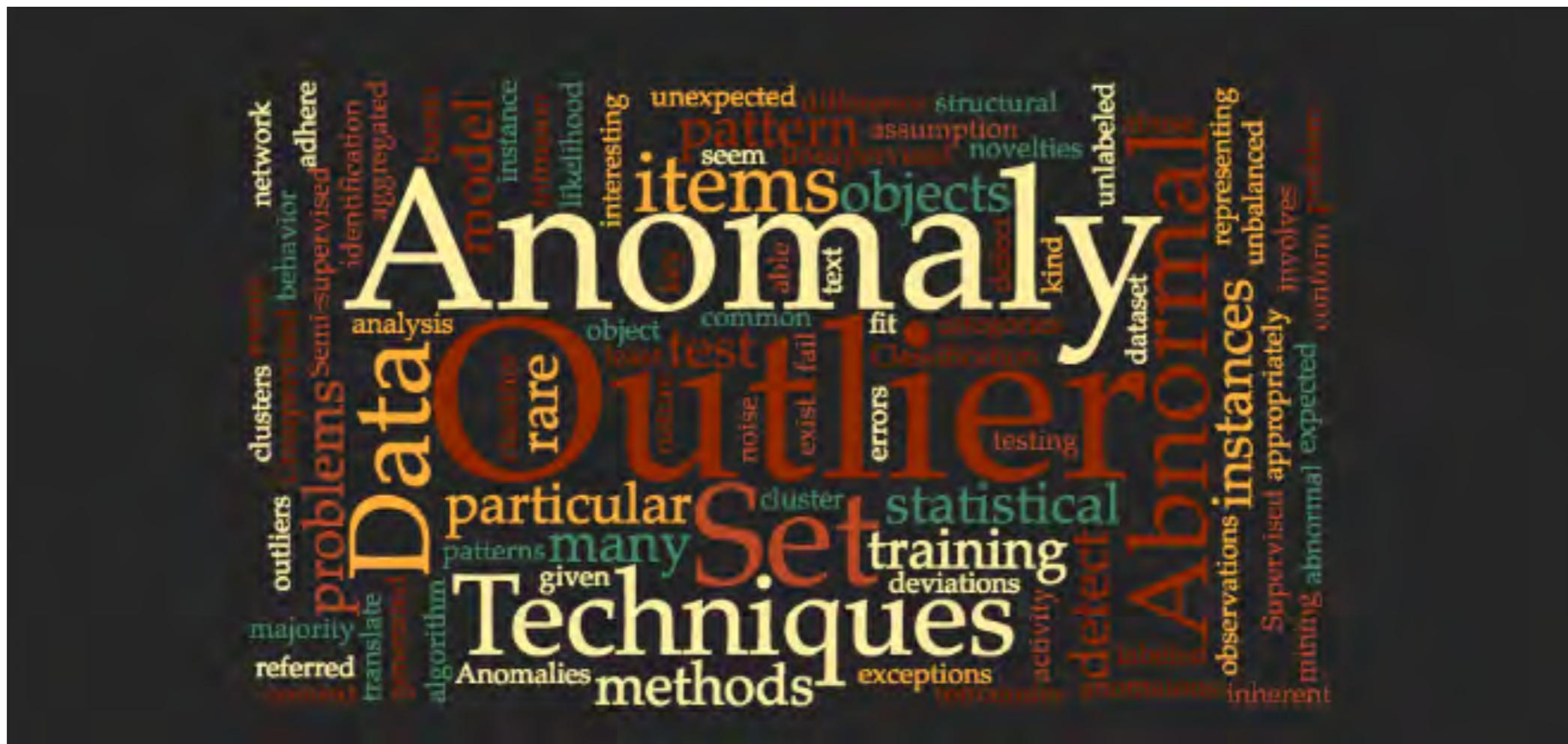
- Agriculture Pest, Water Control



- Monitor Energy Consumption

Define **alerts**, predict **faults**, detect **intrusions** & **threats**

# Data Anomaly Detection



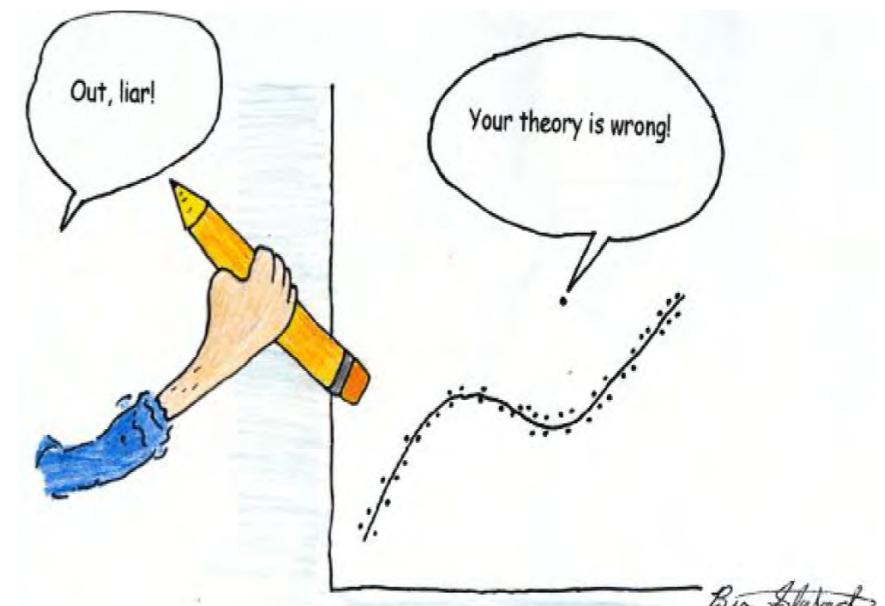
# Data Anomalies

What are anomalies?

"An **outlier** is an **observation** in a dataset that appears to be **inconsistent** with the remainder of that dataset" [Johnson 1992]

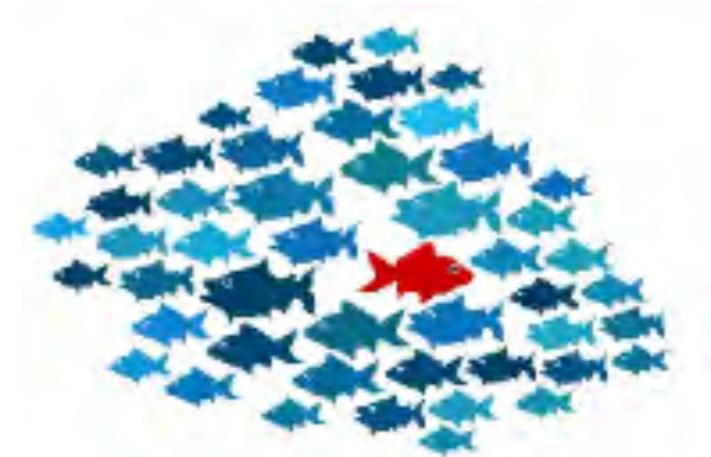
"An **outlier** is an **observation** that **deviates** so much from **other observations** as to **arouse suspicion** that it was generated by a **different mechanism**" [Hawkins 1980]

- **Anomaly**, or **outlier**, or **deviation**, or **novelty detection**, aims to measure whether a data point (or a subset) considerably differs than the remainder of the data points
  - On a scatter plot of the data, they lie far away from other data
  - Anomalies = Outliers + Novelties



# Cause of Data Anomalies

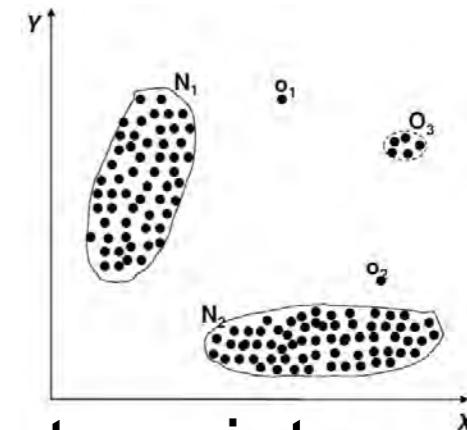
- Errors in the data collection or measurement process
  - Because of human **error** (intentional or not), a **problem** with a measuring device or the presence of noise
  - Insights extracted from “dirty data” are probably erroneous and thus the decisions to be made are **likely unsound** (e.g., incur a high rate of *false positives* and *false negatives*)
- Changes in the “data generation” process, e.g. some given statistical process (not an error, **novelties** in data)
  - Abnormal data deviate from this generating mechanism because it is of a **different type**
  - Novel class of observations provide **valuable insights**



# Types of Anomalies

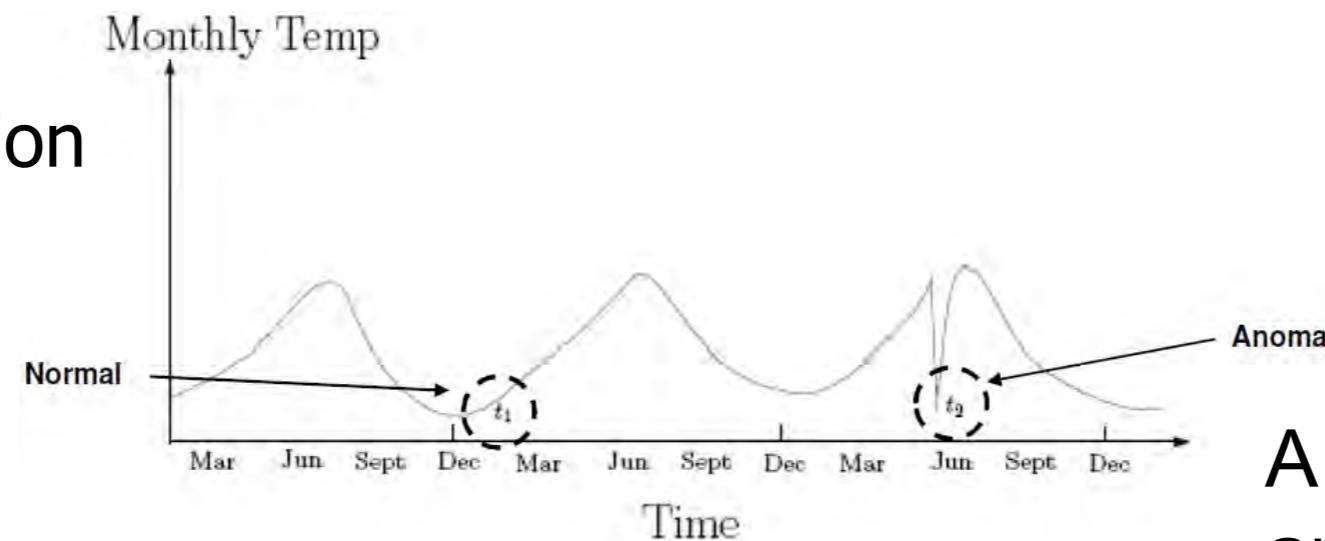
## Point Anomaly Detection

Data point anomalous w.r.t. to the rest of the data



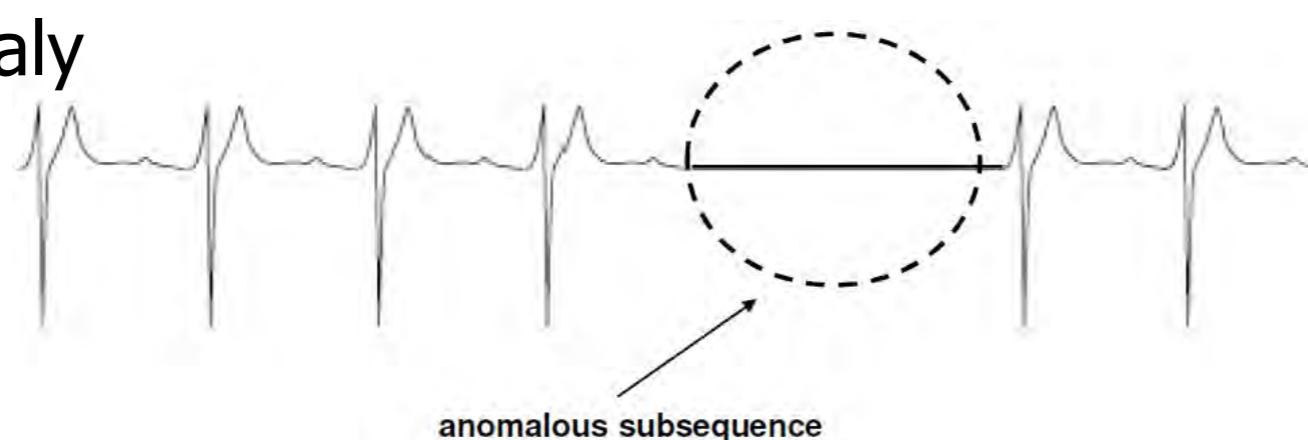
Data point anomalous in a specific **context**; also referred to as **conditional anomalies**

## Contextual Anomaly Detection

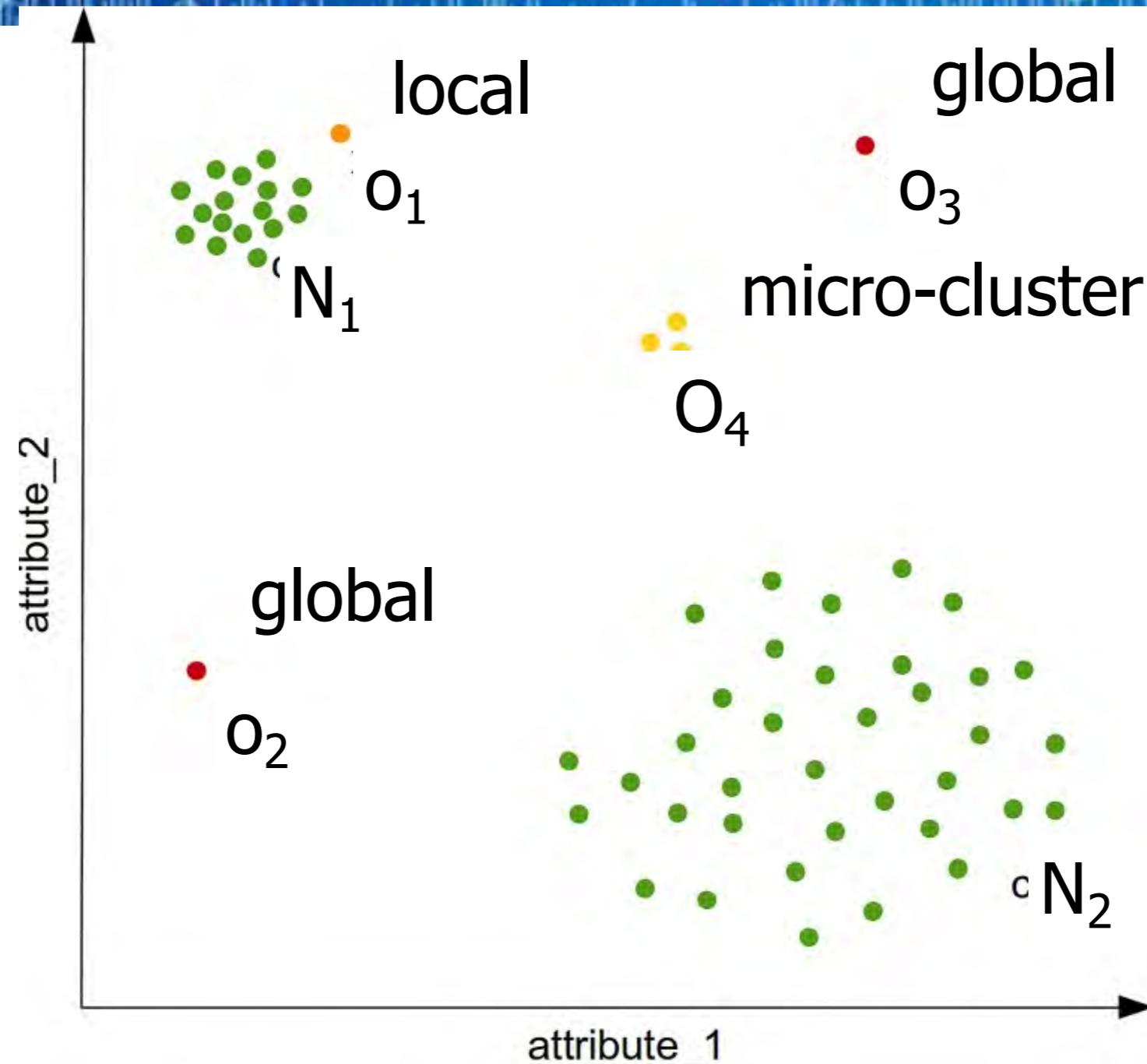


A **group** of **data** points is anomalous; the individual points within a group are not anomalous by themselves

## Group Anomaly Detection



# Scope of Anomaly Detection Methods

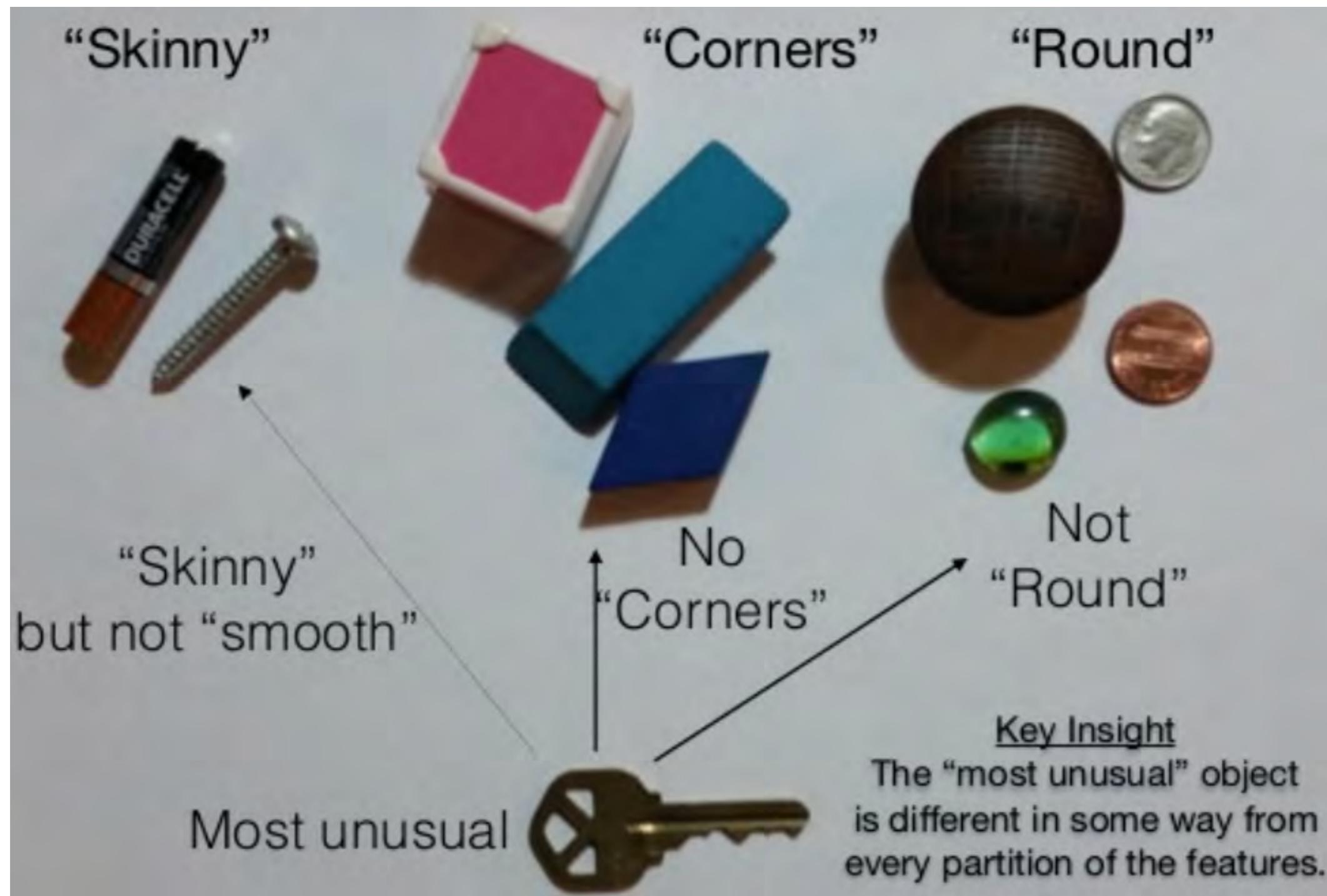


- **Swamping**: wrongly identifying normal instances as anomalies when normal instances are too close to anomalies
- **Masking** : an anomaly is close to another point, anomalous or not

# Example: Most Unusual Objects?



# Example: How to Distinguish Objects?



# High-Dimensional Data: Example

| Object  | Length/Width | Num Surfaces | Smooth |
|---------|--------------|--------------|--------|
| penny   | 1            | 3            | true   |
| dime    | 1            | 3            | true   |
| knob    | 1            | 4            | true   |
| box     | 1            | 6            | true   |
| eraser  | 2.75         | 6            | true   |
| block   | 1.6          | 6            | true   |
| screw   | 8            | 3            | false  |
| battery | 5            | 3            | true   |
| key     | 4.25         | 3            | false  |
| bead    | 1            | 2            | true   |

# Example: How to Distinguish Objects?

Humans used prior knowledge to [select features that separated objects](#)

“skinny”, “corners”, “round”, “smooth”

Objects have been separated based on the chosen features

[Each cluster has been examined to see](#)

which object fits less well in its cluster

and did not fit any other cluster

Learning from humans: [create features that capture object differences](#)

Length/width:

$\geq 1 \Rightarrow$  “skinny”

$= 1 \Rightarrow$  “round”

$< 1 \Rightarrow$  “tinny”

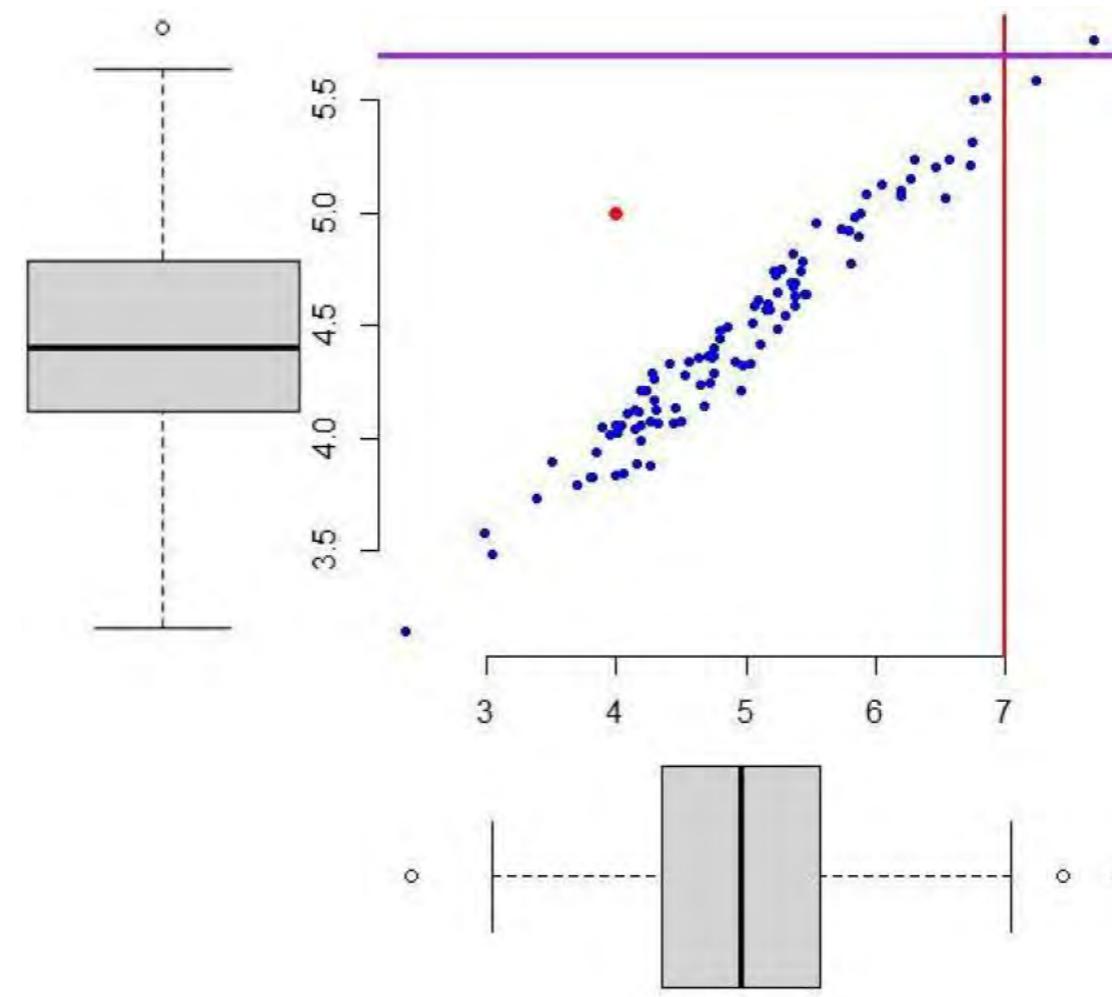
Number of surfaces:

distinct surfaces requires “edges” that have corners

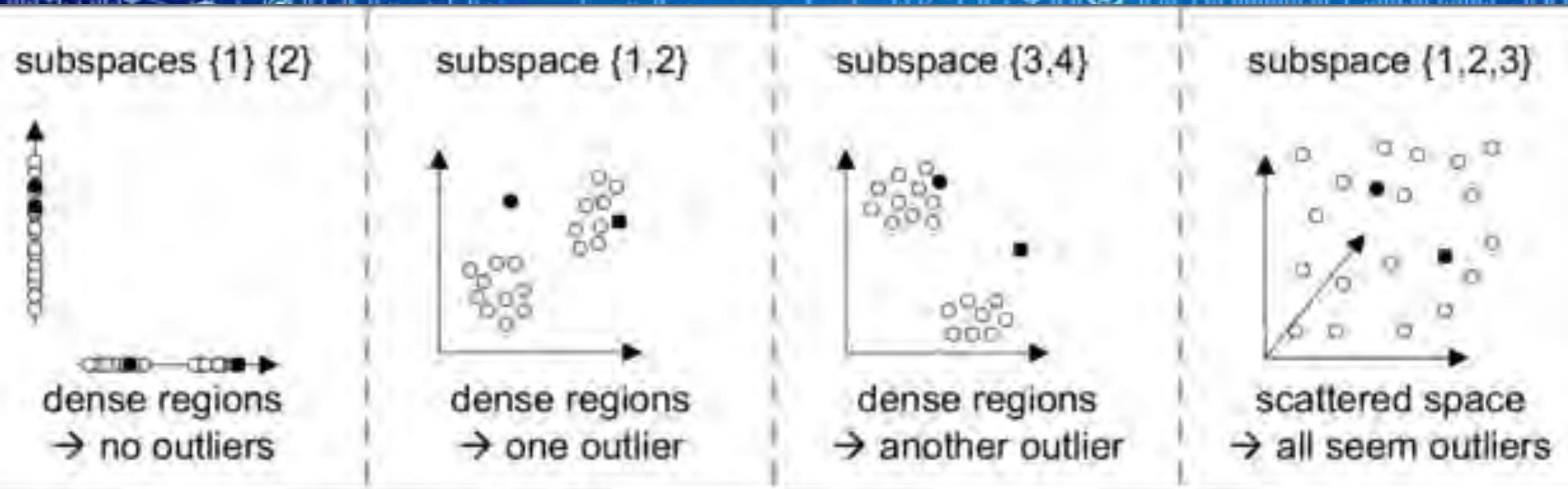
Smooth: true or false

# Multi-Dimensional Anomaly Detection

Anomaly detection in **multi-dimensional** datasets reveals more accurate behavior of the data but at the same time poses various challenges



# Multi-Dimensional Anomaly Detection



Data-snooping bias: given enough features, at least one feature subspace can be found for each data point such that it appears as an anomaly

# Fraud detection

- Detection of criminal activities occurring in commercial organizations.
- Malicious users might be:
  - Employees
  - Actual customers
  - Someone posing as a customer (identity theft)
- Types of fraud
  - Credit card fraud
  - Insurance claim fraud
  - Mobile / cell phone fraud
  - Insider trading
- Challenges
  - Fast and accurate real-time detection
  - Misclassification cost is very high



# Healthcare informatics

- Detect anomalous patient records
  - Indicate disease outbreaks, instrumentation errors, etc.
- Key challenges
  - Only normal labels available
  - Misclassification cost is very high
  - Data can be complex: spatio-temporal



# Industrial damage detection

- Detect faults and failures in complex industrial systems, structural damages, intrusions in electronic security systems, suspicious events in video surveillance, abnormal energy consumption, etc.
- Example: aircraft safety
  - anomalous aircraft (engine) / fleet usage
  - anomalies in engine combustion data
  - total aircraft health and usage management
- Key challenges
  - Data is extremely large, noisy, and unlabelled
  - Most of applications exhibit temporal behavior
  - Detected anomalous events typically require immediate intervention



# Image processing

- Detecting outliers in a image monitored over time
- Detecting anomalous regions within an image
- Used in
  - mammography image analysis
  - video surveillance
  - satellite image analysis
- Key Challenges
  - Detecting collective anomalies
  - Data sets are very large



# Use of data labels in anomaly detection

- Supervised anomaly detection
  - Labels available for both normal data and anomalies
  - Similar to classification with high class imbalance
- Semi-supervised anomaly detection
  - Labels available only for normal data
- Unsupervised anomaly detection
  - No labels assumed
  - Based on the assumption that anomalies are very rare compared to normal data

# Output of anomaly detection

- **Label**
  - Each test instance is given a *normal* or *anomaly* label
  - Typical output of classification-based approaches
- **Score**
  - Each test instance is assigned an anomaly score
    - allows outputs to be ranked
    - requires an additional threshold parameter

# Variants of anomaly detection problem

- Given a dataset D, find all the data points  $\mathbf{x} \in D$  with anomaly scores greater than some threshold t.
- Given a dataset D, find all the data points  $\mathbf{x} \in D$  having the top-n largest anomaly scores.
- Given a dataset D, containing mostly normal data points, and a test point  $\mathbf{x}$ , compute the anomaly score of  $\mathbf{x}$  with respect to D.

# Unsupervised anomaly detection

- No labels available
- Based on assumption that anomalies are very rare compared to “normal” data
- **General steps**
  - Build a profile of “normal” behavior
    - summary statistics for overall population
    - model of multivariate data distribution
  - Use the “normal” profile to detect anomalies
    - anomalies are observations whose characteristics differ significantly from the normal profile

# Techniques for anomaly detection

- Statistical
- Proximity-based
- Density-based
- Clustering-based

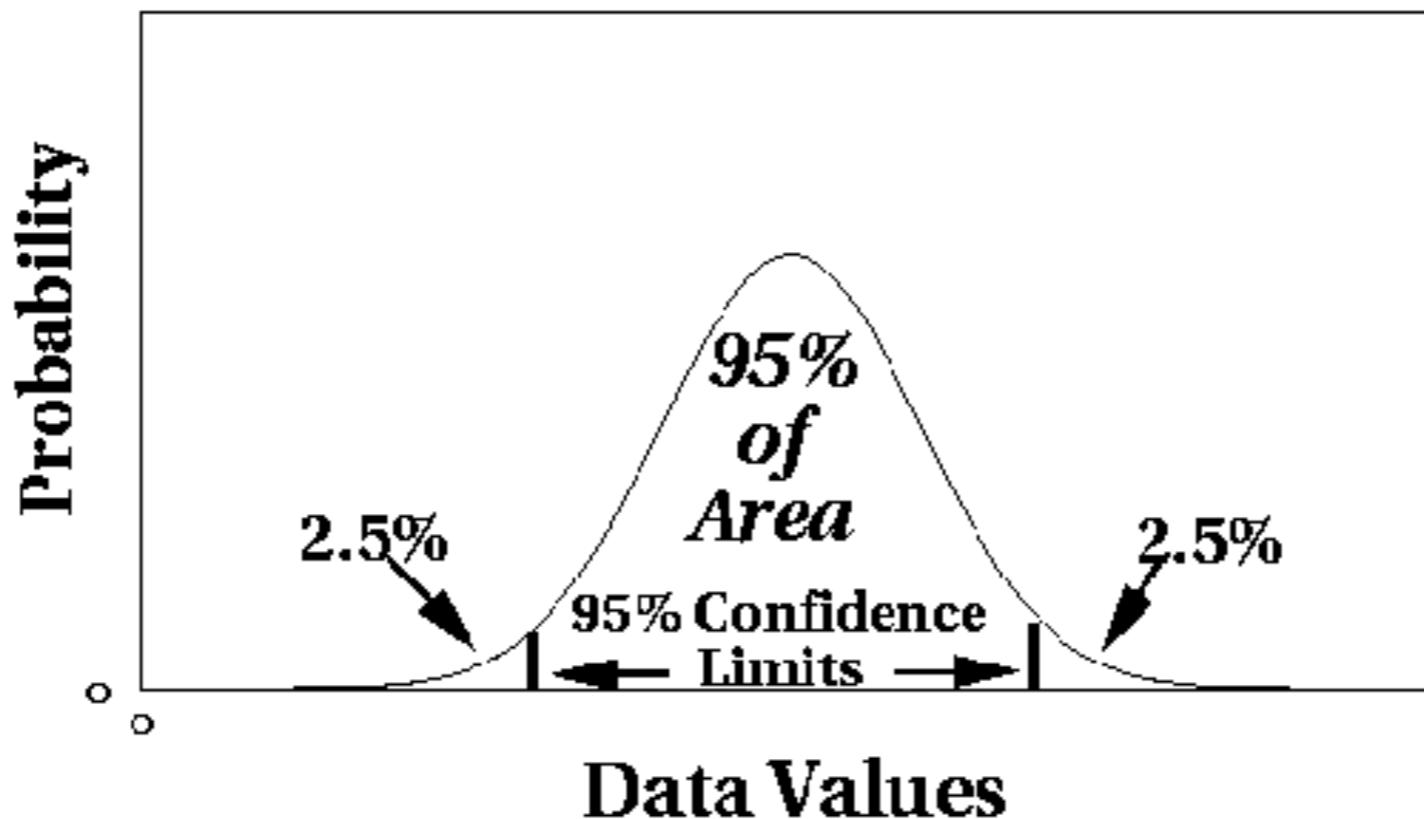
# Statistical outlier detection

- Outliers are objects that are fit poorly by a statistical model.
- Estimate a parametric model describing the distribution of the data
- Apply a statistical test that depends on
  - Properties of test instance
  - Parameters of model (e.g., mean, variance)
  - Confidence limit (related to number of expected outliers)

# Statistical outlier detection

Univariate Gaussian distribution

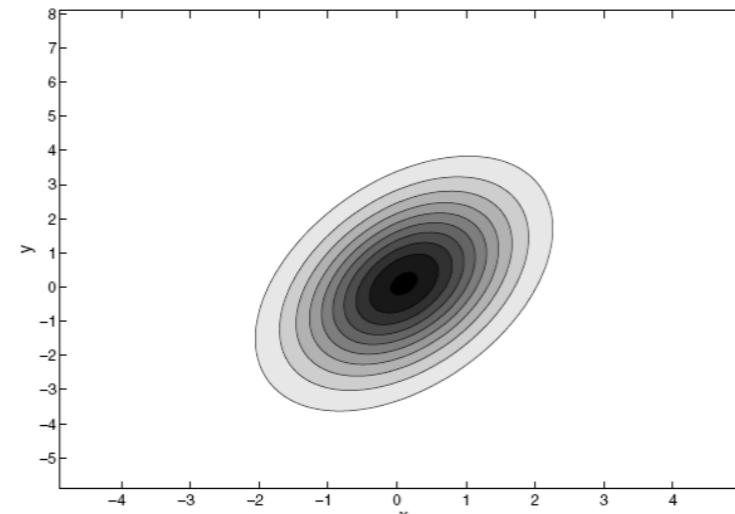
Outlier defined by z-score > threshold



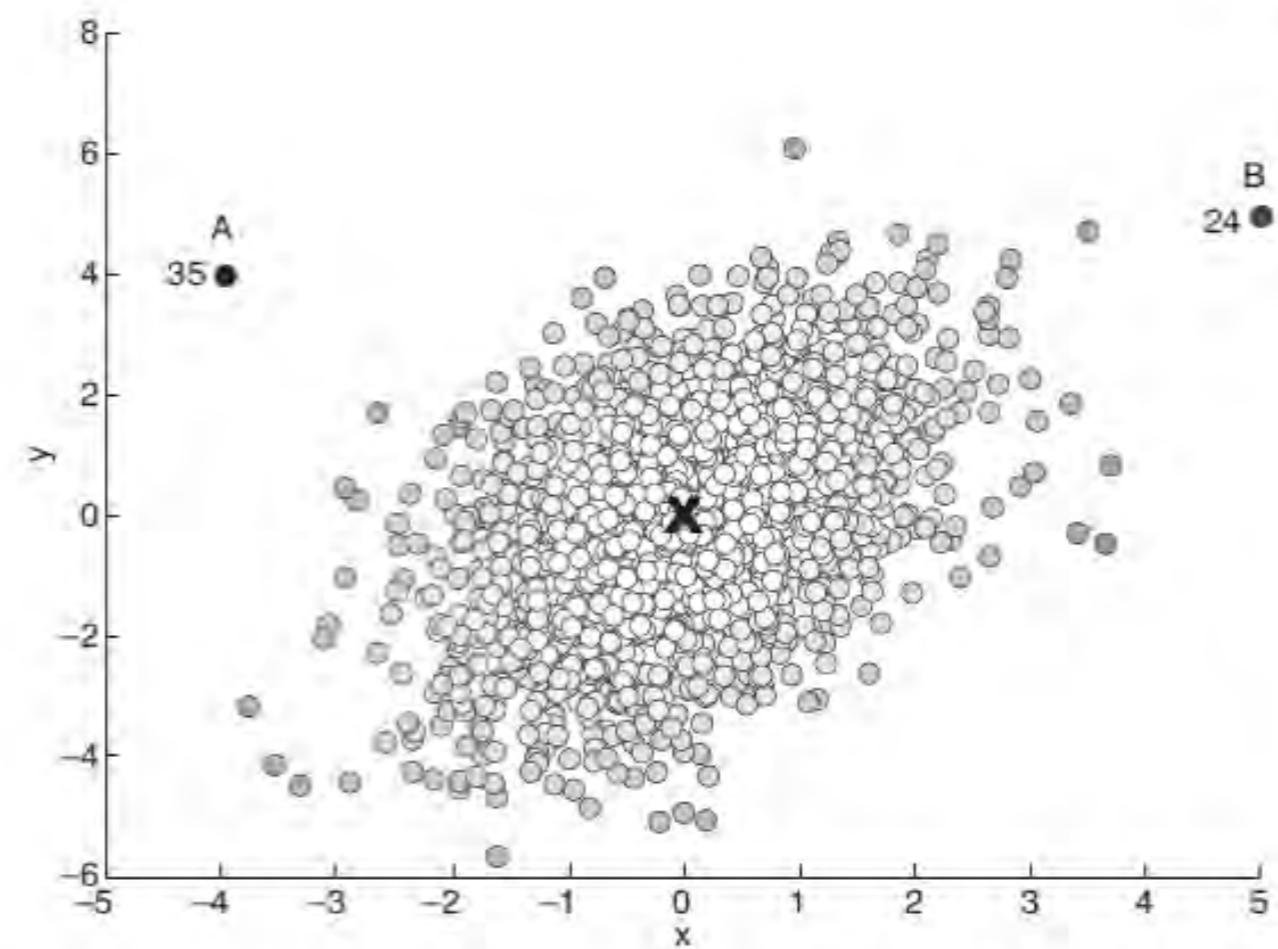
# Statistical anomaly detection

Multivariate Gaussian distribution

Outlier defined by Mahalanobis distance > threshold



| Distance |           |             |
|----------|-----------|-------------|
|          | Euclidean | Mahalanobis |
| A        | 5.7       | 35          |
| B        | 7.1       | 24          |



# Grubbs' test

- Detect outliers in univariate data
- Assume data comes from normal distribution
- Detects one outlier at a time, remove the outlier, and repeat

- $H_0$ : There is no outlier in data

- $H_A$ : There is at least one outlier

- Grubbs' test statistic:

$$G = \frac{\max |X - \bar{X}|}{S}$$

- Reject  $H_0$  if:

$$G > \frac{(N-1)}{\sqrt{N}} \sqrt{\frac{t_{(\alpha/2, N-2)}^2}{N-2 + t_{(\alpha/2, N-2)}^2}}$$

# Likelihood approach

- Assume the dataset D contains samples from a mixture of two probability distributions:
  - M (majority distribution)
  - A (anomalous distribution)
- General approach:
  - Initially, assume all the data points belong to M
  - Let  $L_t(D)$  be the log likelihood of D at time t
  - For each point  $x_t$  that belongs to M, move it to A
    - Let  $L_{t+1}(D)$  be the new log likelihood.
    - Compute the difference,  $\Delta = L_t(D) - L_{t+1}(D)$
    - If  $\Delta > c$  (some threshold), then  $x_t$  is declared as an anomaly and moved permanently from M to A

# Likelihood approach

Data distribution,  $D = (1 - \lambda) M + \lambda A$

$M$  is a probability distribution estimated from data

Can be based on any modeling method (naïve Bayes, maximum entropy, etc)

$A$  is initially assumed to be uniform distribution

Likelihood at time  $t$ :

$$L_t(D) = \prod_{i=1}^N P_D(x_i) = \left( (1 - \lambda)^{|M_t|} \prod_{x_i \in M_t} P_{M_t}(x_i) \right) \left( \lambda^{|A_t|} \prod_{x_i \in A_t} P_{A_t}(x_i) \right)$$

$$LL_t(D) = |M_t| \log(1 - \lambda) + \sum_{x_i \in M_t} \log P_{M_t}(x_i) + |A_t| \log \lambda + \sum_{x_i \in A_t} \log P_{A_t}(x_i)$$

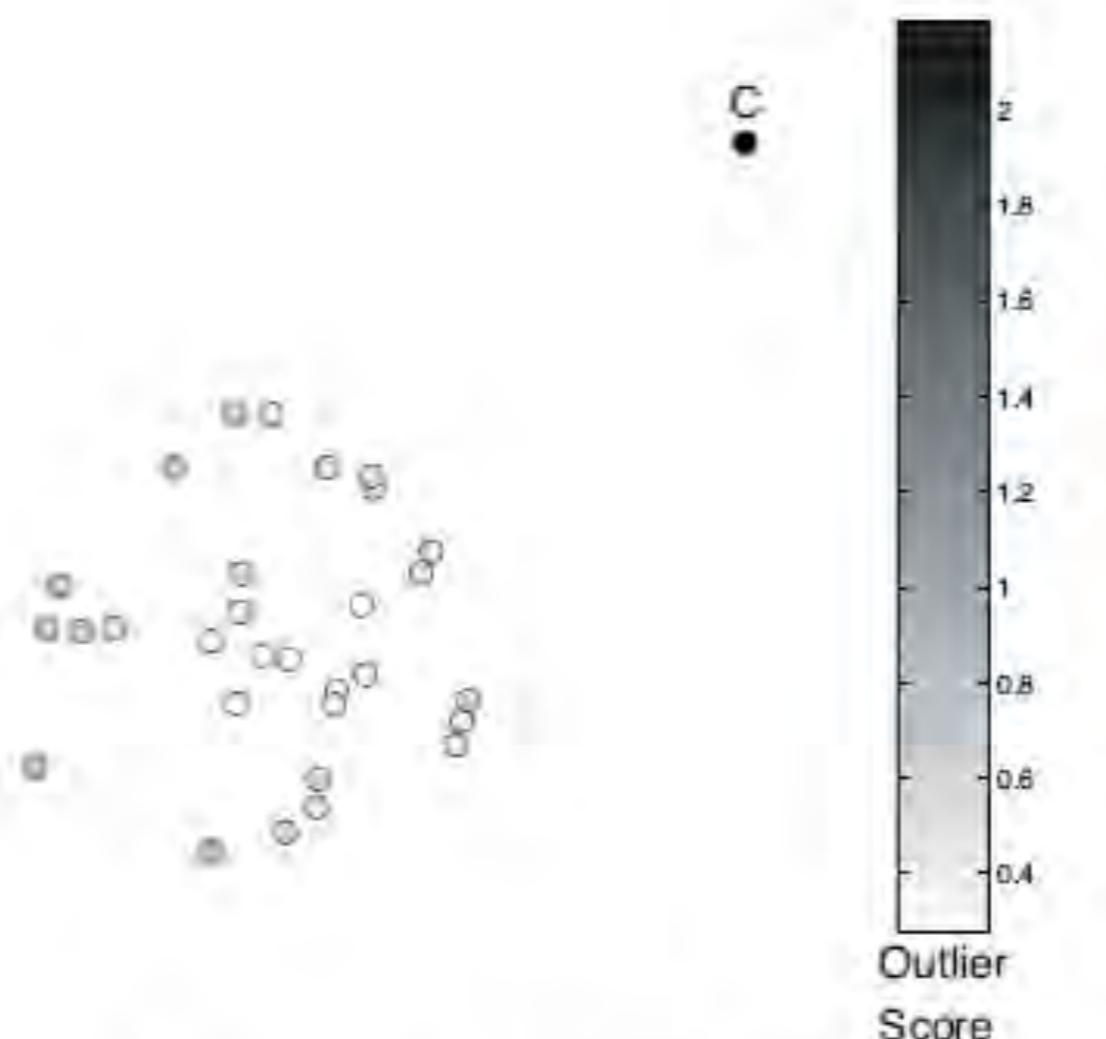
# Statistical outlier detection

- Pros
  - Statistical tests are well-understood and well-validated.
  - Quantitative measure of degree to which object is an outlier.
- Cons
  - Data may be hard to model parametrically.
    - multiple modes
    - variable density
  - In high dimensions, data may be insufficient to estimate true distribution.

# Proximity-based outlier detection

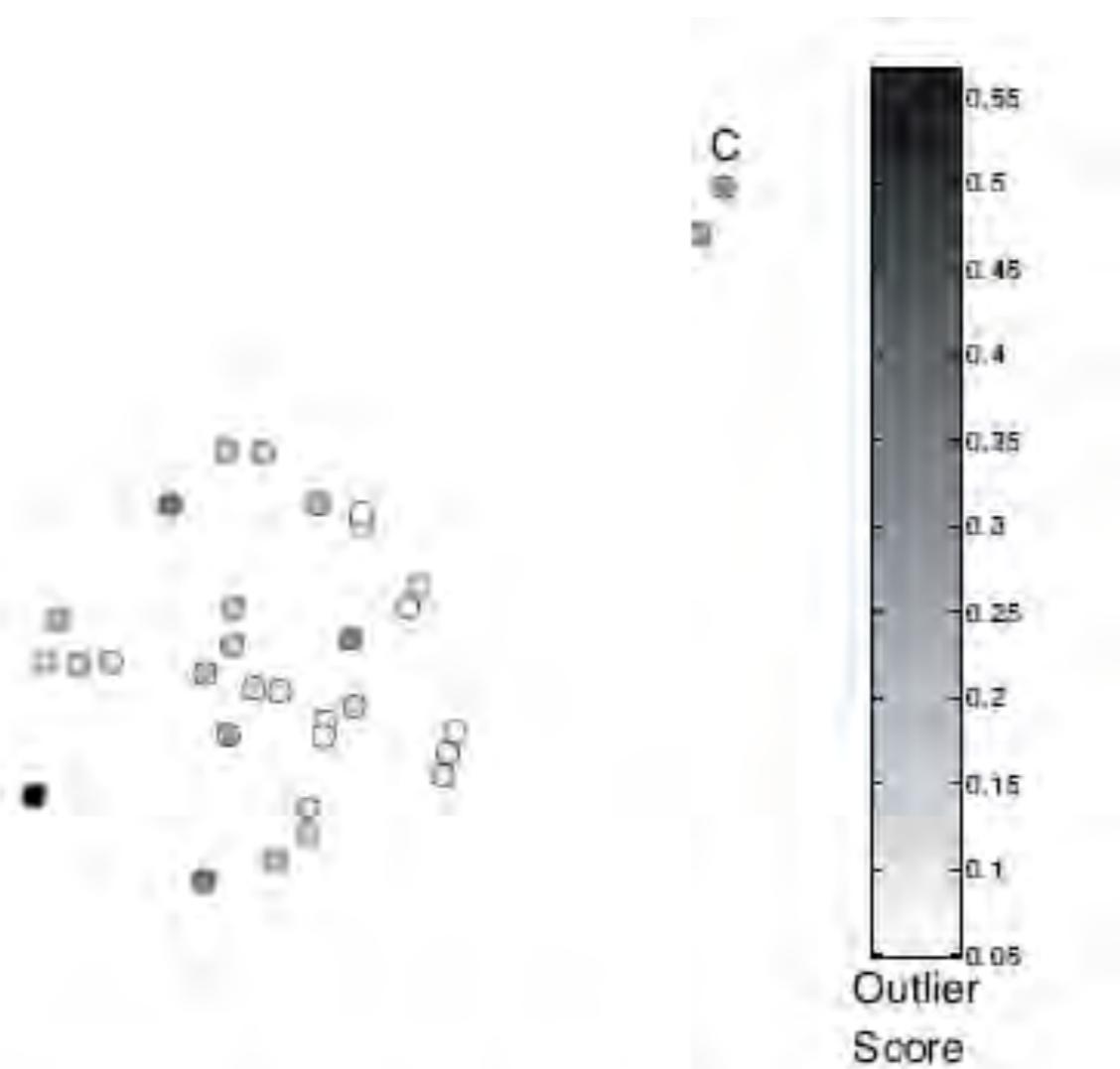
- **Outliers are objects far away from other objects.**
- Common approach:
  - Outlier score is distance to  $k^{\text{th}}$  nearest neighbor.
  - Score sensitive to choice of  $k$ .

# Proximity-based outlier detection



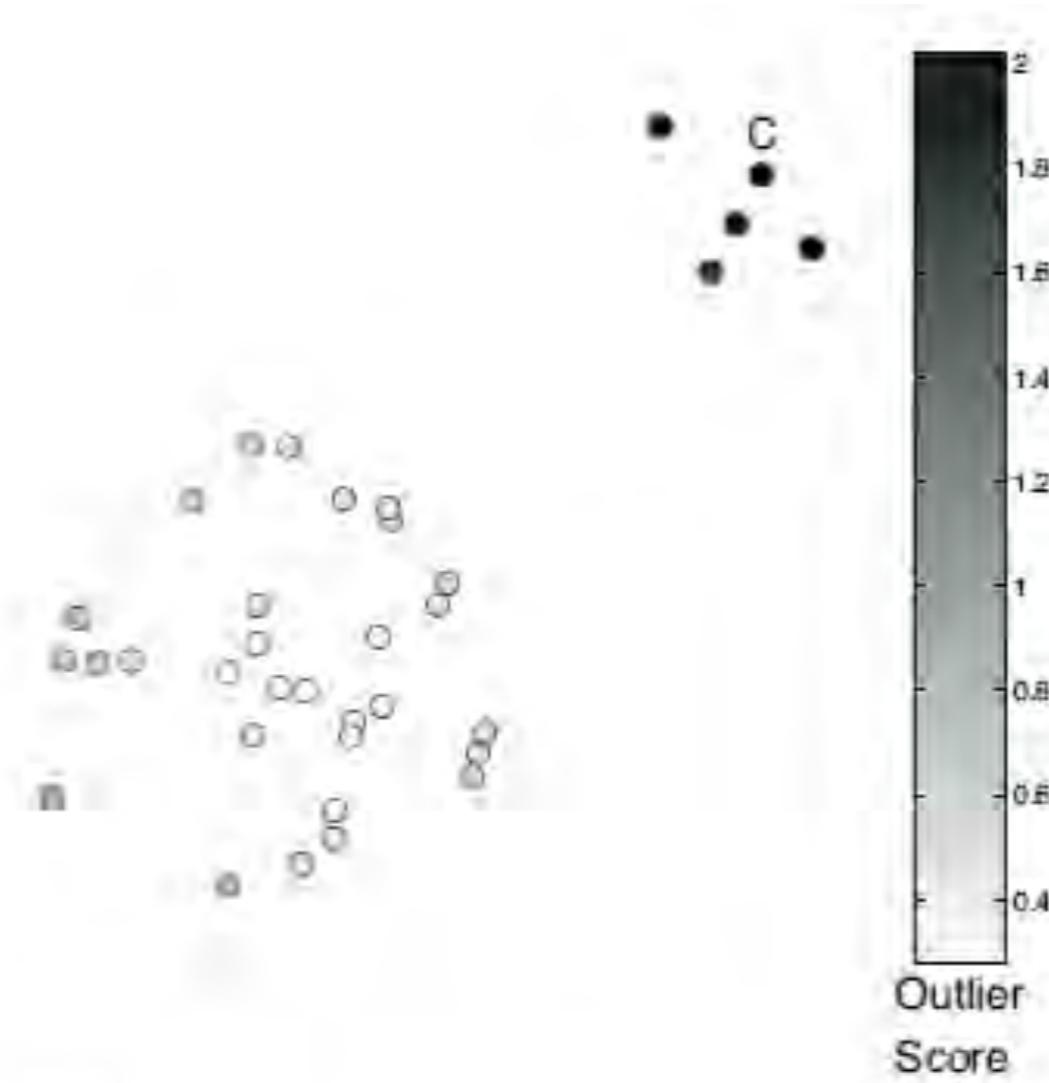
**Figure 10.4.** Outlier score based on the distance to fifth nearest neighbor.

# Proximity-based outlier detection



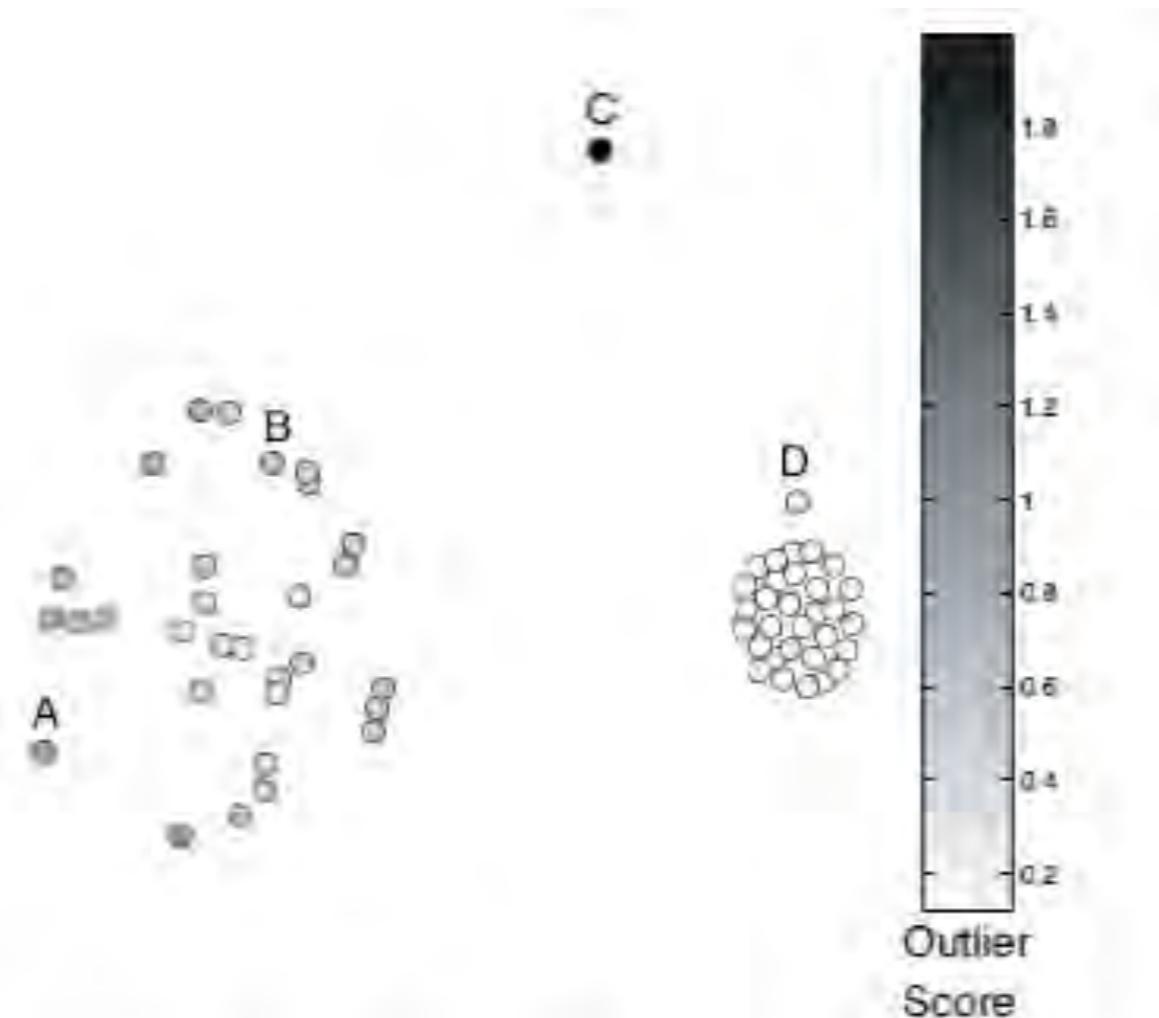
**Figure 10.5.** Outlier score based on the distance to the first nearest neighbor. Nearby outliers have low outlier scores.

# Proximity-based outlier detection



**Figure 10.6.** Outlier score based on distance to the fifth nearest neighbor. A small cluster becomes an outlier.

# Proximity-based outlier detection



**Figure 10.7.** Outlier score based on the distance to the fifth nearest neighbor. Clusters of differing density.

# Proximity-based outlier detection

- Pros
  - Easier to define a proximity measure for a dataset than determine its statistical distribution.
  - Quantitative measure of degree to which object is an outlier.
  - Deals naturally with multiple modes.
- Cons
  - $O(n^2)$  complexity.
  - Score sensitive to choice of  $k$ .
  - Does not work well if data has widely variable density.

# Density-based outlier detection

- Outliers are objects in regions of **low density**.

- Outlier score is inverse of density around object.
- Scores usually based on proximities.
- Example scores:

- Reciprocal of average distance to  $k$  nearest neighbors:

$$\text{density}(\mathbf{x}, k) = \left( \frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x}, k)} \text{distance}(\mathbf{x}, \mathbf{y}) \right)^{-1}$$

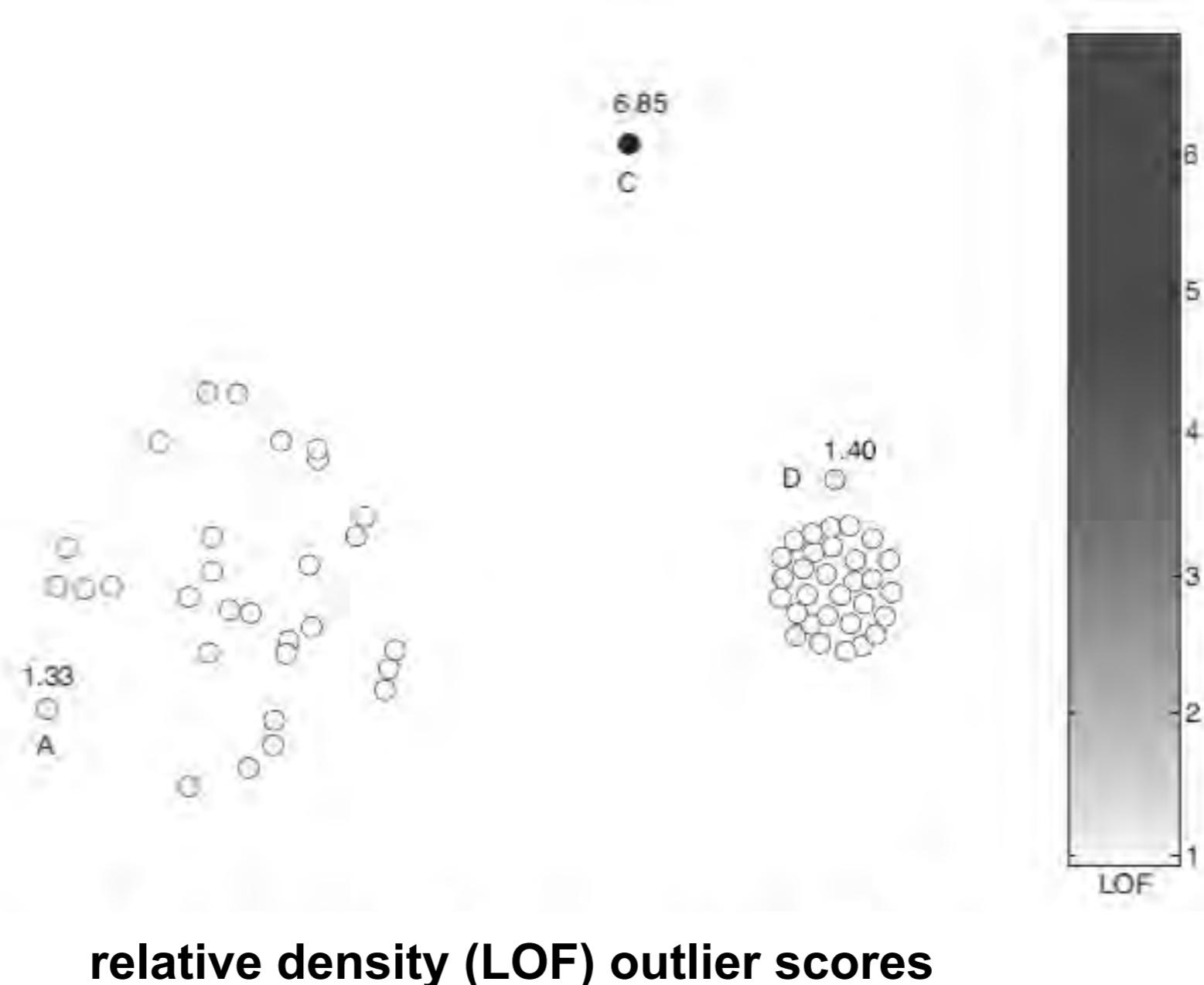
- Number of objects within fixed radius  $d$  (DBSCAN).
- These two example scores work poorly if data has variable density.

# Density-based outlier detection

- Relative density outlier score (Local Outlier Factor, LOF)
  - Reciprocal of average distance to  $k$  nearest neighbors, relative to that of those  $k$  neighbors.

$$\text{relative density}(\mathbf{x}, k) = \frac{\text{density}(\mathbf{x}, k)}{\frac{1}{k} \sum_{\mathbf{y} \in N(\mathbf{x}, k)} \text{density}(\mathbf{y}, k)}$$

# Density-based outlier detection



# Density-based outlier detection

- **Pros**
  - Quantitative measure of degree to which object is an outlier.
  - Can work well even if data has variable density.
- **Cons**
  - $O(n^2)$  complexity
  - Must choose parameters
    - $k$  for nearest neighbor
    - $d$  for distance threshold

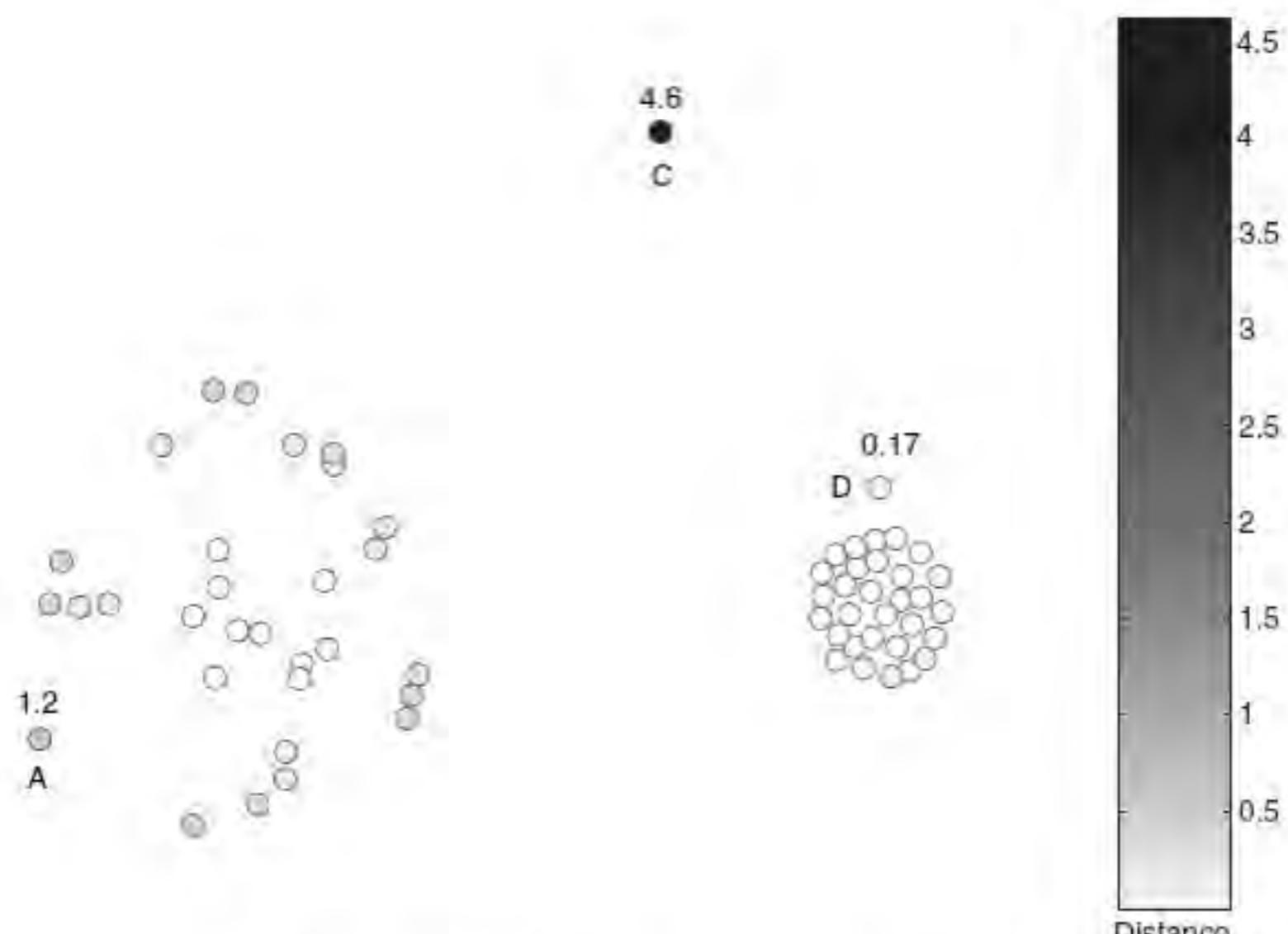
# Cluster-based outlier detection

- **Outliers are objects that do not belong strongly to any cluster.**
- Approaches:
  - Assess degree to which object belongs to any cluster.
  - Eliminate object(s) to improve objective function.
  - Discard small clusters far from other clusters.
- Issue:
  - Outliers may affect initial formation of clusters.

# Cluster-based outlier detection

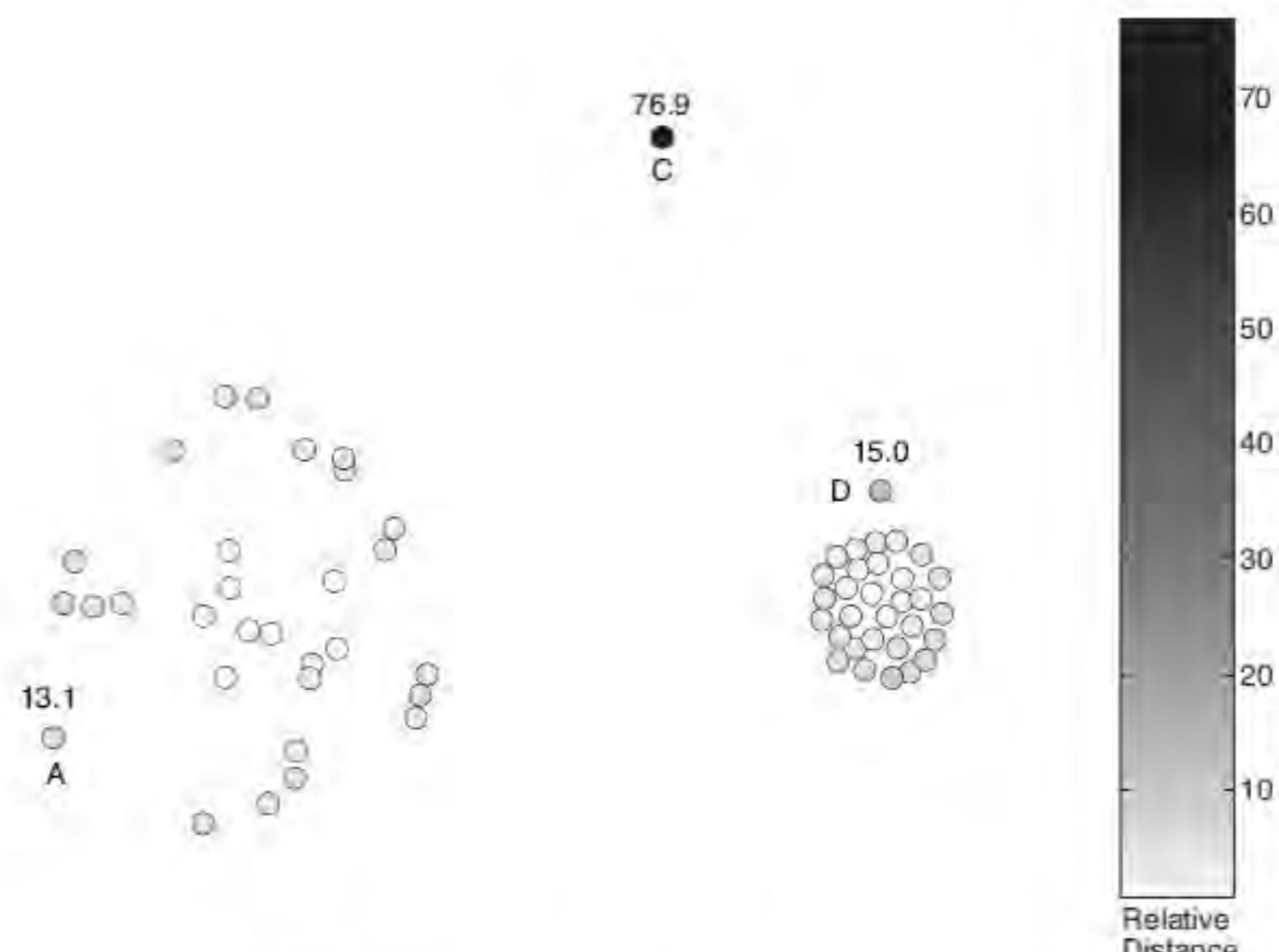
- Assess degree to which object belongs to any cluster.
- For prototype-based clustering (e.g. k-means), use distance to cluster centers.
  - To deal with variable density clusters, use relative distance:
$$\frac{\text{distance}(\mathbf{x}, \text{centroid}_C)}{\text{median}(\{\forall_{x' \in C} \text{distance}(\mathbf{x}', \text{centroid}_C)\})}$$
- Similar concepts for density-based or connectivity-based clusters.

# Cluster-based outlier detection



**distance of points from nearest centroid**

# Cluster-based outlier detection



**relative distance of points from nearest centroid**

# Cluster-based outlier detection

Eliminate object(s) to improve objective function.

- 1) Form initial set of clusters.
- 2) Remove the object which most improves objective function.
- 3) Repeat step 2) until ...

# Cluster-based outlier detection

- Discard small clusters far from other clusters.
- Need to define thresholds for “small” and “far”.

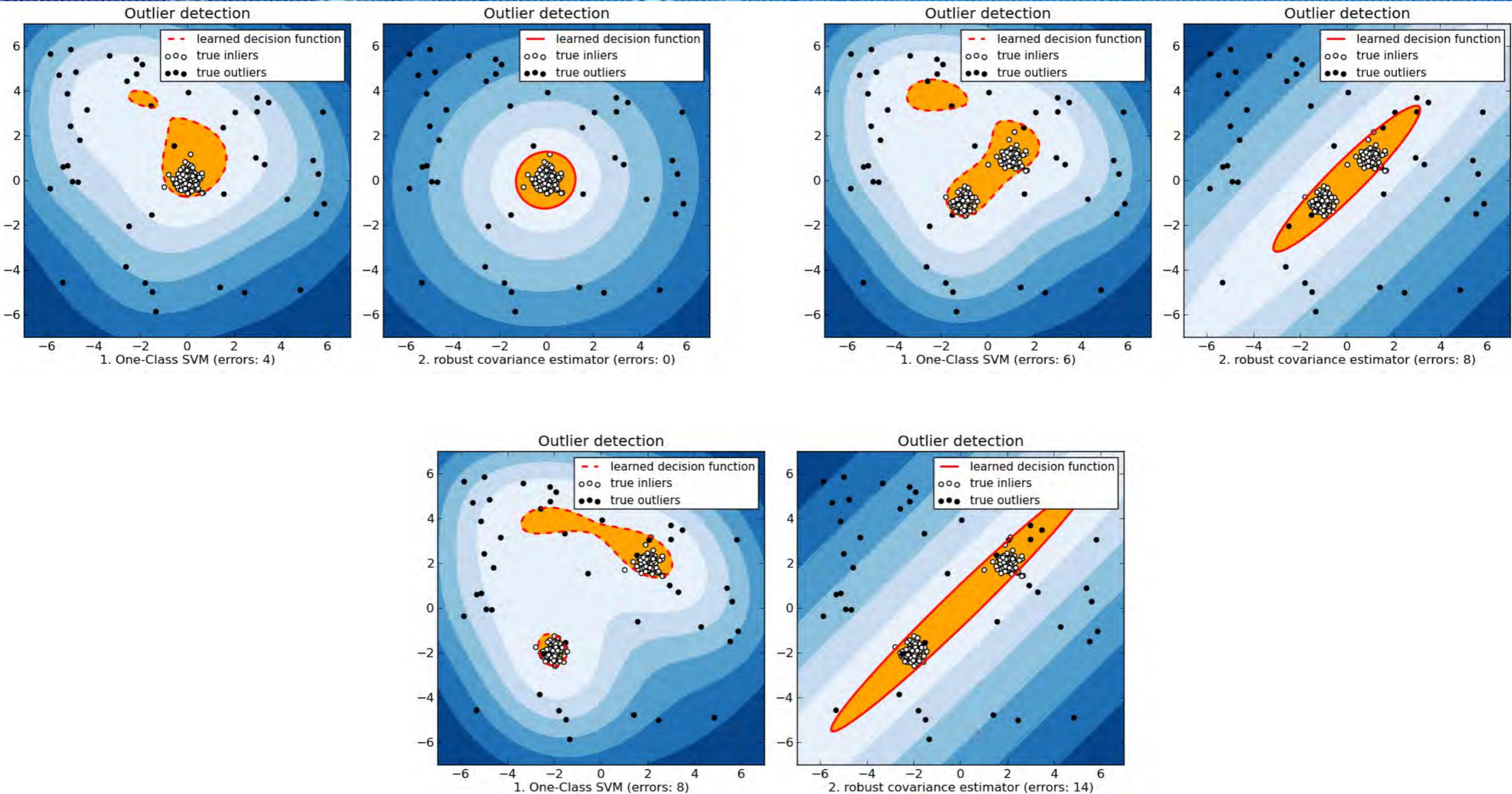
# Cluster-based outlier detection

- **Pro:**
  - Some clustering techniques have  $O(n)$  complexity.
  - Extends concept of outlier from single objects to groups of objects.
- **Cons:**
  - Requires thresholds for minimum size and distance.
  - Sensitive to number of clusters chosen.
  - Hard to associate outlier score with objects.
  - Outliers may affect initial formation of clusters.

# One-class support vector machines

- **Data is unlabelled, unlike usual SVM setting.**
- **Goal:** find hyperplane (in higher-dimensional kernel space) which encloses as much data as possible with minimum volume.
  - Tradeoff between amount of data enclosed and tightness of enclosure; controlled by regularization of slack variables.

# One-class SVM vs. Gaussian envelope

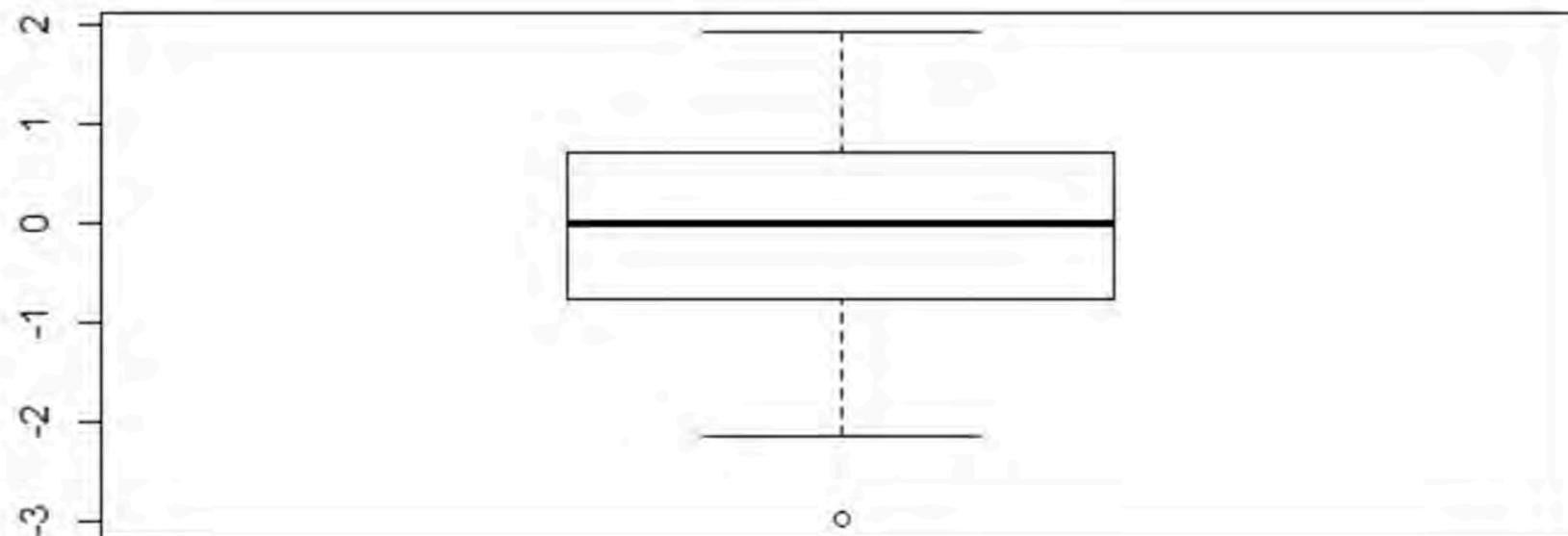




# Anomaly Detection using R

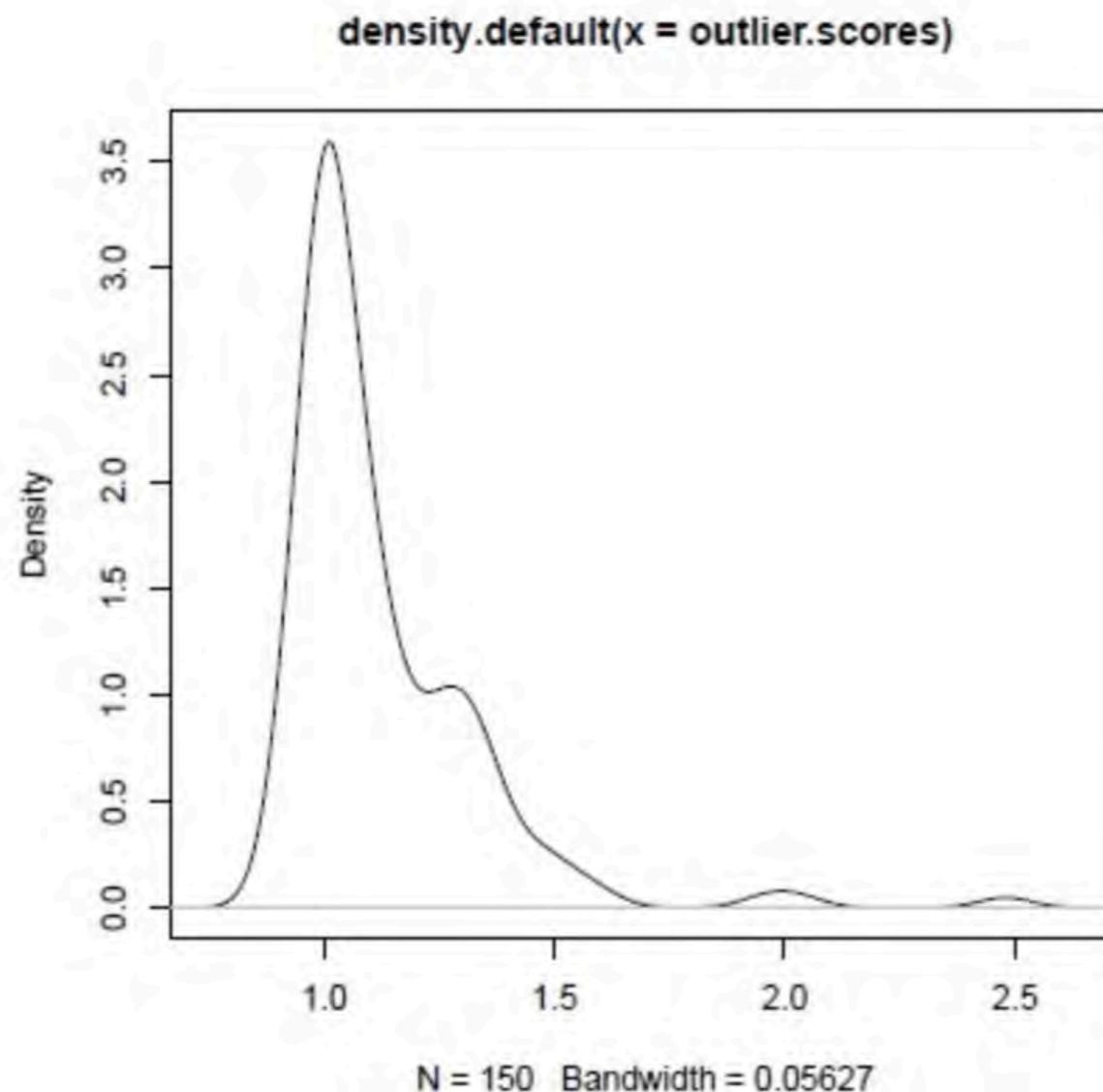
# Statistics Based

```
> y <- rnorm(100)
> boxplot(y)
> identify(rep(1, length(y)), y, labels = seq_along(y))
```



# Density-based

```
> library(DMwR)
> # remove "Species", which is a categorical column
> iris2 <- iris[,1:4]
> outlier.scores <- lofactor(iris2, k=5)
> plot(density(outlier.scores))
```

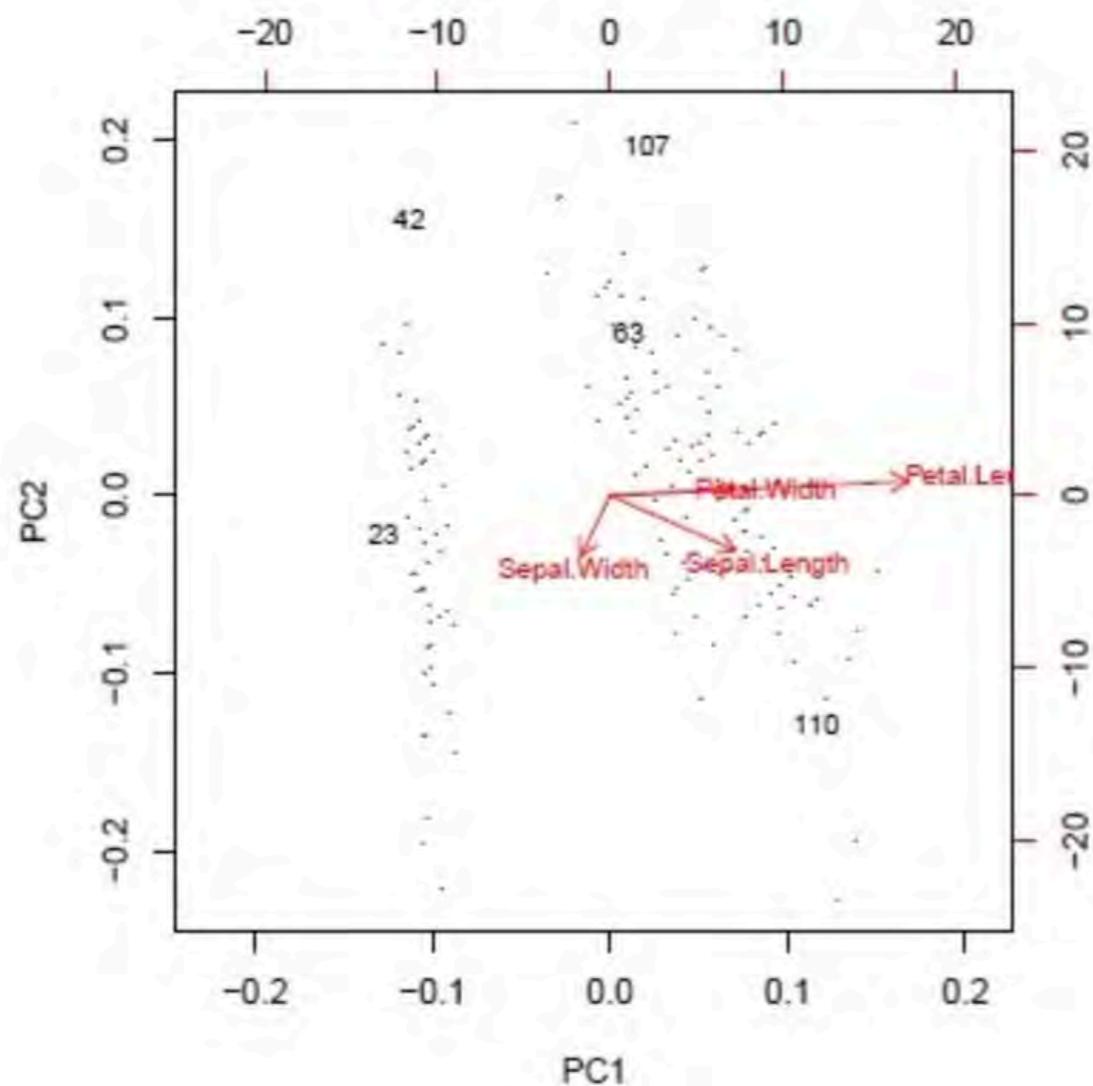


```
> # pick top 5 as outliers
> outliers <- order(outlier.scores, decreasing=T)[1:5]
> # who are outliers
> print(outliers)
[1] 42 107 23 110 63
```

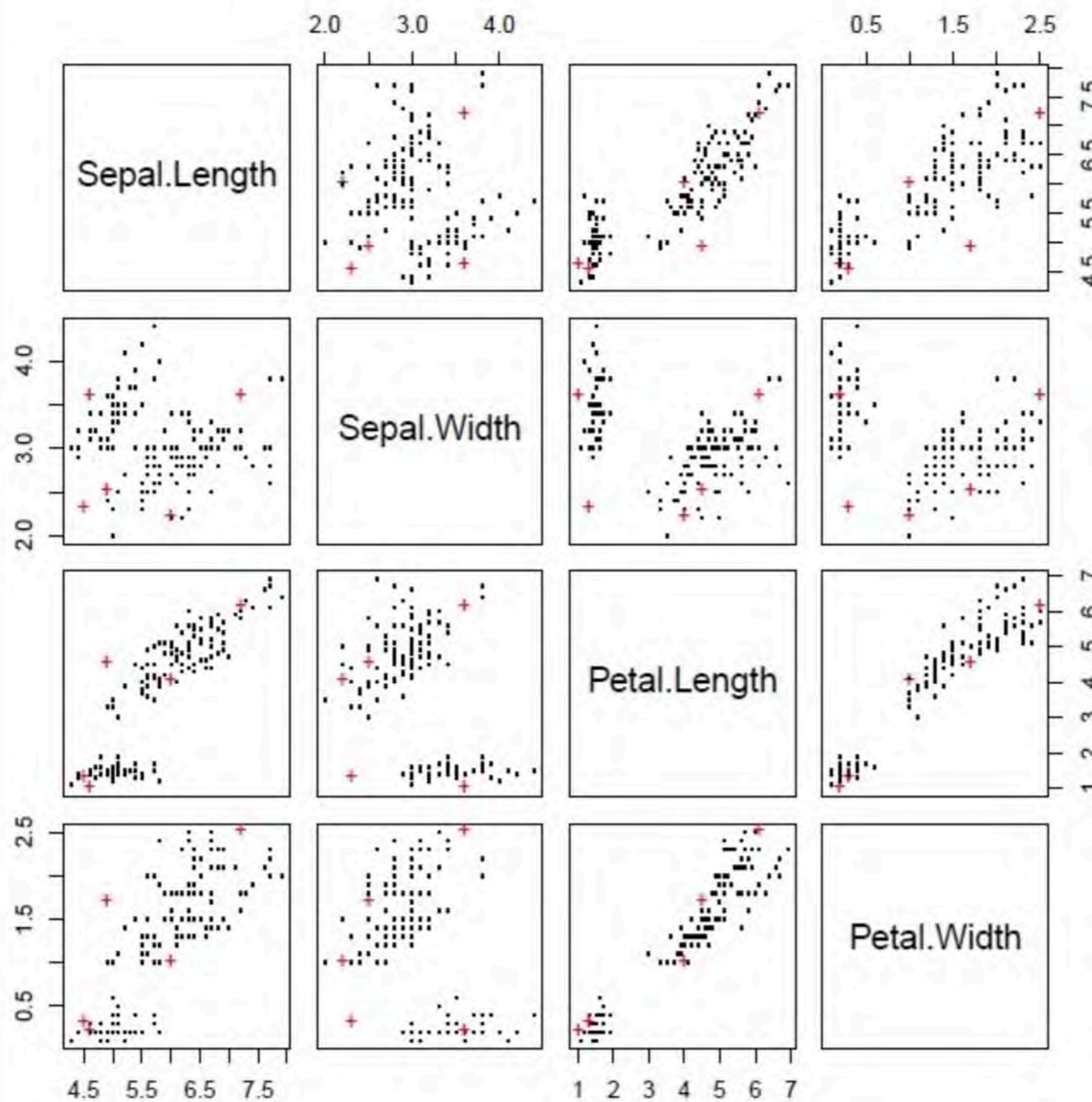
## Visualize Outliers with Plots

Next, we show outliers with a biplot of the first two principal components.

```
> n <- nrow(iris2)
> labels <- 1:n
> labels[-outliers] <- "."
> biplot(prcomp(iris2), cex=.8, xlabs=labels)
```



```
> pch <- rep(".", n)
> pch[outliers] <- "+"
> col <- rep("black", n)
> col[outliers] <- "red"
> pairs(iris2, pch=pch, col=col)
```



## Parallel Computation of LOF Scores

Package Rlof provides function `lof()`, a parallel implementation of the LOF algorithm. Its usage is similar to the above `lofactor()`, but `lof()` has two additional features of supporting multiple values of `k` and several choices of distance metrics. Below is an example of `lof()`.

```
> library(Rlof)
> outlier.scores <- lof(iris2, k=5)
> # try with different number of neighbors (k = 5,6,7,8,9 and 10)
> outlier.scores <- lof(iris2, k=c(5:10))
```

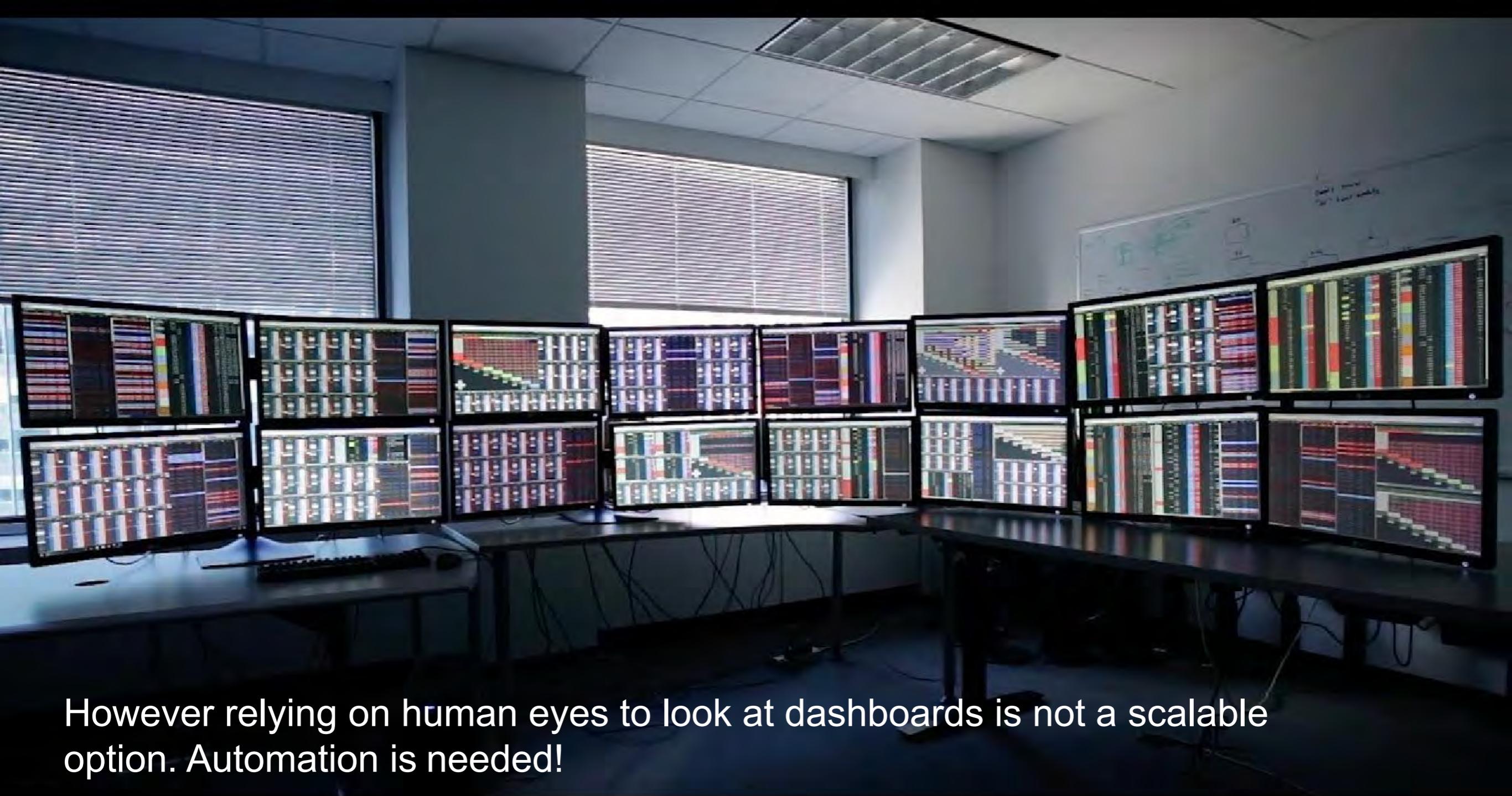
## Workshop 4.18 Practice Anomaly detection Techniques

Try practice anomaly detection techniques from page 564-568



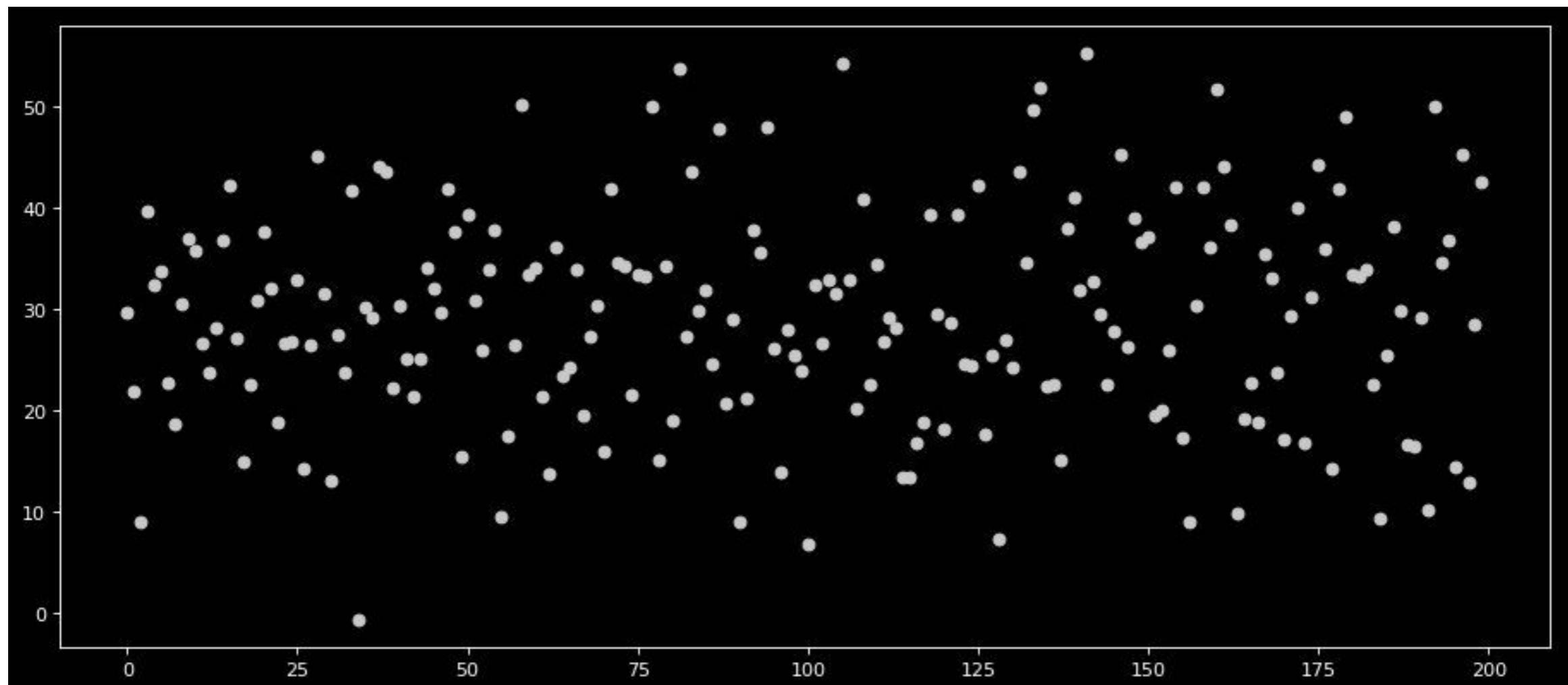
# Anomaly detection in Time Series

# Sensors Everywhere

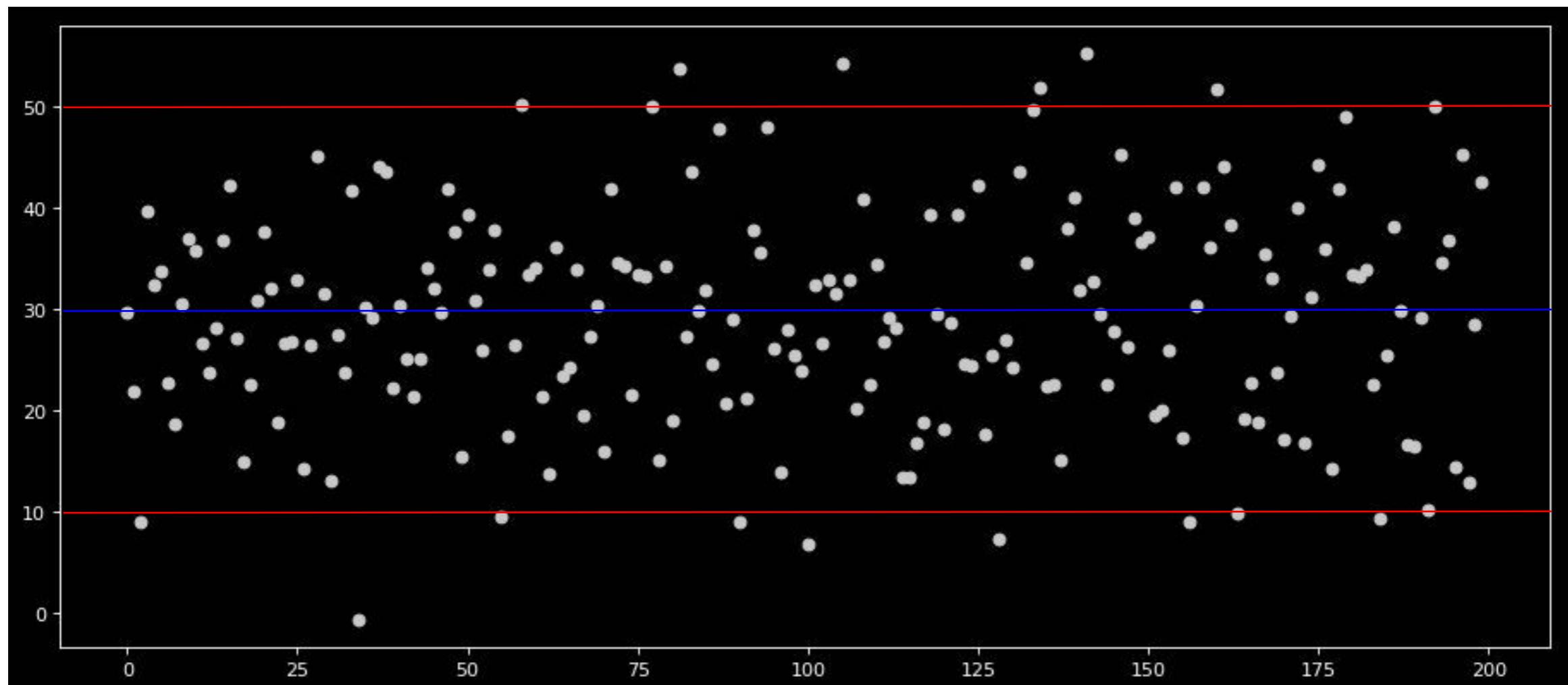


However relying on human eyes to look at dashboards is not a scalable option. Automation is needed!

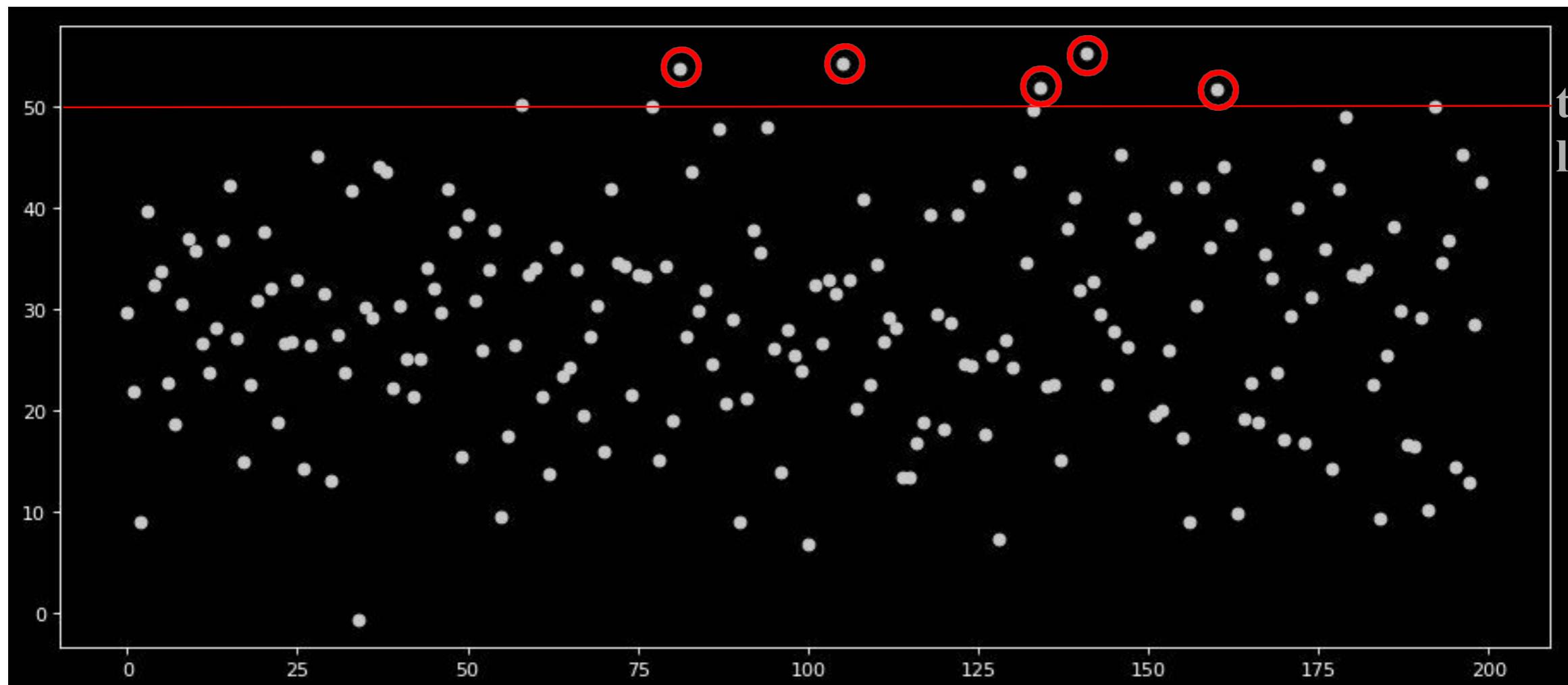
# Case 1



# Case 1

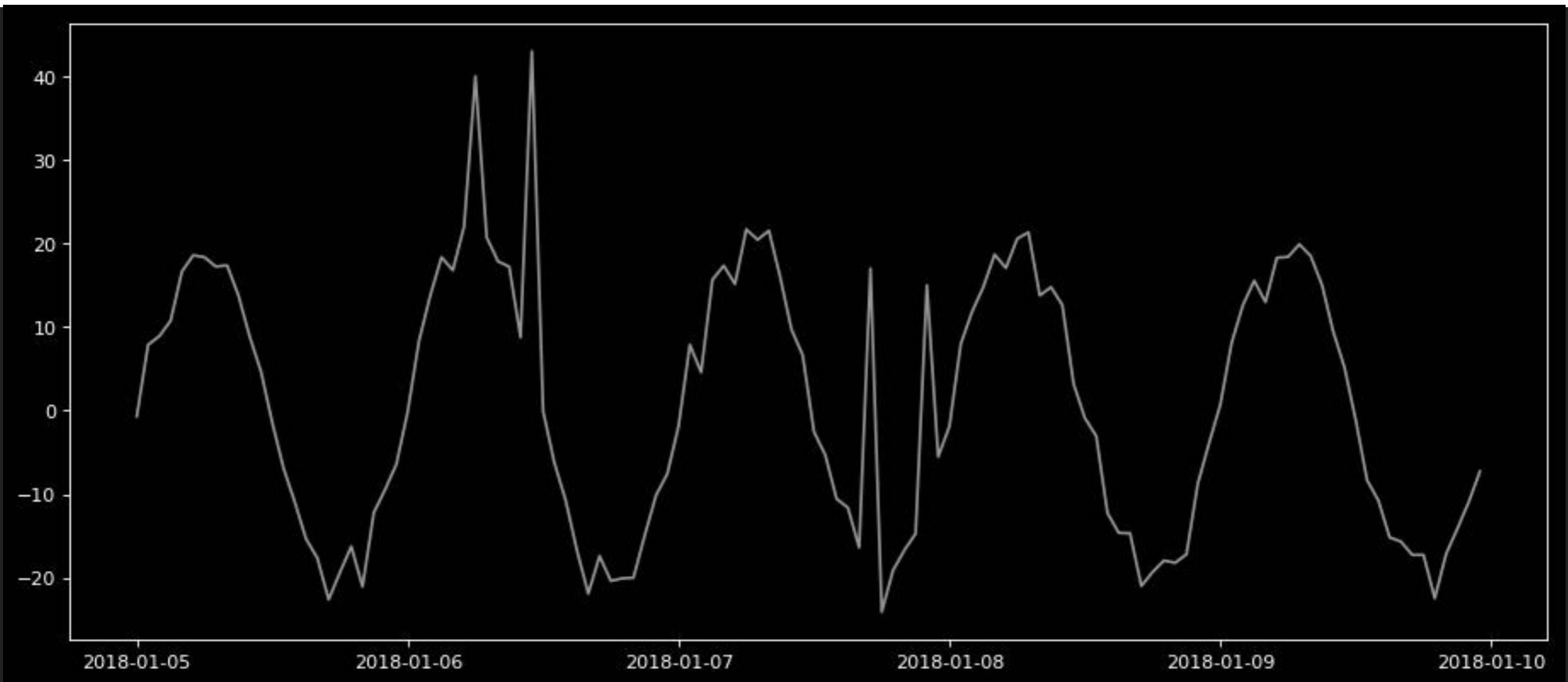


# Case 1



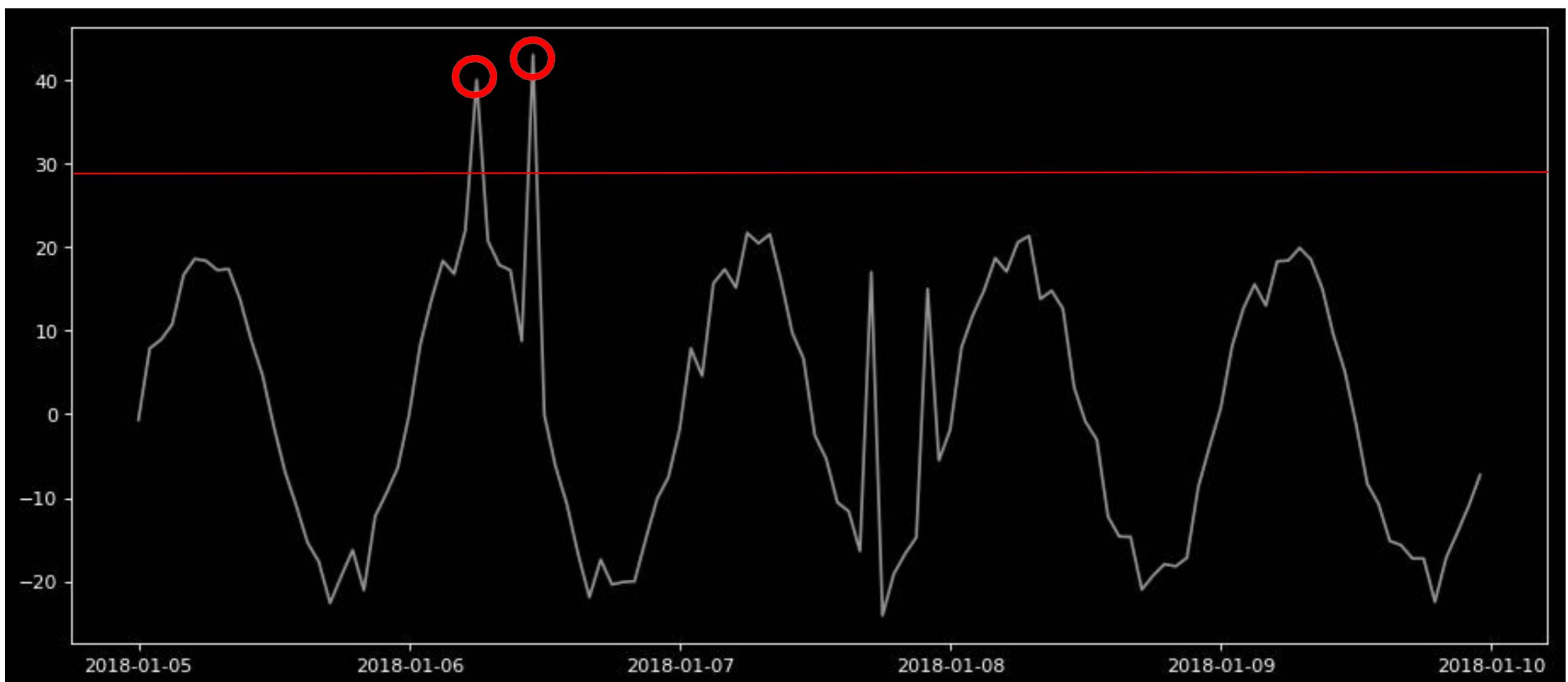
threshold  
ld

# Case 2

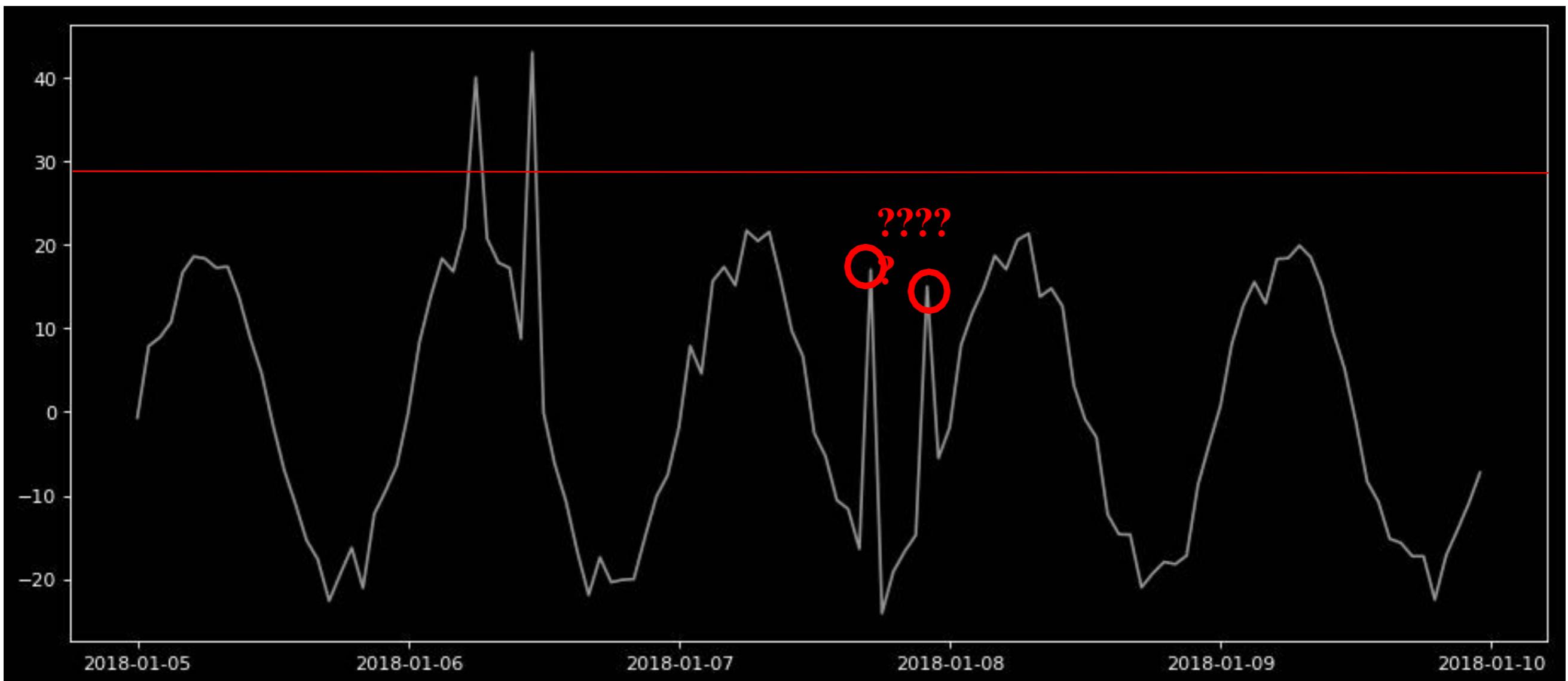


How to isolate global and local anomalies in seasonal signal ?

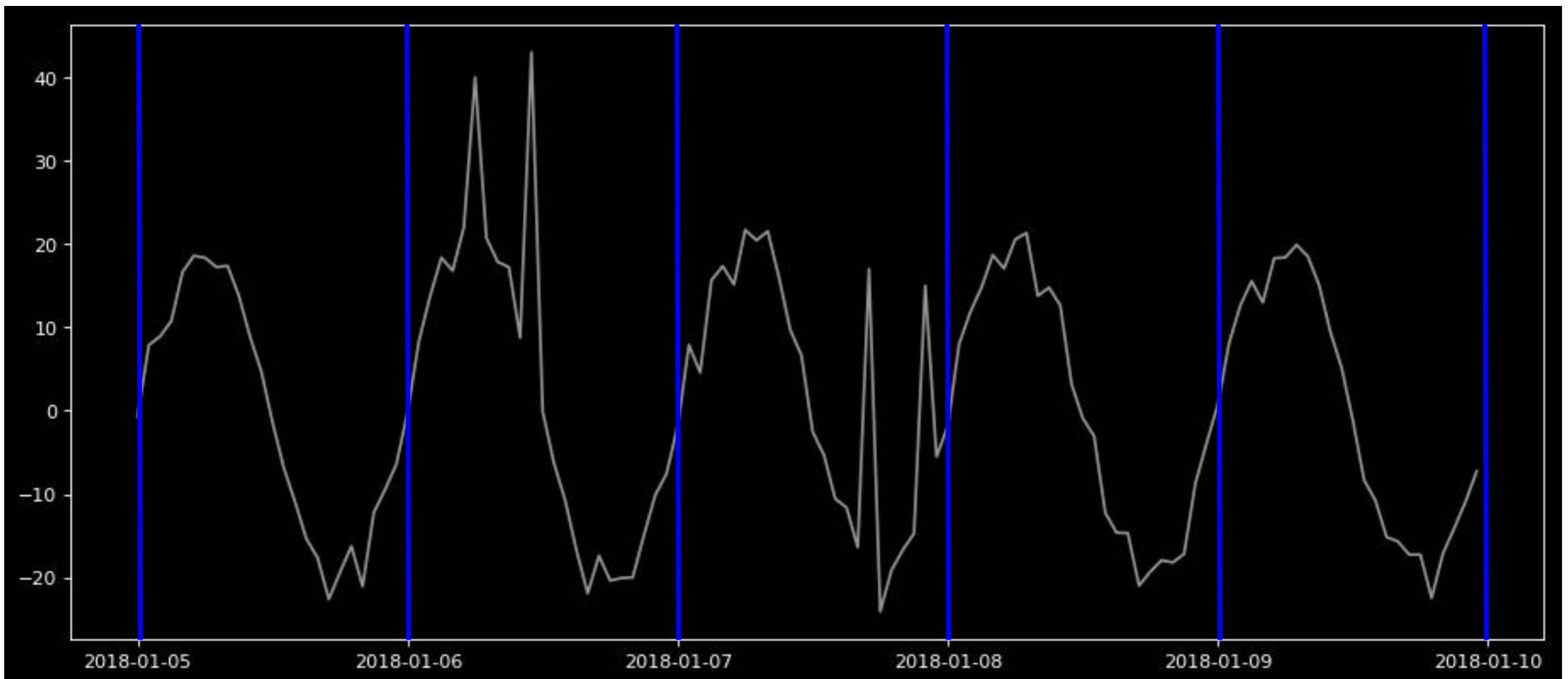
# Case 2



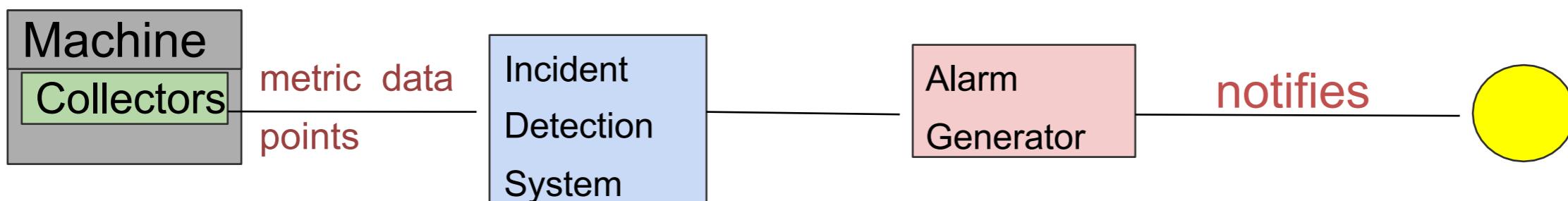
# Case 2



# Seasonality and Frequency



# Automated System



# Type of anomaly in Time Series

- **Additive Outliers**
  - Unexpected growth of users in a short period of time that looks like a **spike**
- **Temporal Changes**
  - Example : Your server goes down and you see zero or a really low number of users for some short period of time.
- **Level Shift or Seasonal Level Shifts**
  - In the case that you deal with some conversion funnel, there could be a drop in a conversion rate. If this happens, the target metric usually doesn't change the shape of a signal, but rather its total value for a period.

# Anomaly Detection Techniques in Time Series

- STL decomposition
- Classification and Regression Trees
- ARIMA

# STL Decomposition

- STL stands for seasonal-trend decomposition procedure based on Loess. This technique gives you an ability to split your time series signal into three parts: seasonal, trend and residue.
  - **Seasonal:** Patterns that repeat with a fixed period of time. For example, a website might receive more visits during weekends; this would produce data with a seasonality of 7 days.
  - **Trend:** The underlying trend of the metrics. A website increasing in popularity should show a general trend that goes up.
  - **Random:** Also call “noise”, “irregular” or “remainder,” this is the residuals of the original time series after the seasonal and trend series are removed.

# STL Decomposition



# STL decomposition

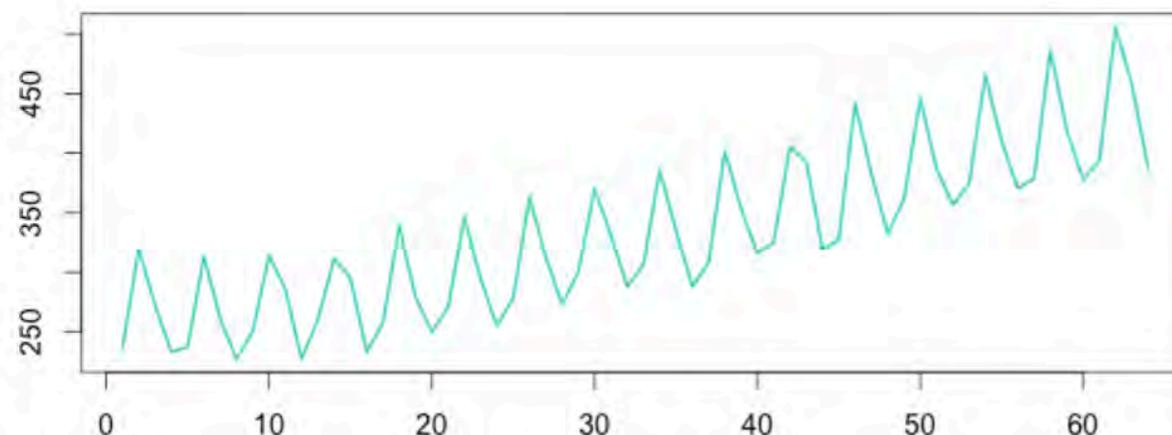
- Additive Decomposition

Additive:  
Time series = Seasonal + Trend + Random

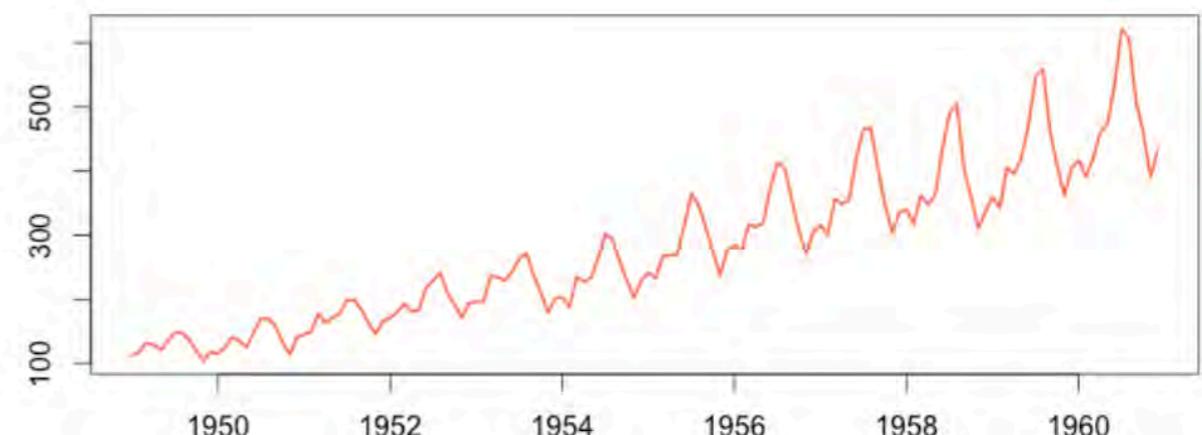
- Multiplicative Decomposition

Multiplicative:  
Time series = Trend \* Seasonal \*Random

# Additive vs Multiplicative

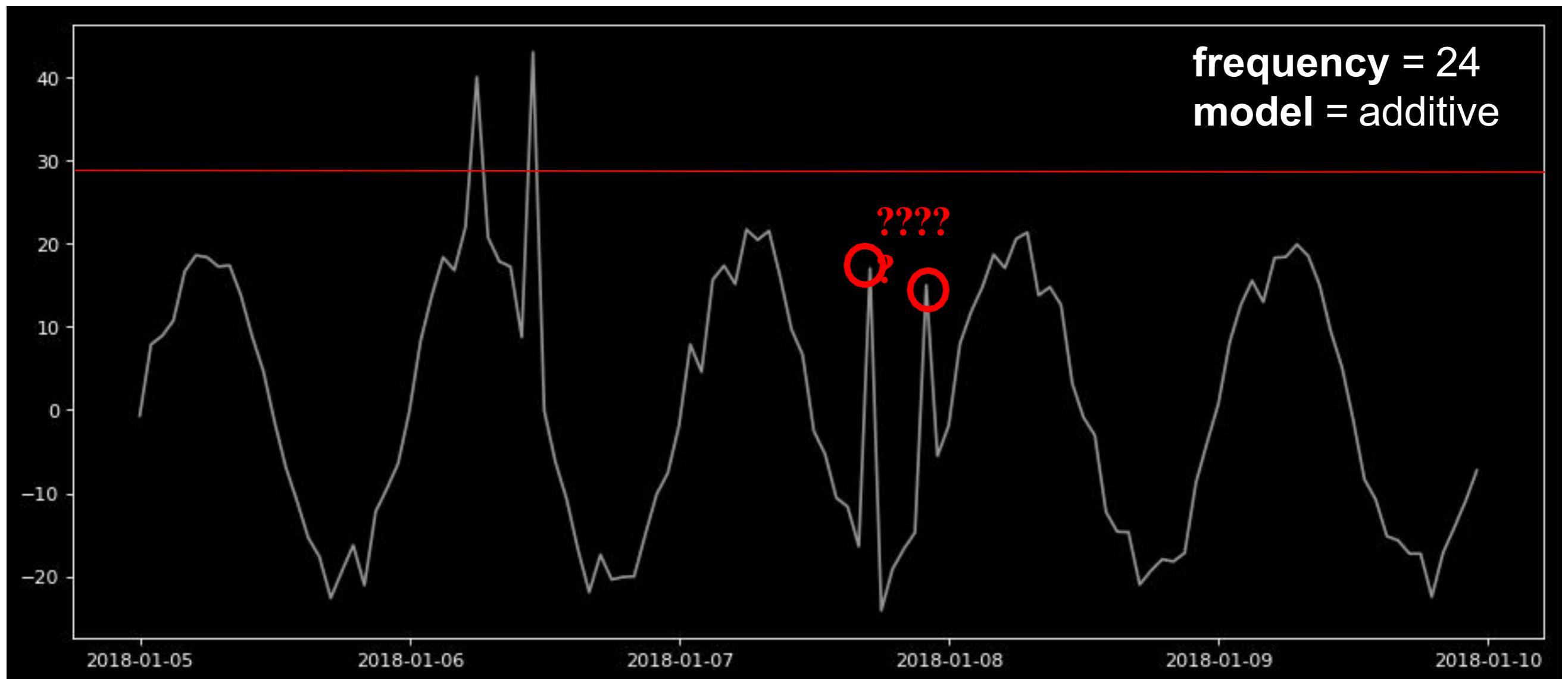


Australian beer production – The seasonal variation looks constant; it doesn't change when the time series value increases. We should use the **additive model**.

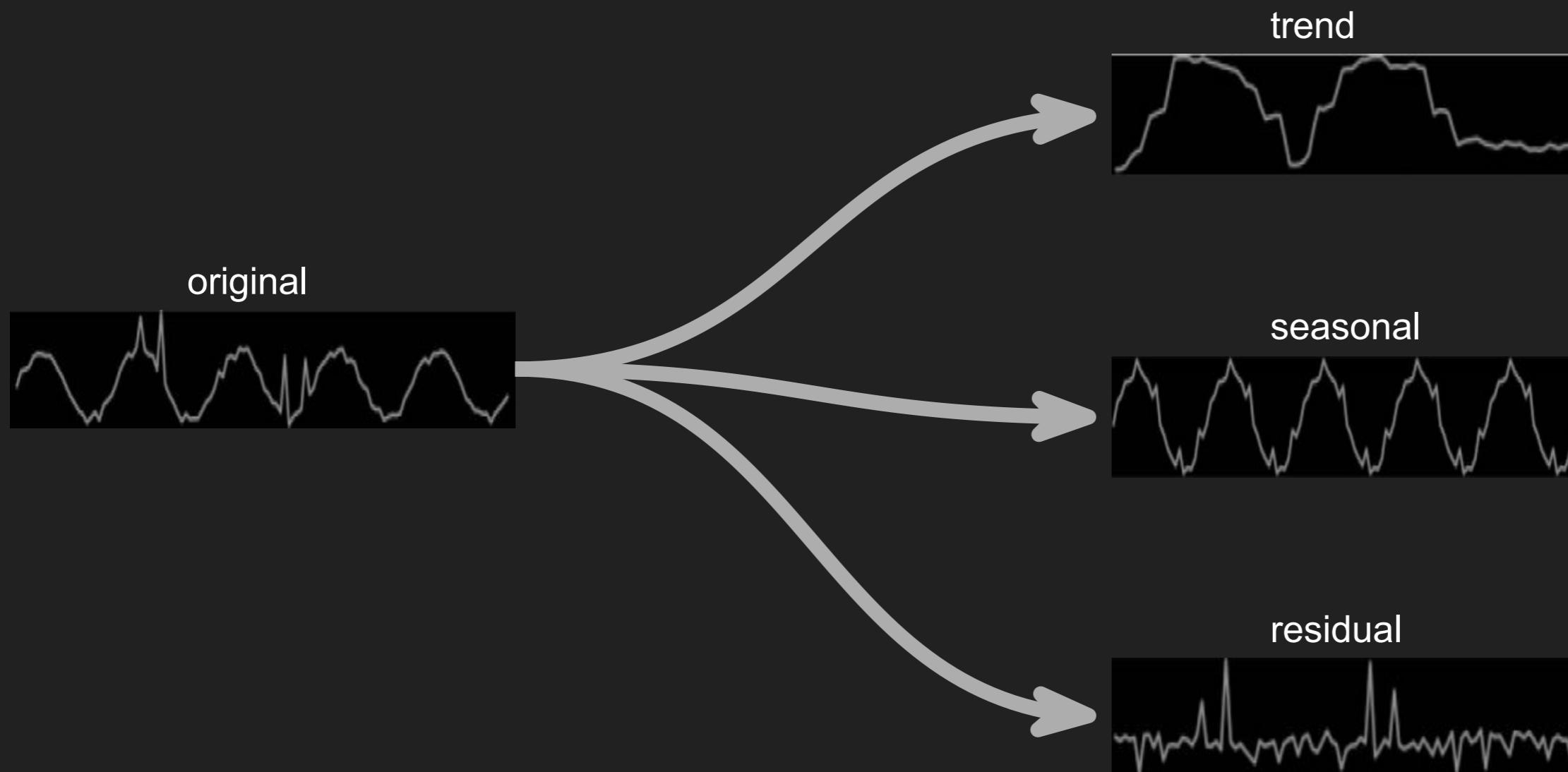


Airline Passenger Numbers – As the time series increases in magnitude, the seasonal variation increases as well. Here we should use the **multiplicative model**.

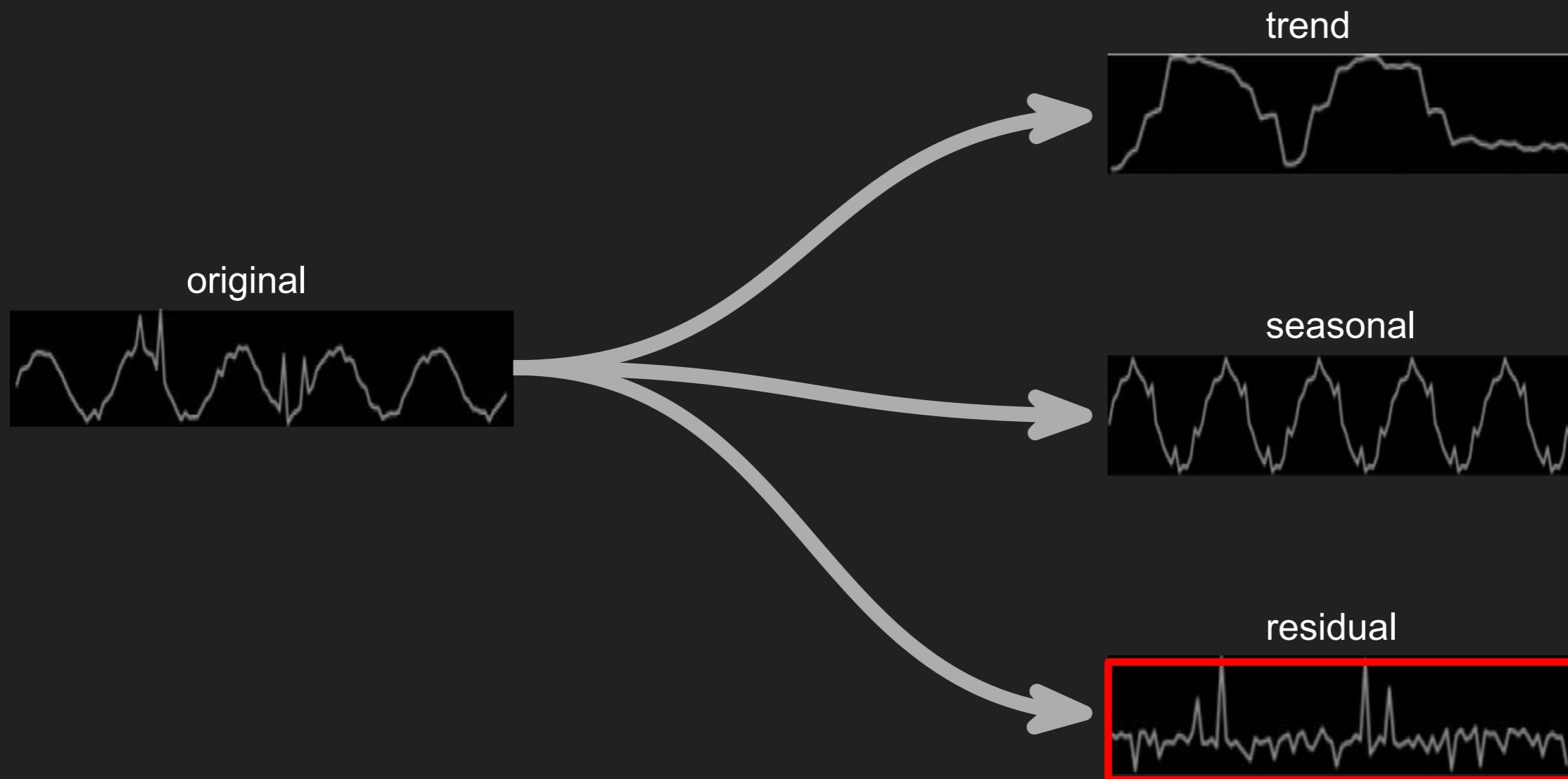
# Case 2



# Seasonal-Trend Decomposition

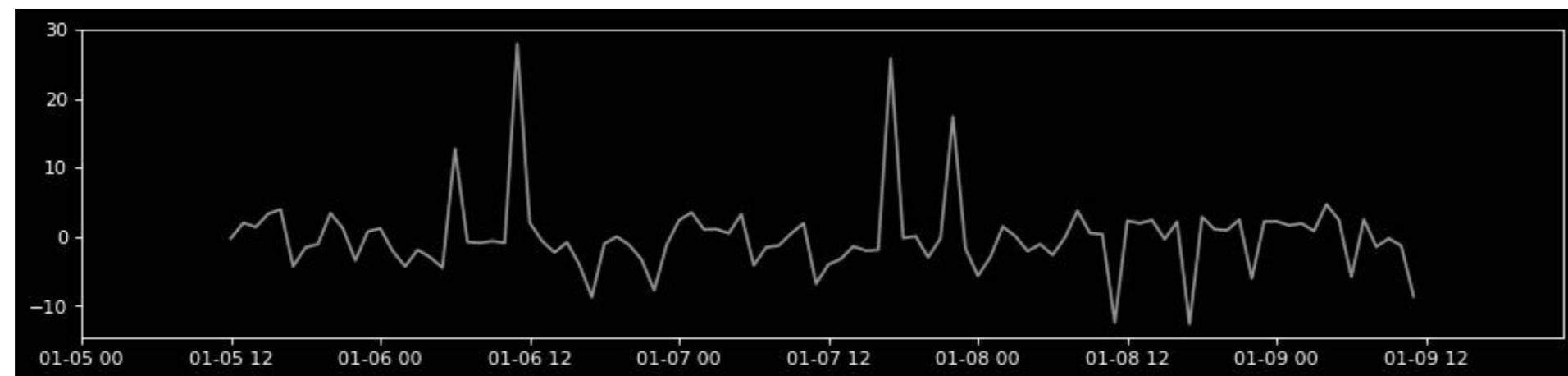
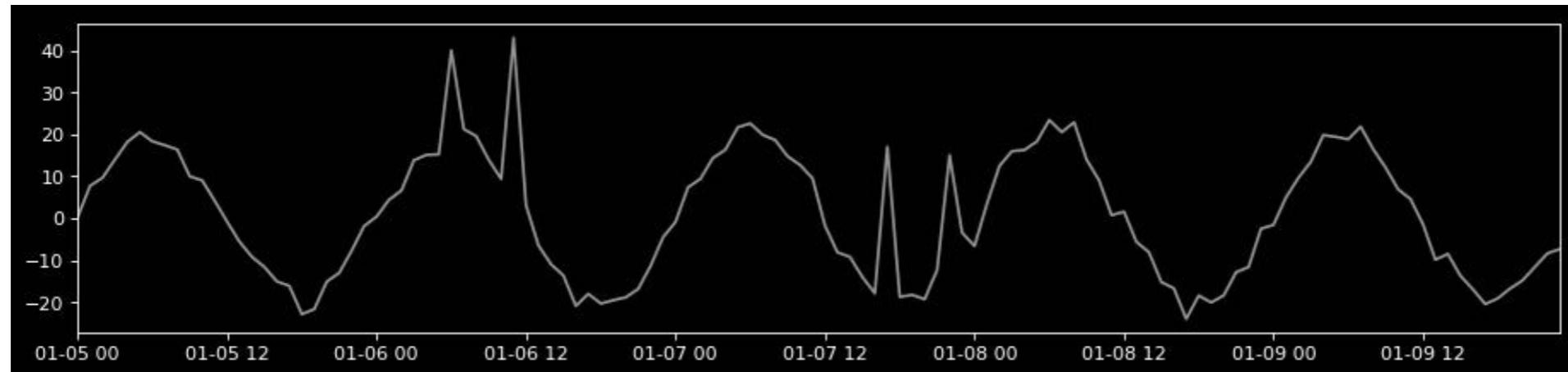


# Seasonal-Trend Decomposition

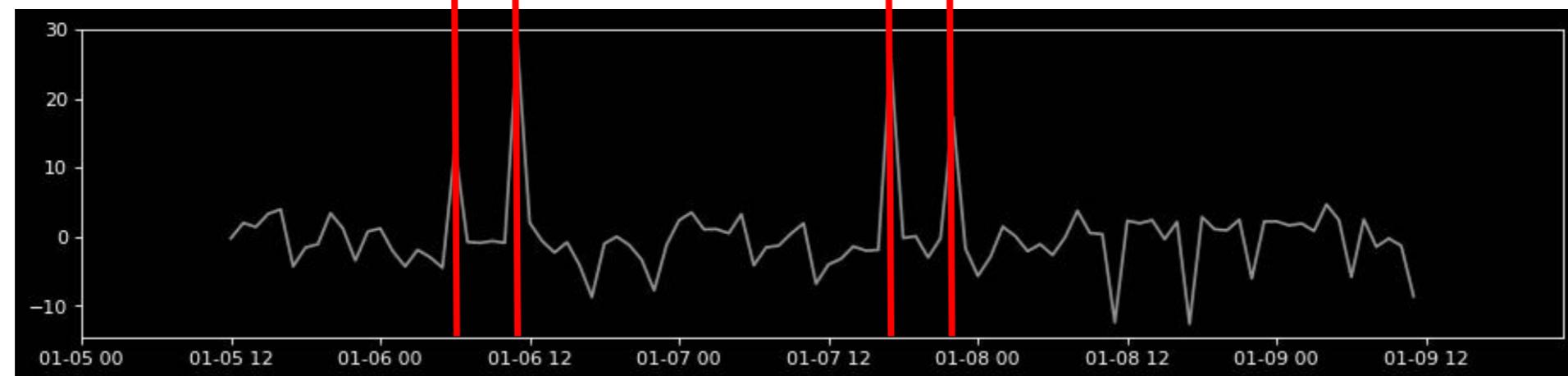
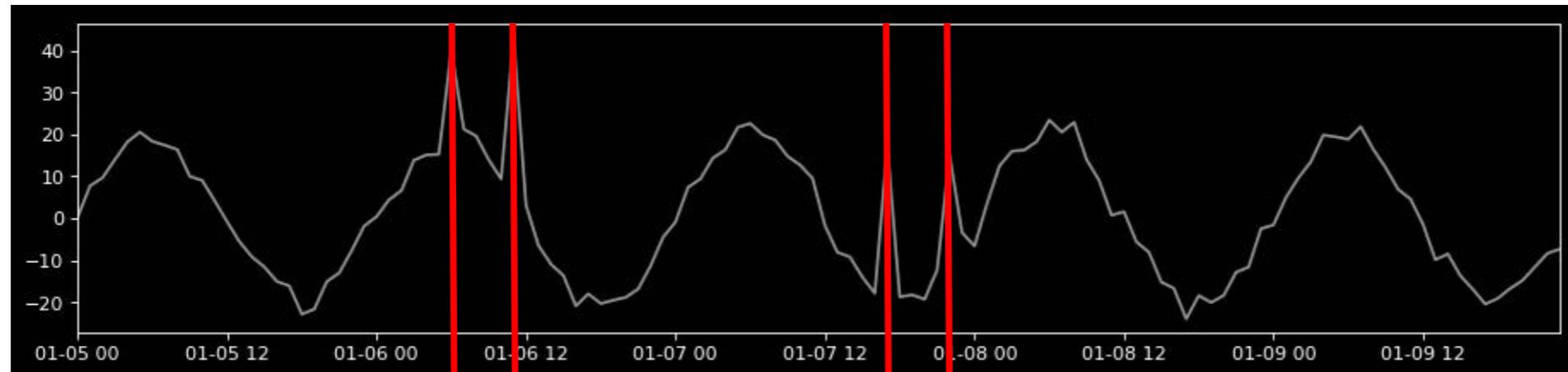


Residual is the component of interest for anomaly detection

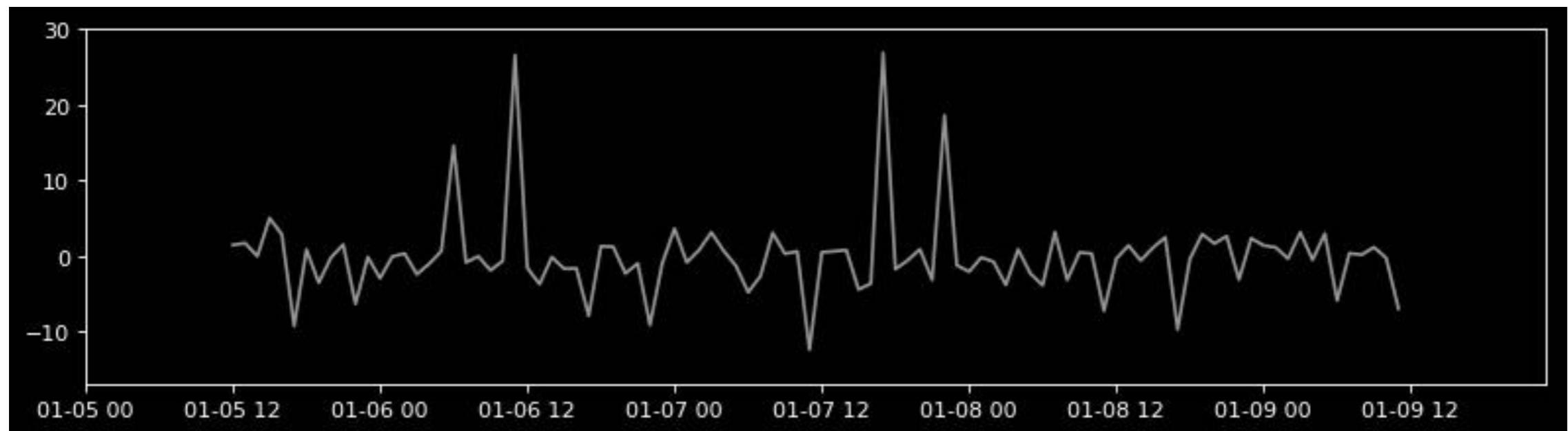
# Seasonal-Trend Decomposition



# Seasonal-Trend Decomposition

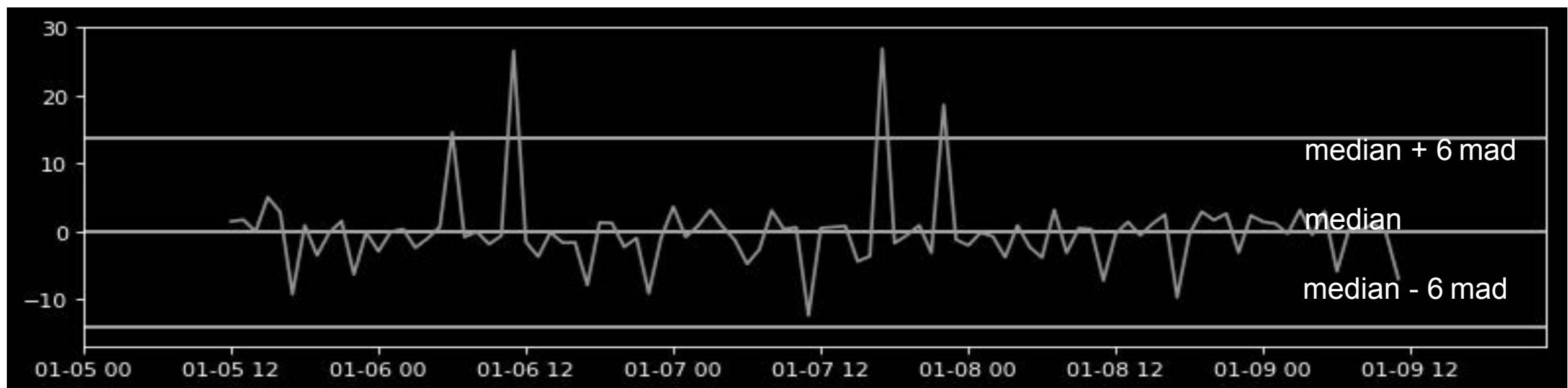


# Seasonal-Trend Decomposition

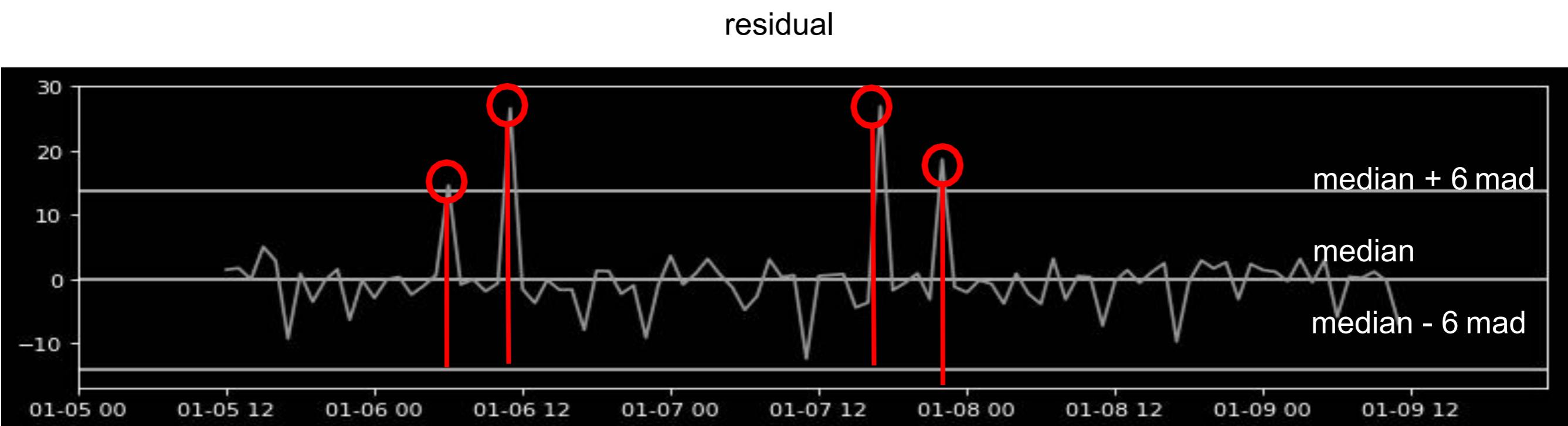


# Seasonal-Trend Decomposition

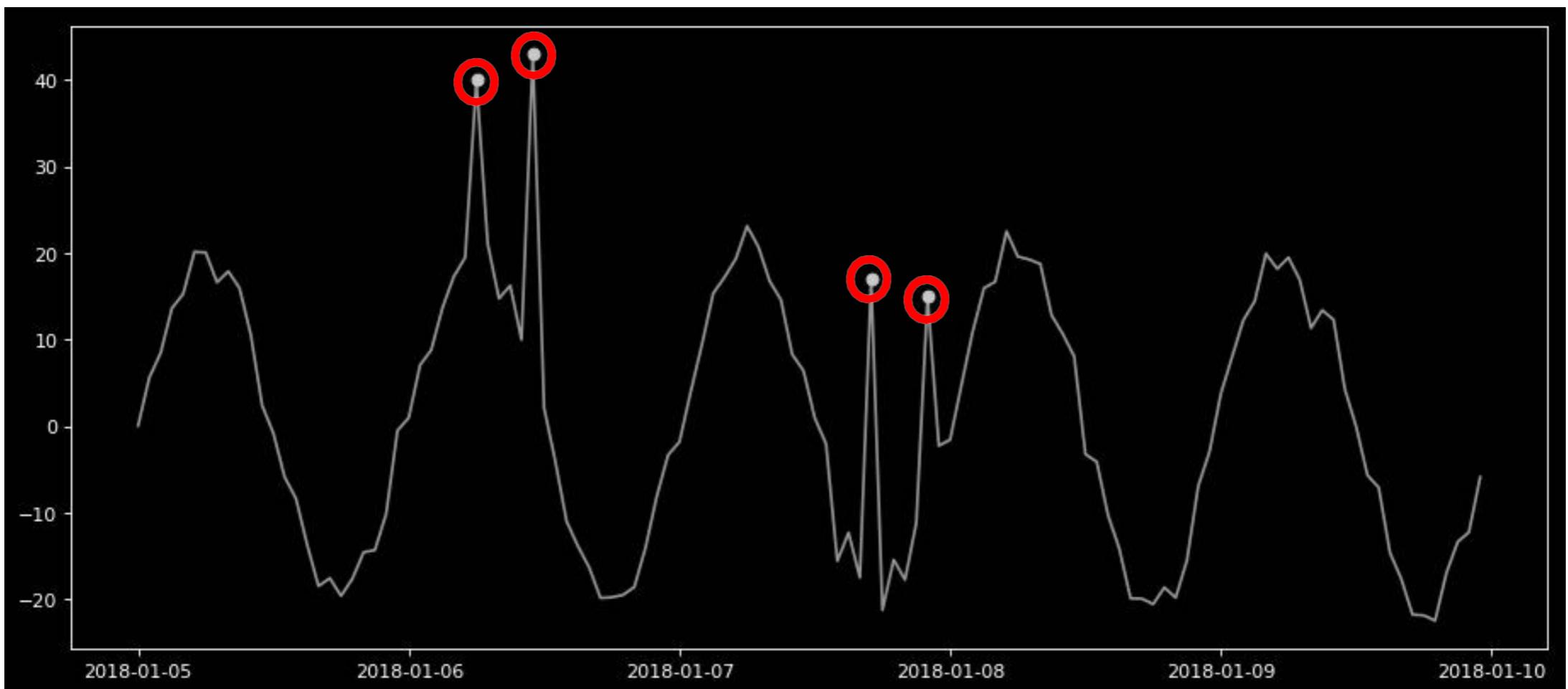
residual



# Seasonal-Trend Decomposition



# Residual Extraction



# Residual Extraction

## Pros:

- Works well with seasonal time series - global and local anomalies
- Few parameters to optimize (compared to other models)
- Algorithm implementation is simple given statistics libraries as available

## Cons:

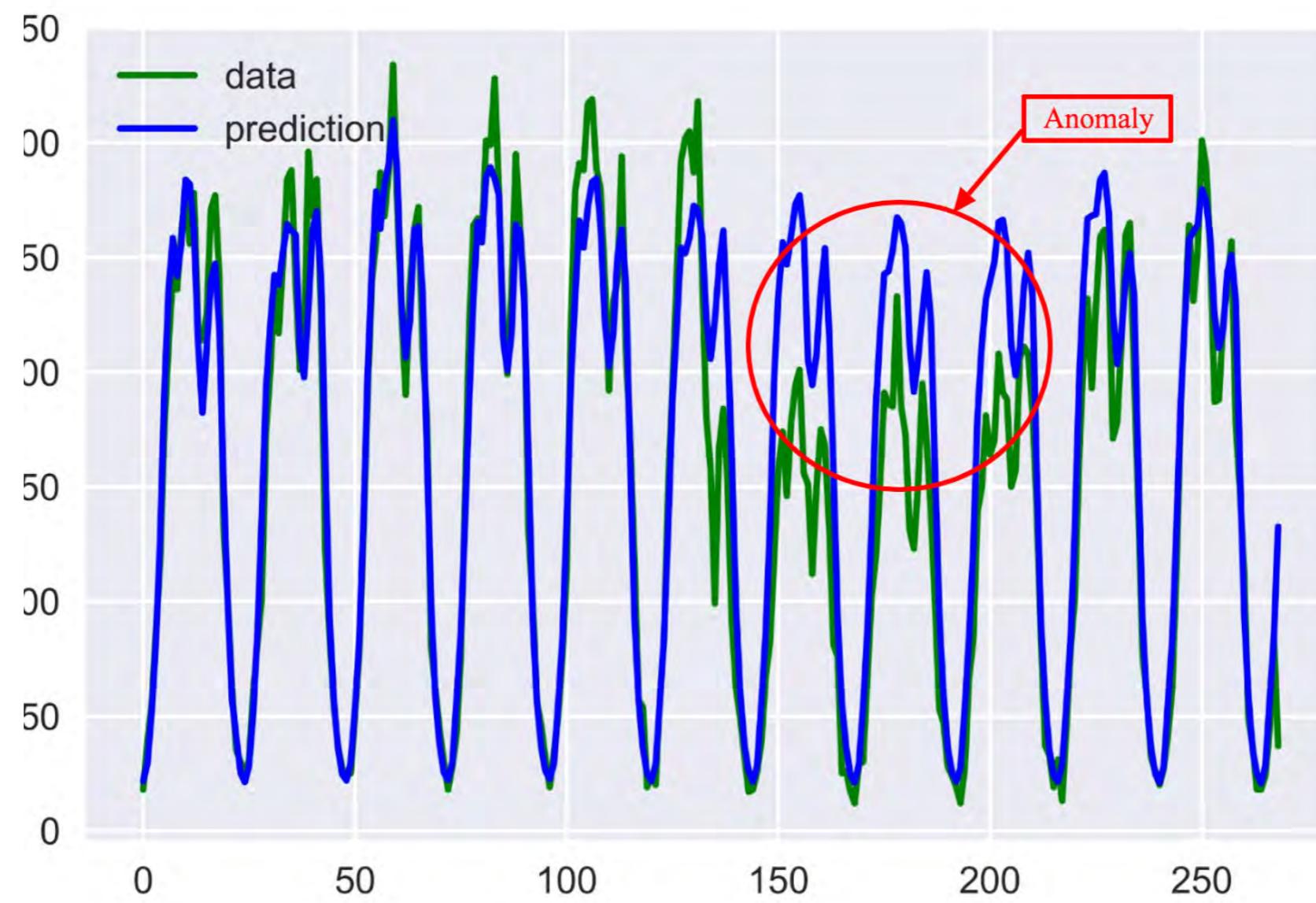
- Need to know how to adjust period parameter for each time series
- Need to know how to adjust anomaly factor so to avoid noisy results
- Works only for seasonal time series where residual is a normal distribution

# Classification and Regression Trees

- **Classification and regression trees** is one of the most robust and most effective machine learning techniques. It may also be applied to anomaly detection problems in several ways.
  - First, you can use supervised learning to teach trees to classify anomaly and non-anomaly data points. In order to do that you'd need to have labeled anomaly data points.
  - The second approach is to use unsupervised learning to teach **CART** to predict the next data point in your series and have some confidence interval or prediction error as in the case of the STL decomposition approach. You can check if your data point lies inside or outside the confidence interval using **Generalized ESD test**, or **Grubbs' test**.

# Classification and Regression Trees

The most popular implementation to perform learning for trees is [the xgboost library](#).





## Pros

- The strength of this approach is that it's not bound in any sense to the structure of your signal, and you can introduce many feature parameters to perform the learning and get sophisticated models.

## Cons

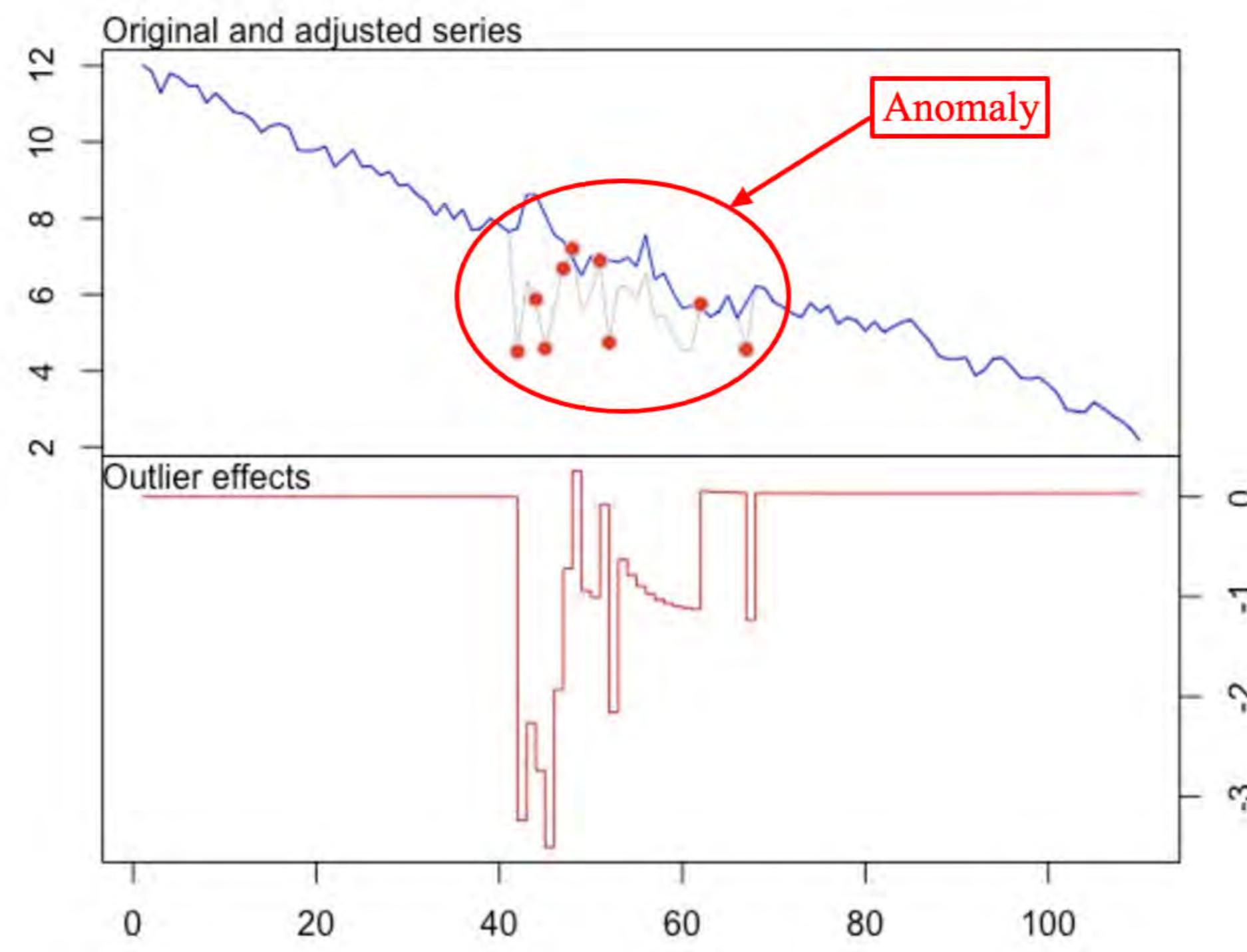
- The weakness is a growing number of features can start to impact your computational performance fairly quickly. In this case, you should select features consciously.

# ARIMA

- ARIMA is a very simple method by design, but still powerful enough to forecast signals and to find anomalies in it.
- It's based on an approach that several points from the past generate a forecast of the next point with the addition of some random variable, which is usually white noise. As you can imagine, forecasted points in the future will generate new points and so on. Its obvious effect on the forecast horizon: the signal gets smoother.
- The difficult part in appliance of this method is that you should select the number of differences, number of autoregressions, and forecast error coefficients.

# ARIMA

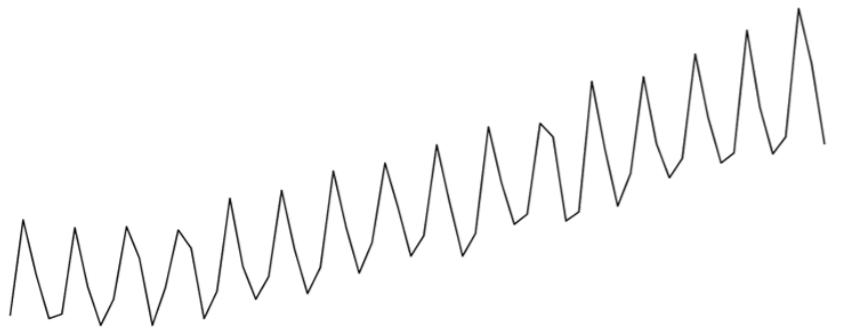
- Another obstacle is that your signal should be stationary after differencing.  
In simple words, it means your signal shouldn't be dependent on time, which is a significant constraint.
- Anomaly detection is done by building an adjusted model of a signal by using outlier points and checking if it's a better fit than the original model by utilizing t-statistics.





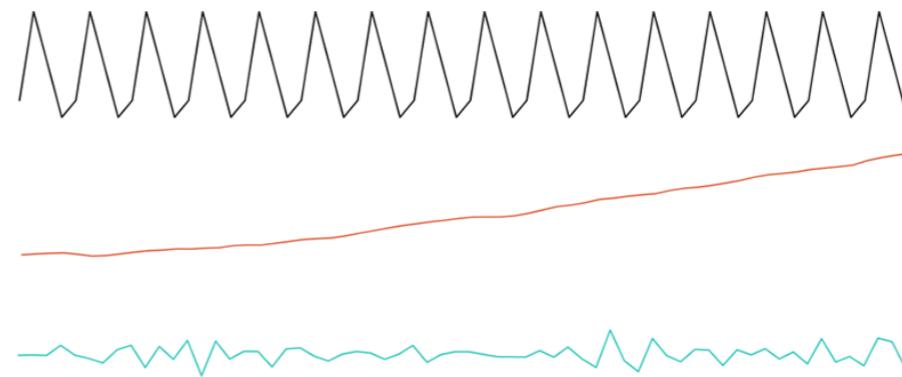
# Anomaly Detection in Time Series with R

# STL Decomposition



=

Seasonal  
+  
Trend  
+  
Random

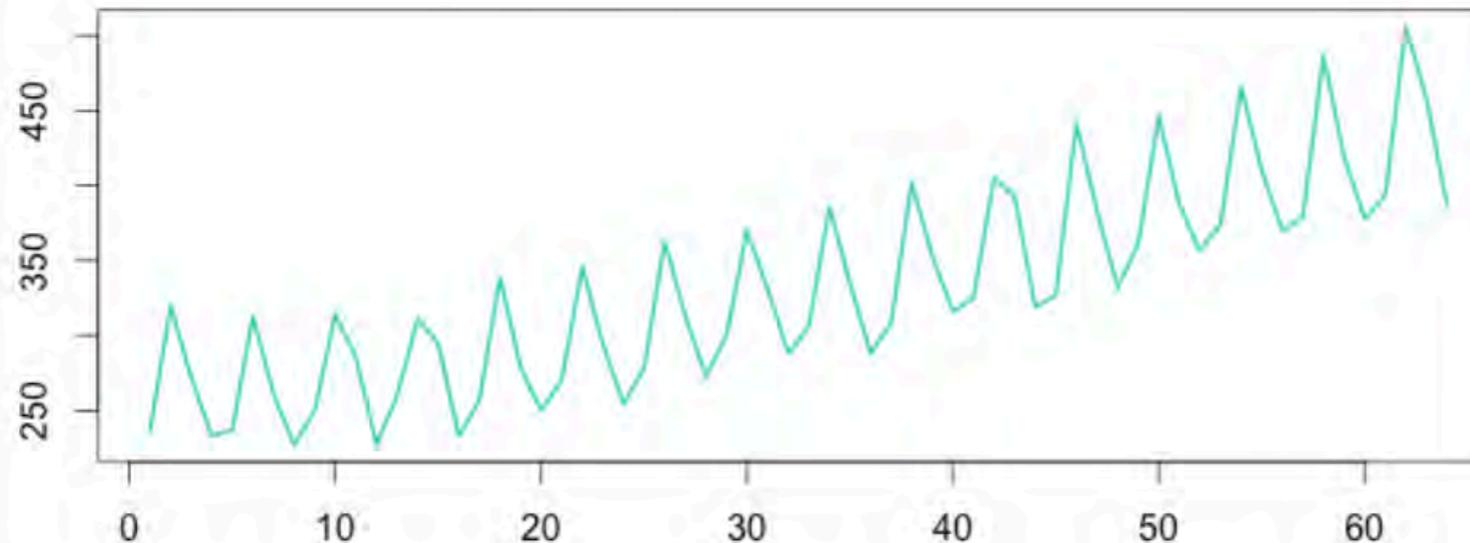


# 1. Import Data

## Additive

As mentioned previously, a good example of additive time series is beer production. As the metric values increase, the seasonality stays relatively constant.

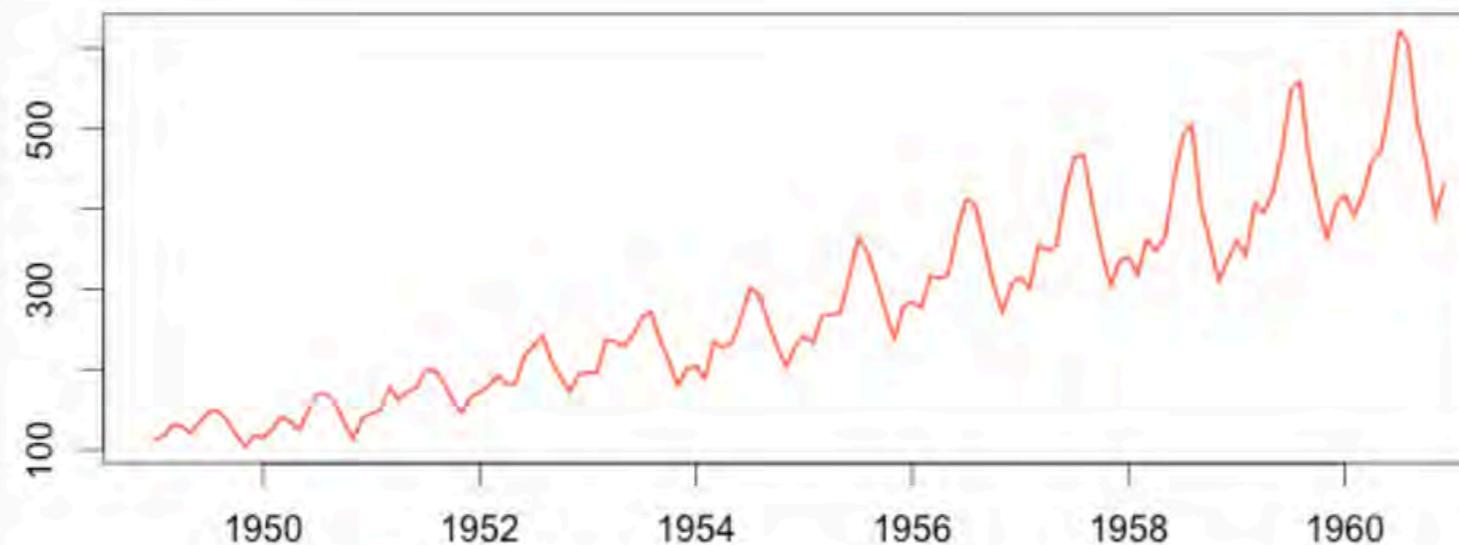
```
1 install.packages("fpp")
2 library(fpp)
3 data(ausbeer)
4 timeserie_beer = tail(head(ausbeer, 17*4+2),17*4-4)
5 plot(as.ts(timeserie_beer))
```



## Multiplicative

Monthly airline passenger figures are a good example of a multiplicative time series. The more passengers there are, the more seasonality is observed.

```
1 install.packages("Ecdat")
2 library(Ecdat)
3 data(AirPassengers)
4 timeserie_air = AirPassengers
5 plot(as.ts(timeserie_air))
```

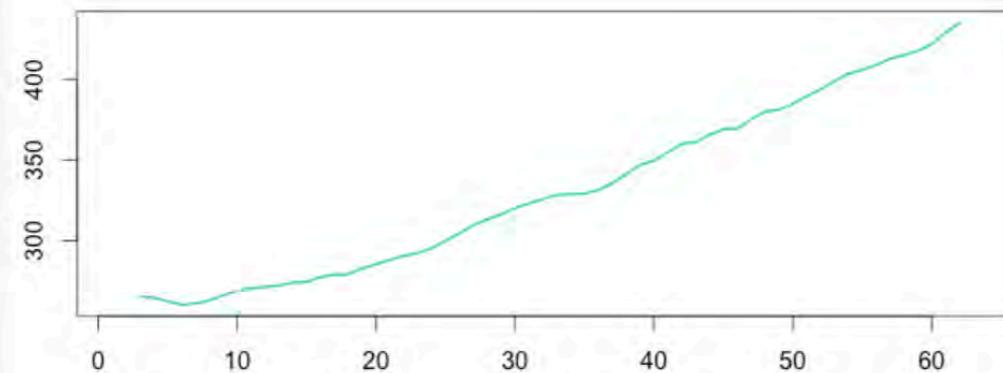
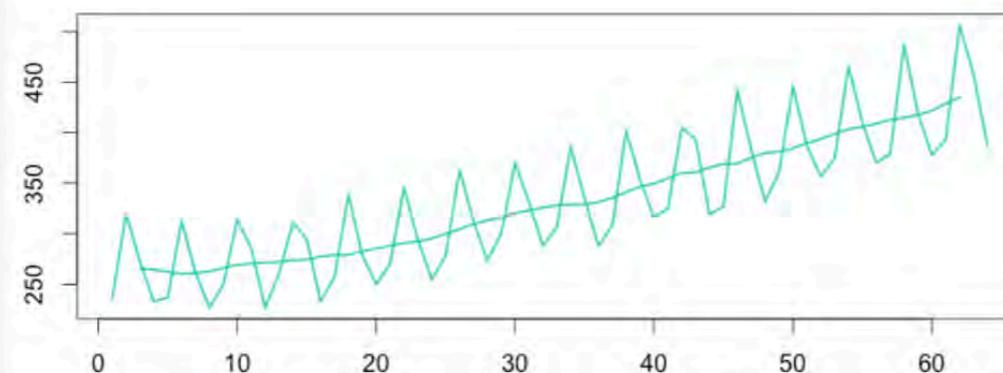


## 2. Detect the trend

### Additive

Australian beer production clearly follows annual seasonality. As it is recorded quarterly, there are 4 data points recorded per year, and we use a moving average window of 4.

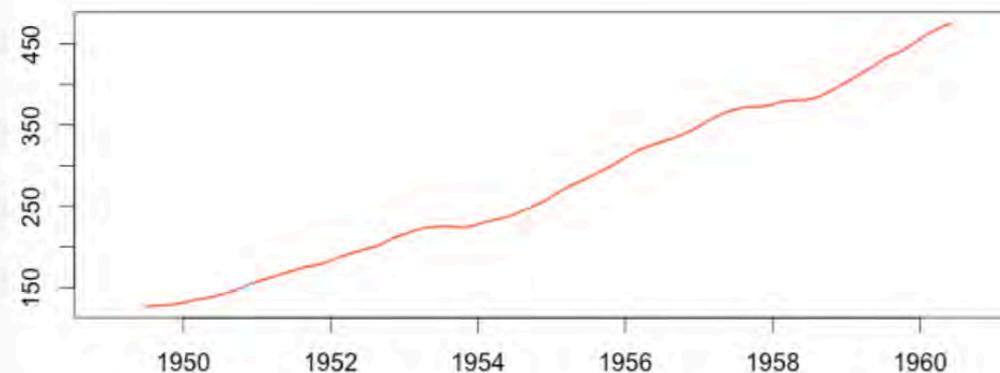
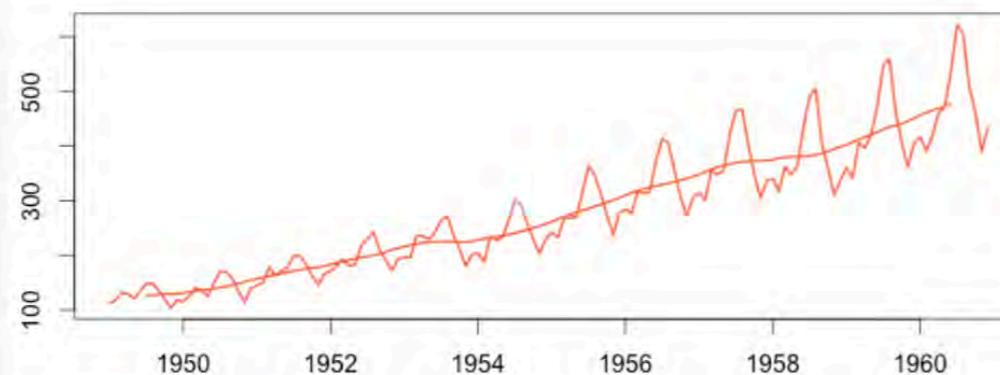
```
1 install.packages("forecast")
2 library(forecast)
3 trend_beer = ma(timeserie_beer, order = 4, centre = T)
4 plot(as.ts(timeserie_beer))
5 lines(trend_beer)
6 plot(as.ts(trend_beer))
```



## Multiplicative

The process here is the same as for the additive model.  
Airline passenger number seasonality also looks annual.  
However, it is recorded monthly, so we choose a **moving average window of 12**.

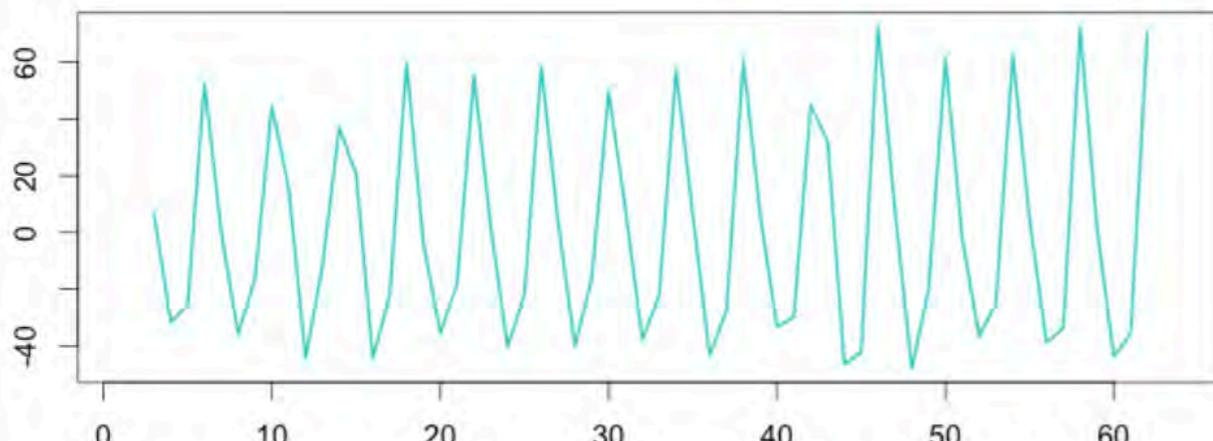
```
1 install.packages("forecast")
2 library(forecast)
3 trend_air = ma(timeserie_air, order = 12, centre = T)
4 plot(as.ts(timeserie_air))
5 lines(trend_air)
6 plot(as.ts(trend_air))
```



# 3. Detrend the Time Series

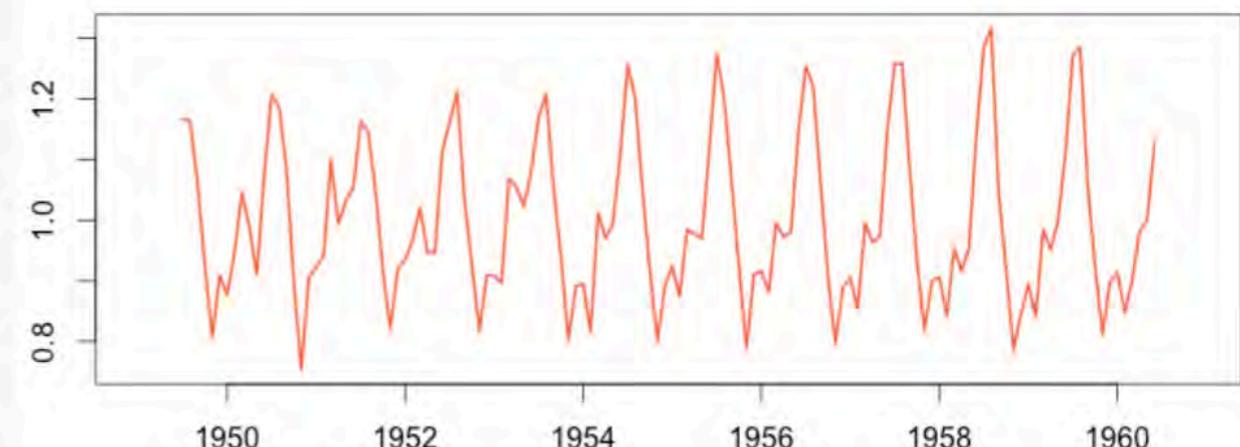
Additive

```
1 detrend_beer = timeserie_beer - trend_beer
2 plot(as.ts(detrend_beer))
```



Multiplicative

```
1 detrend_air = timeserie_air / trend_air
2 plot(as.ts(detrend_air))
```

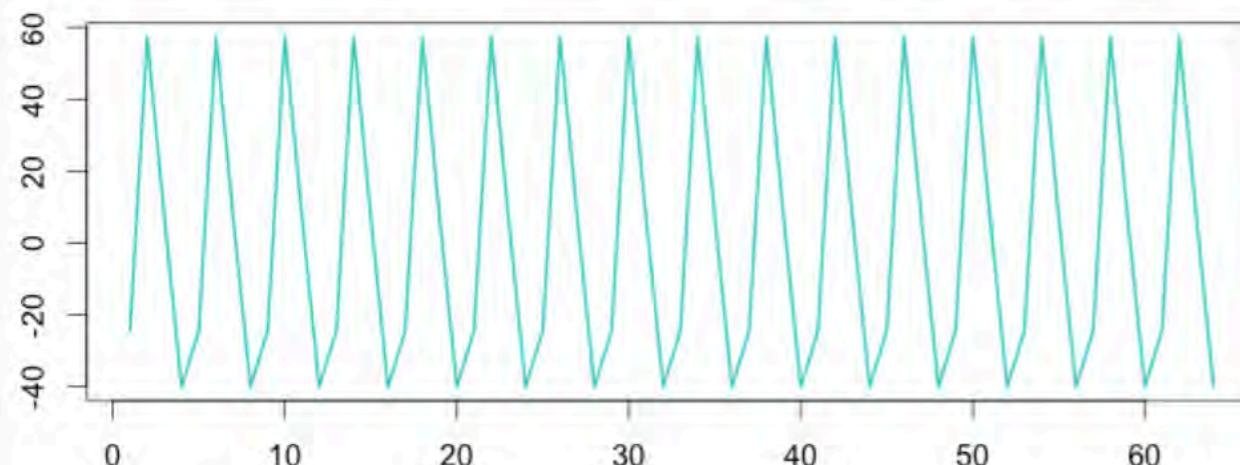


# Average Seasonality

## Additive

Quarterly seasonality: we use a matrix of 4 rows. *The average seasonality is repeated 16 times to create the graphic to be compared later (see below)*

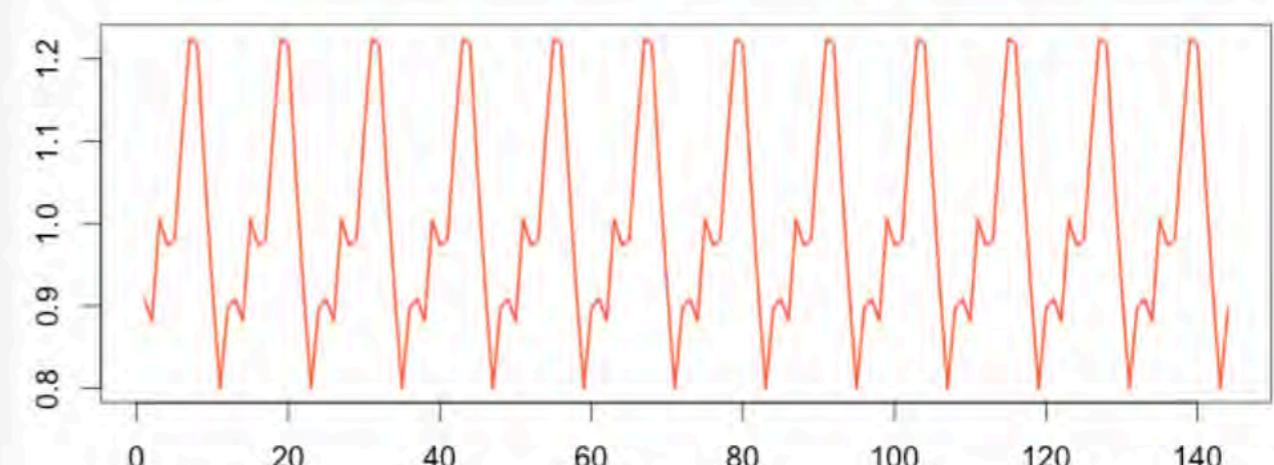
```
1 m_beer = t(matrix(data = detrend_beer, nrow = 4))
2 seasonal_beer = colMeans(m_beer, na.rm = T)
3 plot(as.ts(rep(seasonal_beer, 16)))
```



## Multiplicative

Monthly seasonality: we use a matrix of 12 rows. *The average seasonality is repeated 12 times to create the graphic we will compare later (see below)*

```
1 m_air = t(matrix(data = detrend_air, nrow = 12))
2 seasonal_air = colMeans(m_air, na.rm = T)
3 plot(as.ts(rep(seasonal_air, 12)))
```

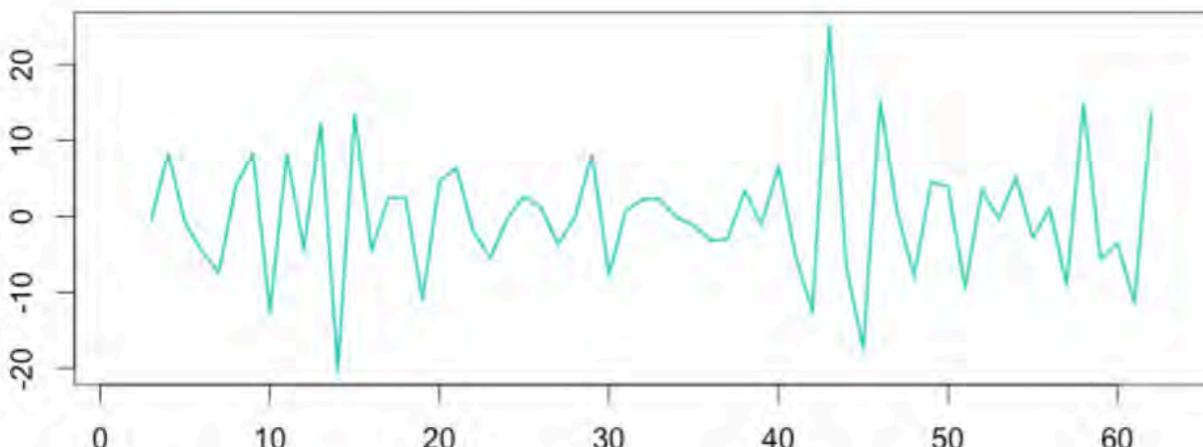


# 5. Examining Remaining Random Noise

## Additive

The additive formula is "Time series = Seasonal + Trend + Random", which means "Random = Time series – Seasonal – Trend"

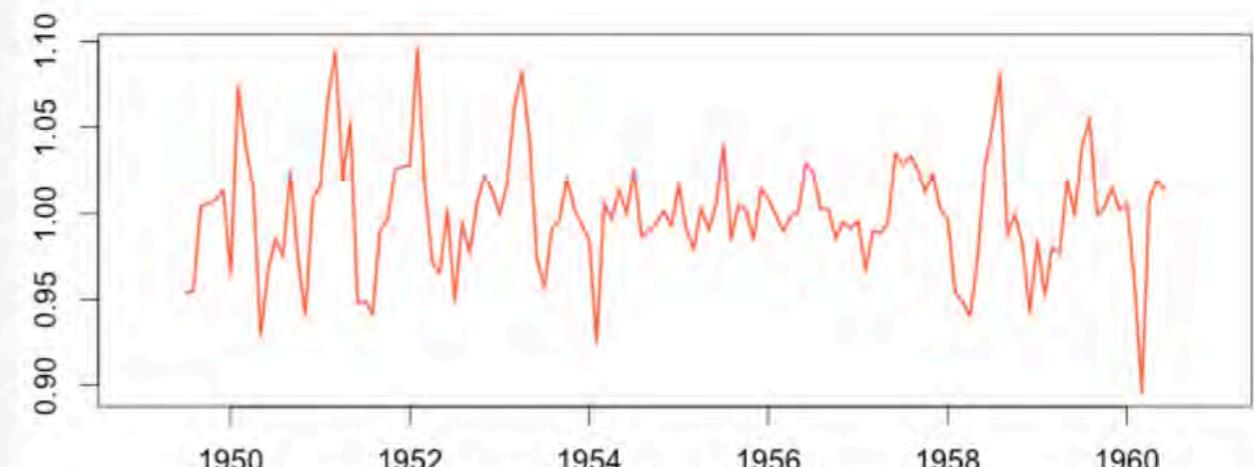
```
1 random_beer = timeserie_beer - trend_beer - seasonal_beer
2 plot(as.ts(random_beer))
```



## Multiplicative

The multiplicative formula is "Time series = Seasonal \* Trend \* Random", which means "Random = Time series / (Trend \* Seasonal)"

```
1 random_air = timeserie_air / (trend_air * seasonal_air)
2 plot(as.ts(random_air))
```

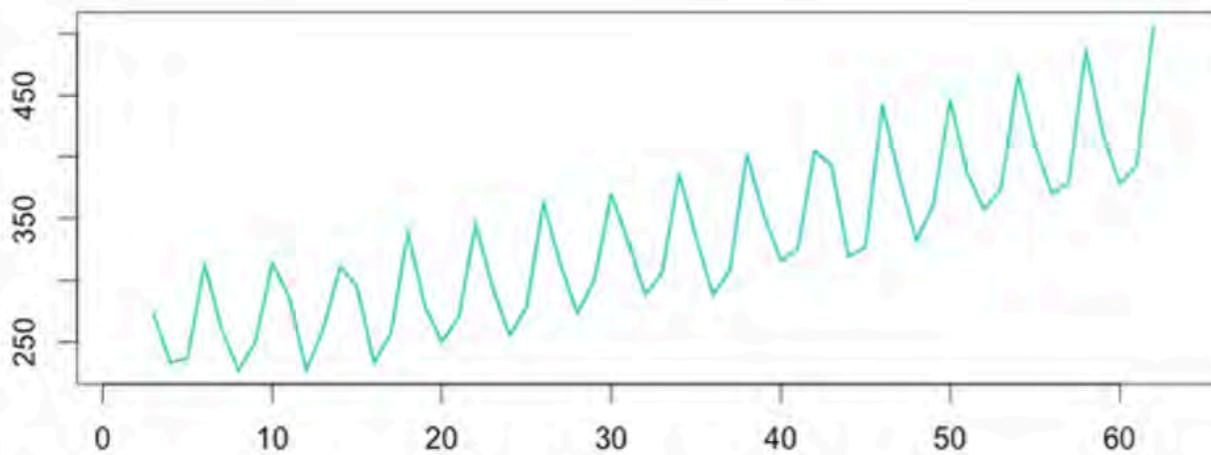


# 6. Reconstruct the Original Signal

## Additive

The additive formula is "Time series = Seasonal + Trend + Random", which means "Random = Time series - Seasonal - Trend"

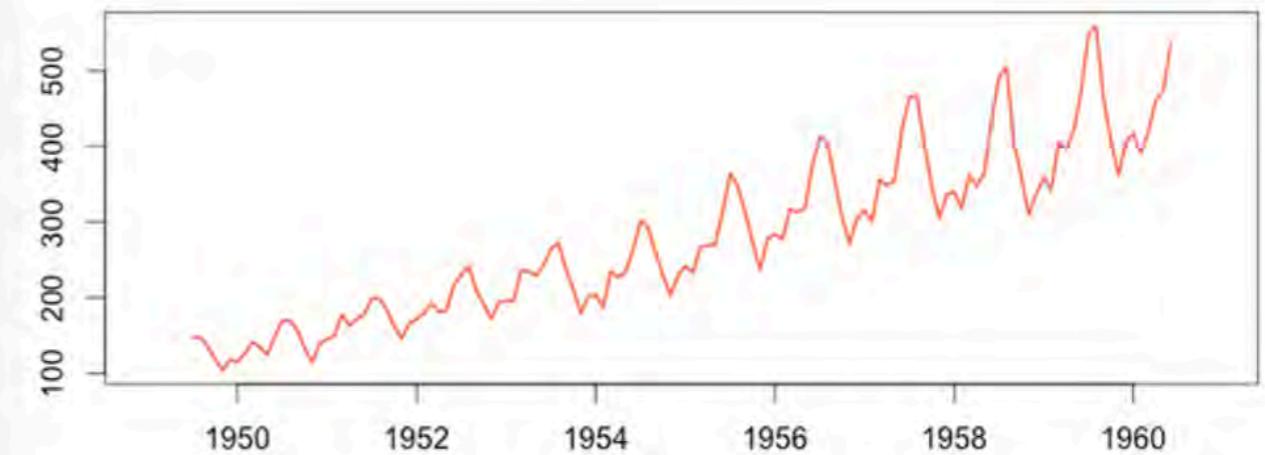
```
1 recomposed_beer = trend_beer+seasonal_beer+random_beer
2 plot(as.ts(recomposed_beer))
```



## Multiplicative

The multiplicative formula is "Time series = Seasonal \* Trend \* Random", which means "Random = Time series / (Trend \* Seasonal)"

```
1 recomposed_air = trend_air*seasonal_air*random_air
2 plot(as.ts(recomposed_air))
```



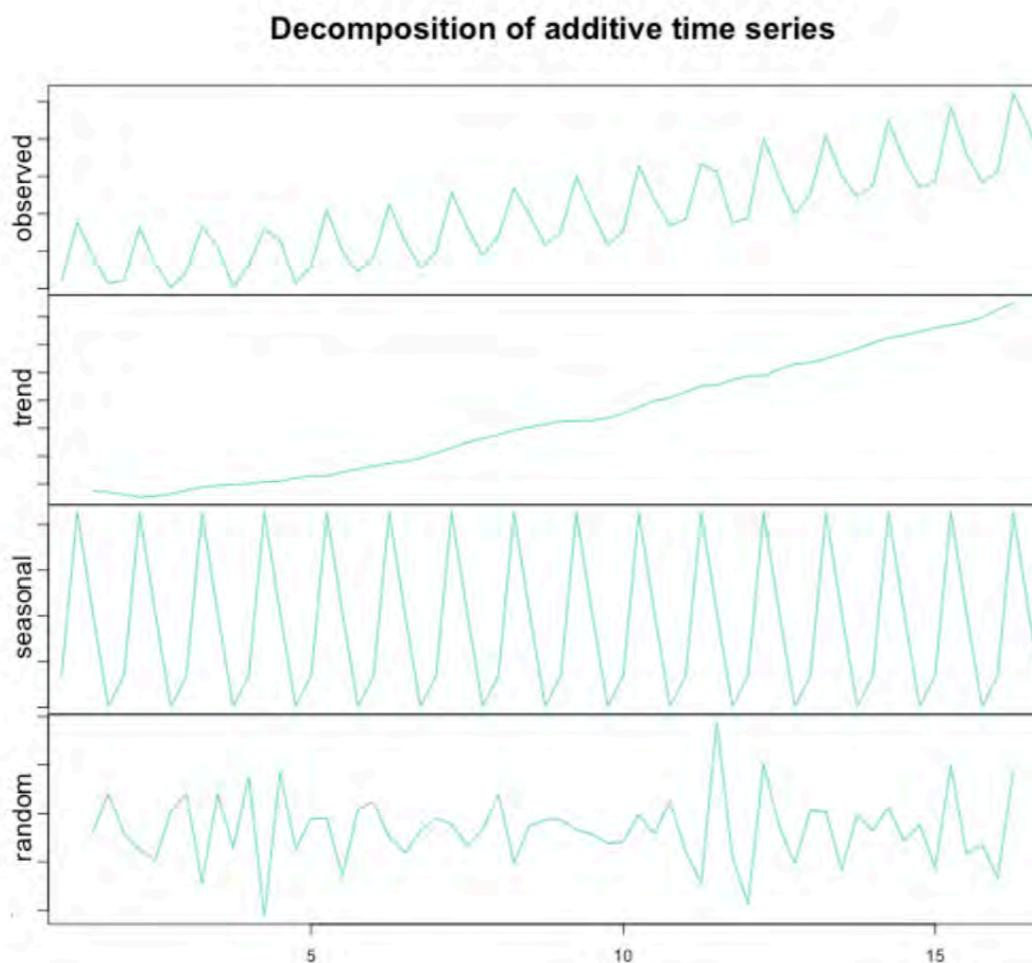
# Decompose function

## Additive

The only requirement: seasonality is quarterly (frequency = 4)

Using the DECOMPOSE( ) function:

```
1 ts_beer = ts(timeserie_beer, frequency = 4)
2 decompose_beer = decompose(ts_beer, "additive")
3
4 plot(as.ts(decompose_beer$seasonal))
5 plot(as.ts(decompose_beer$trend))
6 plot(as.ts(decompose_beer$random))
7 plot(decompose_beer)
```

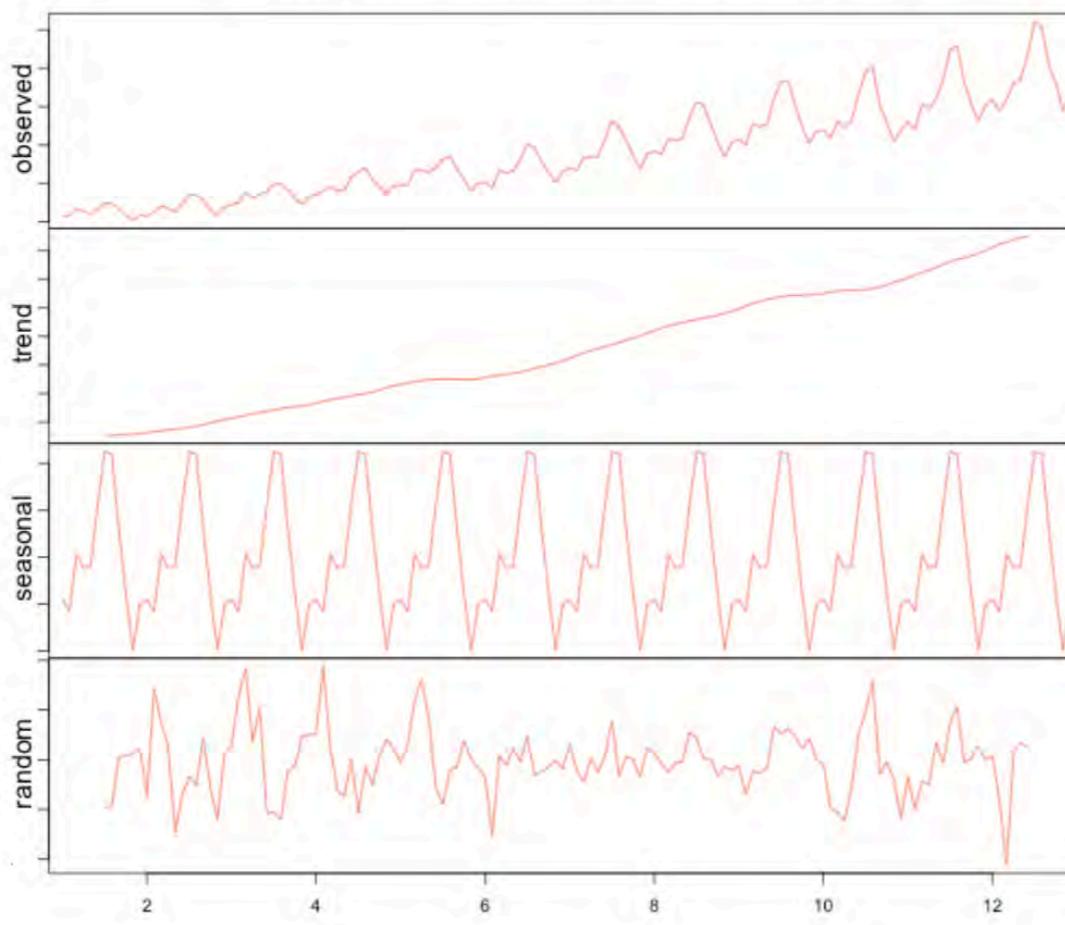


## Multiplicative

The only requirement: seasonality is monthly (frequency = 12)

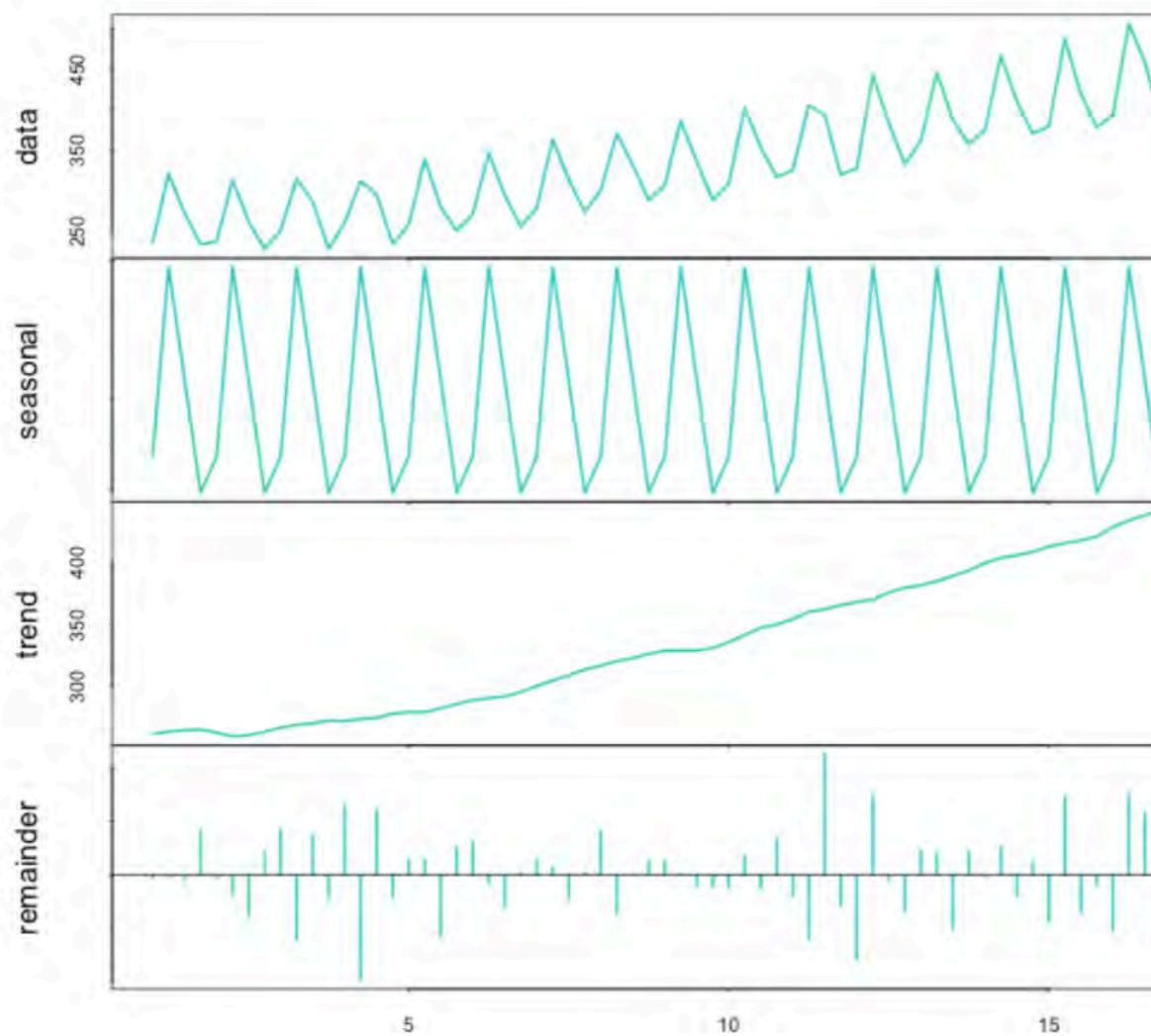
```
1 ts_air = ts(timeserie_air, frequency = 12)
2 decompose_air = decompose(ts_air, "multiplicative")
3
4 plot(as.ts(decompose_air$seasonal))
5 plot(as.ts(decompose_air$trend))
6 plot(as.ts(decompose_air$random))
7 plot(decompose_air)
```

Decomposition of multiplicative time series



# STL function

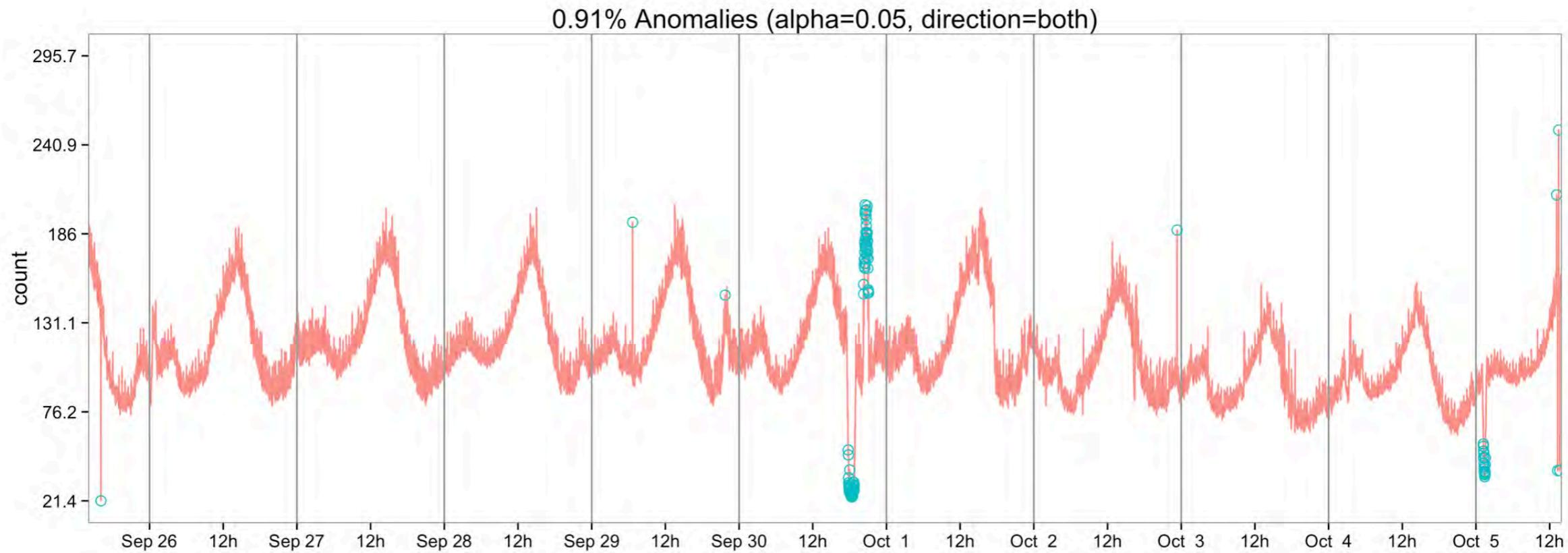
```
1 ts_beer = ts(timeserie_beer, frequency = 4)
2 stl_beer = stl(ts_beer, "periodic")
3 seasonal_stl_beer <- stl_beer$time.series[,1]
4 trend_stl_beer <- stl_beer$time.series[,2]
5 random_stl_beer <- stl_beer$time.series[,3]
6
7 plot(ts_beer)
8 plot(as.ts(seasonal_stl_beer))
9 plot(trend_stl_beer)
10 plot(random_stl_beer)
11 plot(stl_beer)
```



# SDL Decomposition

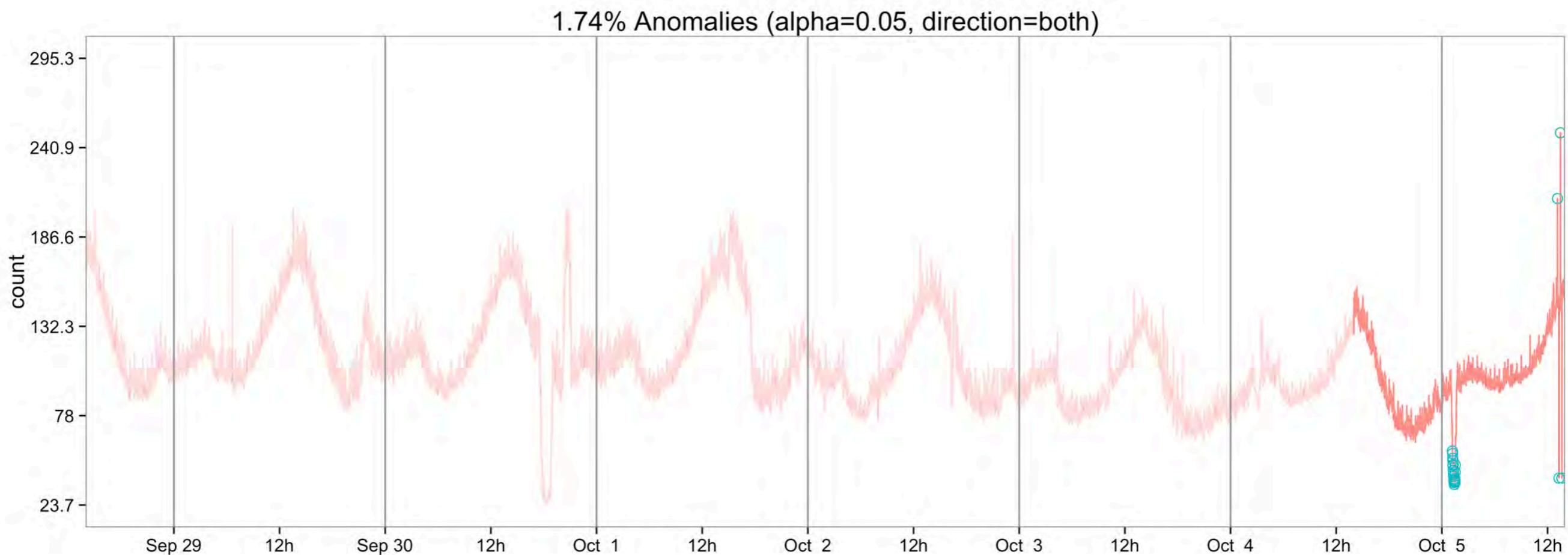
```
install.packages("devtools")
devtools::install_github("twitter/AnomalyDetection")
library(AnomalyDetection)
```

```
data(raw_data)
res = AnomalyDetectionTs(raw_data, max_anoms=0.02, direction='both', plot=TRUE)
res$plot
```



# SDL Decomposition

```
res = AnomalyDetectionTs(raw_data, max_anoms=0.02, direction='both', only_last="day", plot=TRUE)
res$plot
```



# CART

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
train <- agaricus.train
test <- agaricus.test

str(train)

List of 2
$ data :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
..@ i : int [1:143286] 2 6 8 11 18 20 21 24 28 32 ...
..@ p : int [1:127] 0 369 372 3306 5845 6489 6513 8380 8384 10991 ...
..@ Dim : int [1:2] 6513 126
..@ Dimnames:List of 2
..$: NULL
..$: chr [1:126] "cap-shape=bell" "cap-shape=conical" "cap-shape=convex" "ci"
..@ x : num [1:143286] 1 1 1 1 1 1 1 1 1 1 ...
..@ factors: list()
$ label: num [1:6513] 1 0 0 1 0 0 0 1 0 0 ...
```

## Basic training

We are using the `train` data. As explained above, both `data` and `label` are stored in a `list`.

In a *sparse* matrix, cells containing `0` are not stored in memory. Therefore, in a dataset mainly made of `0`, memory size is reduced. It is very common to have such a dataset.

We will train decision tree model using the following parameters:

- `objective = "binary:logistic"` : we will train a binary classification model ;
- `max.depth = 2` : the trees won't be deep, because our case is very simple ;
- `nthread = 2` : the number of cpu threads we are going to use;
- `nrounds = 2` : there will be two passes on the data, the second one will enhance the model by further reducing the difference between ground truth and prediction.

```
bstSparse <- xgboost(data = train$data, label = train$label, max.depth = 2, eta = 1,
nthread = 2, nrounds = 2, objective = "binary:logistic")
```

```
[0] train-error:0.046522
[1] train-error:0.022263
```

## Perform the prediction

The purpose of the model we have built is to classify new data. As explained before, we will use the `test` dataset for this step.

```
pred <- predict(bst, test$data)

size of the prediction vector
print(length(pred))

[1] 1611

limit display of predictions to the first 10
print(head(pred))

[1] 0.28583017 0.92392391 0.28583017 0.28583017 0.05169873 0.92392391
```

These numbers doesn't look like *binary classification* `{0,1}`. We need to perform a simple transformation before being able to use these results.

## Transform the regression in a binary classification

The only thing that **XGBoost** does is a *regression*. **XGBoost** is using `label` vector to build its *regression* model.

How can we use a *regression* model to perform a binary classification?

If we think about the meaning of a regression applied to our data, the numbers we get are probabilities that a datum will be classified as `1`. Therefore, we will set the rule that if this probability for a specific datum is `> 0.5` then the observation is classified as `1` (or `0` otherwise).

```
prediction <- as.numeric(pred > 0.5)
print(head(prediction))
```

```
[1] 0 1 0 0 0 1
```

# Measuring model performance

To measure the model performance, we will compute a simple metric, the *average error*.

```
err <- mean(as.numeric(pred > 0.5) != test$label)
print(paste("test-error=", err))

[1] "test-error= 0.0217256362507759"
```

# ARIMA

- \* Additive Outliers
- \* Level Shifts
- \* Transient Change
- \* Innovation Outliers
- \* Seasonal Level Shifts

**An Additive Outlier (AO)** represents an isolated spike.

**A Level Shift (LS)** represents an abrupt change in the mean level and it may be seasonal (Seasonal Level Shift, SLS) or not.

**A Transient Change (TC)** represents a spike that takes a few periods to disappear.

**An Intervention Outlier (IO)** represents a shock in the innovations of the model.

Pre-specified outliers are able to satisfactorily describe the dynamic pattern of untypical effects and can be captured by means of intervention variables.

# ARIMA

Import Library

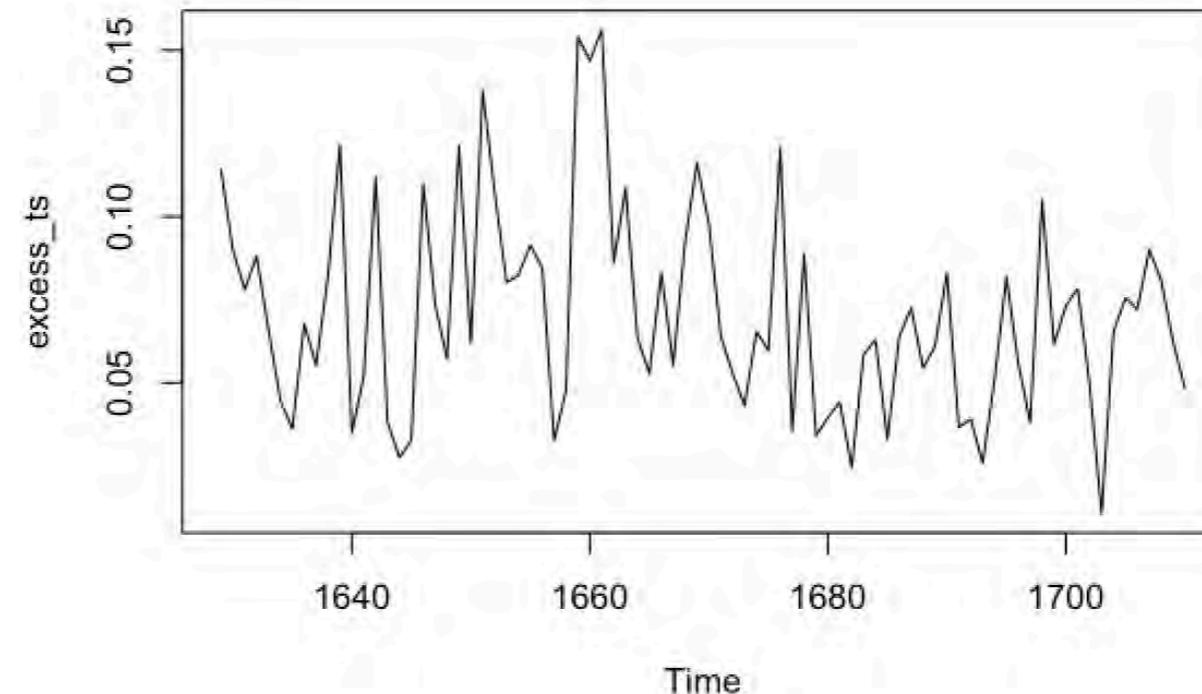
```
library(tsoutliers)
library(TSA)
library(lmtest)
library(astsa)
```

# ARIMA

```
url <- "https://www.openintro.org/stat/data/arbutnot.csv"
abutnotdot <- read.csv(url, header=TRUE)

boys_ts <- ts(abutnotdot$boys, frequency=1, start = abutnotdot$year[1])
girls_ts <- ts(abutnotdot$girls, frequency=1, start = abutnotdot$year[1])

delta_ts <- boys_ts - girls_ts
excess_ts <- delta_ts/girls_ts
plot(excess_ts)
```



```
outliers_excess_ts <- tso(excess_ts, types = c("TC", "AO", "LS", "IO", "SLS"))
outliers_excess_ts
```

Series: excess\_ts

Regression with ARIMA(0,0,0) errors

Coefficients:

|      | intercept | TC31 |
|------|-----------|------|
|      | 0.066     | 0.10 |
| s.e. | 0.003     | 0.02 |

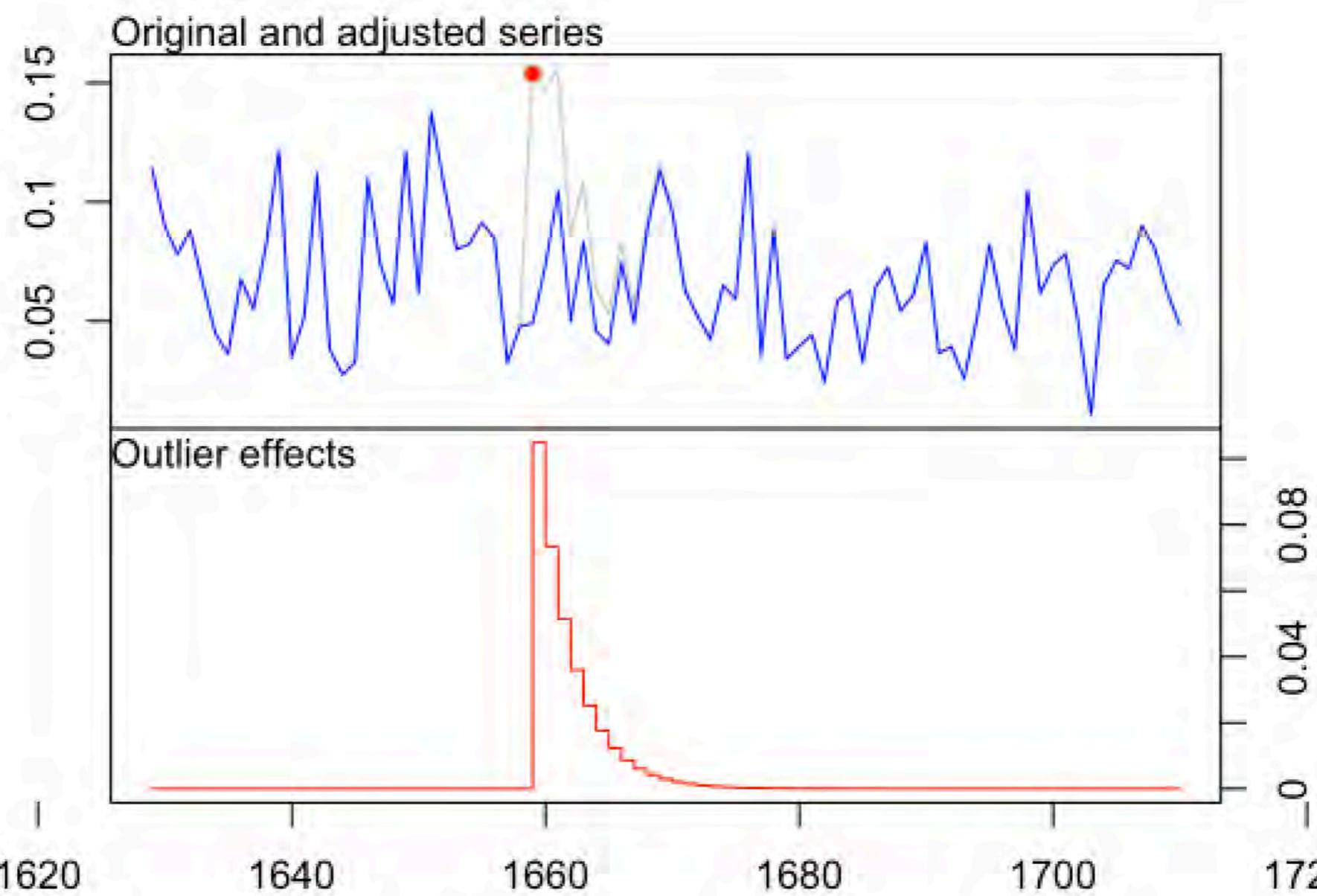
sigma^2 estimated as 0.000738: log likelihood=180

AIC=-355 AICc=-354 BIC=-347

Outliers:

|   | type | ind | time | coefhat | tstat |
|---|------|-----|------|---------|-------|
| 1 | TC   | 31  | 1659 | 0.105   | 5.28  |

```
plot(outliers_excess_ts)
```



```
| outliers_excess_ts$outliers
```

```
> outliers_excess_ts$outliers
 type ind time coefhat tstat
1 TC 31 1659 0.1 5.3
```



the effect of such transient change, comparing the original time series under analysis with the same without such transient change.

```
#length of our time series
n <- length(excess_ts)

time index where the outliers have been detected
(outliers_idx <- outliers_excess_ts$outliers$ind)

transient change outlier at the same time index as found
for our time series
mo_tc <- outliers("TC", outliers_idx)

transient change effect data is stored into a one-column
matrix, tc
tc <- outliers.effects(mo_tc, n)
```

```
> tc
```

TC31

```
[1,] 0.0e+00
[2,] 0.0e+00
[3,] 0.0e+00
[4,] 0.0e+00
[5,] 0.0e+00
```

```
[30,] 0.0e+00
[31,] 1.0e+00
[32,] 7.0e-01
[33,] 4.9e-01
[34,] 3.4e-01
[35,] 2.4e-01
[36,] 1.7e-01
[37,] 1.2e-01
```

```
converting to a number
coefhat <- as.numeric(outliers_excess_ts$outliers["coefhat"])

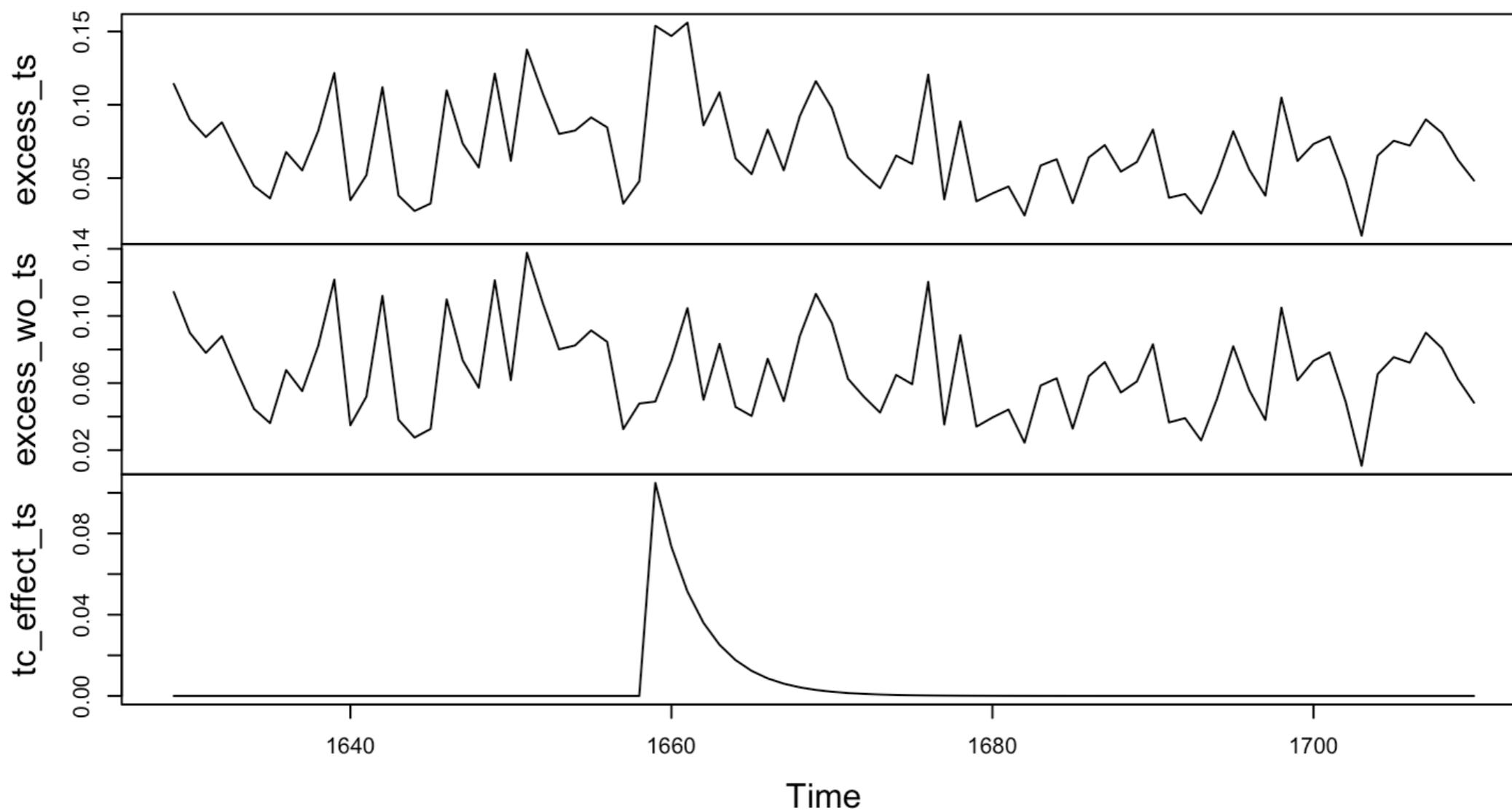
obtaining the transient change data with same magnitude
as determined by the tso() function
tc_effect <- coefhat*tc

defining a time series for the transient change data
tc_effect_ts <- ts(tc_effect, frequency = frequency(excess_ts),
 start = start(excess_ts))

subtracting the transient change intervention to the original
time series, obtaining a time series without the transient
change effect
excess_wo_ts <- excess_ts - tc_effect_ts

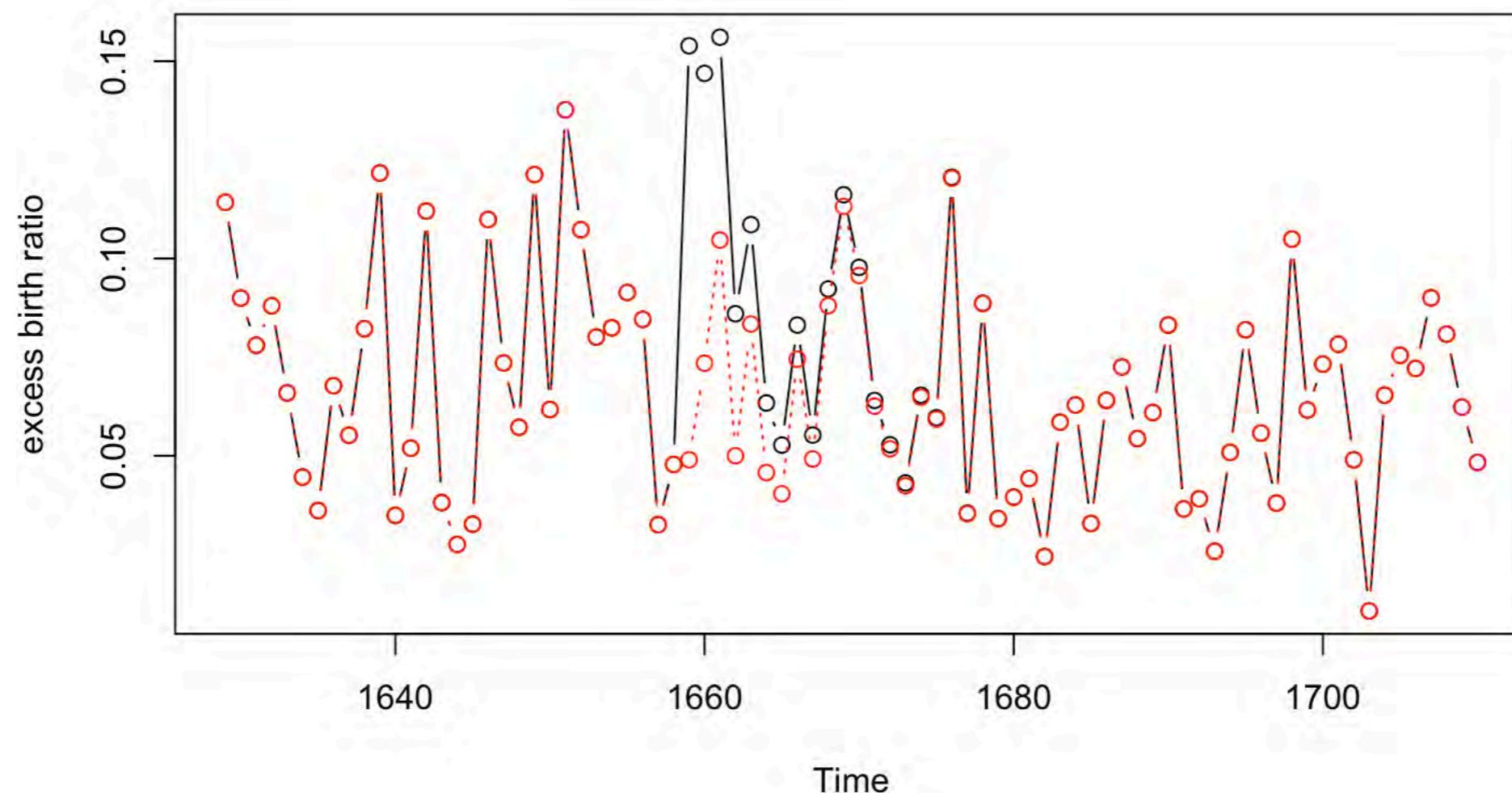
plot of the original, without intervention and transient
change time series
plot(cbind(excess_ts, excess_wo_ts, tc_effect_ts))
```

**cbind(excess\_ts, excess\_wo\_ts, tc\_effect\_ts)**



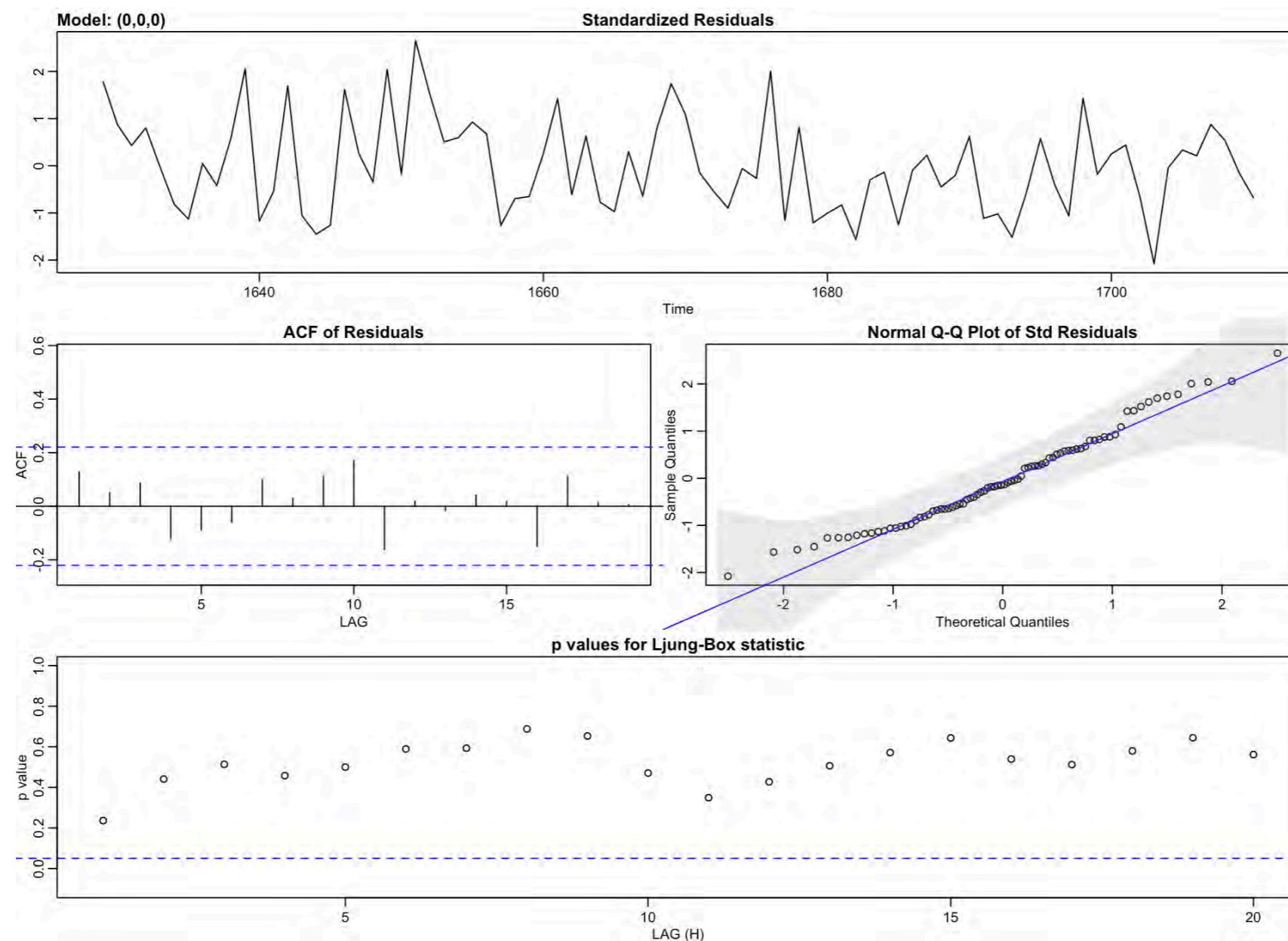
highlight the difference between the original time series and same time series without the transient change effect.

```
plot(excess_ts, type = 'b', ylab = "excess birth ratio")
lines(excess_wo_ts, col = 'red', lty = 3, type = 'b')
```



check on the residuals of the time series without the transient change effect confirms validity of the ARIMA(0,0,0) model.

```
sarima(excess_wo_ts, p=0, d=0, q=0)
```



# Workshop 4.19 Anomaly Detection in Time Series

Try all techniques from 604-634

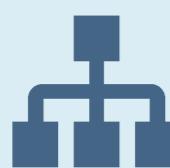


# Introduction to Deep Learning

# What is DeepLearning?



Part of the machine learning field of learning representations of data. Exceptional effective at learning patterns.

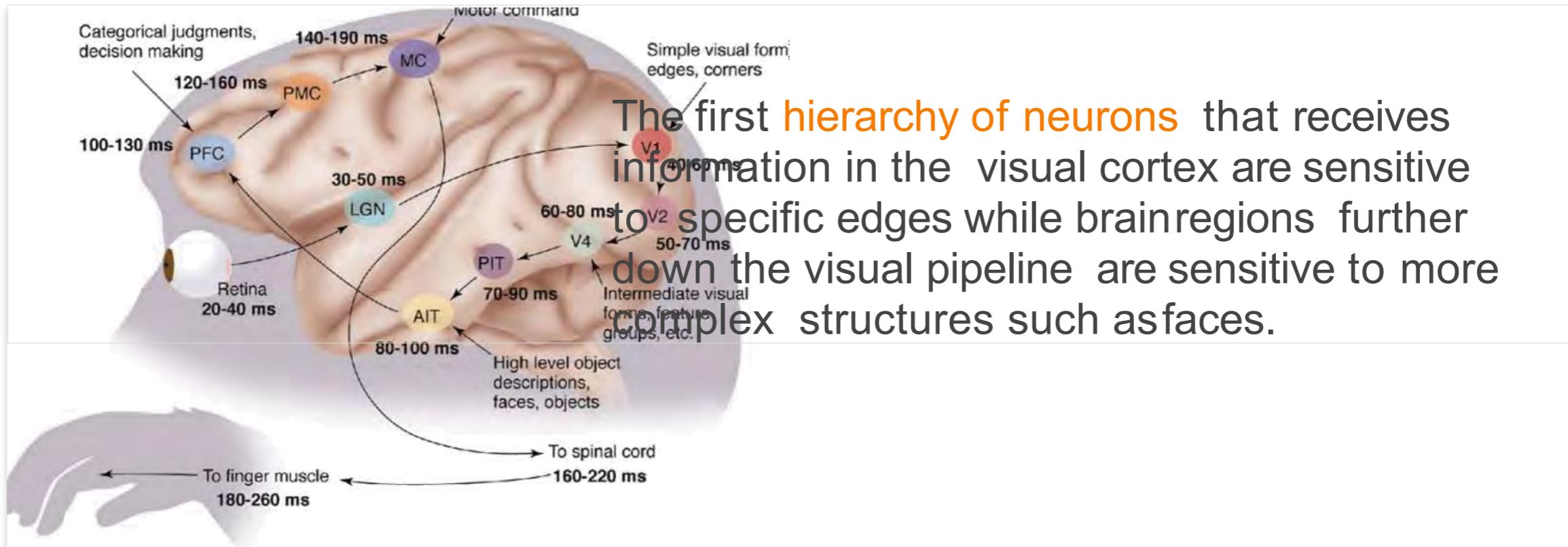


Utilizes learning algorithms that derive meaning out of data by using a **hierarchy** of multiple layers that **mimic the neural networks of our brain**.



If you provide the system tons of information, it begins to understand it and respond in useful ways.

# Inspired by the Brain



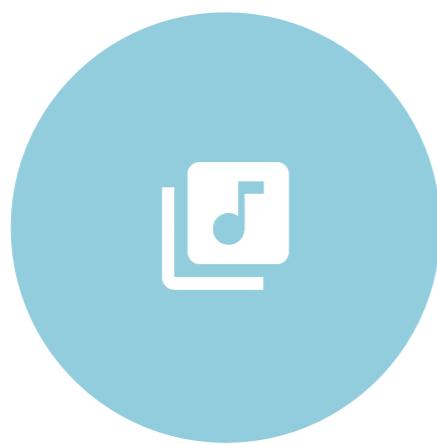
Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.

1

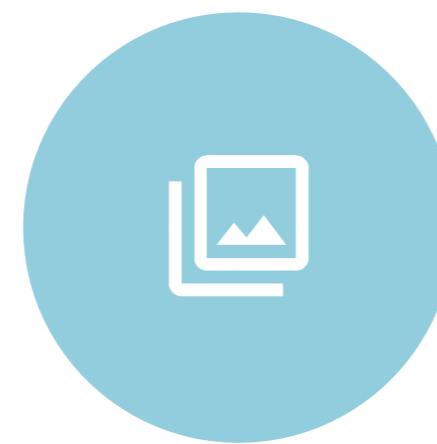
One learning algorithm hypothesis: all significant mental algorithms are learned except for the learning and reward machinery itself.

# Why Deep Learning?

## Applications



Speech  
Recognition



Computer  
Vision



Natural Language  
Processing

# A brief History

Along time ago...

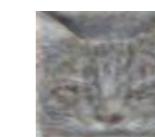


**1958** Perceptron

**1974** Backpropagation



Convolution Neural Networks for  
Handwritten Recognition



Google Brain Project  
16k Cores



**2012**

**1969**

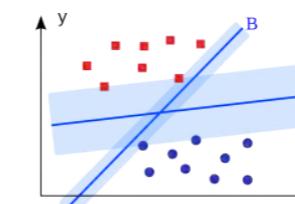
Perceptron criticized



awkward silence (AI Winter)

**1995**

SVM reigns



**2006**

Restricted  
Boltzmann  
Machine



**2012**

AlexNet wins  
ImageNet

IMAGENET

# A brief History

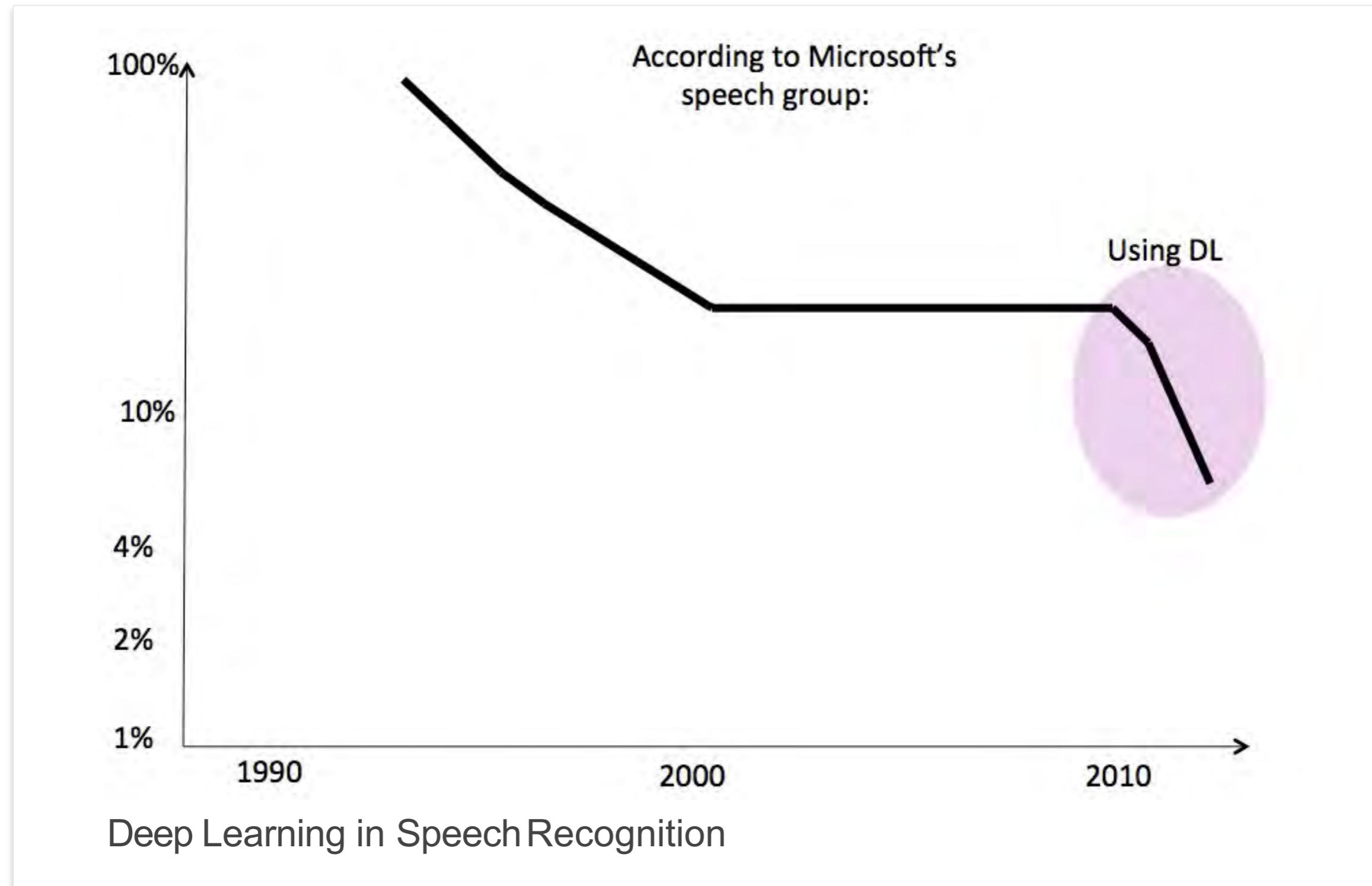
The Big Bang aka “One net to rule them all”



ImageNet: The “computer vision World Cup”

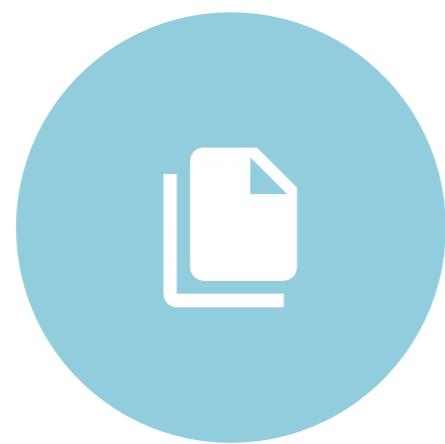
# A brief History

The Big Bang aka "One net to rule them all"

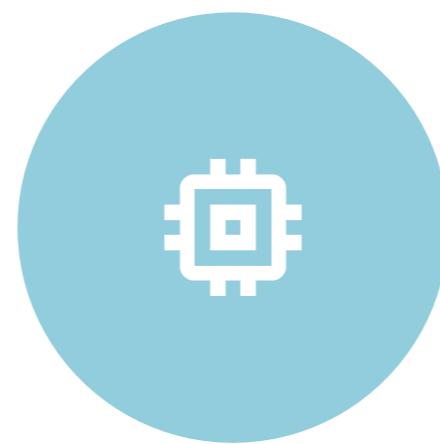


# What changed?

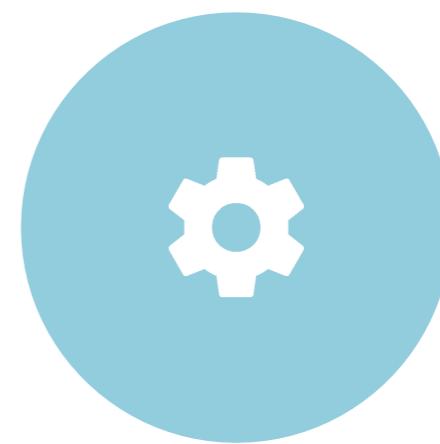
Old wine in new bottles



Big Data  
(Digitalization)



Computation  
(Moore's Law, GPUs)



Algorithmic  
Progress

# The Big Players

## Superstar Researchers



Geoffrey Hinton: University of Toronto & Google



Yann LeCun: New York University & Facebook



Andrew Ng: Stanford & Baidu



Yoshua Bengio: University of Montreal



Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

# The Big Players

Companies

facebook

 Microsoft

YAHOO!

Google



IBM



NVIDIA.

Baidu 百度

# The Big Players

## Startups



vicarious

nervana

ersatz<sup>labs</sup>



sentient



DEEPMIND



AlchemyAPI<sup>TM</sup>  
An IBM Company

enlitic

SKYMI<sup>ND</sup>



Numenta  
OpenAI

clarifai

SIGNALSENSE



cortica<sup>TM</sup>  
In Every Image



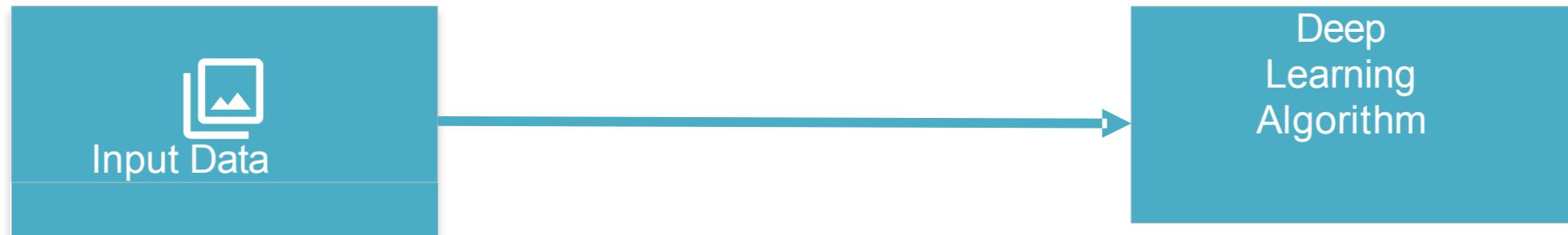
MetaMind

wit.ai DNNresearch

Acquired

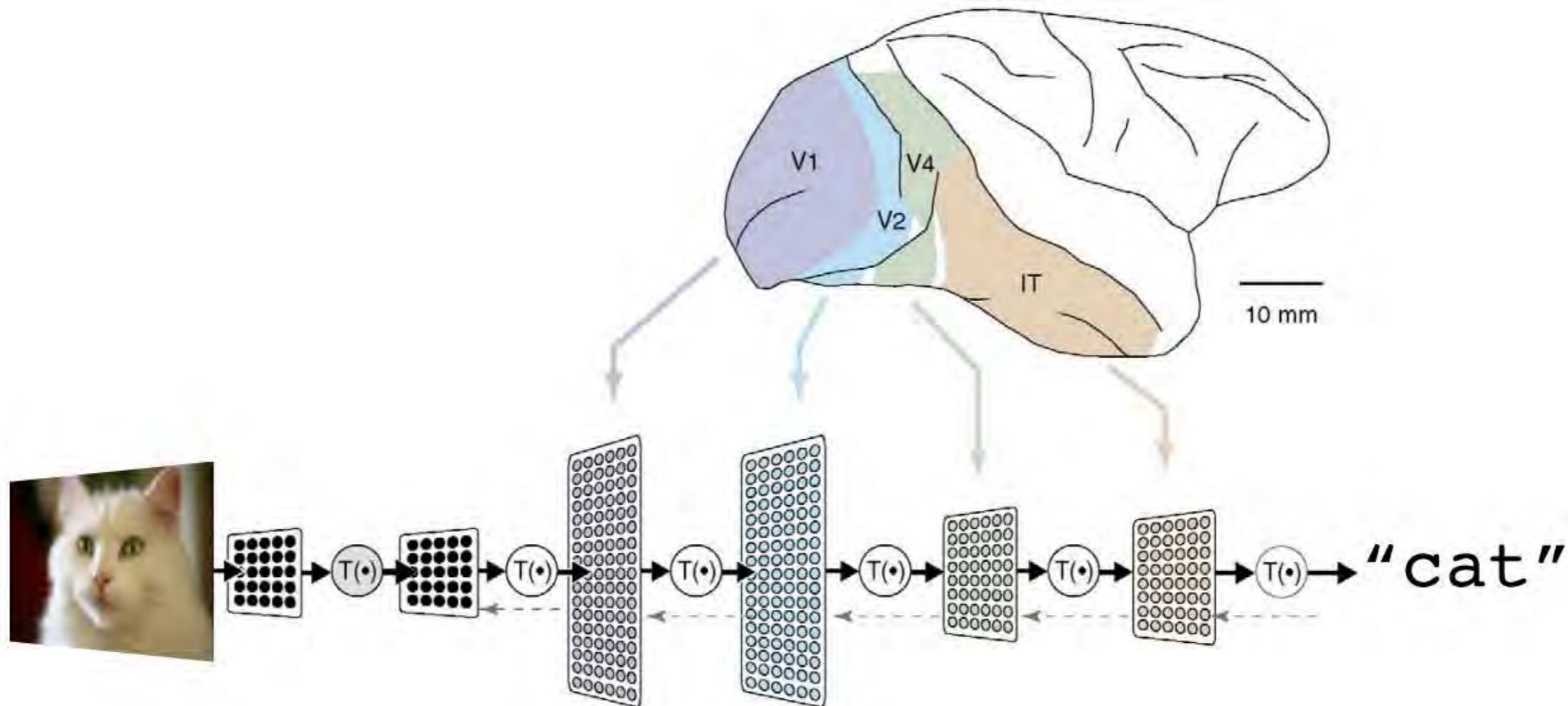
# Deep Learning - Basics

No more feature engineering



# Deep Learning - Basics

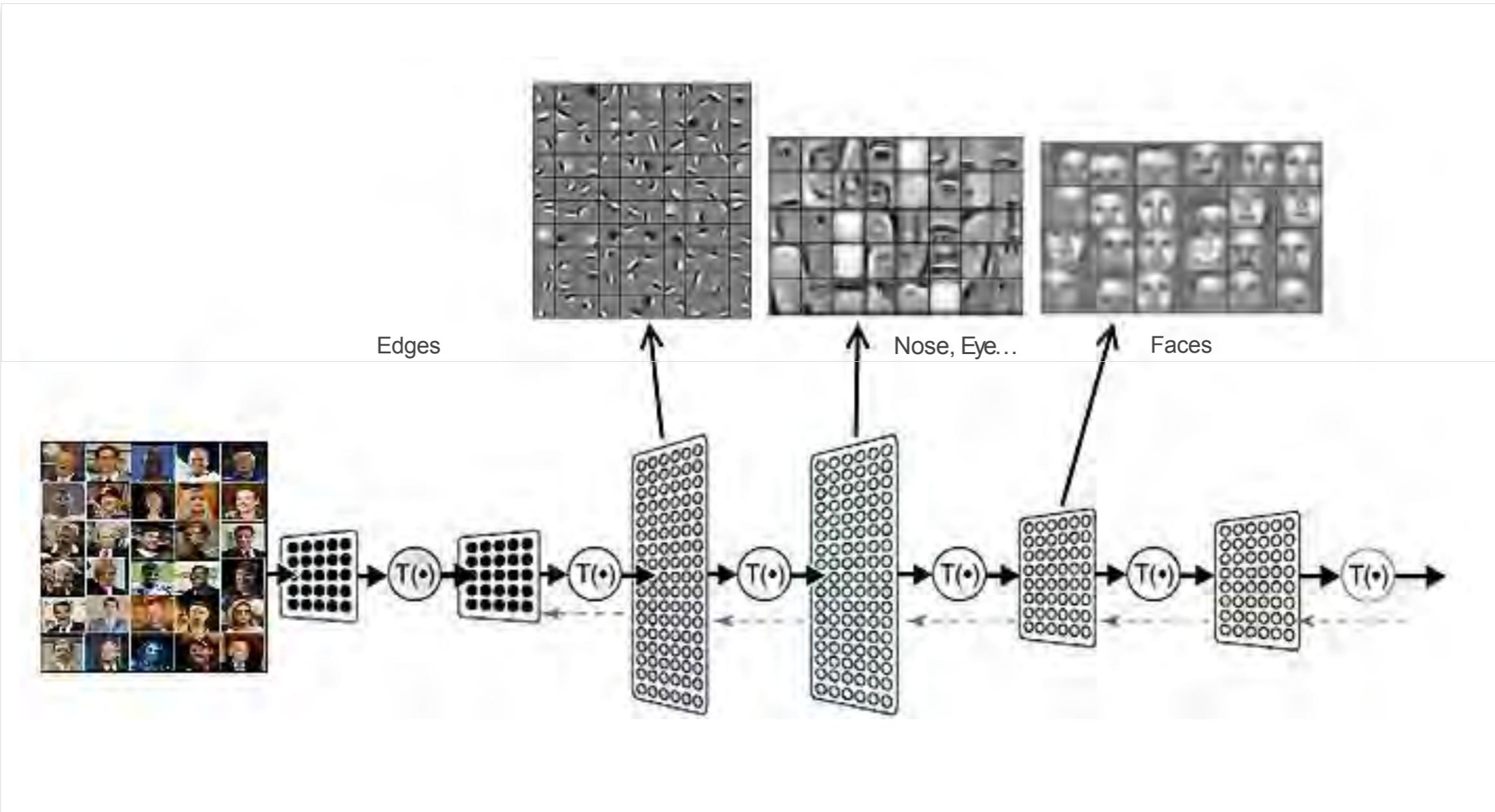
## Architecture



A deep neural network consists of a **hierarchy of layers**, whereby each layer **transforms the input data** into more abstract representations (e.g. edge  $\rightarrow$  nose  $\rightarrow$  face). The output layer combines those features to make predictions.

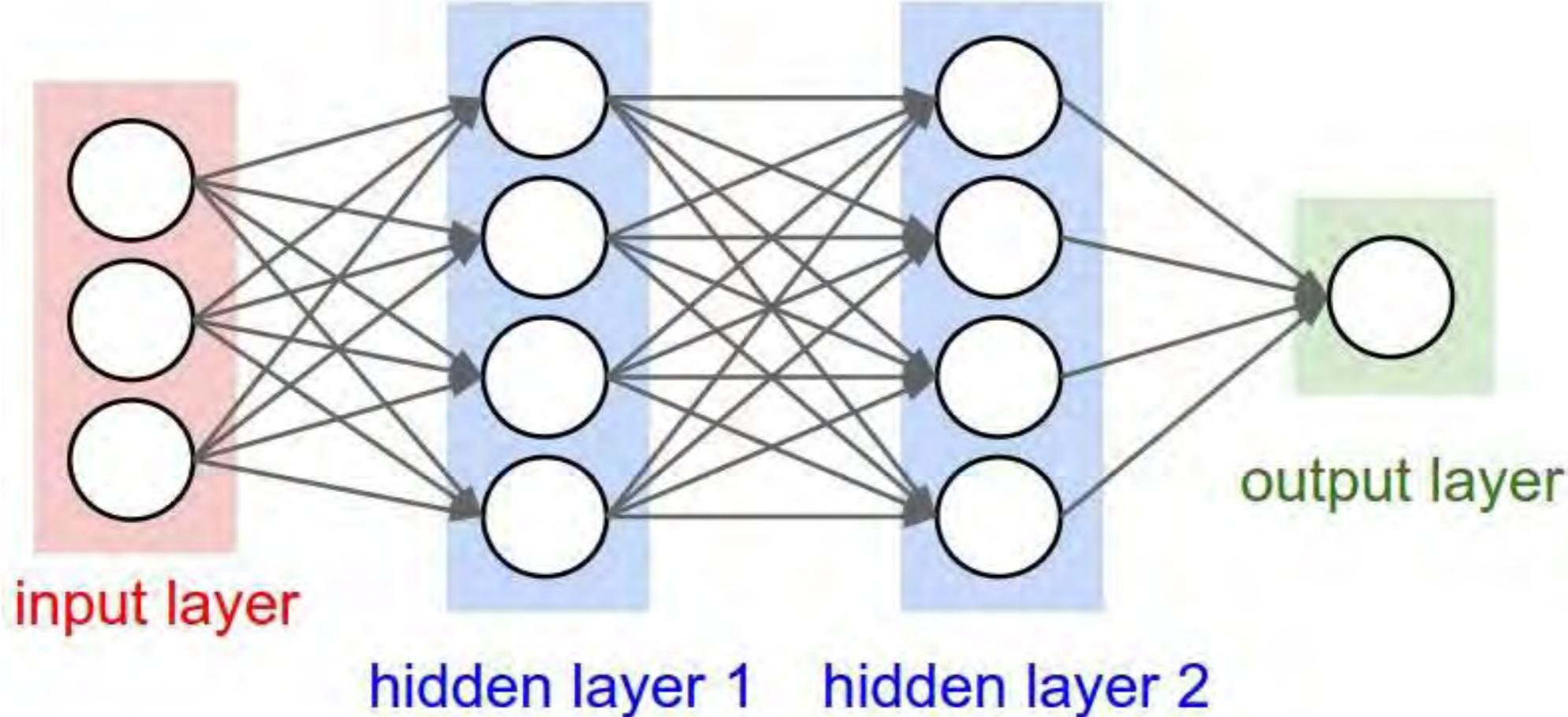
# Deep Learning - Basics

What did it learn?



# Deep Learning - Basics

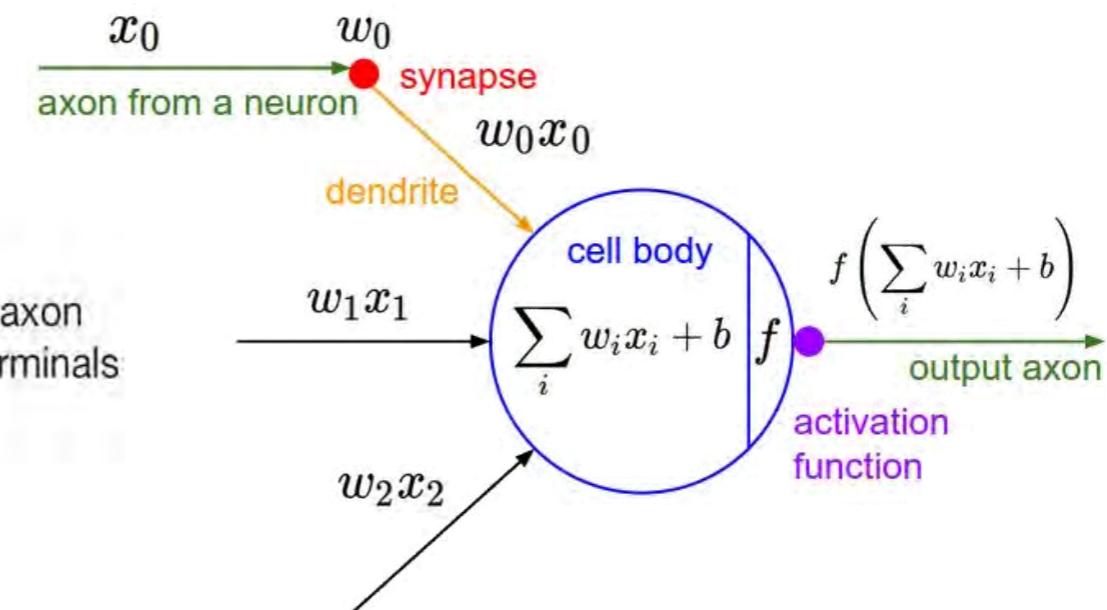
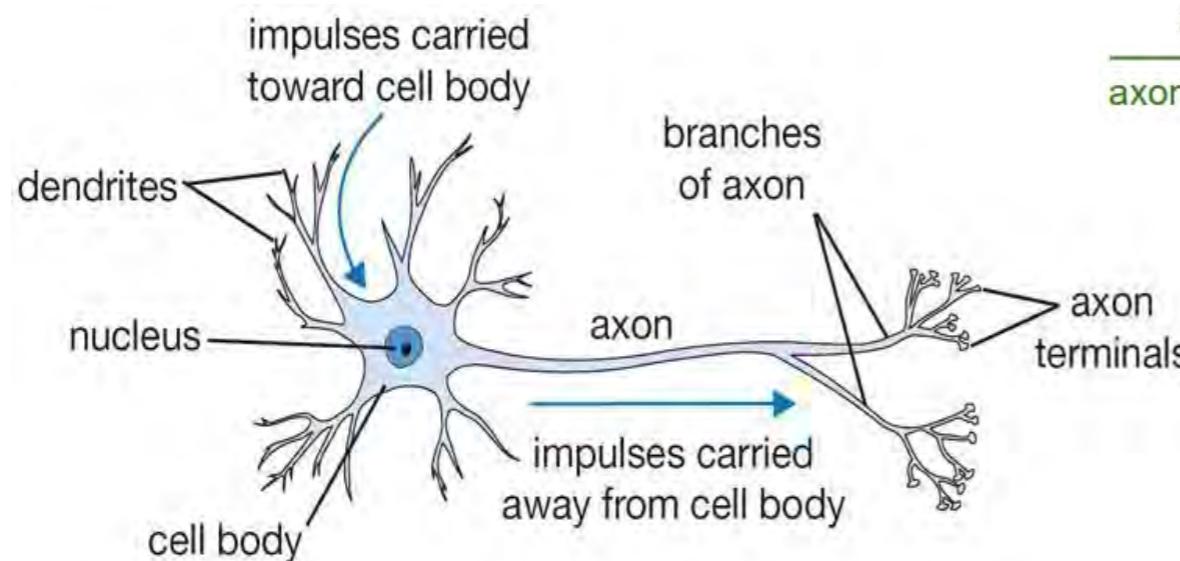
## Artificial Neural Networks



Consists of one input, one output and multiple fully-connected hidden layers in- between. Each layer is represented as a series of neurons and **progressively extracts higher and higher-level features** of the input until the final layer essentially makes a decision about what the input shows. The more layers the network has, the higher- level features it will learn.

# Deep Learning - Basics

## The Neuron



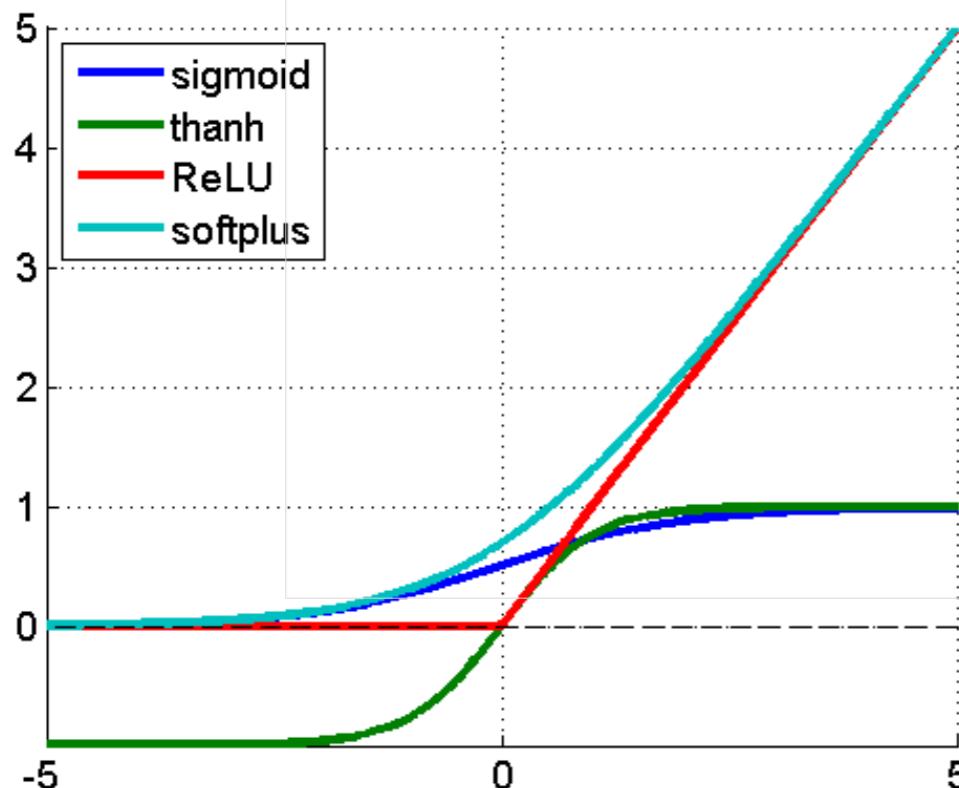
An artificial neuron contains a **nonlinear activation function** and has several incoming and outgoing **weighted connections**.



Neurons are **trained to filter and detect specific features** or patterns (e.g. edge, nose) by receiving weighted input, transforming it with the activation function and passing it to the outgoing connections.

# Deep Learning - Basics

## Non-linear Activation Function



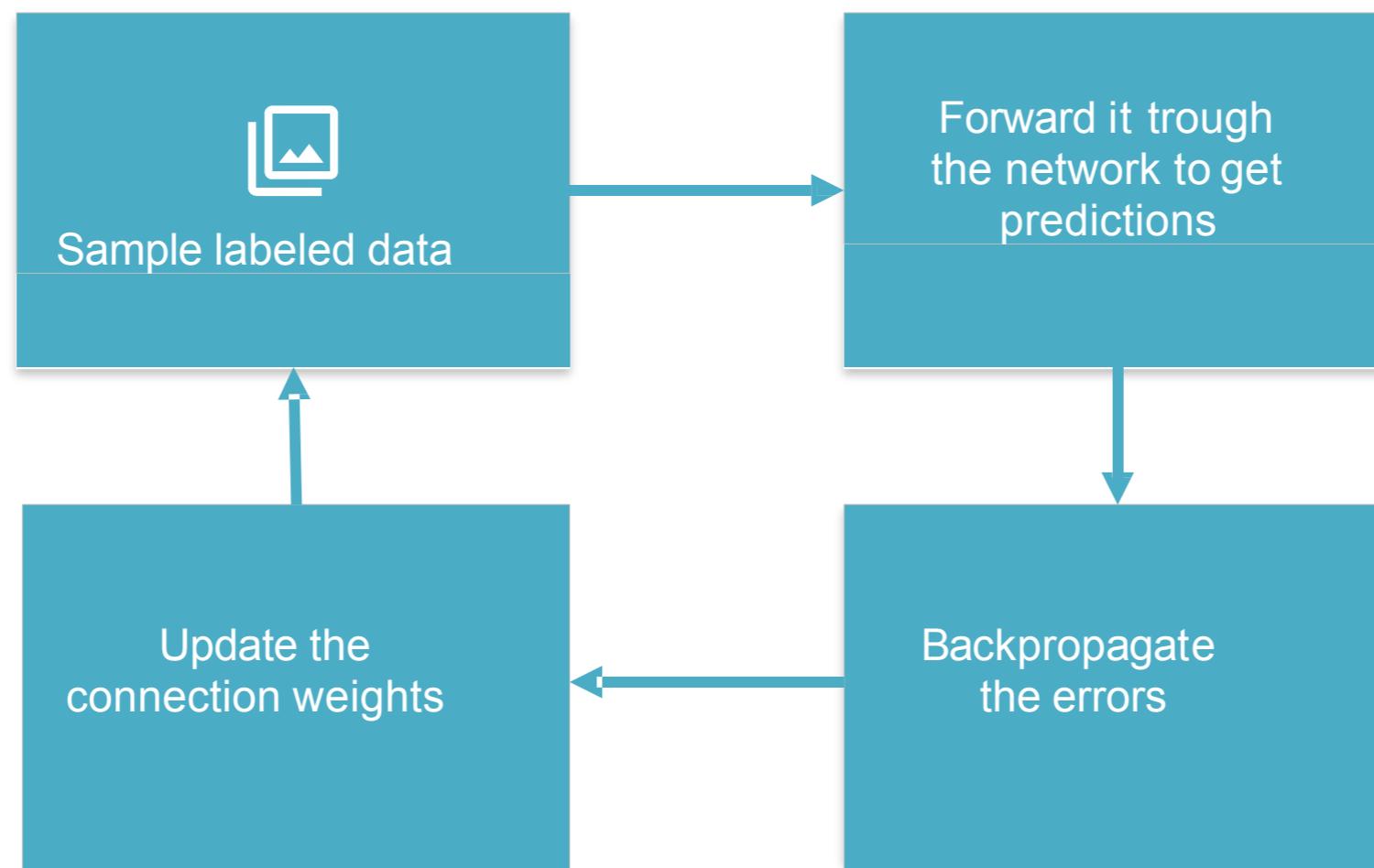
Most deep networks use **ReLU** -  $\max(0, x)$  - nowadays for hidden layers, since it trains much faster, is more expressive than logistic function and prevents the gradient vanishing problem.



**Non-linearity** is needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function.

# Deep Learning - Basics

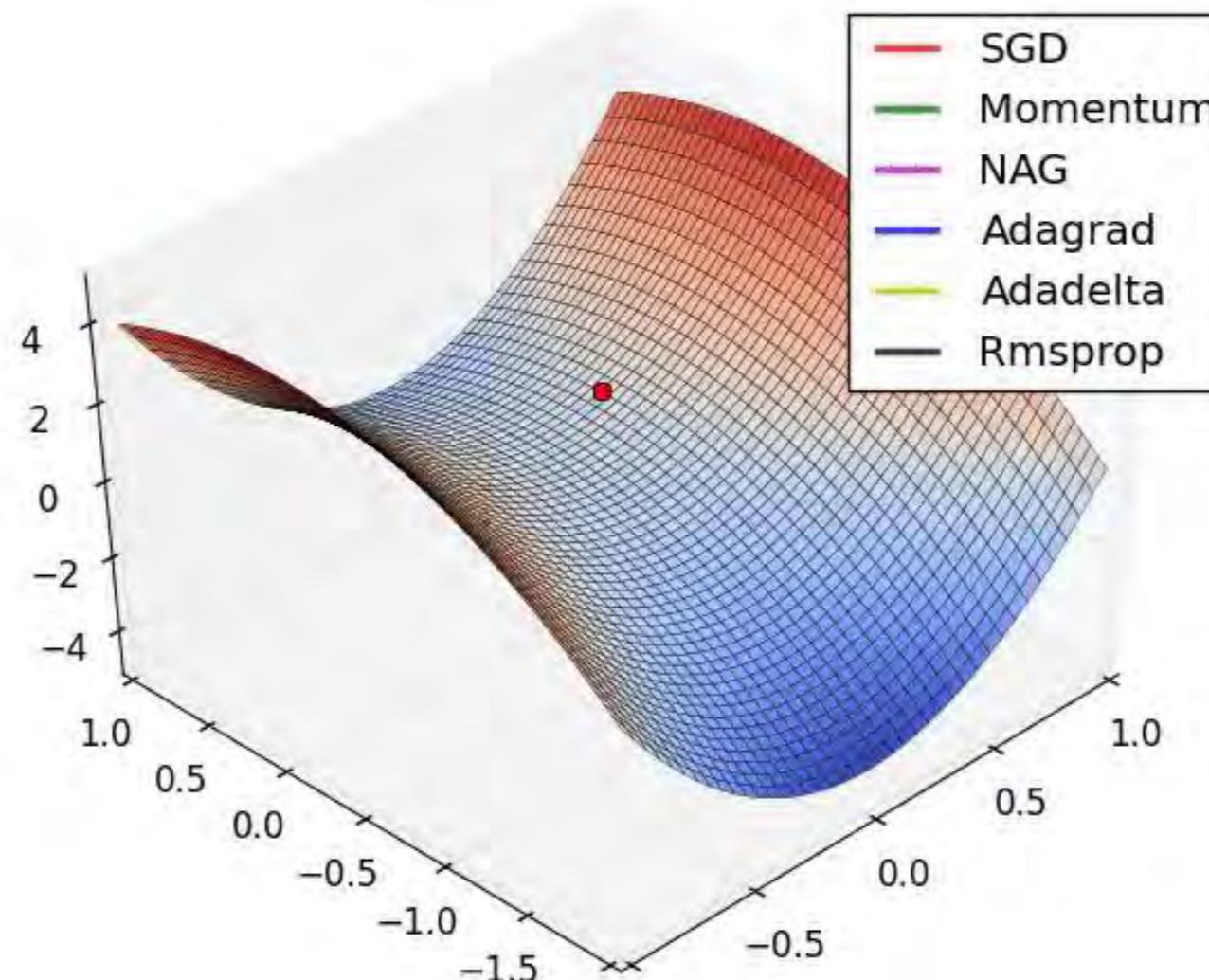
## The Training Process



Learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then **using this error signal to change the weights** (or parameters) so that predictions get more accurate.

# DeepLearning - Basics

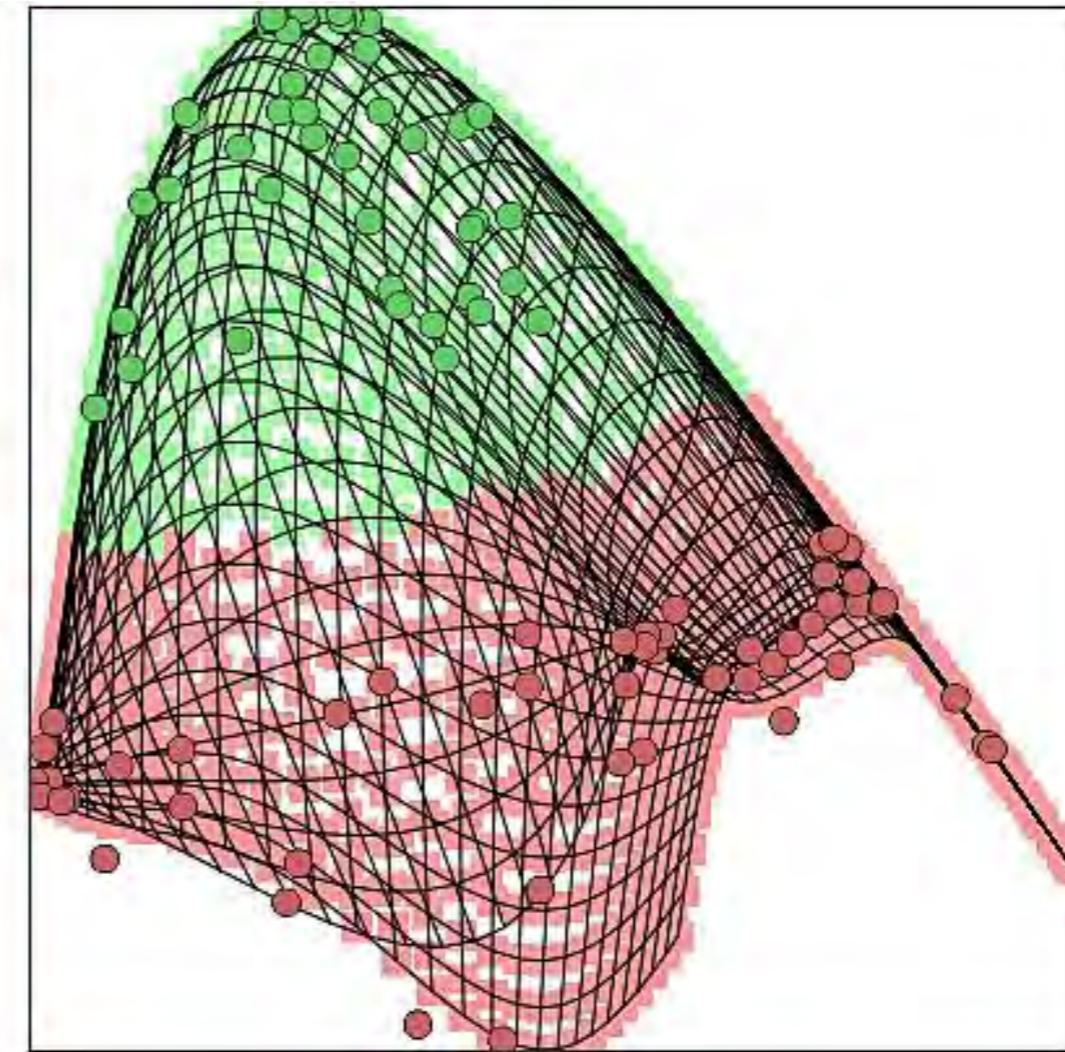
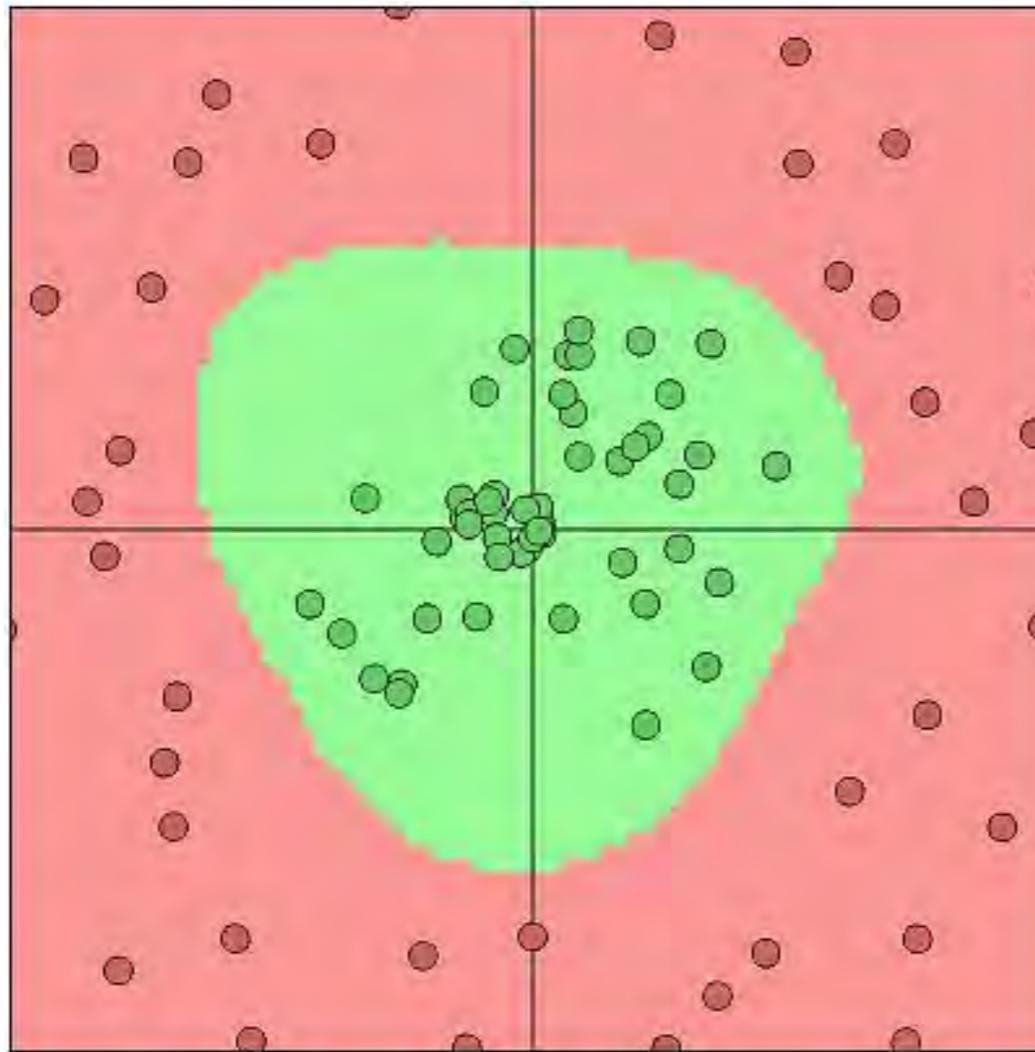
## Gradient Descent



Gradient Descent finds the (local) minimum of the cost function (used to calculate the output error) and is used to adjust the weights.

# Deep Learning - Basics

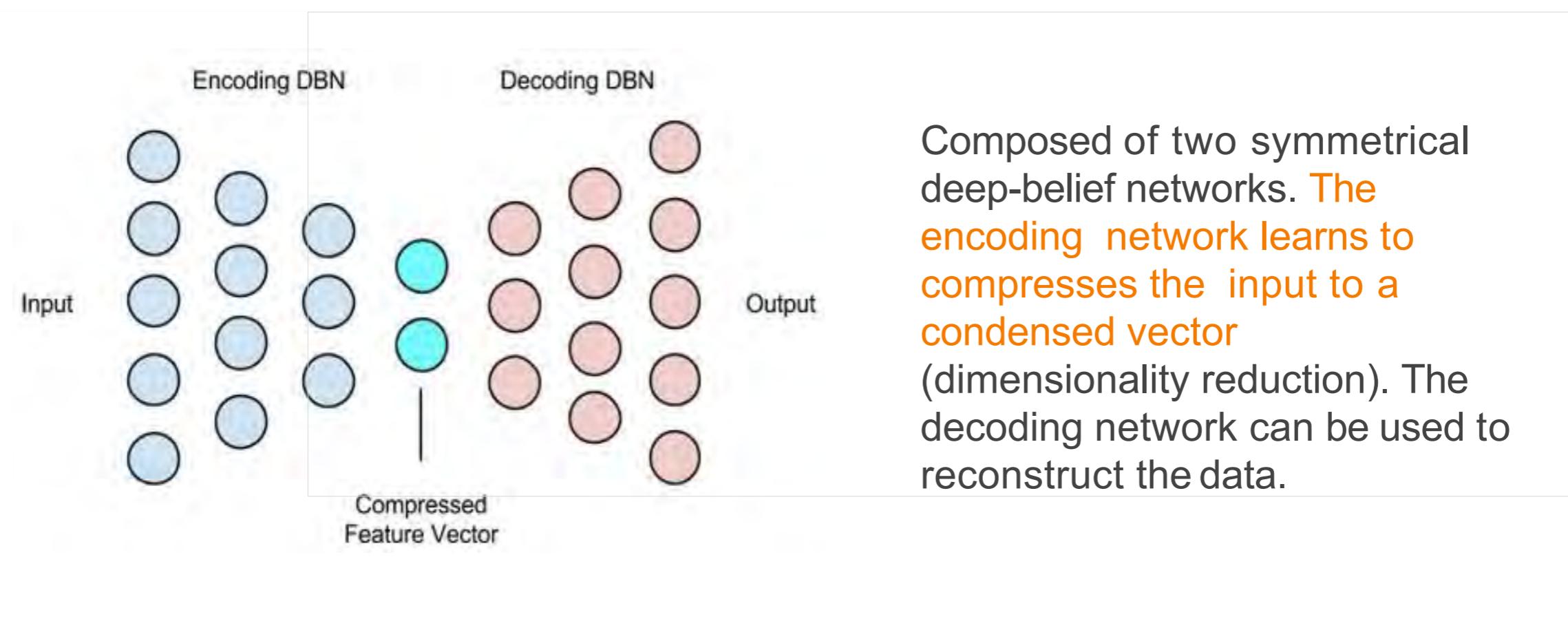
Data transformation in other dimensions



A neural network is **transforming the data into other dimensions** to solve the specified problem.

# Deep Learning - Basics

## Deep Autoencoders

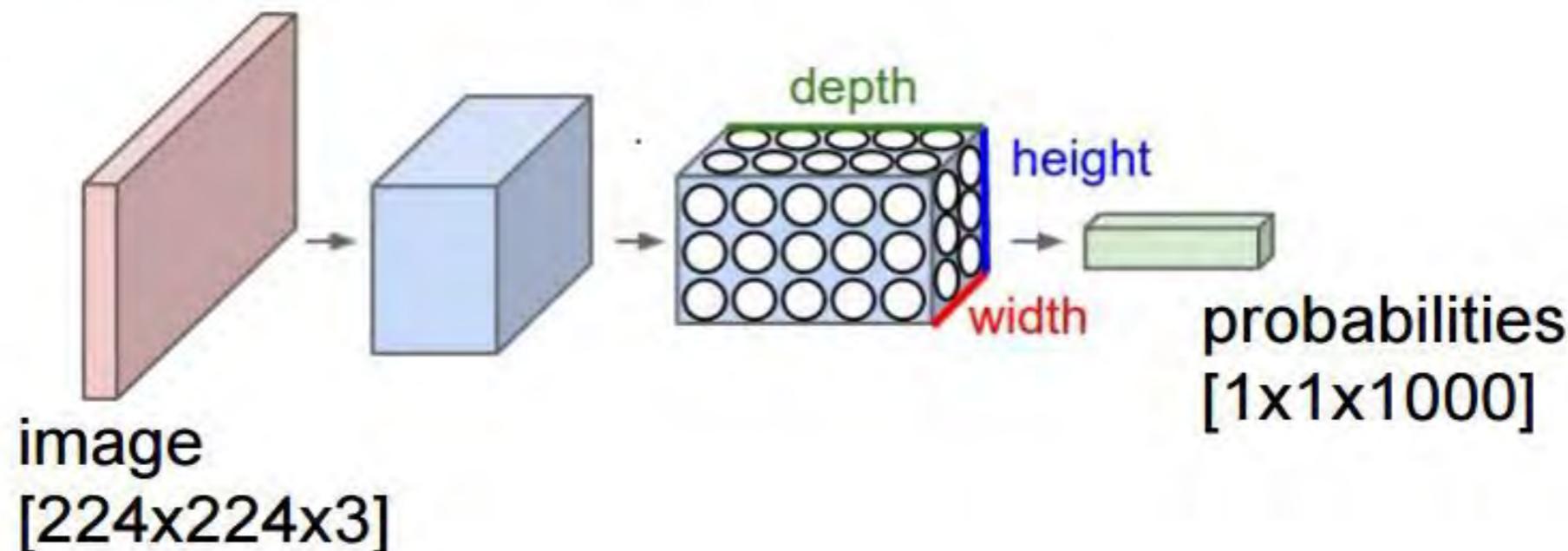


Topic Modeling: Document in a collection is converted to a Bag-of-Words and transformed to a compressed feature vector using an autoencoder. The distance from every other document-vector can be measured and nearby document-vectors fall under the same topic.

# Deep Learning - Basics

## Convolutional Neural Nets(CNN)

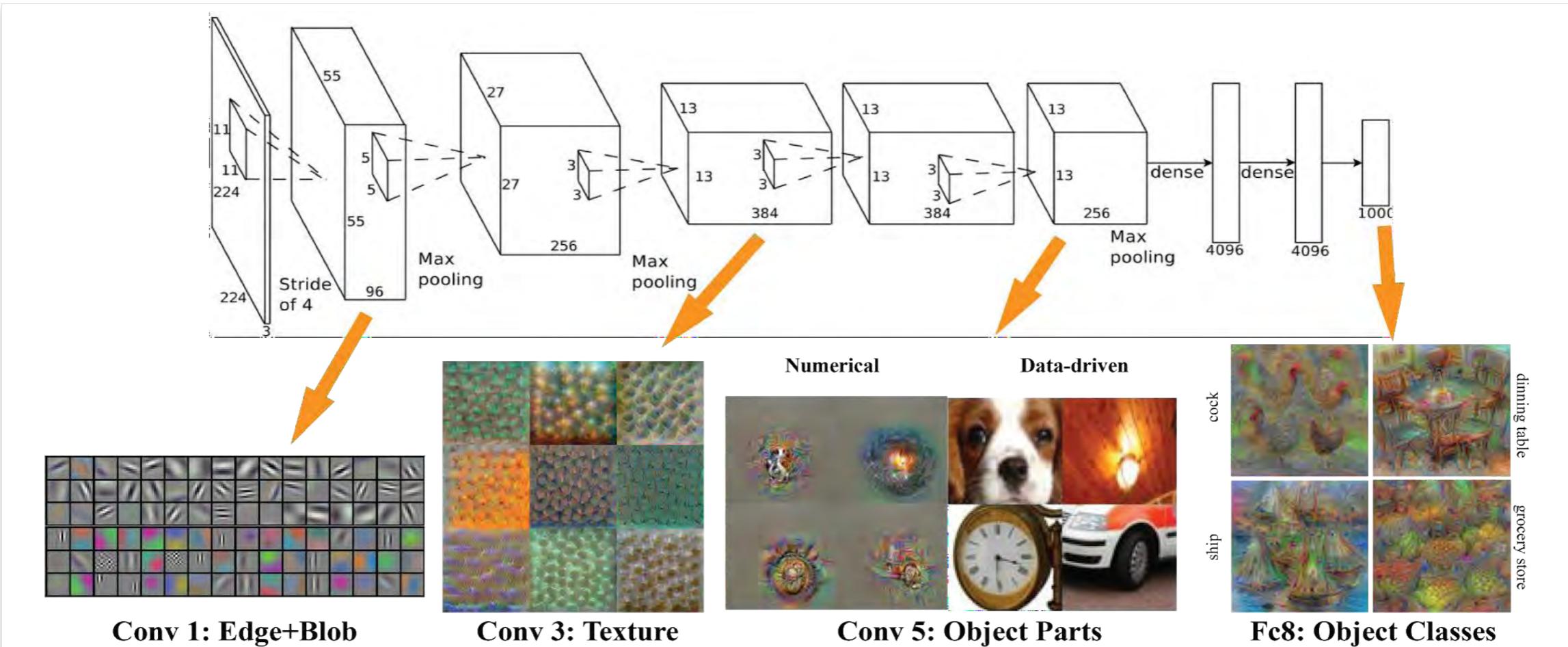
Convolutional Neural Networks learn a complex representation of visual data using vast amounts of data. They are **inspired by the human visual system** and learn **multiple layers of transformations**, which are applied on top of each other to extract a progressively more sophisticated representation of the input.



Every layer of a CNN **takes a 3D volume of numbers and outputs a 3D volume of numbers**. E.g. Image is a  $224 \times 224 \times 3$  (RGB) cube and will be transformed to  $1 \times 1000$  vector of probabilities.

# Deep Learning - Basics

## Convolutional Neural Nets(CNN)

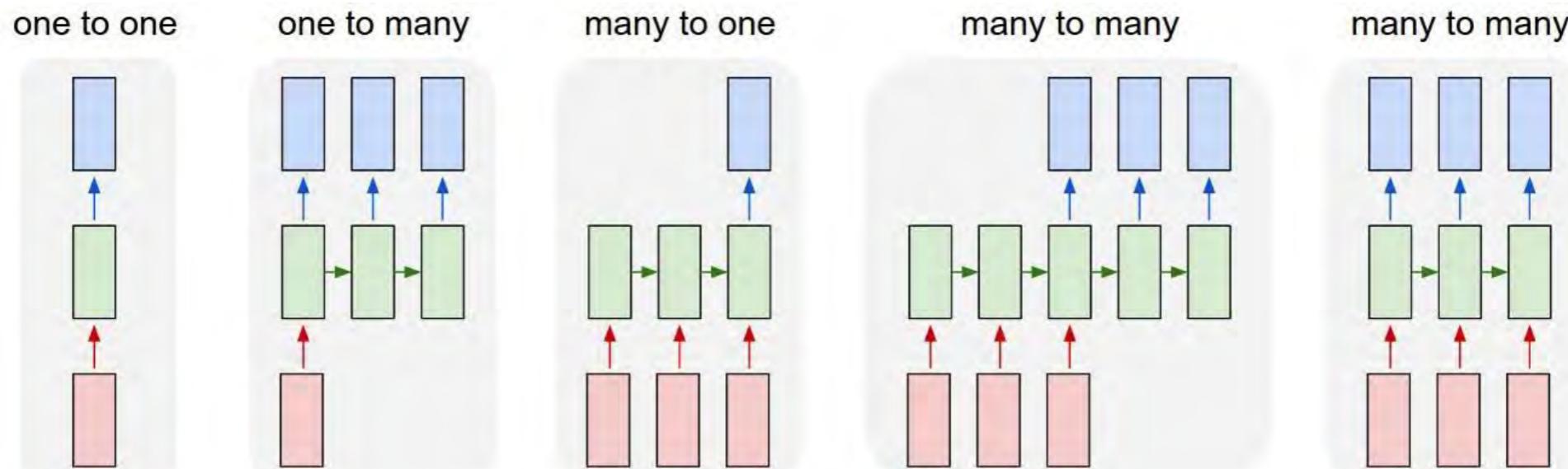


Convolution layer is a feature detector that automagically learns to **filter out not needed information** from an input by using convolution kernel.

Pooling layers compute the max or **average value of a particular feature over a region** of the input data (*downsizing of input images*). Also helps to detect objects in some unusual places and reduces memory size.

# Deep Learning - Basics

## Recurrent Neural Nets (RNN)



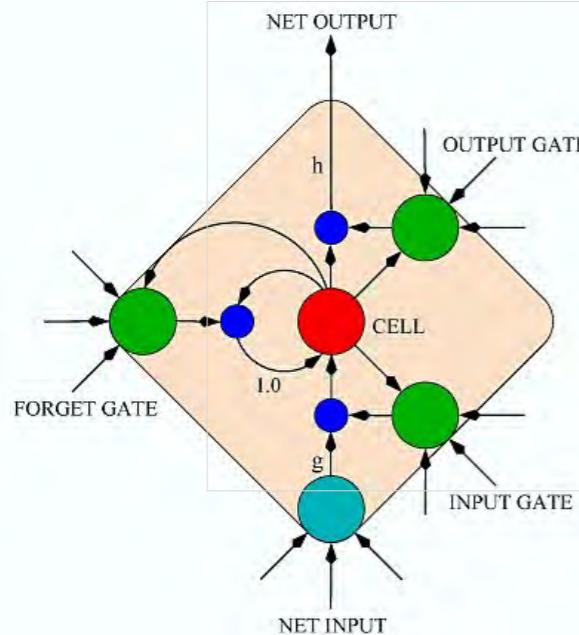
RNNs are general computers which can learn algorithms to map input sequences to output sequences (flexible-sized vectors). The output vector's contents are influenced by the entire history of inputs.



State-of-the-art results in time series prediction, adaptive robotics, handwriting recognition, image classification, speech recognition, stock market prediction, and other sequence learning problems.  
*Everything can be processed sequentially.*

# Deep Learning - Basics

## Long Short-Term Memory RNN(LSTM)



A Long Short-Term Memory (LSTM) network is a particular type of recurrent network that **works slightly better in practice**, owing to its more powerful update equation and some appealing back propagation dynamics.



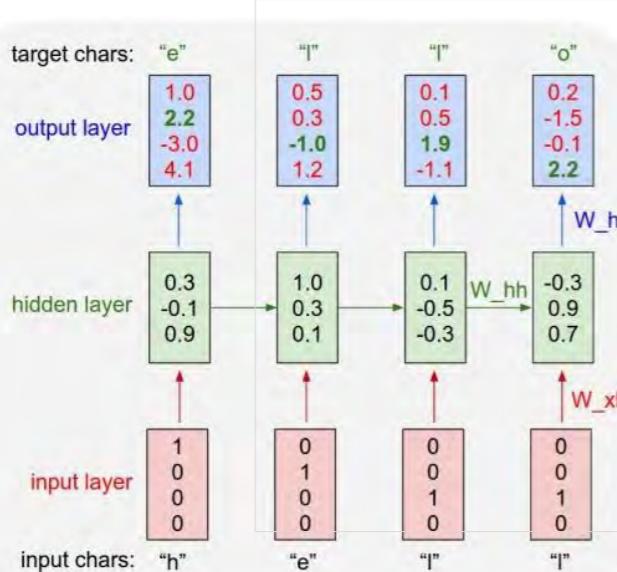
The LSTM units give the network **memory cells with read, write and reset operations**. During training, the network can learn when it should remember data and when it should throw it away.



Well-suited to learn from experience to classify, process and predict time series when there are **very long time lags of unknown size between important events**.

# Deep Learning - Basics

## Recurrent Neural Nets (RNN) – Generating Text



To train the RNN, insert characters sequentially and predict the probabilities of the next letter. Backpropagate error and update RNN's weights to increase the confidence of the correct letter (green) and decrease the confidence of all other letters (red).

The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servitious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS) [<http://www.humah.yahoo.com/guardian.cfm/7754800786d17551963s89.htm>]

Trained on structured Wikipedia markdown. Network learns to spell English words completely from scratch and copy general syntactic structures.

# Deep Learning - Basics

# Recurrent Neural Nets (RNN) – Generating Text

To **generate text**, we feed a character into the trained RNN and get a distribution over what characters are likely to come next (*red = likely*). We sample from this distribution, and feed it right back in to get the next letter.

glish [[ weekly newspaper ]] \* [[ YNetNews ]] \* [[ http://www.ynetnews.co.ilish c[ Caakly ] cawspaper ]) \* [ hTaa at ]) (' ( http://www.bacahets.co iaci - lhSoip] i sec ] enp] s . ' [ Co \* wess ] s a[ a d : xne. waea.. awatoa een , pCci et ned lox ] gicil || s' [ sAmFeSahon ] t' : : , imomw- 2 ♦ pii isoessis . / er syz . sf penn alruel | rra . ' # \* : oDuFreiuep , : b1edr . < ahb - nptwt . xi gh adpeamArbdeorpitee eldts - | T{ [ BaAyTp oSwao . . oacst p. tco a2drul wooc l ens

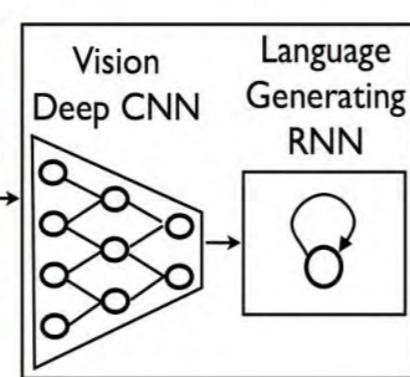
This highlighted neuron gets very excited (*green* = excited, *blue* = *not excited*) when the RNN is **inside the [[ ]]** markdown environment and turns off outside of it.

<http://www.ynetnews.com/>] English-language website of Israel's largest English-language site of Israelis singing songs. xne.waea.awatoa.s &ntiaca-sardeelh oan t bisanfanreif' aatd mw-2@piiisoesssis. /ern.c] (dceen epesaaiki ieel edh,irthraonse, cose dr.<ahb-nptwt.xi gh/ma) Tvdryzi couedl su:tha-oo tu,stuif lvepery stp,tcoa2drulwoclensr] p. li vaod, eytcc-n dm-oibuvv] bbimsulta tlybn

The RNN is likely using this neuron to remember if it is inside a URL or not.

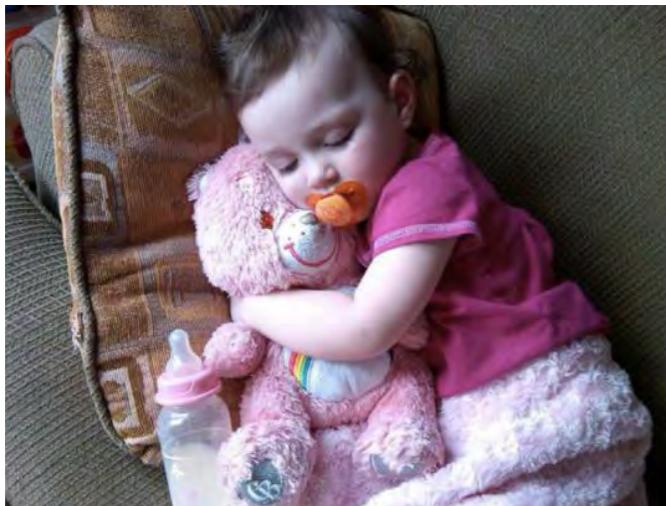
# Deep Learning - Basics

## Image Captioning – Combining CNN and RNN



**A group of people shopping at an outdoor market.**  
**There are many vegetables at the fruit stand.**

Neural Image Caption Generator **generates fitting natural-language captions only based on the pixels** by combining a vision CNN and a language-generating RNN.



A close up of a child holding a stuffed animal



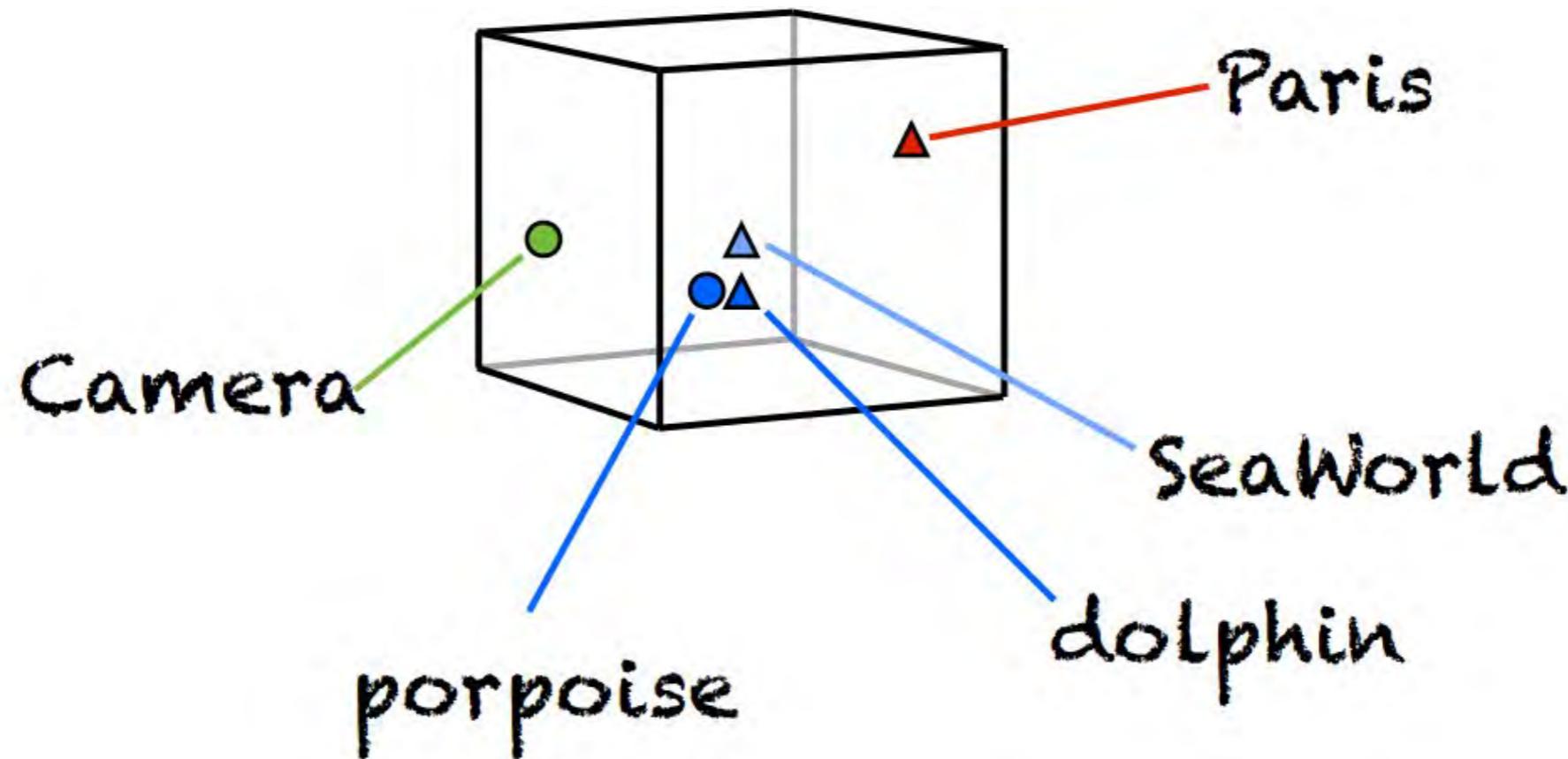
Two pizzas sitting on top of a stove top oven



A man flying through the air while riding a skateboard

# Deep Learning - Basics

## Natural Language Processing – Embeddings

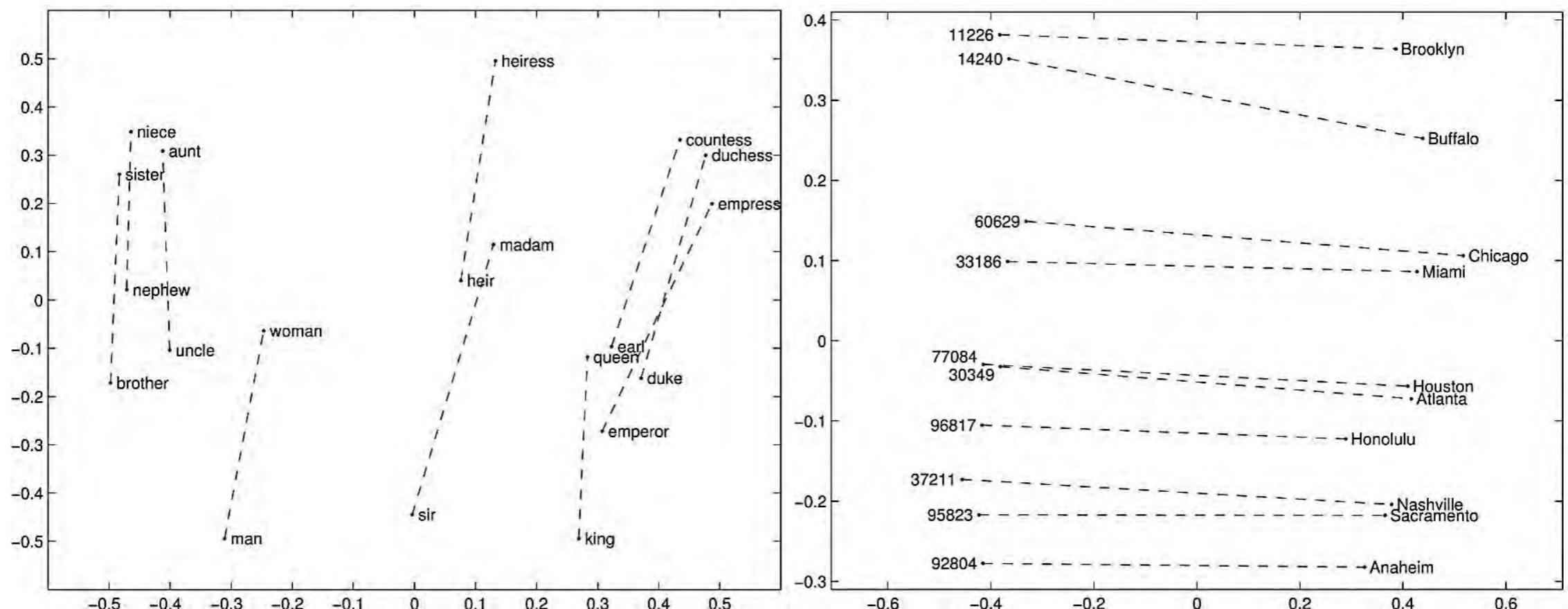


Embeddings are used to turn textual data (words, sentences, paragraphs) into high-dimensional vector representations and group them together with semantically similar data in a **vectorspace**. Thereby, **computer can detect similarities mathematically**.

# Deep Learning - Basics

## Natural Language Processing – Word2Vec

Word2Vec is an unsupervised learning algorithm for obtaining **vector representations for words**. These vectors were trained for a specific domain on a very large textual data set. GloVe is a better performing alternative.

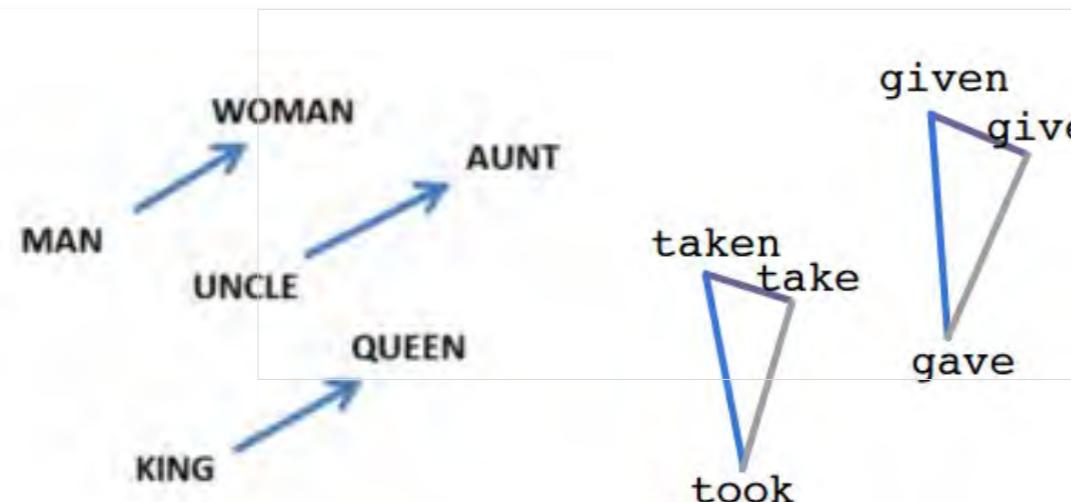


It detects similarities mathematically by grouping the vectors of similar words together.  
All it needs is **words co-occurrence** in the given corpus.

# Deep Learning - Basics

## Natural Language Processing – Word2Vec

| FRANCE      | JESUS   | XBOX        | REDDISH   | SCRATCHED | MEGABITS   |
|-------------|---------|-------------|-----------|-----------|------------|
| AUSTRIA     | GOD     | AMIGA       | GREENISH  | NAILED    | OCTETS     |
| BELGIUM     | SATI    | PLAYSTATION | BLUISH    | SMASHED   | MB/S       |
| GERMANY     | CHRIST  | MSX         | PINKISH   | PUNCHED   | BIT/S      |
| ITALY       | SATAN   | IPOD        | PURPLISH  | POPPED    | BAUD       |
| GREECE      | KALI    | SEGA        | BROWNISH  | CRIMPED   | CARATS     |
| SWEDEN      | INDRA   | PSNUMBER    | GREYISH   | SCRAPED   | KBIT/S     |
| NORWAY      | VISHNU  | HD          | GRAYISH   | SCREWED   | MEGAHERTZ  |
| EUROPE      | ANANDA  | DREAMCAST   | WHITISH   | SECTIONED | MEGAPIXELS |
| HUNGARY     | PARVATI | GEFORCE     | SILVERY   | SLASHED   | GBIT/S     |
| SWITZERLAND | GRACE   | CAPCOM      | YELLOWISH | RIPPED    | AMPERES    |



Woman – Man ≈ Aunt - Uncle  
King - Male + Female ≈ Queen  
Human - Animal ≈ Ethics

# Deep Learning - Basics

## Natural Language Processing – Thought Vectors

Thought vectors is a way of **embedding thoughts in vector space**. Their features will represent **how each thought relates to other thoughts**.

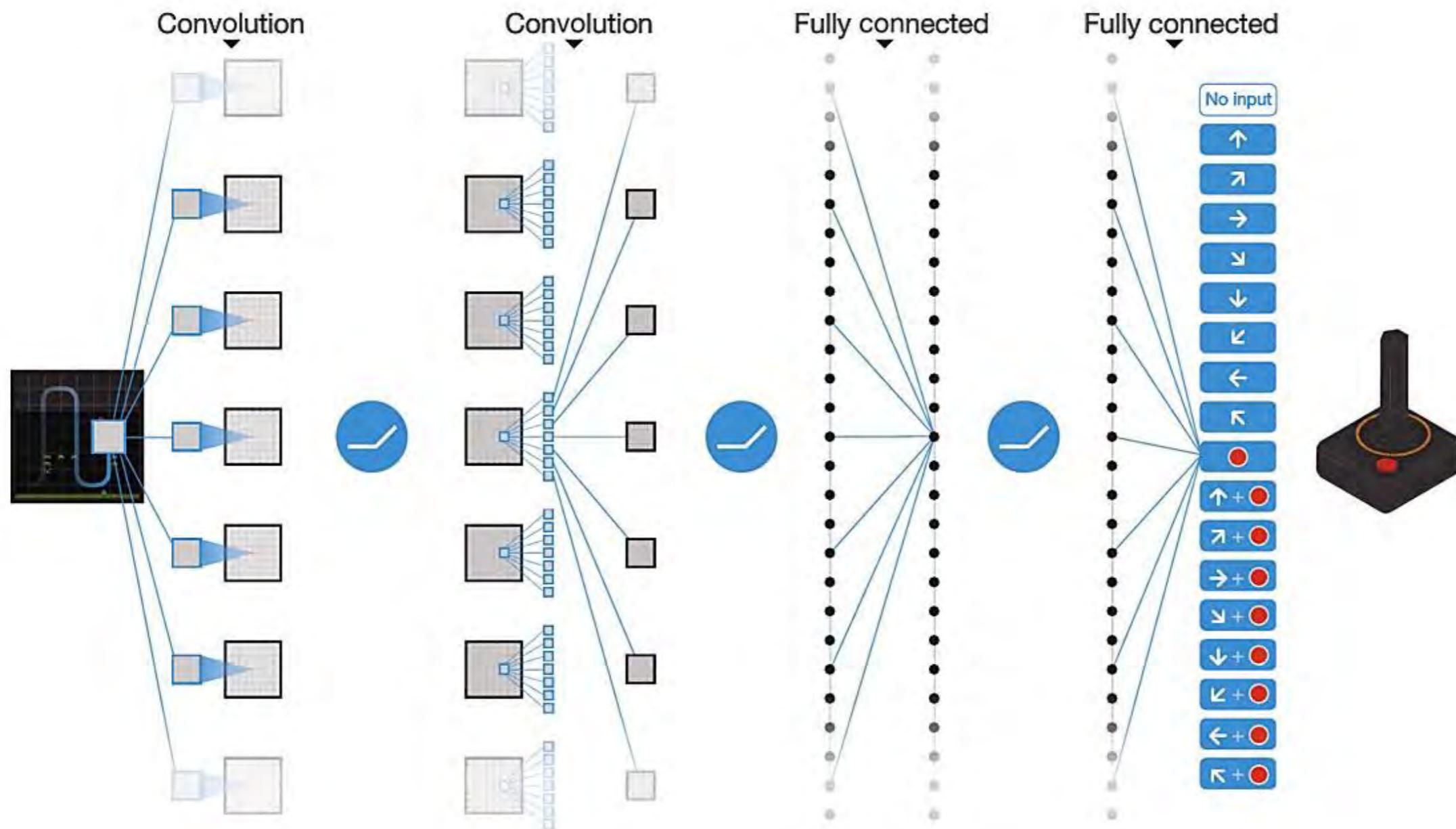
By reading every document on the web, computers might be able to reason like humans do by mimicking the thoughts expressed in content.



A neural machine translation is trained on bilingual text using a encoder and decoder RNN. For translation, the **input sentence is transformed into a thought vector**. This vector is used to **reconstruct the given thought** in another language.

# Deep Learning - Basics

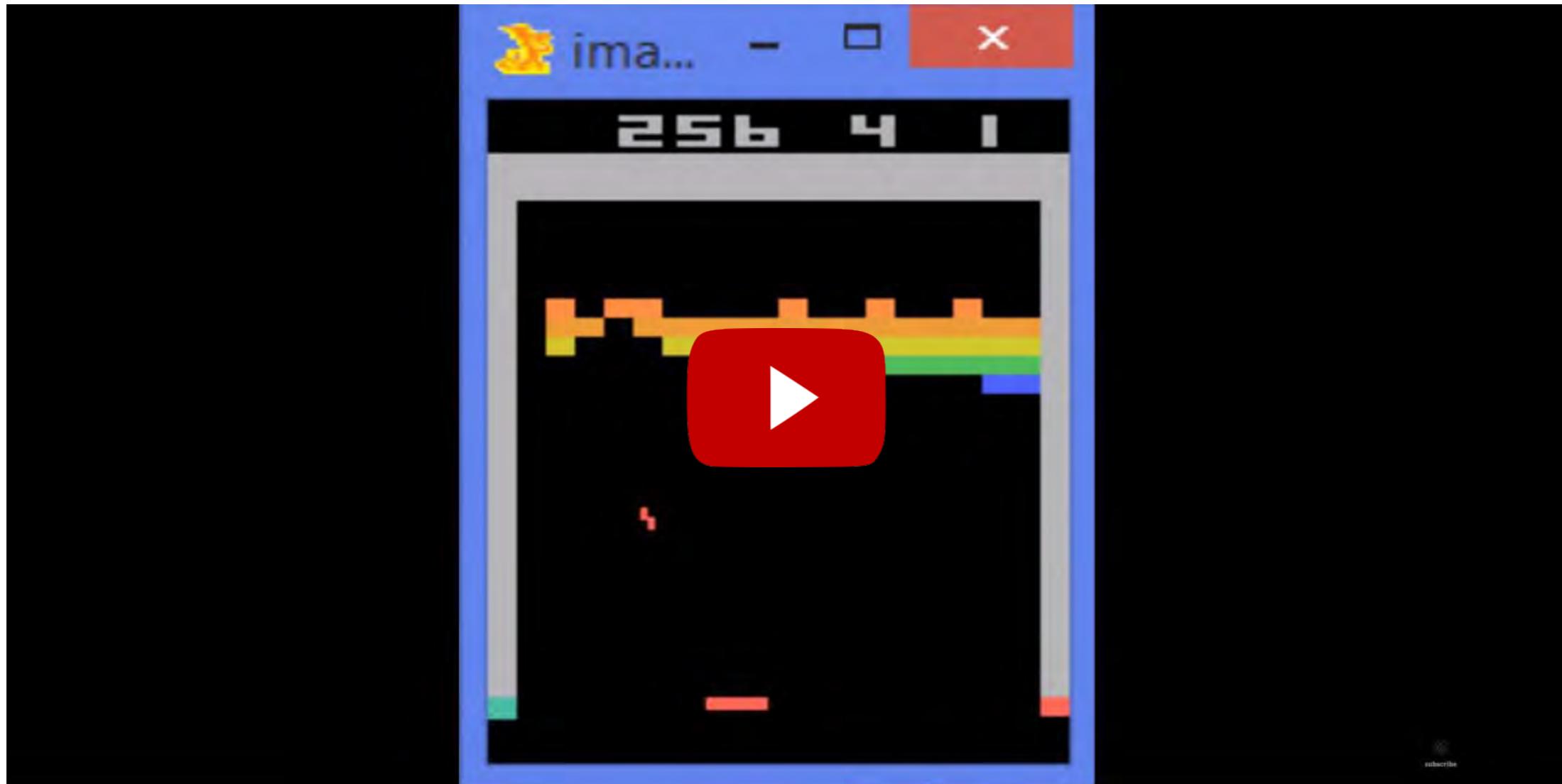
DeepMind Deep Q-Learning



Deep Q-Learning (DQN) is a model-free approach to reinforcement learning using deep networks in environments with discrete action choices

# DeepLearning - Basics

DeepMind Deep Q-Learning



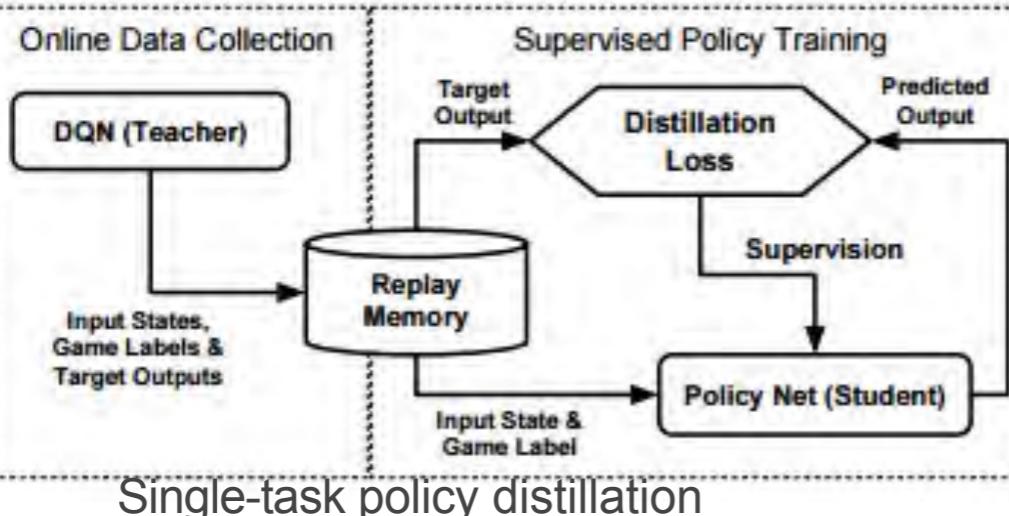
Outperforms humans in over 30 Atari games just by receiving the pixels on the screen with the goal to maximize the score (Reinforcement Learning)

# Deep Learning - Basics

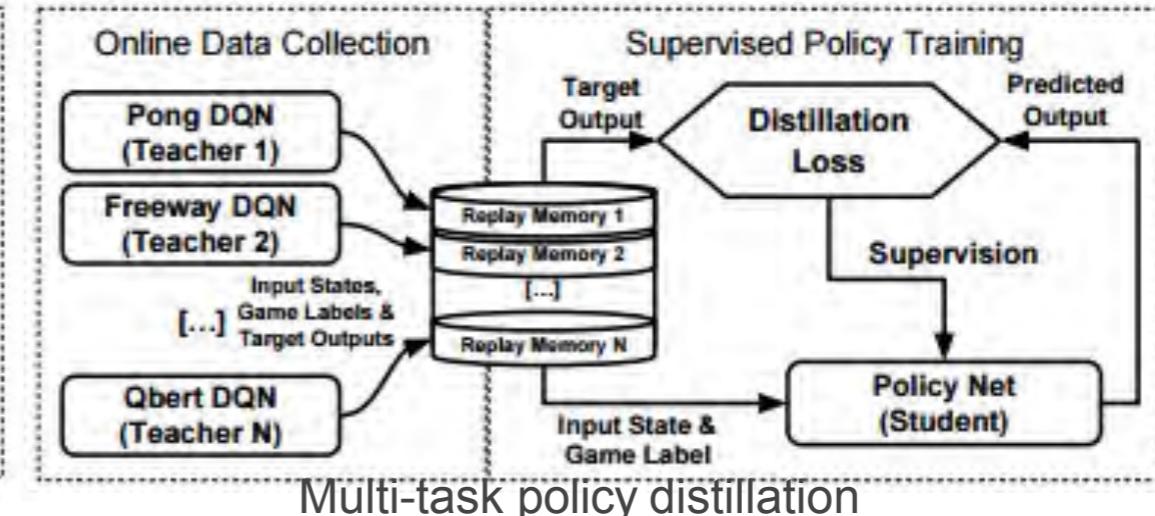
DeepMind Deep Q-Learning



Policy distillation: Extracts the learned state (*policy*) of a reinforcement learning agent (*teacher*) and **trains a new network (*student*)** that performs at the expert level while being dramatically smaller and more efficient.



Single-task policy distillation



Multi-task policy distillation

# Deep Learning - Basics

## Usage Requirements



Large data set with good quality (*input-output mappings*)



Measurable and describable goals (*define the cost*)



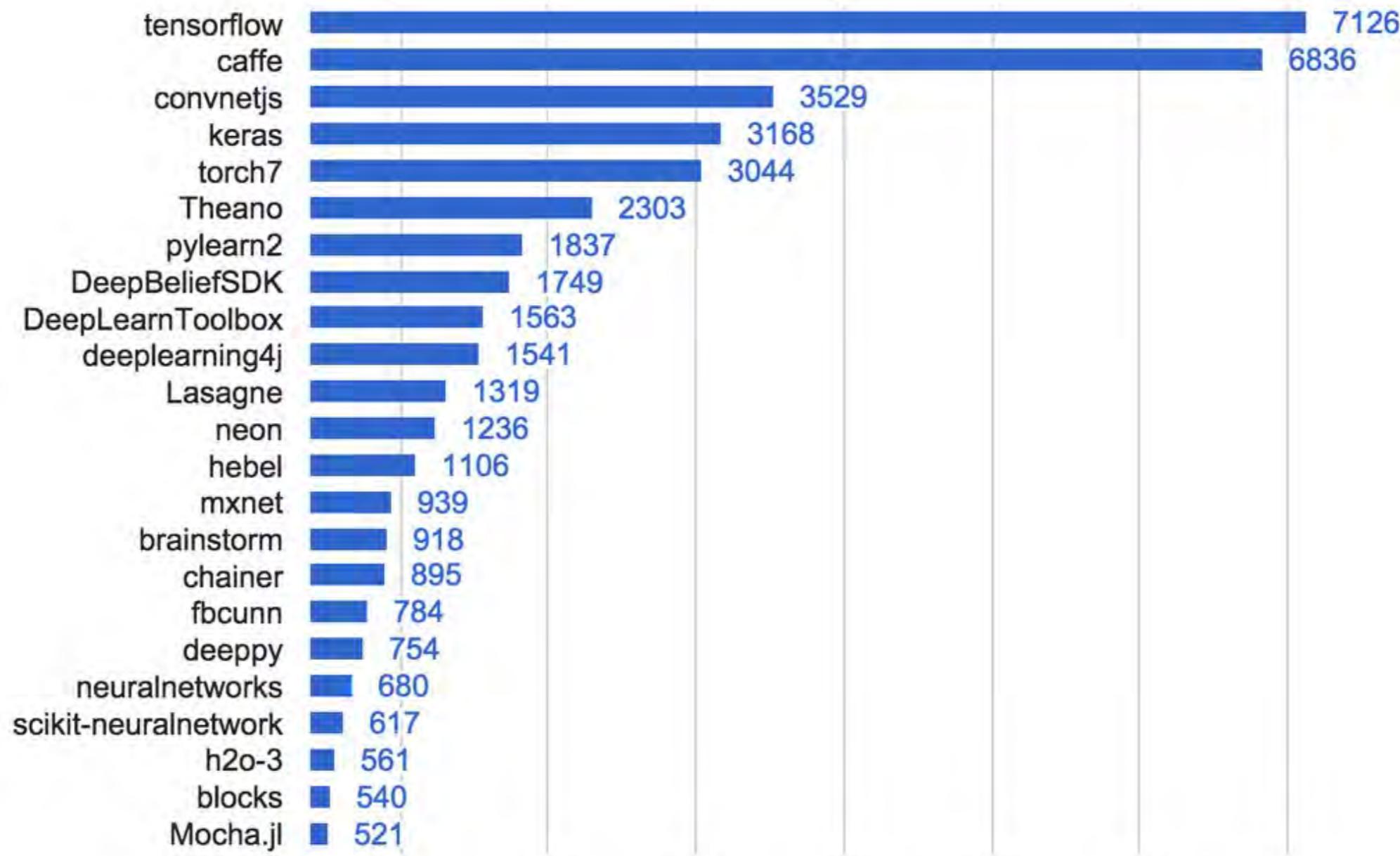
Enough computing power (*AWS GPU Instance*)



Excels in tasks where the basic unit (*pixel, word*) has very little meaning in itself, but the **combination of such units has a useful meaning**

# DeepLearning - Tools

Its all OpenSource



# Deep Learning - Tools

Computing is affordable



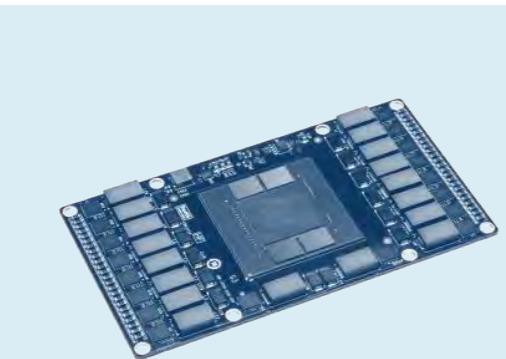
AWS EC2 GPU Spot Instance: *g2.2xlarge - \$0.0782 perHour*



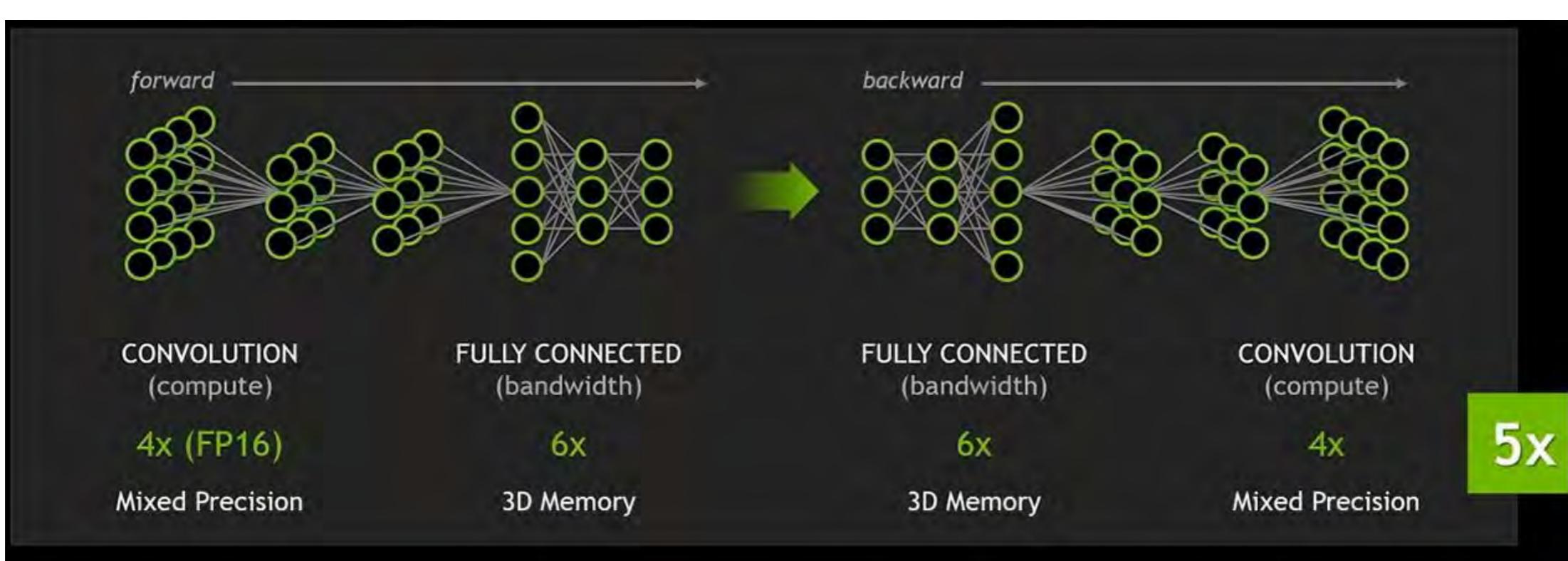
The DIGITS DevBox combines the world's best hardware (4 GPUs), software, and systems engineering for deep learning in a powerful solution that can fit under your desk. Cost: \$15k

# Outlook

## NVIDIA Pascal



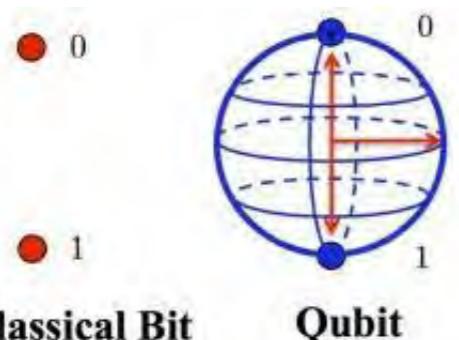
NVIDIA's Pascal GPU architecture will **accelerate deep learning applications** up to 10X beyond the speed of its current-generation Maxwell processors.



# Outlook

## Artificial Quantum Intelligence

Quantum Artificial Intelligence Lab is a joint initiative of NASA and Google to study how **quantum computing might advance machine learning**. This type of computing may provide the most creative and parallelized problem-solving process under the known laws of physics.



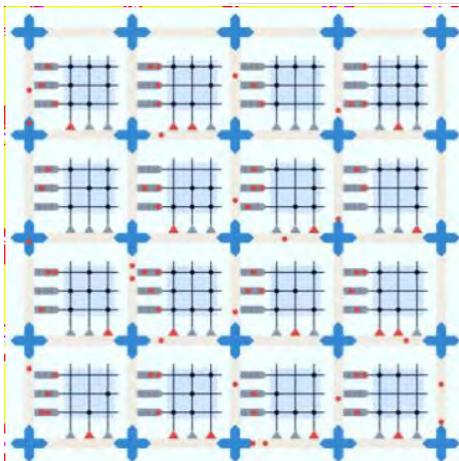
Quantum computers handle what are called quantum bits or qubits that can readily have a **value of one or zero or anything in between**.



Quantum computing represents a paradigm shift, a radical change in the way we do computing and at a scale that has been unimaginable before – *Eric Ladizinsky (Co-founder D-Wave)*

# Outlook

## Neuromorphic Chips



IBM TrueNorth is a **brain-inspired computer chip** that implements networks of integrate-and-fire spiking artificial neurons and uses only a tiny 70 mw of power –**orders of magnitude less energy** than traditional chips. The system is designed to be able to run deep-learning algorithms.

**Traditional computers** focus on language and analytical thinking  
(Left brain)



**Neurosynaptic chips** address the senses and pattern recognition  
(Right brain)



Over the coming years, IBM scientists hope to meld the two capabilities together to create a **holistic computing intelligence**



1 million  
Programmable  
Neurons



256 million  
Programmable  
Synapses



4096  
Neurosynaptic  
Cores

# Outlook

## Deep Learning



Significant advances in **deep reinforcement and unsupervised learning**



Bigger and **more complex architectures** based on various interchangeable modules/techniques



**Deeper models** that can learn from much fewer training cases



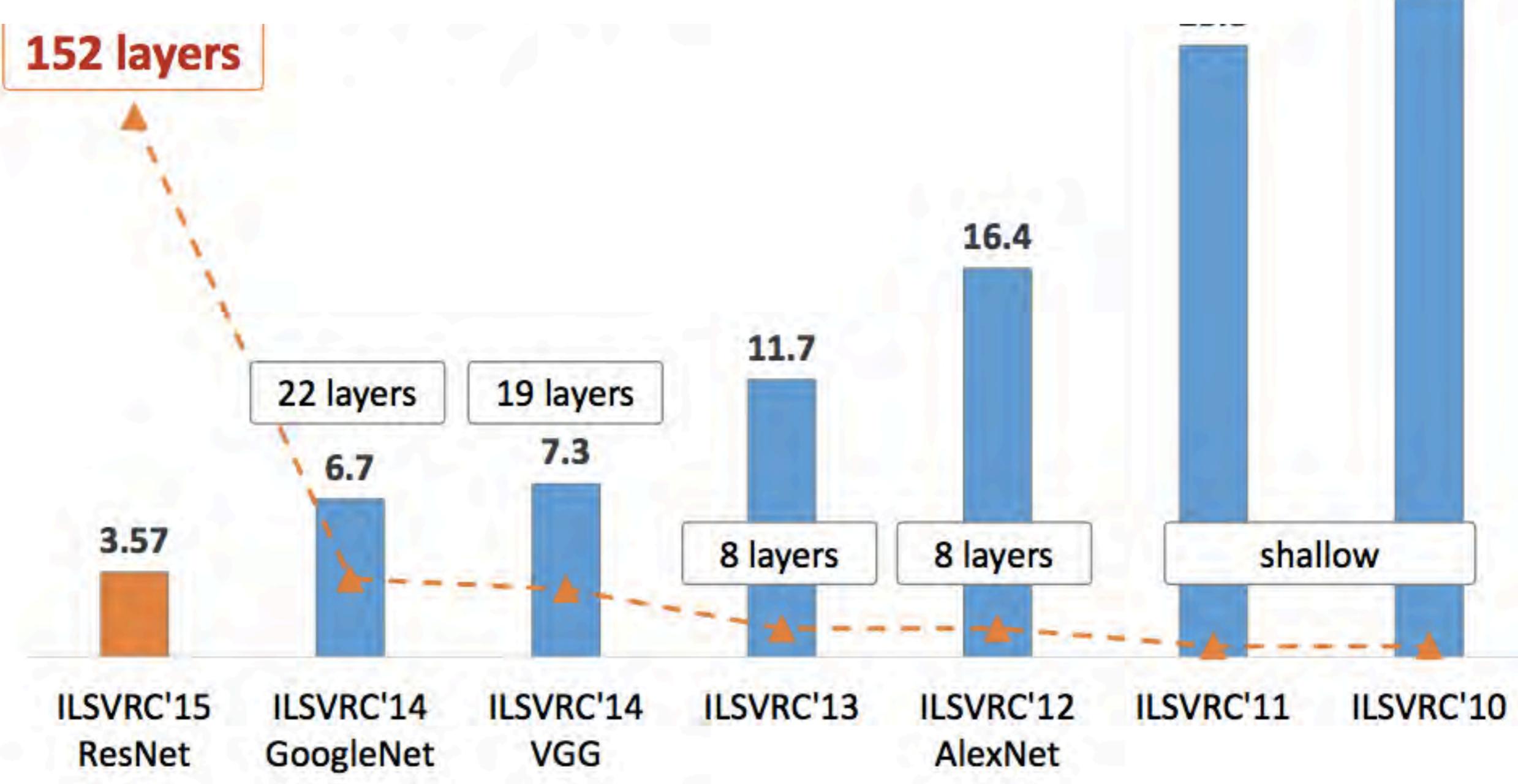
Harder problems such as **video understanding** and **natural language processing** will be successfully tackled by deep learning algorithms



# Convolutional Neural Net

# What happened to my field?

Classification: ImageNet Challenge top-5 error



# What happened to my field?

## Object Detection: PASCAL VOC mean Average Precision (mAP)

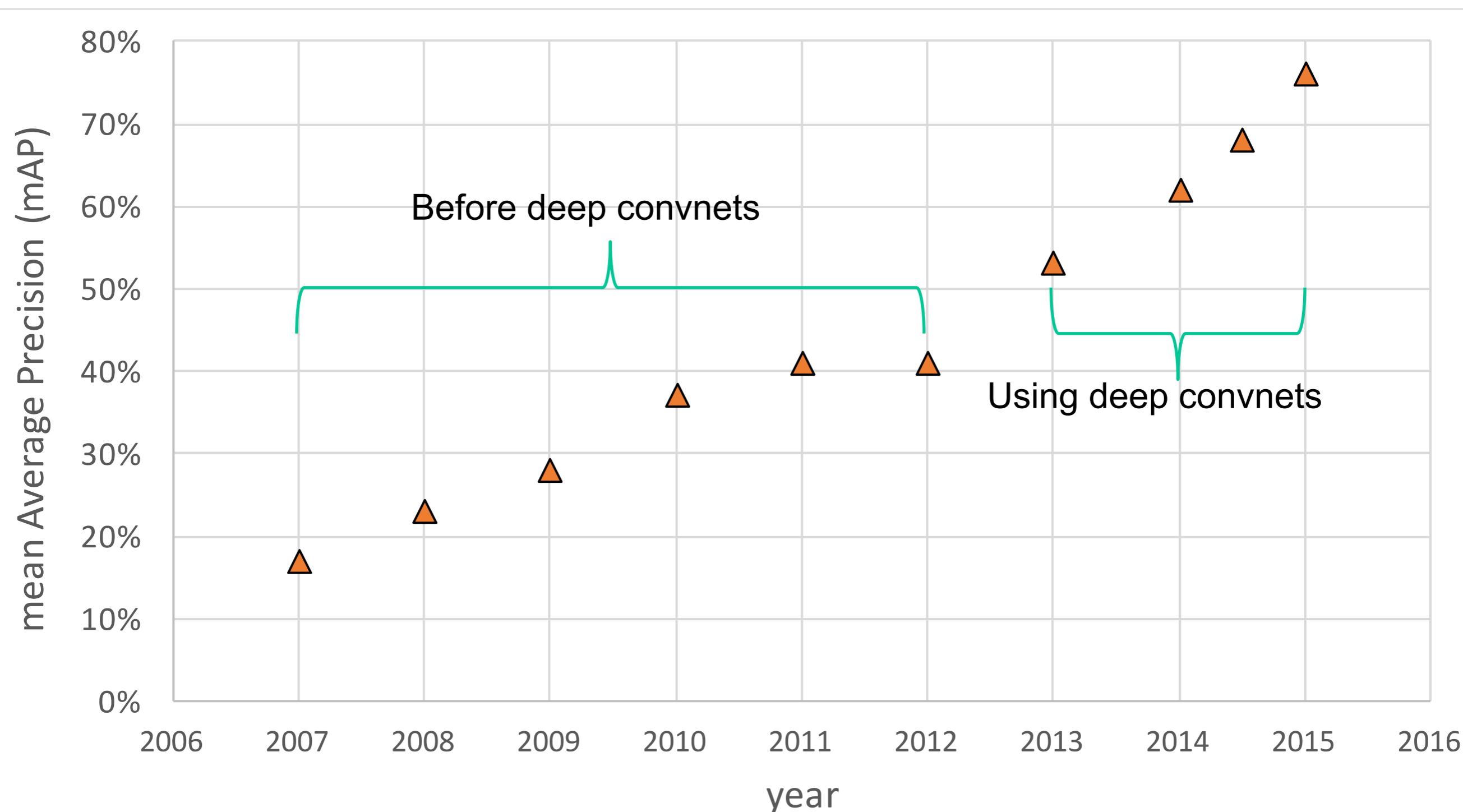
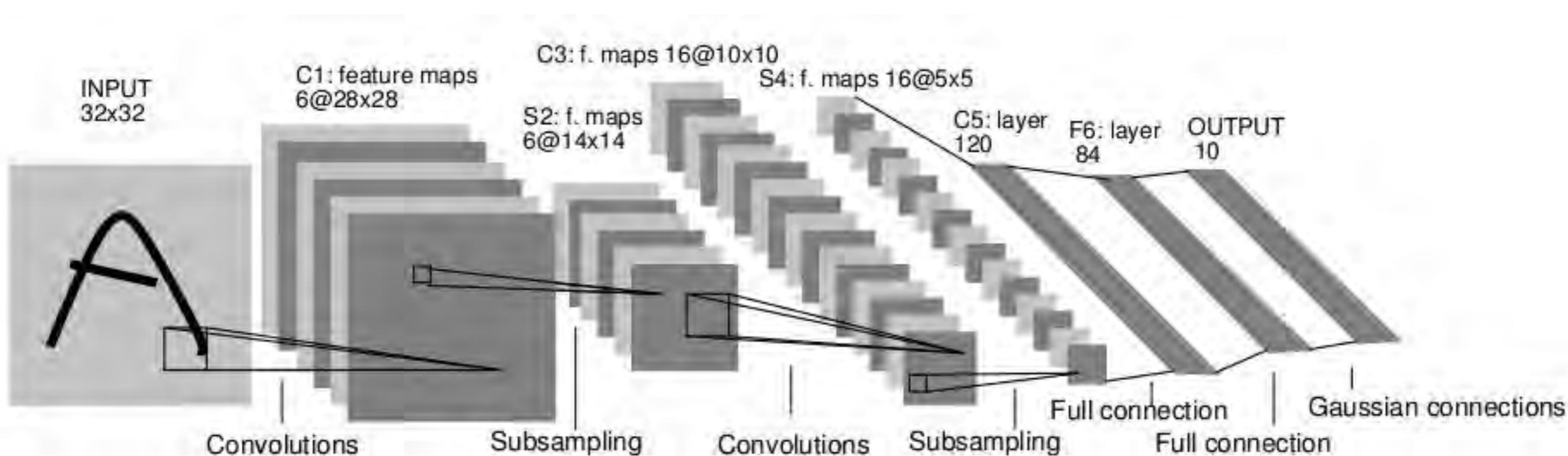


Figure source: Ross Girshick

# Actually, it happened a while ago...

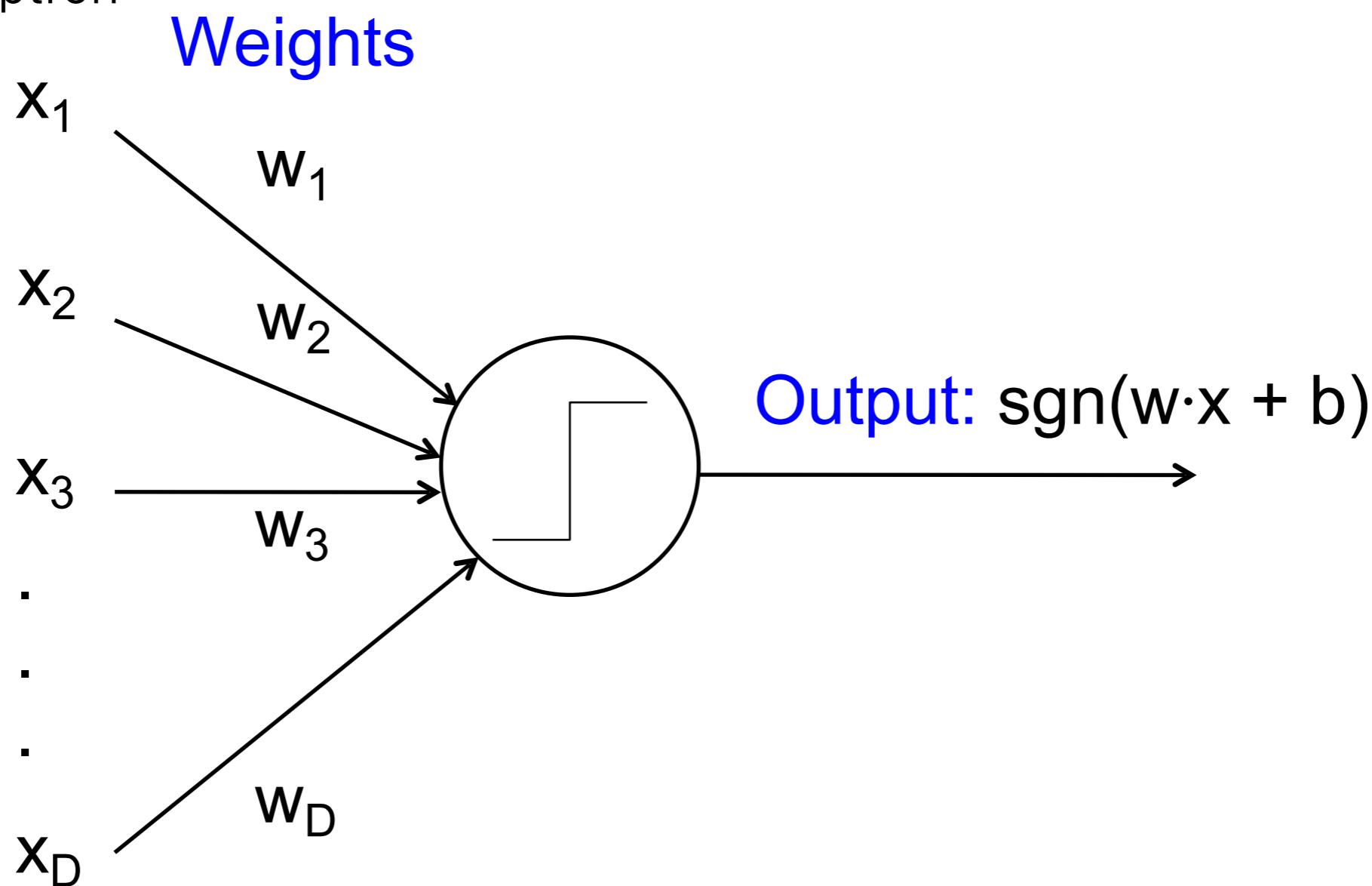
LeNet 5



# Let's back up even more...

Input

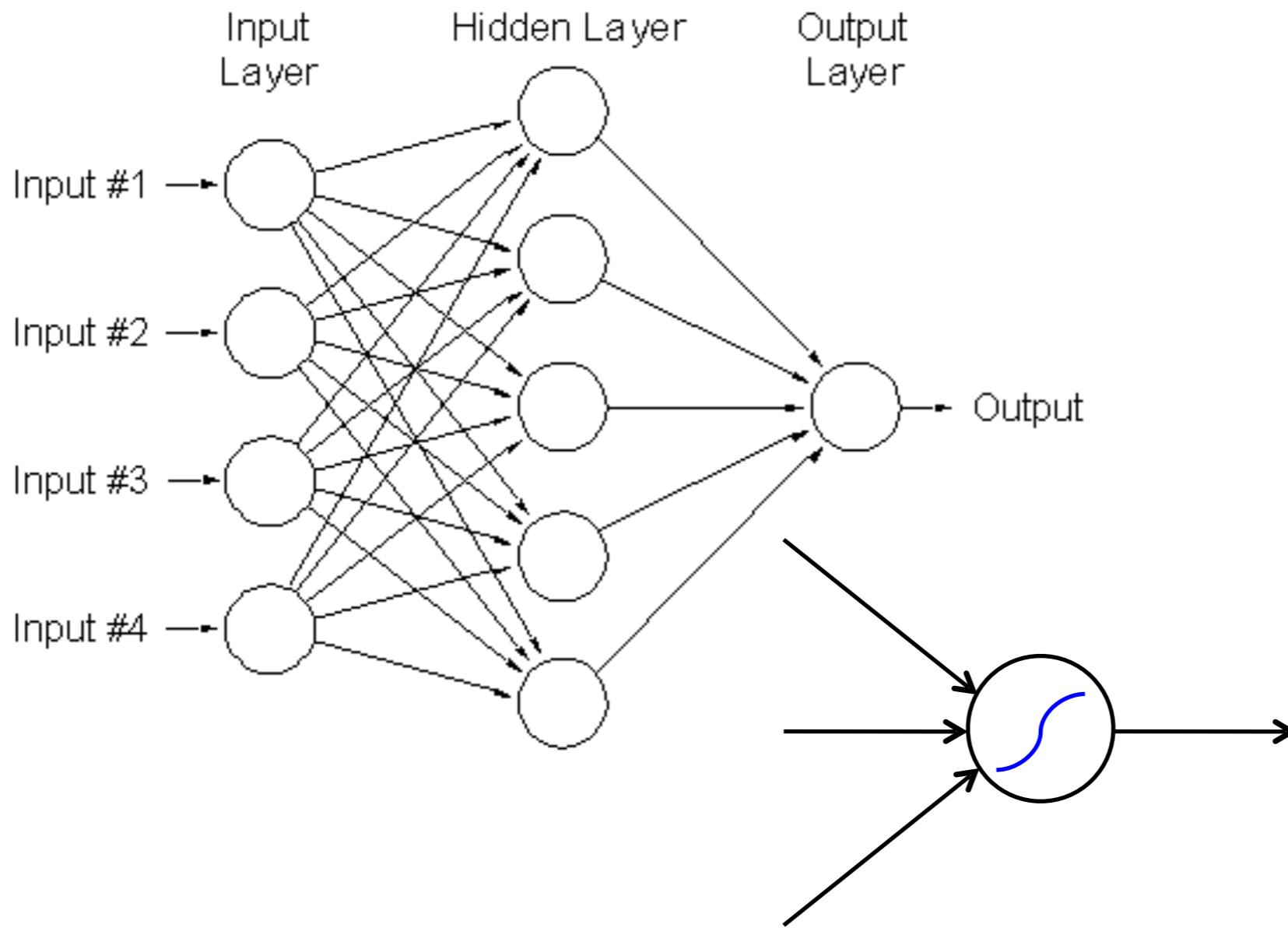
The Perceptron



Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,  
Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408.

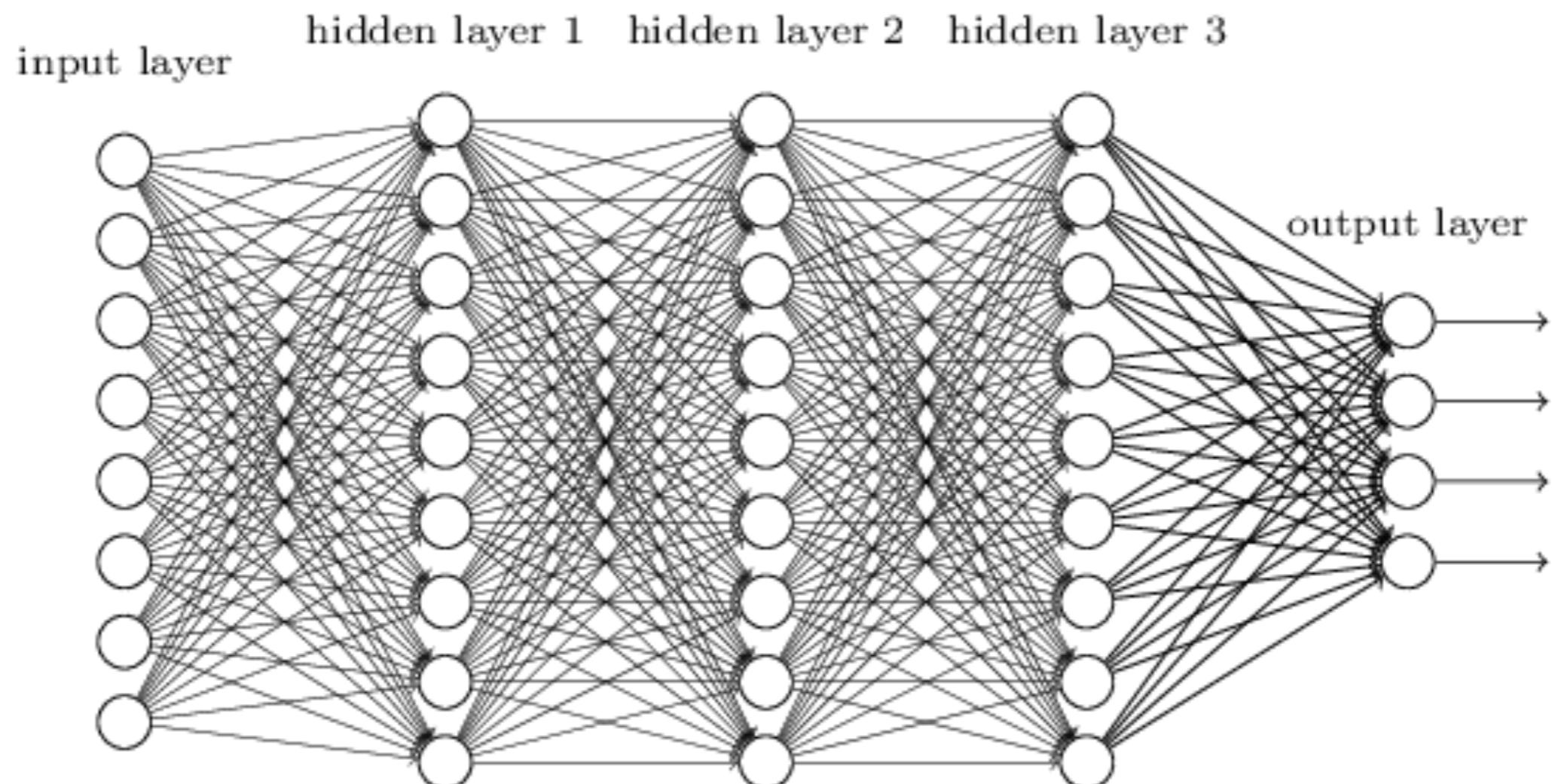
# Two-layer neural network

- Can learn nonlinear functions provided each perceptron has a differentiable nonlinearity



**Sigmoid:** 
$$g(t) = \frac{1}{1 + e^{-t}}$$

# Multi-layer neural network

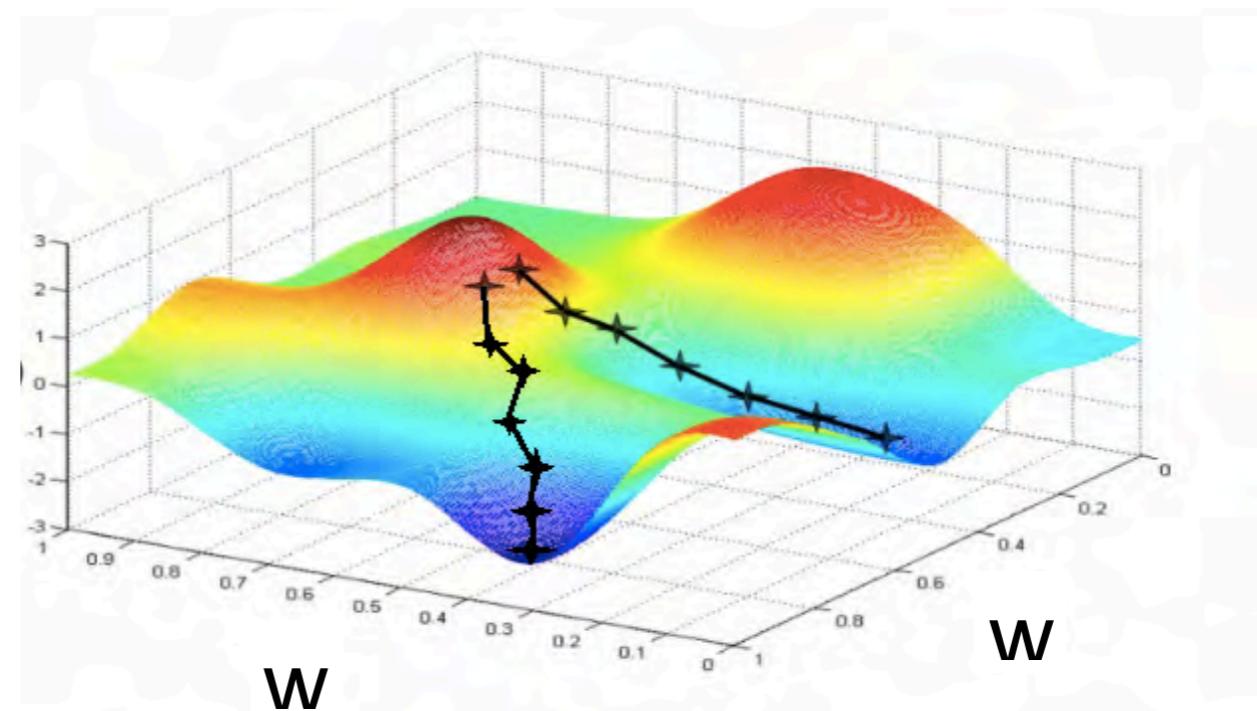


# Training of multi-layer networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$E(\mathbf{w}) = \sum_{i=1}^N (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 \quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

- Update weights by **gradient descent**:



# Training of multi-layer networks

- Find network weights to minimize the *training error* between true and estimated labels of training examples, e.g.:

$$E(\mathbf{w}) = \sum_{i=1}^N (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2 \quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

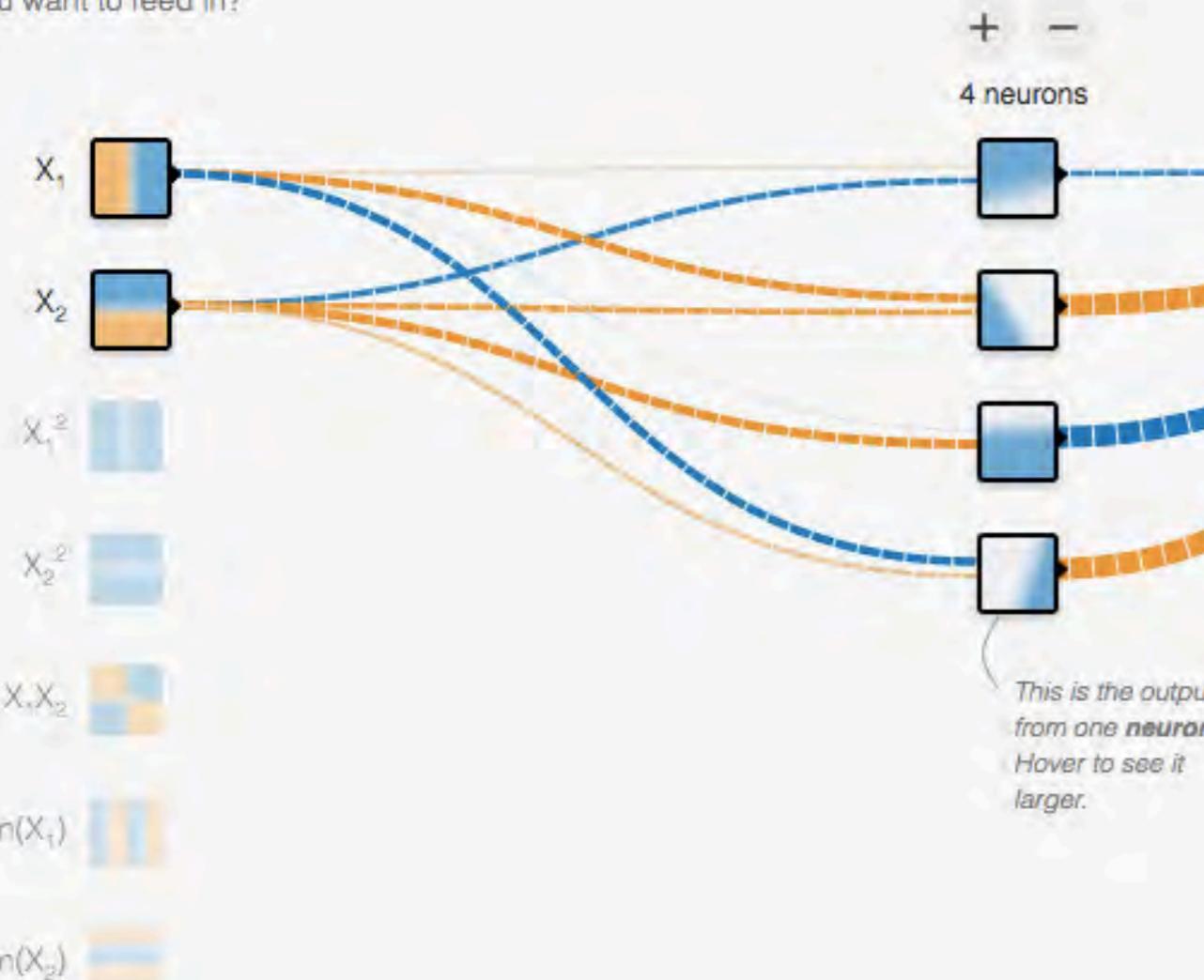
- Update weights by **gradient descent**:
- **Back-propagation**: gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent**: compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

# Multi-Layer Network Demo

INPUT

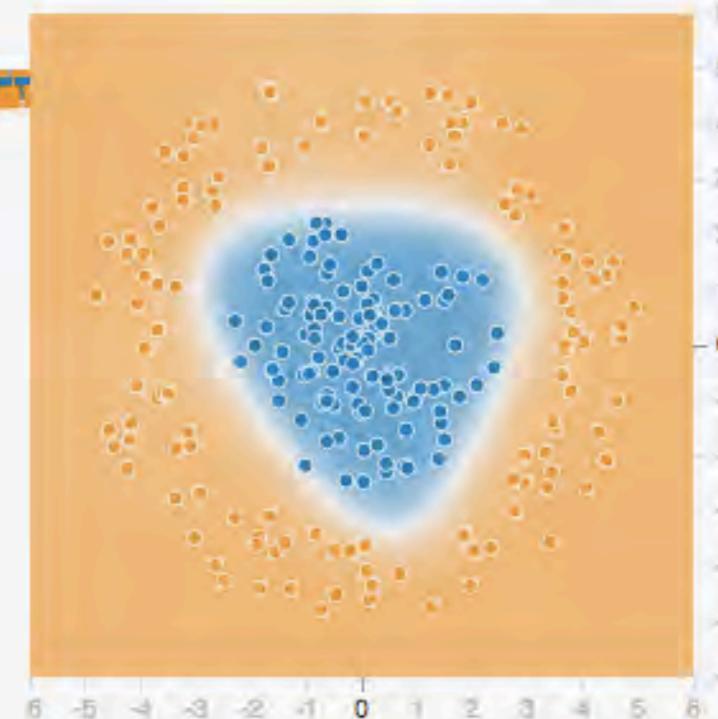
Which properties do you want to feed in?

+ - 1 HIDDEN LAYER



OUTPUT

Test loss 0.020  
Training loss 0.013



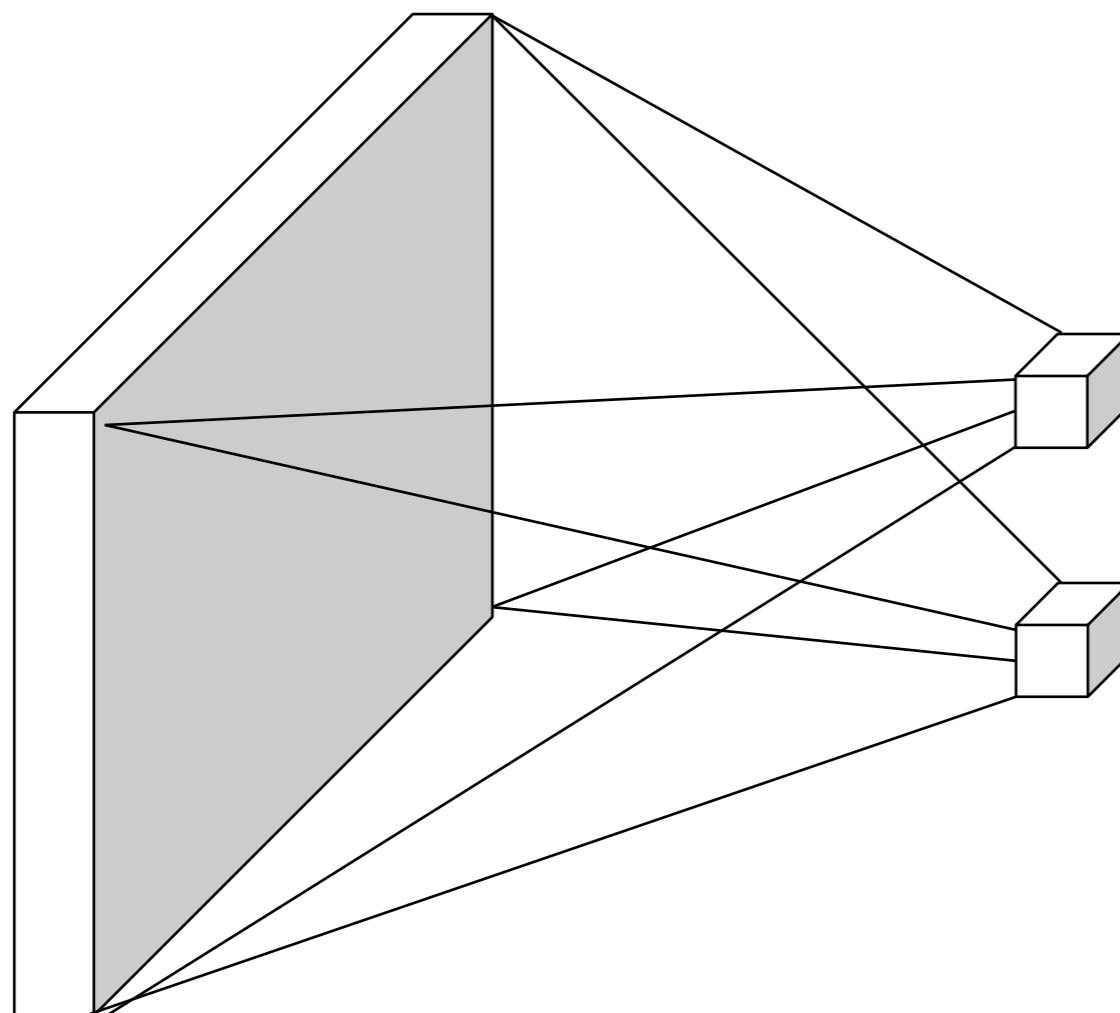
Colors shows data, neuron and weight values.

Show test data

Discretize output

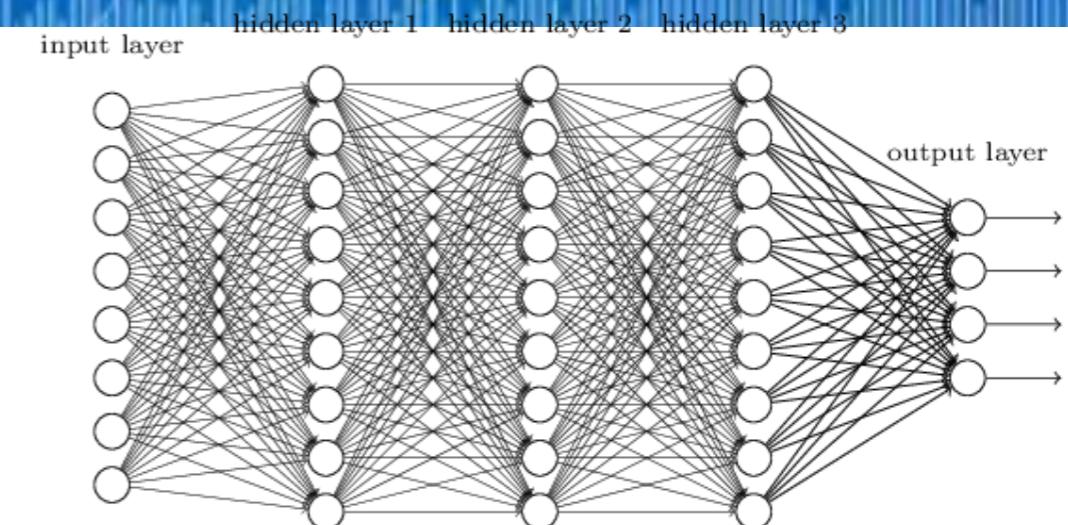
<http://playground.tensorflow.org/>

# From fully connected to convolutional networks

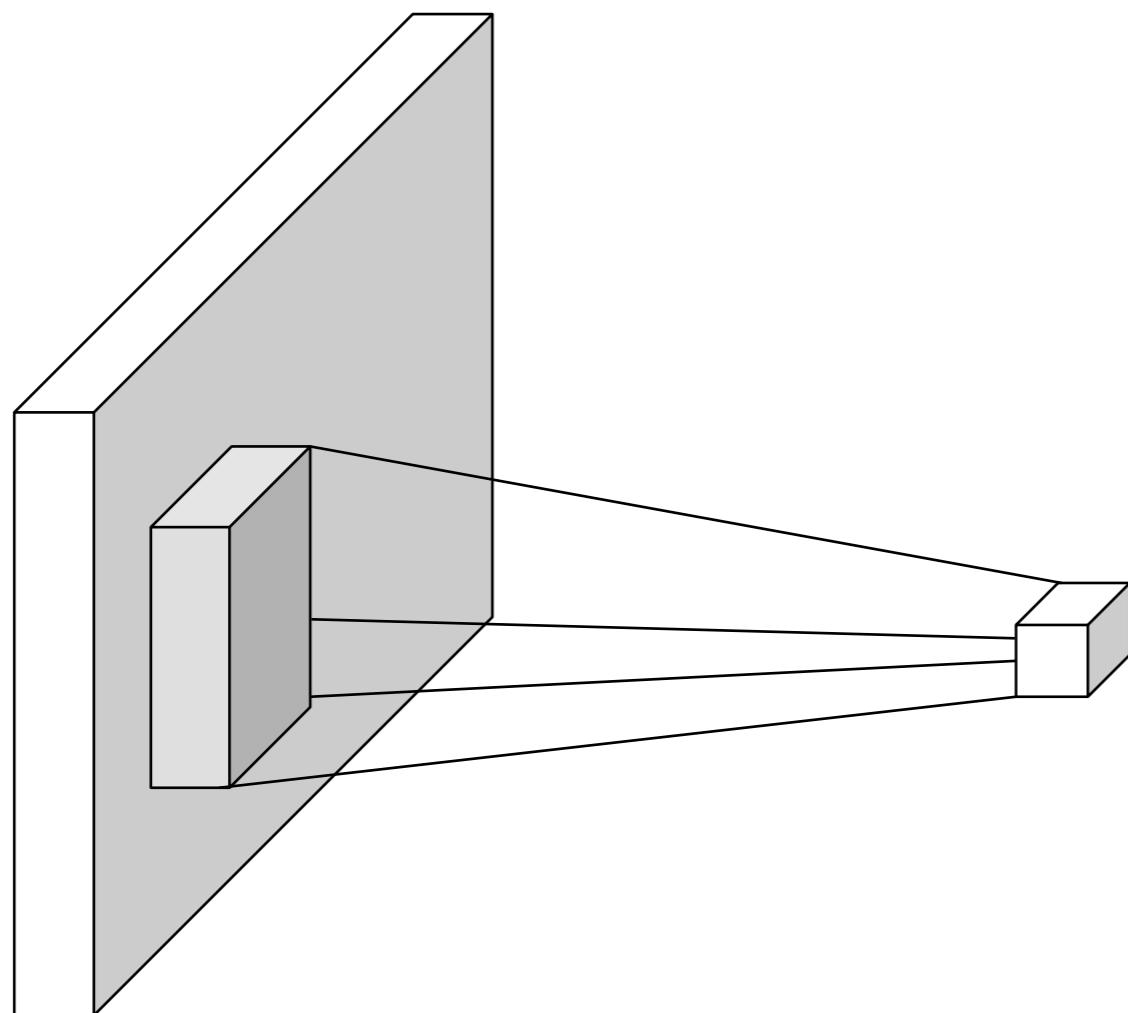


image

Fully connected layer



# From fully connected to convolutional networks

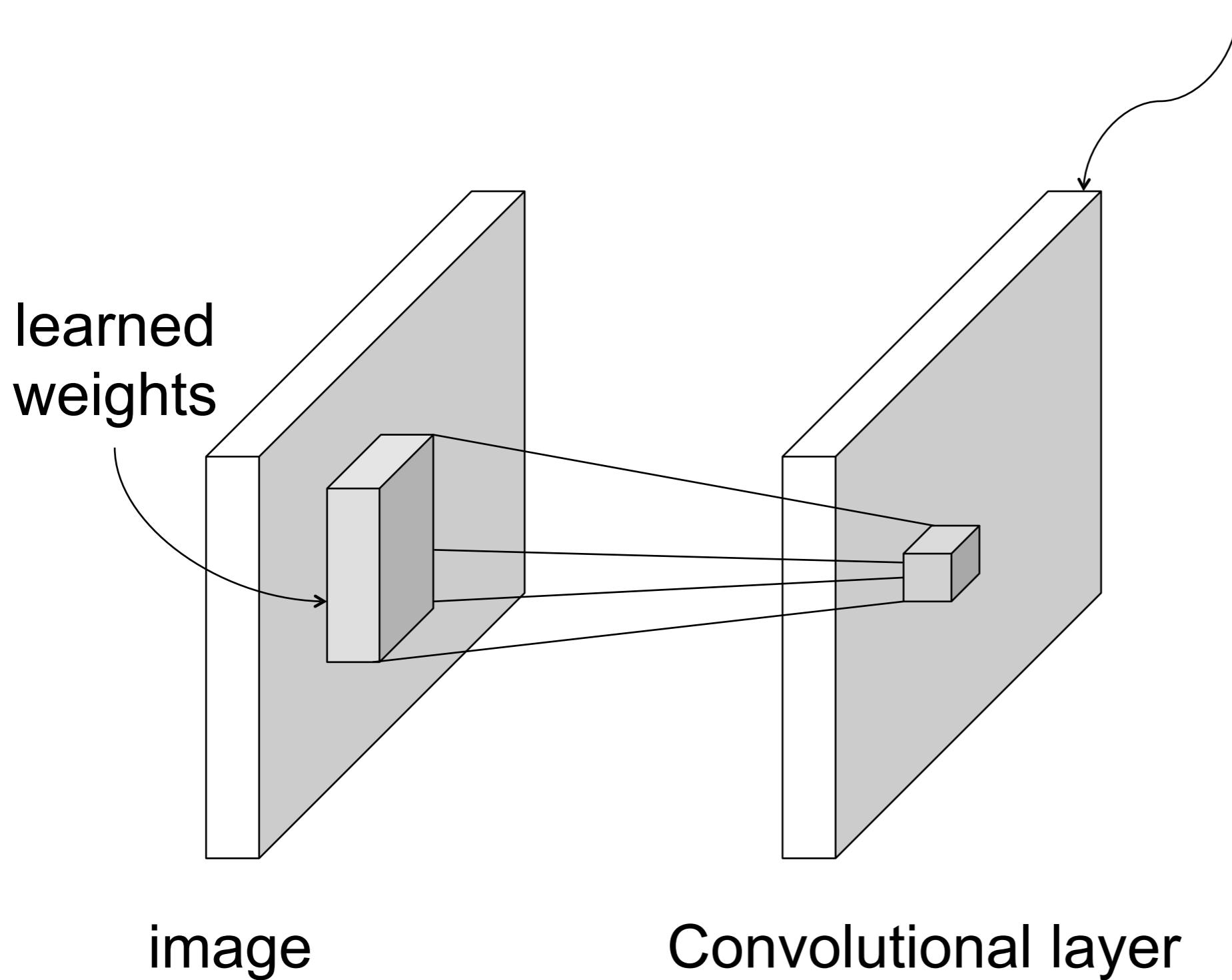


image

Convolutional layer

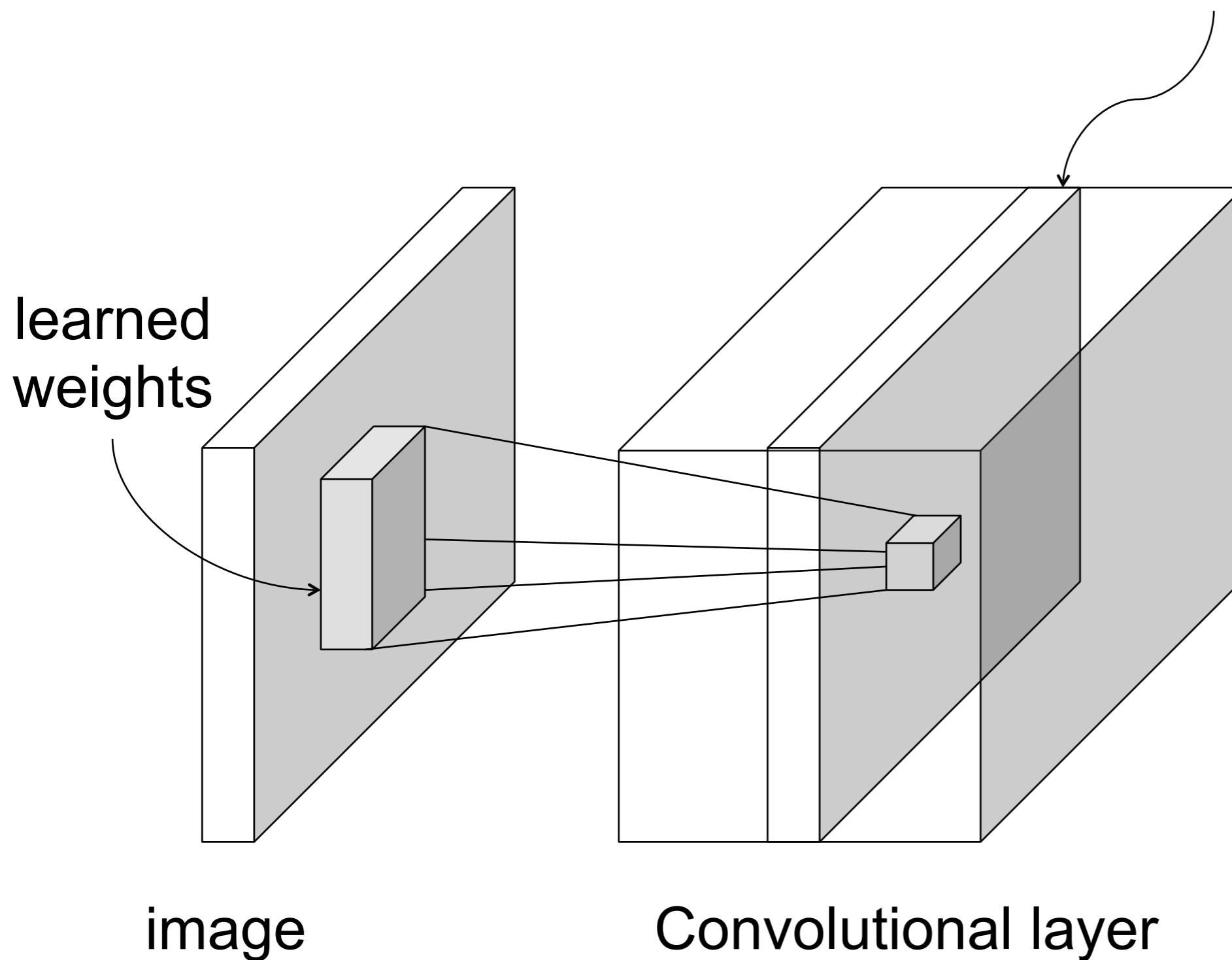
# From fully connected to convolutional networks

feature map

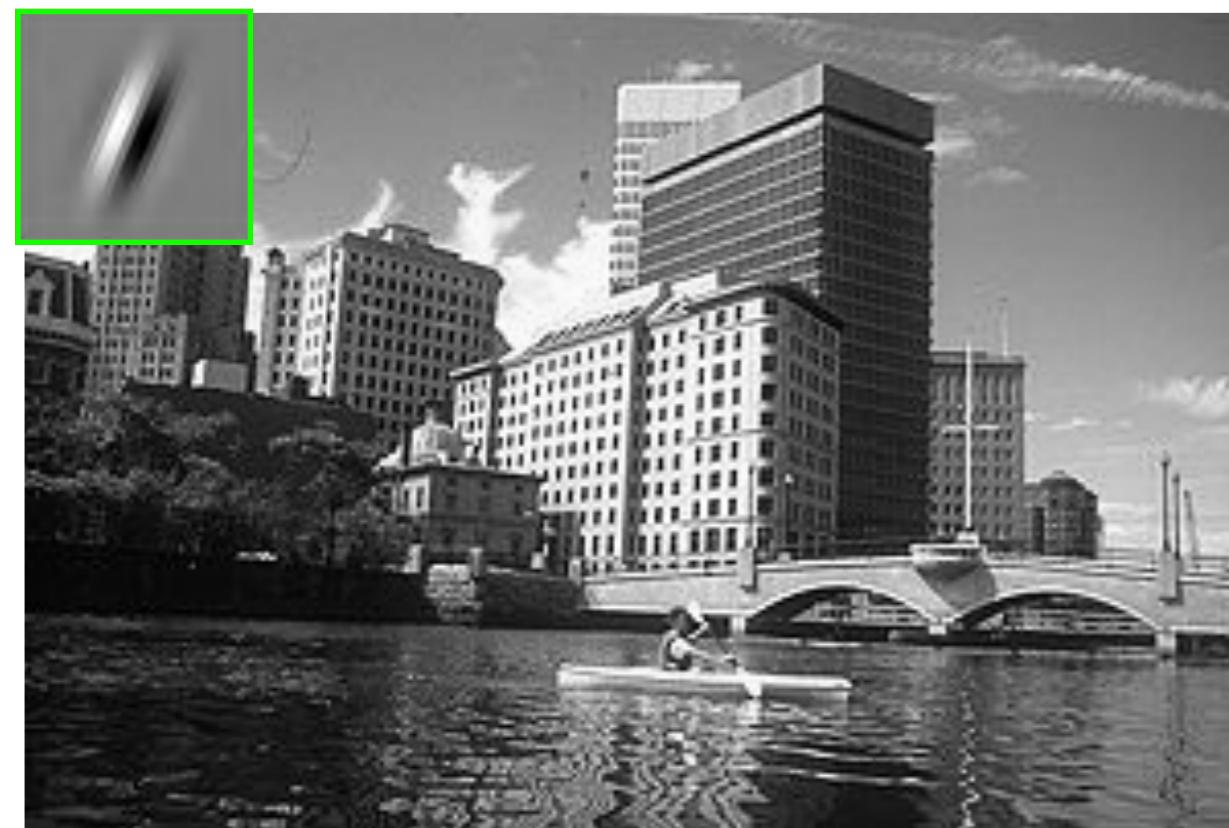


# From fully connected to convolutional networks

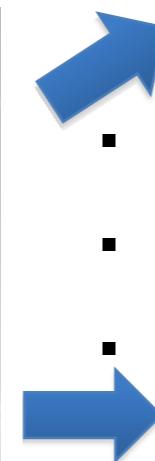
feature map



# Convolution as feature extraction



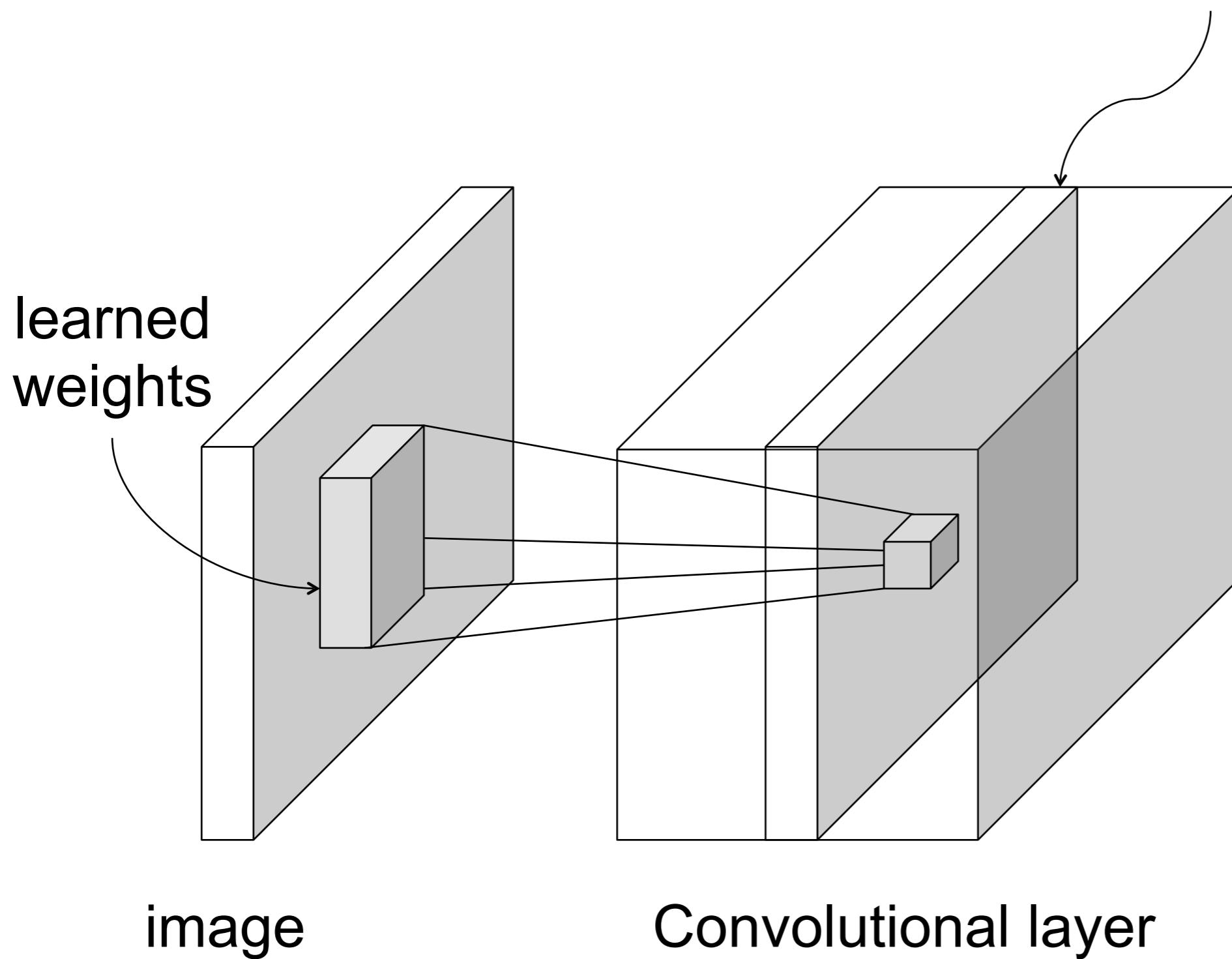
Input



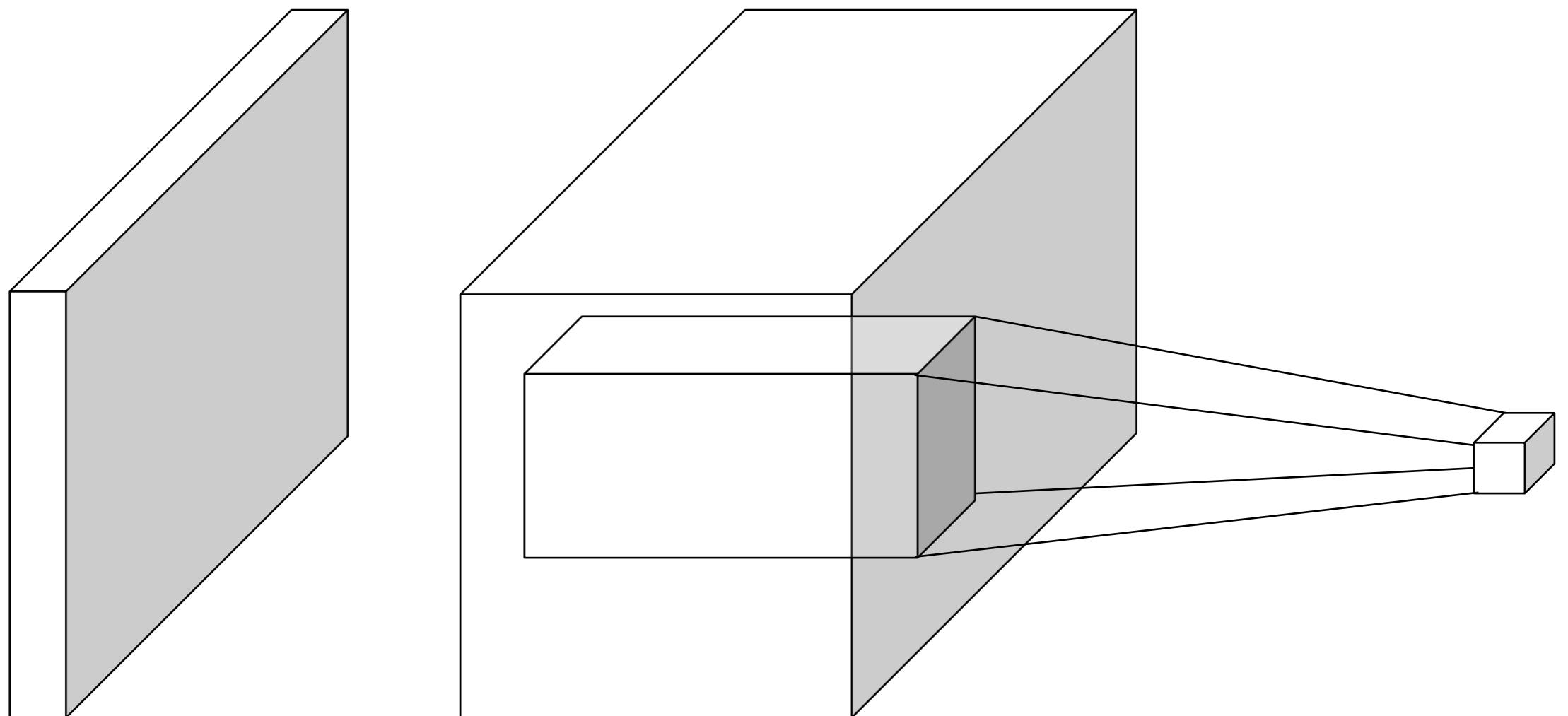
Feature Map

# From fully connected to convolutional networks

feature map



# From fully connected to convolutional networks

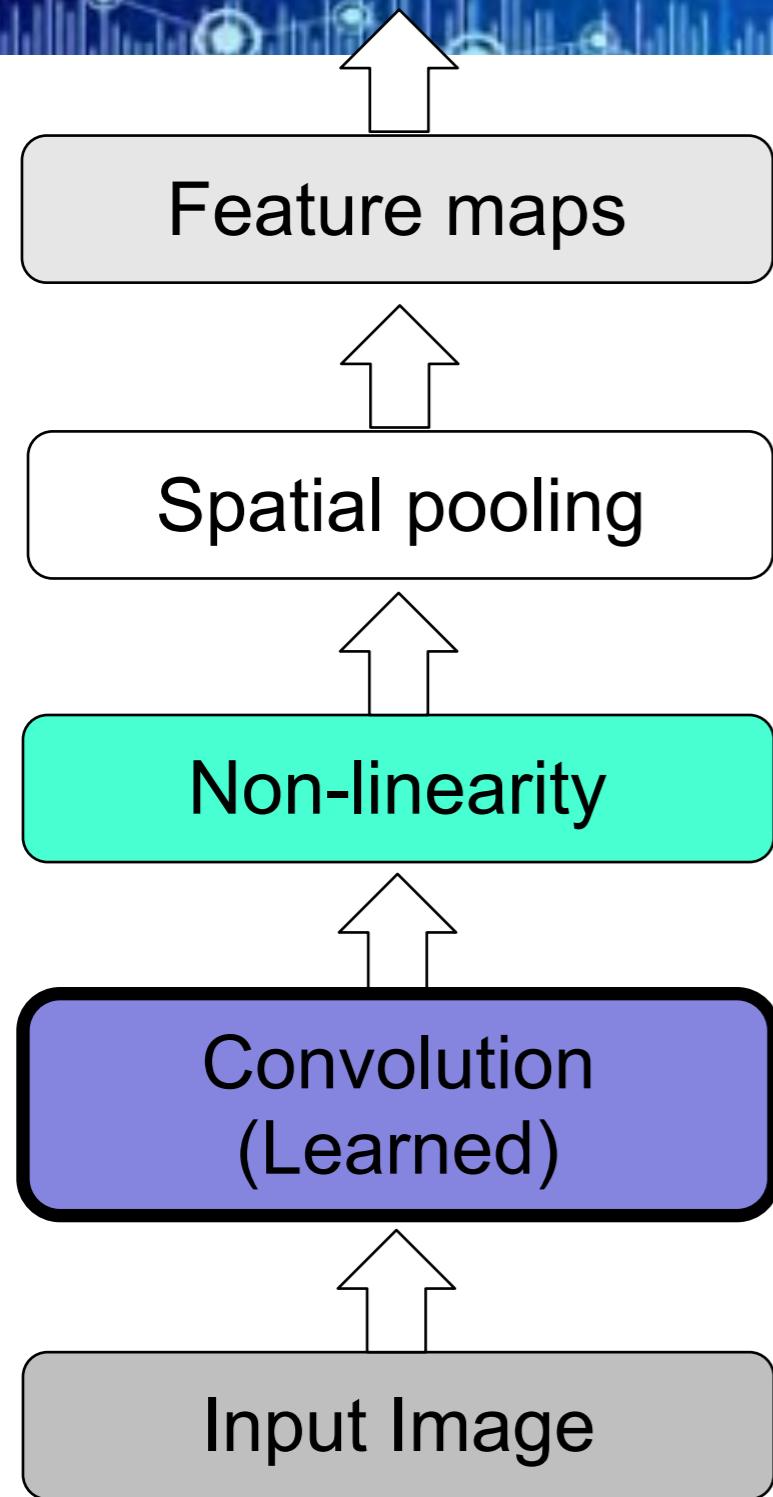


image

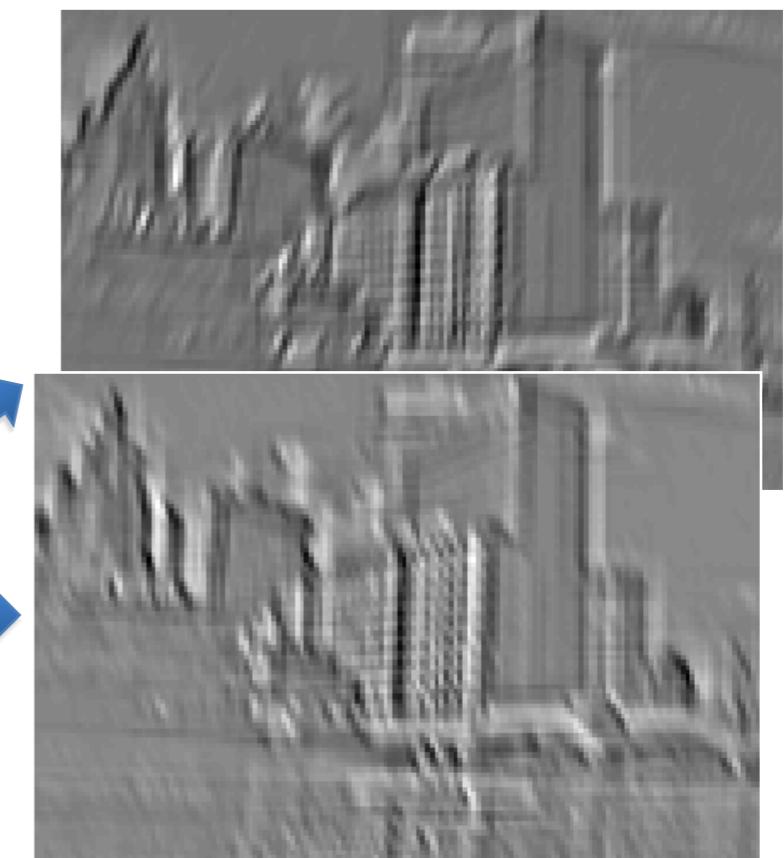
Convolutional layer

next  
layer  
694

# Key operations in a CNN



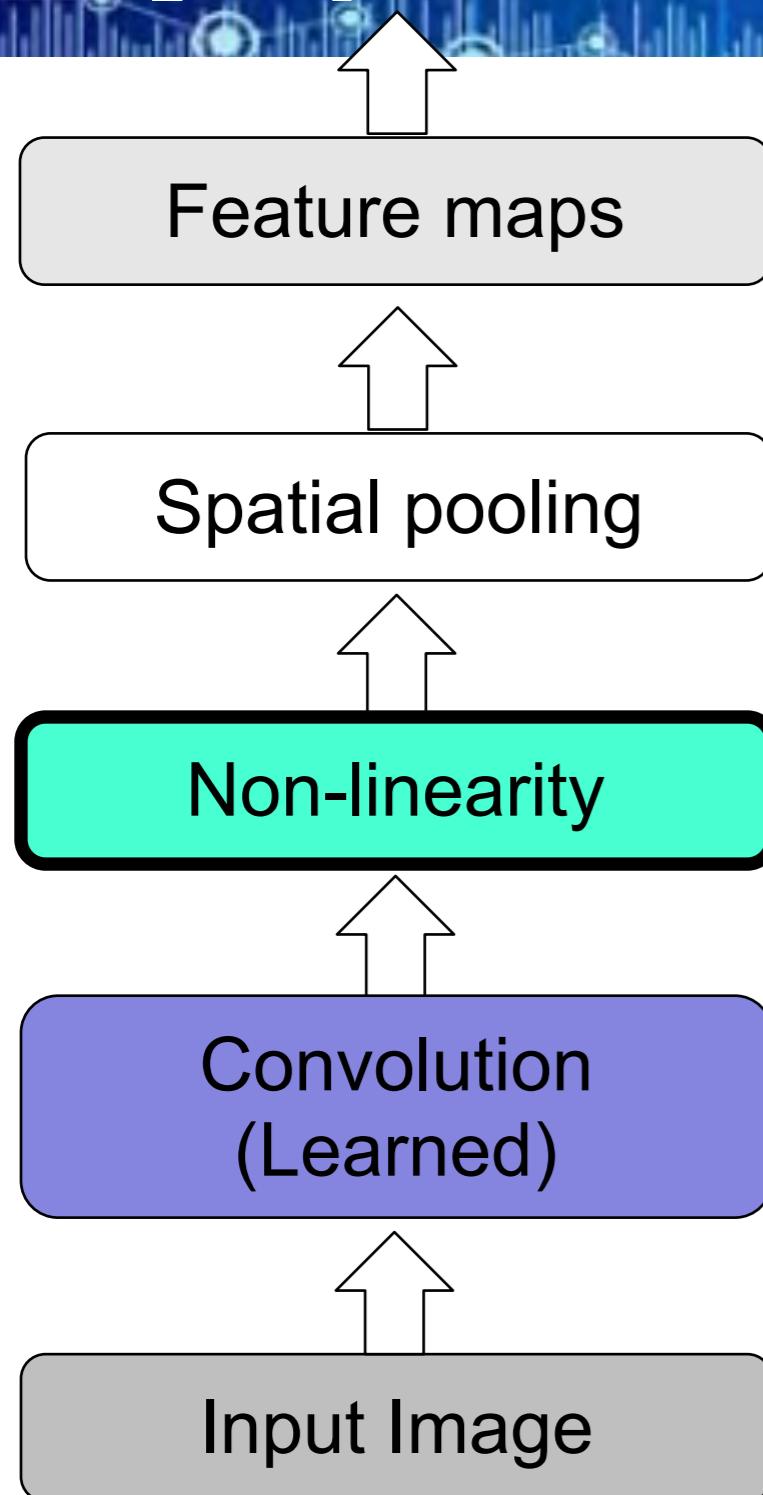
Input



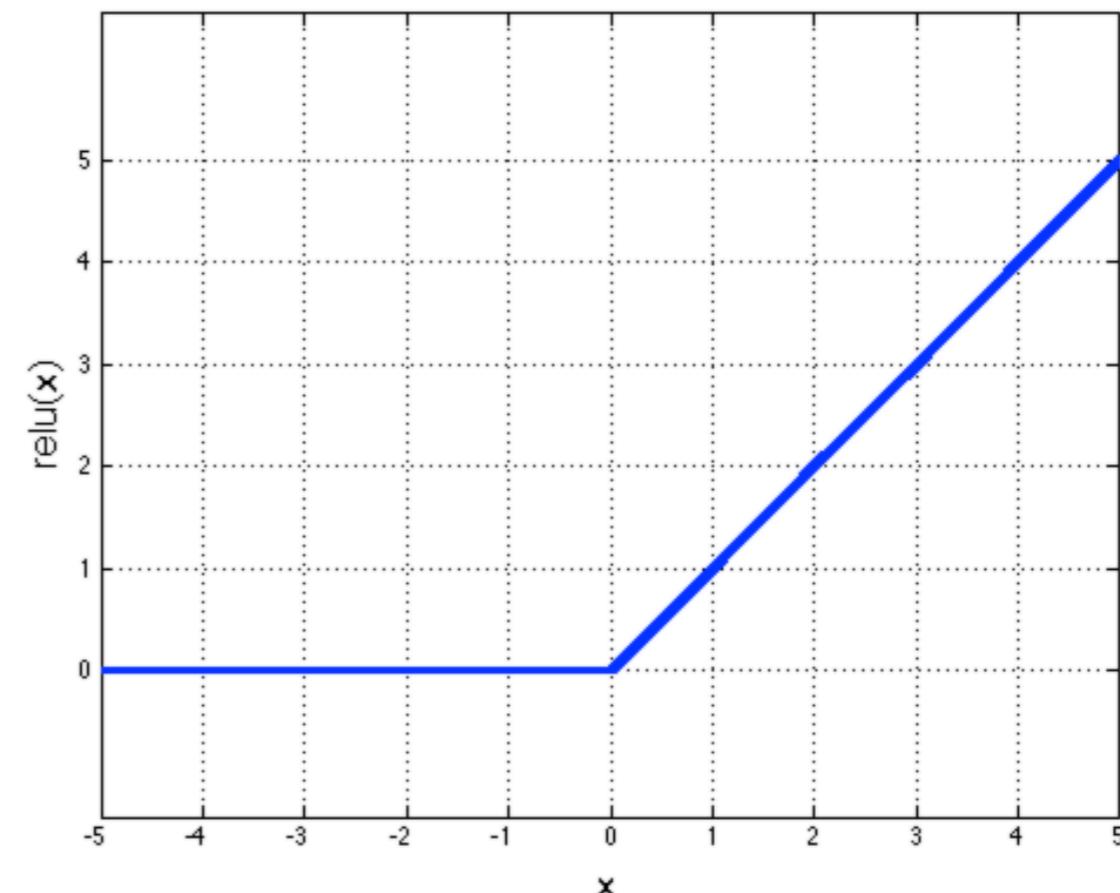
Feature Map

695

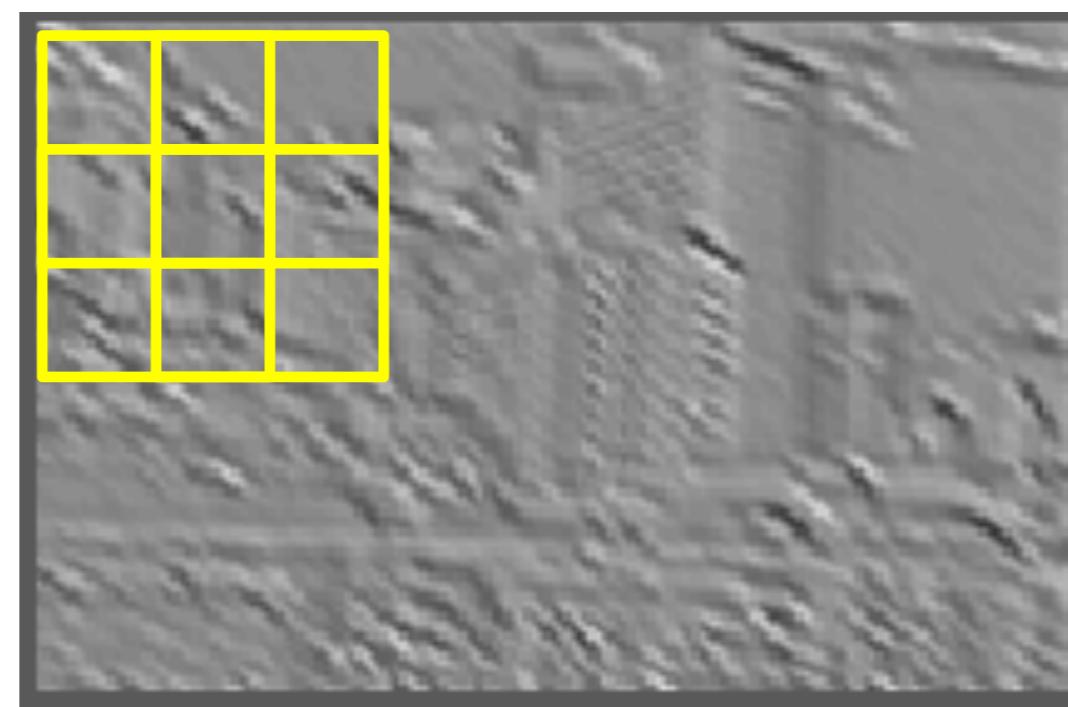
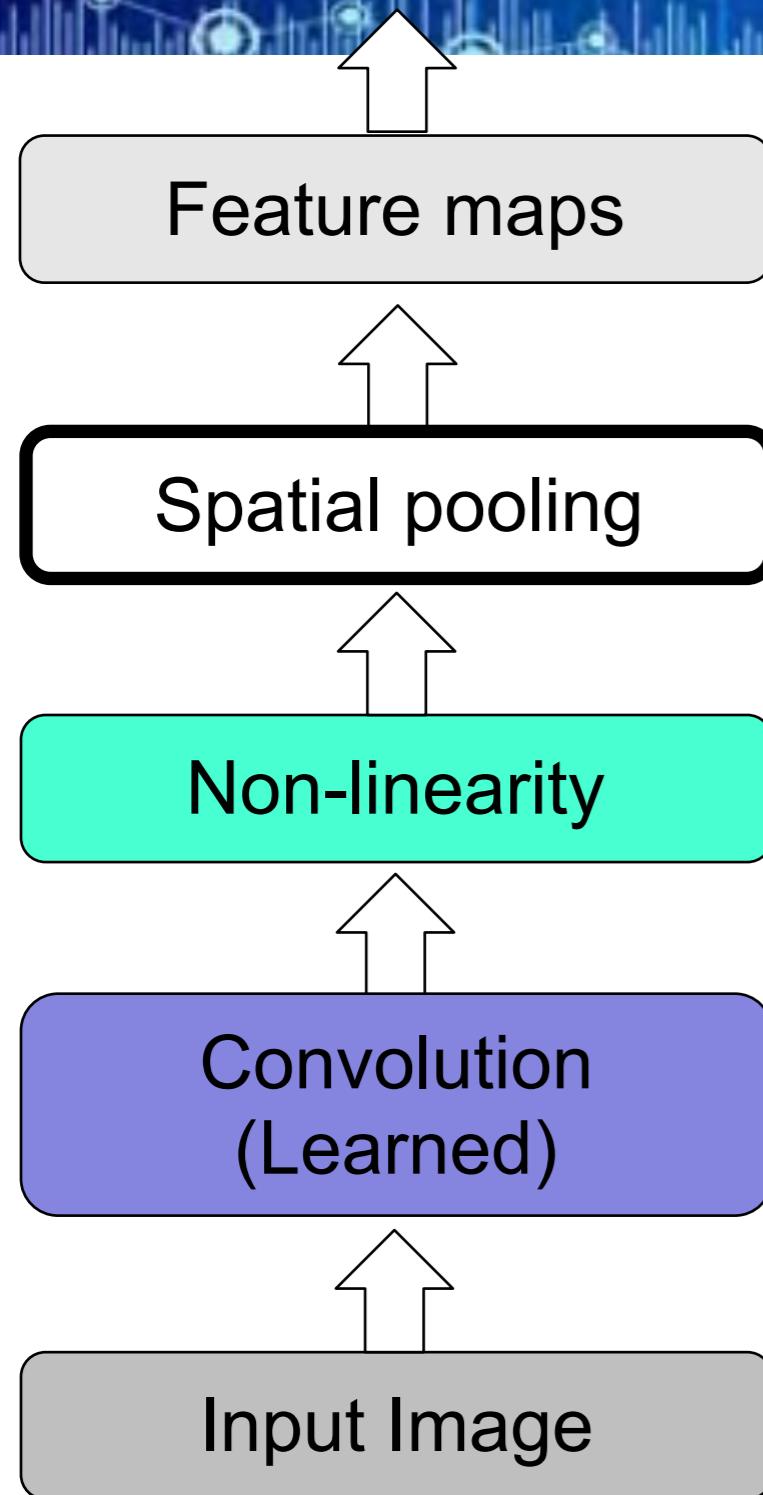
# Key operations



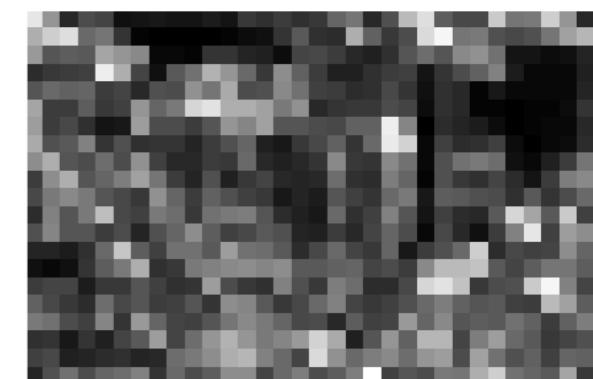
Rectified Linear Unit (ReLU)



# Key operations

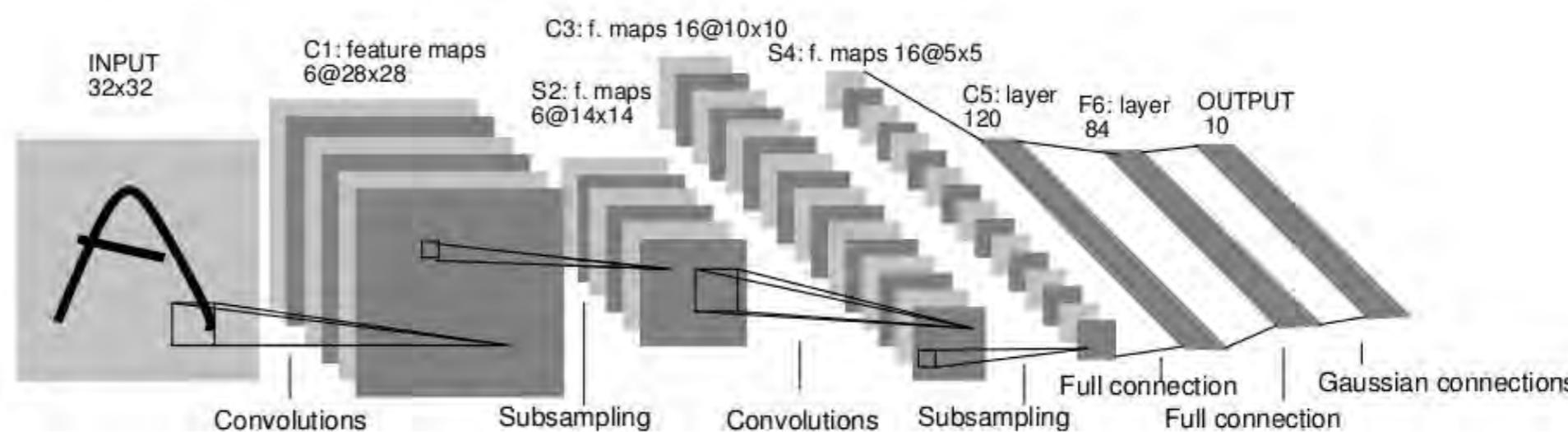


**Max**

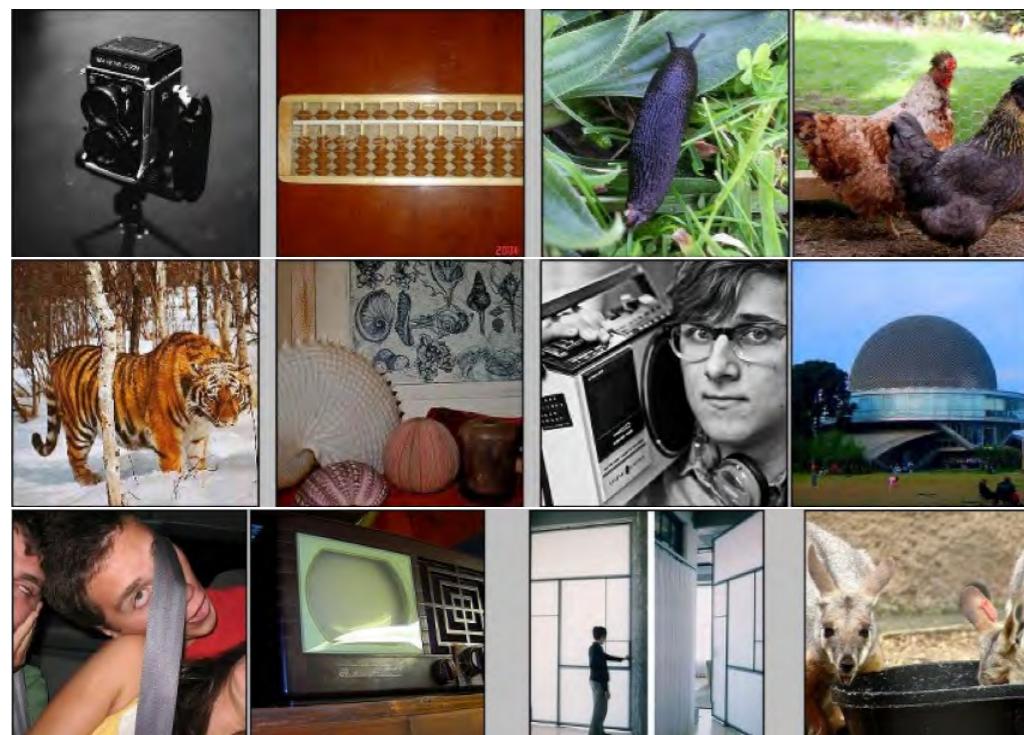


# LeNet-5

- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples



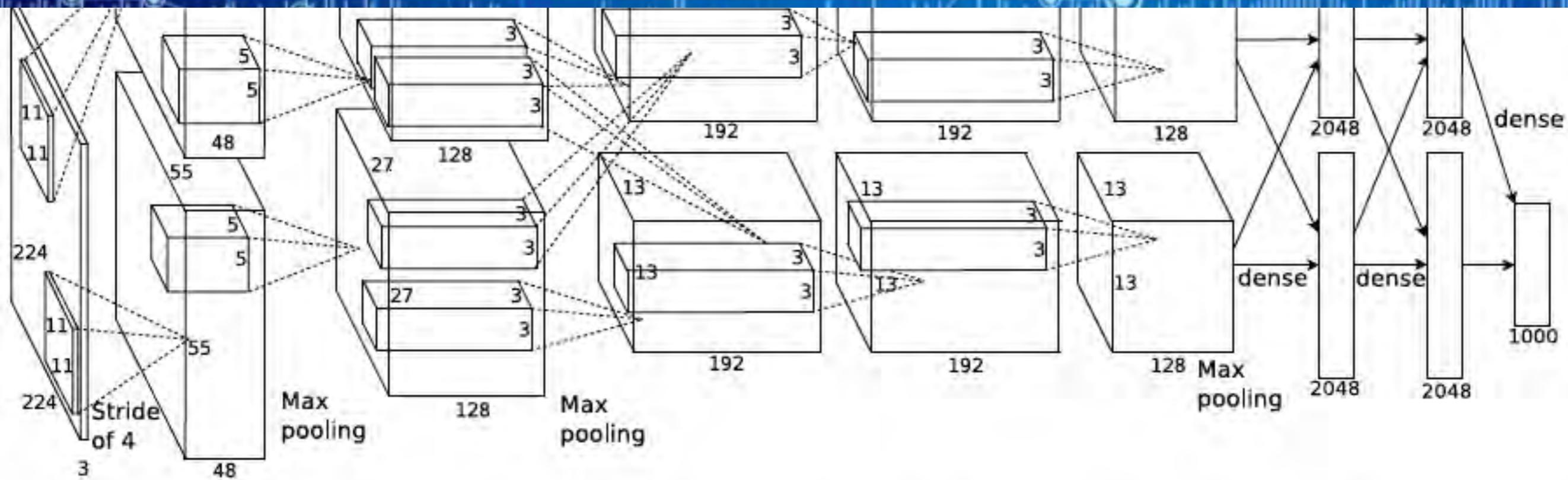
# Fast forward to the arrival of big visual



- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC): 1.2 million training images, 1000 classes

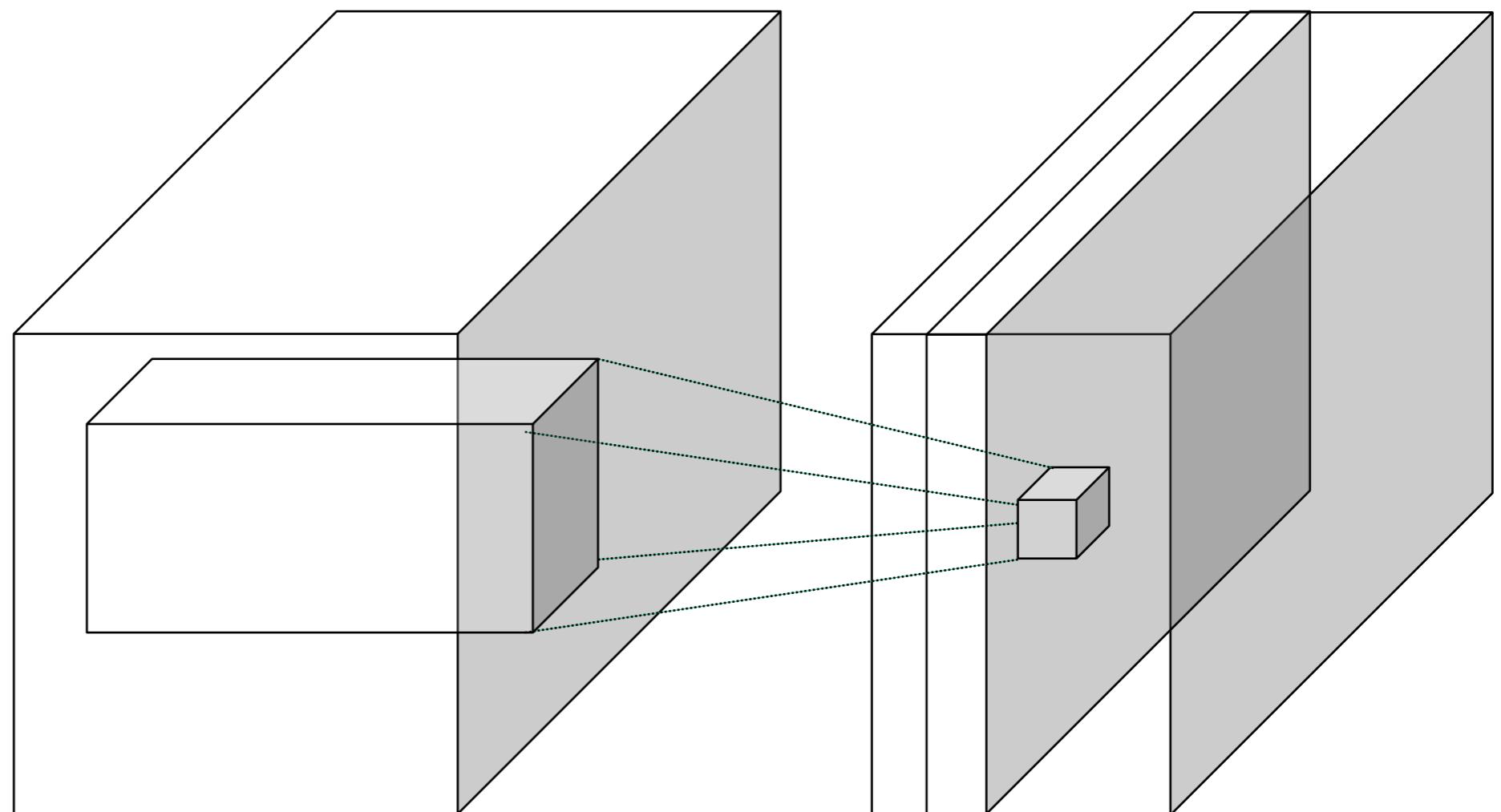
[www.image-net.org/challenges/LSVRC/](http://www.image-net.org/challenges/LSVRC/)

# AlexNet: ILSVRC 2012 winner



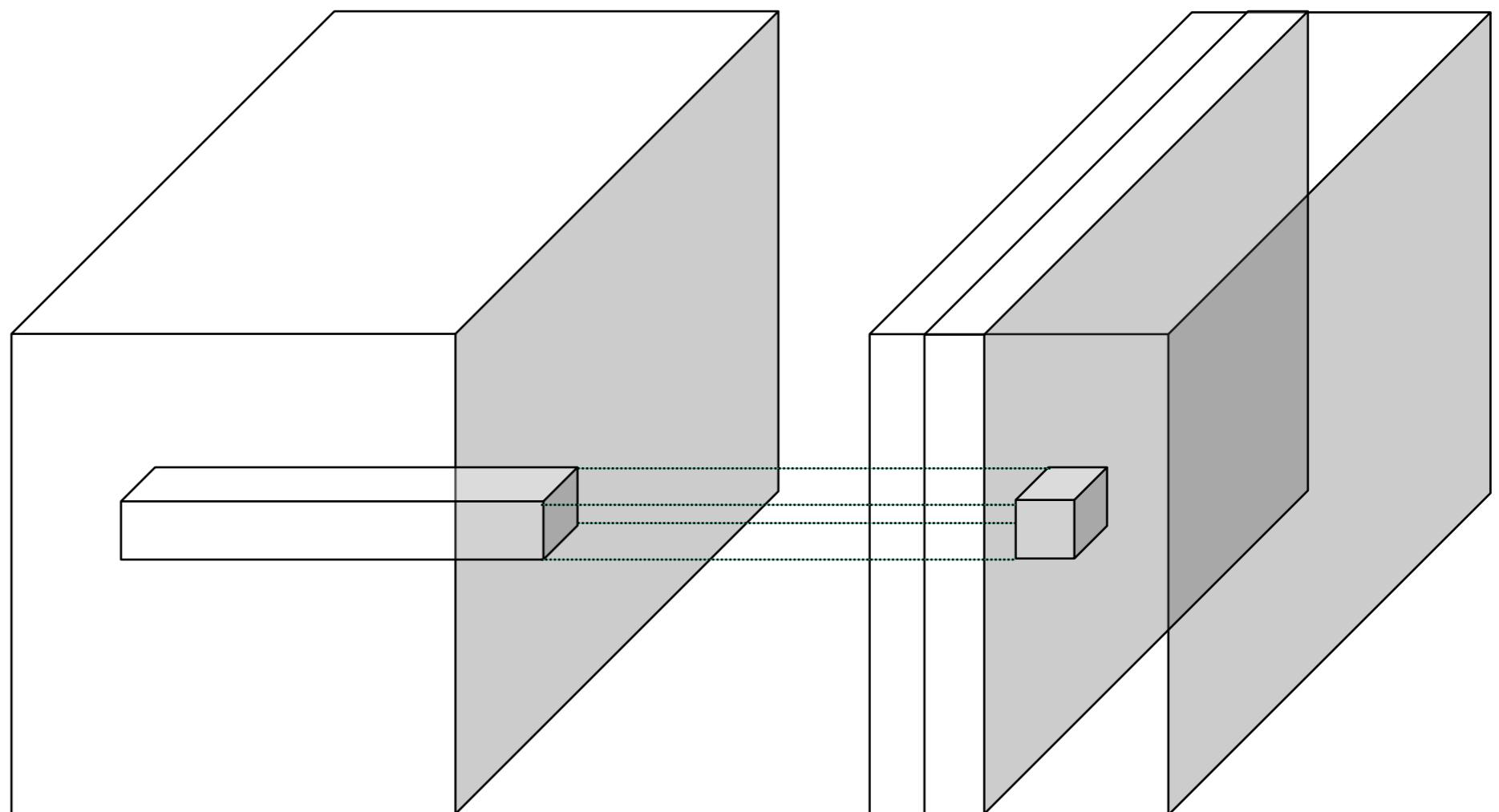
- Similar framework to LeNet but:
  - Max pooling, ReLU nonlinearity
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

# 1x1 convolutions



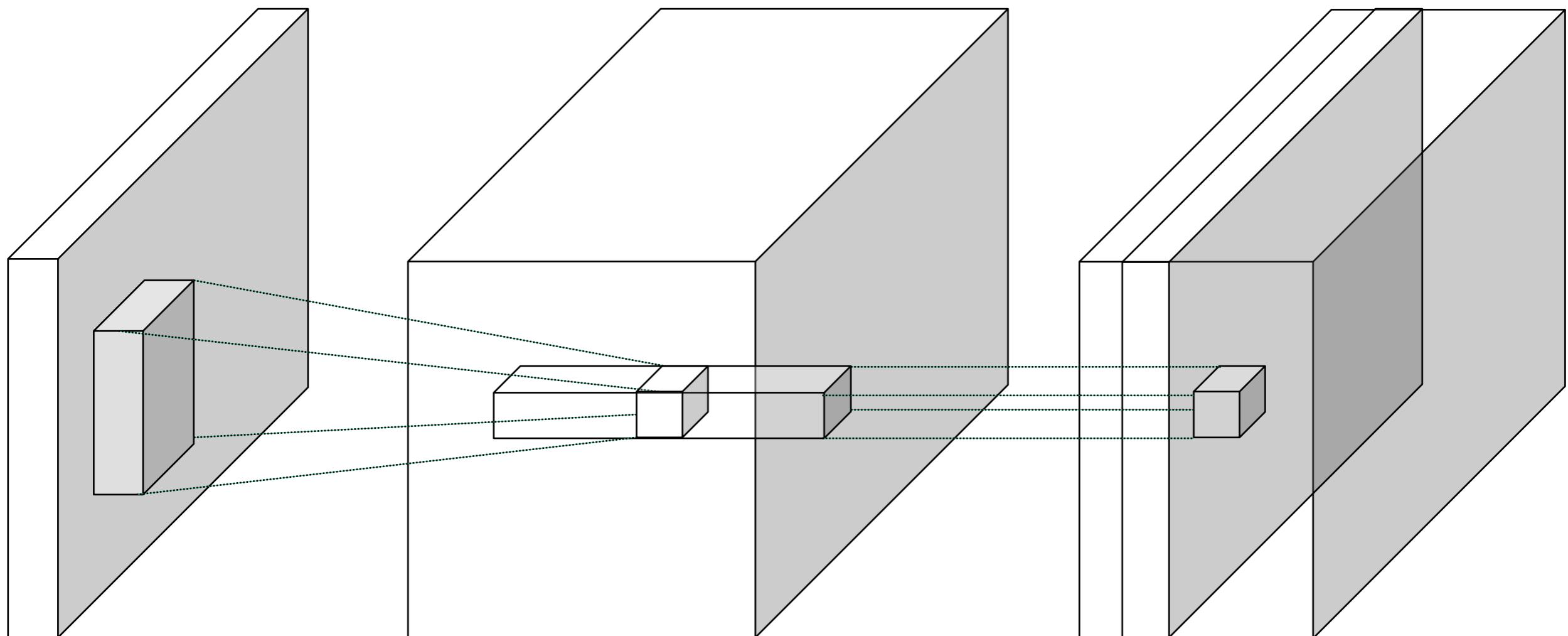
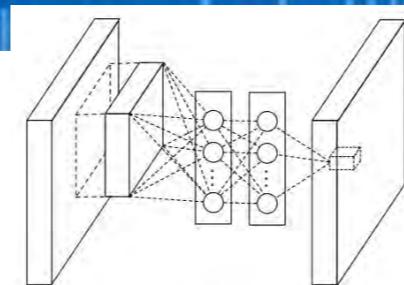
conv layer

# 1x1 convolutions



1x1 conv layer

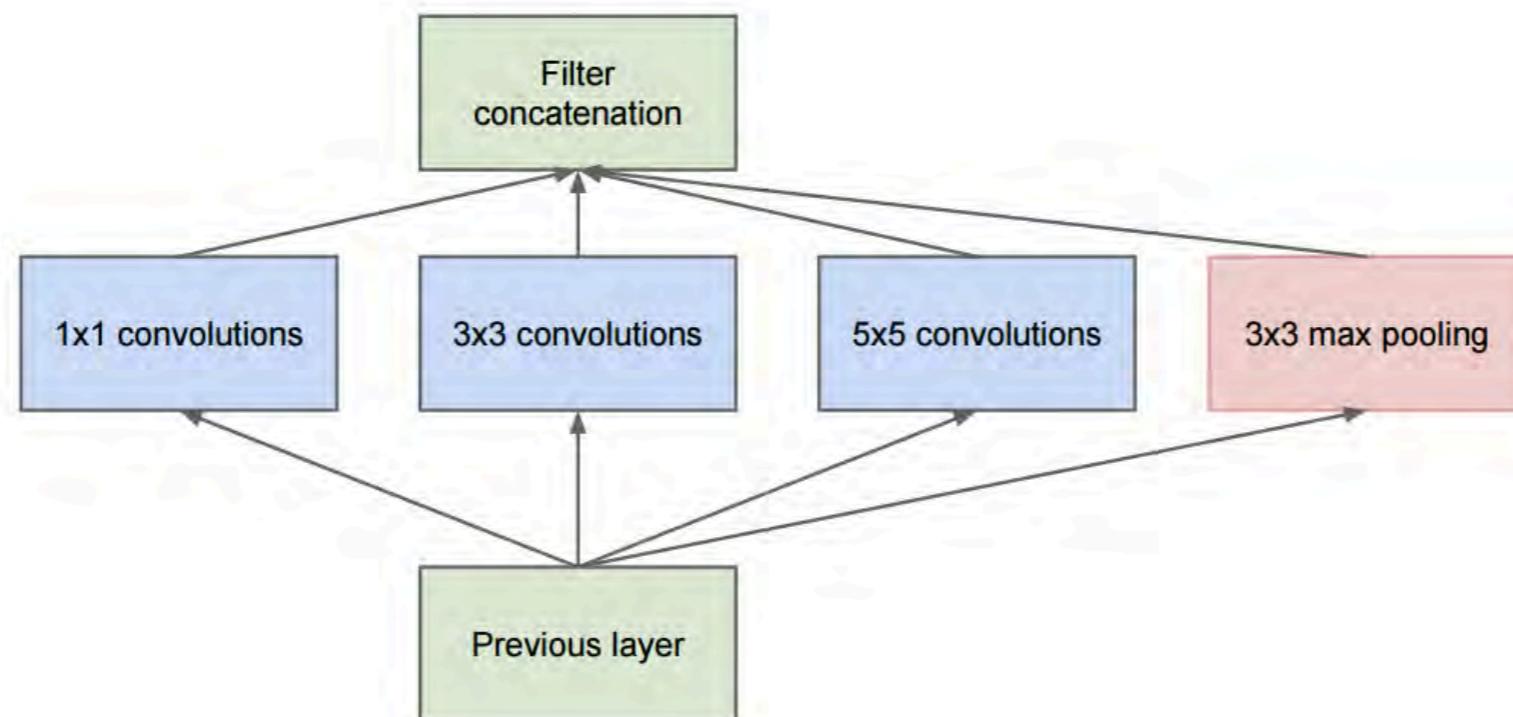
# 1x1 convolutions



1x1 conv layer

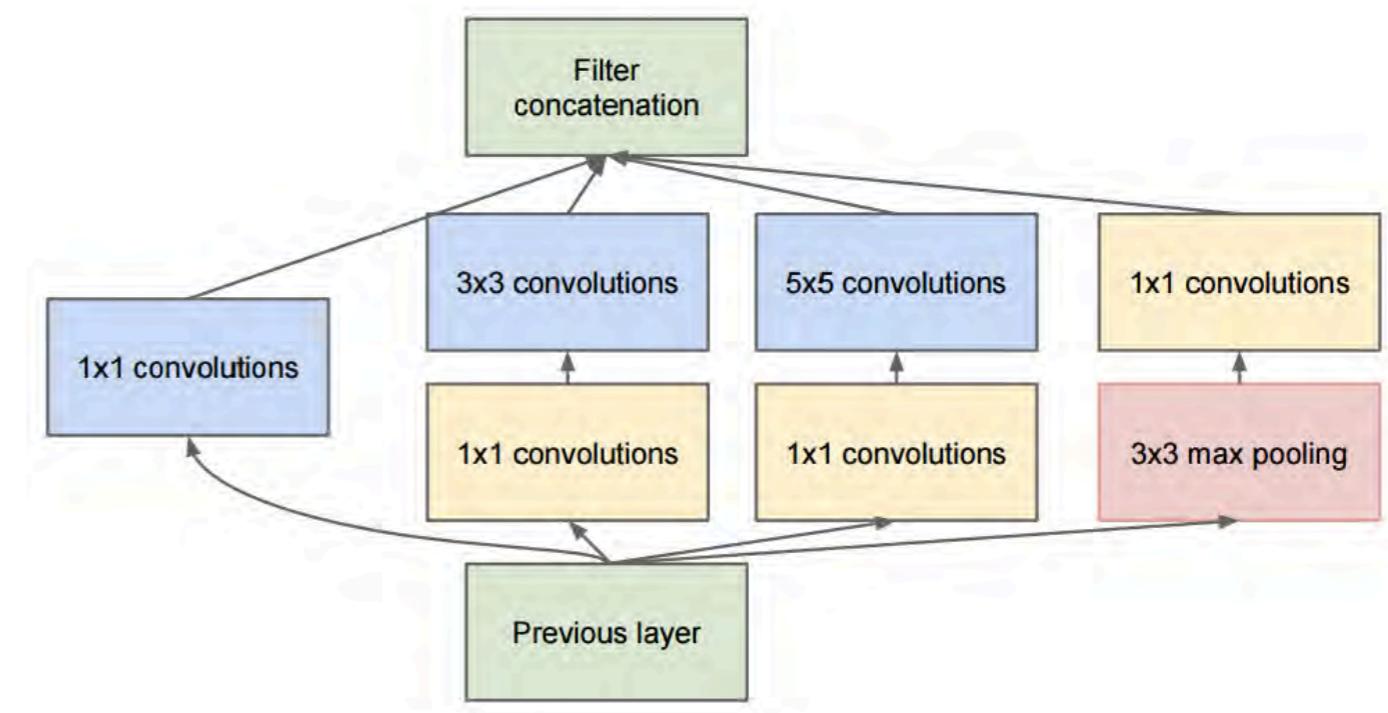
# GoogLeNet

- The Inception Module
  - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps

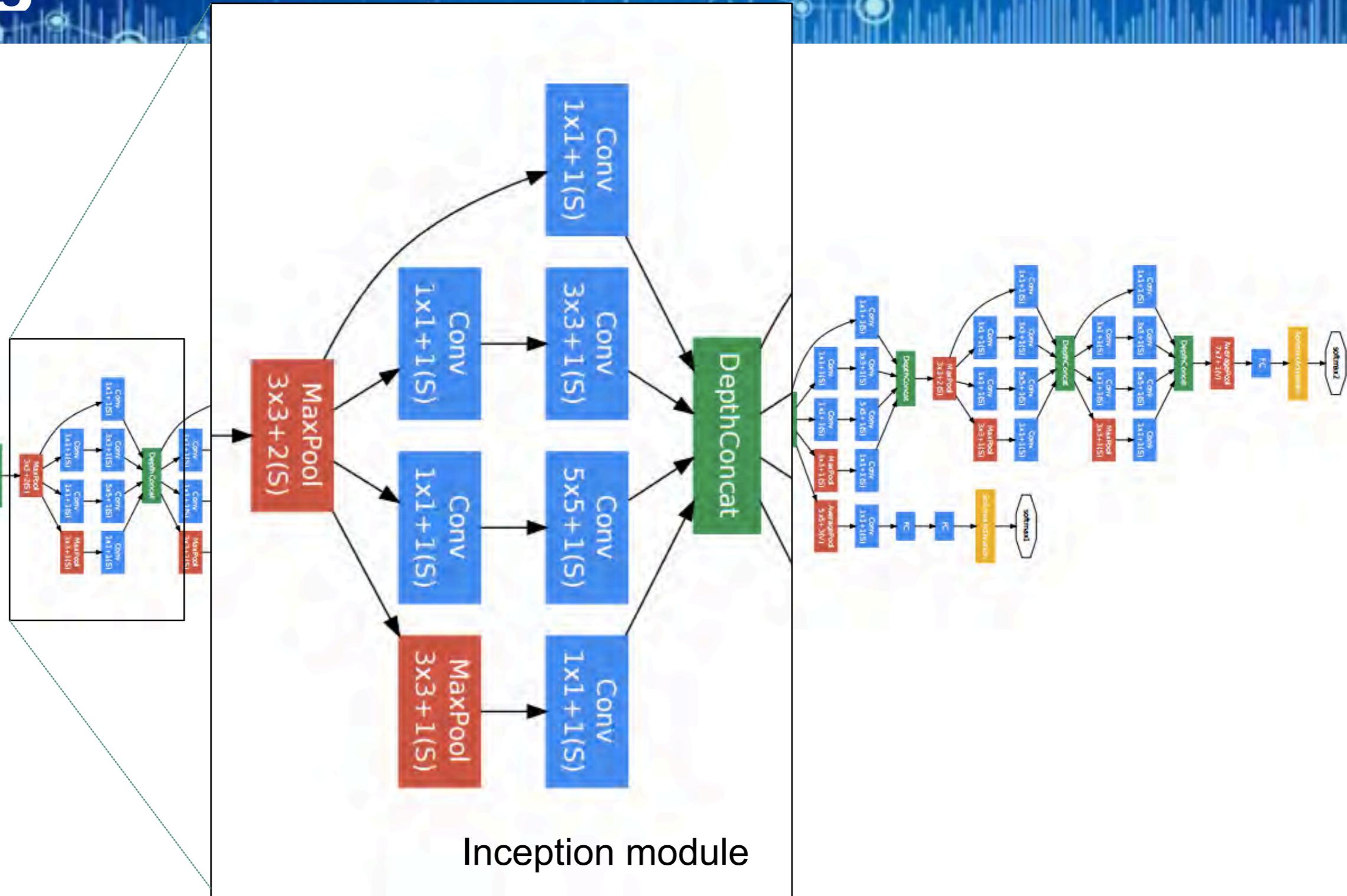


# GoogLeNet

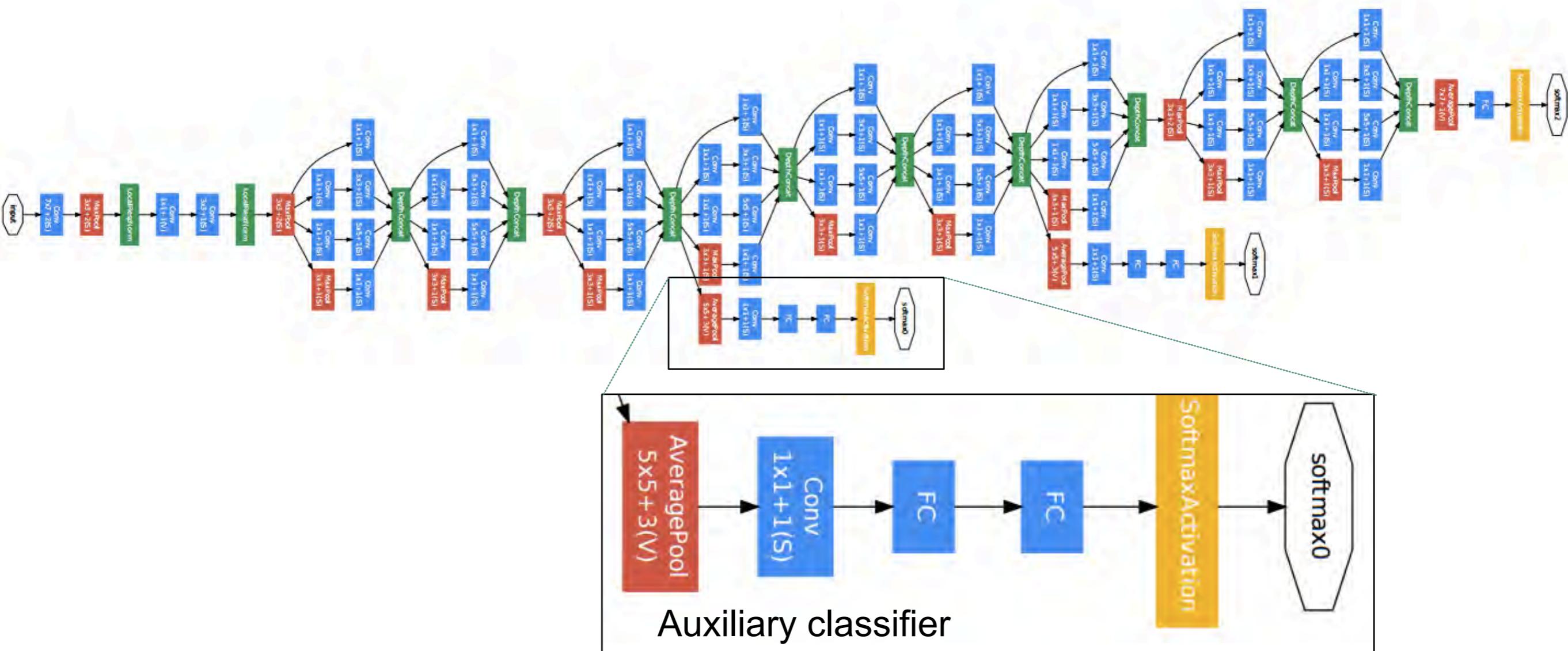
- The Inception Module
  - Parallel paths with different receptive field sizes and operations are meant to capture sparse patterns of correlations in the stack of feature maps
  - Use 1x1 convolutions for dimensionality reduction before expensive convolutions



# GoogLeNet



# GoogLeNet

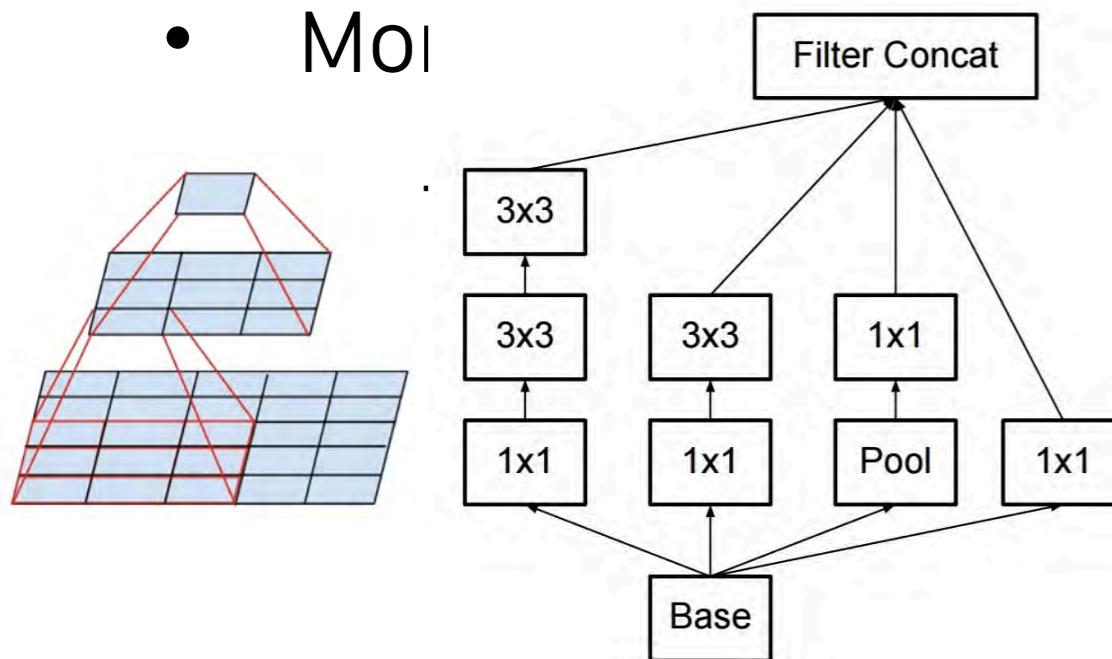


# GoogLeNet

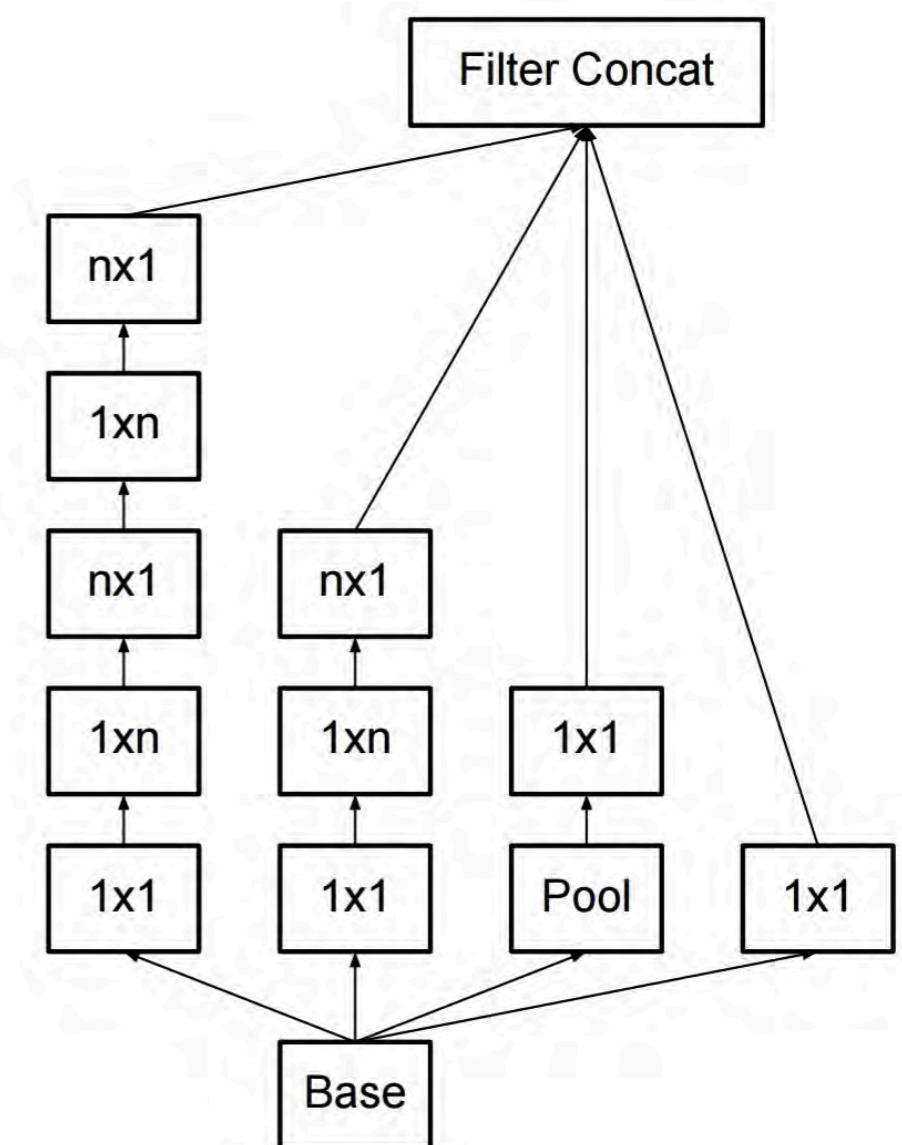
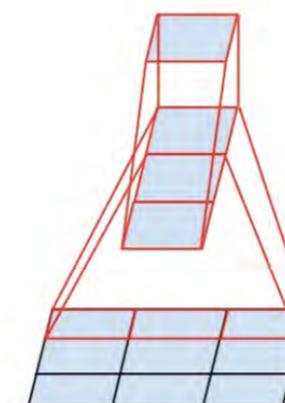
| type           | patch size/<br>stride | output<br>size             | depth | #1×1 | #3×3<br>reduce | #3×3 | #5×5<br>reduce | #5×5 | pool<br>proj | params | ops  |
|----------------|-----------------------|----------------------------|-------|------|----------------|------|----------------|------|--------------|--------|------|
| convolution    | $7 \times 7 / 2$      | $112 \times 112 \times 64$ | 1     |      |                |      |                |      |              | 2.7K   | 34M  |
| max pool       | $3 \times 3 / 2$      | $56 \times 56 \times 64$   | 0     |      |                |      |                |      |              |        |      |
| convolution    | $3 \times 3 / 1$      | $56 \times 56 \times 192$  | 2     |      | 64             | 192  |                |      |              | 112K   | 360M |
| max pool       | $3 \times 3 / 2$      | $28 \times 28 \times 192$  | 0     |      |                |      |                |      |              |        |      |
| inception (3a) |                       | $28 \times 28 \times 256$  | 2     | 64   | 96             | 128  | 16             | 32   | 32           | 159K   | 128M |
| inception (3b) |                       | $28 \times 28 \times 480$  | 2     | 128  | 128            | 192  | 32             | 96   | 64           | 380K   | 304M |
| max pool       | $3 \times 3 / 2$      | $14 \times 14 \times 480$  | 0     |      |                |      |                |      |              |        |      |
| inception (4a) |                       | $14 \times 14 \times 512$  | 2     | 192  | 96             | 208  | 16             | 48   | 64           | 364K   | 73M  |
| inception (4b) |                       | $14 \times 14 \times 512$  | 2     | 160  | 112            | 224  | 24             | 64   | 64           | 437K   | 88M  |
| inception (4c) |                       | $14 \times 14 \times 512$  | 2     | 128  | 128            | 256  | 24             | 64   | 64           | 463K   | 100M |
| inception (4d) |                       | $14 \times 14 \times 528$  | 2     | 112  | 144            | 288  | 32             | 64   | 64           | 580K   | 119M |
| inception (4e) |                       | $14 \times 14 \times 832$  | 2     | 256  | 160            | 320  | 32             | 128  | 128          | 840K   | 170M |
| max pool       | $3 \times 3 / 2$      | $7 \times 7 \times 832$    | 0     |      |                |      |                |      |              |        |      |
| inception (5a) |                       | $7 \times 7 \times 832$    | 2     | 256  | 160            | 320  | 32             | 128  | 128          | 1072K  | 54M  |
| inception (5b) |                       | $7 \times 7 \times 1024$   | 2     | 384  | 192            | 384  | 48             | 128  | 128          | 1388K  | 71M  |
| avg pool       | $7 \times 7 / 1$      | $1 \times 1 \times 1024$   | 0     |      |                |      |                |      |              |        |      |
| dropout (40%)  |                       | $1 \times 1 \times 1024$   | 0     |      |                |      |                |      |              |        |      |
| linear         |                       | $1 \times 1 \times 1000$   | 1     |      |                |      |                |      |              | 1000K  | 1M   |
| softmax        |                       | $1 \times 1 \times 1000$   | 0     |      |                |      |                |      |              |        |      |

# Inception v2, v3

- Regularize training with [batch normalization](#), reducing importance of auxiliary classifier
- Mol

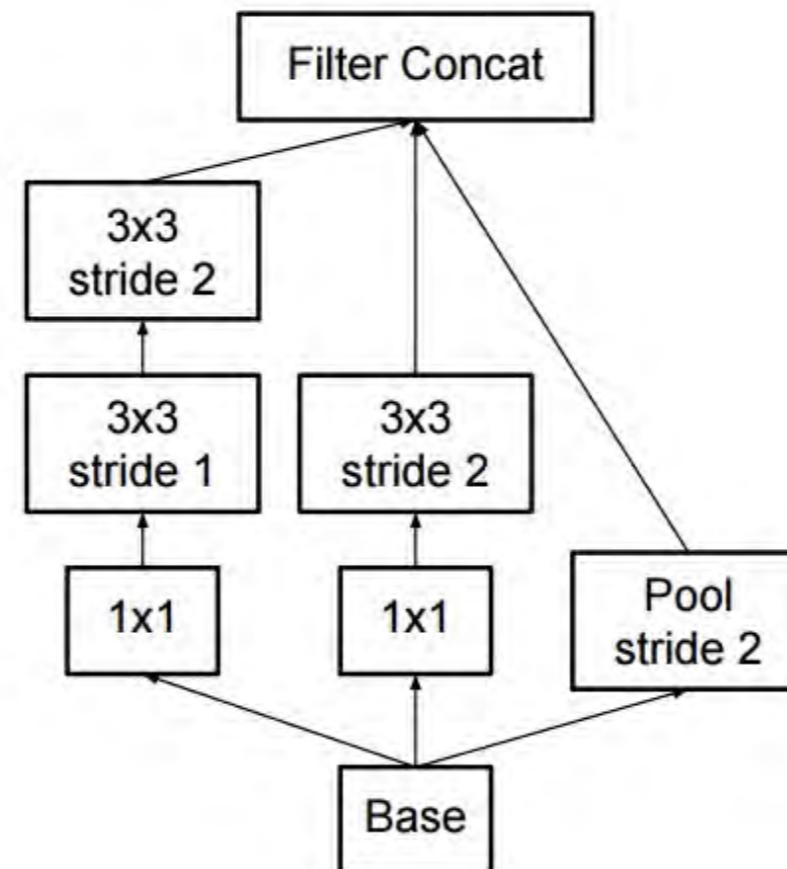


tion modu

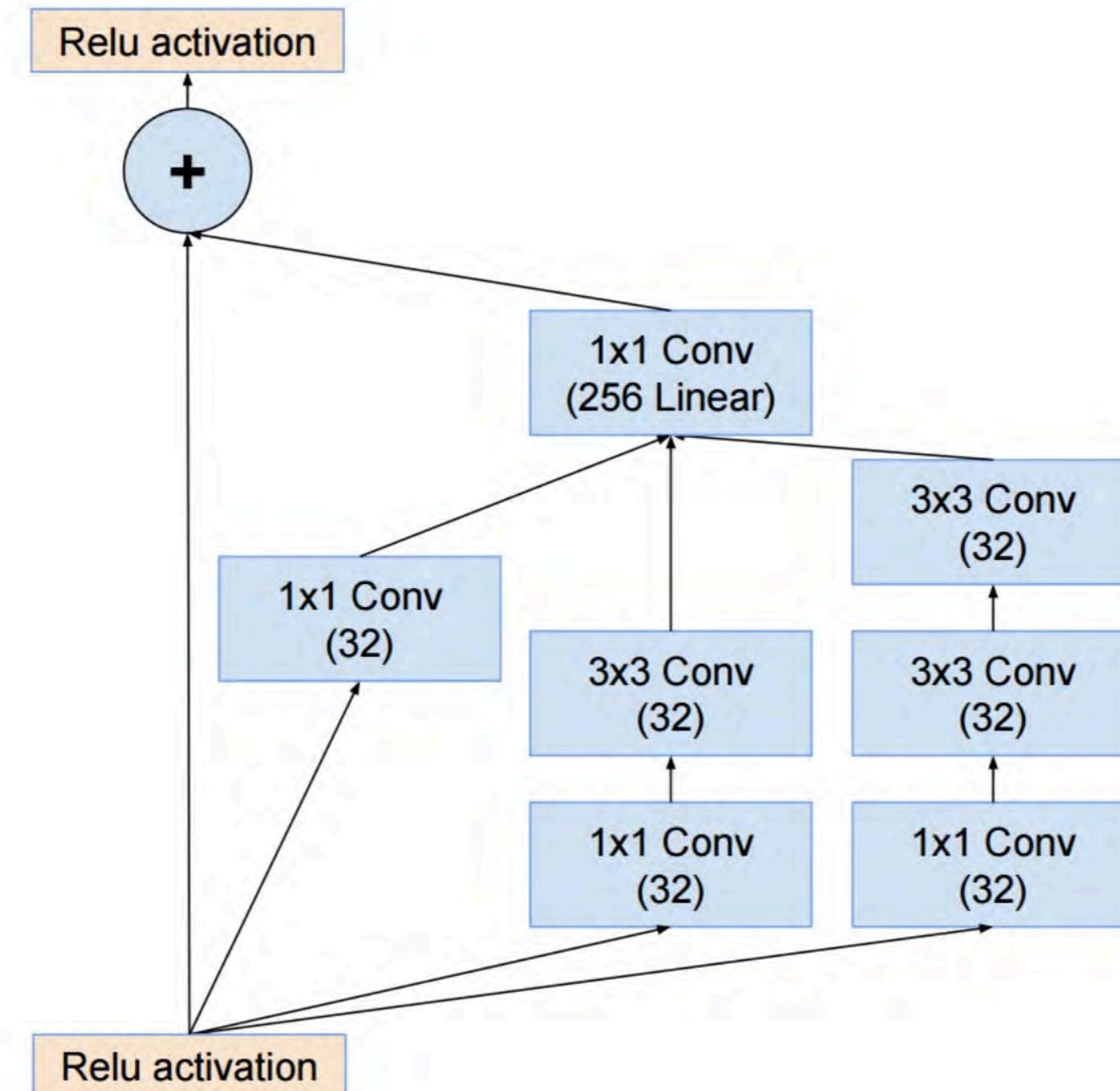


# Inception v2, v3

- Regularize training with [batch normalization](#), reducing importance of auxiliary classifiers
- More variants of inception modules with aggressive factorization
- Increase the spatial resolution while decreasing



# Inception v4

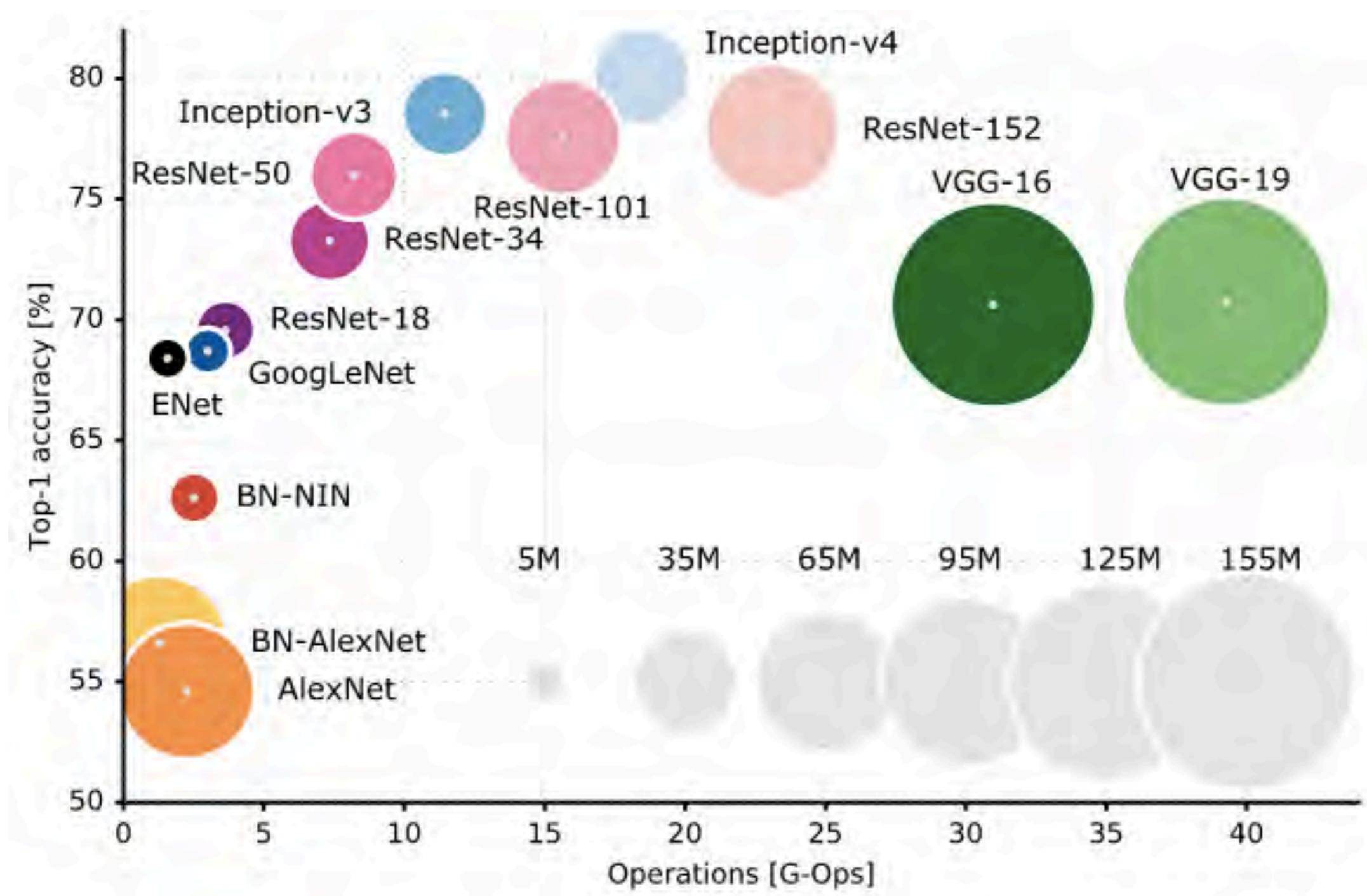


C. Szegedy et al., [Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning](#), arXiv 2016

# Summary: ILSVRC 2012-2015

| Team                                         | Year | Place | Error (top-5) | External data |
|----------------------------------------------|------|-------|---------------|---------------|
| SuperVision – Toronto<br>(AlexNet, 7 layers) | 2012 | -     | 16.4%         | no            |
| SuperVision                                  | 2012 | 1st   | 15.3%         | ImageNet 22k  |
| Clarifai – NYU (7 layers)                    | 2013 | -     | 11.7%         | no            |
| Clarifai                                     | 2013 | 1st   | 11.2%         | ImageNet 22k  |
| VGG – Oxford (16 layers)                     | 2014 | 2nd   | 7.32%         | no            |
| GoogLeNet (19 layers)                        | 2014 | 1st   | 6.67%         | no            |
| ResNet (152 layers)                          | 2015 | 1st   | 3.57%         |               |
| Human expert*                                |      |       | 5.1%          |               |

# Accuracy vs. efficiency



# Design principles

- Reduce filter sizes (except possibly at the lowest layer), factorize filters aggressively
- Use 1x1 convolutions to reduce and expand the number of feature maps judiciously
- Use skip connections and/or create multiple paths through the network



# CNN in R using Tensorflow

[Products](#)[Why Anaconda?](#)[Solutions](#)[Resources](#)[Company](#)[Contact Us](#)[Download](#)

## EARLY BIRD REGISTRATION IS OPEN!

### AnacondaCON 2020

June 3-5, 2020 JW Marriott | Austin, TX

Get your early-bird tickets. On sale now!

[Learn More](#)

## Solutions for Data Science Practitioners and Enterprise Machine Learning

<https://www.anaconda.com>



## Anaconda 2019.10 for macOS Installer

### Python 3.7 version

[Download](#)

64-Bit Graphical Installer (654 MB)

64-Bit Command Line Installer (424 MB)

### Python 2.7 version

[Download](#)

64-Bit Graphical Installer (637 MB)

64-Bit Command Line Installer (409 MB )

# Setup Tensorflow for R

```
library(tensorflow)
library(keras)

install_tensorflow(method = "conda", version = "1.15",
 envname = "r-deeplearning")

reticulate::use_condaenv(condaenv="r-deeplearning",
 conda="/Users/kris/anaconda3/bin/conda",
 required=TRUE)
```

# Convolutional Neural Network (CNN)

This tutorial demonstrates training a simple Convolutional Neural Network (CNN) to classify CIFAR images. Because this tutorial uses the Keras Sequential API, creating and training our model will take just a few lines of code.

## Setup %

```
library(tensorflow)
library(keras)
```

## Download and prepare the CIFAR10 dataset

The CIFAR10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

```
cifar <- dataset_cifar10()
```

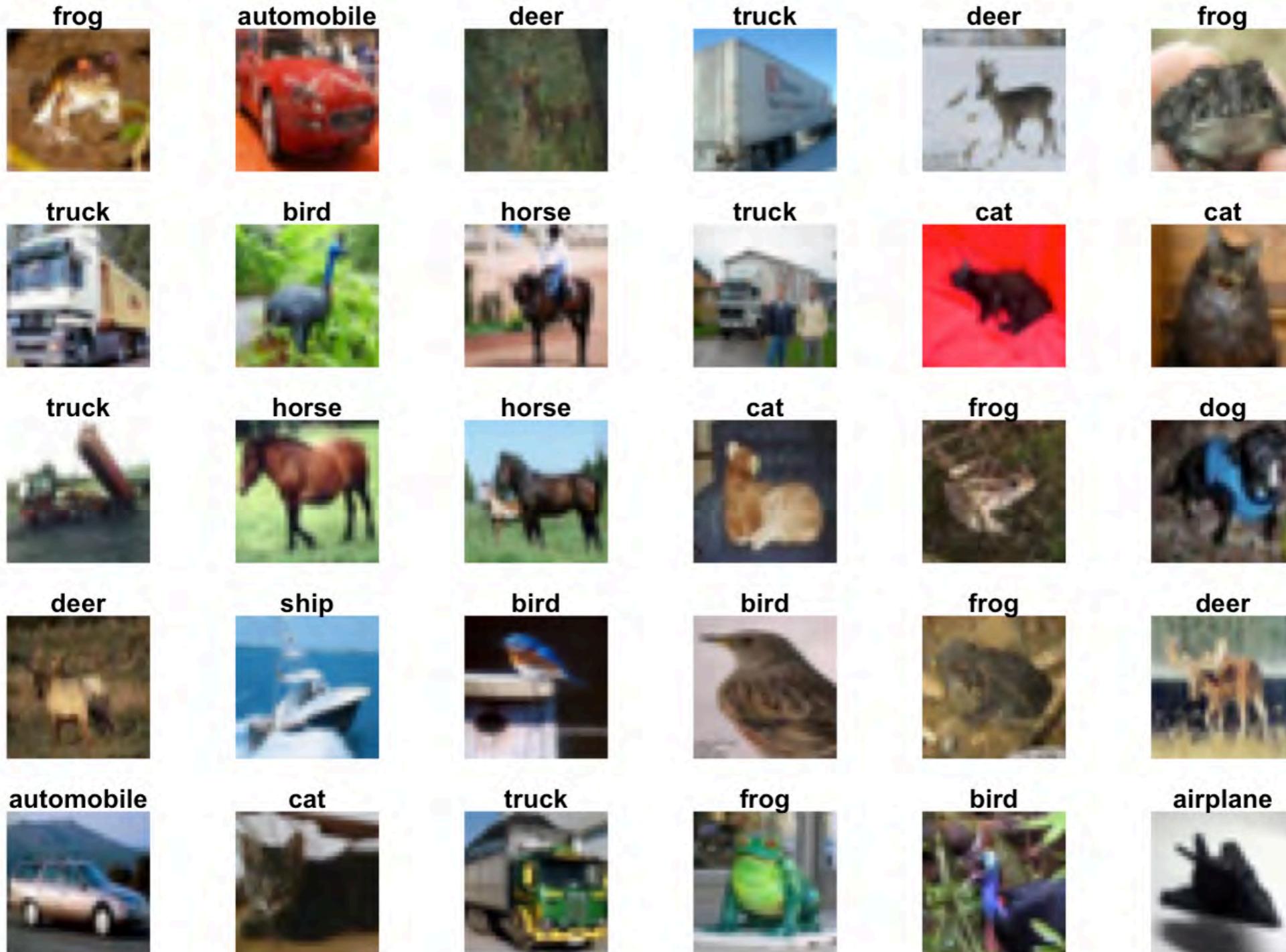
## Verify the data

To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image.

```
class_names <- c('airplane', 'automobile', 'bird', 'cat', 'deer',
 'dog', 'frog', 'horse', 'ship', 'truck')

index <- 1:30

par(mfcol = c(5,6), mar = rep(1, 4), oma = rep(0.2, 4))
cifar$train$x[index,,,] %>%
 purrr::array_tree(1) %>%
 purrr::set_names(class_names[cifar$train$y[index] + 1]) %>%
 purrr::map(as.raster, max = 255) %>%
 purrr::iwalk(~{plot(.x); title(.y)})
```



## Create the convolutional base

The 6 lines of code below define the convolutional base using a common pattern: a stack of Conv2D and MaxPooling2D layers.

As input, a CNN takes tensors of shape (image\_height, image\_width, color\_channels), ignoring the batch size. If you are new to these dimensions, color\_channels refers to (R,G,B). In this example, you will configure our CNN to process inputs of shape (32, 32, 3), which is the format of CIFAR images. You can do this by passing the argument `input_shape` to our first layer.

```
model <- keras_model_sequential() %>%
 layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = "relu",
 input_shape = c(32,32,3)) %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu") %>%
 layer_max_pooling_2d(pool_size = c(2,2)) %>%
 layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = "relu")
```

```
summary(model)
```

```
Model: "sequential"

Layer (type) Output Shape Param
=====
conv2d (Conv2D) (None, 30, 30, 32) 896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
conv2d_1 (Conv2D) (None, 13, 13, 64) 18496
max_pooling2d_1 (MaxPooling2D) (None, 6, 6, 64) 0
conv2d_2 (Conv2D) (None, 4, 4, 64) 36928
=====
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0

```

## Add Dense layers on top

To complete our model, you will feed the last output tensor from the convolutional base (of shape (3, 3, 64)) into one or more Dense layers to perform classification. Dense layers take vectors as input (which are 1D), while the current output is a 3D tensor. First, you will flatten (or unroll) the 3D output to 1D, then add one or more Dense layers on top. CIFAR has 10 output classes, so you use a final Dense layer with 10 outputs and a softmax activation.

```
model %>%
 layer_flatten() %>%
 layer_dense(units = 64, activation = "relu") %>%
 layer_dense(units = 10, activation = "softmax")
```

```
summary(model)
```

```
Model: "sequential"

Layer (type) Output Shape Param
=====
conv2d (Conv2D) (None, 30, 30, 32) 896

max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0

conv2d_1 (Conv2D) (None, 13, 13, 64) 18496

max_pooling2d_1 (MaxPooling2D) (None, 6, 6, 64) 0

conv2d_2 (Conv2D) (None, 4, 4, 64) 36928

flatten (Flatten) (None, 1024) 0

dense (Dense) (None, 64) 65600

dense_1 (Dense) (None, 10) 650
=====
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0

```

## Compile and train the model

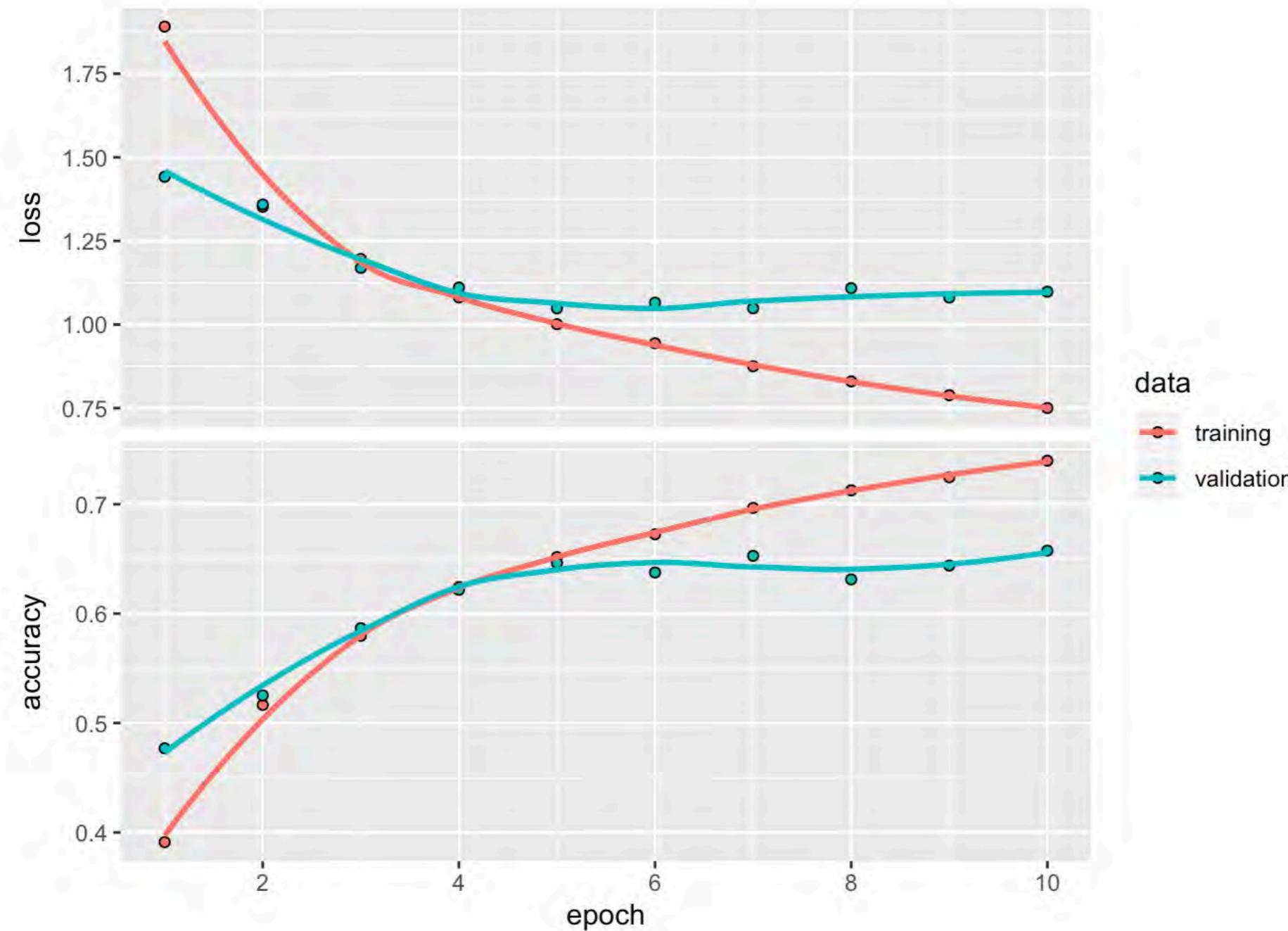
```
model %>% compile(
 optimizer = "adam",
 loss = "sparse_categorical_crossentropy",
 metrics = "accuracy"
)

history <- model %>%
 fit(
 x = cifar$train$x, y = cifar$train$y,
 epochs = 10,
 validation_data = uname(cifar$test),
 verbose = 2
)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 - 33s - loss: 1.8909 - accuracy: 0.3912 - val_loss: 1.4419 - val_accuracy: 0.4
Epoch 2/10
50000/50000 - 31s - loss: 1.3519 - accuracy: 0.5165 - val_loss: 1.3589 - val_accuracy: 0.5
Epoch 3/10
50000/50000 - 32s - loss: 1.1962 - accuracy: 0.5797 - val_loss: 1.1695 - val_accuracy: 0.5
Epoch 4/10
50000/50000 - 34s - loss: 1.0811 - accuracy: 0.6243 - val_loss: 1.1108 - val_accuracy: 0.6
Epoch 5/10
50000/50000 - 34s - loss: 1.0010 - accuracy: 0.6516 - val_loss: 1.0483 - val_accuracy: 0.6
Epoch 6/10
50000/50000 - 33s - loss: 0.9441 - accuracy: 0.6725 - val_loss: 1.0655 - val_accuracy: 0.6
Epoch 7/10
50000/50000 - 33s - loss: 0.8754 - accuracy: 0.6966 - val_loss: 1.0486 - val_accuracy: 0.6
Epoch 8/10
50000/50000 - 33s - loss: 0.8300 - accuracy: 0.7127 - val_loss: 1.1089 - val_accuracy: 0.6
Epoch 9/10
50000/50000 - 33s - loss: 0.7887 - accuracy: 0.7246 - val_loss: 1.0807 - val_accuracy: 0.6
Epoch 10/10
50000/50000 - 34s - loss: 0.7503 - accuracy: 0.7398 - val_loss: 1.0982 - val_accuracy: 0.6
```

## Evaluate the model

```
plot(history)
```



# Evaluate Model

```
evaluate(model, cifar$test$x, cifar$test$y, verbose = 0)
```

```
$loss
[1] 1.098242

$accuracy
[1] 0.6576
```

Our simple CNN has achieved a test accuracy of over 70%. Not bad for a few lines of code!

# Workshop 4.20 - CNN with Wave data

Using datagenerator.R code from Github.

```
df <- readRDS("data/df.rds") %>% sample_frac(1)
id_train <- sample(nrow(df), size = 0.7*nrow(df))

ds_train <- data_generator(df[id_train,], 32L)
ds_test <- data_generator(df[-id_train,], 32, shuffle = FALSE)
```

Using command wave file, create CNN model to predict command.

# Setup Tensorflow for R

```
library(tensorflow)
library(keras)

install_tensorflow(method = "conda", version = "1.15",
 envname = "r-deeplearning")

library(tensorflow)
library(keras)

install_tensorflow(method = "conda", version = "1.15-gpu",
 envname = "r-deeplearning")

reticulate::use_condaenv(condaenv="r-deeplearning",
 conda="/Users/kris/anaconda3/bin/conda",
 required=TRUE)
```

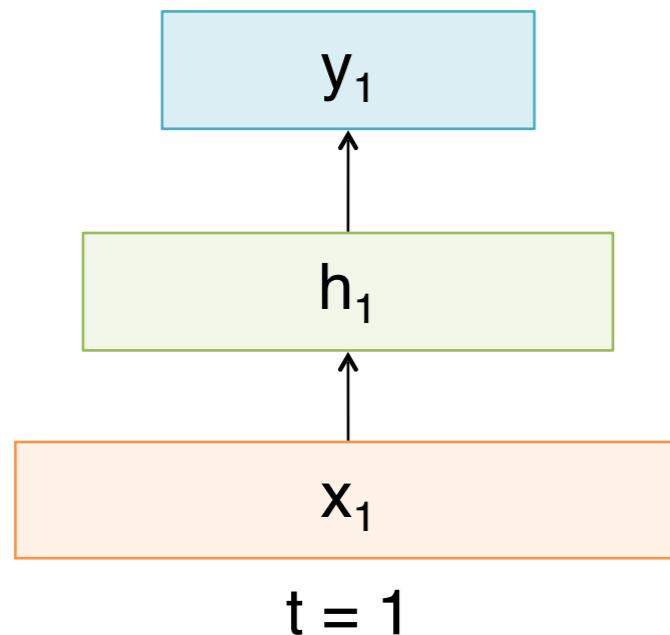


# Recurrent Neural Net

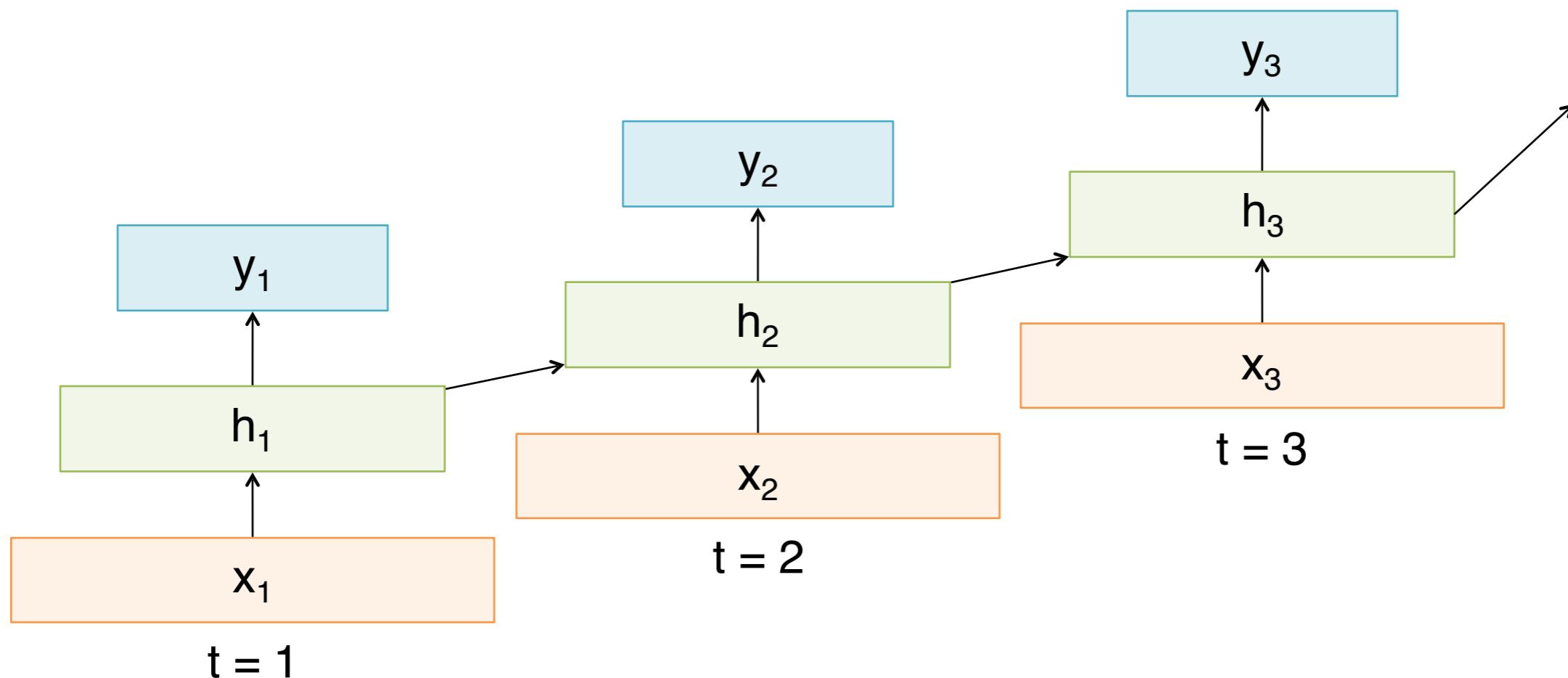
# Recurrent Neural Networks (RNNs)

- Recurrent Neural Networks take the previous output or hidden states as inputs.  
The composite input at time  $t$  has some historical information about the happenings at time  $T < t$
- RNNs are useful as their intermediate values (state) can store information about past inputs for a time that is not fixed a priori

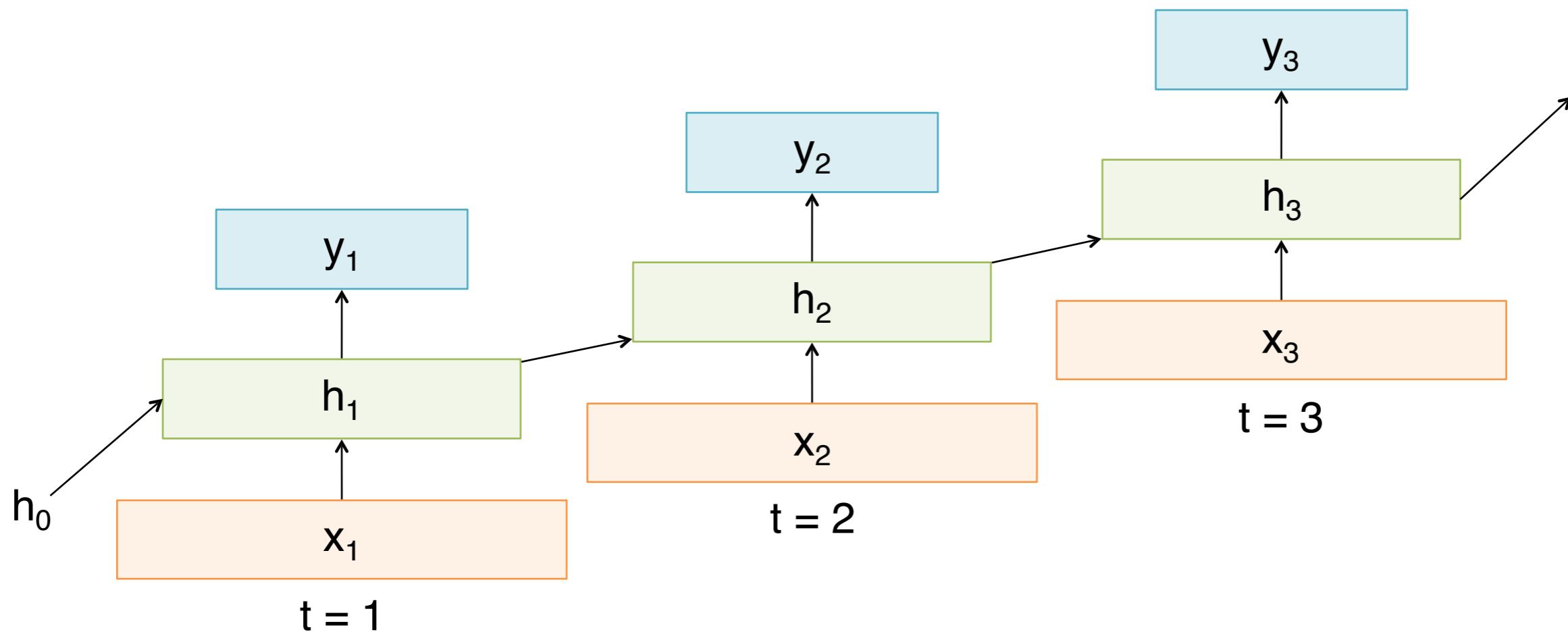
# Sample Feed-forward Network



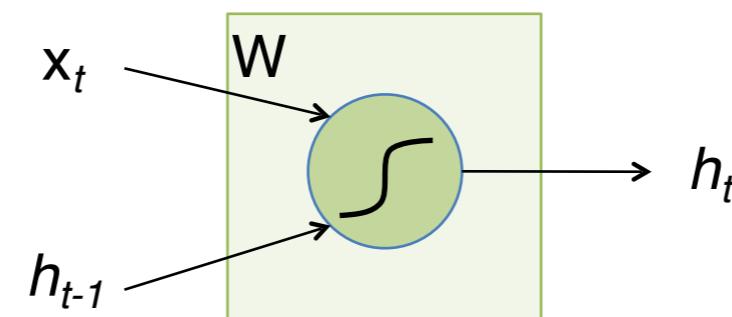
# Sample RNN



# Sample RNN

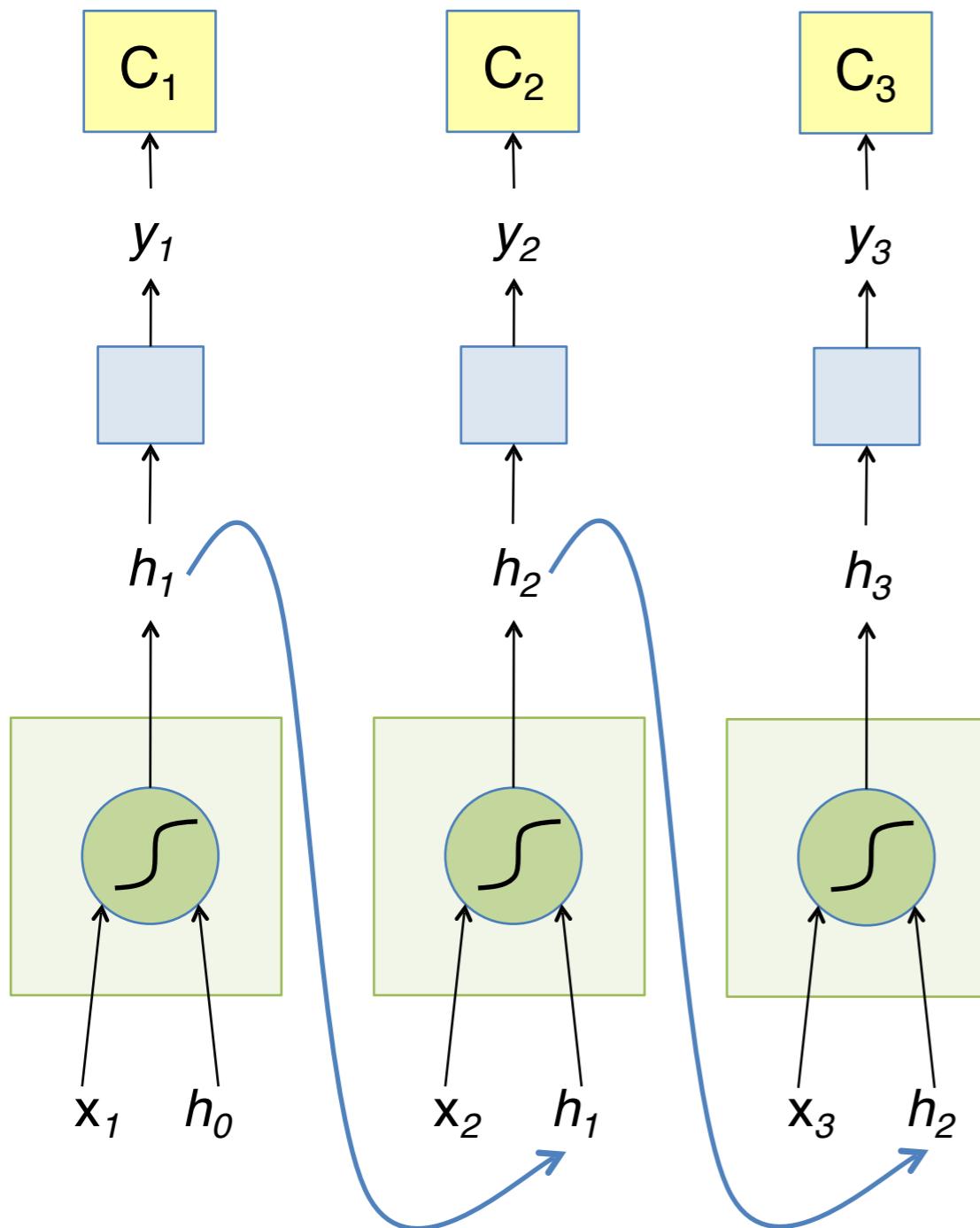


# The Vanilla RNN Cell



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

# The Vanilla RNN Forward

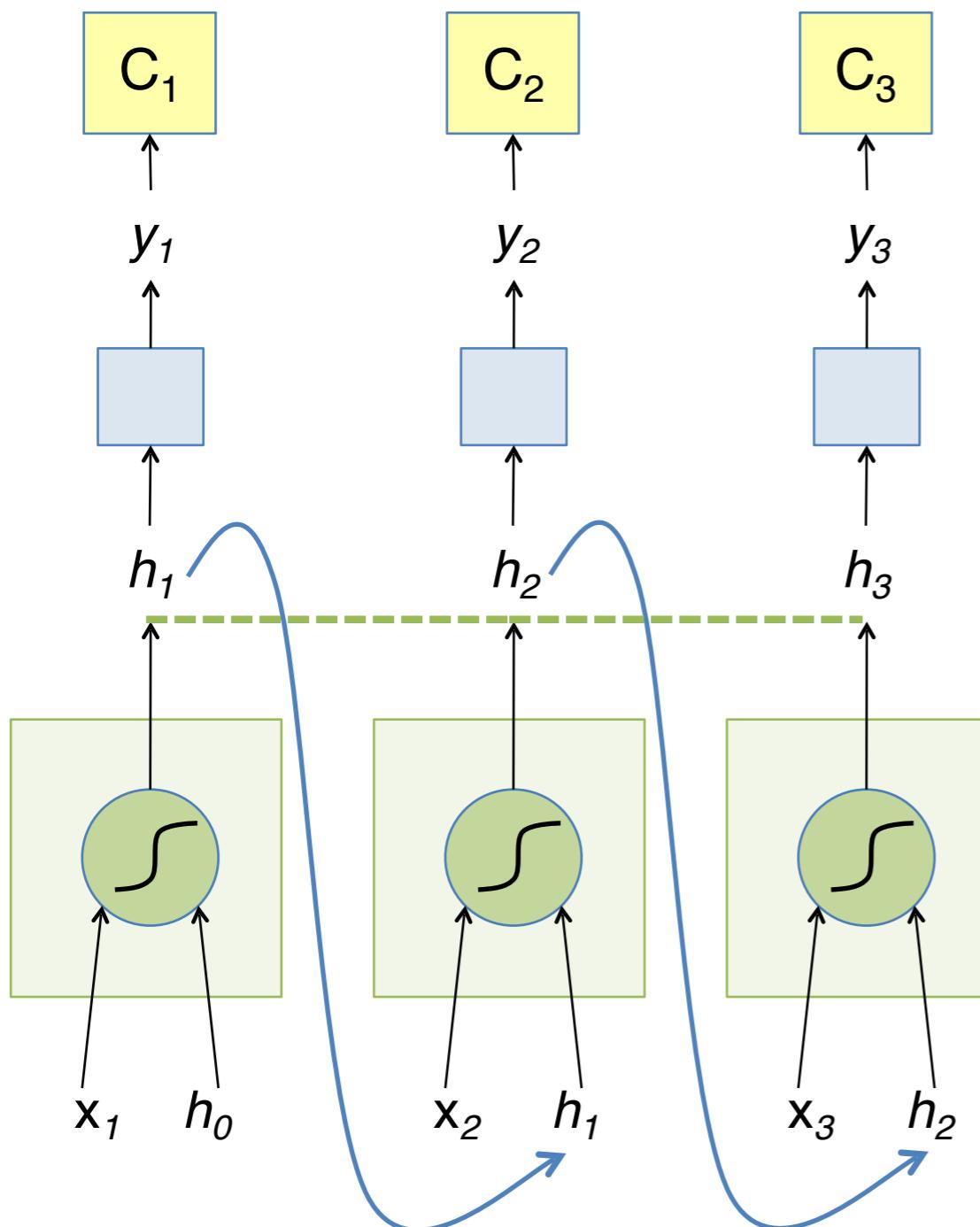


$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = F(h_t)$$

$$C_t = \text{Loss}(y_t, \text{GT}_t)$$

# The Vanilla RNN Forward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = F(h_t)$$

$$C_t = \text{Loss}(y_t, \text{GT}_t)$$

----- indicates shared weights

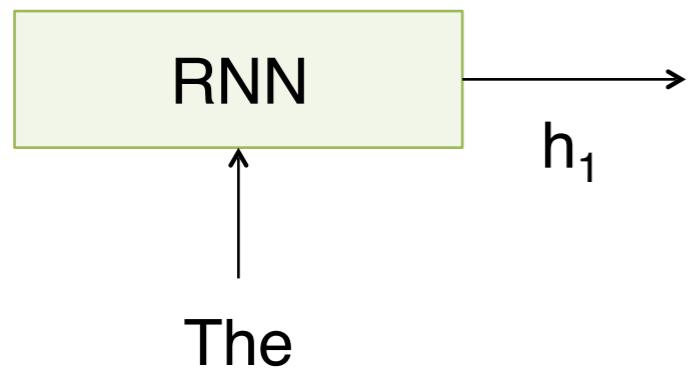
# Recurrent Neural Networks (RNNs)

- Note that the weights are shared over time
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps

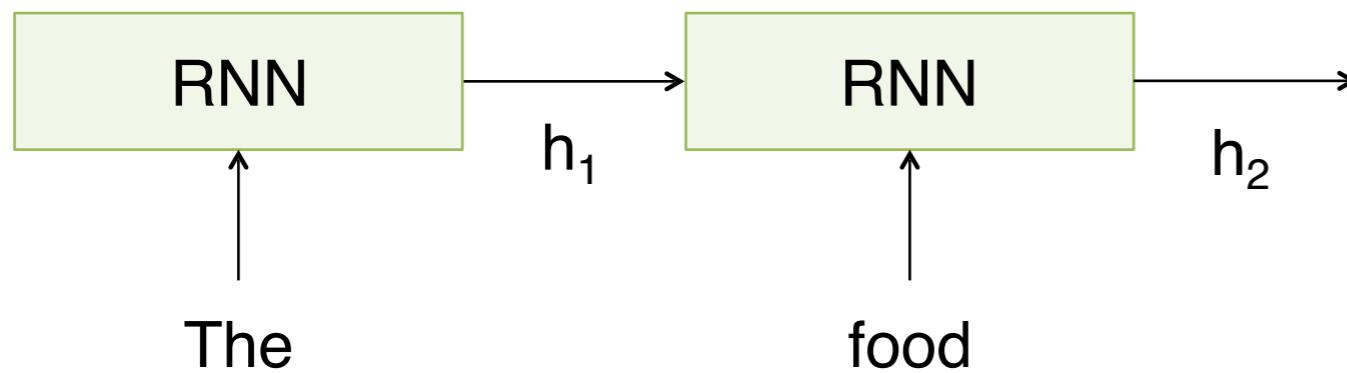
# Sentiment Classification

- Classify a restaurant review from Yelp! OR movie review from IMDB OR ... as positive or negative
- Inputs: Multiple words, one or more sentences
- Outputs: Positive / Negative classification
- “The food was really good”
- “The chicken crossed the road because it was uncooked”

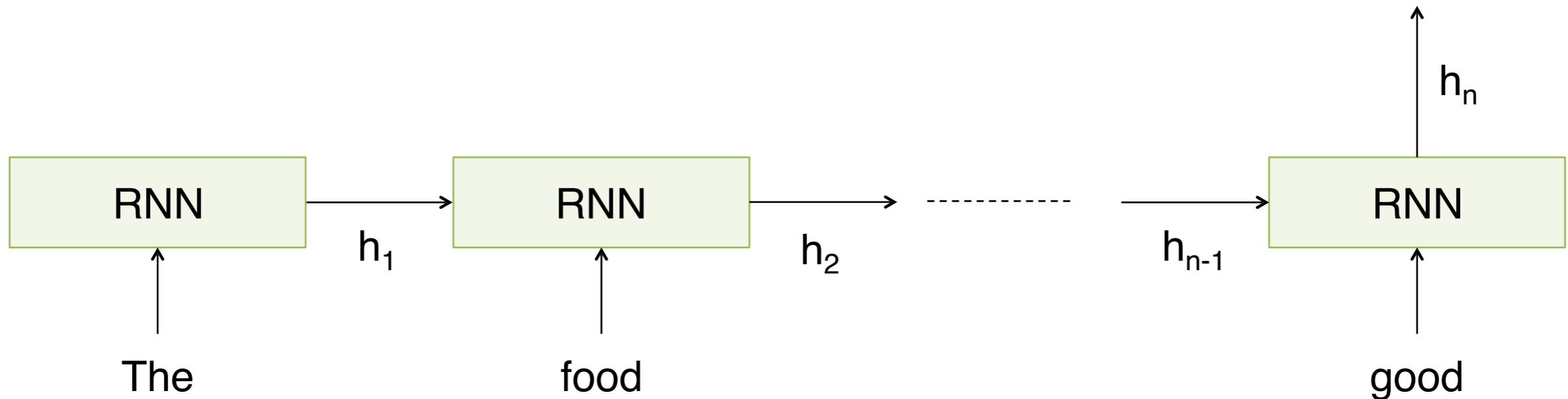
# Sentiment Classification



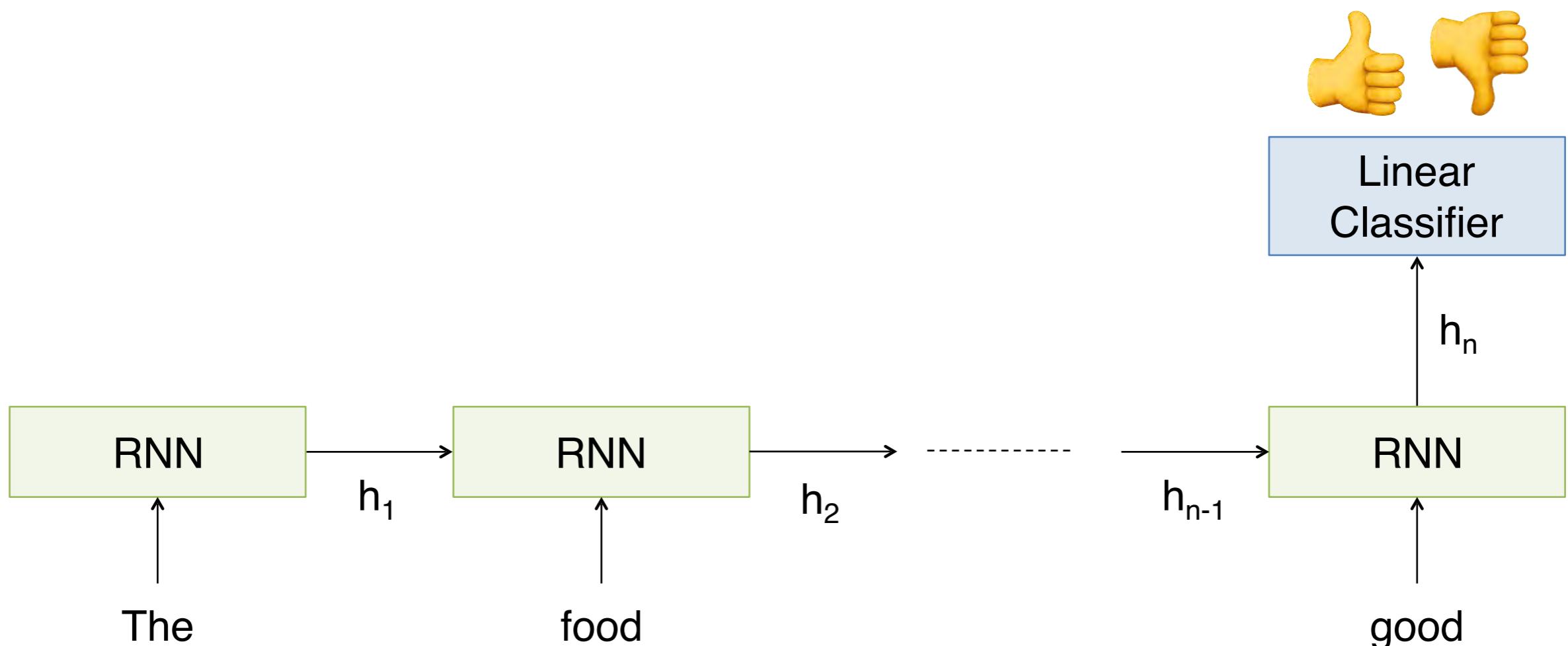
# Sentiment Classification



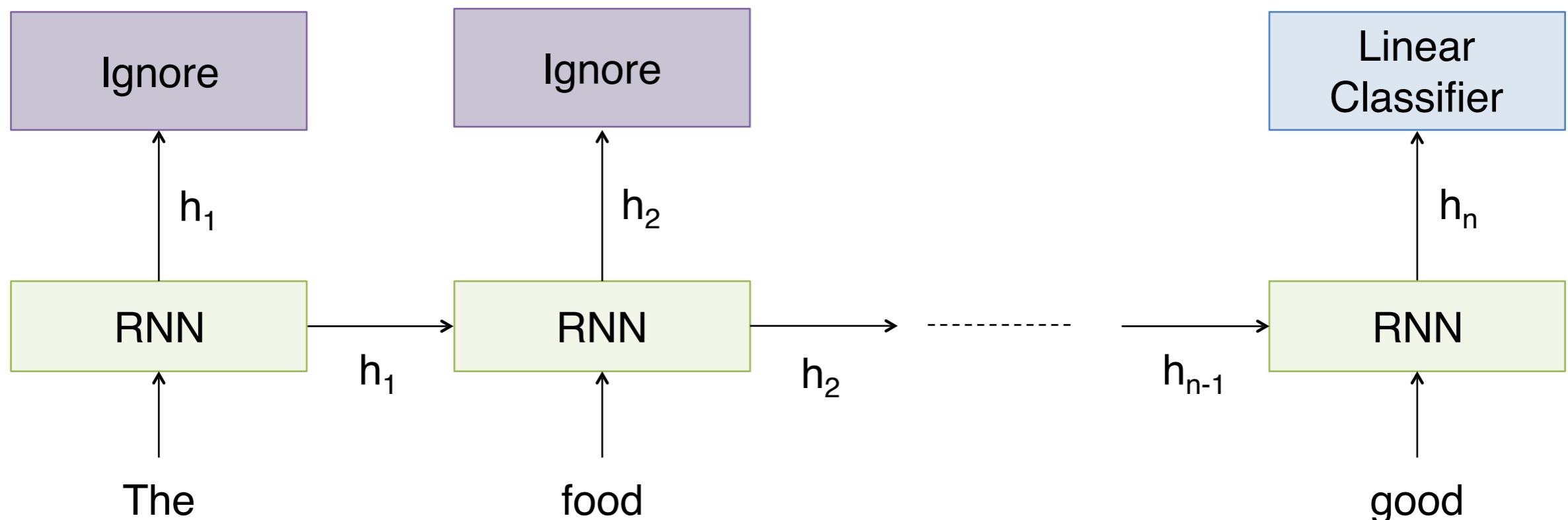
# Sentiment Classification



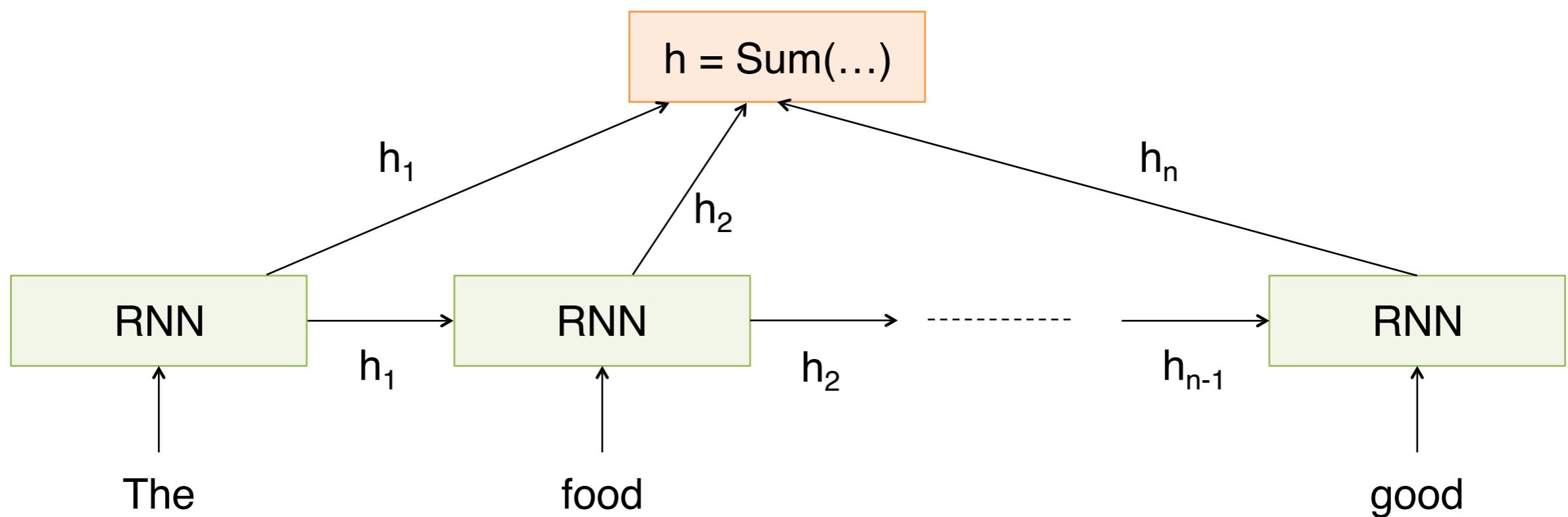
# Sentiment Classification



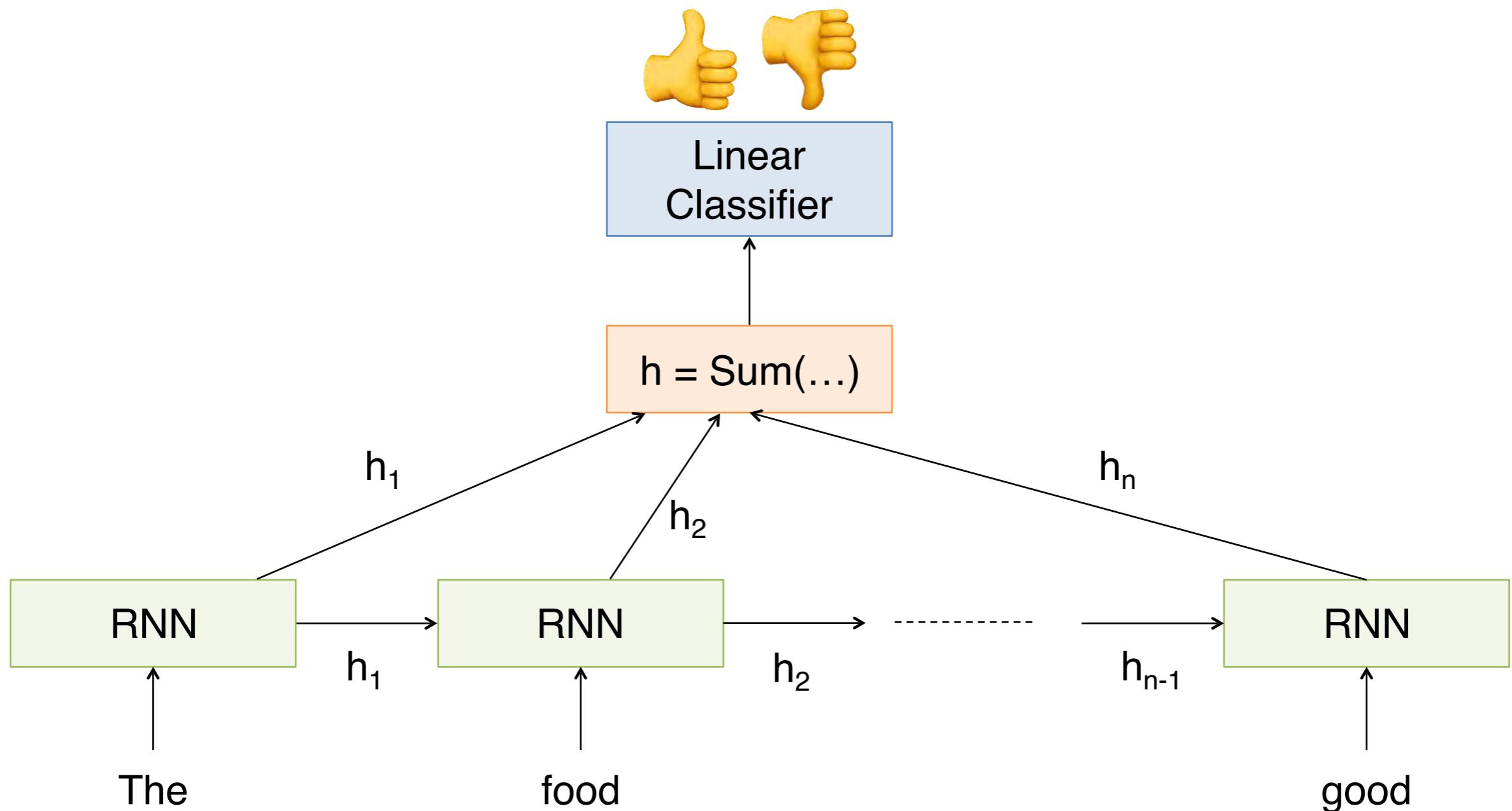
# Sentiment Classification



# Sentiment Classification

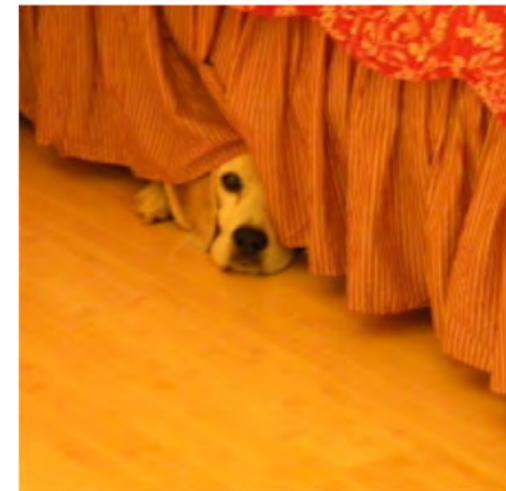


# Sentiment Classification

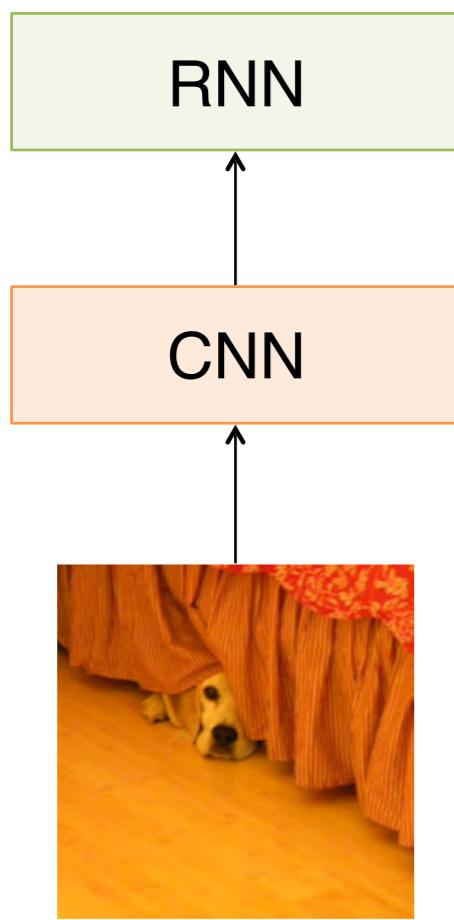


# Image Captioning

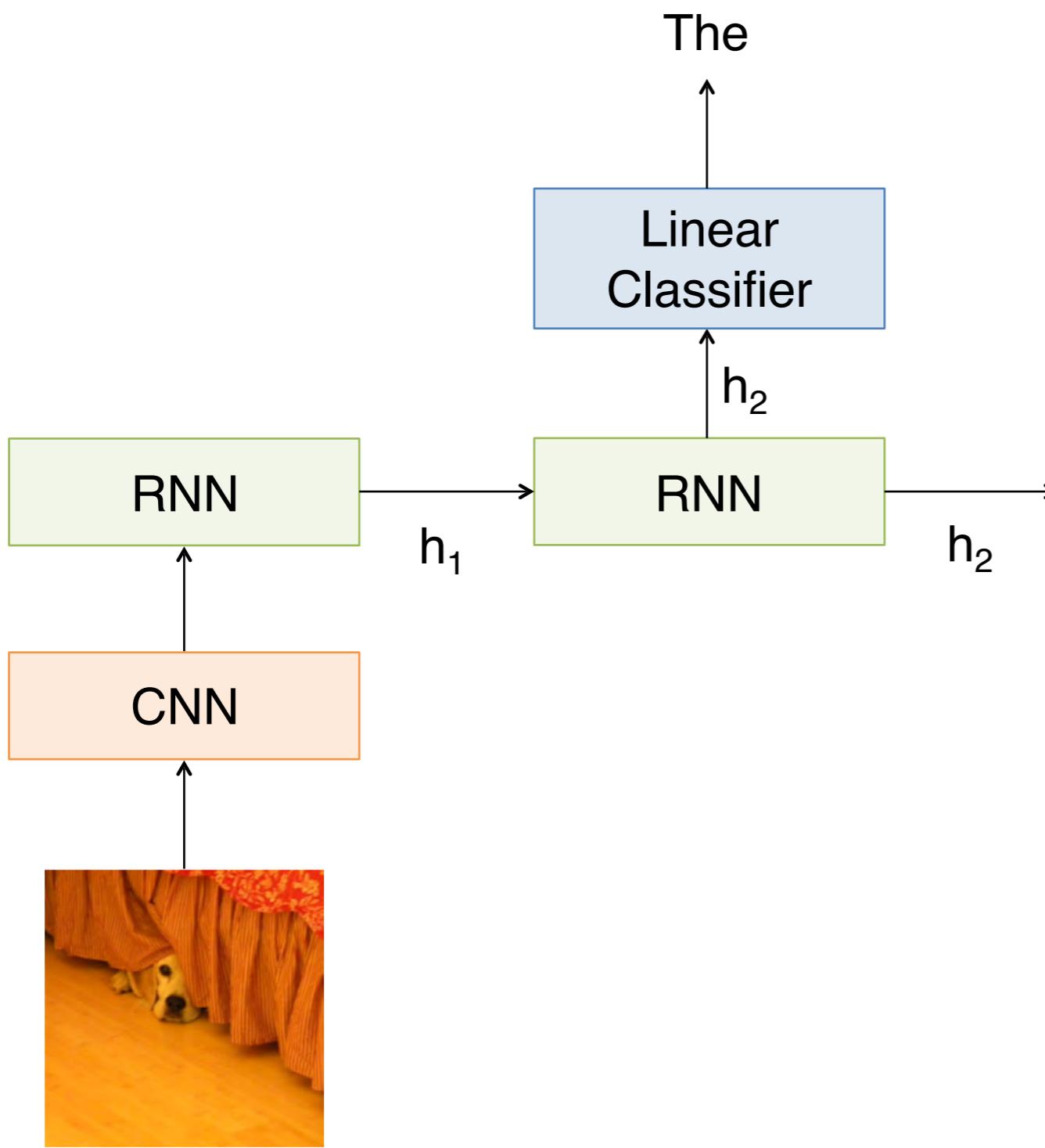
- Given an image, produce a sentence describing its contents
- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (let's consider one sentence)  
: The dog is hiding



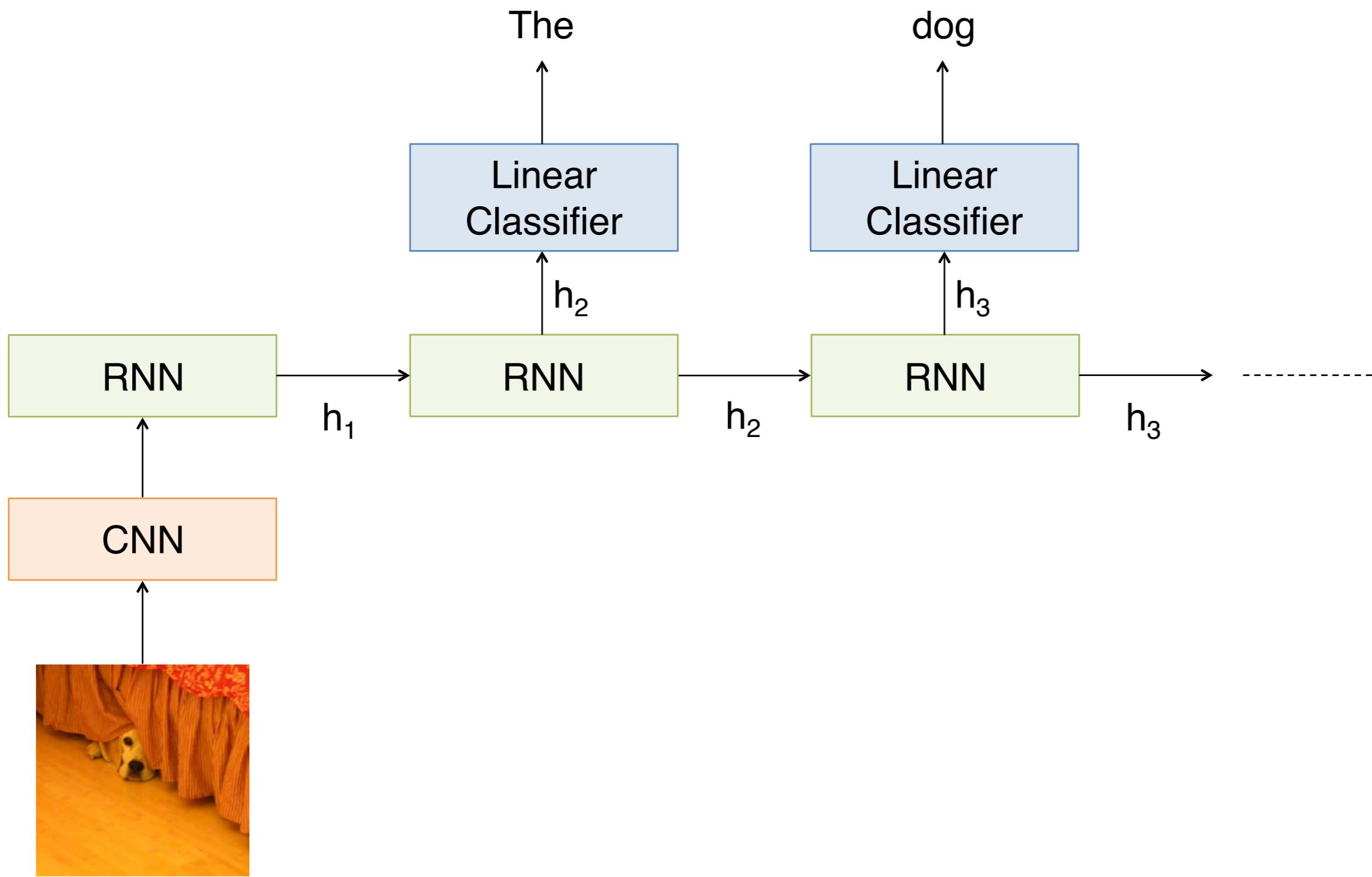
# Image Captioning



# Image Captioning



# Image Captioning



# RNN Outputs: Image Captions

A person riding a motorcycle on a dirt road.



Two dogs play in the grass.



A herd of elephants walking across a dry grass field.



A group of young people playing a game of frisbee.



Two hockey players are fighting over the puck.



A close up of a cat laying on a couch.



# RNN Outputs: Language Modeling

VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not aps, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are  
hand,  
That Caesar and my goodly father's world;  
When I was heaven of presence and our fleets,  
We spare with hours, but cut thy council I am great,  
Murdered and by thy master's ready there  
My power to give thee but so much as hell:  
Some service in the noble bondman here,  
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the  
courtesy of your law,  
Your sight and several breath, will  
wear the gods  
With his heads, and my hands are  
wonder'd at the deeds,  
So drop upon your lordship's head,  
and your opinion  
Shall be against your honour.

# Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

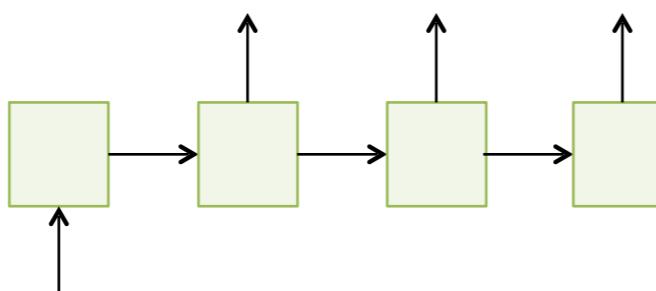
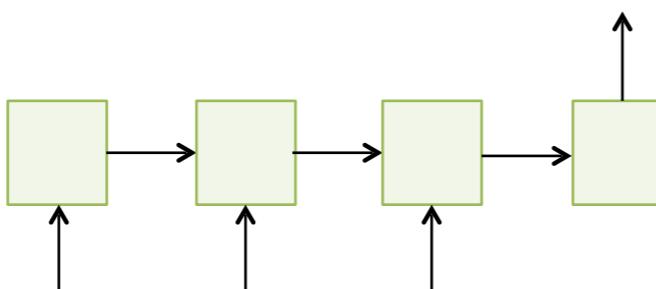


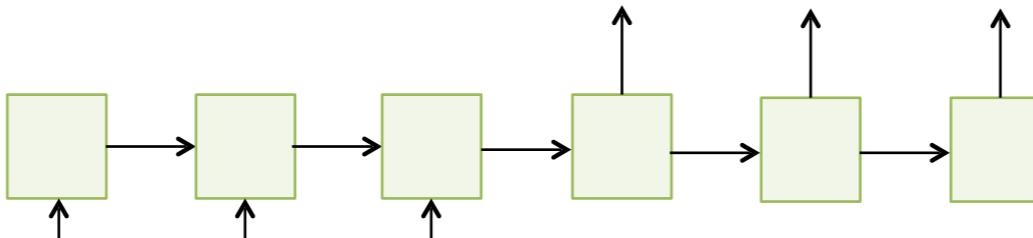
Image Captioning

Multiple - Single



Sentiment Classification

Multiple - Multiple



Translation

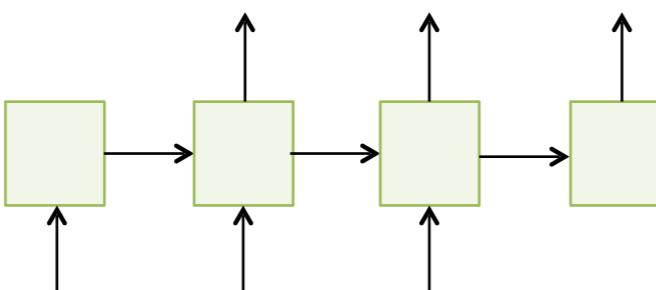


Image Captioning

# Input – Output Scenarios

Note: We might deliberately choose to frame our problem as a

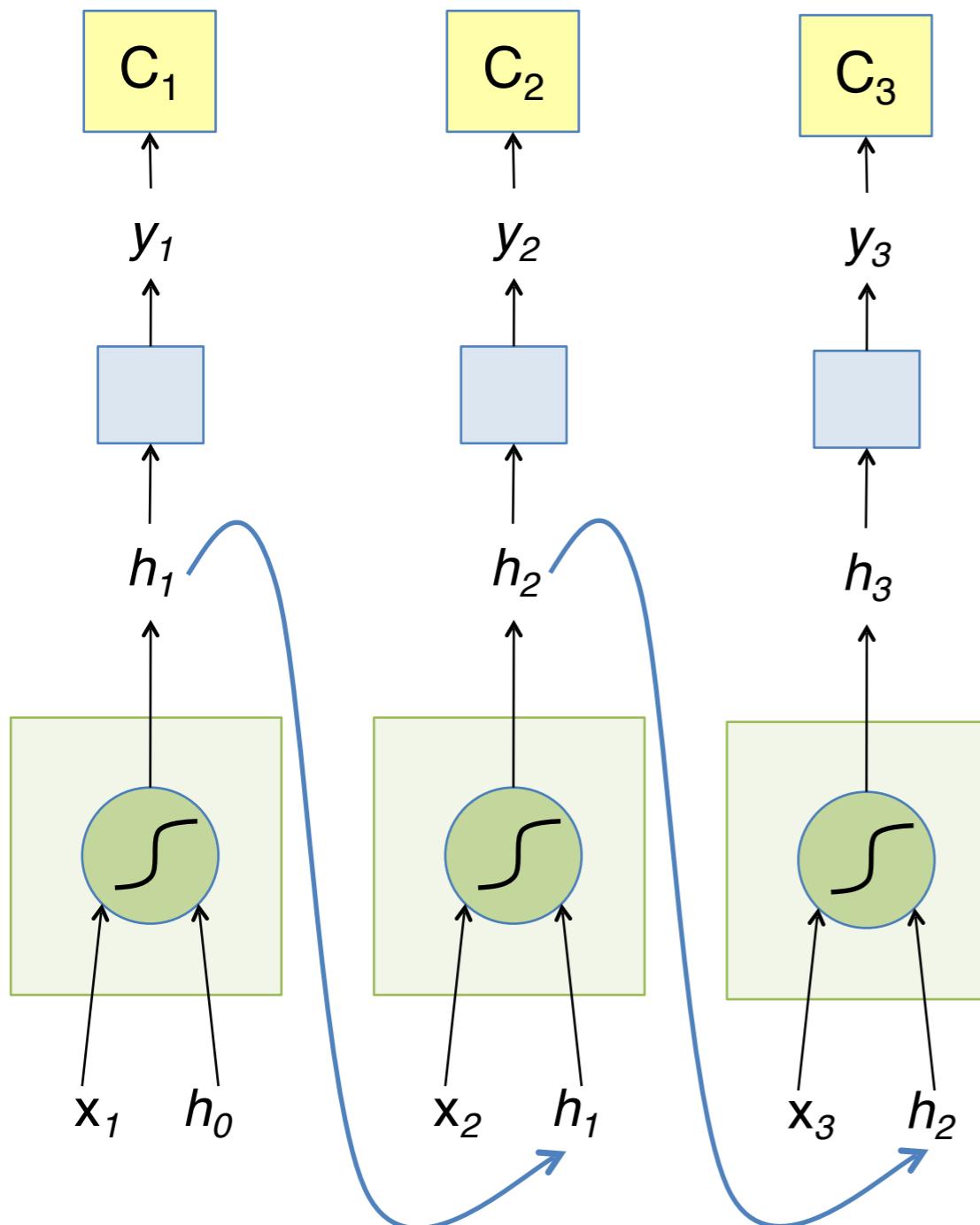
particular input-output scenario for ease of training or better performance.

For example, at each time step, provide previous word as

input for image captioning

(Single-Multiple to Multiple-Multiple).

# The Vanilla RNN Forward



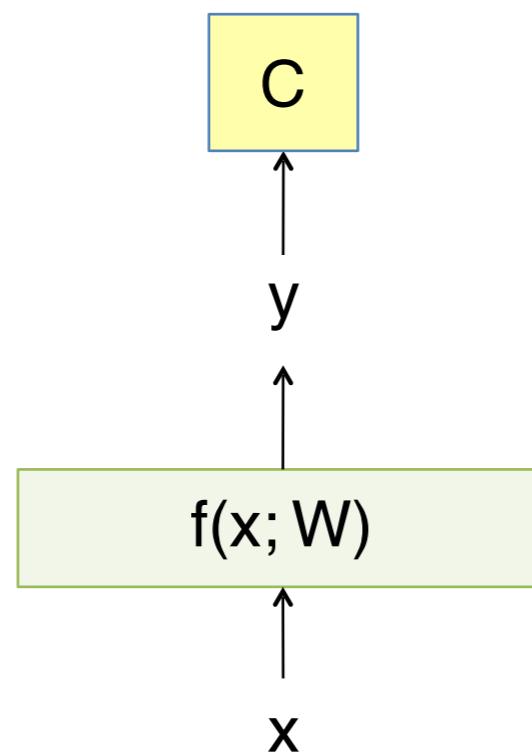
$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = F(h_t)$$

$$C_t = \text{Loss}(y_t, \text{GT}_t)$$

“Unfold” network through time by making copies at each time-step

# BackPropagation Refresher



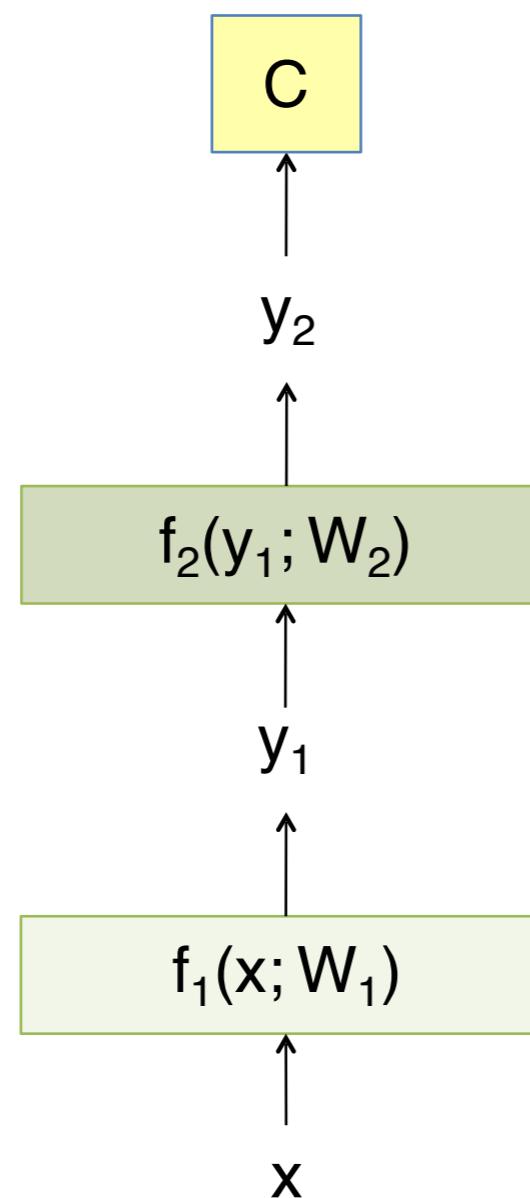
$$y = f(x; W)$$
$$C = \text{Loss}(y, y_{GT})$$

SGD Update

$$W \leftarrow W - \eta \frac{\partial C}{\partial W}$$

$$\frac{\partial C}{\partial W} = \left( \frac{\partial C}{\partial y} \right) \left( \frac{\partial y}{\partial W} \right)$$

# Multiple Layers



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

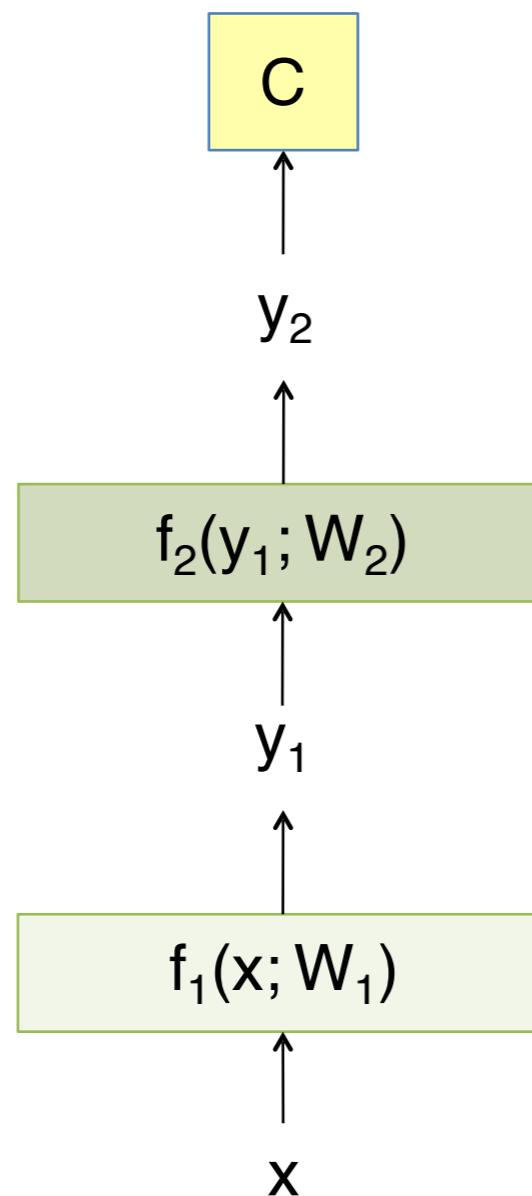
$$C = \text{Loss}(y_2, y_{GT})$$

SGD Update

$$W_2 \leftarrow W_2 - \eta \frac{\partial C}{\partial W_2}$$

$$W_1 \leftarrow W_1 - \eta \frac{\partial C}{\partial W_1}$$

# Chain Rule for Gradient Computation



$$y_1 = f_1(x; W_1)$$

$$y_2 = f_2(y_1; W_2)$$

$$C = \text{Loss}(y_2, y_{GT})$$

Find  $\frac{\partial C}{\partial W_1}, \frac{\partial C}{\partial W_2}$

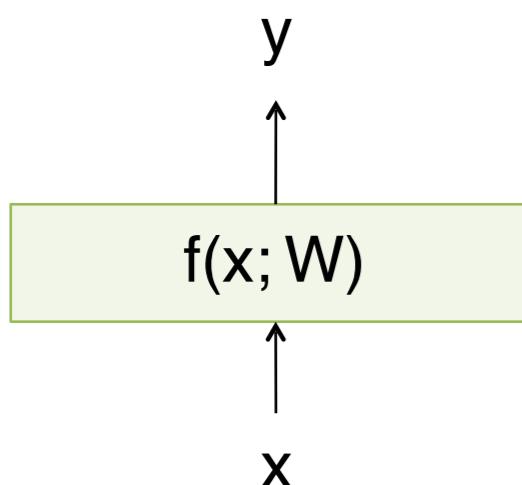
$$\frac{\partial C}{\partial W_2} = \left( \frac{\partial C}{\partial y_2} \right) \left( \frac{\partial y_2}{\partial W_2} \right)$$

$$\frac{\partial C}{\partial W_1} = \left( \frac{\partial C}{\partial y_1} \right) \left( \frac{\partial y_1}{\partial W_1} \right)$$

$$= \left( \frac{\partial C}{\partial y_2} \right) \left( \frac{\partial y_2}{\partial y_1} \right) \left( \frac{\partial y_1}{\partial W_1} \right)$$

Application of the Chain Rule

# Chain Rule for Gradient Computation



Given:  $\left( \frac{\partial C}{\partial y} \right)$

We are interested in computing:  $\left( \frac{\partial C}{\partial W} \right), \left( \frac{\partial C}{\partial x} \right)$

Intrinsic to the layer are:

$\left( \frac{\partial y}{\partial W} \right)$  - How does output change due to params

$\left( \frac{\partial y}{\partial x} \right)$  - How does output change due to inputs

$$\left( \frac{\partial C}{\partial W} \right) = \left( \frac{\partial C}{\partial y} \right) \left( \frac{\partial y}{\partial W} \right) \quad \left( \frac{\partial C}{\partial x} \right) = \left( \frac{\partial C}{\partial y} \right) \left( \frac{\partial y}{\partial x} \right)$$

# Chain Rule for Gradient Computation

$$\left( \frac{\partial C}{\partial y} \right) \downarrow f(x; W)$$

$$\left( \frac{\partial C}{\partial x} \right)$$

Given:  $\left( \frac{\partial C}{\partial y} \right)$

We are interested in computing:  $\left( \frac{\partial C}{\partial W} \right), \left( \frac{\partial C}{\partial x} \right)$

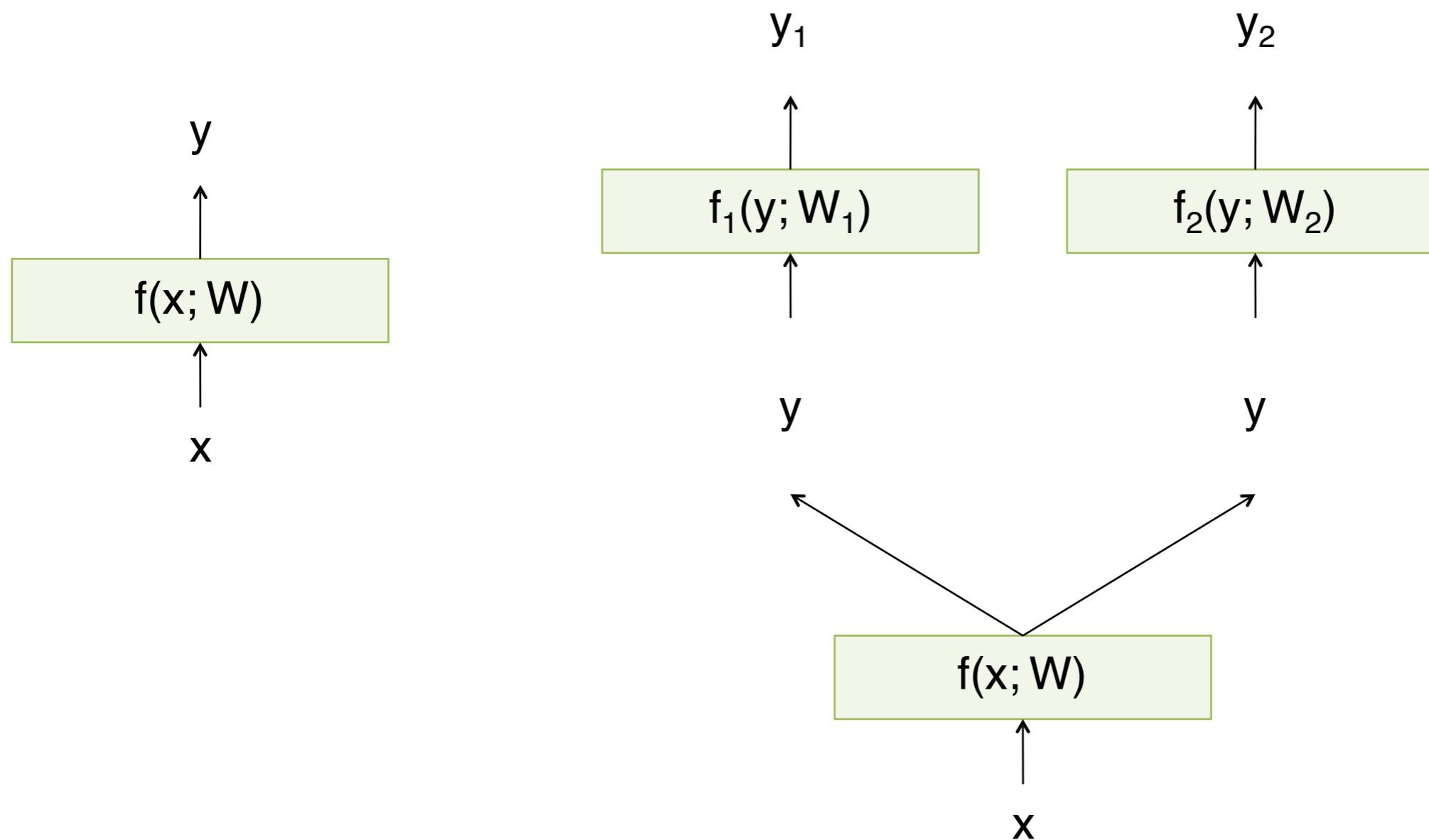
Intrinsic to the layer are:

$\left( \frac{\partial y}{\partial W} \right)$  - How does output change due to params

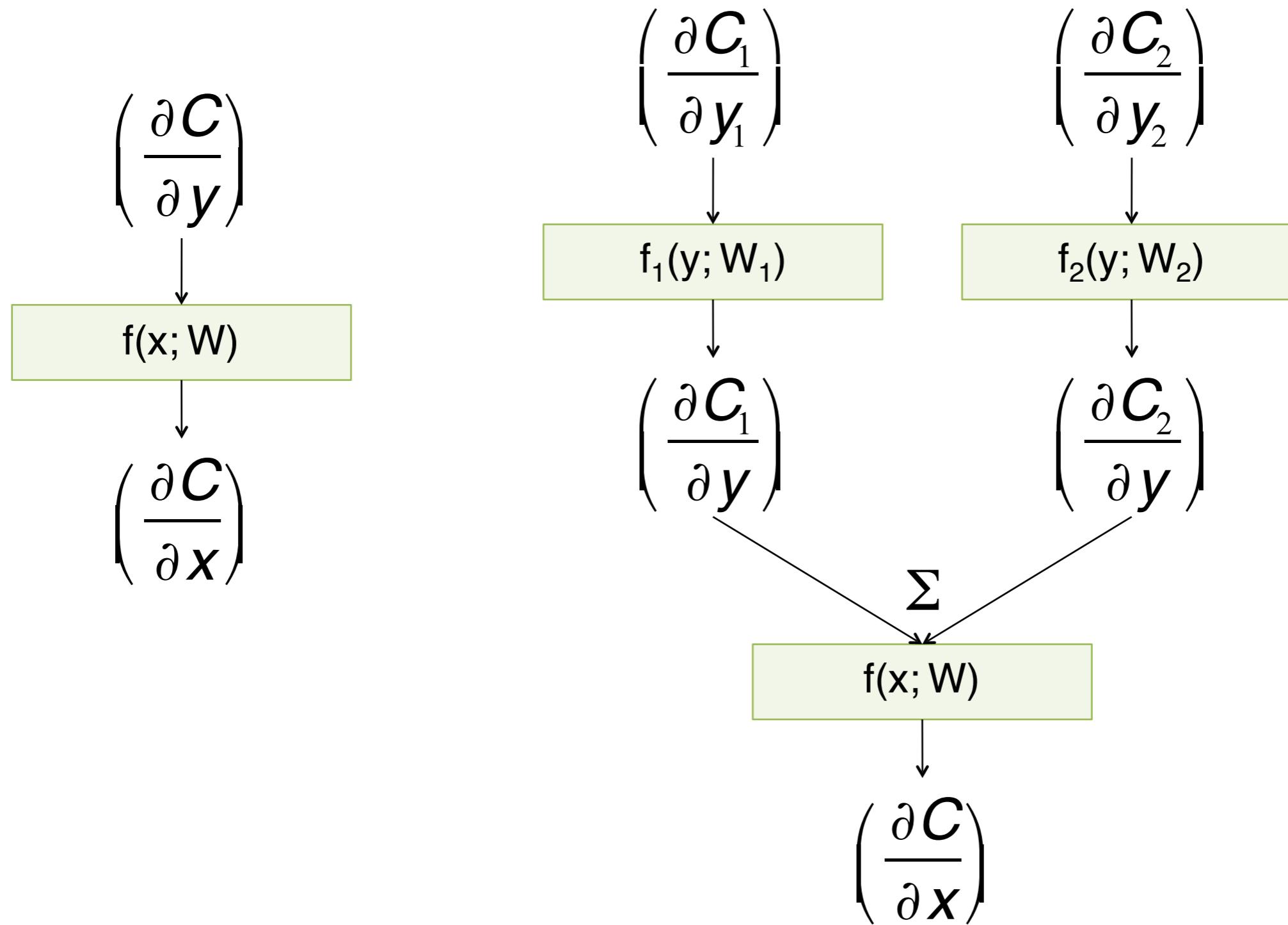
$\left( \frac{\partial y}{\partial x} \right)$  - How does output change due to inputs

$$\left( \frac{\partial C}{\partial W} \right) = \left( \frac{\partial C}{\partial y} \right) \left( \frac{\partial y}{\partial W} \right) \quad \left( \frac{\partial C}{\partial x} \right) = \left( \frac{\partial C}{\partial y} \right) \left( \frac{\partial y}{\partial x} \right)$$

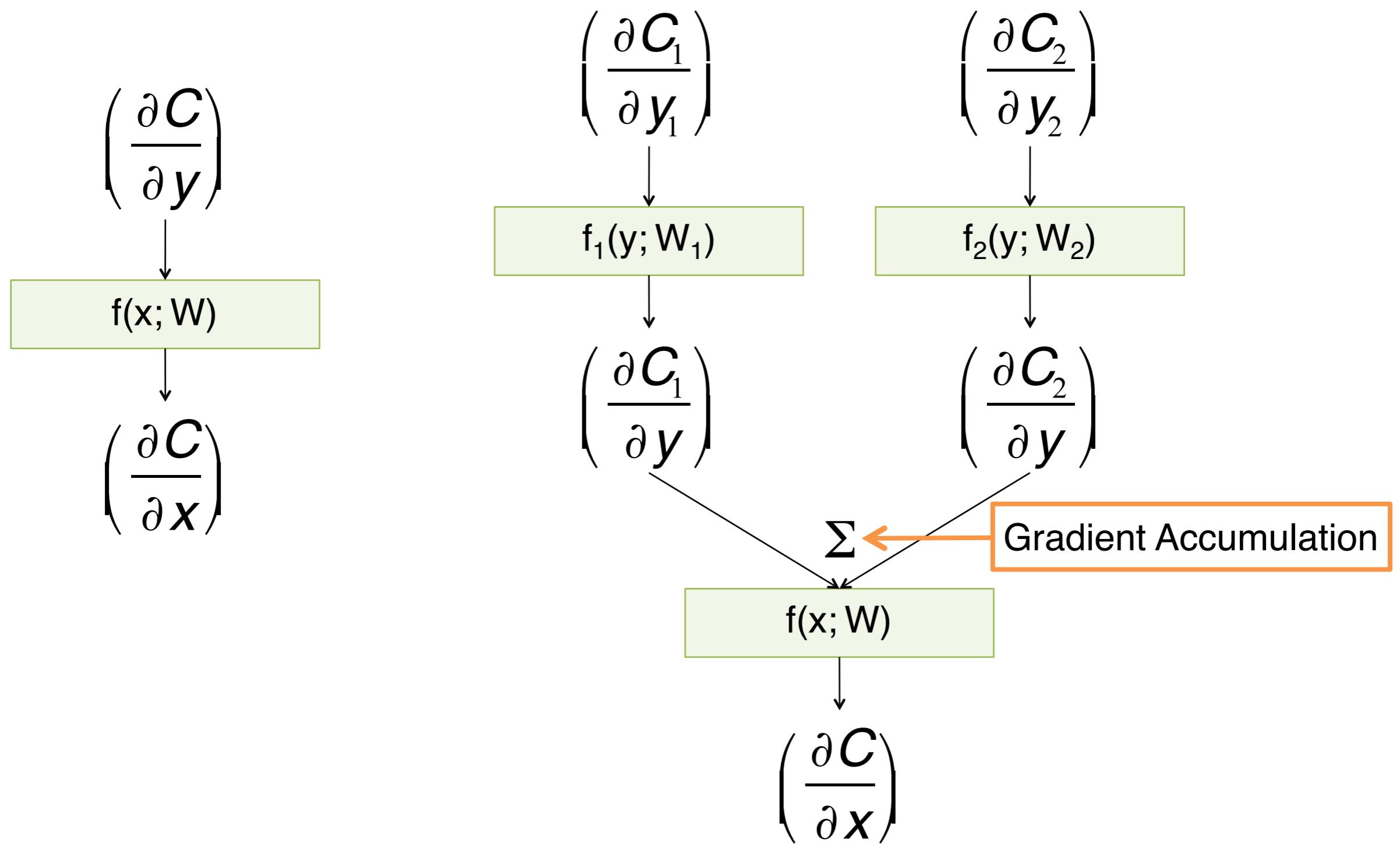
# Extension to Computational Graphs



# Extension to Computational Graphs



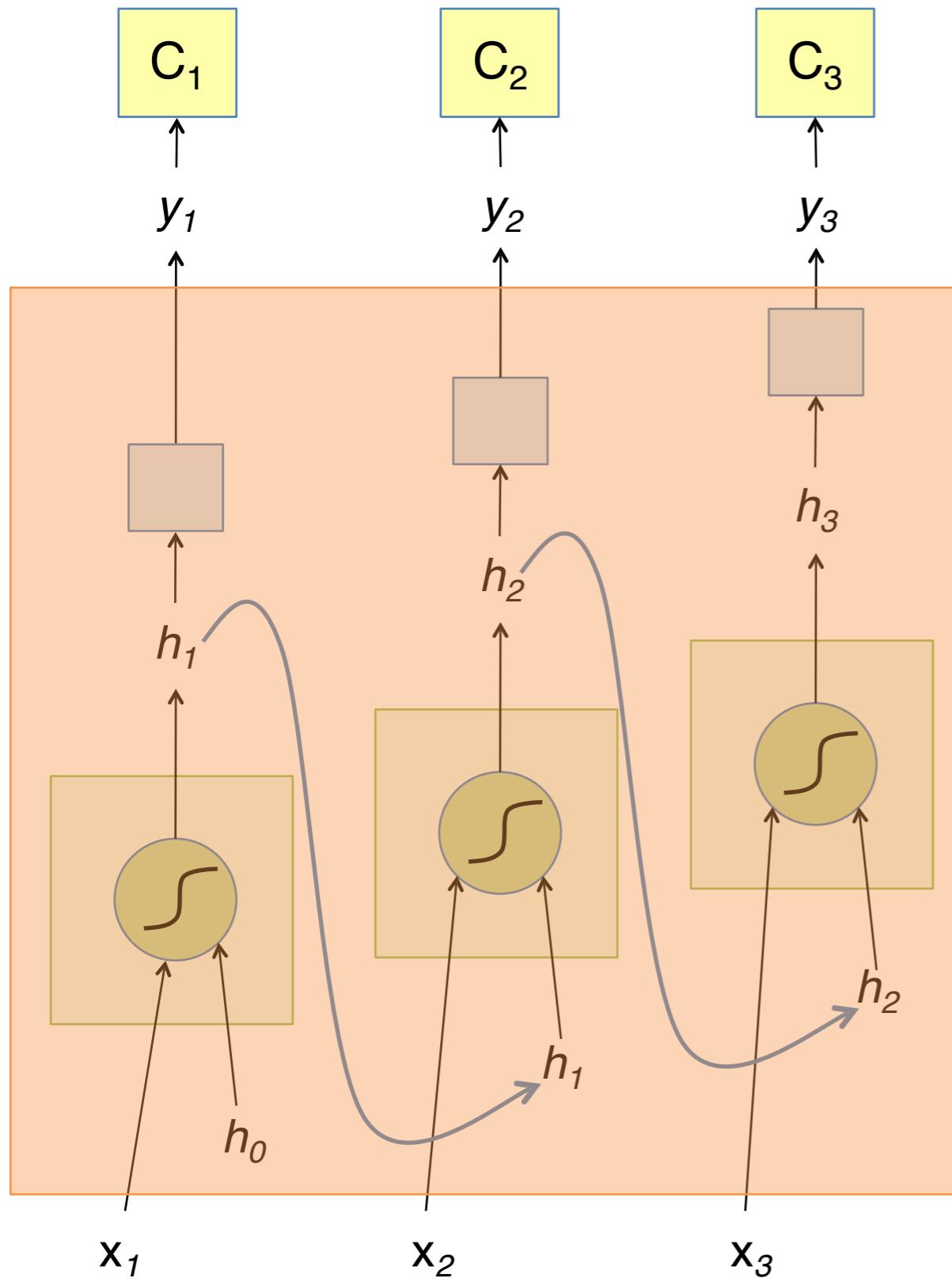
# Extension to Computational Graphs



# Back Propagation Through Time (BPTT)

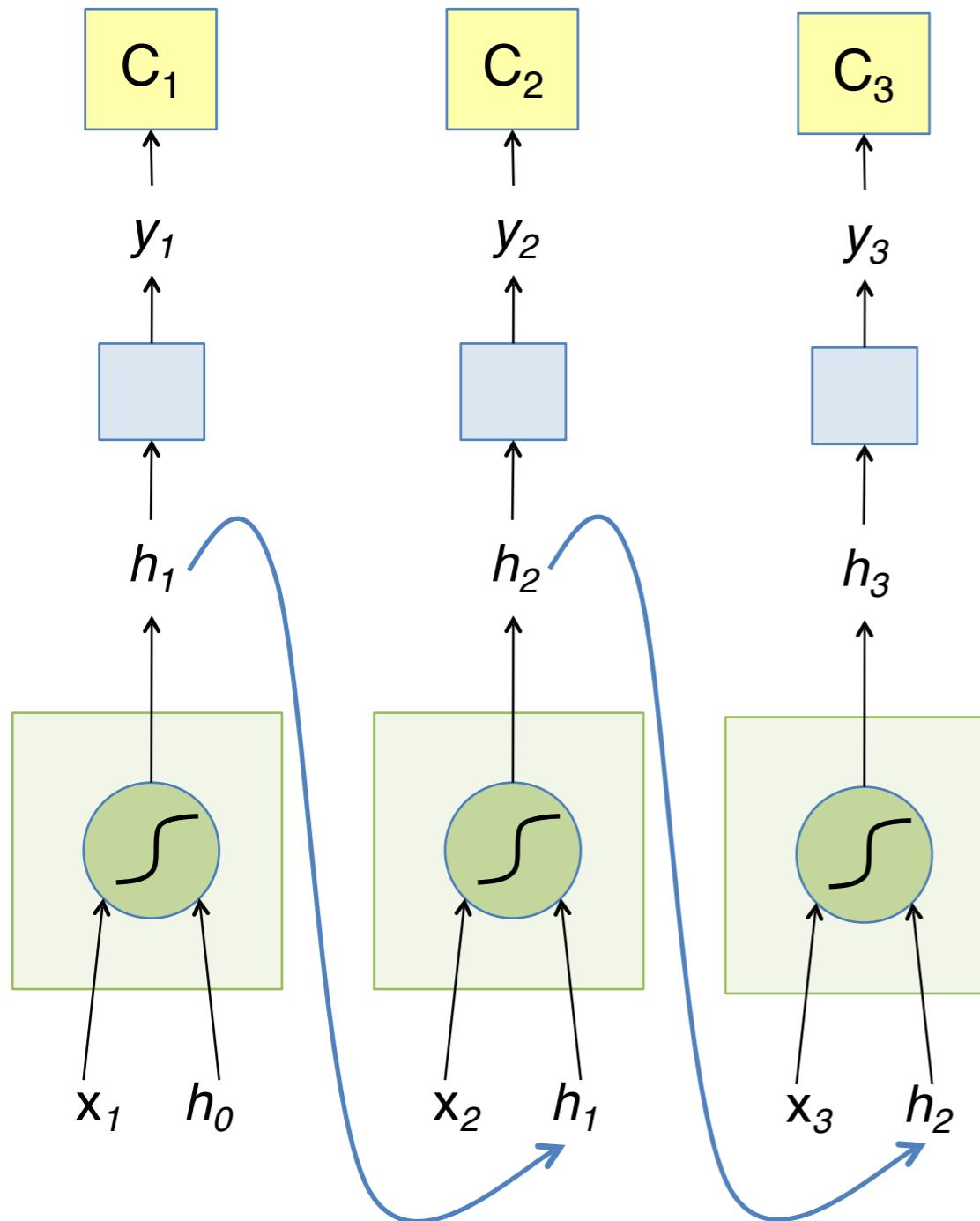
- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights

# The Unfolded Vanilla RNN

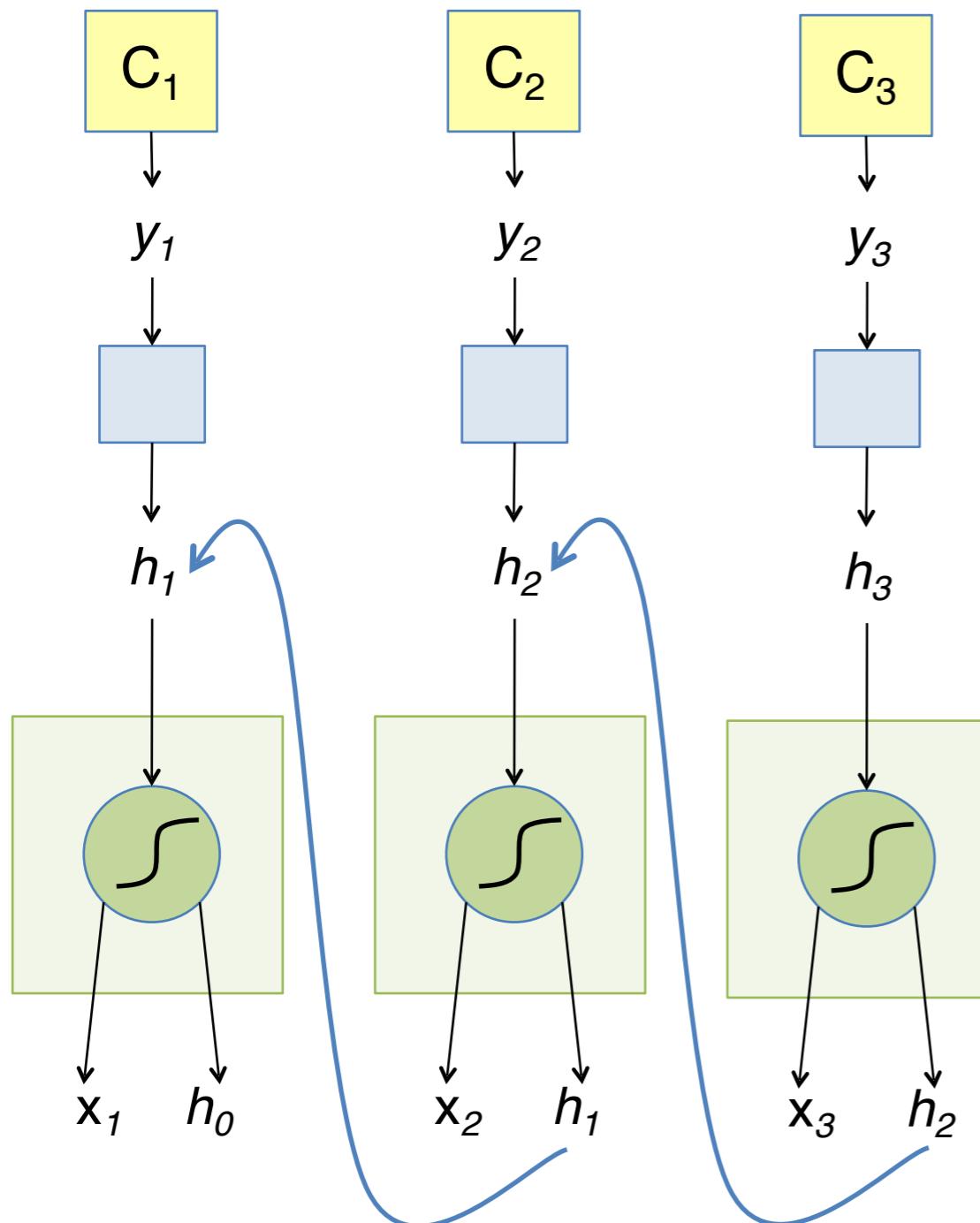


- Treat the unfolded network as one big feed-forward network!
- This big network takes in entire sequence as an input
- Compute gradients through the usual backpropagation
- Update shared weights

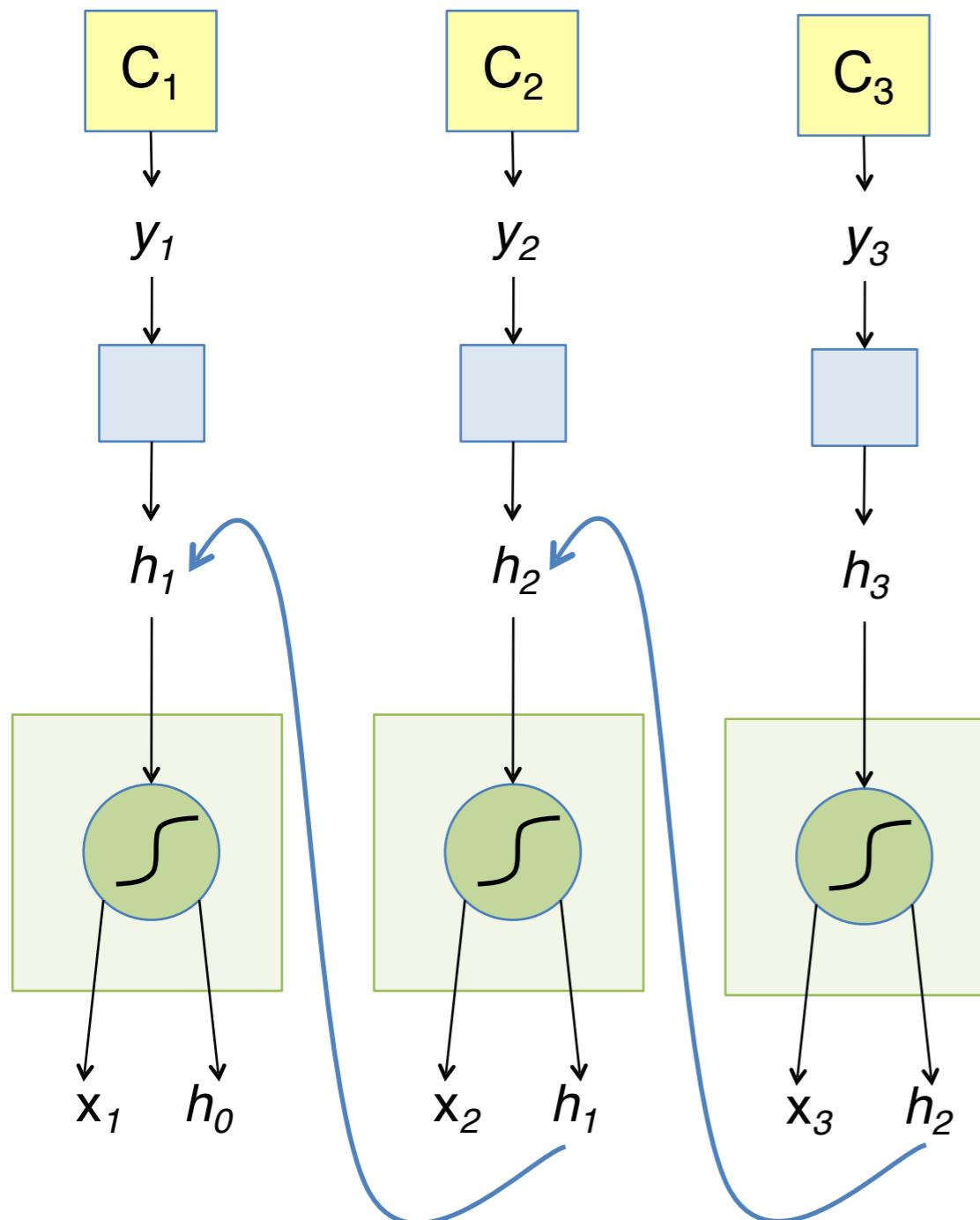
# The Unfolded Vanilla RNN Forward



# The Unfolded Vanilla RNN Backward



# The Vanilla RNN Backward



$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$y_t = F(h_t)$$

$$C_t = \text{Loss}(y_t, \text{GT}_t)$$

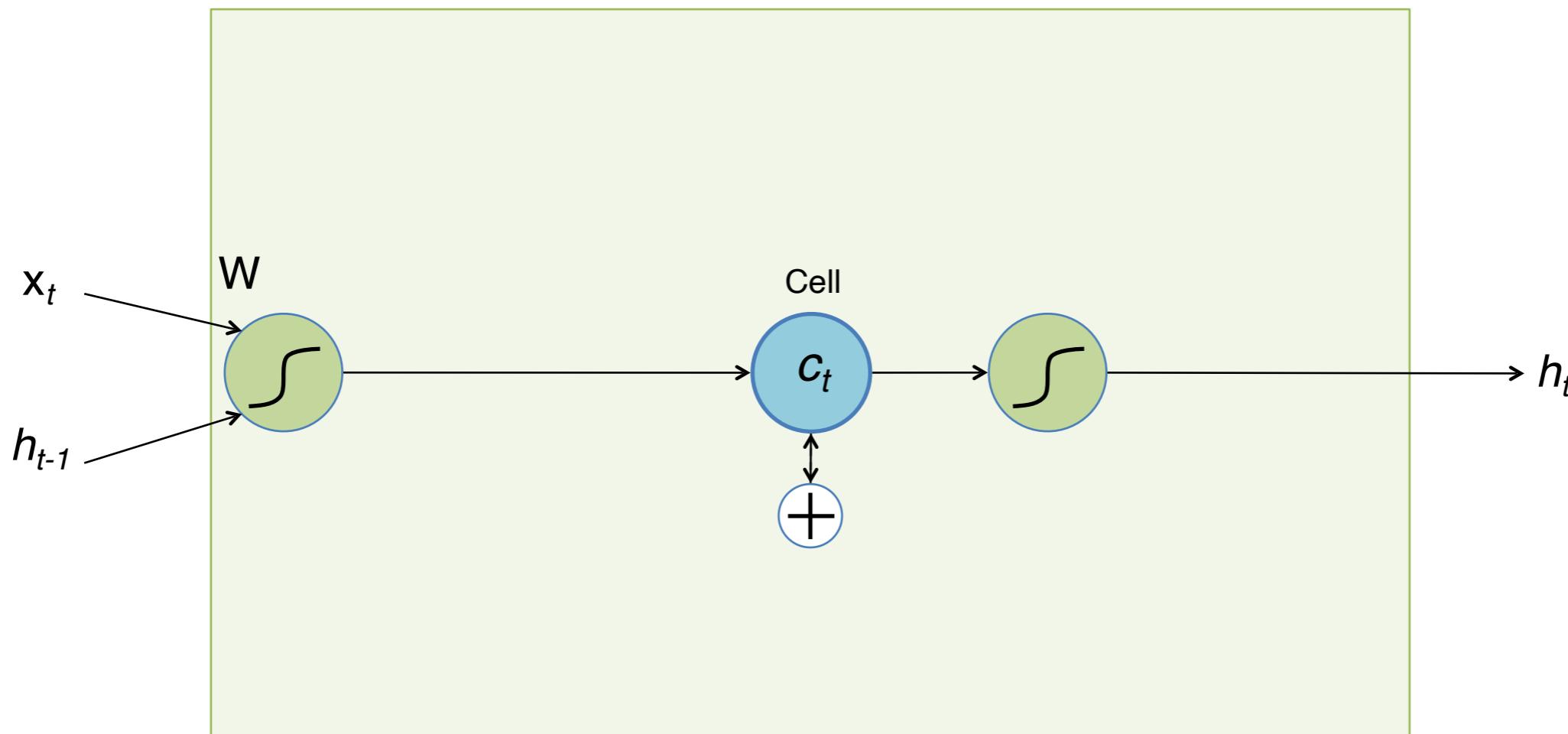
$$\begin{aligned} \frac{\partial C_t}{\partial h_1} &= \left( \frac{\partial C_t}{\partial y_t} \right) \left( \frac{\partial y_t}{\partial h_1} \right) \\ &= \left( \frac{\partial C_t}{\partial y_t} \right) \left( \frac{\partial y_t}{\partial h_t} \right) \left( \frac{\partial h_t}{\partial h_{t-1}} \right) \dots \left( \frac{\partial h_2}{\partial h_1} \right) \end{aligned}$$

# Long Short-Term Memory (LSTM)<sup>1</sup>

- The LSTM uses this idea of “Constant Error Flow” for RNNs to create a “Constant Error Carousel” (CEC) which ensures that gradients don’t decay
- The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time
- Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved

<sup>1</sup> [Long Short-Term Memory, Hochreiter et al., 1997](#)

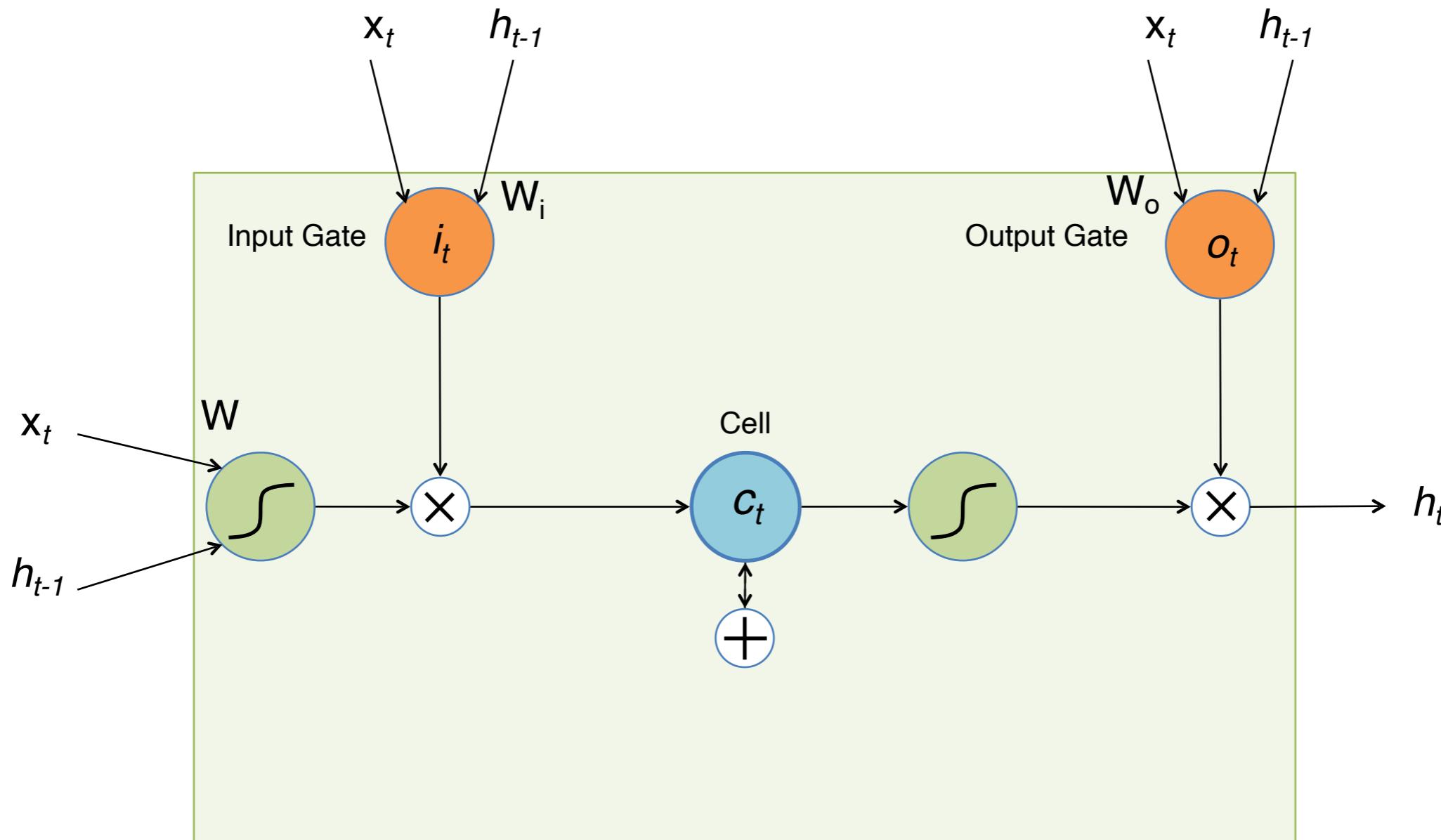
# The LSTM Idea



$$c_t = c_{t-1} + \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \quad h_t = \tanh c_t$$

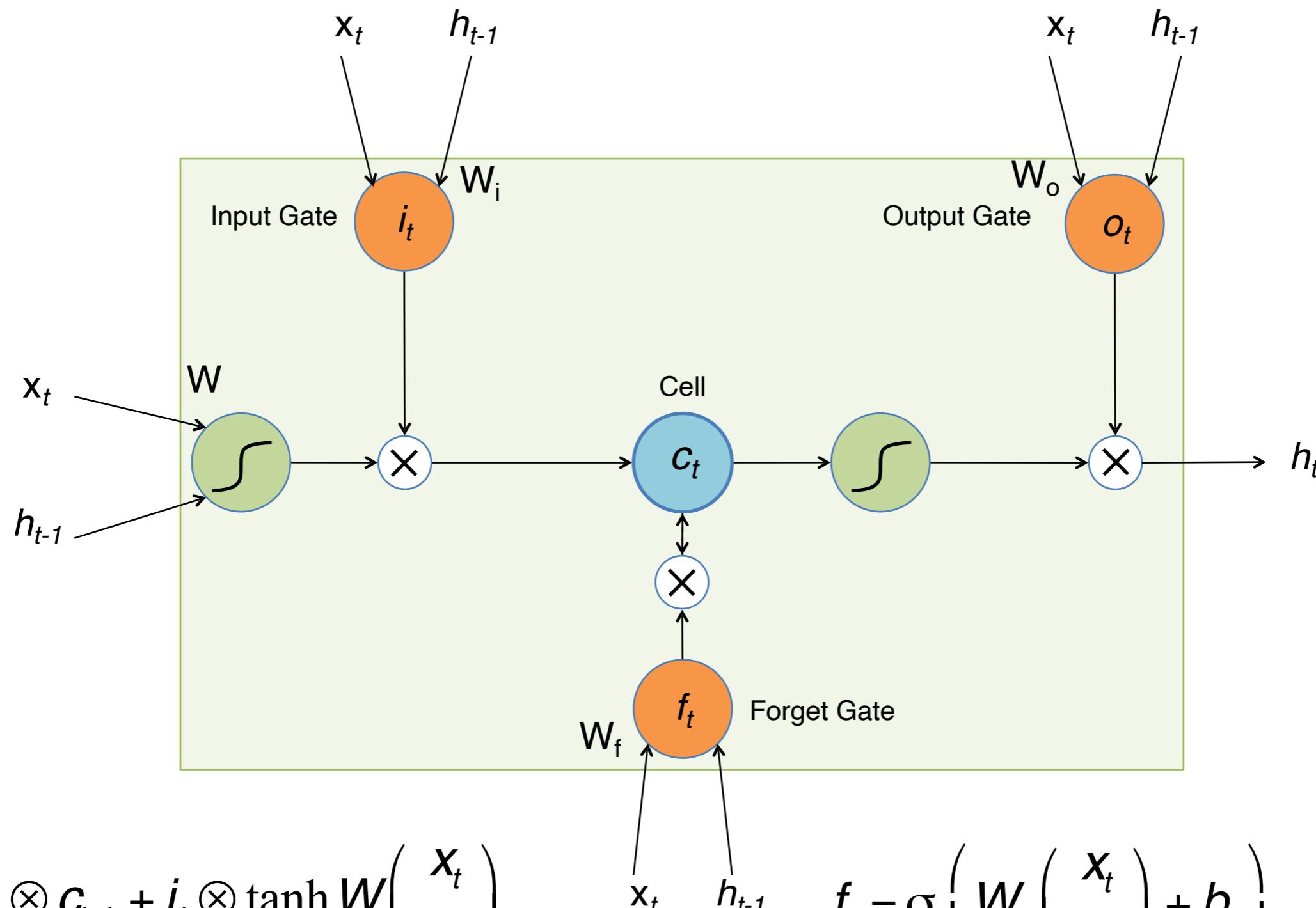
\* Dashed line indicates time-lag

# The Original LSTM Cell



$$c_t = c_{t-1} + i_t \otimes \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \quad h_t = o_t \otimes \tanh c_t \quad i_t = \sigma \left( W_i \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} + b_i \right) \quad \text{Similarly for } o_t$$

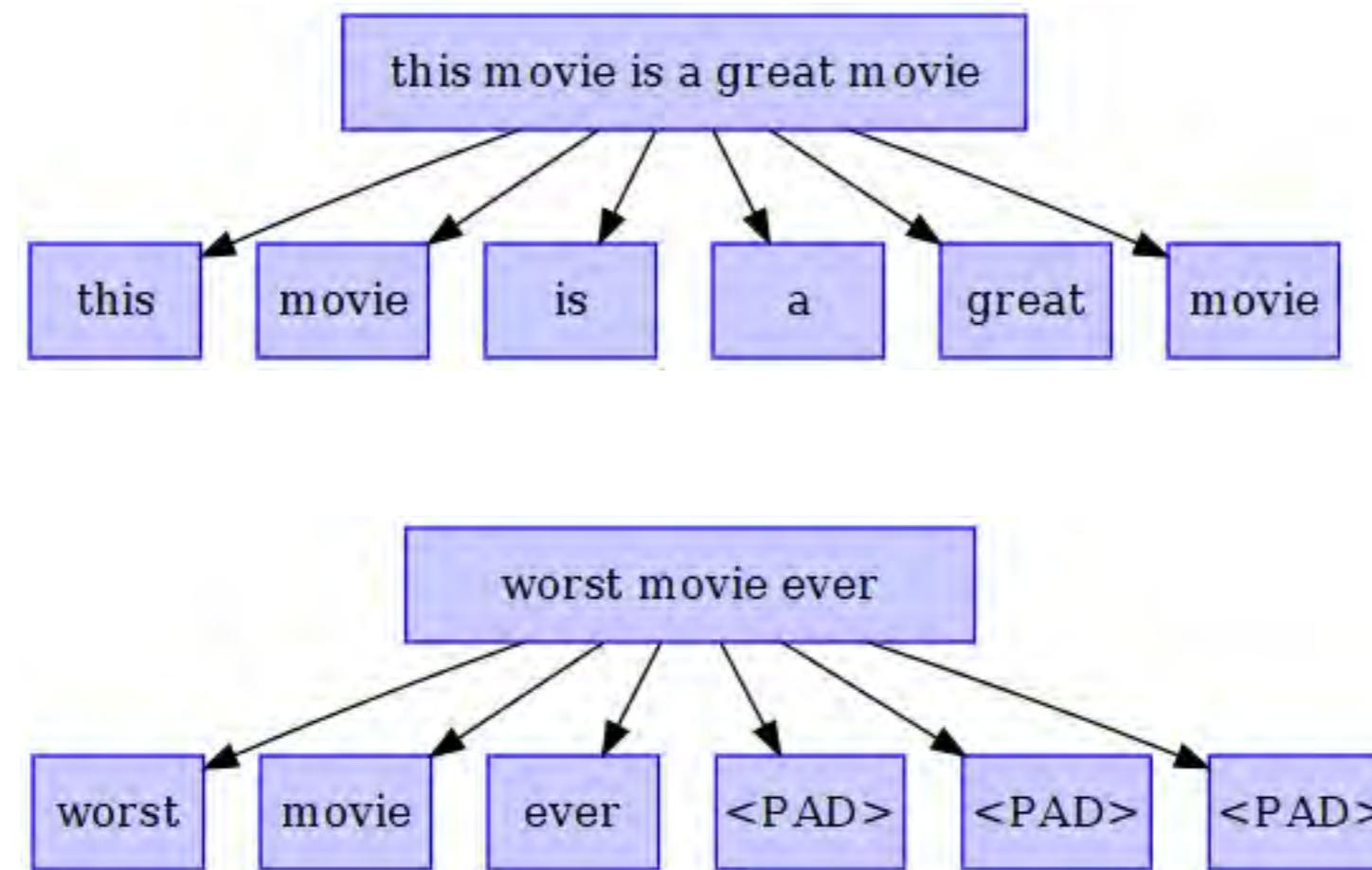
# The Popular LSTM Cell

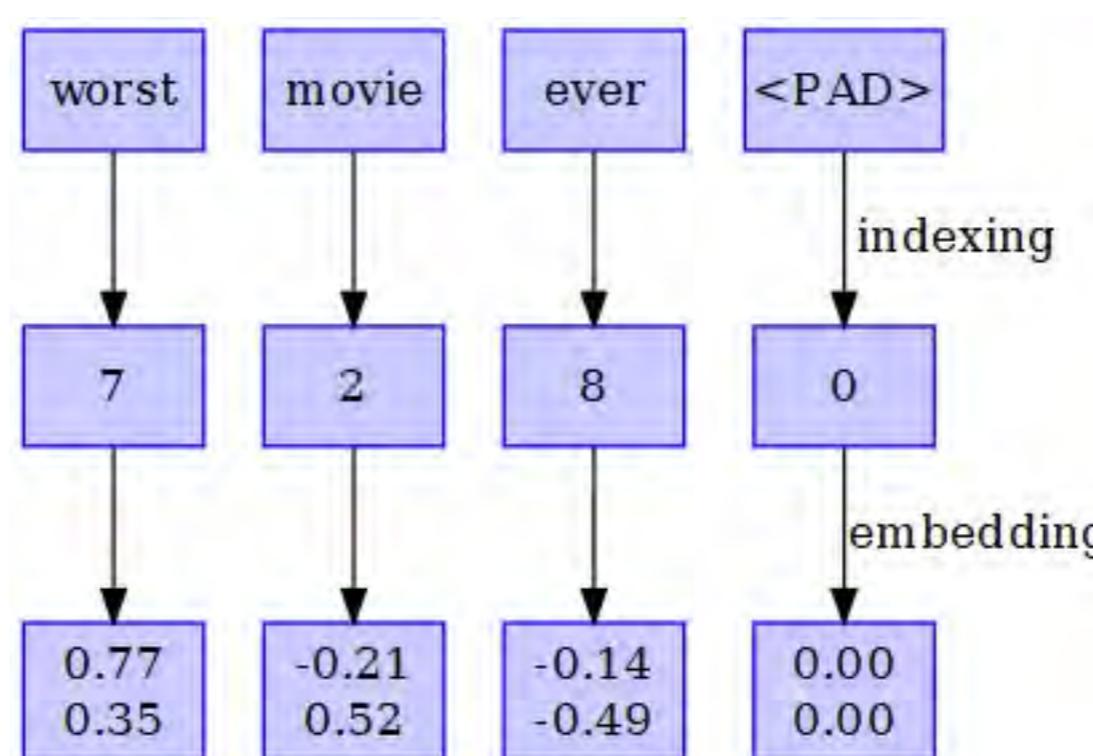
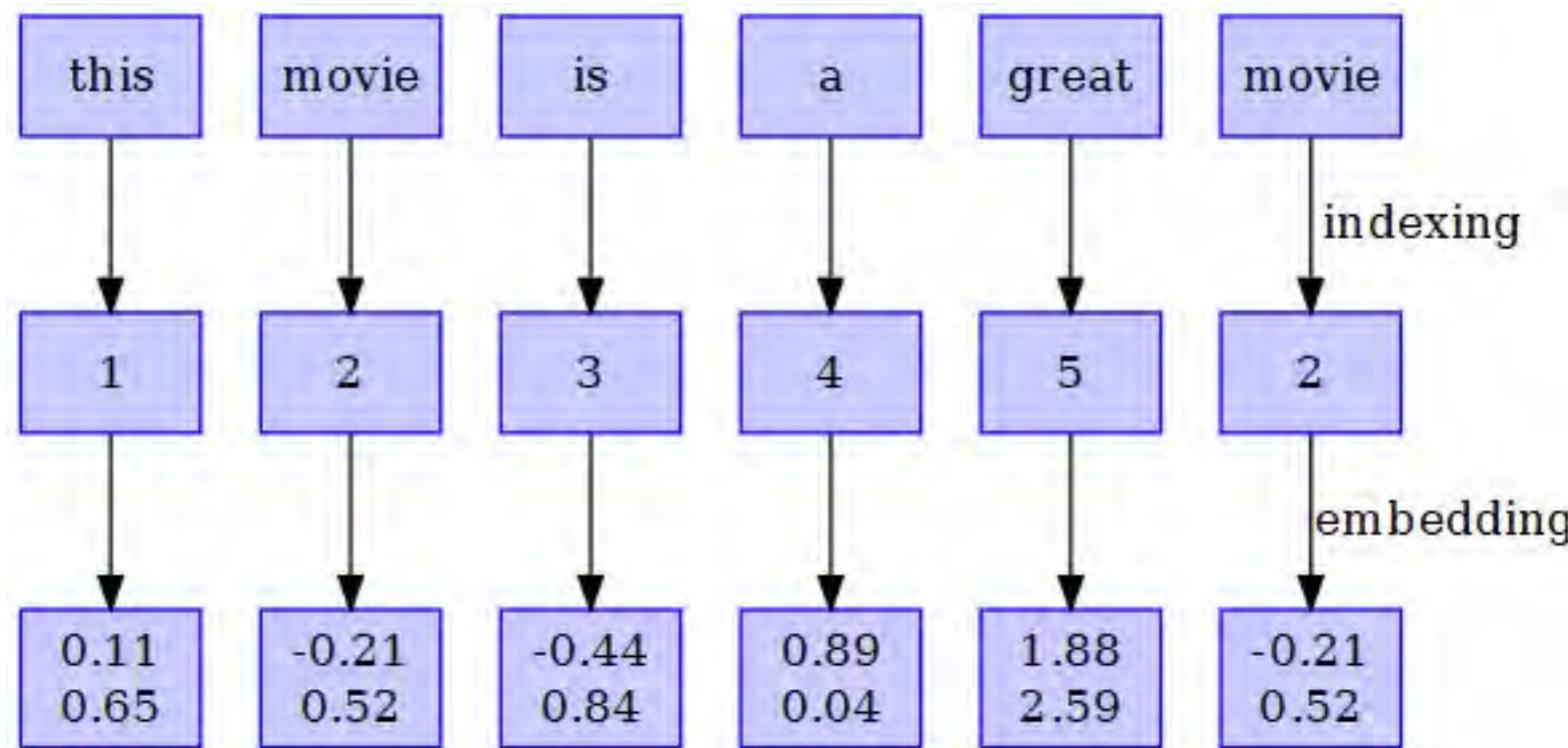




RNN with R

An example of application of RNN to text classification using padded and bucketed data to efficiently handle sequences of varying lengths.





# Workshop 4.21 RNN

<http://dmlc.github.io/rstats/2017/10/11/rnn-bucket-mxnet-R.html>



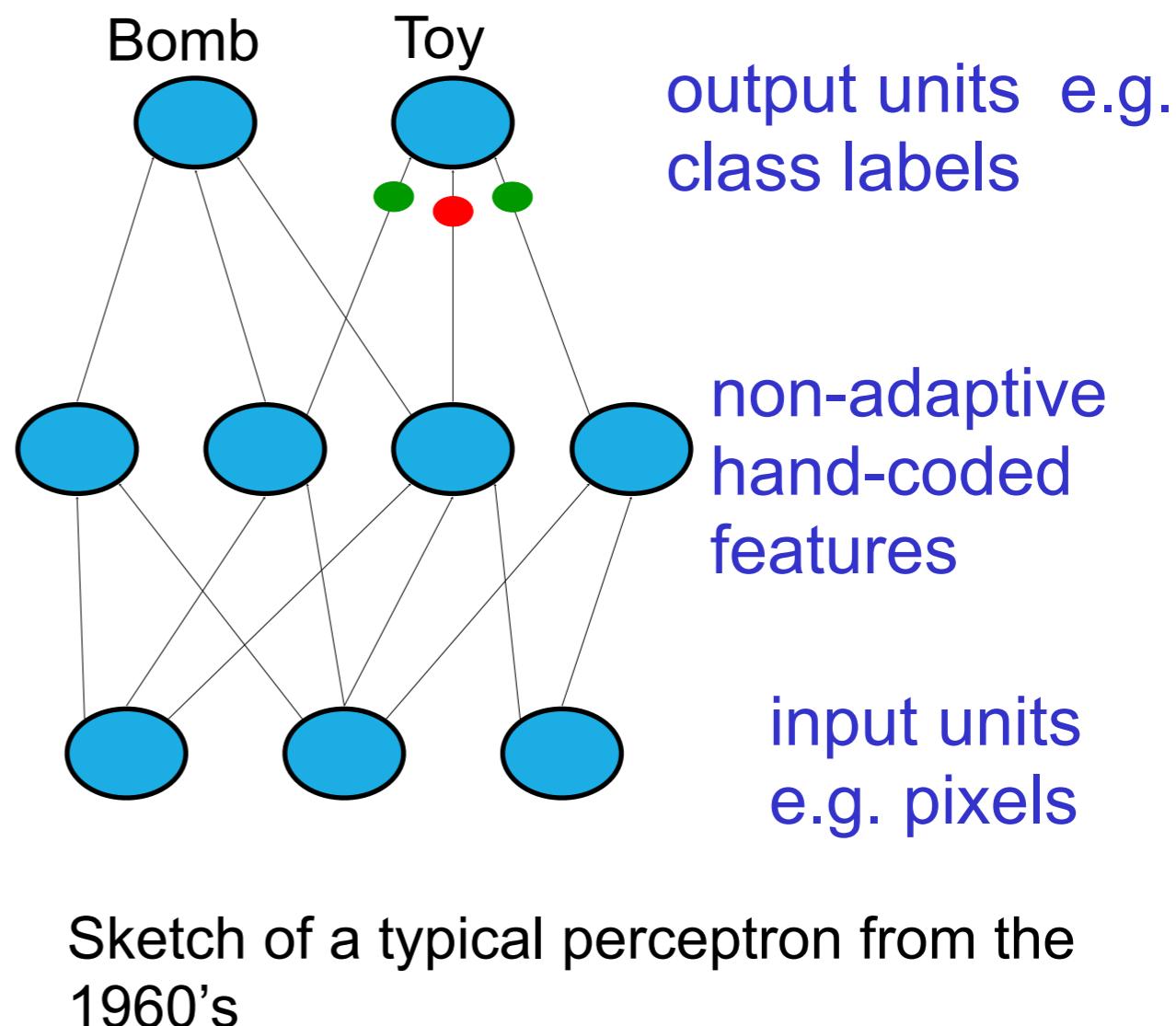
# Deep Belief Networks

# First generation neural networks

Perceptrons (~1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.

There was a neat learning algorithm for adjusting the weights.

But perceptrons are fundamentally limited in what they can learn to do.



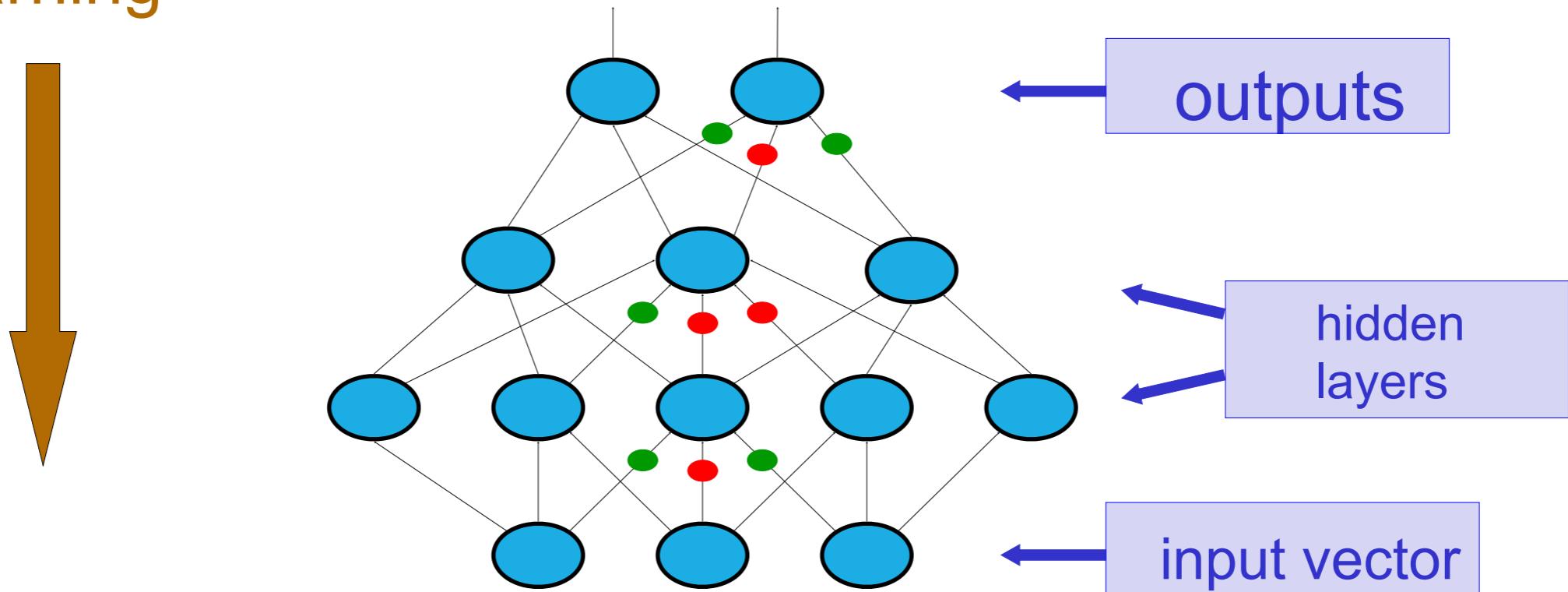
# Support Vector Machine is a perceptron

- Vapnik and his co-workers developed a very clever type of perceptron called a **Support Vector Machine**.
  - Instead of hand-coding the layer of non-adaptive features, each training example is used to create a new feature using a fixed recipe.
    - The feature computes how similar a test example is to that training example.
    - Then a clever optimization technique is used to select the best subset of the features and to decide how to weight each feature when classifying a test case.
    - But its just a perceptron and has all the same limitations.
  - In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.

# Second generation neural networks (~1985)

Back-propagate  
error signal to  
get derivatives  
for learning

Compare outputs with **correct answer** to get error signal



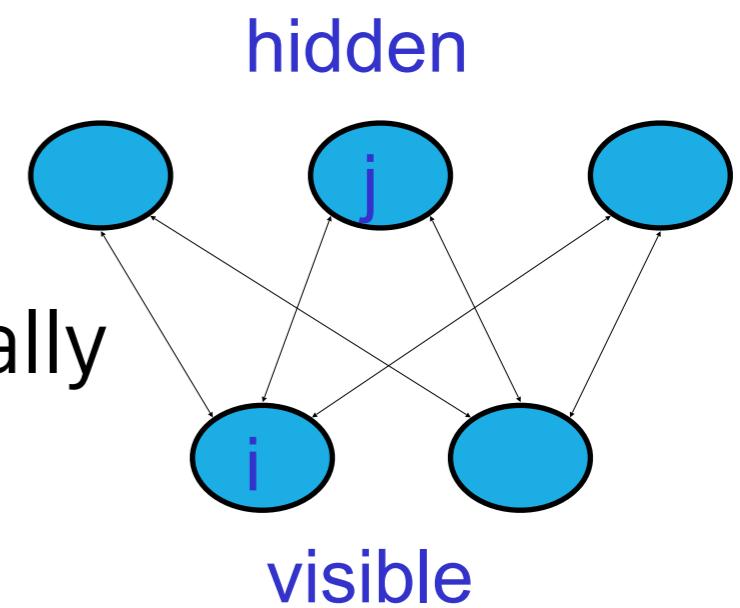
(Slide from Hinton)

# What is wrong with back-propagation?

- It requires labeled training data.
  - Almost all data is unlabeled.
- The learning time does not scale well
  - It is very slow in networks with multiple hidden layers.
- It can get stuck in poor local optima.
  - These are often quite good, but for deep nets they are far from optimal
- **Deep learning (partially) overcomes these difficulties**  
by using undirected graphical model

# Restricted Boltzmann Machines (RBM)

- We restrict the connectivity to make learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM, the hidden units are conditionally independent given the visible states.
- So we can quickly get an unbiased sample from the posterior distribution when given a data-vector.



# RBM: Weights → Energies → Probabilities

- Joint distribution  $p(\mathbf{v}, \mathbf{h}; \theta)$  is defined in terms of an energy function  $E(\mathbf{v}, \mathbf{h}; \theta)$

$$p(\mathbf{v}, \mathbf{h}; \theta) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}; \theta))}{Z}$$

- For a Bernoulli-Bernoulli RBM

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i=1}^V \sum_{j=1}^H w_{ij} v_i h_j - \sum_{i=1}^V b_i v_i - \sum_{j=1}^H a_j h_j$$

- For a Gaussian-Bernoulli RBM

$$E(\mathbf{v}, \mathbf{h}; \theta) = - \sum_{i=1}^V \sum_{j=1}^H w_{ij} v_i h_j + \frac{1}{2} \sum_{i=1}^V (v_i - b_i)^2 - \sum_{j=1}^H a_j h_j$$

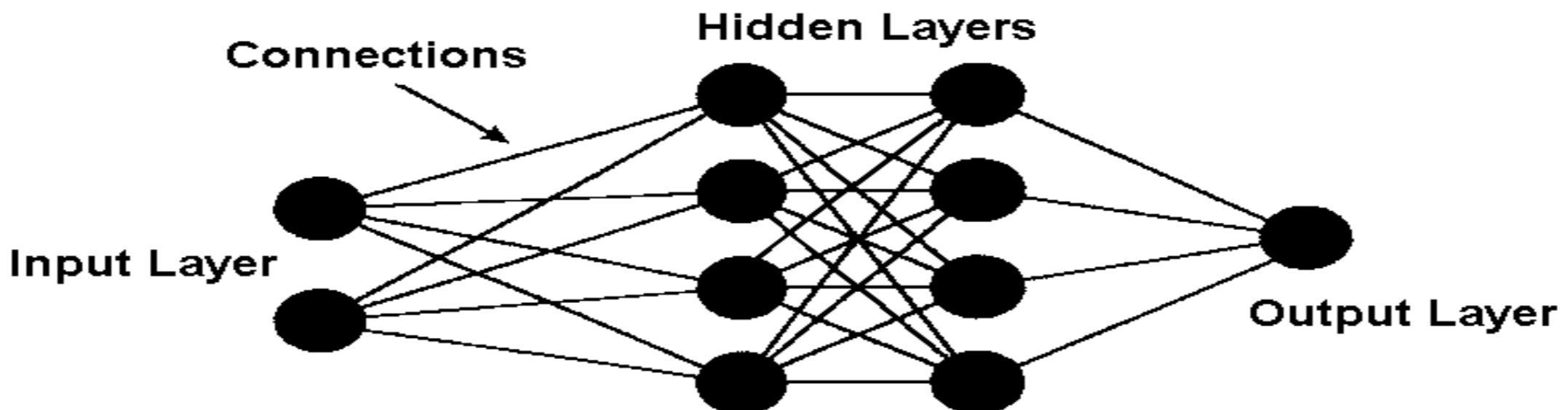
- $p(\mathbf{v}, \mathbf{h}; \theta) \rightarrow$  generative model!

# RBM versus GMM

- Gaussian Mixture Model
  - Local representation
  - (In practice,) data vector explained by only a single Gaussian
  - Tend to over-fit
- Bernoulli-Gaussian RBM
  - Distributed representation, very powerful
  - Product of Gaussians
  - Tend to under-fit

# Deep Neural Networks

- Multiple research results in 2006-2009 suggest that a **deep** structure is **more effective** than a **shallow** one **when trained in the right way** [Hinton 2012]
- A **Deep Neural Network** is a **feed-forward ANN** with **more than one** hidden layer [Hinton 2012]

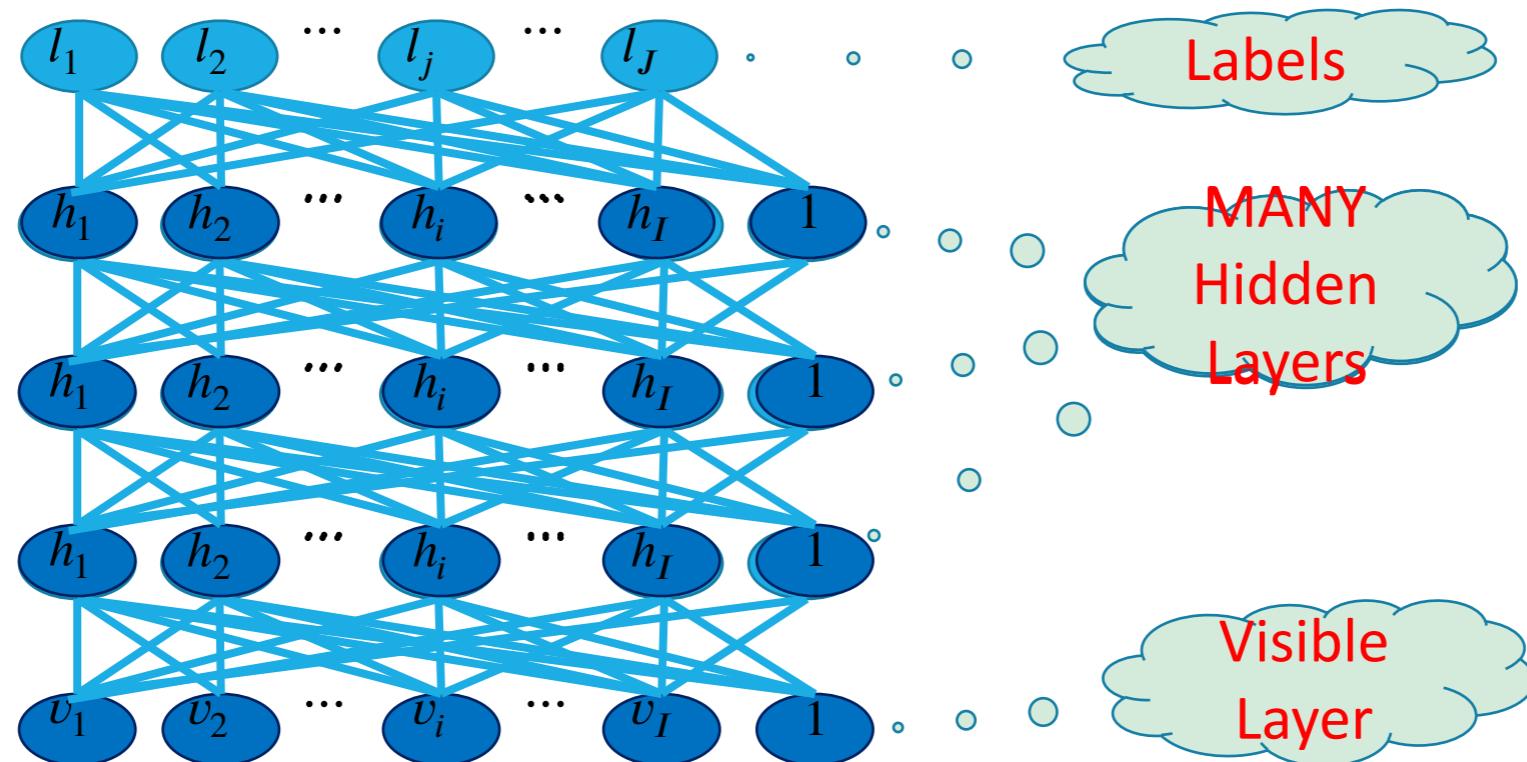


# Building a Deep Network

- This is the main reason why RBM's are interesting (as a building block)
- First train a layer of hidden units that receive input directly from the data (image, speech, coded text, etc).
- Then treat the activations of hidden units (the trained "features") as if they were "data" and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
  - The proof is complicated (Hinton et al, 2006)
  - Based on an equivalence between an RBM and a deep directed model

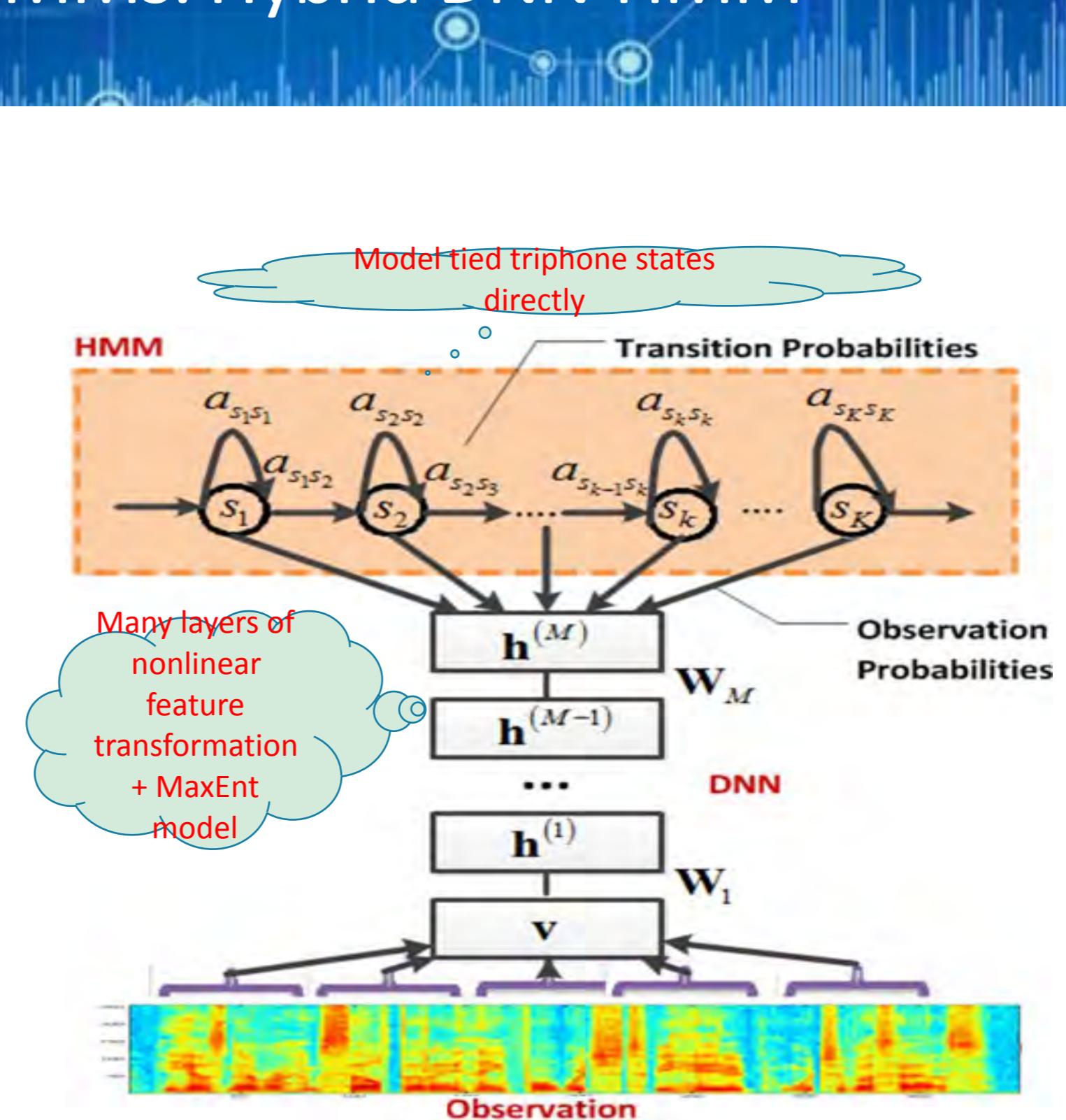
# Deep Belief Net (DBN) & Deep Neural Net (DNN)

- DBN: Undirected at top two layers which is an RBM; directed Bayes net (top-down) at lower layers (good for **synthesis and recognition**)
- DNN: Multi-layer perceptron (bottom up) + unsupervised pre-training w. RBM weights (good for **recognition only**)



# DNNs in place of GMMs: Hybrid DNN-HMM

- One DNN for all states of all HMMs
- **Input:** current frame + C frames before and after → **A total of  $2C+1$  vectors**
- **Output:** 1 probability for each state of each HMM
- **What DNN Models:**  
 $P(\text{State} \mid \text{Observation})$



# DNN-HMM (hybrid) Summary

- Basic neural nets with senone outputs can be surprisingly effective
- DNN-HMMs work very well compared to GMM-HMMs on a large vocabulary task with high acoustic variability
- Importance of being deep
- The quality of the alignment used to produce training labels is important
- Experiments on Switchboard are more promising (MSR: interspeech 2011)

# How to train such a DNN?

2 steps: ([Hinton 2012][Dahl 2012])

1. Pre-train a Deep Belief Network (Unsupervised)

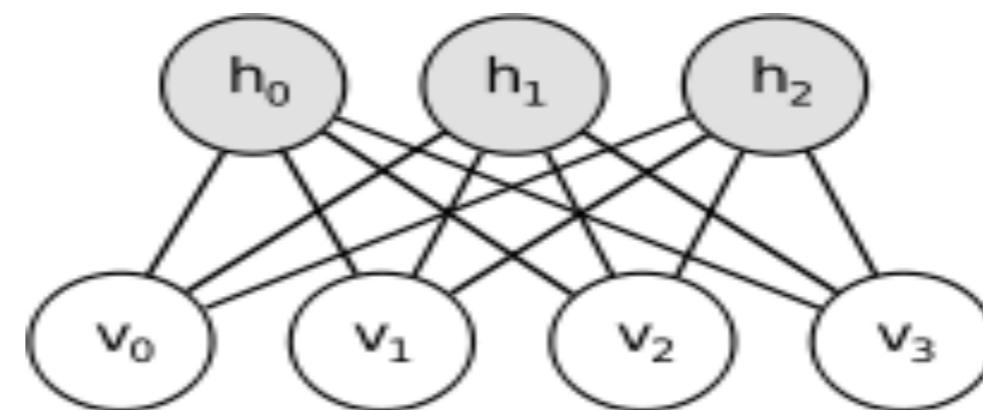
This generatively models the input space and its structure

2. Add a softmax layer to the trained DBN; make it directed and **fine-tune the weights using standard back-propagation**  
(Supervised)

The state labels come from a previously trained HMM-GMM system

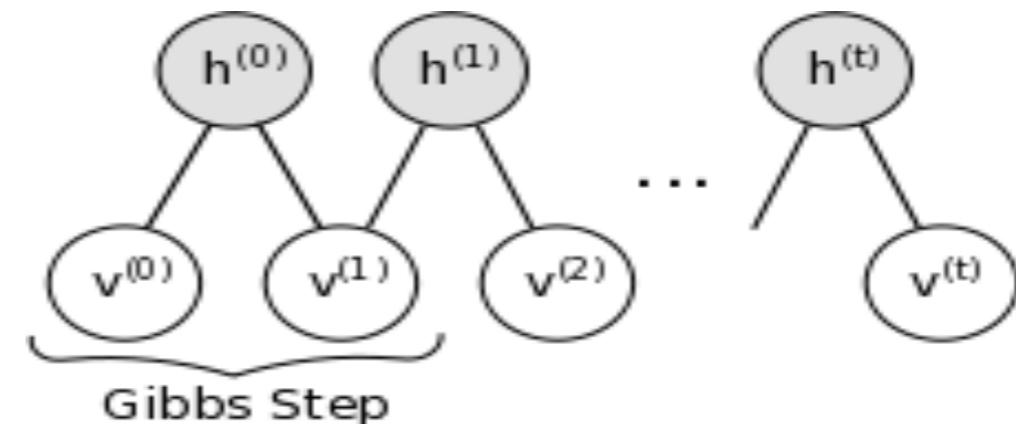
# Unsupervised Pre-training

- Intuition: **directly** training a DNN might result in a **local** extremum due to **very large space** of model parameters
- Pre-training finds **a suitable starting point**
- A Deep Belief Network (DBN) is a stack of Restricted Boltzmann Machines (RBMs)
- An RBM is an undirected graphical model with two layers: a visible and a latent (hidden) layer:



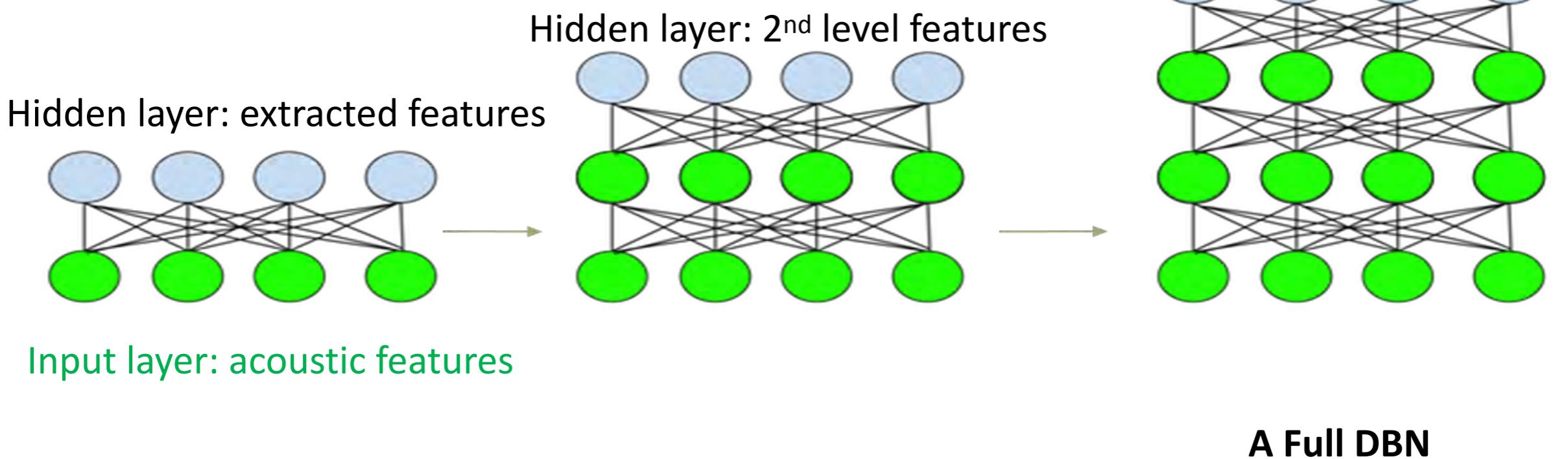
# Unsupervised Pre-training

- An RBM defines a joint probability measure over the latent and visible variables:
- $P(\vec{h}, \vec{v}; W) = \frac{e^{-E(\vec{h}, \vec{v}; W)}}{\text{constant}}$ , where  $E(\vec{h}, \vec{v}; W)$  is the total energy of joint configuration  $\vec{h}, \vec{v}$
- For real-valued RBMs:  $E(\vec{h}, \vec{v}; W)$  has a simple form
- A fast variant of Gibbs Sampling namely **Contrastive Divergence method** is used to train the weight parameters  $W$  [Hinton 2006]



# Unsupervised Pre-training

After a first RBM is trained on the acoustic features (as input), another RBM is placed on top of it and is trained using the latent layer of the first RBM as the input layer of the new RBM. We continue this to form a DBN with arbitrary depth:



# Unsupervised Pre-training (cont.)

- The obtained DBN is a generative model of input acoustic features and is a deep model of the input space
- **Add one softmax layer on top of this DBN with zero weights and train (fine-tune) the whole network discriminatively**

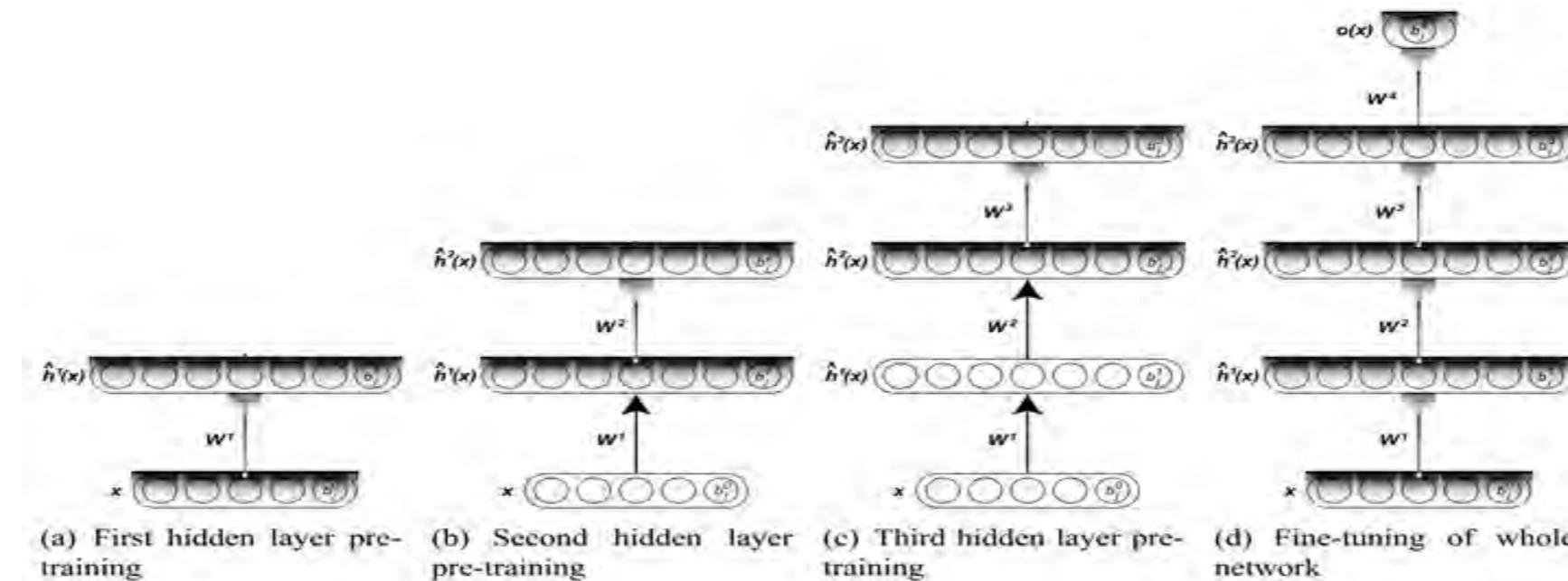
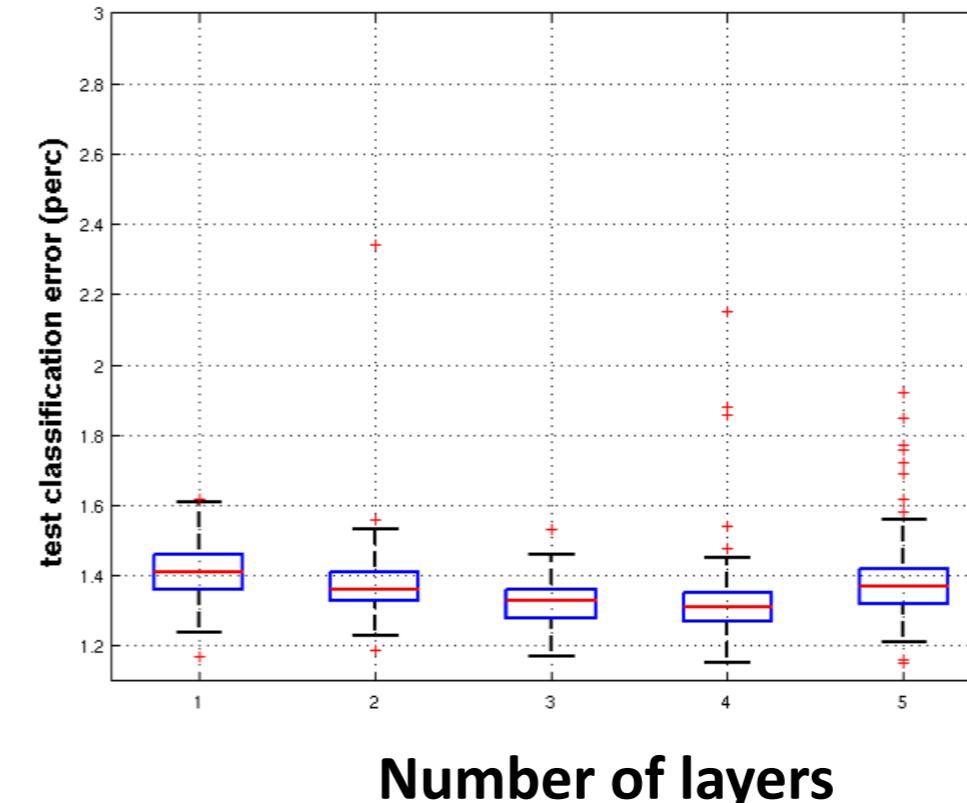
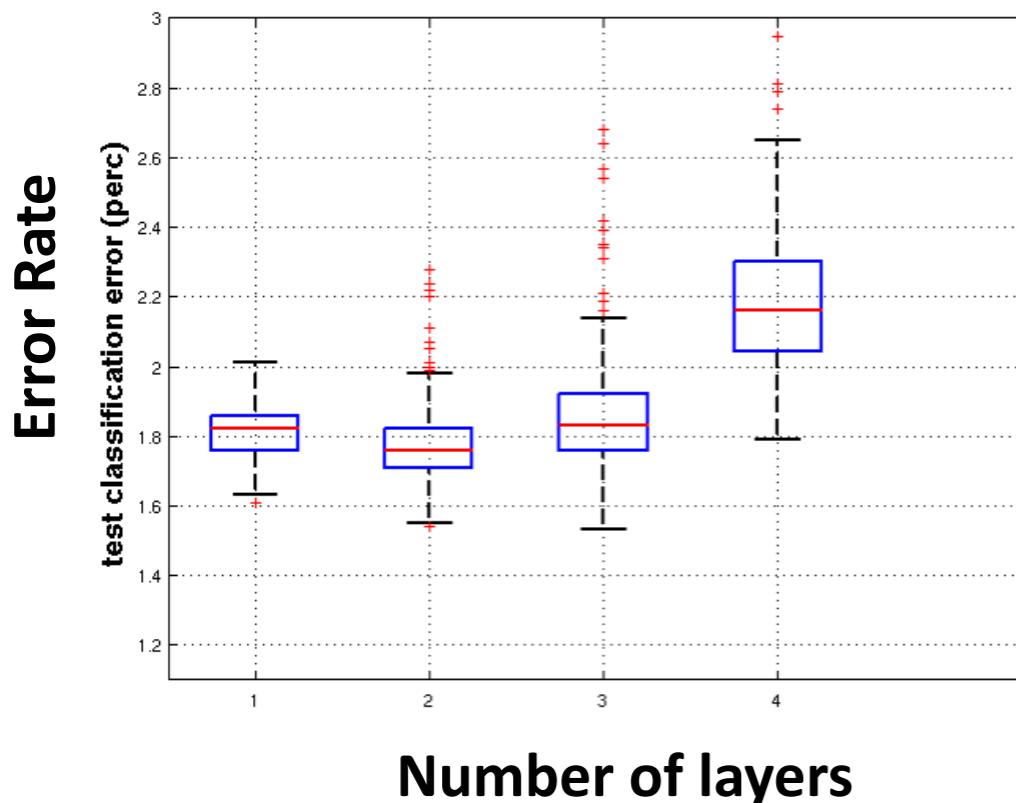


Figure from [Larochelle 2009]

# Summary of DNN-HMMs

- GMMs while quite powerful have a weakness
- ANNs can model manifold data structures more efficiently and with fewer parameters [Hinton 2012]
- DNNs are essentially deeper ANNs. So what is the difference between now and 2 decades ago ?
  - Hardware advances: GPU
  - Algorithms: Pre-training (Contrastive Divergence), better loss functions, ...

# Deep vs. Shallow



Two researches in 2010 and 2012 show that effect of pre-training is not remarkable in LVCSR [Dahl 2012, Yu 2010]

# Pros and Cons of DNN-HMM

- Pros:
  - Better acoustic modeling → Consistent higher accuracies
  - More robust to noise
  - Usually faster at decoding time
- Cons:
  - Training can be longer
  - Needs a pre-trained HMM-GMM system



DBN in R

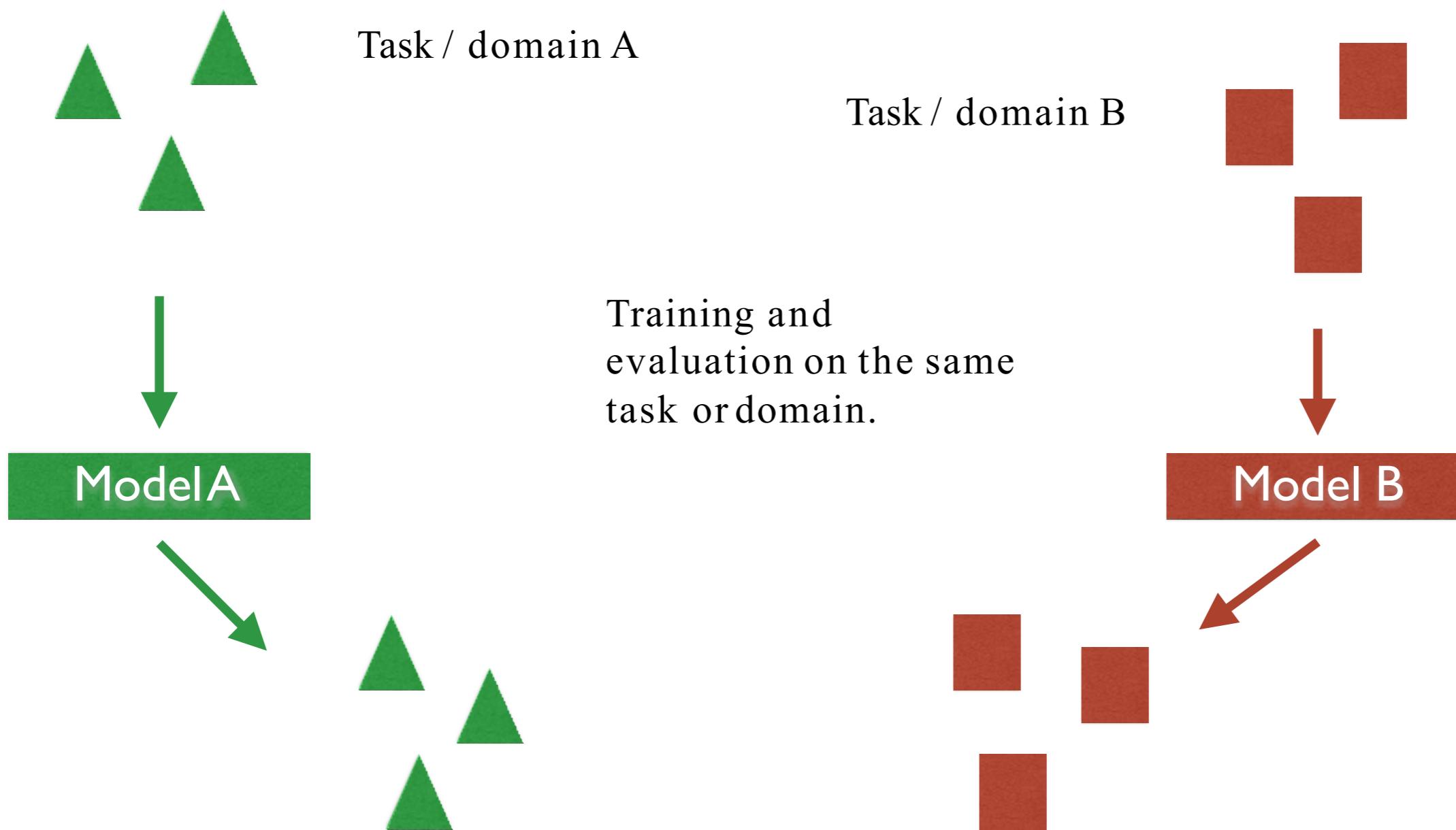




# Transfer Learning

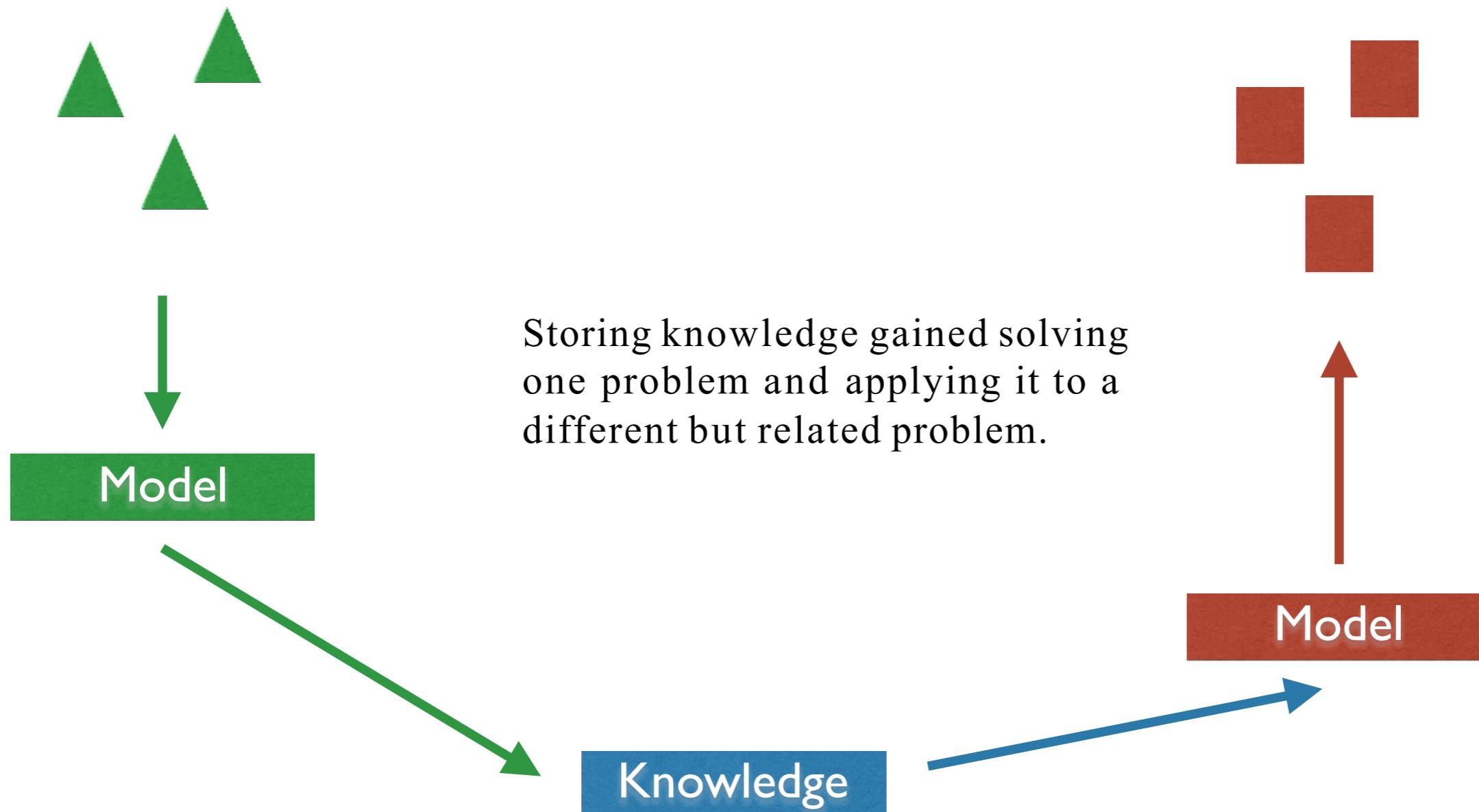
# What is Transfer Learning?

## Traditional ML



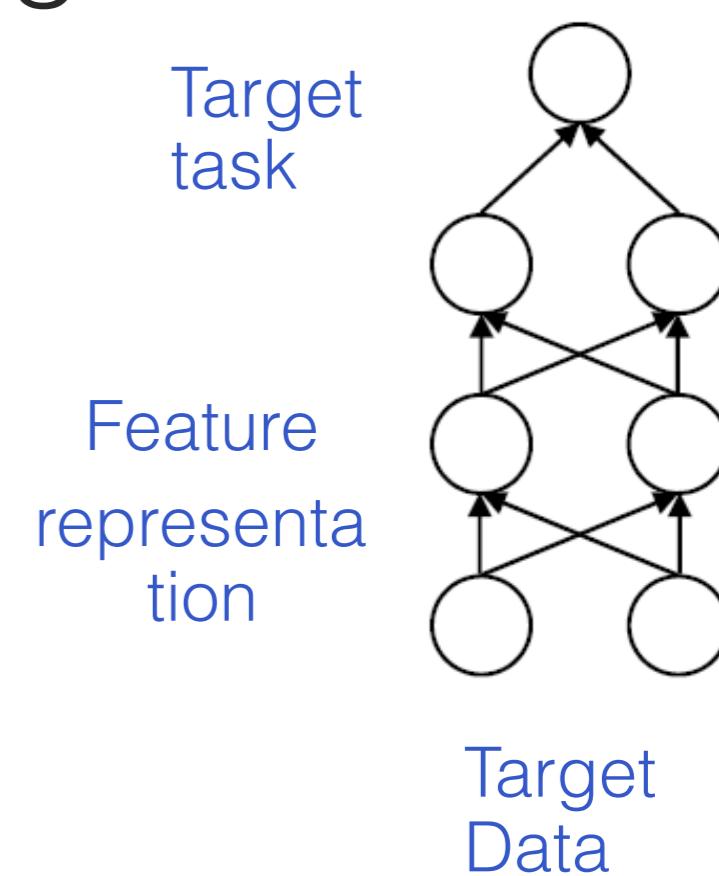
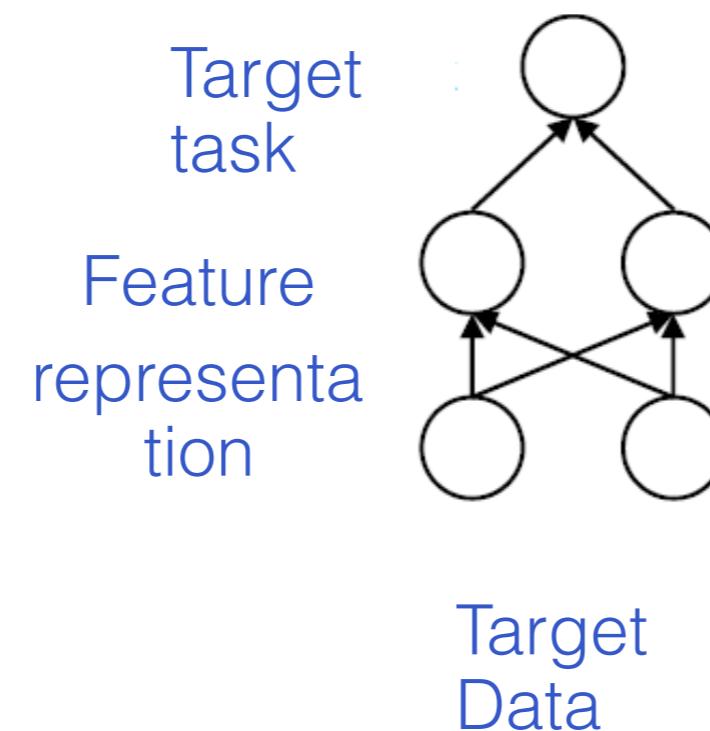
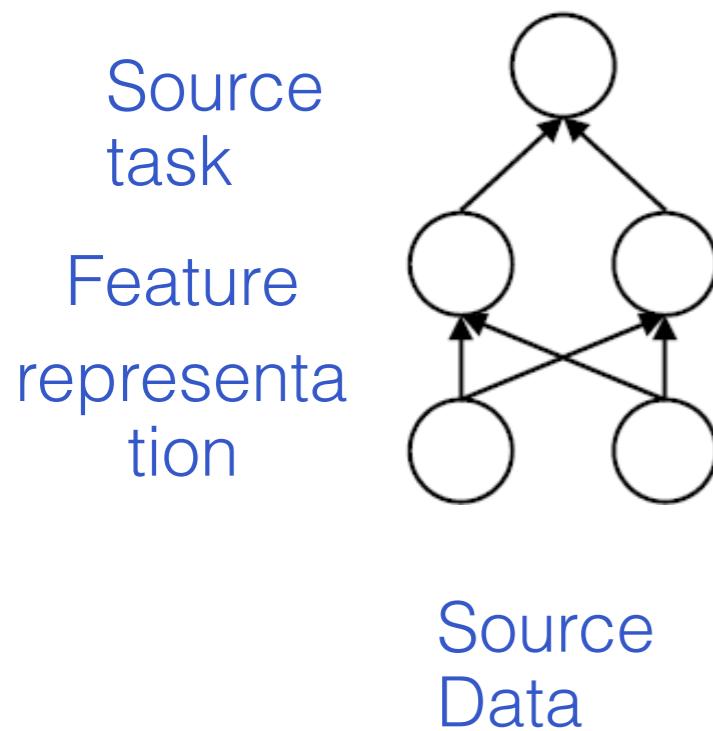
# What is Transfer Learning?

## Transfer learning



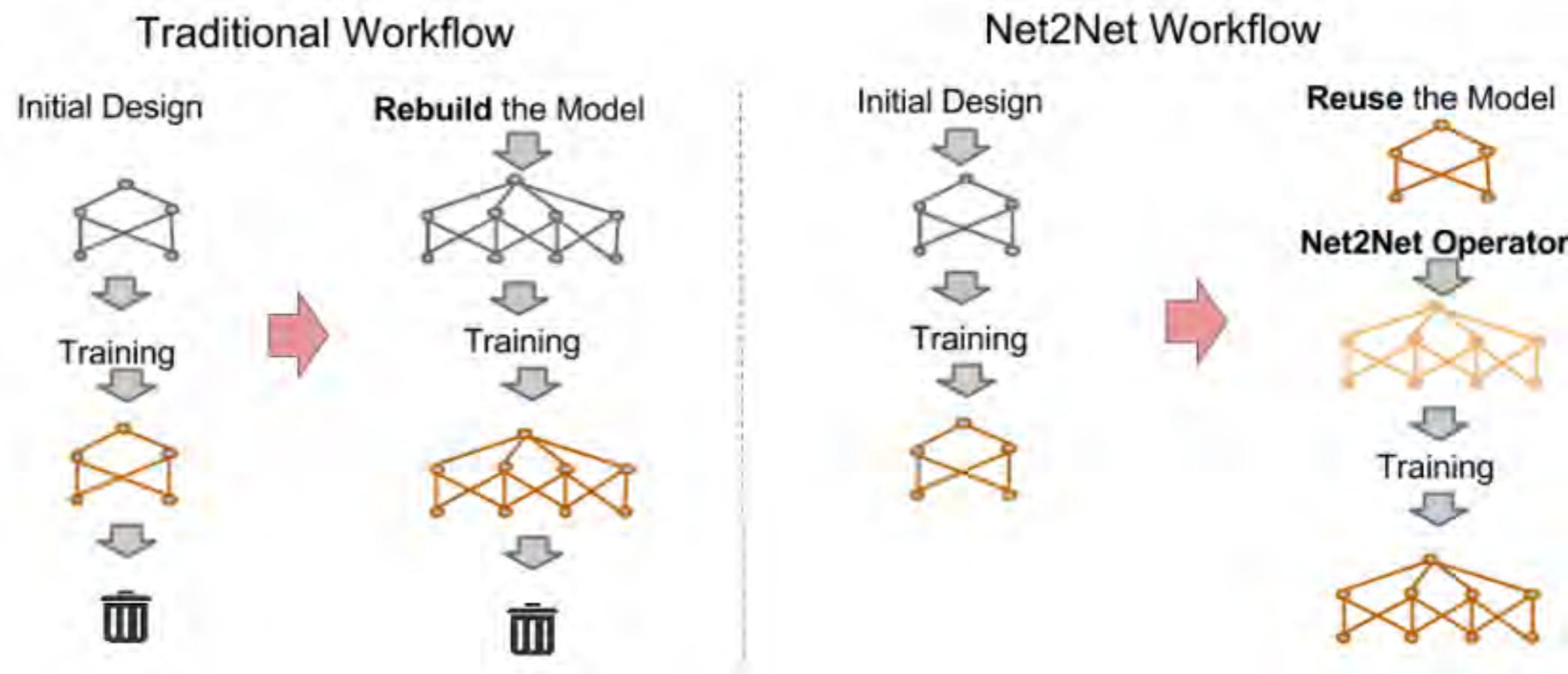
# Transfer Learning in Deep Learning

- Feed new data for domain adaptation
- Build higher layers for training another task (feature transfer)



# Net2Net Transfer

Net2Net reuses information of already trained model to speedup training of new model



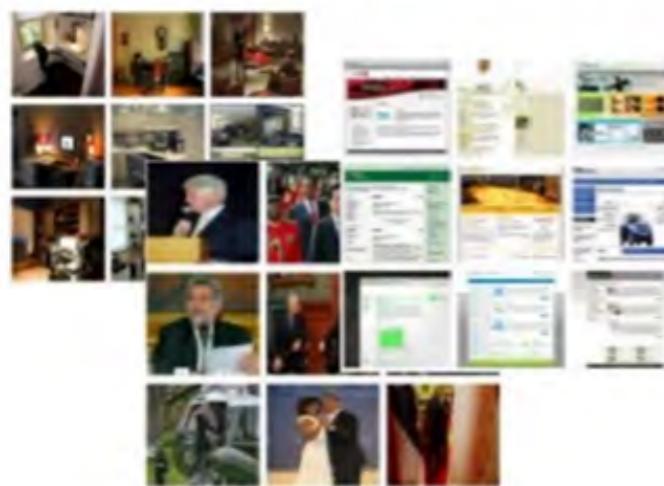
# Transfer Learning in Deep Learning

Why use a pre-trained model?

- It's faster (it's pre-trained)
- It's cheaper (no need for GPU farm)
- It generalizes (avoid overfitting)



# Transfer Learning



# Transfer Learning in Deep Learning

ConvNet as fixed feature extractor

- Example: Use a car classifier to further classify the car brand
- Very fast

Fine-tuning the ConvNet

- Fine-tune weights of pre-trained CNN
- Fine-tune all layers / Keep early layers fixed and fine-tune remaining layers
- Slow



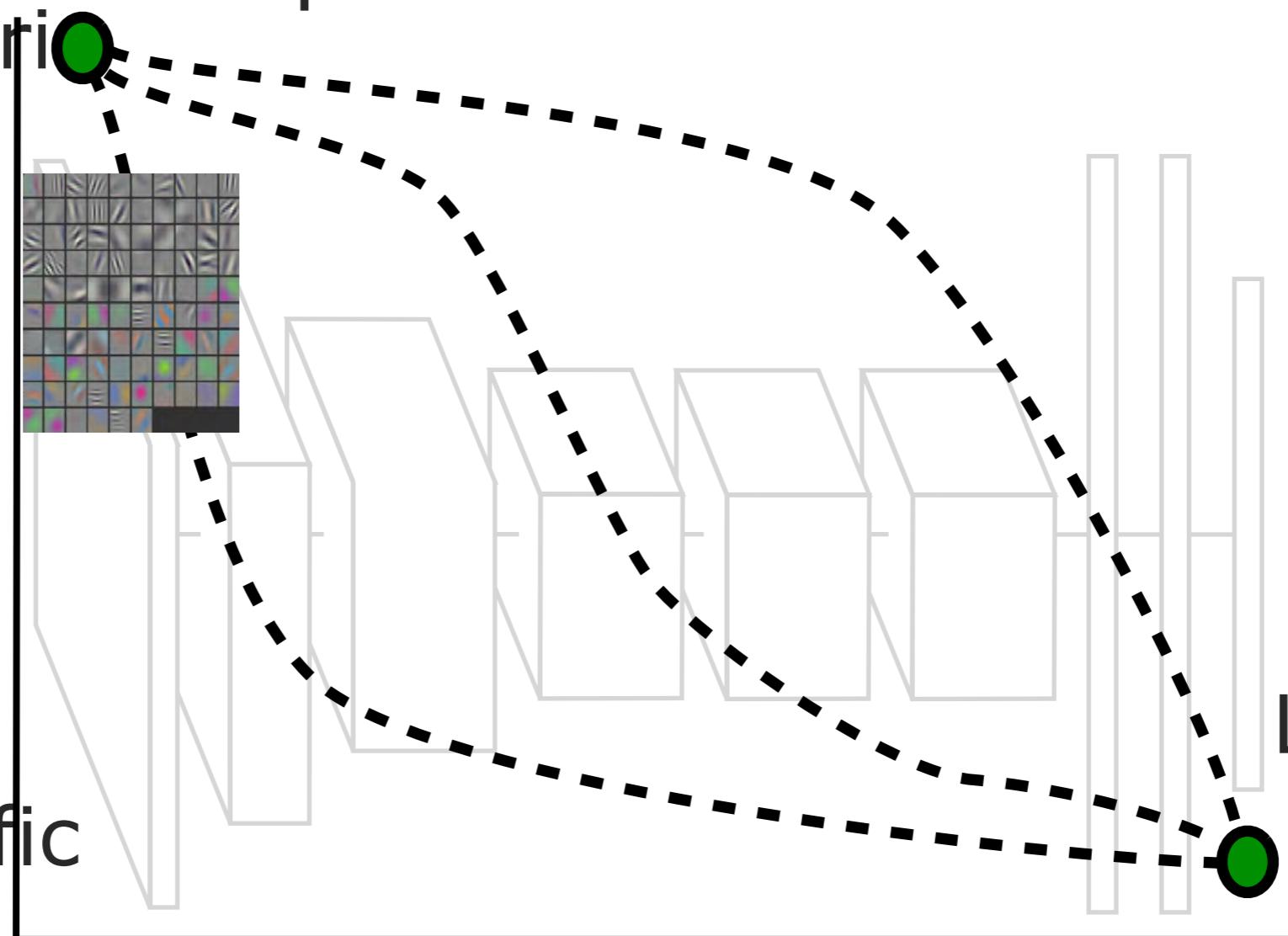
# Generic to Specific

generic

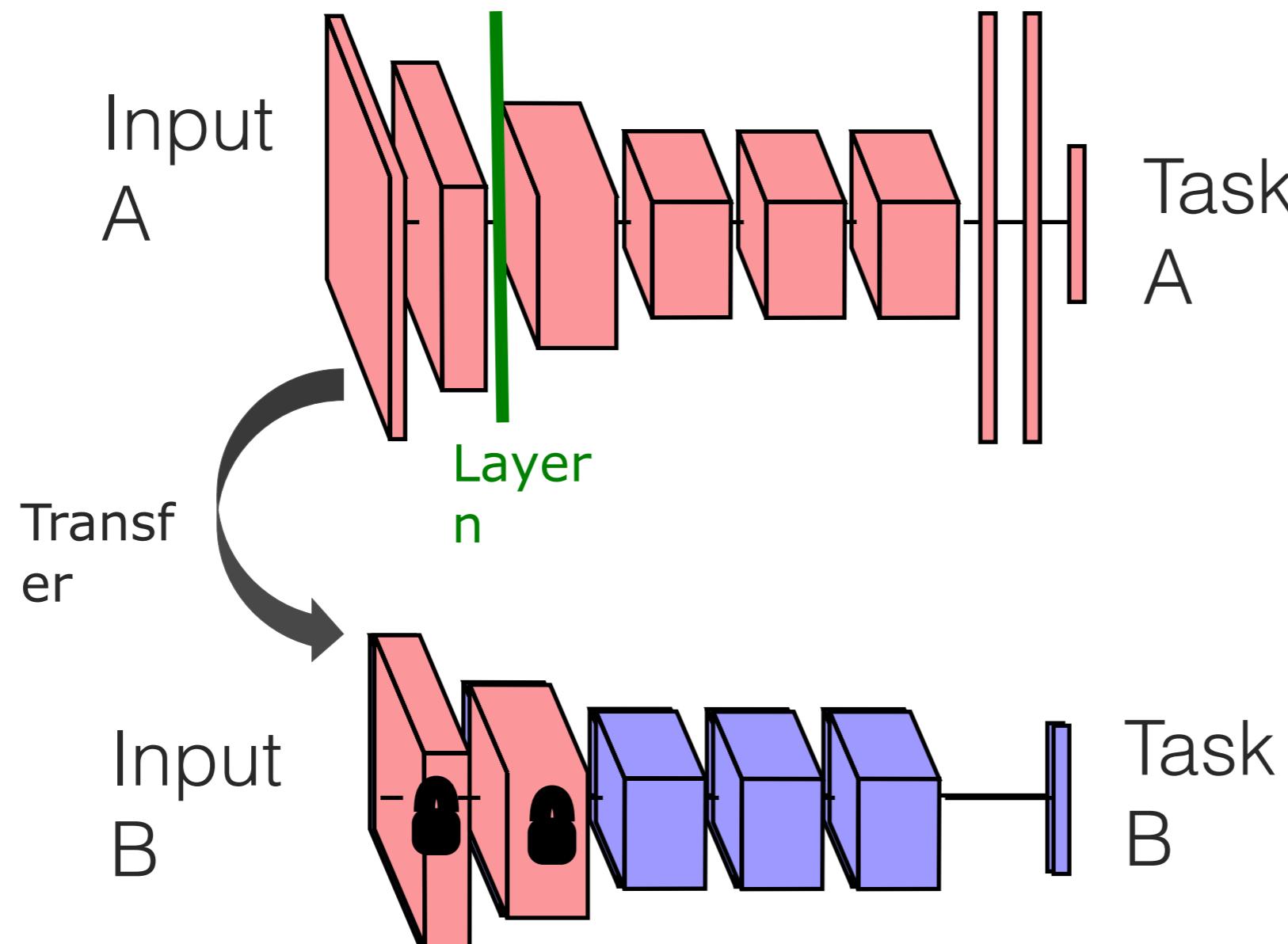
specific

Layers

Lion



# Fine-tuning



# When To Use Transfer Learning?

- Dependence on size of the new dataset and similarity to the original dataset
- Features are more generic in early layers and more dataset-specific in later layers
- 4 Scenarios

|           | Relatively Smaller                                                                                                         | Relatively Large                                                     |
|-----------|----------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| Similar   | <ul style="list-style-type: none"><li>- Similar higher-level features</li><li>- Train classifier on CNN features</li></ul> | <ul style="list-style-type: none"><li>- Fine-tune</li></ul>          |
| Different | <ul style="list-style-type: none"><li>- Train classifier from activations somewhere early in the network</li></ul>         | <ul style="list-style-type: none"><li>- Train from scratch</li></ul> |



# Transfer Learning in R

## Setup

```
library(keras)
library(tfhub)
```

## DOWNLOAD THE CLASSIFIER

Use `layer_hub` to load a mobilenet and wrap it up as a keras layer. Any TensorFlow 2 compatible image classifier URL from [tfhub.dev](https://tfhub.dev) will work here.

```
classifier_url = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/2"

image_shape <- c(224L, 224L, 3L)

classifier <- layer_hub(handle = classifier_url, input_shape = image_shape)
```

```
1 image_url <- "https://storage.googleapis.com/download.tensorflow.org/example_images/grace_hopper.jpg"
2
3 img <- pins::pin(image_url, name = "grace_hopper") %>%
4 tensorflow::tfioread_file() %>%
5 tensorflow::tf$image$decode_image(dtype = tf$float32) %>%
6 tensorflow::tf$image$resize(size = image_shape[-3])
7
8 img %>%
9 as.array() %>%
10 as.raster() %>%
11 plot()
```





Add a batch dimension, and pass the image to the model.

```
result <- img %>%
 tf$expand_dims(0L) %>%
 classifier()
```

The result is a 1001 element vector of logits, rating the probability of each class for the image.

So the top class ID can be found with argmax:

```
predicted_class <- tf$argmax(result, axis = 1L) %>% as.integer()
predicted_class
```

```
[1] 653
```

```
13 labels_url <- "https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt"
14
15 imagenet_labels <- pins::pin(labels_url, "imagenet_labels") %>%
16 readLines()
17
18 img %>%
19 as.array() %>%
20 as.raster() %>%
21 plot()
22 #
23 title(paste("Prediction:", imagenet_labels[predicted_class + 1]))
```

Prediction: military uniform



# Simple transfer learning

```
title(paste("Prediction:", imagenet_labels[predicted_class + 1]))

flowers <- pins::pin("https://storage.googleapis.com/download.tensorflow.org/
example_images/flower_photos.tgz", "flower_photos")

image_generator <- image_data_generator(rescale=1/255)
image_data <- flowers[1] %>%
 dirname() %>%
 dirname() %>%
 flow_images_from_directory(image_generator, target_size = image_shape[-3])

str(reticulate::iter_next(image_data))
```

```
RUN THE CLASSIFIER ON A BATCH OF IMAGES
image_batch <- reticulate::iter_next(image_data)
predictions <- classifier(tf$constant(image_batch[[1]], tf$float32))
predicted_classnames <- imnet_labels[as.integer(tf$argmax(predictions, axis = 1L) + 1L)]

par(mfcol = c(4,8), mar = rep(1, 4), oma = rep(0.2, 4))
image_batch[[1]] %>%
 purrr::array_tree(1) %>%
 purrr::set_names(predicted_classnames) %>%
 purrr::map(as.raster) %>%
 purrr::iwalk(~{plot(.x); title(.y)})
```

cardoon



fountain



jack-o'-lantern



daisy



binder



mushroom



daisy



daisy



ostrich



titi



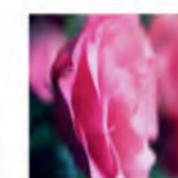
rapeseed



chickadee



ice lolly



bell pepper



tile roof



cardoon



daisy



daisy



vase



low lady's slip coral fungus



mushroom



daisy



umbrella



park bench



brassiere



bee



buckeye



daisy



greenhouse



bee



picket fence



# Download Trained Model

```
feature_extractor_url <- "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/2"

feature_extractor_layer <- layer_hub(handle = feature_extractor_url,
 input_shape = image_shape)

feature_batch <- feature_extractor_layer(tf$constant(image_batch[[1]]), tf$float32())
feature_batch
```

```
tf.Tensor(
[[0.13427277 0.16856195 0.283491 ... 0.00557477 0. 0.8134863]
[0.00754159 0.49517953 0.18708485 ... 0.01621983 0. 0.]
[0. 0.60116017 0. ... 0. 0.05334444 0.00277256]
...
[0.6140208 1.3715637 0. ... 0.02907389 0.11318099 0.12228318]
[1.2423071 1.0235544 0.170658 ... 0.51680547 0. 0.]
[0.5452022 0.2789958 0.16163555 ... 0.1076004 0.01267634 0.],
shape=(32, 1280)
```

## ATTACH A CLASSIFICATION HEAD

Now let's create a sequential model using the feature extraction layer and add a new classification layer.

```
model <- keras_model_sequential(list(
 feature_extractor_layer,
 layer_dense(units = image_data$num_classes, activation='softmax')))
```

```
summary(model)
```

```
Model: "sequential"

Layer (type) Output Shape Param #

keras_layer_1 (KerasLayer) (None, 1280) 2257984

dense (Dense) (None, 5) 6405

Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984

```

## Train the model

Use compile to configure the training process:

```
model %>% compile(
 optimizer = "adam",
 loss = "categorical_crossentropy",
 metrics = "accuracy"
)
```

Now use the `fit` method to train the model.

To keep this example short train just 2 epochs.

```
history <- model %>% fit_generator(
 image_data, epochs=2,
 steps_per_epoch = image_data$n / image_data$batch_size,
 verbose = 2
)
```

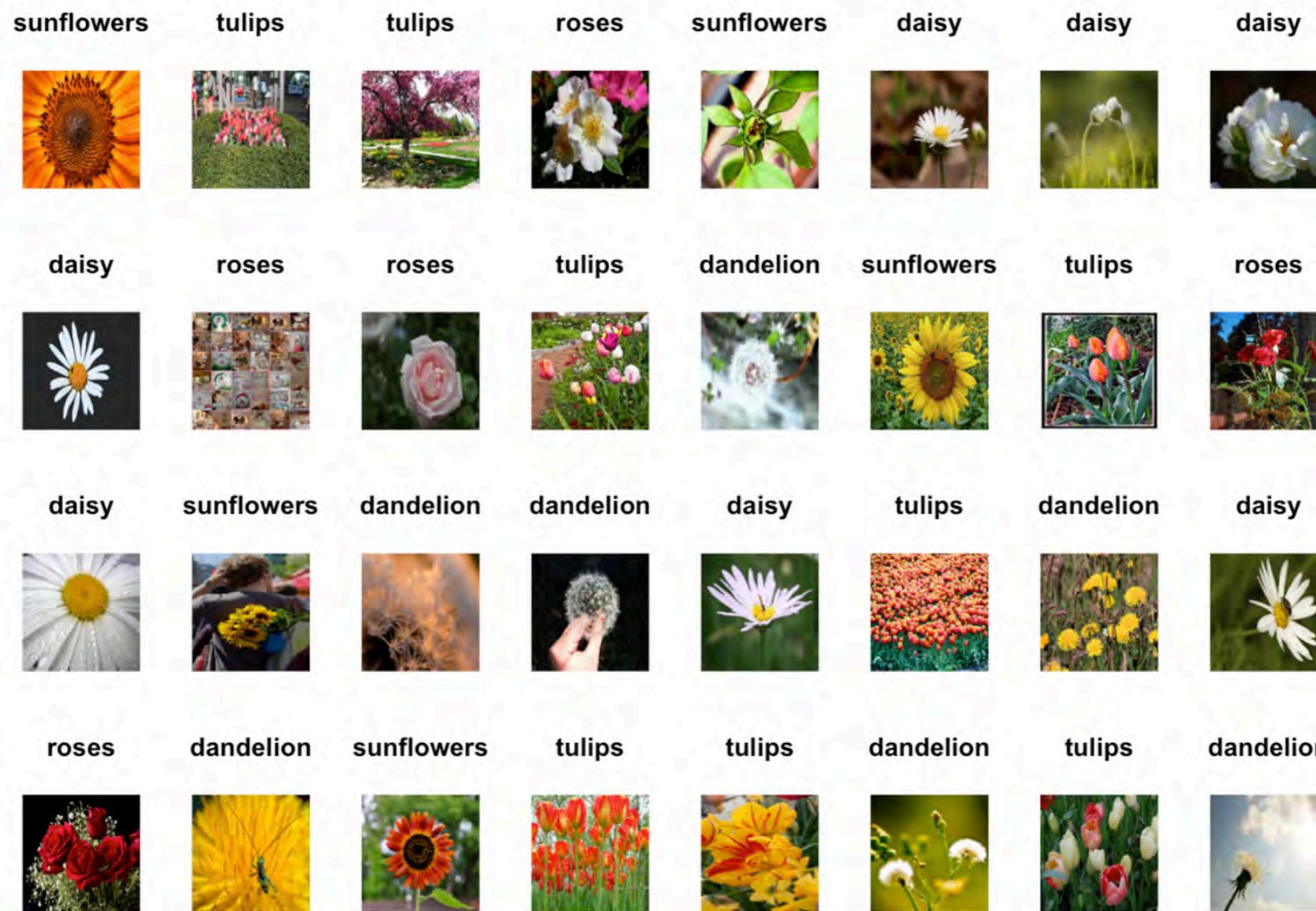
```
Epoch 1/2
114/114 - 255s - loss: 0.6656 - accuracy: 0.7567
Epoch 2/2
114/114 - 250s - loss: 0.3308 - accuracy: 0.8931
```

```

image_batch <- reticulate::iter_next(image_data)
predictions <- predict_classes(model, image_batch[[1]])

par(mfcol = c(4,8), mar = rep(1, 4), oma = rep(0.2, 4))
image_batch[[1]] %>%
 purrr::array_tree(1) %>%
 purrr::set_names(names(image_data$class_indices)[predictions + 1]) %>%
 purrr::map(as.raster) %>%
 purrr::iwalk(~{plot(.x); title(.y)})

```



# Workshop 4.22 Transfer Learning

Try the example by yourself

[https://tensorflow.rstudio.com/tutorials/advanced/images/transfer-learning-  
hub/](https://tensorflow.rstudio.com/tutorials/advanced/images/transfer-learning-hub/)



# Reinforcement Learning



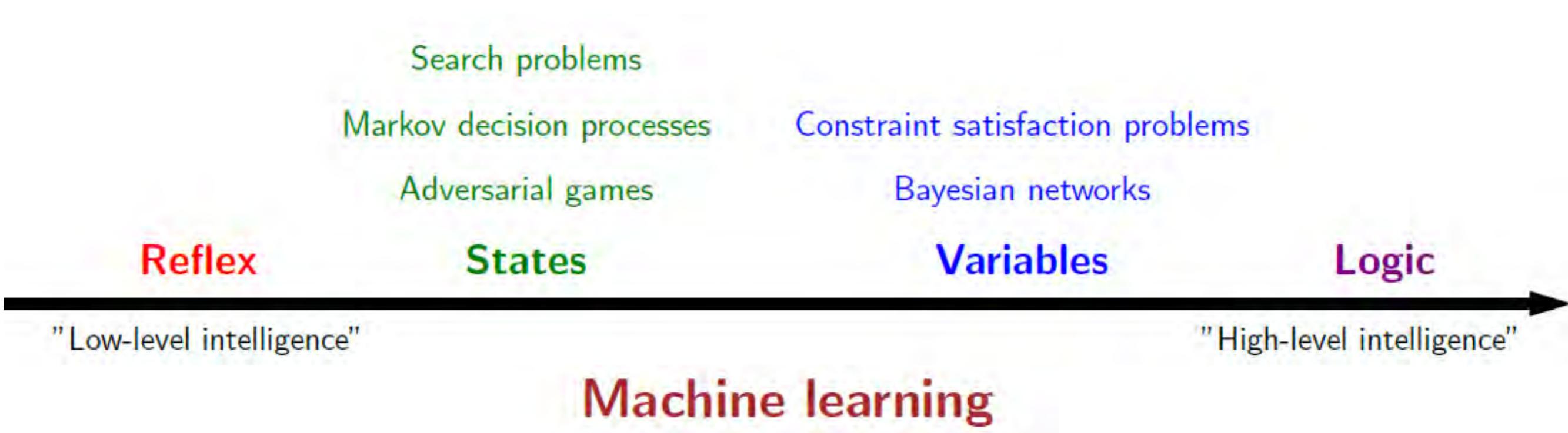
VICE News | HBO

# Artificial Intelligence

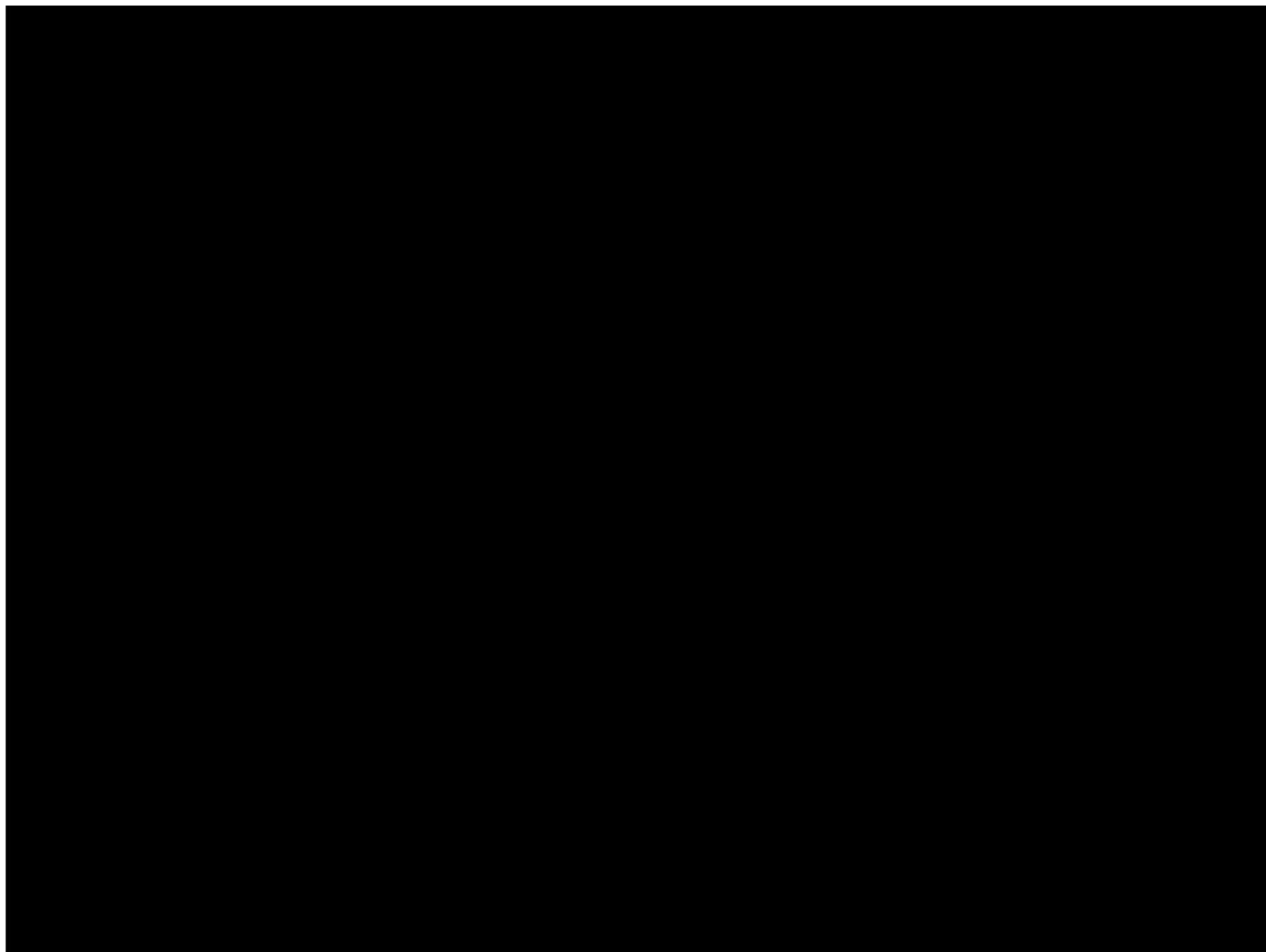


Reinforcement Learning

# Course Topics



# Example: Learning to Play



<http://www.nature.com/nature/journal/v518/n7540/pdf/nature14236.pdf#videos>  
<https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner>

# More Examples

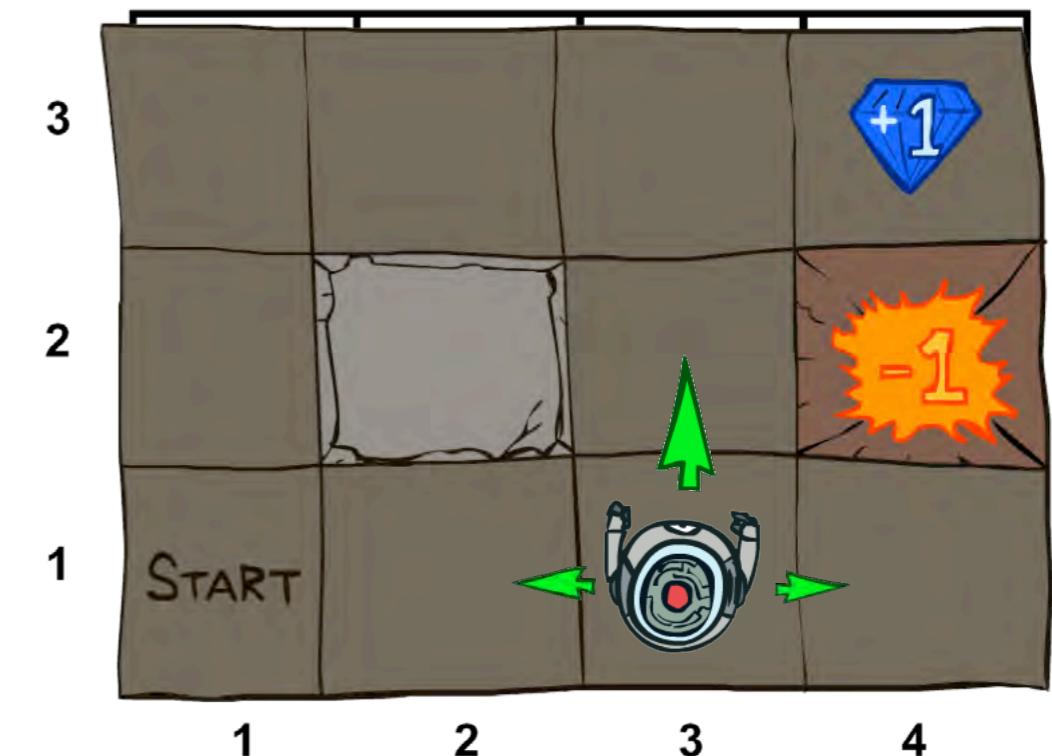
Learning Hand-Eye Coordination for Robotic  
Grasping with Deep Learning and Large-Scale  
Data Collection

Sergey Levine      Peter Pastor  
Alex Krizhevsky    Deirdre Quillen  
Google

[https://www.youtube.com/watch?v=cXaic\\_k80uM&feature=youtu.be](https://www.youtube.com/watch?v=cXaic_k80uM&feature=youtu.be)

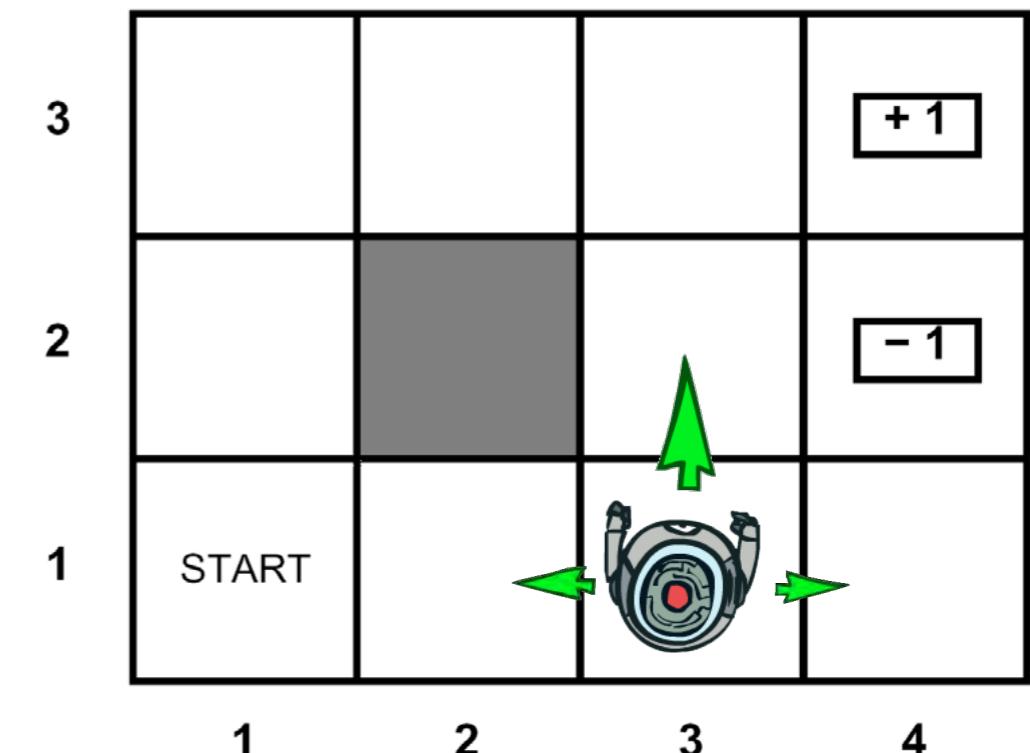
# Example: Grid World

- A maze-like problem
  - The agent lives in a grid
  - Walls block the agent's path
- Noisy movement: actions do not always go as planned
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- The agent receives rewards each time step
  - Small “living” reward each step (can be negative)
  - Big rewards come at the end (good or bad)
- Goal: maximize sum of rewards -- Utilities



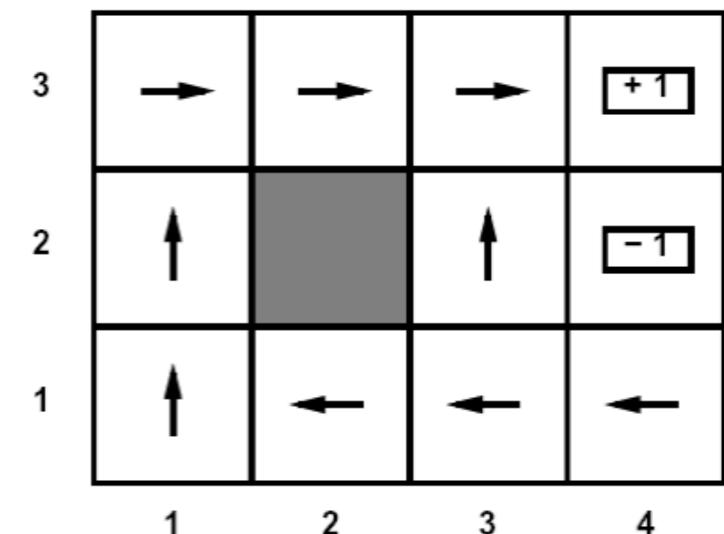
# Markov Decision Processes

- An MDP is defined by:
  - A set of states  $s \in S$
  - A set of actions  $a \in A$
  - A transition function  $T(s, a, s')$ 
    - Probability that  $a$  from  $s$  leads to  $s'$ , i.e.,  $P(s'|s, a)$
    - Also called the model or the dynamics
  - A reward function  $R(s, a, s')$ 
    - Sometimes just  $R(s)$  or  $R(s')$
  - A start state
  - Maybe a terminal state



# Policies

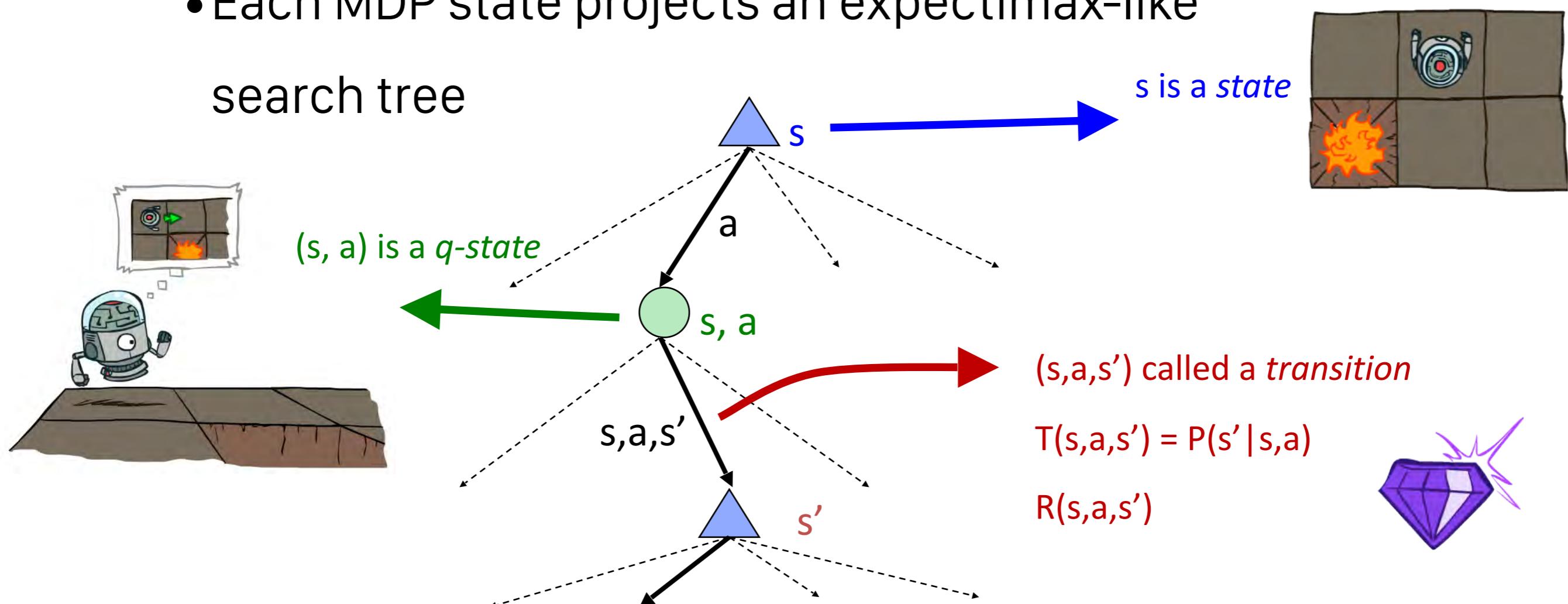
- In deterministic **single-agent** search problems, we wanted an **optimal plan**, or **sequence of actions**, from start to a goal
- For MDPs, we want an **optimal policy**  $\pi^*: S \rightarrow A$ 
  - A policy  $\pi$  gives an action for each state
  - An optimal policy is one that maximizes **expected utility** if followed
  - An explicit policy defines a reflex agent
- Expectimax didn't compute entire policies
  - It computed the action for a single state only



Optimal policy when  $R(s, a, s') = -0.03$  for all non-terminals  $s$

# MDP Search Trees

- Each MDP state projects an expectimax-like search tree



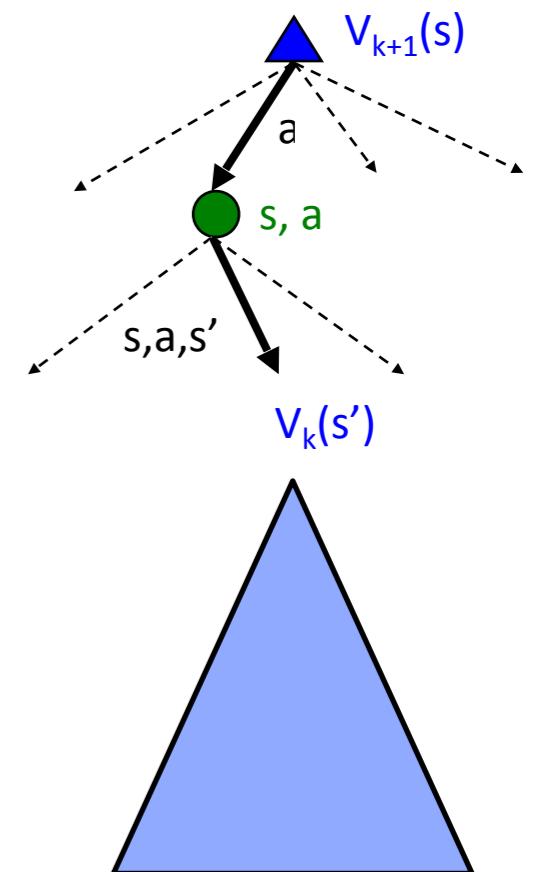
$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

# Value Iteration

- Start with  $V_0(s) = 0$ : no time steps left means an expected reward sum of zero
- Given vector of  $V_k(s)$  values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$



- Repeat until convergence
- Complexity of each iteration:  $O(S^2A)$

# Snapshot of Demo – Gridworld V Values

- The value (utility) of a state  $s$ :  $V^*(s) = \max_a Q^*(s, a)$   
 $V^*(s)$  = expected utility starting in  $s$  and acting optimally
- The value (utility) of a q-state  $(s, a)$ :  
 $Q^*(s, a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally
- The optimal policy:  
 $\pi^*(s)$  = optimal action from state  $s$

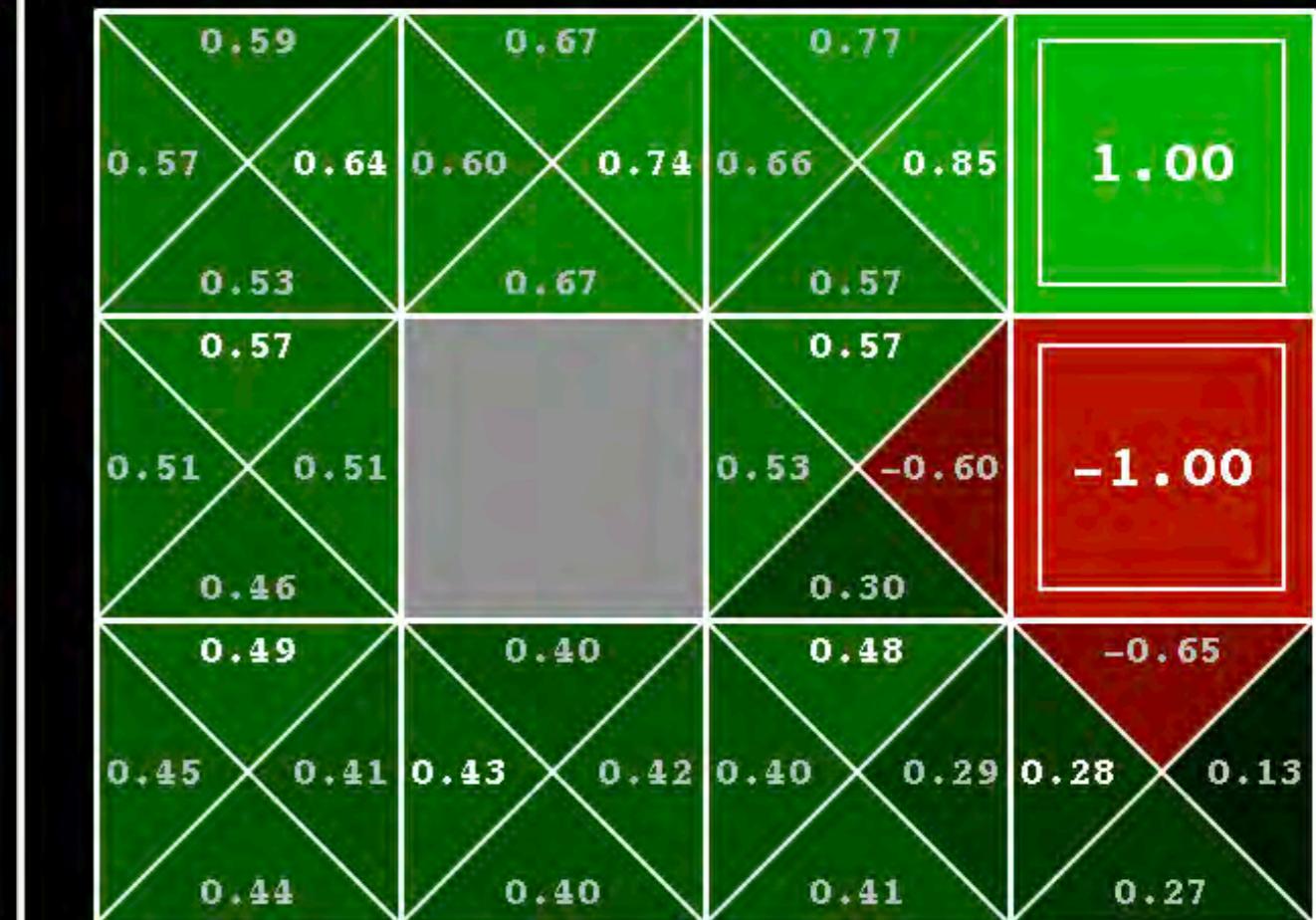
$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

Noise = 0.2

Discount = 0.9



VALUES AFTER 100 ITERATIONS



Q-VALUES AFTER 100 ITERATIONS

# Policy Iteration

- **Policy Evaluation**: For fixed current policy  $\pi$ , find values with policy evaluation:
  - Iterate **until values converge**:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$$

- **Policy Improvement**: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

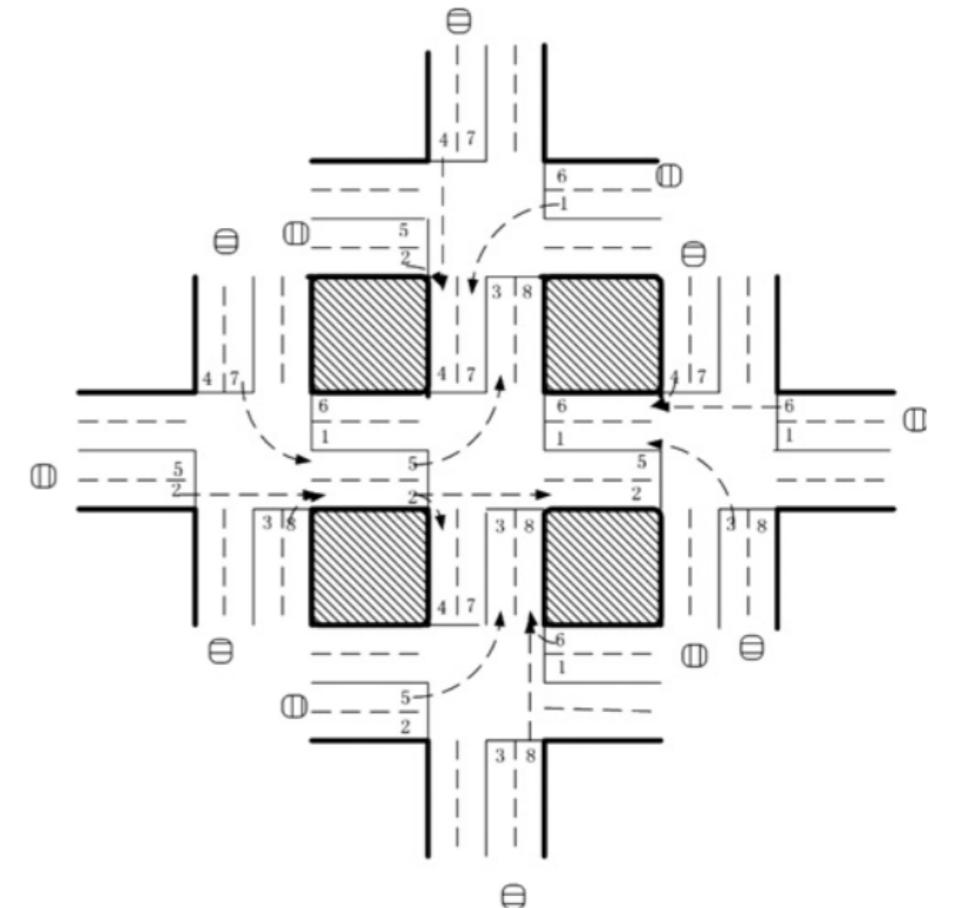
$$\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^{\pi_i}(s')]$$

- Compare with Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

# What is RL in real world Application?

- Resources management in computer system
- Traffic Light Control
- Robotics
- Web System Configuration
- Chemistry
- Personalized Recommendations



# What is RL in real world Application?

- Bidding and Advertising
- Games

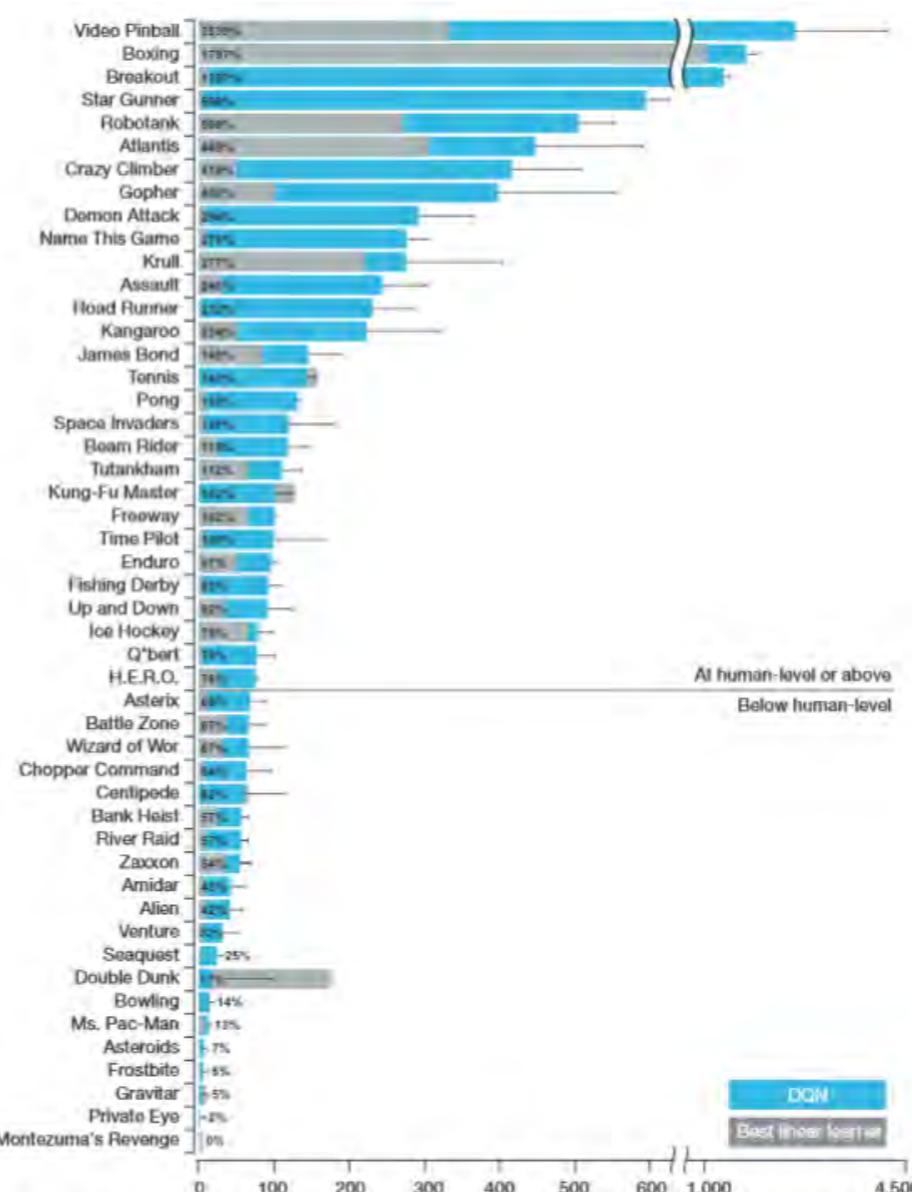
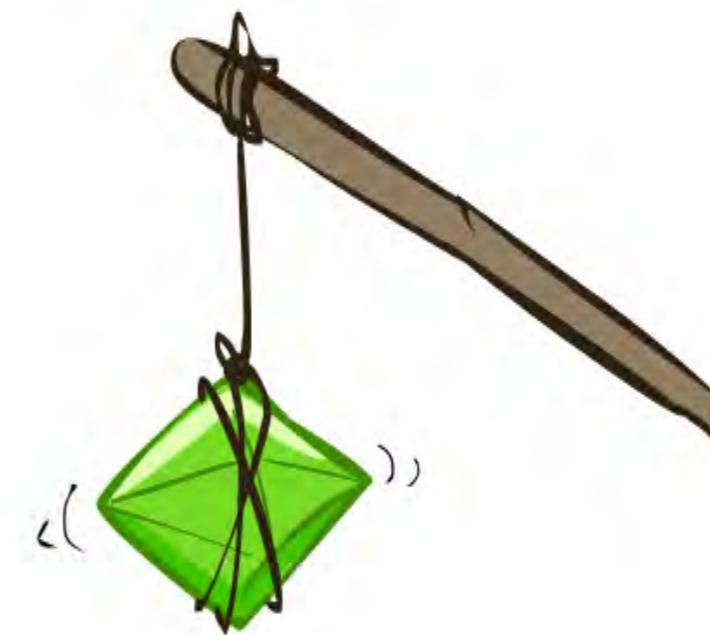


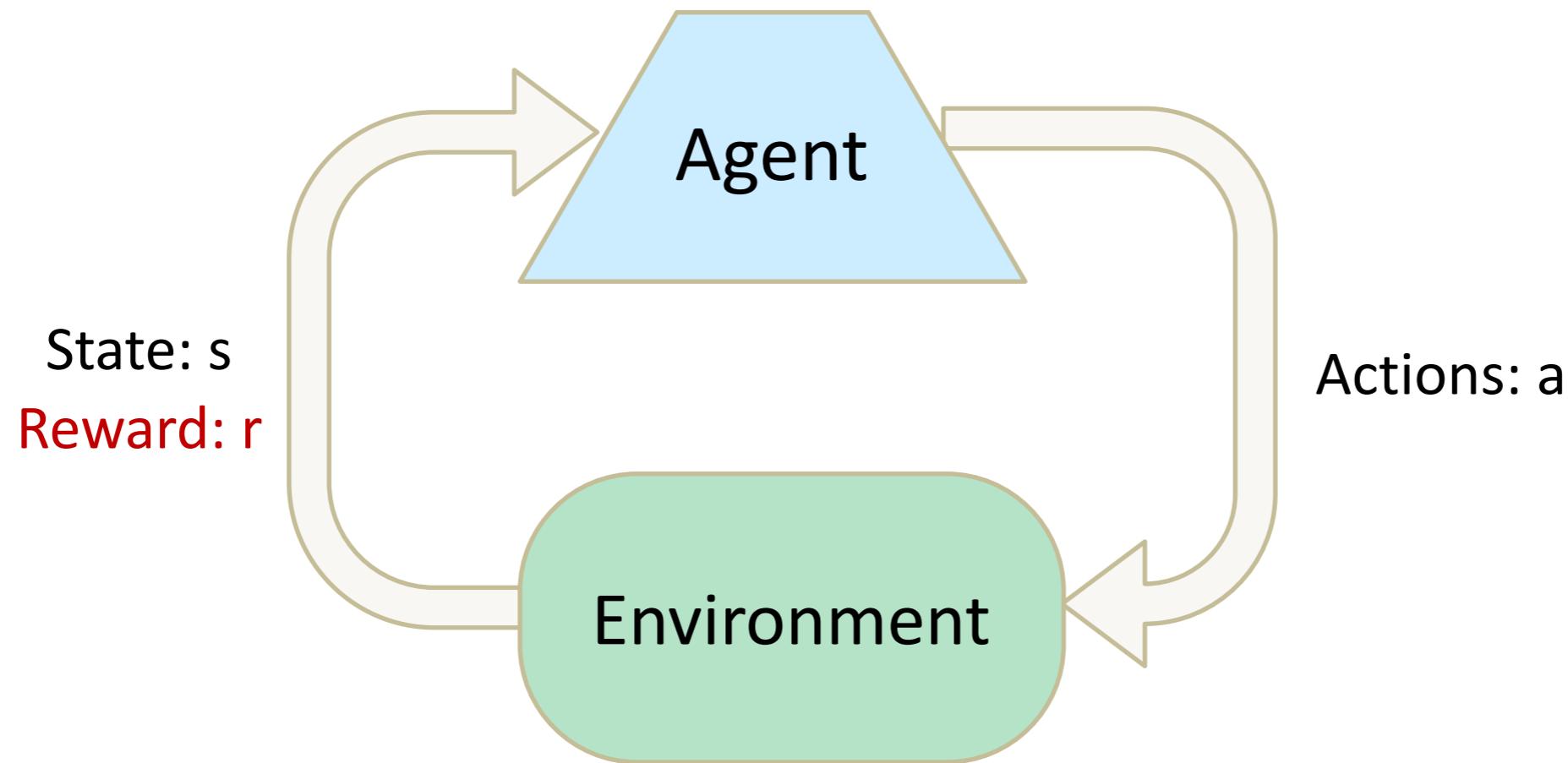
Figure 3 | Comparison of the DQN agent with the best reinforcement learning methods<sup>15</sup> in the literature. The performance of DQN is normalized with respect to a professional human games tester (that is, 100% level) and random play (that is, 0% level). Note that the normalized performance of DQN, expressed as a percentage, is calculated as:  $100 \times (\text{DQN score} - \text{random play score}) / (\text{human score} - \text{random play score})$ . It can be seen that DQN

outperforms competing methods (also see Extended Data Table 2) in almost all the games, and performs at a level that is broadly comparable with or superior to a professional human games tester (that is, operationalized as a level of 75% or above) in the majority of games. Audio output was disabled for both human players and agents. Error bars indicate s.d. across the 30 evaluation episodes, starting with different initial conditions.

# Reinforcement Learning



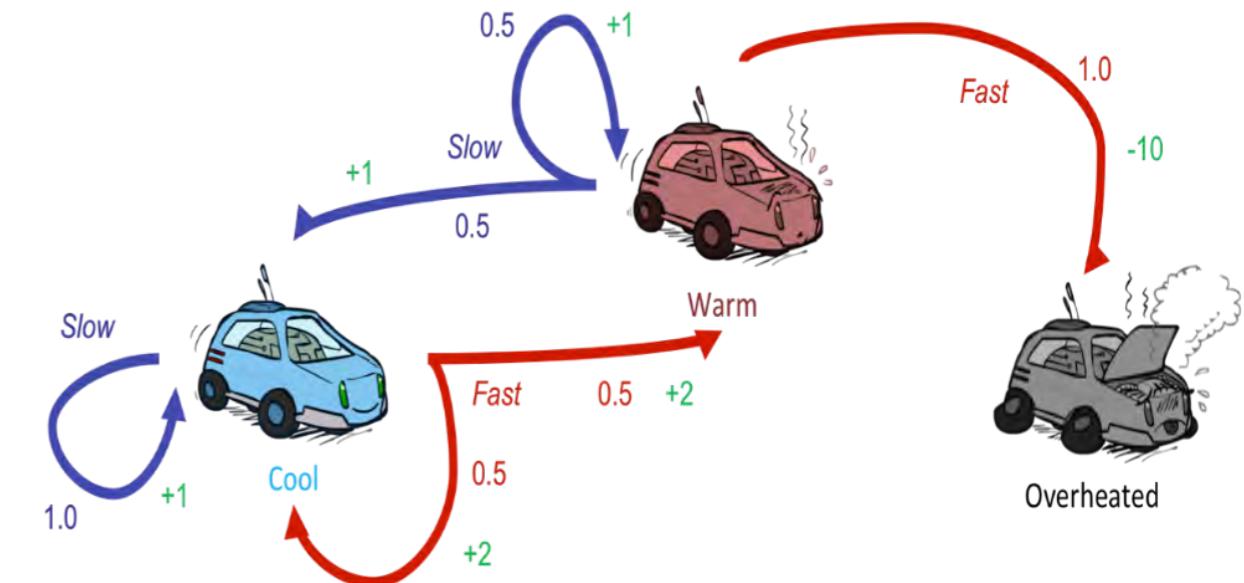
# Reinforcement Learning



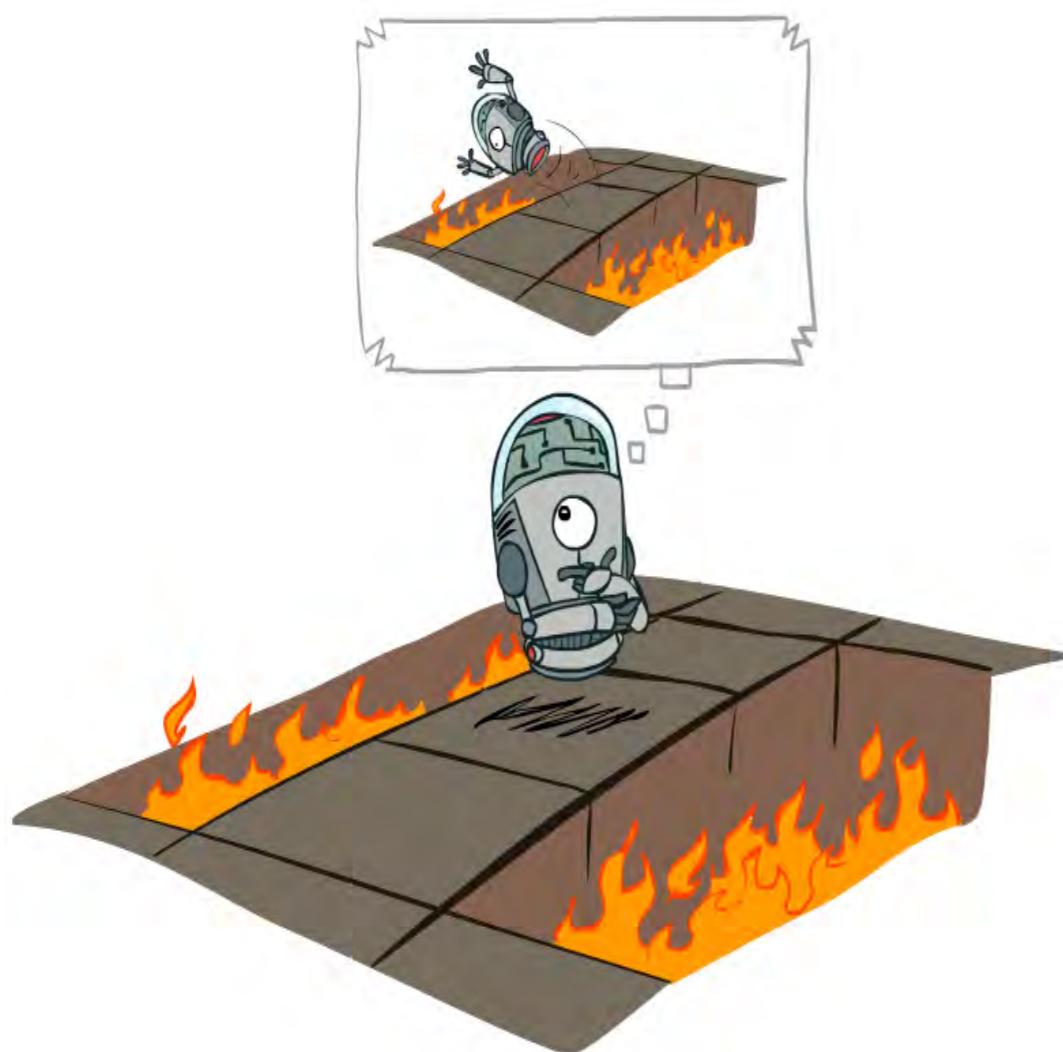
- Basic idea:
  - Receive feedback in the form of **rewards**
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to **maximize expected rewards**
  - All learning is based on observed samples of outcomes!

# Reinforcement Learning

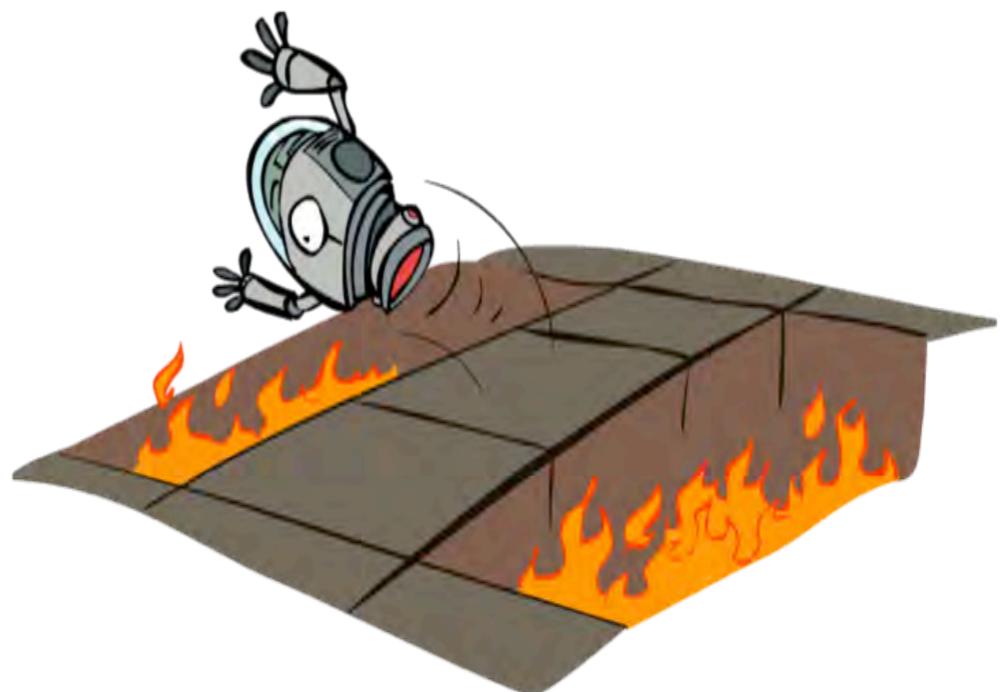
- Still assume a Markov decision process (MDP):
  - A set of states  $s \in S$
  - A set of actions (per state)  $A$
  - A model  $T(s,a,s')$
  - A reward function  $R(s,a,s')$
- Still looking for a policy  $\pi(s)$
- New twist: don't know  $T$  or  $R$ 
  - I.e. we don't know which states are good or what the actions do
  - Must actually try actions and states out to learn



# Offline (MDPs) vs. Online (RL)



Offline  
Solution

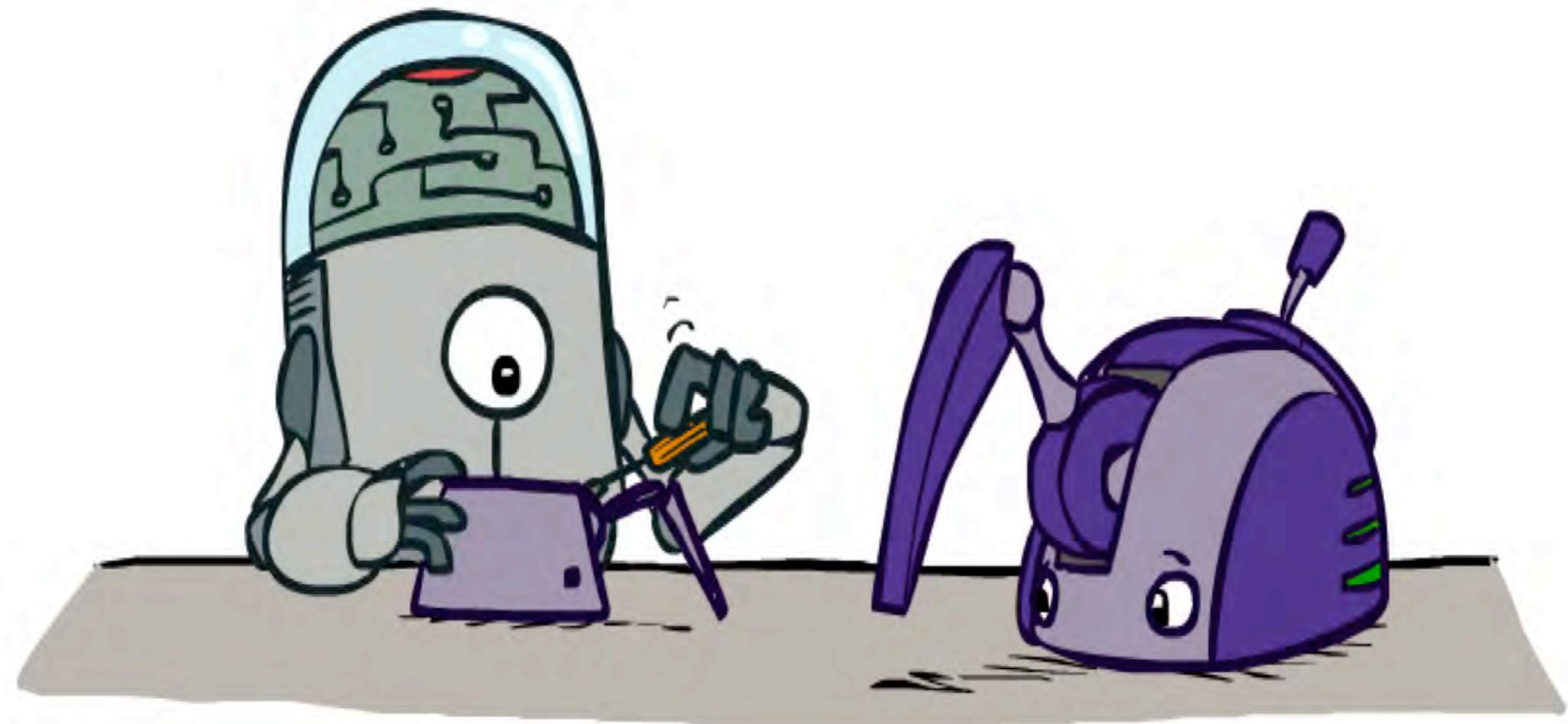


Online  
Learning

# Outline

- **Model-based learning**
  - Learn  $T(s,a,s')$  and  $R(s,a,s')$
- **Model-free learning**
  - Passive Reinforcement Learning
    - Direct evaluation
    - Sample based policy evaluation
    - Temporal Difference Learning
  - Active Reinforcement Learning
    - Q-learning

# Model-Based Learning

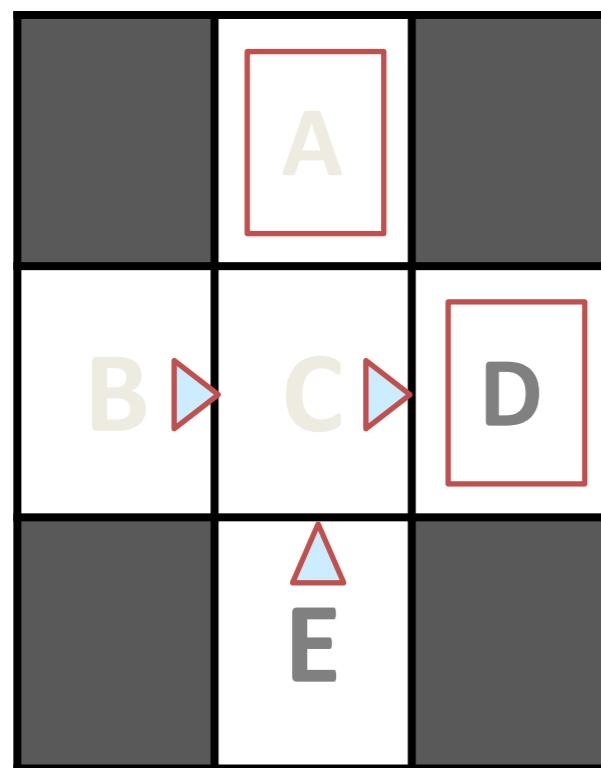


# Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct
$$\hat{T}(s, a, s')$$
- Step 1: Learn empirical MDP model
  - Count outcomes  $s'$  for each  $s, a$
  - Normalize to give an estimate of
  - Discover each  $\hat{R}(s, a, s')$  when we experience  $(s, a, s')$
- Step 2: Solve the learned MDP
  - For example, use value iteration, as before



# Example: Model-Based Learning

| Input Policy $\pi$                                                                | Observed Episodes (Training)                                      | Learned Model                                                                                          |
|-----------------------------------------------------------------------------------|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
|  | Episode 1<br>B, east, C, -1<br>C, east, D, -1<br>D, exit, x, +10  | $\hat{T}(s, a, s')$<br><br>T(B, east, C) = 1.00<br>T(C, east, D) = 0.75<br>T(C, east, A) = 0.25<br>... |
|                                                                                   | Episode 2<br>B, east, C, -1<br>C, east, D, -1<br>D, exit, x, +10  |                                                                                                        |
|                                                                                   | Episode 3<br>E, north, C, -1<br>C, east, D, -1<br>D, exit, x, +10 | $\hat{R}(s, a, s')$<br><br>R(B, east, C) = -1<br>R(C, east, D) = -1<br>R(D, exit, x) = +10<br>...      |
| Assume: $\gamma = 1$                                                              | Episode 4<br>E, north, C, -1<br>C, east, A, -1<br>A, exit, x, -10 |                                                                                                        |

# Example: Expected Age

Goal: Compute expected age of students

Known P(A)

$$E[A] = \sum_a P(a) \cdot a = 0.35 * 20 + \dots$$

Without P(A), instead collect samples  $[a_1, a_2, \dots a_N]$

Unknown P(A): “Model Based”

Why does this work? Because eventually you learn the right model.

$$\hat{P}(a) = \frac{\text{num}(a)}{N}$$

$$E[A] \approx \sum_a \hat{P}(a) \cdot a$$

Unknown P(A): “Model Free”

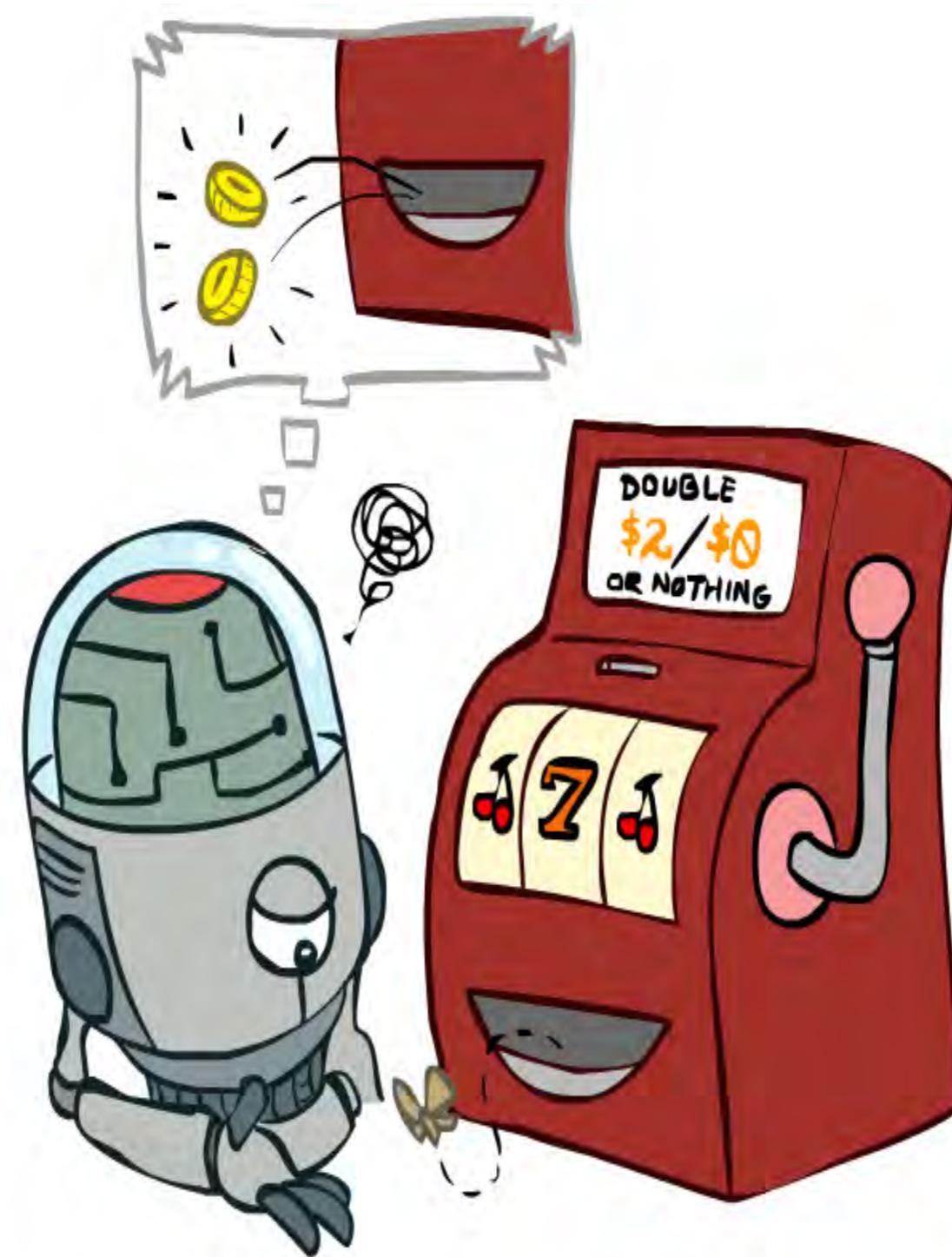
$$E[A] \approx \frac{1}{N} \sum_a a$$

Why does this work? Because samples appear with the right frequencies.

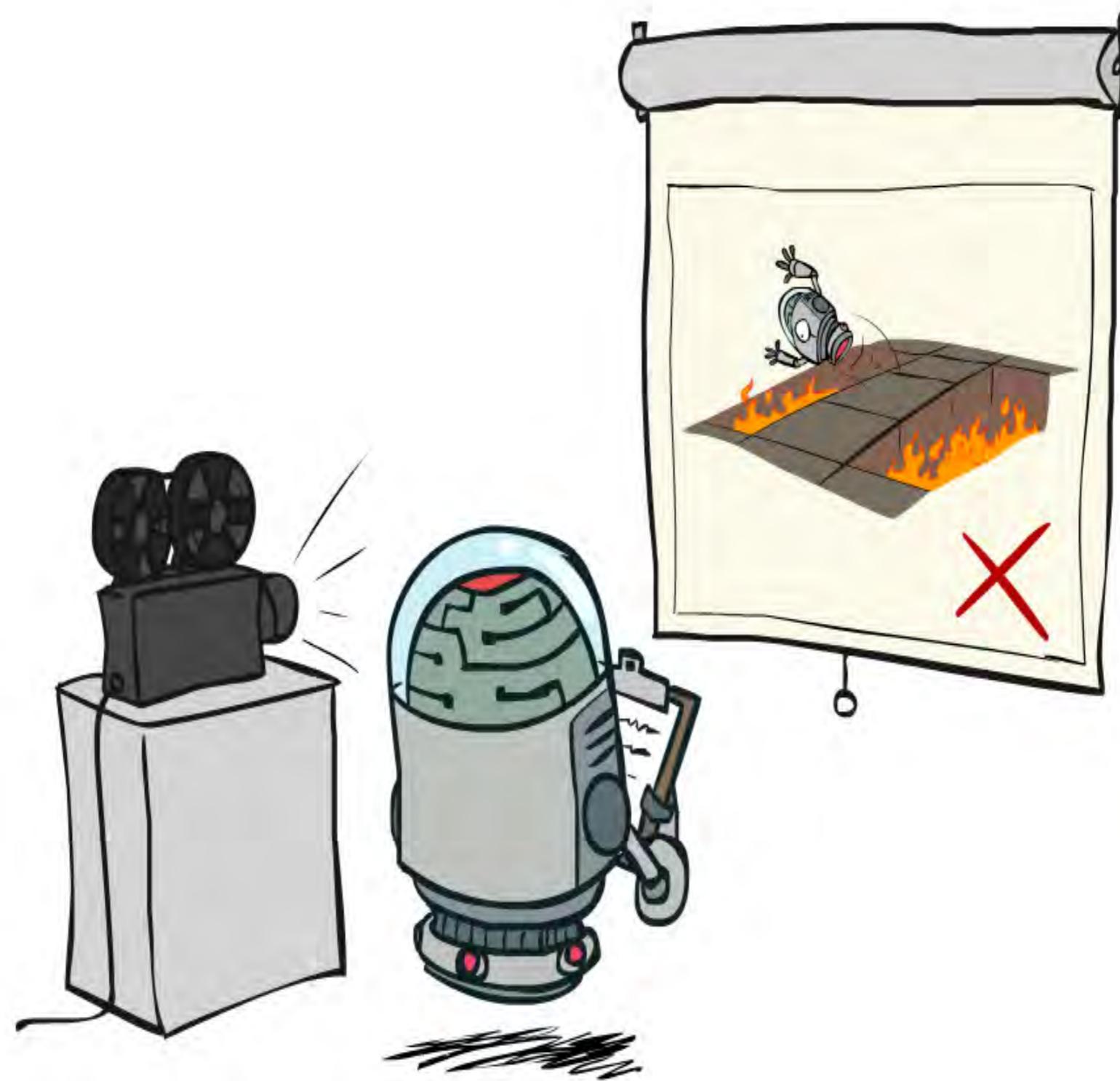
# Outline

- Model-based learning
  - Learn  $T(s,a,s')$  and  $R(s,a,s')$
- Model-free learning
  - **Passive Reinforcement Learning**
    - Direct evaluation
    - Sample based policy evaluation
    - Temporal Difference Learning
  - **Active Reinforcement Learning**
    - Q-learning

# Model-Free Learning

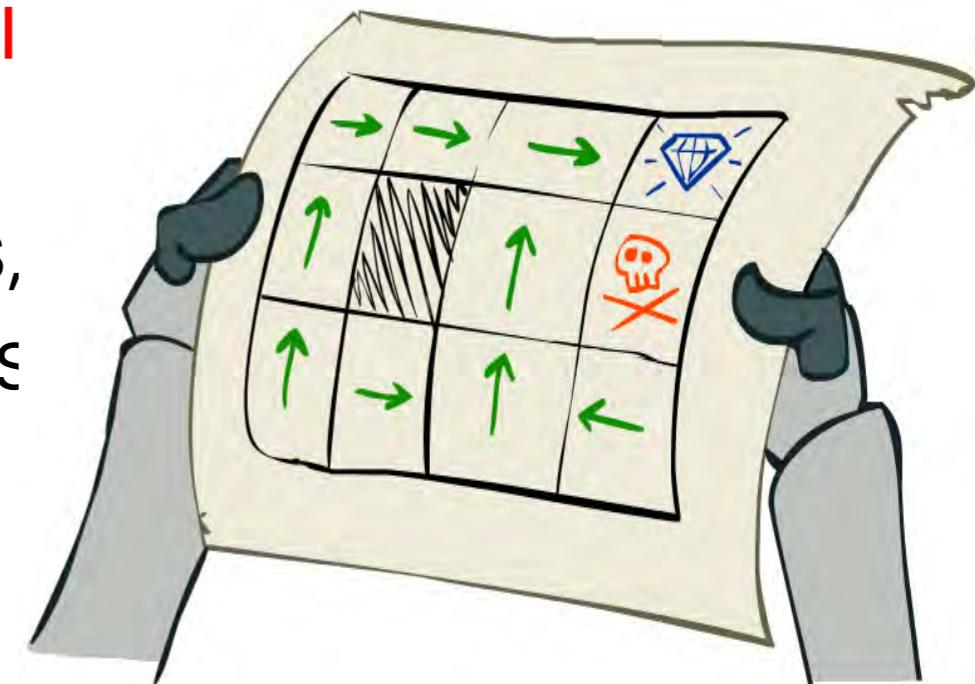


# Passive Reinforcement Learning



# Passive Reinforcement Learning

- Simplified task: policy evaluation
  - Input: a fixed policy  $\pi(s)$
  - You don't know the transitions  $T(s, a, s')$
  - You don't know the rewards  $R(s, a, s')$
  - Goal: learn the state values
- In this case:
  - Learner is “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning! You actually take actions in the world.



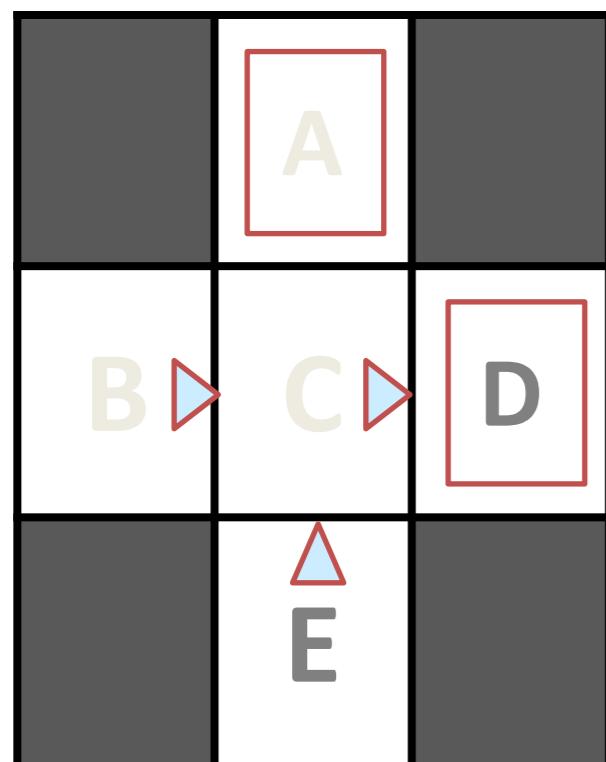
# Direct Evaluation

- Goal: Compute values for each state under  $\pi$
- Idea: Average together observed sample values
  - Act according to  $\pi$
  - Every time you visit a state, write down what **the sum of discounted rewards turned out to be**
  - Average those samples
- This is called direct evaluation



# Example: Direct Evaluation

Input  
Policy  $\pi$



Assume:  $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 2

B, east, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 3

E, north, C, -1  
C, east, D, -1  
D, exit, x, +10

Episode 4

E, north, C, -1  
C, east, A, -1  
A, exit, x, -10

Output Values

|   |     |    |
|---|-----|----|
|   | -10 |    |
| A | +8  | +4 |
| B | C   | D  |
|   | -2  |    |
| E |     |    |

Model Free!

# Problems with Direct Evaluation

- What's good about direct evaluation?
  - It's easy to understand
  - It doesn't require any **knowledge** of T, R
  - It eventually computes the correct average values, using just sample transitions
- What bad about it?
  - It wastes information about state connections
  - Each state must be learned separately
  - So, it takes a long time to learn

Output Values

|         |          |          |
|---------|----------|----------|
|         | -10<br>A |          |
| +8<br>B | +4<br>C  | +10<br>D |
|         | -2<br>E  |          |

*If B and E both go to C under this policy, how can their values be different?  
--Does not re-use C's value!*

# Outline

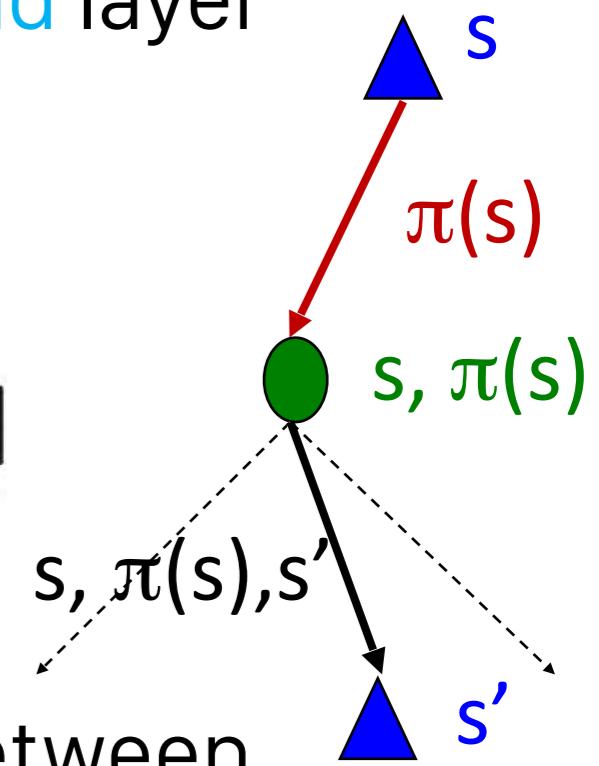
- Model-based learning
  - Learn  $T(s,a,s')$  and  $R(s,a,s')$
- Model-free learning
  - Passive Reinforcement Learning
    - Direct evaluation
    - Sample based policy evaluation
    - Temporal Difference Learning
  - Active Reinforcement Learning
    - Q-learning

# Why Not Use Policy Evaluation?

- Simplified Bellman updates calculate  $V$  for a fixed policy:
- Each round, replace  $V$  with a **one-step-look-ahead** layer over  $V$

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



- This approach fully exploited the connections between the states
- Unfortunately, we need **T** and **R** to do it!
- Key question: how can we do this update to  $V$  without knowing **T** and **R**?
- In other words, how to we take a weighted average without knowing the weights?

# Sample-Based Policy Evaluation?

- We want to improve our estimate of  $V$  by computing these averages:

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^{\pi}(s')]$$

- Idea: Take samples of outcomes  $s'$  (by doing the action!) and average

$$\text{sample}_1 = R(s, \pi(s), s'_1) + \gamma V_k^{\pi}(s'_1)$$

$$\text{sample}_2 = R(s, \pi(s), s'_2) + \gamma V_k^{\pi}(s'_2)$$

...

$$\text{sample}_n = R(s, \pi(s), s'_n) + \gamma V_k^{\pi}(s'_n)$$

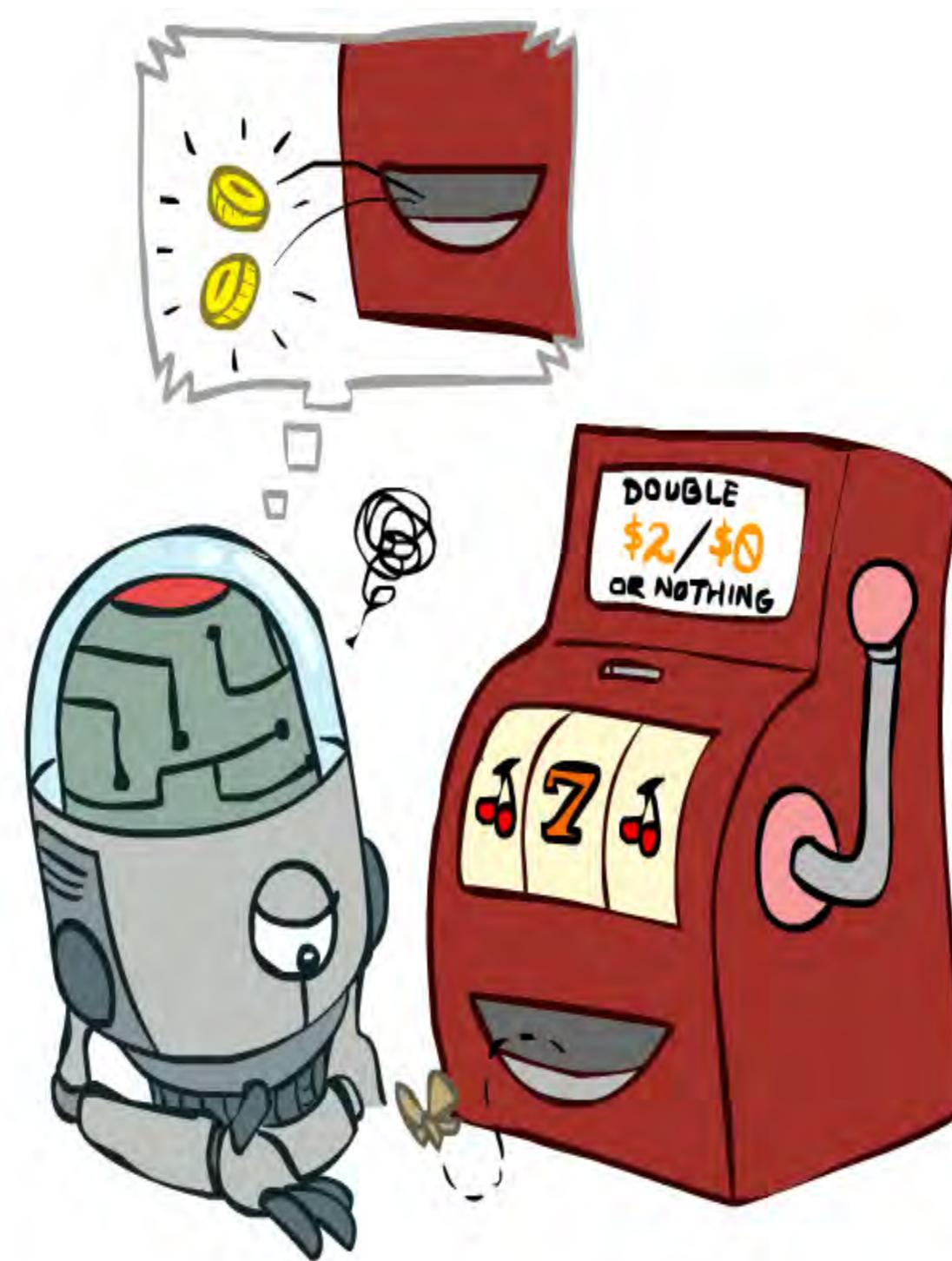
$$V_{k+1}^{\pi}(s) \leftarrow \frac{1}{n} \sum_i \text{sample}_i$$

*Almost! Still not online learning.  
We can't rewind time to get  
sample after sample from state  $s$ .*

# Outline

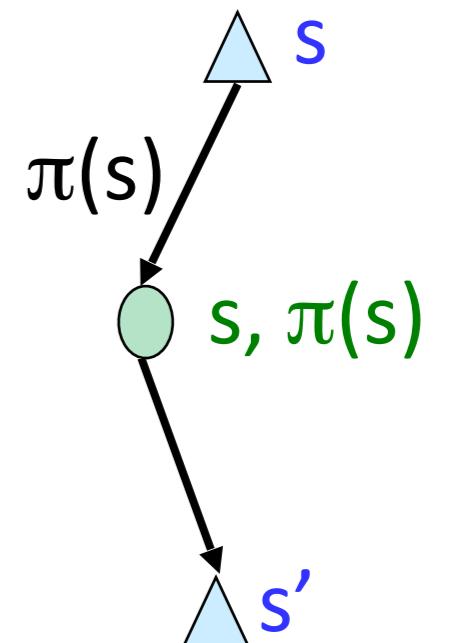
- Model-based learning
  - Learn  $T(s,a,s')$  and  $R(s,a,s')$
- Model-free learning
  - Passive Reinforcement Learning
    - Direct evaluation
    - Sample based policy evaluation
    - Temporal Difference Learning
  - Active Reinforcement Learning
    - Q-learning

# Temporal Difference Learning



# Temporal Difference Learning

- Big idea: learn from every experience!
  - Update  $V(s)$  each time we experience a transition  $(s, a, s', r)$
  - Likely outcomes  $s'$  will contribute updates more often
- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average
  - Better estimate of the future after running more times of RL
    - Slowly forget the past



Sample of  $V(s)$ :  $\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')$

Update to  $V(s)$ :  $V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha)\text{sample}$

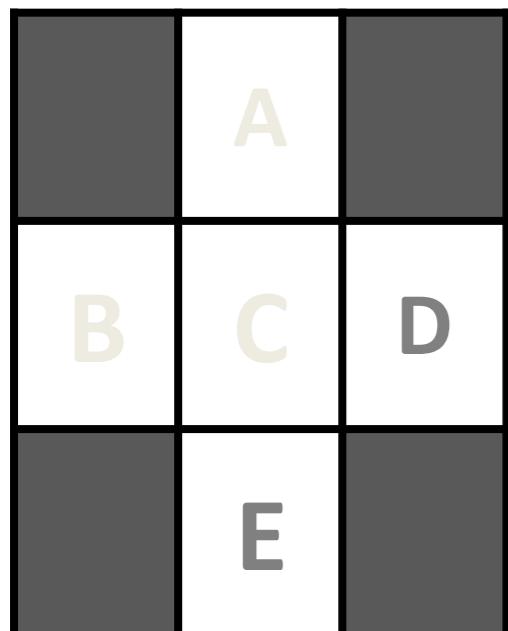
Same update:  $V^\pi(s) \leftarrow V^\pi(s) + \alpha(\text{sample} - \underline{\underline{V^\pi(s)}})$

What actually  
happened

What we  
thought

# Example: Temporal Difference Learning

States



Observed Transitions

B, east, C, -2

C, east, D, -2

|   |   |   |
|---|---|---|
|   | 0 |   |
| 0 | 0 | 8 |
|   | 0 |   |

|    |   |   |
|----|---|---|
|    | 0 |   |
| -1 | 0 | 8 |
|    | 0 |   |

|    |   |   |
|----|---|---|
|    | 0 |   |
| -1 | 3 | 8 |
|    | 0 |   |

Assume:

$$\gamma = 1,$$

$$\alpha = 1/2$$

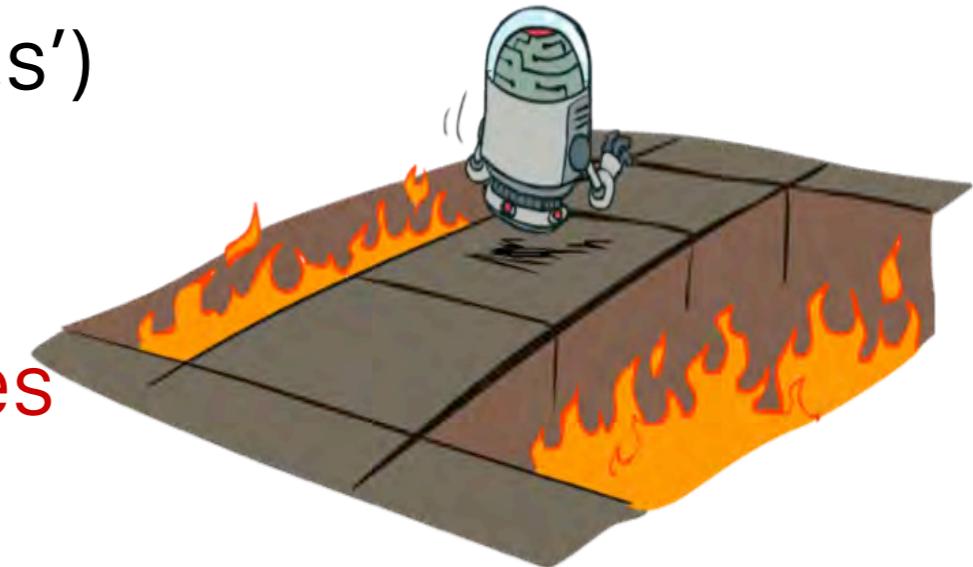
$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [R(s, \pi(s), s') + \gamma V^\pi(s')]$$

# Outline

- Model-based learning
  - Learn  $T(s,a,s')$  and  $R(s,a,s')$
- Model-free learning
  - Passive Reinforcement Learning
    - Direct evaluation
    - Sample based policy evaluation
    - Temporal Difference Learning
  - Active Reinforcement Learning
    - Q-learning

# Active Reinforcement Learning

- Full reinforcement learning: **optimal policies** (like value iteration)
  - You don't know the transitions  $T(s,a,s')$
  - You don't know the rewards  $R(s,a,s')$
  - You choose the actions now
  - **Goal: learn the optimal policy / values**
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: **exploration vs. exploitation**
  - This is NOT offline planning! You actually take actions in the world and find out what happens...



# Detour: Q-Value Iteration

- **Value iteration:** find successive (depth-limited) values
  - Start with  $V_0(s) = 0$ , which we know is right
  - Given  $V_k$ , calculate the depth  $k+1$  values for all states:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

- But **Q-values** are more useful, so compute them instead
  - Start with  $Q_0(s, a) = 0$ , which we know is right
  - Given  $Q_k$ , calculate the depth  $k+1$  q-values for all q-states:

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$$

$$V^*(s) = \max_a Q^*(s, a)$$

# Snapshot of Demo – Gridworld V Values

- The value (utility) of a state  $s$ :

$$V^*(s) = \max_a Q^*(s, a)$$

$V^*(s)$  = expected utility starting in  $s$  and acting optimally

- The value (utility) of a q-state  $(s, a)$ :

$Q^*(s, a)$  = expected utility starting out having taken action  $a$  from state  $s$  and (thereafter) acting optimally

- The optimal policy:

$\pi^*(s)$  = optimal action from state  $s$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$$

Noise = 0.2

Discount = 0.9



VALUES AFTER 100 ITERATIONS



Q-VALUES AFTER 100 ITERATIONS

# Q-Learning

- Q-Learning: sample-based Q-value iteration

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right]$$

- Learn  $Q(s, a)$  values as you go

- Receive a sample  $(s, a, s', r)$
- Consider your old estimate:  $Q(s, a)$
- Consider your new sample estimate:

$$\text{sample} = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

- Incorporate the new estimate into a running average:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) [\text{sample}]$$



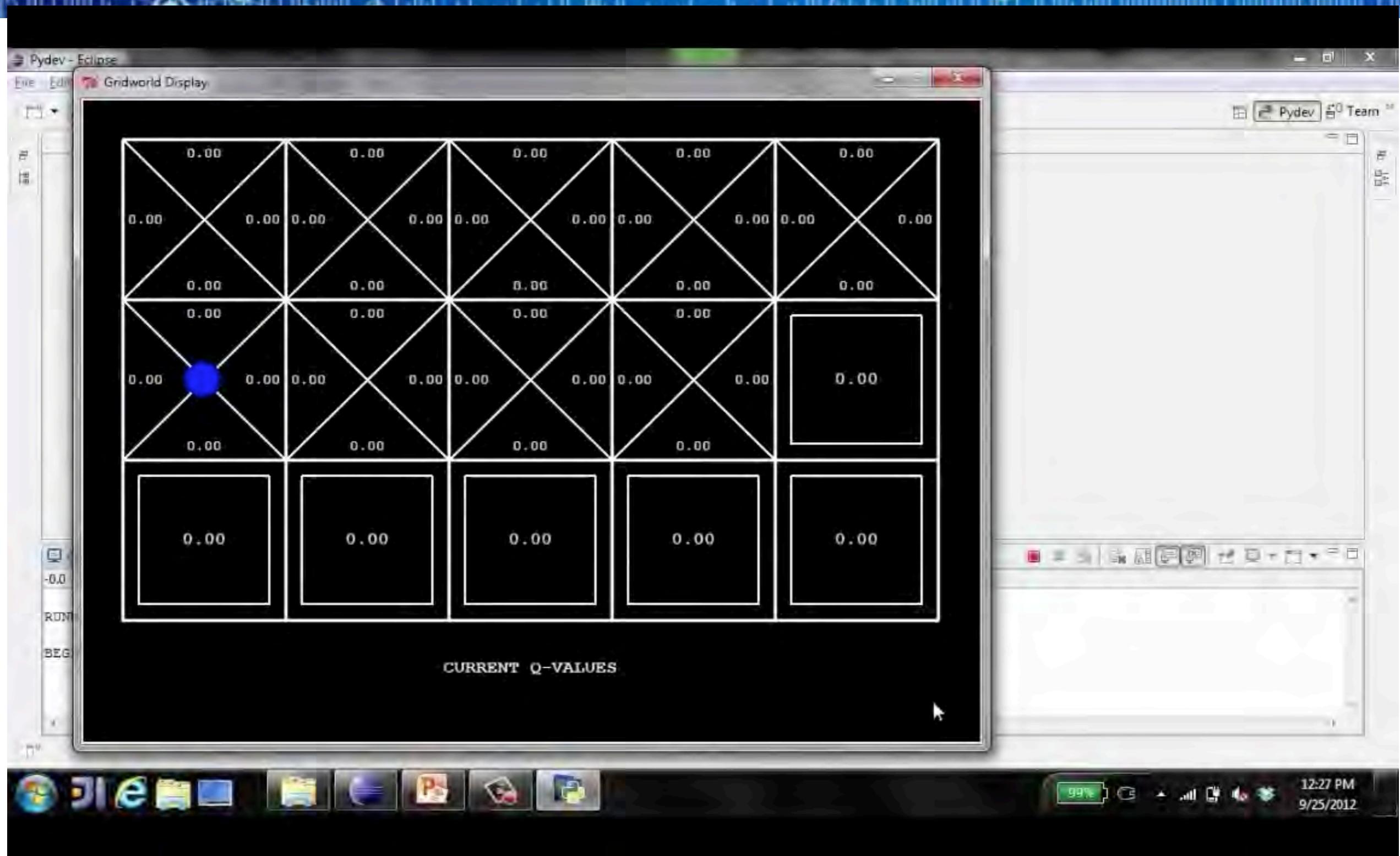
Compare:

$$\text{sample} = R(s, \pi(s), s') + \gamma V^\pi(s')$$

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + (\alpha) \text{sample}$$

$$V^\pi(s) \leftarrow V^\pi(s) + \alpha(\text{sample} - V^\pi(s))$$

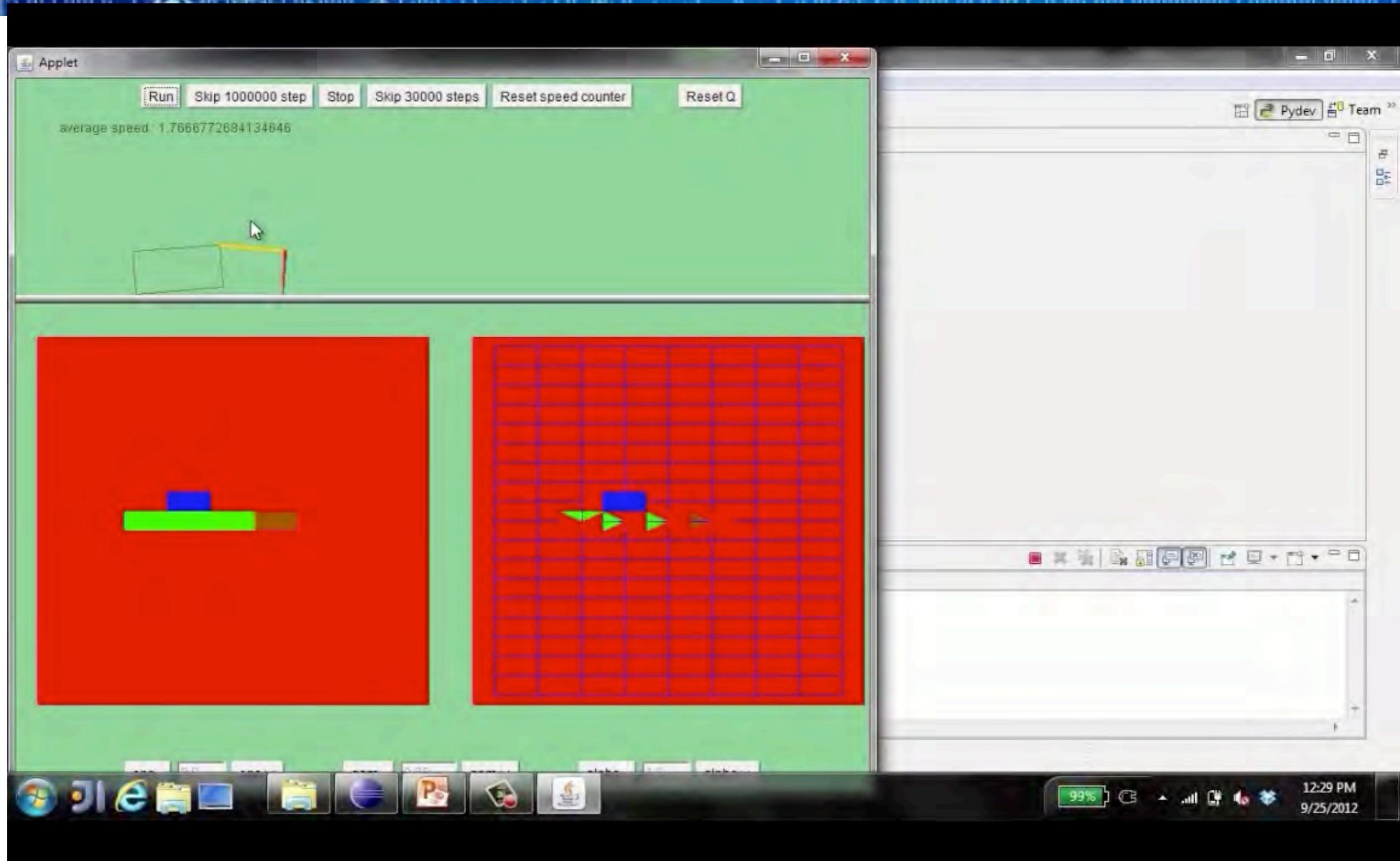
# Video of Demo Q-Learning -- Gridworld



# Video of Demo Crawler Bot

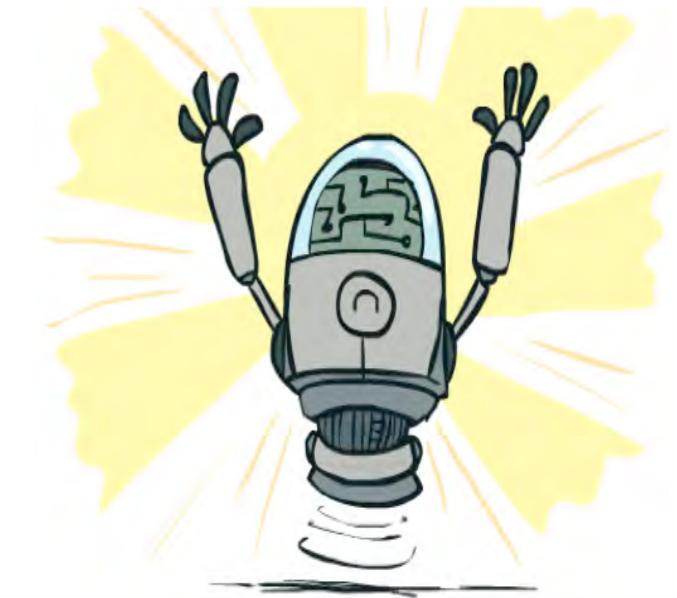


# Video of Demo Q-Learning -- Crawler



# Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy  
-- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - ... but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



# The Story So Far: MDPs and RL

## Known MDP: Offline Solution

| Goal                            | Technique                |
|---------------------------------|--------------------------|
| Compute $V^*$ , $Q^*$ , $\pi^*$ | Value / policy iteration |
| Evaluate a fixed policy $\pi$   | Policy evaluation        |

## Unknown MDP: Model-Based

| Goal                            | Technique            |
|---------------------------------|----------------------|
| Compute $V^*$ , $Q^*$ , $\pi^*$ | VI/PI on approx. MDP |
| Evaluate a fixed policy $\pi$   | PE on approx. MDP    |

## Unknown MDP: Model-Free

| Goal                            | Technique      |
|---------------------------------|----------------|
| Compute $V^*$ , $Q^*$ , $\pi^*$ | Q-learning     |
| Evaluate a fixed policy $\pi$   | Value Learning |





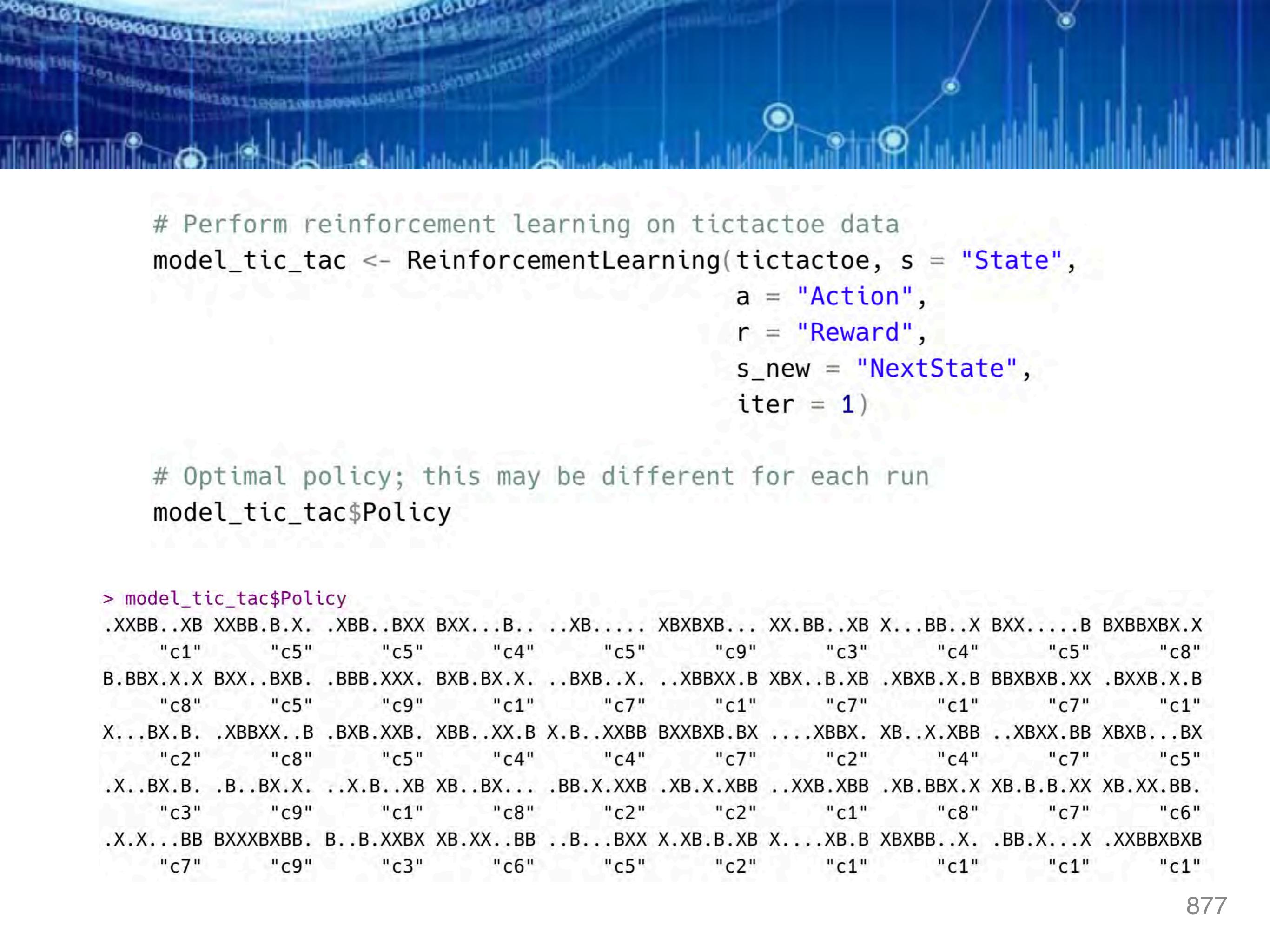
# Reinforcement Learning using R

# Example 1

```
install_github("nproellochs/ReinforcementLearning")
library(ReinforcementLearning)

Load dataset
data("tictactoe")
tail(tictactoe,5)

> tail(tictactoe,5)
 State Action NextState Reward
406537 ...B...X c2 .X.B.B..X 0
406538 .X.B.B..X c3 .XXBBB..X -1
406539 c8 .B.....X 0
406540 .B.....X. c7 .BB...XX. 0
406541 .BB...XX. c9 .BB...XXX 1
>
```



```
Perform reinforcement learning on tictactoe data
model_tic_tac <- ReinforcementLearning(tictactoe, s = "State",
 a = "Action",
 r = "Reward",
 s_new = "NextState",
 iter = 1)

Optimal policy; this may be different for each run
model_tic_tac$Policy

> model_tic_tac$Policy
.XXBB..XB XXBB.B.X. .XBB..BXX BXX...B.. .XB..... XBXBXB... XX.BB..XB X...BB..X BXX.....B BXBBXBX.X
 "c1" "c5" "c5" "c4" "c5" "c9" "c3" "c4" "c5" "c8"
B.BBX.X.X BXX..BXB. .BBB.XXX. BXB.BX.X. ..BXB..X. ..XBBXX.B XBX..B.XB .XBXB.X.B BBXBXB.XX .BXXB.X.B
 "c8" "c5" "c9" "c1" "c7" "c1" "c7" "c1" "c7" "c1"
X...BX.B. .XBBXX..B .BXB.XXB. XBB..XX.B X.B..XXBB BXXXBXB.BXXBBX. XB..X.XBB ..XBXX.BB XBXB...BX
 "c2" "c8" "c5" "c4" "c4" "c7" "c2" "c4" "c7" "c5"
.X..BX.B. .B..BX.X. ..X.B..XB XB..BX... .BB.X.XXB .XB.X.XBB ..XXB.XBB .XB.BBX.X XB.B.B.XX XB.XX.BB.
 "c3" "c9" "c1" "c8" "c2" "c2" "c1" "c8" "c7" "c6"
.X.X...BB BXXXBXBB. B..B.XXBX XB.XX..BB ..B...BXX X.XB.B.XB X....XB.B XBXB..X. .BB.X...X .XXBBXBXB
 "c7" "c9" "c3" "c6" "c5" "c2" "c1" "c1" "c1" "c1"
```

```
Reward; this may be different for each run
model_tic_tac$Reward
```

```
> model_tic_tac$Reward
[1] 5449
```

# Example 2

```
print(gridworldEnvironment)
|
> print(gridworldEnvironment)
function (state, action)
{
 next_state <- state
 if (state == state("s1") && action == "down")
 next_state <- state("s2")
 if (state == state("s2") && action == "up")
 next_state <- state("s1")
 if (state == state("s2") && action == "right")
 next_state <- state("s3")
 if (state == state("s3") && action == "left")
 next_state <- state("s2")
 if (state == state("s3") && action == "up")
 next_state <- state("s4")
 if (next_state == state("s4") && state != state("s4")) {
 reward <- 10
 }
 else {
 reward <- -1
 }
 out <- list(NextState = next_state, Reward = reward)
 return(out)
}
<bytecode: 0x7f807c3b61c8>
<environment: namespace:ReinforcementLearning>
```



```
states <- c("s1", "s2", "s3", "s4")
actions <- c("up", "down", "left", "right")

Generate 1000 iterations
sequences <- sampleExperience(N = 1000,
 env = gridworldEnvironment,
 states = states,
 actions = actions)
```

> sequences

|   | State | Action | Reward | NextState |    |    |       |    |    |
|---|-------|--------|--------|-----------|----|----|-------|----|----|
| 1 | s2    | up     | -1     | s1        | 45 | s4 | up    | -1 | s4 |
| 2 | s2    | right  | -1     | s3        | 46 | s3 | up    | 10 | s4 |
| 3 | s2    | right  | -1     | s3        | 47 | s2 | left  | -1 | s2 |
| 4 | s4    | left   | -1     | s4        | 48 | s4 | up    | -1 | s4 |
| 5 | s3    | down   | -1     | s3        | 49 | s1 | right | -1 | s1 |
| 6 | s4    | up     | -1     | s4        | 50 | s2 | up    | -1 | s1 |

```
#Solve the problem
solver_rl <- ReinforcementLearning(sequences,
 s = "State",
 a = "Action",
 r = "Reward",
 s_new = "NextState")

#Getting the policy;
solver_rl$Policy

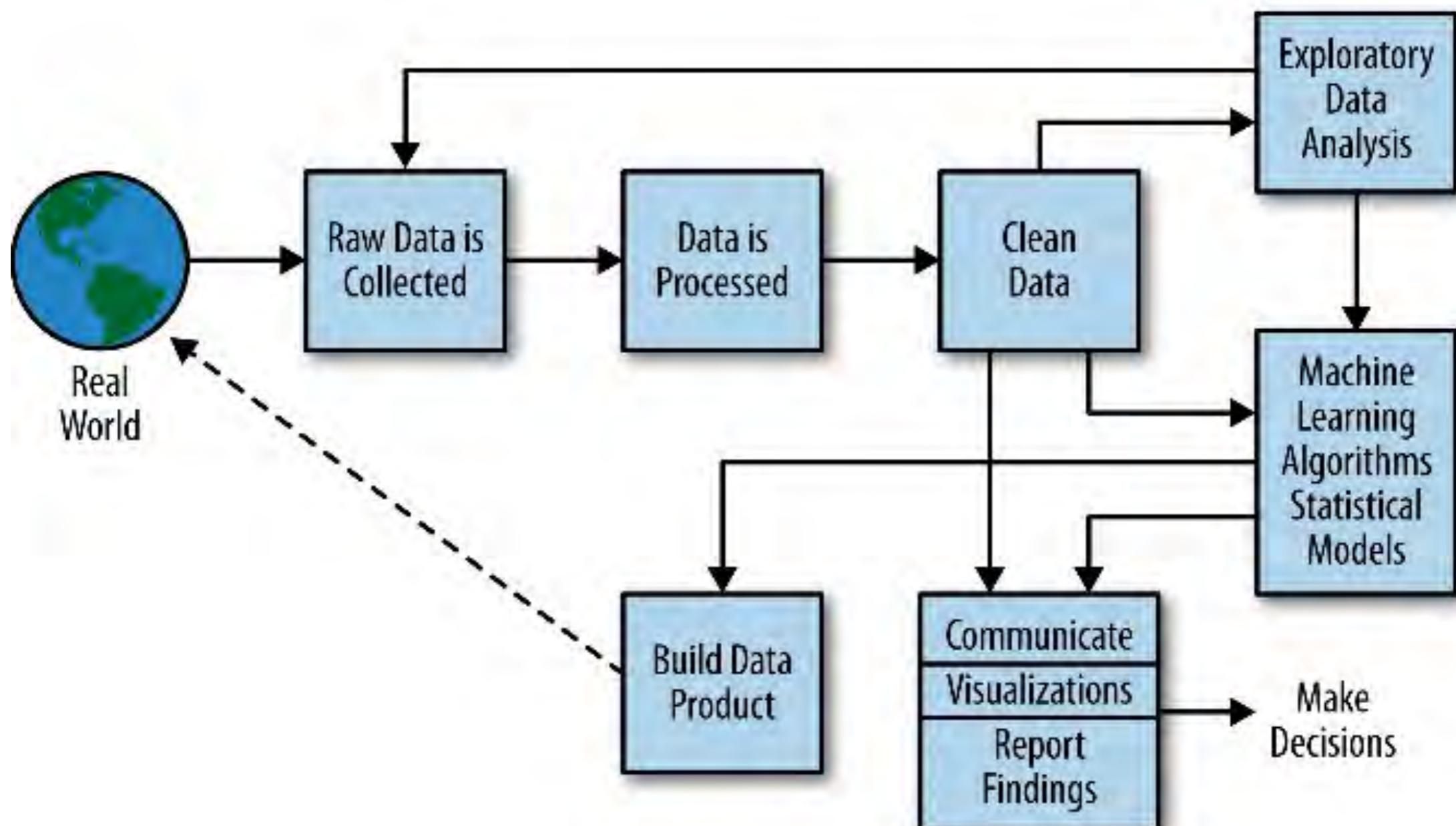
> solver_rl$Policy
 s1 s2 s3 s4
"down" "right" "up" "down"
```

```
#Getting the Reward;
solver_rl$Reward
```

```
> solver_rl$Reward
[1] -384
```



# Conclusion





# Contact

Contact :

Email: [veerasak.kritsanapraphan@gmail.com](mailto:veerasak.kritsanapraphan@gmail.com)

Twitter: @veerasakk