Universiteit Leiden

Business Intelligence & Process Modelling

Frank Takes

Universiteit Leiden

Lecture 5 — Reinforcement Learning & Python

Recap

- **Business Intelligence**: anything that aims at providing actionable information that can be used to support business decision making
  - Business Intelligence
  - Visual Analytics
  - **Descriptive Analytics**
  - **Predictive Analytics**
- Process Modelling (April and May)

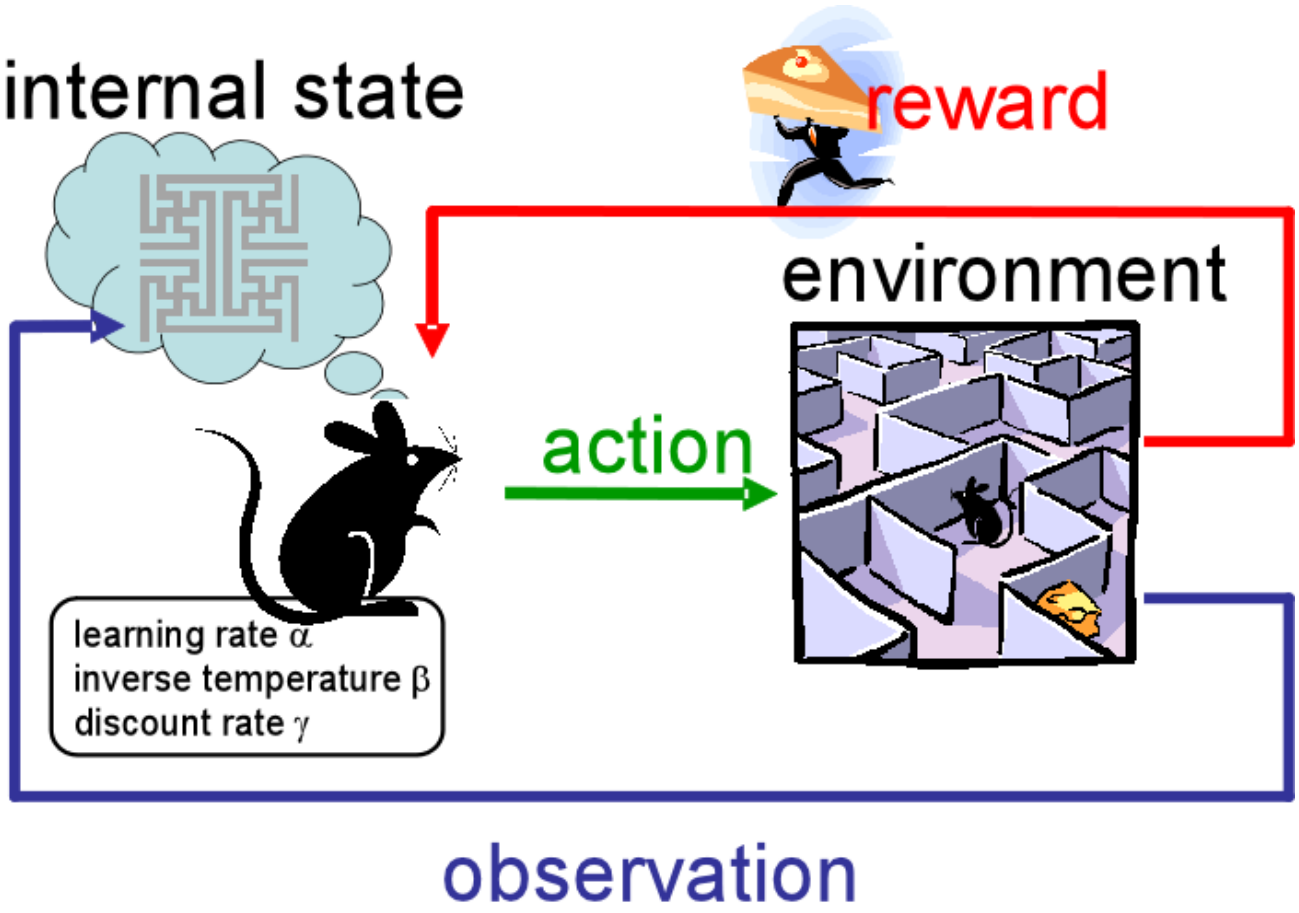
Categories of techniques

- Machine learning
  - Supervised learning: learning on labeled data
  - Semi-supervised learning: partially labeled data
  - Unsupervised learning: leaning/mining on unlabeled data
  - **Reinforcement learning**: agents learning to act in an environment

Universiteit Leiden

Reinforcement learning

Reinforcement learning



https://www.cs.utexas.edu/~eladlieb/rl_interaction.png

Google Deepmind

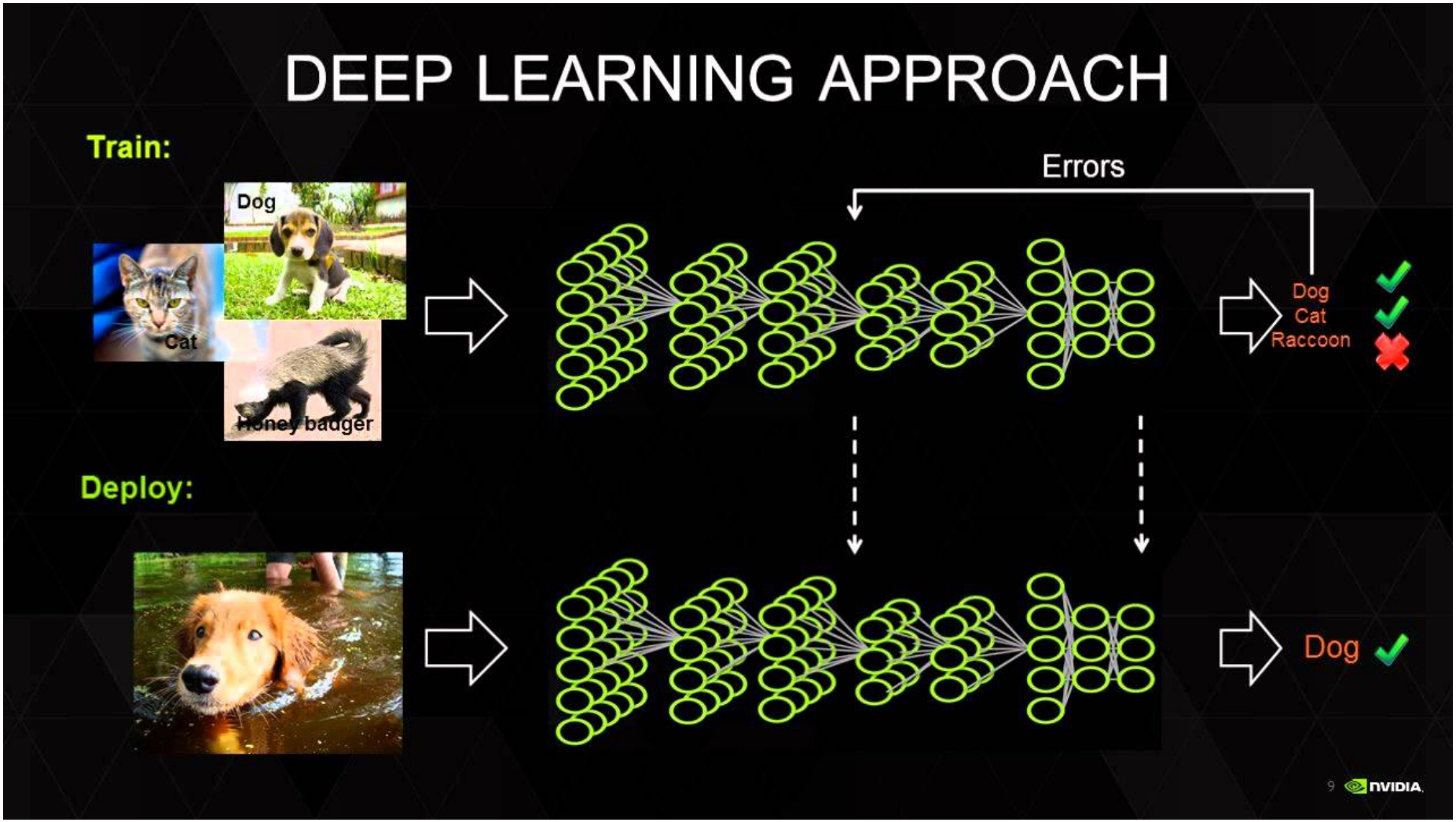
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis, **Human-level control through deep reinforcement learning**, *Nature* 518, 529–533, 2015.

http://dx.doi.org/10.1038/nature14236

Google Deepmind

- Silver et al.. **Mastering the game of Go with deep neural networks and tree search**, *Nature* 529, 484–489, 2016.

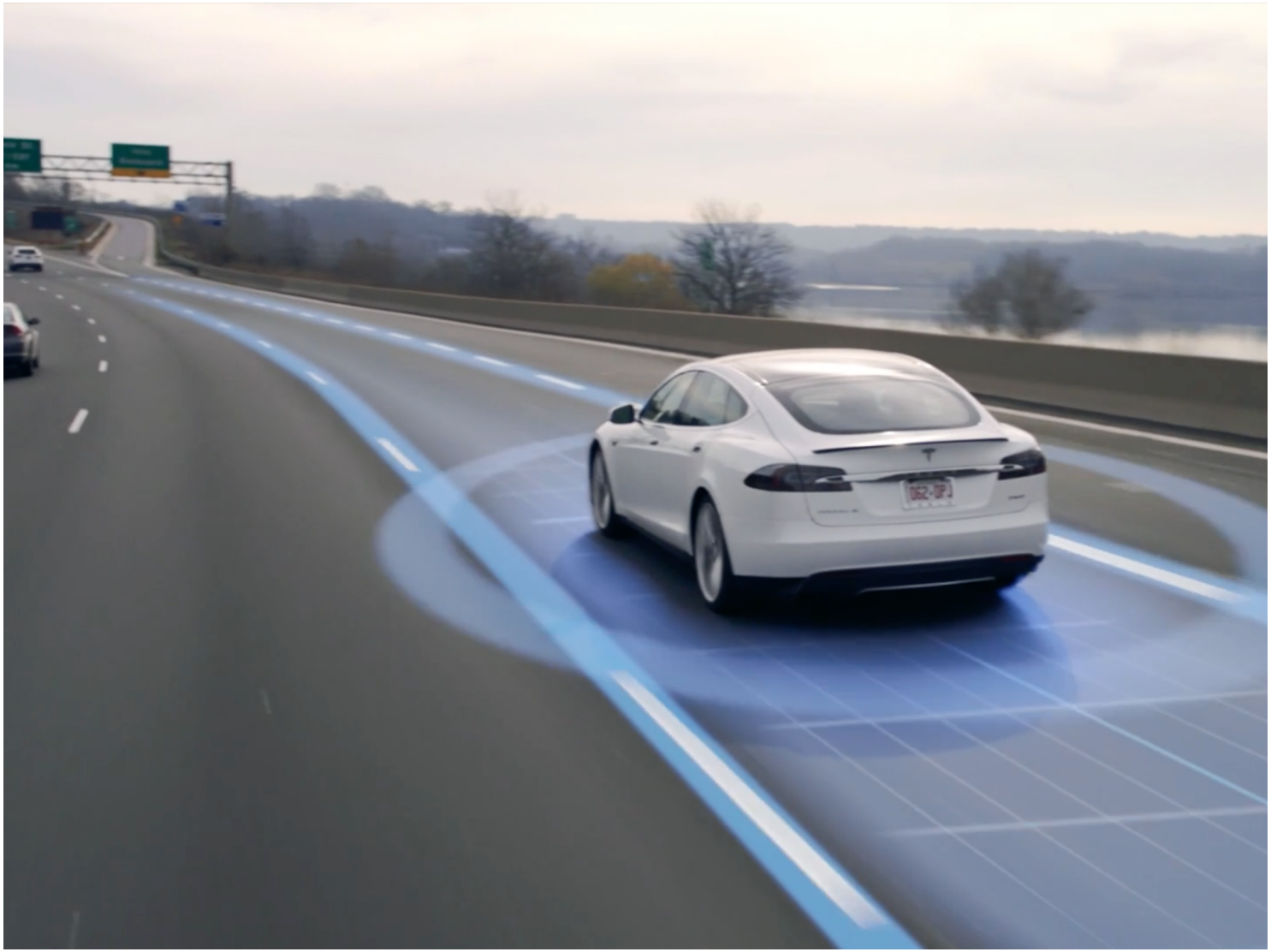
Deep learning



Watson wins Jeopardy

https://www.youtube.com/watch?v=YgYSv2KSyWg

Self-driving cars

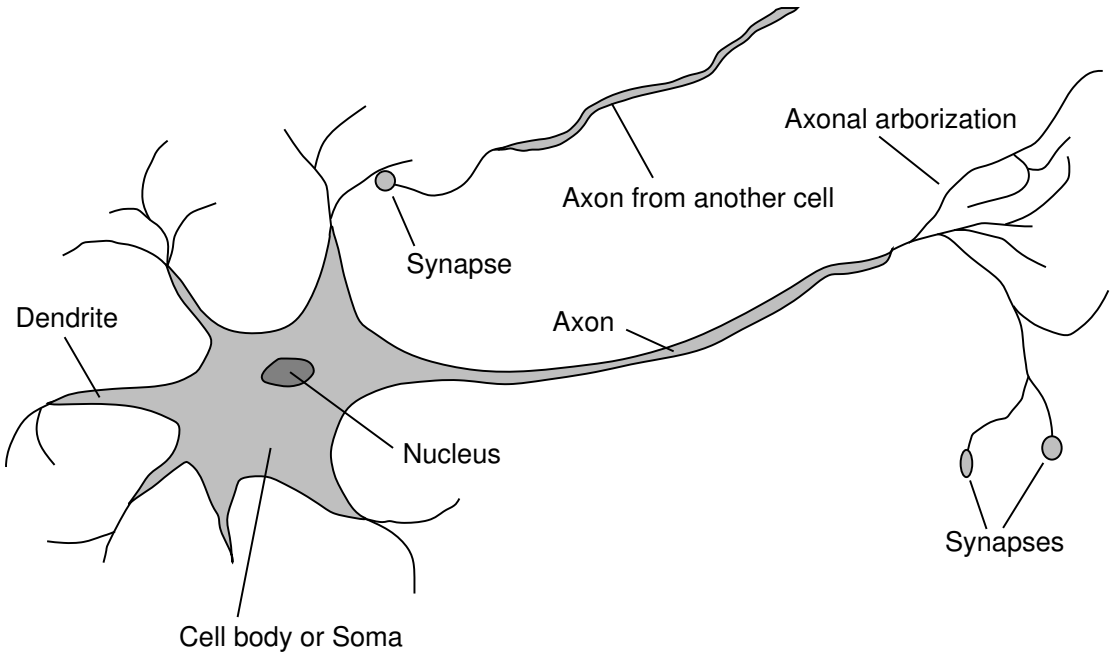


Universiteit Leiden

Neurale netwerken

Hersenen

De menselijke hersenen bestaan uit 10^{11} neuronen (grootte ≈ 0.1 mm; meer dan 20 types), die onderling met axonen (lengte ≈ 1 cm) en synapsen verbonden zijn:

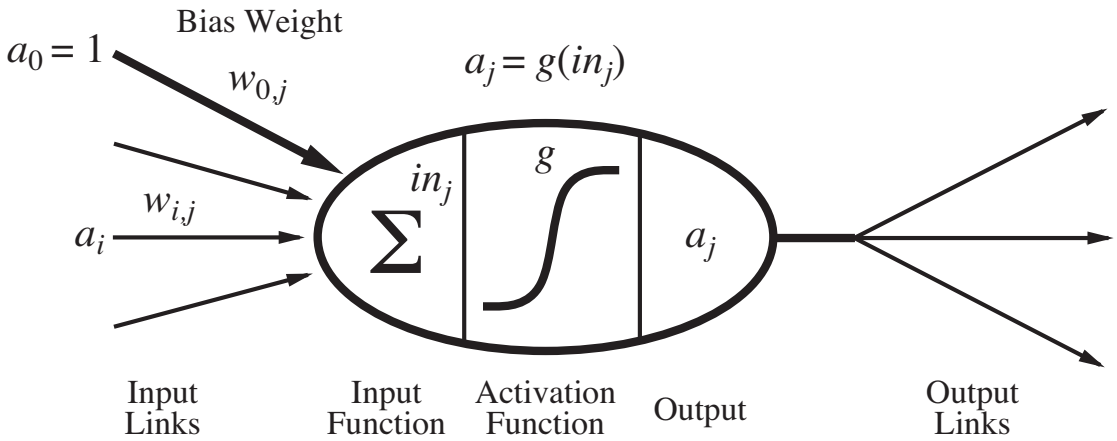


Werking hersenen

- Signalen worden doorgegeven via een nogal gecompliceerde electro-chemische reactie.
- Als de electrische potentiaal van het cellichaam een zekere drempelwaarde haalt, wordt een puls/actie-potentiaal/spike-train op het axon gezet.
- Er zijn excitatory (verhogen potentiaal) en inhibitory (verlagen potentiaal) synapsen.

Units

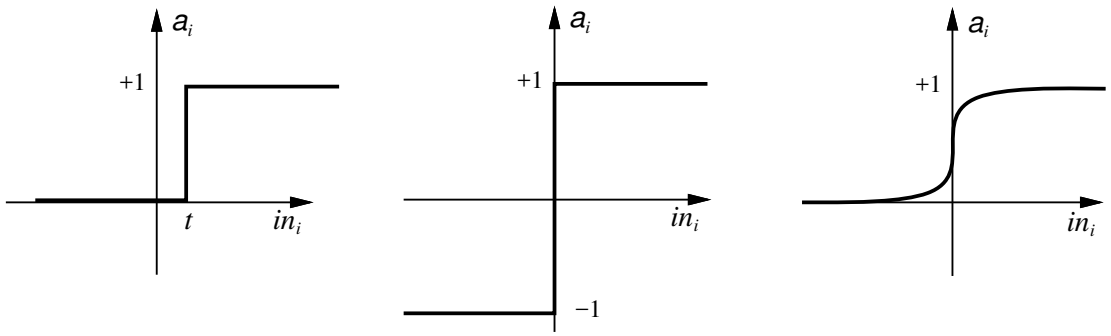
We modelleren de neuronen door middel van eenheden (units) die we ook weer neuronen noemen:



Er geldt: de input van neuron i is $in_i = \sum_j W_{j,i}\, a_j$ (de gewogen som van de inputs), waarbij de a_j's de activaties van de inkomende verbindingen (inputs) zijn, en $W_{j,i}$ hun gewichten ($W_{j,i}$ weegt de verbinding tussen de twee neuronen j en i). De activatie (output) van neuron i is $a_i = g(in_i)$, met g de activatie-functie.

Activatie-functies

Veelgebruikte activatie-functies zijn:



(a) Step function **(b) Sign function** **(c) Sigmoid function**

$\text{step}_t(x) = 1$ als $x \geq t$; 0 als $x < t$ (drempelwaarde t)
$\text{sign}(x) = 1$ als $x \geq 0$; -1 als $x < 0$
$\text{sigmoid}(x) = 1/(1 + e^{-\beta x})$ (vaak kiest men $\beta = 1$)

Bias-knopen

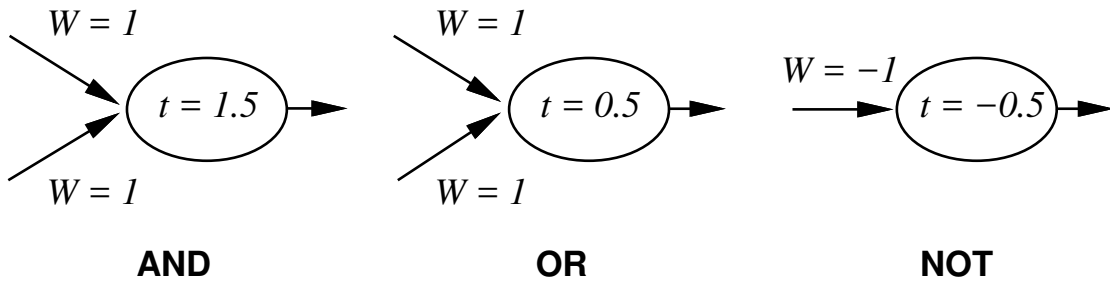
De drempelwaarde t binnen de neuronen kan "gesimuleerd" worden door een extra (constante) activatie -1 in een zogeheten bias-knoop 0 met gewicht $W_{0,i} = t$ op de verbinding van 0 naar i:

$$\text{step}_t \left(\sum_{j=1}^{n} W_{j,i}\ a_j \right) = \text{step}_0 \left(\sum_{j=0}^{n} W_{j,i}\ a_j \right)$$

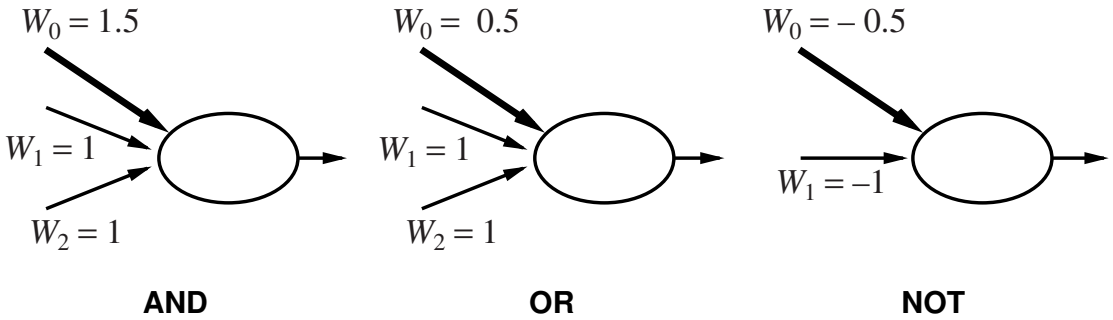
Zo worden drempelwaardes en gewichten uniform behandeld.
De functie step_0 heet ook wel de Heaviside-functie, en lijkt op de functie sign.

Booleaanse functies

Alle Booleaanse functies kunnen gerepresenteerd worden door netwerken met geschikte neuronen:

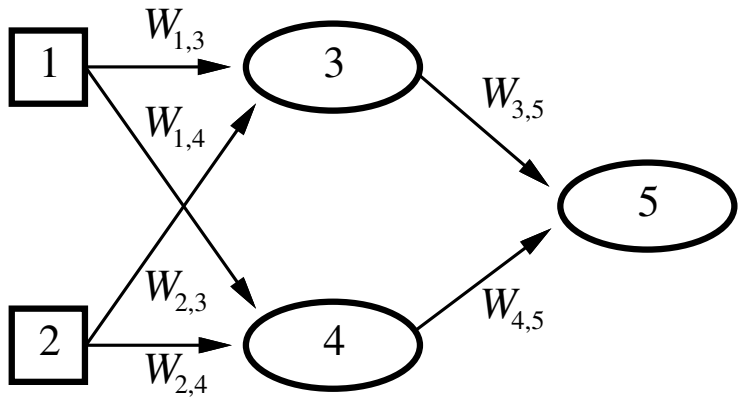


En mét bias-knopen:



Soorten netwerken

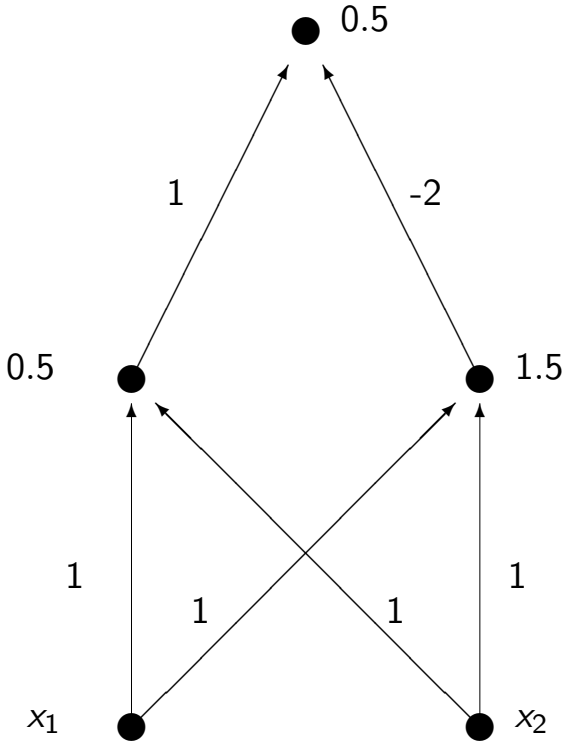
Een feed-forward netwerk heeft gerichte takken en geen cykels (i.t.t. een recurrent netwerk). Vaak is zo'n netwerk in lagen georganiseerd:

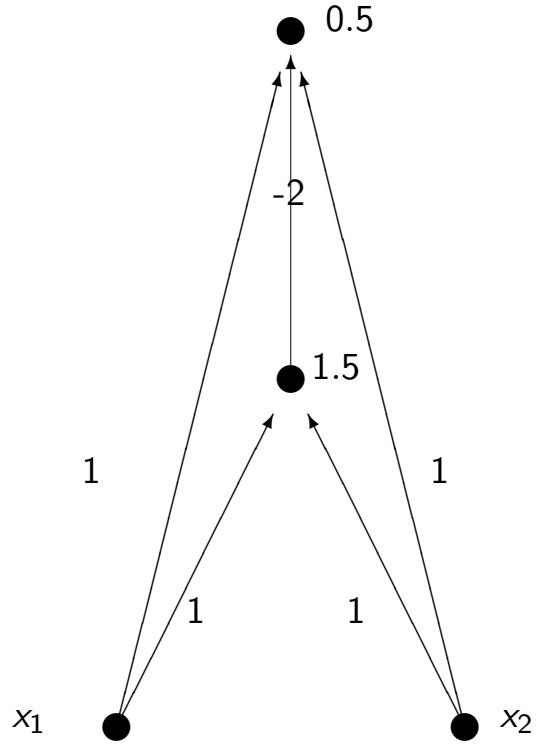


Dit netwerk heeft twee input-knopen 1 en 2, twee verborgen (= hidden) knopen 3 en 4, en één uitvoer-knoop 5. Het representeert de functie:

$$\begin{aligned} a_5 &= g(W_{3,5}\, a_3 + W_{4,5}\, a_4) \\ &= g(W_{3,5}\, g(W_{1,3}\, a_1 + W_{2,3}\, a_2) \\ &\quad + W_{4,5}\, g(W_{1,4}\, a_1 + W_{2,4}\, a_2)) \end{aligned}$$

Voorbeelden





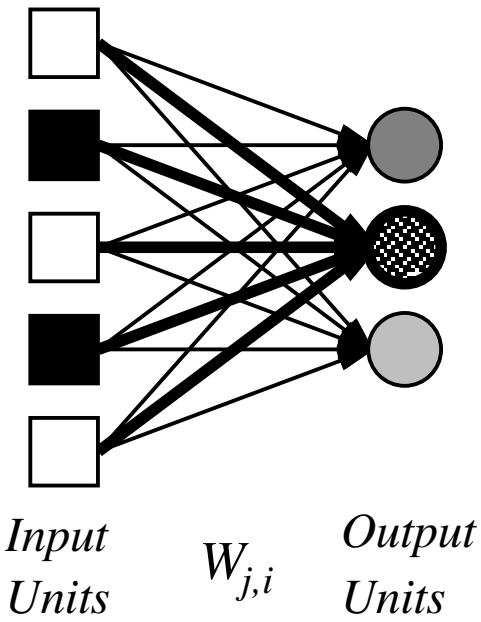
De drempels staan naast de knopen.
Het linker feed-forward netwerk representeert de `XOR`-functie, de verborgen units zijn in feite een `OR` en een `AND`.
Het rechter netwerk representeert dezelfde functie, maar is niet "conventioneel" (geen lagen).

Feed-forward netwerken

Een feed-forward netwerk zonder verborgen neuronen heet een perceptron. Een meerlaags (= multi-layer) netwerk heeft één of meer verborgen lagen, en alle pijlen van laag ℓ gaan naar laag $(\ell + 1)$.
Met één (voldoende grote) laag met verborgen units kunt je elke continue functie benaderen, en met twee lagen zelfs elke discontinue functie (Cybenko).
De output van een netwerk hangt af van de parameters, de gewichten.
Bij *te veel* parameters is er gevaar voor overfitting: het netwerk generaliseert dan niet goed.

Perceptron

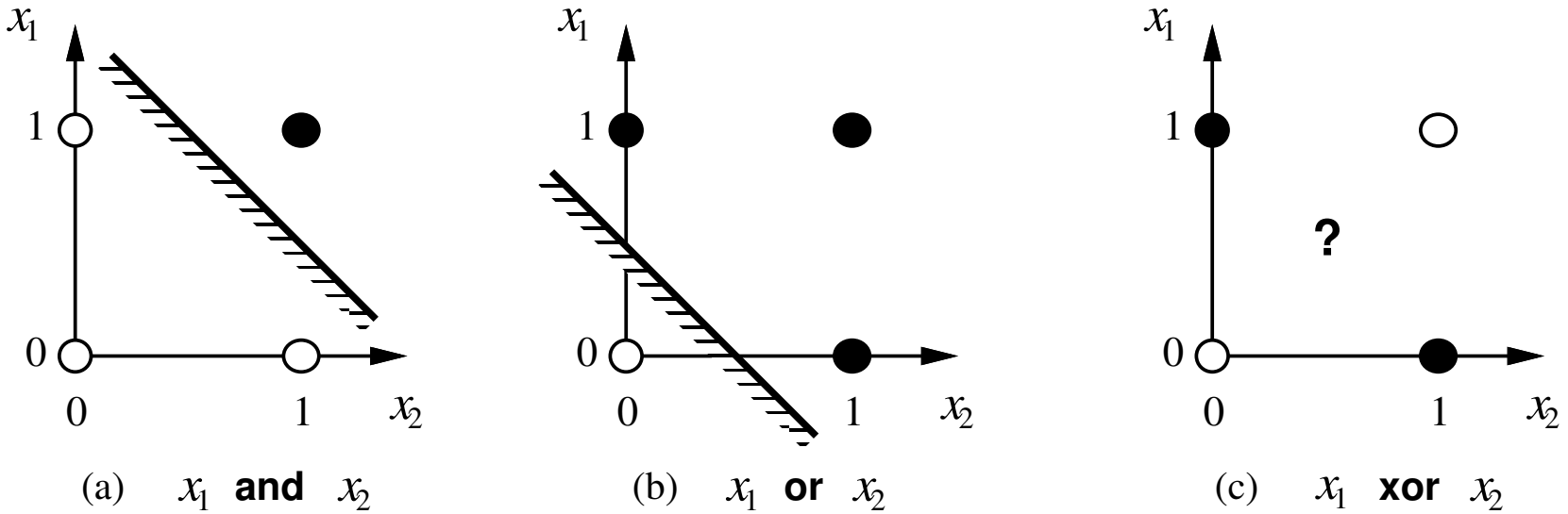
In een perceptron (geen verborgen units!) zijn de uitvoerknopen onafhankelijk:



Voor een enkele output-unit geldt dat de uitvoer 1 is indien $\sum_j W_j x_j = W_0 x_0 + W_1 x_1 + \ldots + W_n x_n \geq 0$ en anders 0. Hierbij is $(x_1, \ldots, x_n)$ de invoer en $x_0 = -1$ de bias-knoop.

Wat kan een perceptron?

De majority functie (uitvoer 1 $\Leftrightarrow$ meer dan de helft van de n inputs is 1) kan eenvoudig gemaakt worden: kies $W_0 = n/2$ en $W_j = 1$ voor $j = 1, 2, \ldots, n$.
De vergelijking $-W_0 + W_1 x_1 + \ldots + W_n x_n \geq 0$ laat zien dat je precies Booleaanse functies kunt maken die lineair te scheiden zijn, de `XOR`-functie dus niet:



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Hoe leert een perceptron?

Gegeven genoeg trainings-voorbeelden, kan een perceptron elke Booleaanse lineair te scheiden functie leren. Een (hyper)vlak scheidt positieve en negatieve voorbeelden.
Rosenblatt's algoritme uit 1957/60 werkt als volgt:

$$W_j \longleftarrow W_j + \alpha \cdot x_j \cdot \mathrm{Error}$$

met $\mathrm{Error} = \text{correcte uitvoer} - \text{net-uitvoer}$ en $\alpha > 0$ de leersnelheid (= learning rate). De correcte uitvoer heet wel de target, het doel.
Als $\mathrm{Error} = 1$ en $x_j = 1$, wordt W_j ietsje opgehoogd, in de hoop dat de net-uitvoer hoger wordt.

Perceptron — voorbeeld

We willen een perceptron x_1 AND x_2 leren, met $\alpha = 0.1$:

	x0	x1	x2	W_0	W_1	W_2	uitvoer	target	error
1	-1	1	1	-0.160	-0.606	-0.217	0.000	1.000	1.000
2	-1	1	0	-0.260	-0.506	-0.117	0.000	0.000	0.000
3	-1	0	1	-0.260	-0.506	-0.117	1.000	0.000	-1.000
4	-1	1	0	-0.160	-0.506	-0.217	0.000	0.000	0.000
5	-1	1	0	-0.160	-0.506	-0.217	0.000	0.000	0.000
6	-1	1	1	-0.160	-0.506	-0.217	0.000	1.000	1.000
7	-1	0	0	-0.260	-0.406	-0.117	1.000	0.000	-1.000
8	-1	0	0	-0.160	-0.406	-0.117	1.000	0.000	-1.000
...									
70	-1	1	0	0.140	0.194	0.183	1.000	0.000	-1.000
71	-1	1	1	0.240	0.094	0.183	1.000	1.000	0.000
...									

Probleem: wanneer stop je?

Perceptron — anders leren

We kunnen in plaats van de discontinue stapfunctie ook een gladde sigmoide g gebruiken in de knopen. Een vergelijkbaar leeralgoritme gaat dan als volgt.

Zij $E = \frac{1}{2}\text{Error}^2 = \frac{1}{2}\left(y - g(\sum_{j=0}^{n} W_j x_j)\right)^2$ met y de target. Met behulp van gradient descent bepalen we in welke richting de fout het snelst *stijgt*:

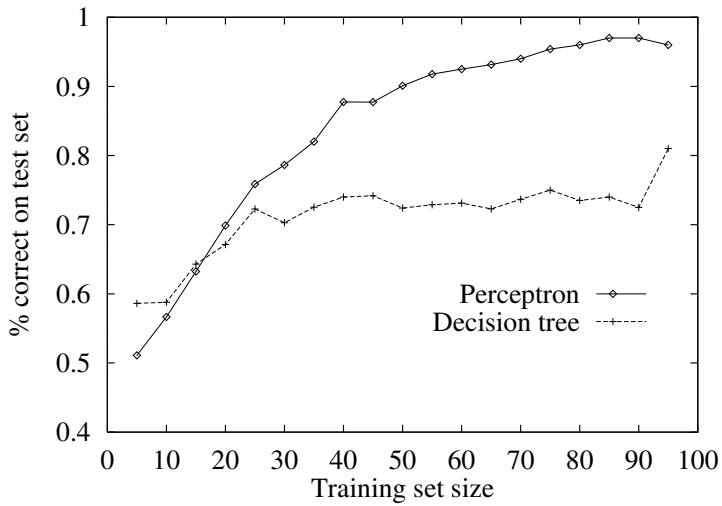
$$\frac{\partial E}{\partial W_j} = \text{Error} \cdot \frac{\partial \text{Error}}{\partial W_j} = -\text{Error} \cdot g'(\text{in}) \cdot x_j$$

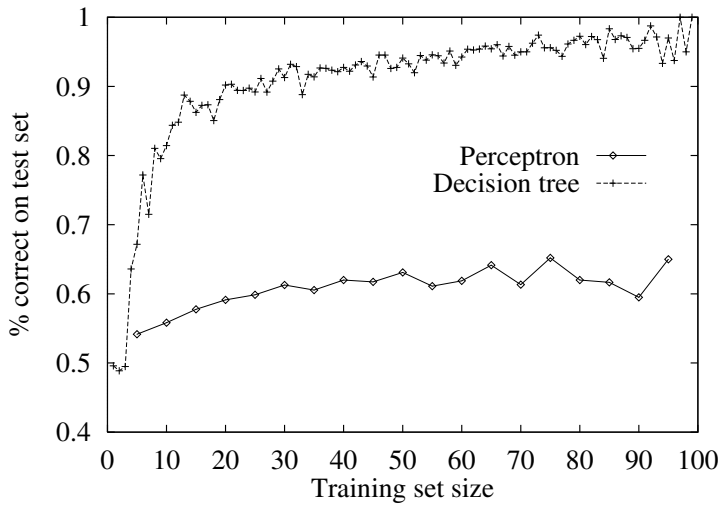
met $\text{in} = \sum_{j=0}^{n} W_j x_j$. Dus leerregel (let op de extra $-$, we willen de fout laten *dalen*):

$$W_j \longleftarrow W_j + \alpha \cdot \text{Error} \cdot g'(\text{in}) \cdot x_j \quad .$$

Perceptron — vergelijking

Voor een lineair te scheiden majority probleem (11 inputs) is het perceptron veel beter dan een decision tree (links)
Maar voor een meer complex probleem is het omgekeerd (rechts)





Codering

Er zijn allerlei keuzes om invoer en uitvoer te coderen. Je kunt bijvoorbeeld locaal coderen: stel dat een variabele 3 waarden kan hebben: Geen, Gemiddeld en Veel; dit kun je dan in één knoop coderen als 0.0, 0.5 en 1.0 respectievelijk. Maar ook met drie aparte knopen, en dan als 1–0–0, 0–1–0 en 0–0–1 respectievelijk: gedistribueerd coderen. Je kunt zelfs bij de foutmaat ervoor zorgen dat je waarden als 1–1–0.9 niet fijn vindt, en daar van weg trainen ("softmax").

Leerproces: training

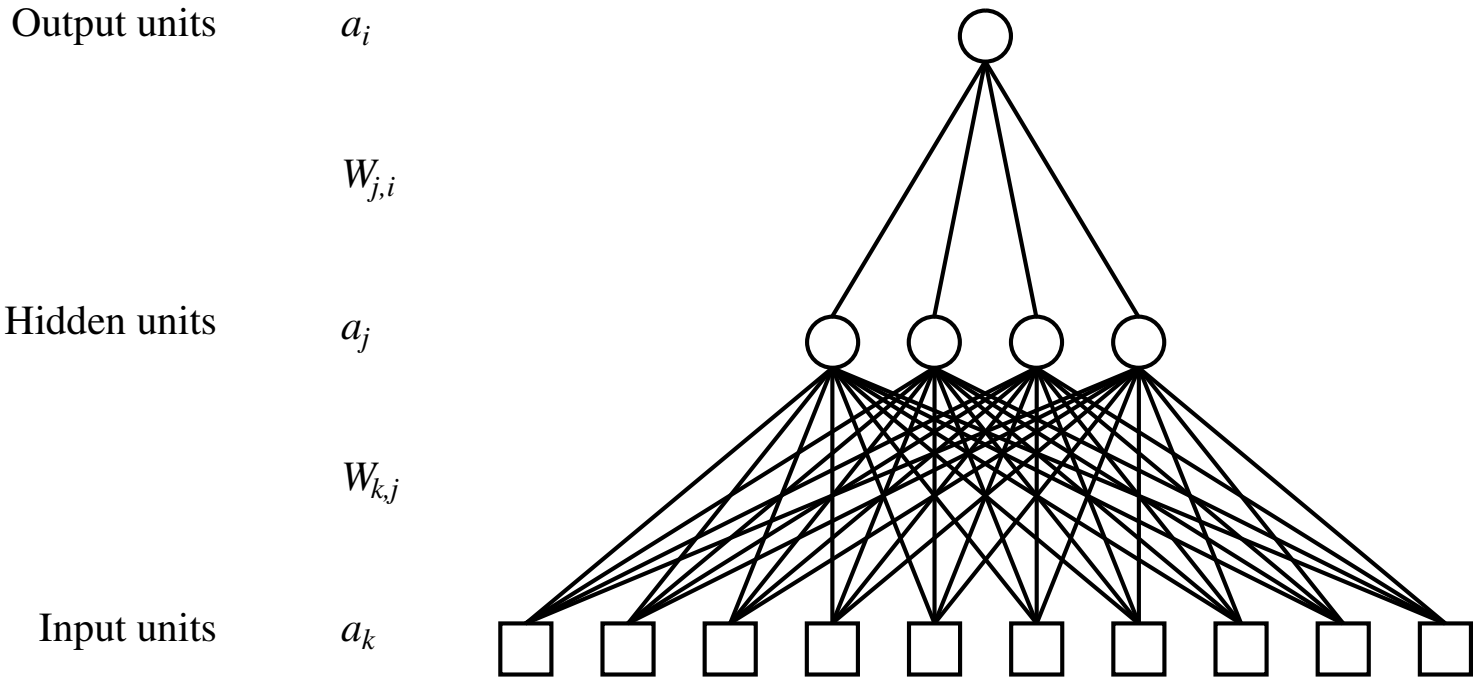
Hoe verloopt nu het leren bij Neurale Netwerken, het aanpassen van de gewichten, kortom: de training?
Je begint met een random initialisatie van de gewichten. Vervolgens biedt je één voor één voorbeelden aan uit een zogeheten trainingsset. Deze voorbeelden moeten in een willekeurige volgorde staan. Steeds geef je een invoer, en met behulp van de juiste uitvoer pas je volgens je trainings-algoritme de gewichten aan.
Je gaat net zolang door totdat (bijvoorbeeld) de fout op een apart gehouden validatieset niet meer daalt — of zelfs gaat stijgen (overtraining).
Je rapporteert tot slot over het gedrag op een (weer apart gehouden) testset.

Cross-validation

Bij elk leer-algoritme kun je cross-validation gebruiken om overfitting tegen te gaan.
Bij "k-fold cross-validation" (vaak $k = 5$ of $k = 10$) draai je k experimenten, waarbij je steeds een $1/k$-de deel van de data apart zet om als testset te gebruiken — en de rest als trainingsset. De testset is dus steeds een ander random gekozen deel!
Als $k = n$ met n de grootte van de dataset, heet deze techniek wel "leave-one-out". En dan heb je ook nog "ensemble leren", AdaBoost, enzovoorts. Zie verder het college Data Mining.

Meerlaags netwerk

We kijken nu naar een meerlaags neuraal netwerk. Meestal zijn alle lagen onderling *volledig* verbonden.



De notatie is ietwat dubbelzinnig: zit $W_{1,2}$ op twee plaatsen? Je kunt of knopen doornummeren of meerdere W's hanteren. We gebruiken index i voor de uitvoerlaag, j voor de verborgen laag en k voor de invoerlaag. De a_k's zijn de input(s) — wat we eerder x_k noemden.

Back-propagation 1

Het meest bekende leerschema voor Neurale Netwerken is back-propagation (1969, 198?), waarbij we — nadat een invoer een uitvoer heeft opgeleverd — de fout teruggeven (propageren) van uitvoerlaag richting invoerlaag.
Definieer Error_i als de fout in de i-de uitvoer (dat wil zeggen: i-de target minus i-de uitvoer van het netwerk). De leerregel is dan net als eerder:

$$W_{j,i} \longleftarrow W_{j,i} + \alpha \cdot a_j \cdot \Delta_i \quad \text{met} \quad \Delta_i = \mathrm{Error}_i \cdot g'(\mathrm{in}_i)$$

voor gewichten $W_{j,i}$ van knoop j in de verborgen laag naar knoop i in de uitvoerlaag.

Back-propagation 2

Voor gewicht $W_{k,j}$ van de k-de invoerknoop naar knoop j in de verborgen laag is de leerregel:

$$W_{k,j} \longleftarrow W_{k,j} + \alpha \cdot a_k \cdot \Delta_j \quad \text{met} \quad \Delta_j = g'(\text{in}_j) \sum_i W_{j,i} \Delta_i$$

Hierbij geldt:

- α is de leersnelheid
- a_k is de activatie van de k-de invoerknoop
- in_j is de gewogen invoer voor de j-de verborgen knoop
- $W_{j,i}$ is het gewicht op de verbinding tussen de j-de verborgen knoop en de i-de uitvoerknoop
- Δ_i is de i-de "uitvoer-delta", zie de vorige sheet

Back-propagation algoritme

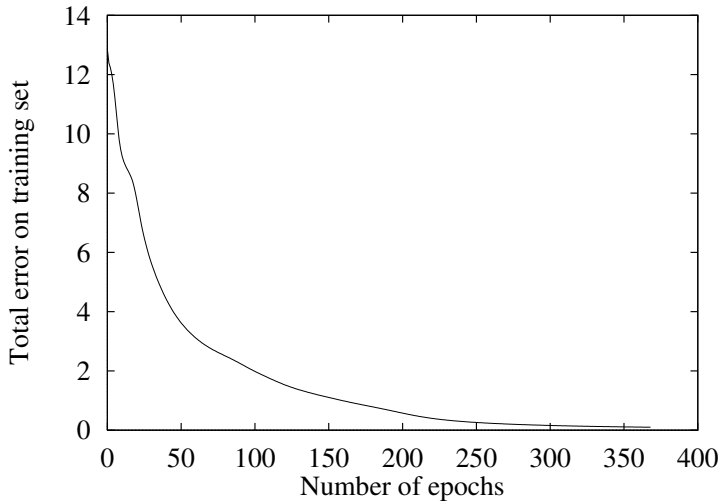
Het back-propagation-algoritme voor een netwerk met één verborgen laag gaat nu als volgt:

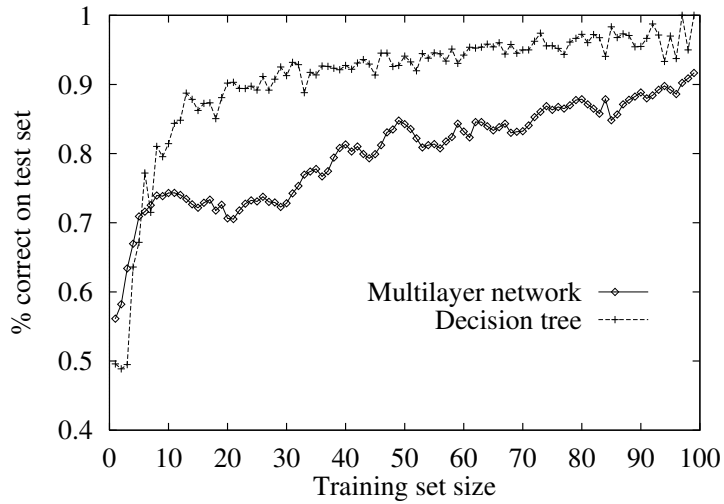
repeat
  for each e **in** trainingsset **do**
    maak de a_k's gelijk aan de x_k's
    bereken de a_j's en de a_i's (outputs)
    bereken de Δ_i's en de Δ_j's
    update de $W_{j,i}$'s en de $W_{k,j}$'s
until netwerk "geconvergeerd"

Let erop dat de gewichten goed random (?) geinitialiseerd worden. En bied de voorbeelden in random volgorde aan.

Grafieken

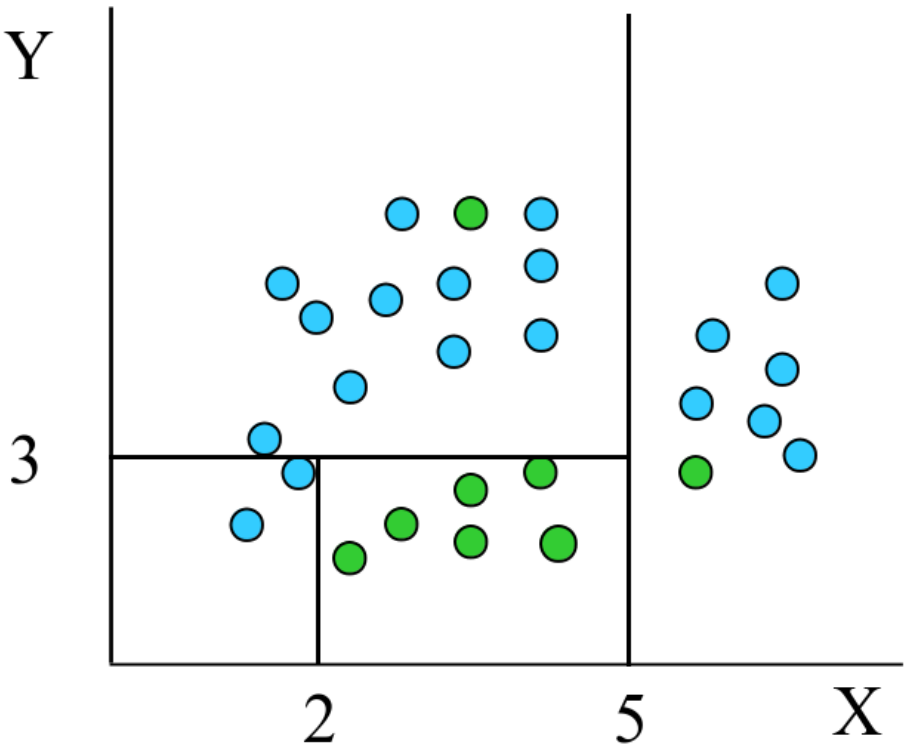
Respectievelijk een trainings-curve en een leercurve:





Een epoch is een ronde waarin alle voorbeelden uit de trainingsset één keer één voor één in random volgorde door het netwerk gegaan zijn.
Soms heb je ∞ veel voorbeelden.
Er bestaat naast deze incrementele benadering ook een batch-benadering.

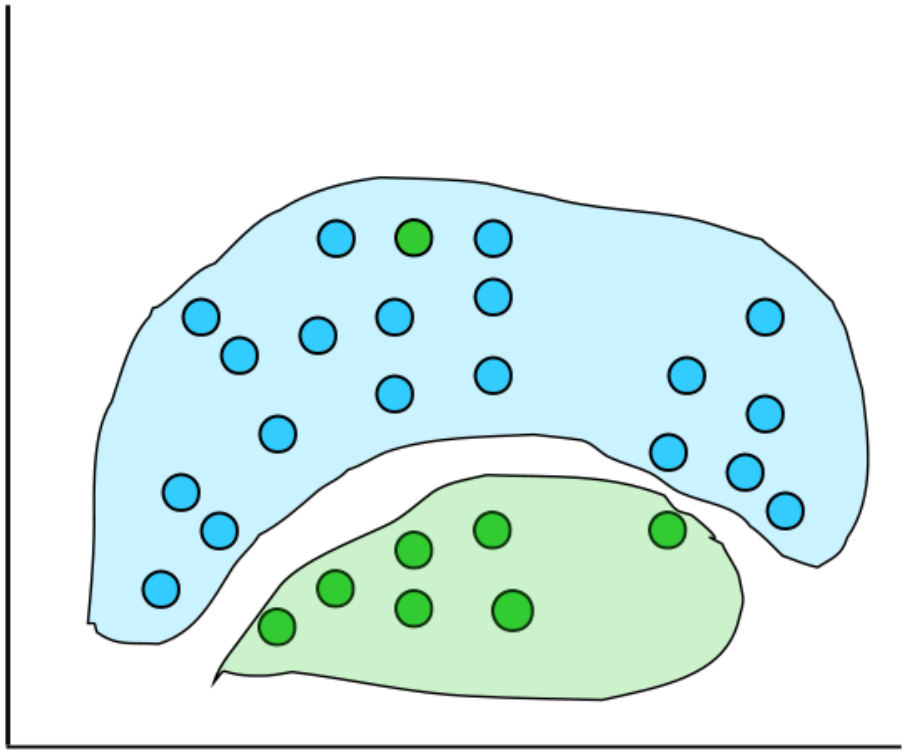
Classification: Decision trees



Decision Tree ($d = 3$)

```
if(X > 5) return BLUE;
elseif(Y > 3) return BLUE;
elseif(X > 2) return GREEN;
else return BLUE;
```

Classification: Neural networks



Neural Networks

- Perceptron
- Multi-level (meerlaags) neural network
- Convolutional neural network

Nog meer soorten

Er zijn nog vele andere soorten netwerken, zoals

- Kohonen's Self Organizing Maps (SOM's)
- Support Vector Machines (SVM's), kernel machines
- Convolutional Neural Networks
  ($\rightarrow$ Deep Learning), met activatiefunctie
  $g(x) = \max(0, x)$, de "rectifier"
- ...

Universiteit Leiden

Python

Python

- Guido van Rossem (1991)
- Directly interpreted (no compilation)
- Multi-paradigm: imperative, functional, procedural
- Cross-platform
- Version 2.7 and version 3.0

`tweakers.net/video/11095/polderpioniers-guido-van-rossum-ontwerper-van-de-programmeertaal-python.html`

Zen of Python

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- ...

Zen of Python

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts
- ...
- "Pythonic" code

PEP 20 – The Zen of Python: `https://www.python.org/dev/peps/pep-0020/`

Zen of Python

Universiteit Leiden

```
import this
"""The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

Beautiful is better than ugly.
Explicit is better than impl..
Simple is better than cσмplεx.
Cσмplεx is better than cOmp1|c@ted.
Flat is better than nested.
S p a r s e  is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
raise PythonicError("Errors should never pass silently.")
# Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
# Although that way may not be obvious at first unless you're Dutch.
Now is better than ...                    never.
Although never is often better thanrightnow.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Control flow

```
1 # this is a comment
2 def foo(x):
3     if x == 0:
4         bar()
5         baz()
6     else:
7         qux(x)
8         foo(x - 1)
```

Control flow

```
1 # this is a comment
2 def foo(x):
3     if x == 0:
4         bar()
5         baz()
6     else:
7         qux(x)
8         foo(x - 1)
```

- Indentation determines control flow
- Colon : starts indentation
- No semicolon ;
- Parentheses (and) often optional

Types

- Integers `int`
- Floats `float`
- `True` or `False` `bool`
- Null object: `None`
- `type()` function returns type

Boolean operators

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

https://docs.python.org/2/library/stdtypes.html

Comparison operators

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

https://docs.python.org/2/library/stdtypes.html

Control flow

```
1  if statement1 and statement2:
2      print("1 and 2 are True")
3  elif statement2 or statement3:
4      print("2 or 3 or both are True")
5  else:
6      print("then just do this")
7
8  while i < 5:
9      print(i)
10     i += 1 # ++ does not exist
11
12 for x in [1,2,3]:
13     print(x)
```

Lists

```
1  emptylist = []
2  alist = ['a', 'b', 'c']
3  print (alist)
4  >>> ['a', 'b', 'c', 'x']
5  alist.append(8)
6  print (alist)
7  >>> ['a', 'b', 'c', 'x', 8]
8  print (len(alist))
9  >>> 5
10 fastlist = [x**2 for x in
       range(0,5)]
```

Lists

```
1  emptylist = []
2  alist = ['a', 'b', 'c']
3  print (alist)
4  >>> ['a', 'b', 'c', 'x']
5  alist.append(8)
6  print (alist)
7  >>> ['a', 'b', 'c', 'x', 8]
8  print (len(alist))
9  >>> 5
10 fastlist = [x**2 for x in
       range(0,5)]
```

- Ordered sequences
- Indexed from `L[0]` to `len(L)-1`
- Can be joined and sliced

List operators

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n, n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x)`	index of the first occurrence of *x* in *s*
`s.count(x)`	total number of occurrences of *x* in *s*

https://docs.python.org/2/library/stdtypes.html

Functions

```
1  def printthis(wellthis):
2      print (wellthis)
3      return;
4
5  def multi(val1, val2):
6      return (val1 * val2)
7
8  def returnmeanddouble(me):
9      return me, me*2;
10
11 print (multi(val2=8, val1=4))
12 >>> 32
13 x, y = returnmeanddouble(8)
14 print (x, y)
15 >>> 8 16
```

Functions

```
1  def printthis(wellthis):
2      print (wellthis)
3      return;
4
5  def multi(val1, val2):
6      return (val1 * val2)
7
8  def returnmeanddouble(me):
9      return me, me*2;
10
11 print (multi(val2=8, val1=4))
12 >>> 32
13 x, y = returnmeanddouble(8)
14 print (x, y)
15 >>> 8 16
```

- May or may not return a value
- Optional arguments
- Optinally named arguments
- May return tuple (immutable list) of values

Sets

```
1 my_set = {1, 2, 3, 4, 8,
      102835643}
2 print(my_set)
3 someset = {0, (), False}
4 emptyset = {}
```

Sets

```
1 my_set = {1, 2, 3, 4, 8,
      102835643}
2 print(my_set)
3 someset = {0, (), False}
4 emptyset = {}
```

- Unordered collection
- Not indexed
- Apply common set operators

Set operators

Operation	Equivalent	Result
`len(s)`		number of elements in set *s* (cardinality)
`x in s`		test *x* for membership in *s*
`x not in s`		test *x* for non-membership in *s*
`s.issubset(t)`	`s <= t`	test whether every element in *s* is in *t*
`s.issuperset(t)`	`s >= t`	test whether every element in *t* is in *s*
`s.union(t)`	`s | t`	new set with elements from both *s* and *t*
`s.intersection(t)`	`s & t`	new set with elements common to *s* and *t*
`s.difference(t)`	`s - t`	new set with elements in *s* but not in *t*
`s.symmetric_difference(t)`	`s ^ t`	new set with elements in either *s* or *t* but not both
`s.copy()`		new set with a shallow copy of *s*

`https://docs.python.org/2/library/stdtypes.html`

Dictionaries

```
1 stuff = {'name':'Frank', 'age':30, 'height': 6*30+14}
2 print (stuff['name'])
3 >>> Frank
4 print (stuff['age'], stuff['height'])
5 >>> 30 194
6 stuff['city'] = "Leiden"
7 print stuff['city']
8 >>> Leiden
```

Dictionaries

```
1 stuff = {'name':'Frank', 'age':30, 'height': 6*30+14}
2 print (stuff['name'])
3 >>> Frank
4 print (stuff['age'], stuff['height'])
5 >>> 30 194
6 stuff['city'] = "Leiden"
7 print stuff['city']
8 >>> Leiden
```

- Unordered indexed collection
- Freely combine datatypes
- Hashing, so 'fast'

Modules/packages

- File `fibo.py` with Fibonacci functions `fib1()` and `fib2()`
- `import fibo`
  Call using `fibo.fib1(...)`
- `from fibo import fib, fib2`
  Call using `fib1(...)`
- `from fibo import *`
  Call using `fib1(...)`
- Standard modules: `str`, `sys`, etc.

Console or script?

- Execute interactively from `python` console
- Execute one script using `python filename.py` command
- Use Jupyter notebooks
- Spyder

. . . notebook

jupyter notebook Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Help Python 3

Markdown Cell Toolbar: None

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

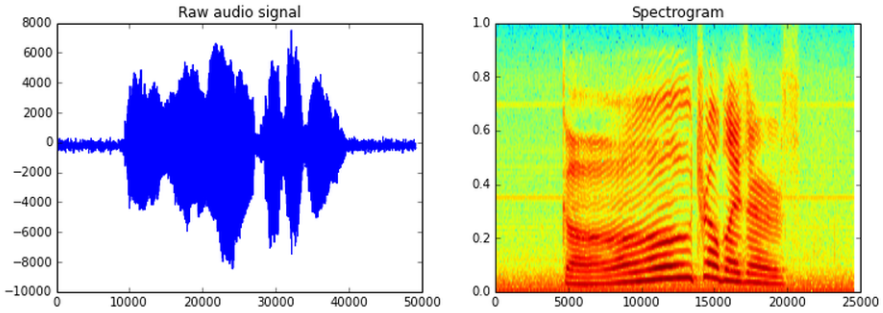
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

In [1]:
```
from scipy.io import wavfile
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin `specgram` routine:

In [2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.plot(x); ax1.set_title('Raw audio signal')
ax2.specgram(x); ax2.set_title('Spectrogram');
```



Universiteit Leiden

Pandas

Pandas

- Python package
- Good for tabular data
- Based on DataFrames
- Extensions for statistics
- Extensions for visualization

`http://pandas.pydata.org`

Pandas essentials

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 df = pd.Function()
```

Loading and saving data

```python
import pandas as pd

# tab separated
df = pd.read_csv('filename.csv', sep='\t')

# no index column
df2 = pd.read_csv('filename.csv', index_col=None)

# limit number of rows
df2 = pd.read_csv('filename.csv', nrows=20)
df.to_csv('out.csv')
```

Viewing data

```
import pandas as pd

df = pd.read_csv('filename.csv', sep='\t')
df.head(10)
df.tail(30)
df.describe
df.shape
df.columns
df.values
```

Selecting data

```python
import pandas as pd

df = pd.read_csv('filename.csv', sep='\t')

# get one column
df['columnX']

# get slice of data
df.iloc[3:5,0:2] # rows,columns

# select based on value
df[df.columnX > 0]

# get copy of data where column value is in list
df2 = df[df['columnX'].isin(['something','else'])]
```

Data types and Missing data

```python
import pandas as pd

df = pd.read_csv('filename.csv', sep='\t')

# drop any row with missing data
df.dropna(how='any')

# fill any missing value with integer 5
df.fillna(value=5)

# interpret column as certain type
df["col1"] = df["col1"].astype("float")
```

http://pandas.pydata.org/pandas-docs/stable/missing_data.html

Adding stuff to data

Universiteit Leiden

```
# add a row
s = df.iloc[2] # get row 2
df.append(s, ignore_index=True)

# add multiple columns
df1 = pd.read_csv(sys.argv[1],sep=';')
df2 = pd.read_csv(sys.argv[2],sep="\t")
df3 = pd.merge(df1,df2,left_on=["idcol1"],right_on= [
    "idcol2"],how="left")
```

Grouping data

```
1 import pandas as pd
2
3 df = pd.read_csv('filename.csv', sep='\t')
4
5 # group the data by somefield, sum numvalue into
      mysum
6 df['mysum'] = df.groupby('somefield')['numvalue'].
      transform('sum')
7
8 # remove duplicate rows
9 df = df.drop_duplicates(cols = 'somefield')
```

Universiteit Leiden

Matplotlib

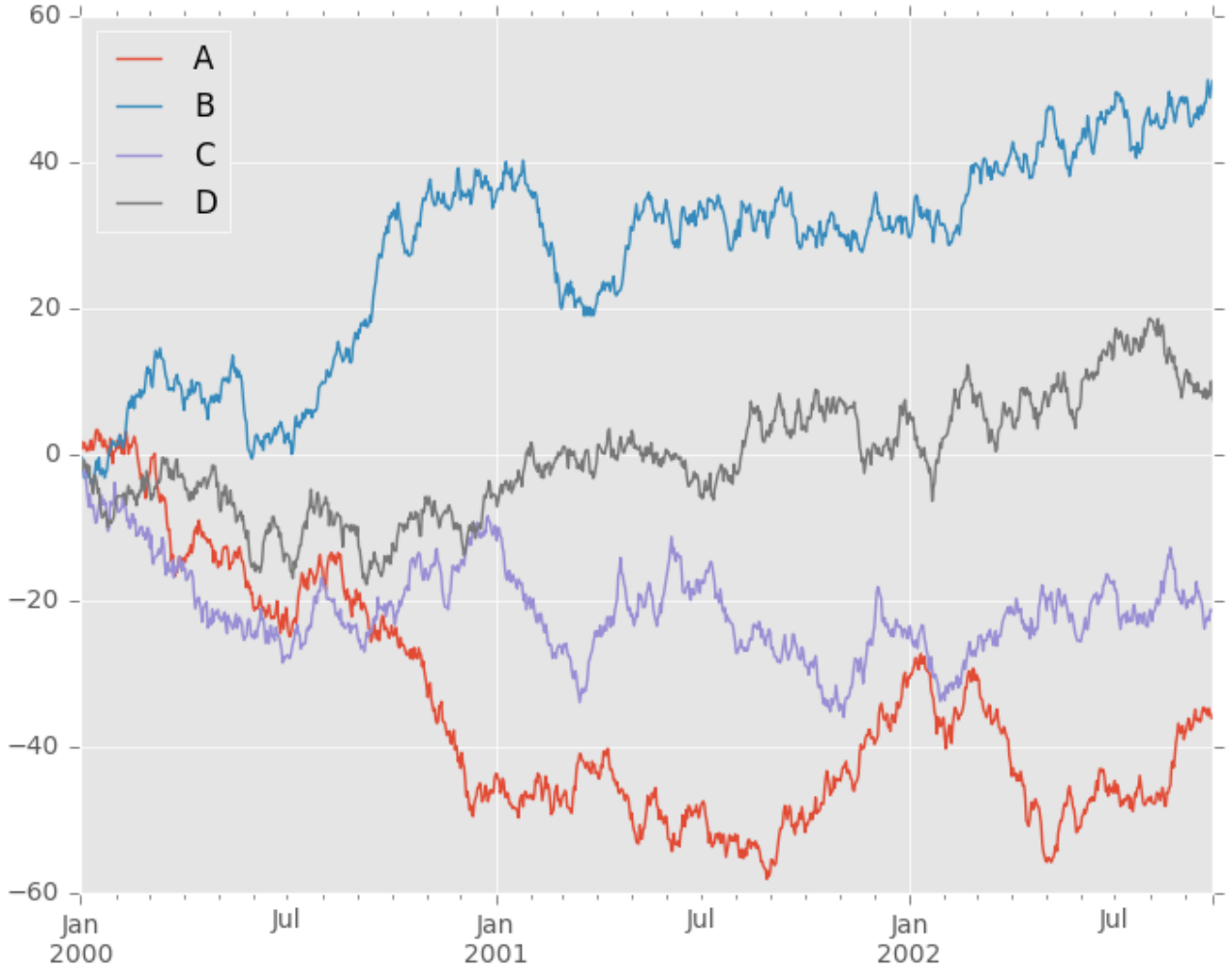
Matplotlib

- Library for plotting
- 2D plotting
- Within Python: PYPLOT
- Large collection of example code snippets

Matplotlib

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3
4  df = pd.read_csv('filename.csv', sep='\t')
5
6  # plot all columns against index to window
7  df.plot(kind='line')
8  df.plot.bar() # alternative
9
10 # plot two columns to window
11 df.plot(x='colA', y='colB')
12
13 # save last plot to file
14 plt.savefig('resultfile.pdf', dpi=600)
```

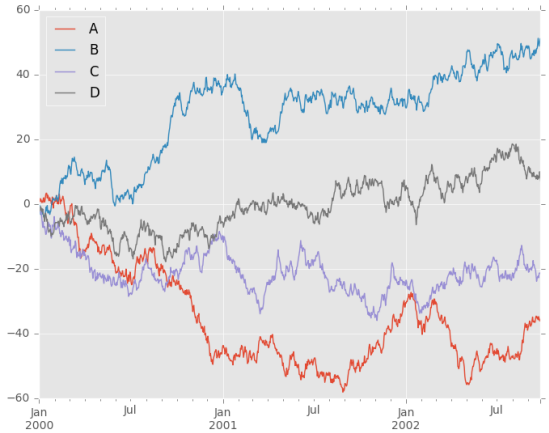
Matplotlib 'line'

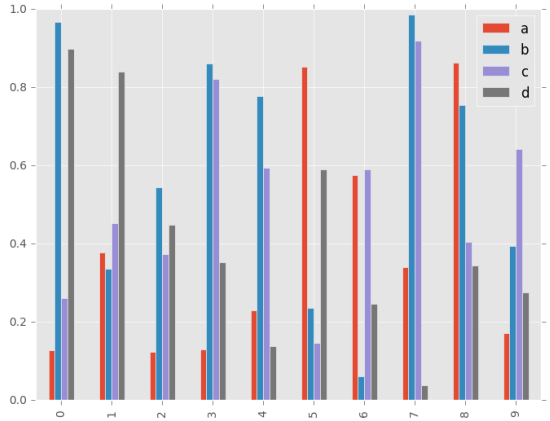


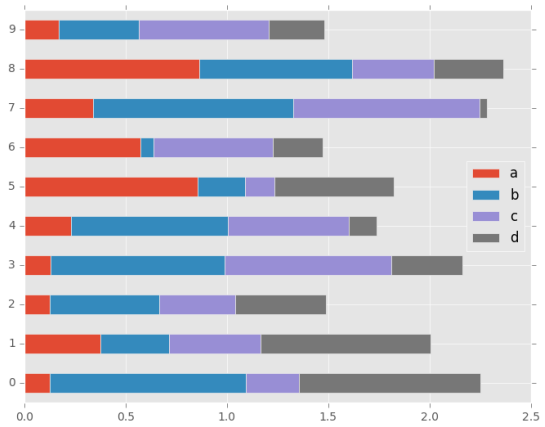
Plot types

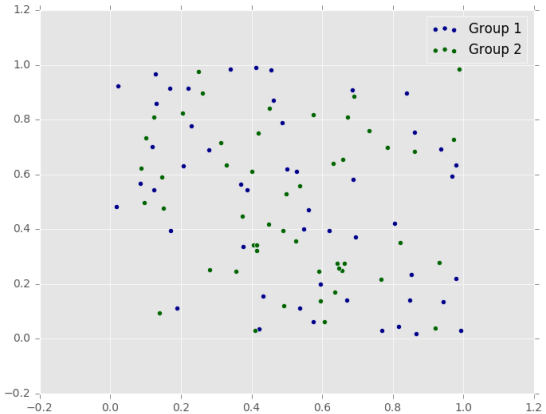
- 'bar' or 'barh' for bar plots
- 'hist' for histogram
- 'box' for boxplot
- 'kde' or 'density' for density plots
- 'area' for area plots
- 'scatter' for scatter plots
- 'hexbin' for hexagonal bin plots
- 'pie' for pie plots

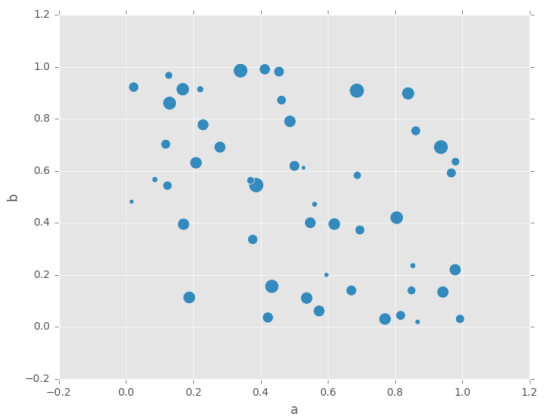
Matplotlib types

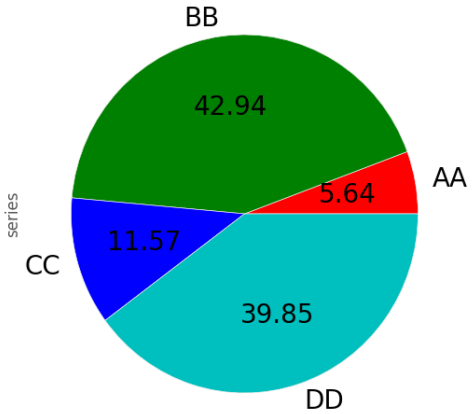








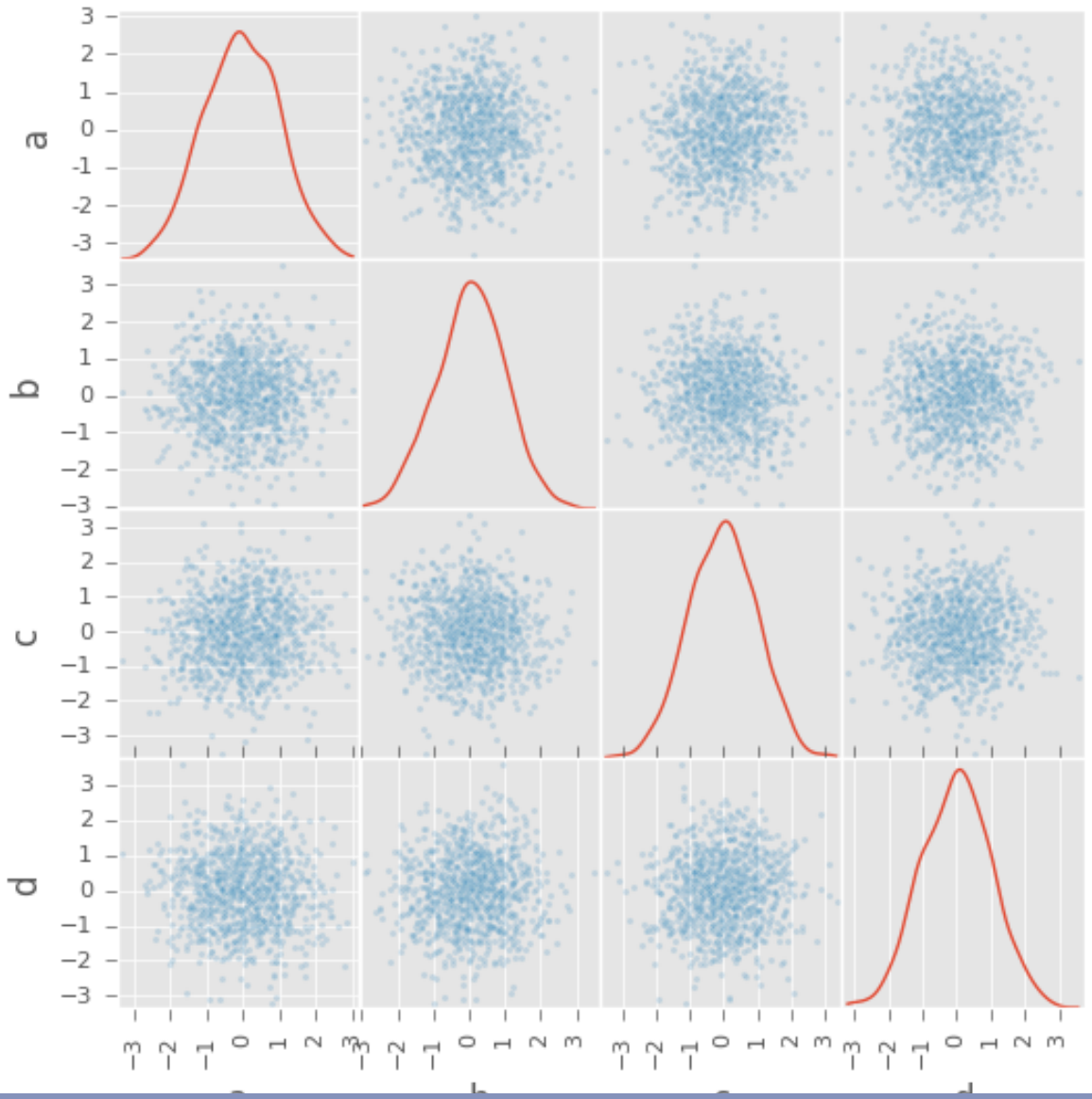




Scatter Matrix

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  from pandas.tools.plotting import scatter_matrix
4
5  # assume numeric input
6  df = pd.read_csv('filename.csv', sep='\t')
7
8  # plot all columns against index to window
9  scatter_matrix(df, alpha=0.2, figsize=(6, 6),
       diagonal='kde')
10
11 # save last plot to file
12 plt.savefig('resultfile.pdf', dpi=600)
```

Scatter Matrix



Control everything

- Axis scale
- Secondary axis
- Axis values and labels
- Custom legend
- Colors and gradients
- ...

Control everything

- Axis scale
- Secondary axis
- Axis values and labels
- Custom legend
- Colors and gradients
- ...
- `http://pandas.pydata.org/pandas-docs/stable/visualization.html`

Universiteit Leiden

Scikit-learn

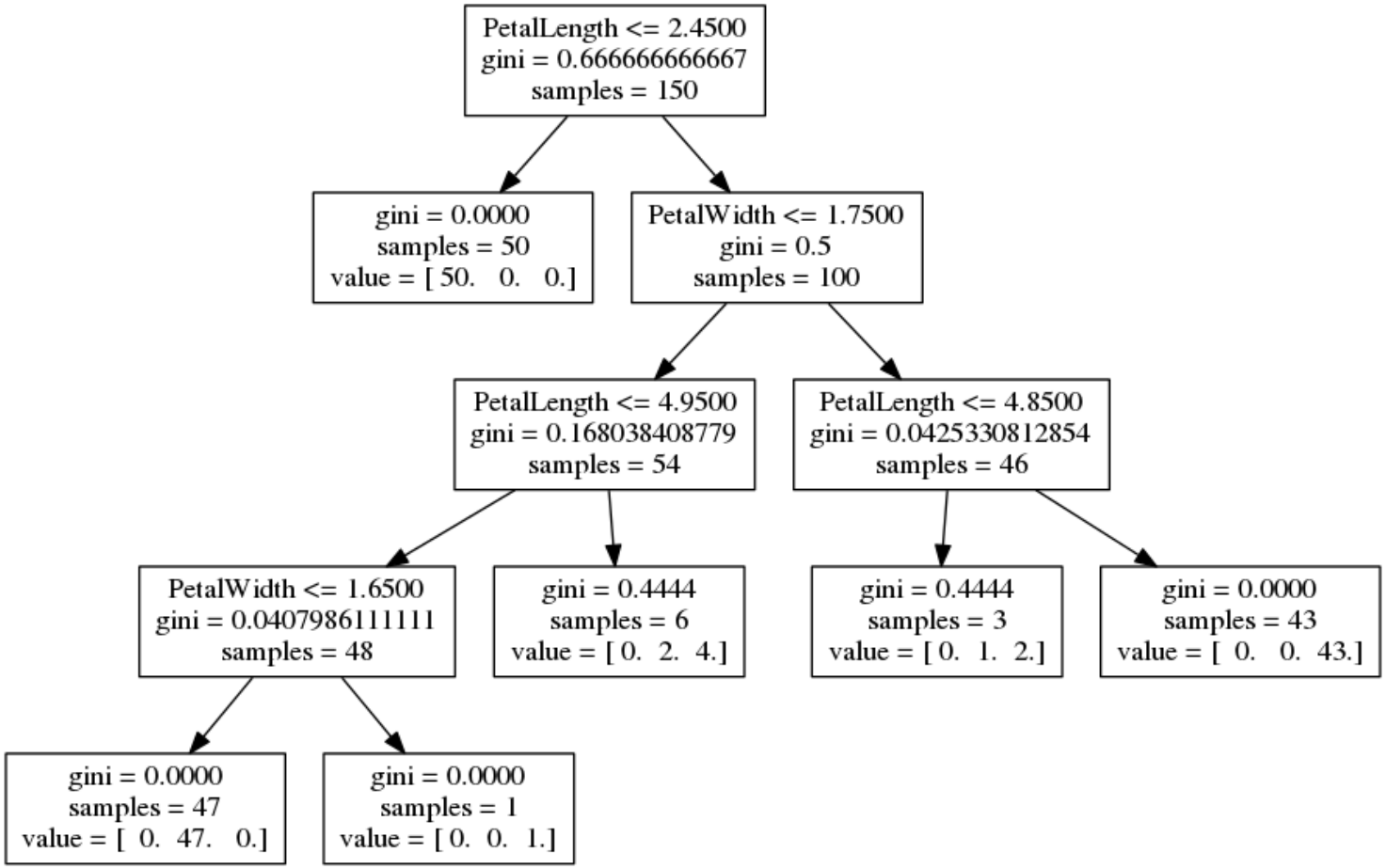
Scikit-learn

- Machine learning in Python
- Good integration with Matplotlib
- Run a learner in one line of code
- Large collection of example code snippets

Scikit-learn

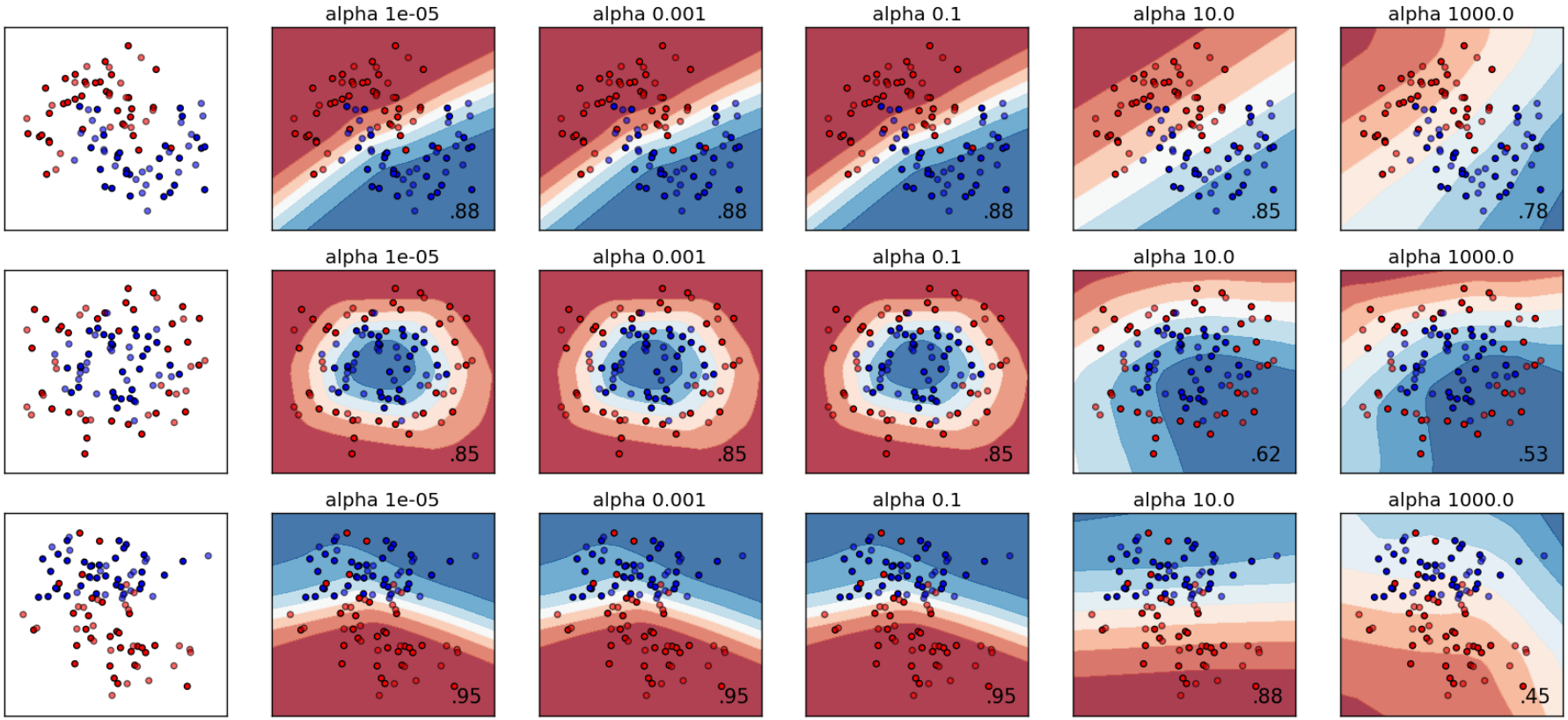
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.tree import DecisionTreeClassifier

# assume proper input
df = pd.read_csv('filename.csv', sep='\t')

# prepare input
numfeatures = 8
features = list(df.columns[:numfeatures])
X = df[features]
y = df["target"]

# fit the model to the data
dt = DecisionTreeClassifier(min_samples_split=20,
    random_state=99)
dt.fit(X, y)
```

Decision tree



http://chrisstrelioff.ws/sandbox/2015/06/08/decision_trees_in_python_with_scikit_learn_and_pandas.html

Neural networks α



Data mining disclaimers

- Data mining may find regularities from history, but history is inherently not the same as the future
- "Patterns are not people"
- Association (correlation) does not dictate trend nor causality
- Some abnormal data (outlier) could be caused by human error
- Interpretation is crucial
- The P-word is screaming for attention ...

Privacy


MARCH 21, 2011
A tale of two Libyas
Plus: Why the U.S. can't sit on the sidelines BY FAREED ZAKARIA
The GOP's misinformation campaign
BY JOE KLEIN
Could your baby be depressed?
THE CULTURE
Word up: A dictionary of slang
TIME
YOUR DATA
FOR SALE
Everything about you is being tracked—get over it
BY JOEL STEIN
Owns a laptop
Household income: $100,000+
Age: 38-39
Likes: fashion
Major life-insurance holder
Likes: online news
Likes: Asian cuisi
Dislikes: cars
Likes: green living
Favorite celebrities: Pe
ZIP code: 1070
Wi-fi warrior
Likes: business & financ
Sister is a la
Job: medical professional
Likes: parenting
Spent $180 on intimate app. & undergarments on Oct. 10, 2010
Male
Mother: Rosalind Burd
Likes: hiking
Household income: $150,000-$175,000
Owns a smart phone
Married
Dislikes: autos & vehicles
Likes: retail
Likes: music
Works at company with 5,000+ employees
Likes: newspapers
Likes: movies
Magazine subscriber
Smart-phone user
No landline
Purchased house in month of November
Has used cocaine
had LASIK surgery
Likes: restaurants
Likes: online shopping
& actresses
Likes: cooking & recipes
Lives in New York City
Dislikes: home & garden
Politically active
House value: $1M-$1.5M
Religion: Jewish
hockey
Owns an RV
Dislikes: pets & animals
Age: 36-45
High net worth
No kids
What data-mining companies think they know about Joel Stein
www.time.com
© POST TYPOGRAPHY — PLEASE CREDIT AND LINK TO → POSTTYPOGRAPHY.COM

Solutions for privacy issues

- Give slightly randomized outputs
- Replace sensitive data with some anonymous ID or hash
- Multi-party hashing
- Multi-party computation
- Roberto J. Bayardo and Ramakrishnan Srikant, Technological Solutions for Protecting Privacy, *IEEE Computer* 36(9): 115–118, 2003.

Lab session March 9

Universiteit Leiden

- Set up your environment, see instructions in `/vol/share/software/datascience/2016-README`
- Get familiar (again) with Python: `https://github.com/jrjohansson/scientific-python-lectures`
- Do the tutorial 10 minutes to Pandas: `http://pandas.pydata.org/pandas-docs/stable/10min.html`
- Replicate metrics from Assignment 1 using this toolstack
- Learn Matplotlib, for example via Lecture 4 of `https://github.com/jrjohansson/scientific-python-lectures` or `http://pandas.pydata.org/pandas-docs/stable/visualization.html`
- Replicate visualizations from Assignment 1 using this toolstack

Credits

Slides on neural networks based on slides from
"Kunstmatige Intelligentie" by Walter Kosters
(`https://liacs.leidenuniv.nl/~kosterswa/AI/neuraal.pdf`)