Neural Networks (2/2)

Lecture 8

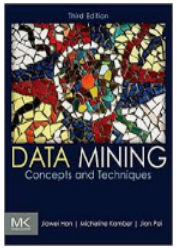
IDS-L8

Chair of Process and Data Science

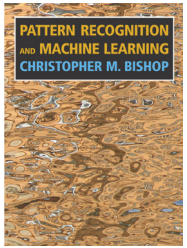
RWTH AACHEN UNIVERSITY

Outline of Today's Lecture

- **Network Training**
  Neural Network Parameters
- **Backpropagation algorithm**
- **Neural networks in practice**
- **Neural networks advantages and disadvantages**
- **A short detour: Naïve Bayesian Classification**



Based on chapter 9 of "data mining concepts and techniques" by Jiawei Han



Chapter 5 of "pattern recognition and machine learning" by Christopher M. Bishop

© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

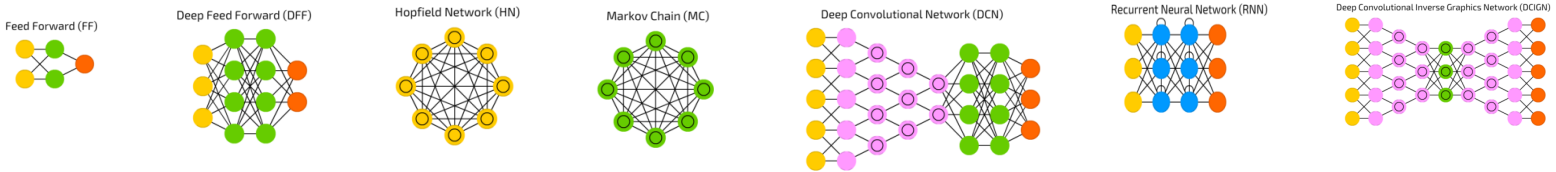
Network Training

© PADS (use only with permission & acknowledgements)

Define Network Parameters

Before training:

- We should decide on the network topology by specifying:
  - The number of units in the input
  - The number of hidden layers (one or more)
    - The number of neurons in each hidden layer
  - The number of neurons in the output layer



© PADS (use only with permission & acknowledgements)

Define Network Parameters

- So we have to answer the following questions before training:
  - How are the inputs selected?
  - How many neurons in the output layer?
  - How are the weights initialized?
  - How many hidden layers and how many neurons?
  - How many examples in the training set?

© PADS (use only with permission & acknowledgements)

Network Parameters
Inputs

- Typically, input values are normalized to fall between 0.0 and 1.0.
- Nominal or discrete-valued attributes may be encoded such that there is one input unit per domain value, e.g. using “One Hot Encoding”.
  - For example, if an attribute X has three possible or known values $\{x_0, x_1, x_2\}$, then each of these values may be encoded by a separate input.
    - If X = x_0, then $x_0 = 1$, $x_1 = 0$, $x_2 = 0$.
    - If X = x_1, then $x_0 = 0$, $x_1 = 1$, $x_2 = 0$.
    - If X = x_2, then $x_0 = 0$, $x_1 = 0$, $x_2 = 1$.

© PADS (use only with permission & acknowledgements)

Network Parameters
Output layers

- Neural networks can be used for both:
  - Classification (to predict the class label of a given input).
  - Numeric prediction (to predict a continuous-valued output).
- For classification, one output neuron may be used to represent two classes (0 or 1)
  - For more than two classes, one output neuron per class.

© PADS (use only with permission & acknowledgements)

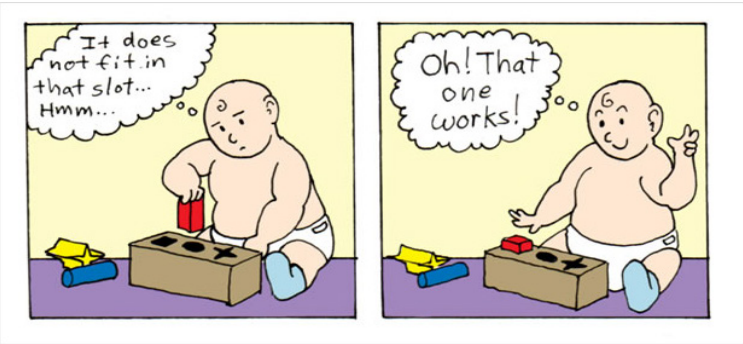
Network Parameters
Weights

- The weights in the neural networks considered here are continuously updated to reduce prediction error.
- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.

© PADS (use only with permission & acknowledgements)

Network Parameters
Hidden layers

- **No clear rules as to the “best” number of hidden layers and neurons**
  - **Network design is a trial-and-error process**



  - **May affect the accuracy of the resulting trained network**

© PADS (use only with permission & acknowledgements)

Network Parameters
Training Data

- It is very important that enough training examples are available to train the network.
- But there is not robust way to indicate the minimal size of the training set.
- A generally accepted guideline is:
  - Rule of thumb:
    - The number of training examples should be at least five to ten times the number of weights of the network.

Kieron Messer, Josef Kittler: Choosing an Optimal Neural Network Size to aid a Search Through a Large Image Database. BMVC 1998: 1-10http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.4785&rep=rep1&type=pdf

© PADS (use only with permission & acknowledgements)

Network Training Basics

- **The most basic method of training a neural network is:**

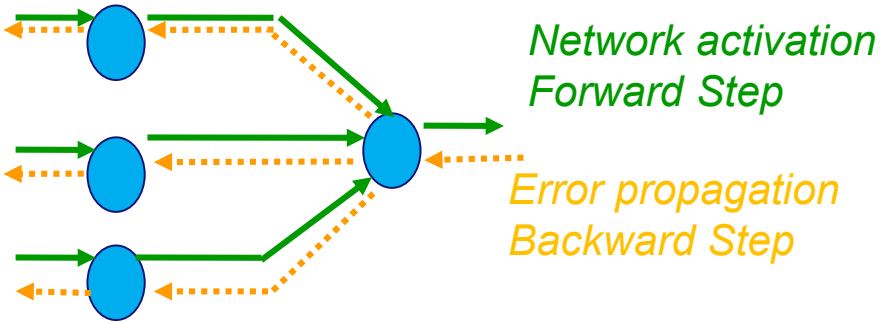
  Trial and Error

  - **If the network isn't behaving the way it should, change the weighting of a random link by a random amount.**
  - **If the accuracy of the network declines:**
    - **Undo the change and make a different one**
  - **It takes time, but the trial and error method does produce results!**

© PADS (use only with permission & acknowledgements)

Back Propagation
Concept

- **Backpropagation training algorithm:**



- **The backpropagation algorithm searches for values of weights:**
  - **that minimize the total error of the network over the set of training examples (training set).**

© PADS (use only with permission & acknowledgements)

Neural Network Training Algorithm

Backpropagation Algorithm

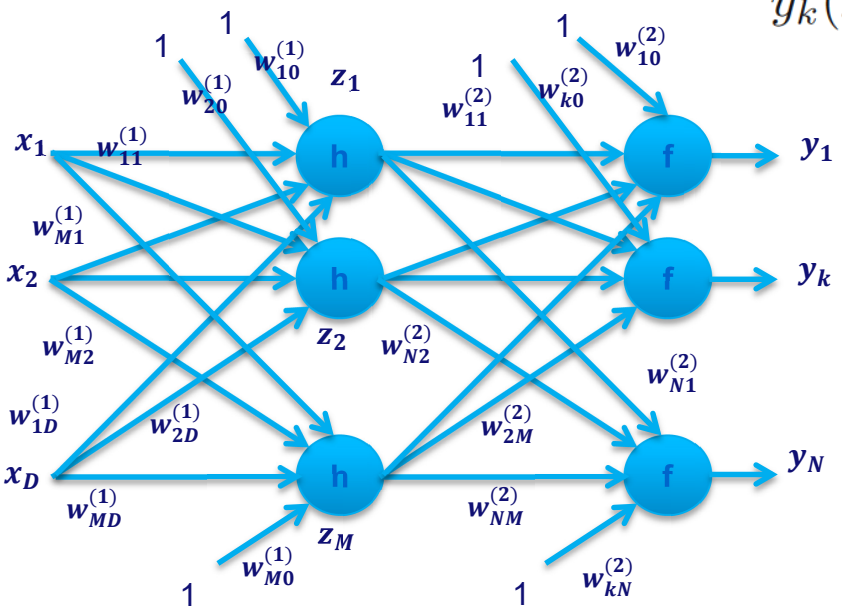
- Backpropagation consists of the repeated application of the following two passes:
  - Forward pass:
    1. The network is activated on one example
    2. The error of neurons of the output layer is computed
  - Backward pass:
    1. The network error is used for updating the weights
    2. Starting at the output layer, the error is propagated backwards through the network, layer by layer
    3. This is done by recursively computing the local deviation of error for each layer.

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
First step-feedforward

- **Recall, a Feedforward Network Function**

Two layer network



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

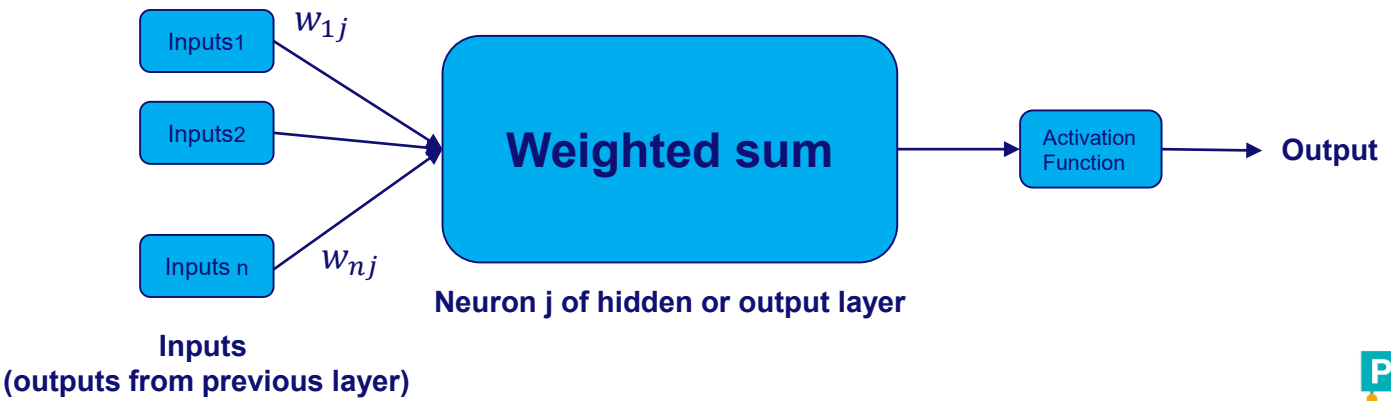
$$z_k = h\left(\sum_j w_{kj} z_j\right)$$

Presented in last lecture.

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
First step- with respect to one neuron

- The inputs to neuron j are outputs from the previous layer
- These are multiplied by their corresponding weights to form a weighted sum



© PADS (use only with permission & acknowledgements)

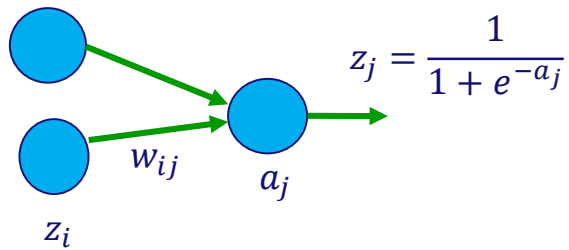
Chair of Process and Data Science

Backpropagation Algorithm
First step- with respect to one neuron view

- **Given a neuron j in a hidden layer or output layer the input to neuron j is:**

$$a_j = \sum_i w_{ij} z_i$$

- Where w_{ij} is the weight of the connection from neuron i in the previous layer to neuron j
- z_i is output of unit i from the previous layer



Note that for clarity we swapped the order of indices (compared to previous lecture/slide). Now w_{ij} is the connection from i to j.

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
First step- activation function

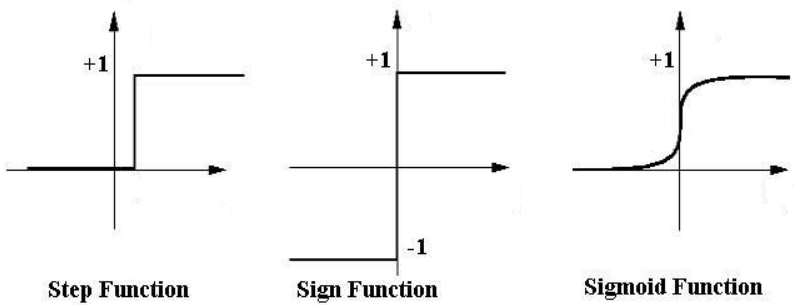
- **Each neuron in the hidden and output layers takes its input and then applies an activation function to it**
  - **The sigmoid function (logistic) is used here.**
  - **Recall that in the previous lecture we simplified this a bit.**
- **Given the input a_j to neuron j, then z_j, the output of neuron j is:**

$$z_j = \frac{1}{1 + e^{-a_j}}$$

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
First step- activation function

- There are different activation function that can be used.
- However, for backpropagation algorithm, we need a differentiable activation function:
  - such as the sigmoid function $z_j = \frac{1}{1 + e^{-a_j}}$
  - The reason is that backpropagation algorithm uses error function derivation (i.e., a variant of gradient descend used in regression).



© PADS (use only with permission & acknowledgements)

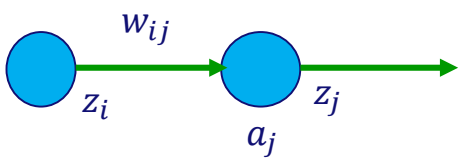
Chair of Process and Data Science

Backpropagation Algorithm
Second step- updating weights

- **Updating weights :**
  - **To reflect the propagated errors**
    - **Weights are updated by the following equations:**

$$\Delta w_{ij} = l E_j z_i$$

$$w_{ijnew} = w_{ijold} + \Delta w_{ij}$$



- *Where Δw_{ij} is the change in weight*

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- error propagation

- **Back propagate the error:**
  - **The error is propagated backward by updating the weights to reflect the error of the network's prediction.**
  - **For a neuron j in the output layer, the error E_j is computed by:**

$$E_j = z_j(1 - z_j)(t_j - z_j)$$

- *Where z_j is the actual output of neuron j*
- *t_j is the known target value of training example*
- *Note! $z_j(1 - z_j)$ is the derivative of the sigmoid function*

© PADS (use only with permission & acknowledgements)

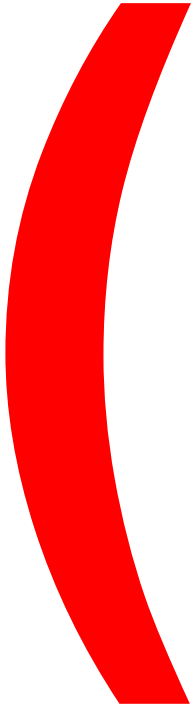
Backpropagation Algorithm
Second step- error propagation

- **Back propagate the error:**
  - **For a neuron j in the hidden layer:**
    - **The weighted sum of the errors of the neurons connected to neuron j in the next layer are considered.**

$$E_j = z_j(1 - z_j)\sum_k E_k w_{jk}$$

- *Where w_{jk} is the weight of the connection from neuron j to a neuron k in the next higher layer*
- *and E_k is the error of neuron k*

© PADS (use only with permission & acknowledgements)



© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- error propagation

- **Derivative of sigmoid function:**

Recall!

- **General differentiation rules:**

$$\begin{cases} \frac{\partial}{\partial_x}\left(\frac{f(x)}{g(x)}\right) = \frac{f'g - fg'}{g^2} \\ \frac{\partial}{\partial_x}(e^x) = e^x \end{cases}$$

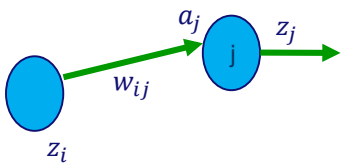
- $y = \frac{1}{1+e^{-x}} \Rightarrow y'(x) = \frac{\partial}{\partial_x}\left(\frac{1}{1+e^{-x}}\right) = \frac{0(1+e^{-x}) - 1(-e^{-x})}{(1+e^{-x})(1+e^{-x})} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) \Rightarrow$

- $sigmoid'(x) = sigmoid(x).(1 - sigmoid(x))$

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- error propagation

- But how E_j is calculated?
- Notice! Every neuron gets some inputs and computes two things:
  - 1. its output value
  - 2. the *local* gradient of its inputs with respect to its output value
- Neurons are communicating gates to each other
  - through the gradient signal
  - whether they want their outputs to increase or decrease (and how strongly)
  - so as to make the final error lower

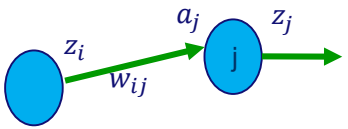


© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Backpropagation Algorithm
Second step- error propagation

- Network's error is calculated based on square error:
  - $Error = \frac{1}{2}(z_j - t_j)^2$

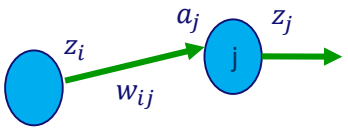


  - We should take steps to decrease Error with respect to w_{ij}: $\frac{\partial Error}{\partial w_{ij}}$
    - Note! Network's error depends on the weight w_{ij} only via summed input a_j to neuron j:
    - So we call $-\frac{\partial Error}{\partial a_j} = E_j$

The minus sign in $-\frac{\partial Error}{\partial a_j}$ was added because the error needs to be corrected in the opposite direction.

Backpropagation Algorithm
Second step- error propagation

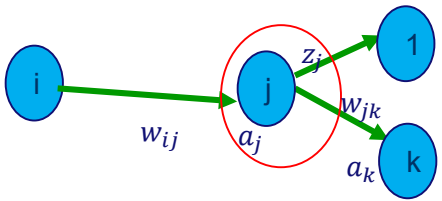
- Now for neuron j in the output layer:
  - we call $-\frac{\partial Error}{\partial a_j} = E_j$



  - $E_j = -\frac{\partial Error}{\partial a_j} = -\frac{\partial Error}{\partial z_j}\frac{\partial z_j}{\partial a_j}$
    - $where\ derivative\ of\ Error\ is\ z_j - t_j$
    - $and\ derivative\ of\ z_j\ is\ z_j(1 - z_j)$
    - $E_j = -z_j(1 - z_j)\left(z_j - t_j\right) = z_j(1 - z_j)\left(t_j - z_j\right)$
  - $-\frac{\partial Error}{\partial w_{ij}} = -\underbrace{\frac{\partial Error}{\partial a_j}}_{E_j}\frac{\partial a_j}{\partial w_{ij}}\ and\ \frac{\partial a_j}{\partial w_{ij}} = z_i \Rightarrow changes\ for\ w_{ij} = E_j z_i$

Backpropagation Algorithm
Second step- error propagation

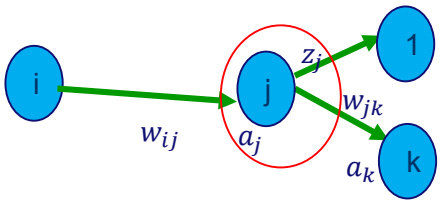
- If j is a neuron in the hidden layer:
  - Same steps: $-\frac{\partial Error}{\partial w_{ij}}$
  - The value of E for a particular hidden neuron can be obtained by propagating the E's backwards from neurons higher up in the network
    - Because we already know the values of the E's for the output neurons:
    - It follows that by recursively applying
    - using the fact that variations in a_j give rise to variations in the error function only through variations in the variables a_k



© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- error propagation

- $E_j = -\frac{\partial Error}{\partial a_j} = \sum_k -\frac{\partial Error}{\partial a_k}\frac{\partial a_k}{\partial a_j}$
- $where\ -\frac{\partial Error}{\partial a_k} = E_k$
- $and\ a_k = z_j \Rightarrow \frac{\partial a_k}{\partial a_j} = z_j(1 - z_j)$
- $\boldsymbol{E_j = z_j(1 - z_j)\sum_k E_k w_{jk}}$



© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- error propagation

- **Difficult? Details are, but general idea is not.**
  - **Just answer of the following questions:**
  - **how much does the *final error* change with respect to the *output of layer j*?**
    - **(or how much is a change in *output*)**
  - **Next, how much does the *output* change with respect to its *input*?**
    - **(or how much is a change in *sum*)**
  - **Finally, how much does the *input* change with respect to the *weights*?**
    - **(or how much is a change in *weights*)**

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- updating weights

- **What is l?** $\Delta w_{ij} = (l) E_j z_i$
  - **The learning rate**
  - **Usually a constant between 0.0 and 1.0**

- **A rule of thumb:**

$$l = \frac{1}{t}$$

- *Where t is the number of iterations through training set so far*

© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

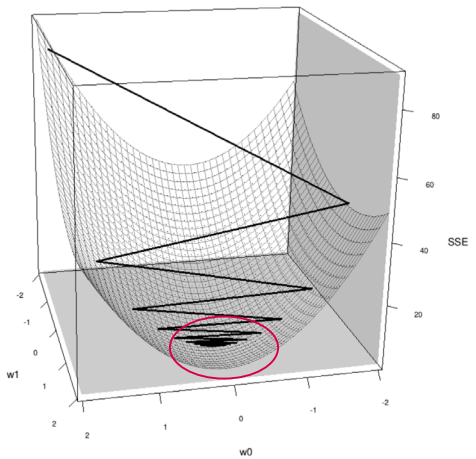
Backpropagation Algorithm
Second step- updating weights

- Why l is needed?
- Backpropagation is error based learning algorithms
- It learns using a gradient descent method:
  - Searches for a set of weights
    - that fits the training data to minimize the error between the network's result and the target value of the training example

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Second step- updating weights

- **Same idea as SVM:**
  - **Gradient decent:**
  - **Compute rate of change of error for each weight**
  - **Follow steepest rate of change**



Looking for this point (value of w_0 and w_1) in which the error is minimum

© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Backpropagation Algorithm
Second step- updating weights

- **Why l is needed?**
  1. **The learning rate helps avoid getting stuck at a local minimum in decision space**
     - **i.e., where the weights appear to converge, but are not the optimum solution**
  2. **Encourages finding the global minimum.**
  - **If l is too small: learning will occur at a very slow pace**
  - **If l is too large: oscillation between inadequate solutions may occur**

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Terminating Condition

- Note! we are updating the weights after the presentation of each training example (case updating)
- Alternatively:
  - The weights are updated after all the training example in the training set have been presented (epoch updating)
- Epoch is one iteration through the training set

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Terminating Condition

- When training should stop?
  - When all Δw_{ij} in the previous epoch are so small
    - as to be below some specified threshold
  - or the percentage of inputs misclassified in the previous epoch is below some threshold
  - or a pre specified number of epochs has expired
- In practice, several hundreds of thousands of epochs may be required before the weights will converge.

© PADS (use only with permission & acknowledgements)

Back Propagation
A Pseudo-Code Algorithm

- Randomly choose the initial weights
- While error is too large
  - For each training pattern (presented in random order)
    - Apply the inputs to the network
    - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer
    - Calculate the error at the outputs
    - Use the output error to compute error signals for pre-output layers
    - Use the error signals to compute weight adjustments
    - Apply the weight adjustments
  - Periodically evaluate the network performance

© PADS (use only with permission & acknowledgements)

Back Propagation

For each neuron (j) in the network

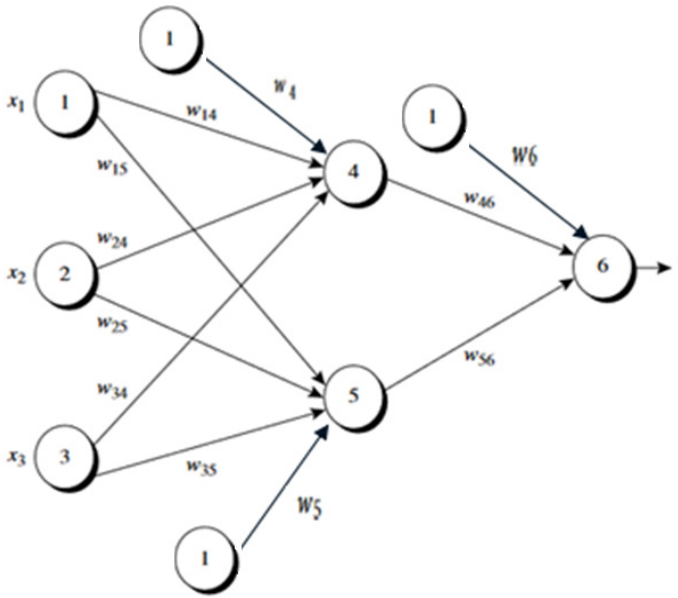
- **For each input set of X:**
  - For each input neuron j:
    - $z_j = a_j$ **(output of an input neuron is its actual input value)**
  - For each hidden or output layer neuron j:
    - $a_j = \sum_i w_{ij} z_i$ **(compute the input of neuron j with respect to the previous layer, i)**
    - $z_j = \frac{1}{1+e^{-a_j}}$ **(compute the output of each neuron j)**
  - For each neuron j in the output layer:
    - $E_j = z_j(1 - z_j)(t_j - z_j)$ **(compute the error)**
  - For each neuron j in the hidden layers:
    - $E_j = z_j(1 - z_j) \sum_k E_k w_{jk}$ **(compute the error with respect to the next higher layer, k)**
  - For each weight Δw_{ij} in network:
    - $\Delta w_{ij} = (l) E_j z_i$
    - $w_{ij} = w_{ij} + \Delta w_{ij}$

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Example

- **Sample calculation for learning by the backpropagation algorithm**
  - **A multilayer feedforward neural network:**
  - **learning rate = 0.9**
  - **First training example**
    - **X = (1,0,1) target = 1**
  - **Initial input and weights**

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	w_4	w_5	w_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1



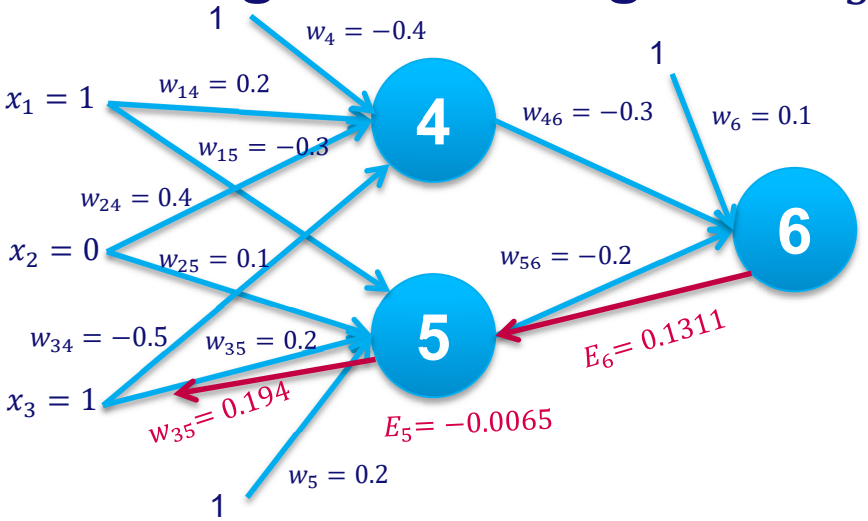
© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Backpropagation Algorithm
Example

- **Imagine the weight of w_{35} :**

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	w_4	w_5	w_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1



Feedforward

$input\ of\ neuron\ 5\colon\ a_5 = -0.3 * 1 + 0 + 1 * 0.2 + 0.2 * 1 = 0.1$

$output\ of\ neuron\ 5\colon\ z_5 = \frac{1}{1+e^{-0.1}} = 0.525$

$\boldsymbol{output\ of\ the\ network} = output\ of\ neuron\ 6\colon$

$a_6 = -0.3 * 0.332 - 0.2 * 0.525 + 0.1 = -0.105$

$z_6 = \frac{1}{1 + e^{-(-0.105)}} = 0.474$

Back Propagate

$Network's\ Error = 1 - 0.474 \Rightarrow$

$E_6 = 0.474(1 - 0.474)(1 - 0.474) = 0.1311$

$neuron\ 5 : E_5 = z_5\,(1 - z_5) * E_6 * w_{56} = 0.525 * (1 - 0.525) * 0.1311 * (-0.2) = -0.0065$

$weight\ 35 : w_{35} = w_{35} + l * E_5 * x_3 = 0.2 + 0.9 * (-0.0065) * 1 = 0.194$

© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Backpropagation Algorithm
Example

- **Network Input and Output Calculations**

Unit, j	Net Input, a_j	Output, z_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

- For each hidden or output layer neuron j:
  - $a_j = \sum_i w_{ij} z_i$ (compute the input of neuron j with respect to the previous layer, i)
  - $z_j = \frac{1}{1+e^{-a_j}}$ (compute the output of each neuron j)

- **Calculation of the E_j at Each Node**

Unit, j	E_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

- For each neuron j in the output layer:
  - $E_j = z_j(1-z_j)(t_j-z_j)$ (compute the error)
- For each neuron j in the hidden layers:
  - $E_j = z_j(1-z_j)\sum_k E_k w_{jk}$ (compute the error with respect to the next higher layer, k)

© PADS (use only with permission & acknowledgements)

Backpropagation Algorithm
Example

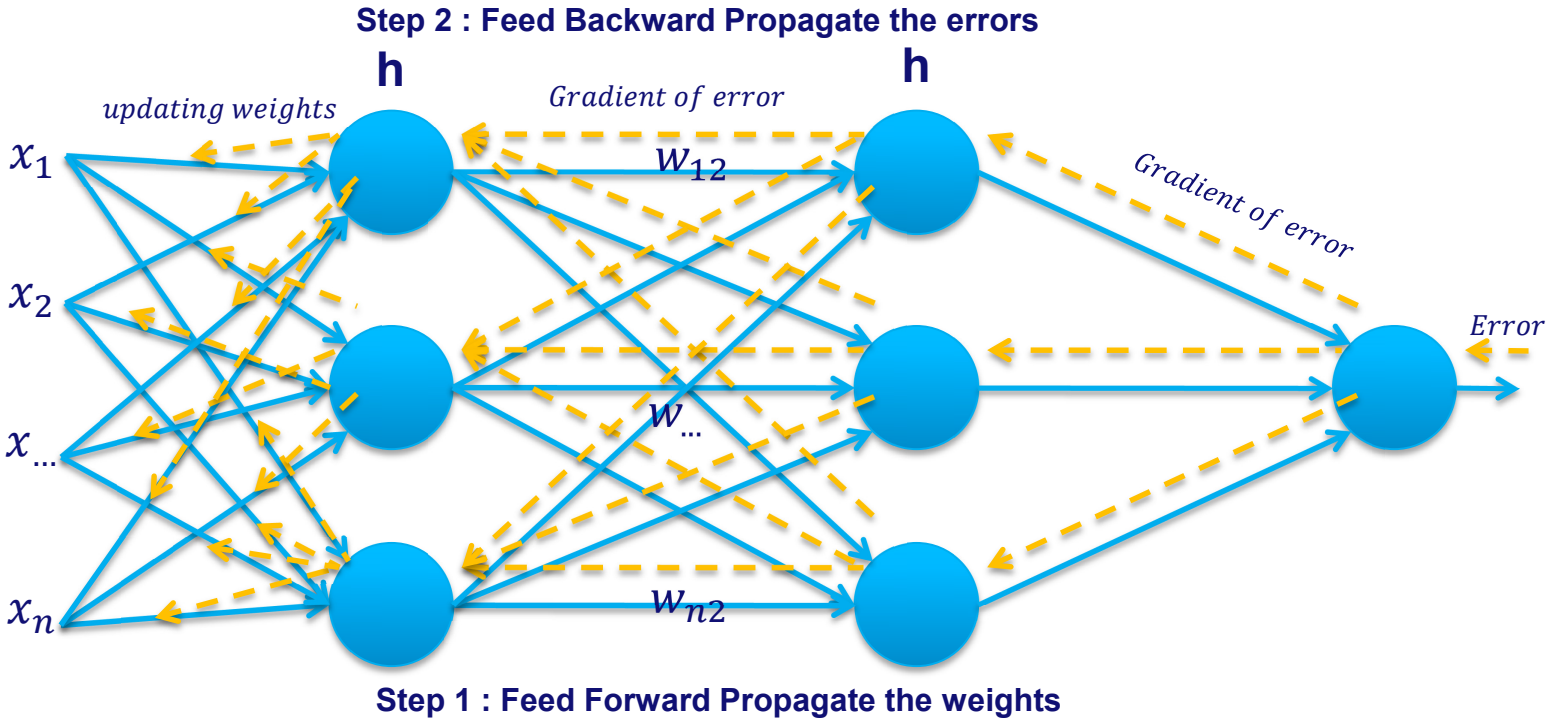
- **Calculations for Weight Updating**

Weight	New Value
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
w_6	$0.1 + (0.9)(0.1311) = 0.218$
w_5	$0.2 + (0.9)(-0.0065) = 0.194$
w_4	$-0.4 + (0.9)(-0.0087) = -0.408$

- For each weight Δw_{ij} in network:
  - $\Delta w_{ij} = (l) E_j z_i$
  - $w_{ij} = w_{ij} + \Delta w_{ij}$

© PADS (use only with permission & acknowledgements)

Backpropagation
Inside the Black Box



© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

The Learning Process

- Recall! Adaptive networks are NNs that allow the change of weights in its connections.
- For the adaptive networks, learning methods can be classified in two categories:
  - Supervised Learning
  - Unsupervised Learning

© PADS (use only with permission & acknowledgements)

Supervised Learning
In Neural Networks

- The human teacher's experience is used to tell the NN which outputs are correct and which are not.
  - The correct classifications gathered from the human teacher on a domain needs to be present.
- The network then learns from its error:
  - It changes its weights to reduce its prediction error

Chair of Process and Data Science

© PADS (use only with permission & acknowledgements)

Unsupervised Learning
In neural networks

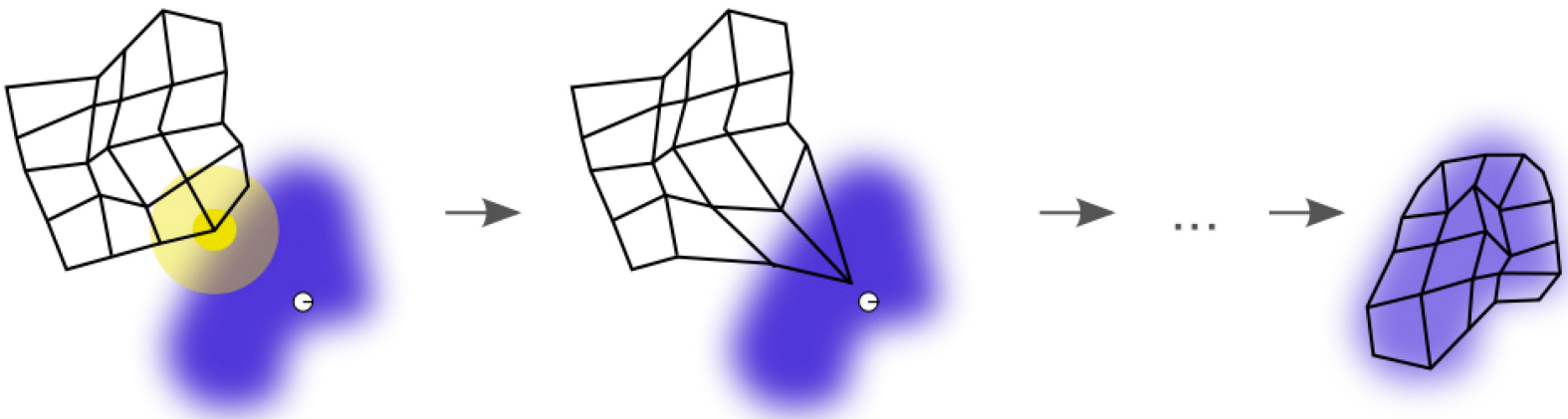
- There is no teacher
  - Based on local information
- Referred to as self-organization
  - Data presented to the network
    - Then detects their collective properties
- Construct clusters of similar patterns
- Useful in domains were instances are checked to match previous scenarios
  - For example, detecting credit card fraud

© PADS (use only with permission & acknowledgements)

Unsupervised Learning

self-organizing Maps (will be explained later in more detail)

- **One of the algorithms for unsupervised neural networks is self-organizing maps**



General View of a self-organizing map

© PADS (use only with permission & acknowledgements)

Unsupervised Learning
self-organizing Maps

- The blue blob is the distribution of the training data
  - The small white point is the current training sample drawn from that distribution.
- At first the SOM nodes are arbitrarily positioned in the data space.
  - the node nearest to the training node(yellow) is selected,
  - and is moved towards the training datum,
  - as (to a lesser extent) are its neighbors on the grid.
- After many iterations the grid tends to approximate the data distribution.

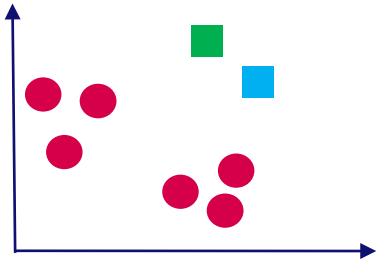
© PADS (use only with permission & acknowledgements)

Unsupervised Learning
self-organizing Maps

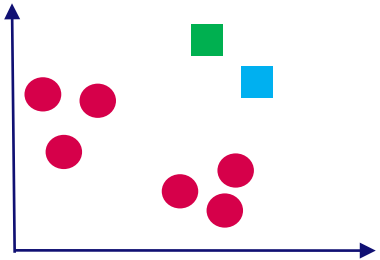
- A simple overview of SOM algorithm:
  - Select random input
  - Compute winner neuron
  - Update neurons
  - Repeat for all input data
  - Classify input data
- Inputs are connected to the neuros with weights
- The neurons are related to each other

© PADS (use only with permission & acknowledgements)

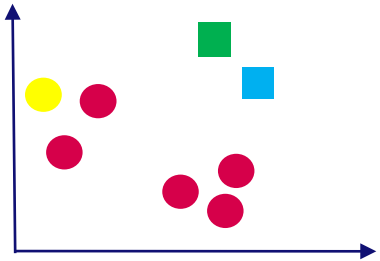
Unsupervised Learning
self-organizing Maps - example



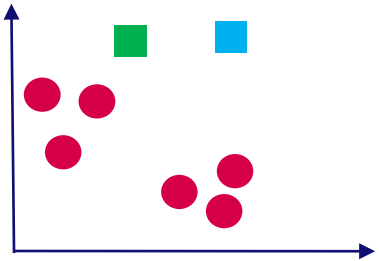
2 neurons □
6 inputs ○



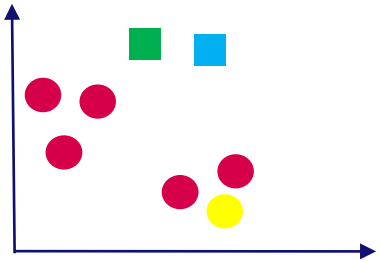
Select random input



Compute winner neuron
Update neurons

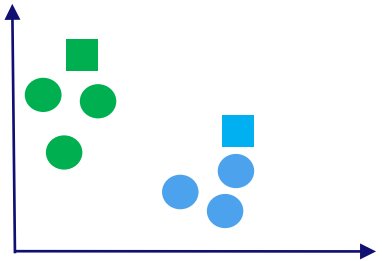


Select random input



Compute winner neuron
Update neurons

Repeat …



Classify input data

© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Neural Networks in practice

© PADS (use only with permission & acknowledgements)

Neural Network in Use
Multiway Classifier

- Neural networks for classification task
  - Single output -> Binary classifier
  - Multiple outputs -> Multiway classification
    - Applied successfully to learning pronunciation
  - For example sigmoid pushes to binary classification

© PADS (use only with permission & acknowledgements)

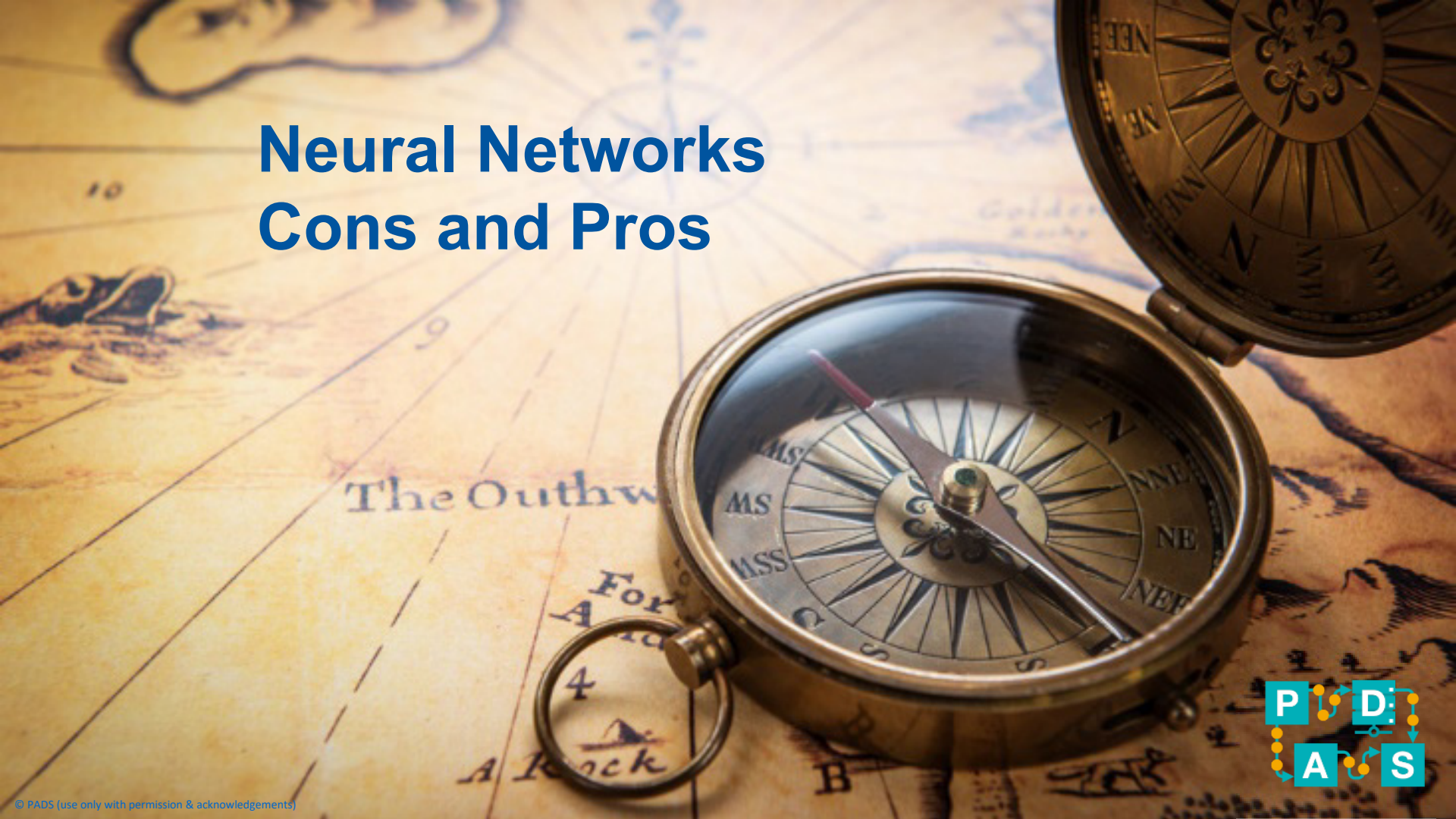
Chair of Process and Data Science

Neural Network in Use

- **Since neural networks are best at identifying patterns or trends they are well suited for prediction or forecasting. Including:**
  - sales forecasting
  - industrial process control
  - customer research
  - data validation
  - risk management
- **ANN are also used for the following specific tasks:**
  - recognition of speakers in communications
  - diagnosis of hepatitis
  - undersea mine detection
  - texture analysis; three-dimensional object recognition
  - hand-written word recognition
  - facial recognition

© PADS (use only with permission & acknowledgements)

Neural Networks
Cons and Pros



© PADS (use only with permission & acknowledgements)

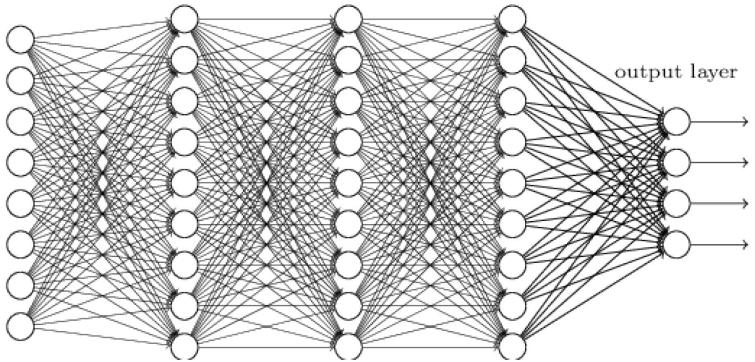
Problems with Neural Networks

- **Interpretation of Hidden Layers**
- **Overfitting**

© PADS (use only with permission & acknowledgements)

Interpretation of Hidden Layers

- **NN treated as black boxes:**
  - **What are the hidden layers doing?!**
    - **Feature Extraction**
    - **The non-linearity in the feature extraction can make interpretation of the hidden layers very difficult.**

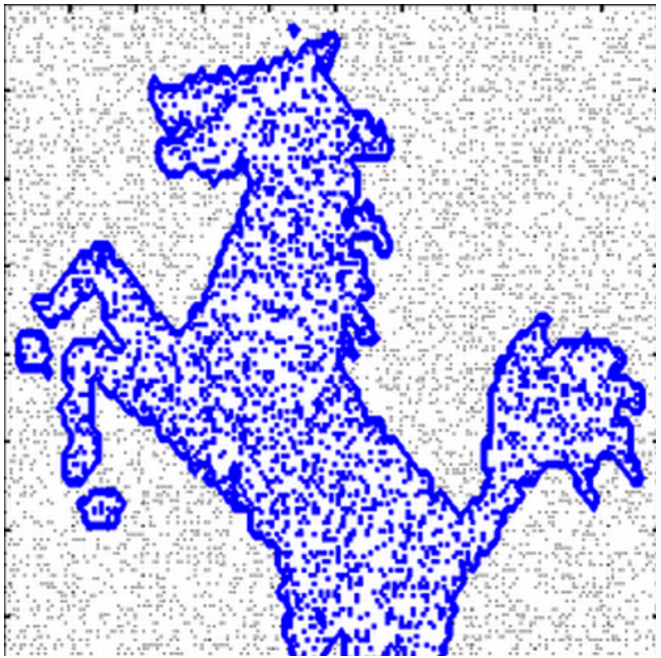


© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

Overfitting in Neural Networks

- **Models with a large number of parameters are able to fit any amount of data available**
  - **Large neural networks can have millions of parameters such as weights.**
  - **So it is very likely that we are overfitting the model.**



Source: Google Image

© PADS (use only with permission & acknowledgements)

Overfitting in Neural Networks

- **Don’t get excited when a complex model fits a data set well.**
  - **With enough parameters, you can fit any data set.**
  - **As John von Neumann said:**

(mathematician, physicist, computer scientist)

> *“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk”*

  - **Many hidden layers/neurons make it easier for the neural network to memorize the training set**

© PADS (use only with permission & acknowledgements)

Cons and Pros

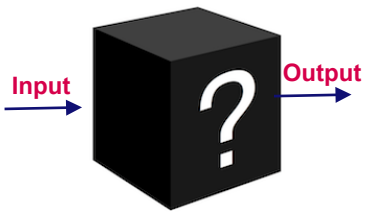
Cons	Pros
Seems biologically implausible	Able to learn any arbitrary function (XOR)
It is a black box! We can't see how it's making decisions?	Works well with noisy data
Works poorly on dense data with few input variables	Rapid recognition speed Has inspired many new algorithms

© PADS (use only with permission & acknowledgements)

Summarized

Disadvantages of Neural Network

- The individual relations between the input variables and the output variables are not controlled well nor visible. The model tends to be a black box or input/output table without clear “rules”.
- The sample size has to be large.
- Requires lot of trial and error
  - so training can be time consuming



© PADS (use only with permission & acknowledgements)

A short detour: Naïve Bayesian Classification

© PADS (use only with permission & acknowledgements)

Yet another supervised learning approach

- **Thomas Bayes (1701-1761), an English statistician and philosopher that invented Bayes' theorem.**
- **Bayes' theorem is often used in probability-based learning.**

© PADS (use only with permission & acknowledgements)

Basic statistics

- $P(X)$ = probability that X holds.
- $P(X, Y)$ = probability that both X and Y hold.
- $P(X|Y)$ = probability that X holds given Y **(conditional probability).**
- **Product rule (always holds):** $P(X, Y) = P(X|Y) \times P(Y)$.
- **Chain rule (always holds):** $P(A, B, \dots, Y, Z)$
  $= P(A, B, \dots, Y|Z)P(Z)$
  $= P(A|B, \dots, Z)P(B|C, \dots, Z) \dots P(Y|Z)P(Z)$.

© PADS (use only with permission & acknowledgements)

Basic statistics

- $P(\neg X) = 1 - P(X)$
- $P(\neg X|Y) = 1 - P(X|Y)$
- $P(Y) = P(Y|X)P(X) + P(Y|\neg X)P(\neg X)$
- **Assuming independence:**
  $P(A, B, \dots, Y, Z) = P(A)P(B) \dots P(Z)$.
  Warning: Only if $A, B, \dots, Y, Z$ are really independent!

© PADS (use only with permission & acknowledgements)

Bayes' Theorem

- $P(X|Y)P(Y) = P(X,Y) = P(Y|X)P(X)$.
- **Rewrite:** $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$

© PADS (use only with permission & acknowledgements)

Bayes' Theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- Example: X = has disease ABC, Y = test is positive.
- Suppose:
  - $P(X) = 0.05$,
  - $P(Y|X) = 0.99$,
  - $P(Y|\neg X) = 0.01$.
- The doctor says you have disease ABC (Y holds, meaning that the test is positive).
- What is the probability you actually have the disease ABC (i.e., $P(X|Y)$)?

© PADS (use only with permission & acknowledgements)

Bayes' Theorem

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- X = has disease ABC, Y = test is positive.
- $P(X) = 0.05$, $P(Y|X) = 0.99$, $P(Y|\neg X) = 0.01$.
- Assume Y holds (doctor says you have the disease).
- $P(Y) = P(Y|X)P(X) + P(Y|\neg X)P(\neg X) = 0.99 \times 0.05 + 0.01 \times 0.95 = 0.059$
- $P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} = \frac{0.99 \times 0.05}{0.059} = 0.83898$

© PADS (use only with permission & acknowledgements)

Applied to classification

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

- X = target feature has a specific value
- Y = descriptive features have specific values
- Probabilities can be estimated in a trivial manner (count fractions of rows).

Generalized Bayes' Theorem

$$P(t = l|\mathbf{q}[1], \ldots, \mathbf{q}[m]) = \frac{P(\mathbf{q}[1], \ldots, \mathbf{q}[m]|t = l)P(t = l)}{P(\mathbf{q}[1], \ldots, \mathbf{q}[m])}$$

© PADS (use only with permission & acknowledgements)

Maximize a Posteriori (MAP)

Bayesian MAP Prediction Model

$$\mathbb{M}_{MAP}(\mathbf{q}) = \underset{l \in levels(t)}{\text{argmax}}\, P(t = l \mid \mathbf{q}[1], \ldots, \mathbf{q}[m])$$

$$= \underset{l \in levels(t)}{\text{argmax}}\, \frac{P(\mathbf{q}[1], \ldots, \mathbf{q}[m] \mid t = l) \times P(t = l)}{P(\mathbf{q}[1], \ldots, \mathbf{q}[m])}$$

Bayesian MAP Prediction Model (without normalization)

$$\mathbb{M}_{MAP}(\mathbf{q}) = \underset{l \in levels(t)}{\text{argmax}}\, P(\mathbf{q}[1], \ldots, \mathbf{q}[m] \mid t = l) \times P(t = l)$$

© PADS (use only with permission & acknowledgements)

Naïve Bayes' Classifier

Naive Bayes' Classifier

$$\mathbb{M}(\mathbf{q}) = \underset{l \in levels(t)}{\operatorname{argmax}} \left(\prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l) \right) \times P(t = l)$$

Assume independence to avoid overfitting!

© PADS (use only with permission & acknowledgements)

Example

ID	Credit History	Guarantor/ CoApplicant	Accomodation	Fraud
1	current	none	own	true
2	paid	none		
3	paid	none		
4	paid	guarantor		
5	arrears	none		
6	arrears	none		
7	current	none		
8	arrears	none		
9	current	none		
10	none	none		
11	current	coapplicant		
12	current	none		
13	current	none		
14	paid	none		
15	arrears	none		
16	current	none		
17	arrears	coapplicant		
18	arrears	none	free	false
19	arrears	none	own	false
20	paid	none	own	false

$$P(fr) = 0.3 \qquad P(\neg fr) = 0.7$$

$$P(\text{CH} = \textit{'none'} \mid fr) = 0.1666 \qquad P(\text{CH} = \textit{'none'} \mid \neg fr) = 0$$

$$P(\text{CH} = \textit{'paid'} \mid fr) = 0.1666 \qquad P(\text{CH} = \textit{'paid'} \mid \neg fr) = 0.2857$$

$$P(\text{CH} = \textit{'current'} \mid fr) = 0.5 \qquad P(\text{CH} = \textit{'current'} \mid \neg fr) = 0.2857$$

$$P(\text{CH} = \textit{'arrears'} \mid fr) = 0.1666 \qquad P(\text{CH} = \textit{'arrears'} \mid \neg fr) = 0.4286$$

$$P(\text{GC} = \textit{'none'} \mid fr) = 0.8334 \qquad P(\text{GC} = \textit{'none'} \mid \neg fr) = 0.8571$$

$$P(\text{GC} = \textit{'guarantor'} \mid fr) = 0.1666 \qquad P(\text{GC} = \textit{'guarantor'} \mid \neg fr) = 0$$

$$P(\text{GC} = \textit{'coapplicant'} \mid fr) = 0 \qquad P(\text{GC} = \textit{'coapplicant'} \mid \neg fr) = 0.1429$$

$$P(\text{ACC} = \textit{'own'} \mid fr) = 0.6666 \qquad P(\text{ACC} = \textit{'own'} \mid \neg fr) = 0.7857$$

$$P(\text{ACC} = \textit{'rent'} \mid fr) = 0.3333 \qquad P(\text{ACC} = \textit{'rent'} \mid \neg fr) = 0.1429$$

$$P(\text{ACC} = \textit{'free'} \mid fr) = 0 \qquad P(\text{ACC} = \textit{'free'} \mid \neg fr) = 0.0714$$

Example

$P(fr)$	=	0.3	$P(\neg fr)$	=	0.7
$P(\text{CH} = \text{'none'} \mid fr)$	=	0.1666	$P(\text{CH} = \text{'none'} \mid \neg fr)$	=	0
$P(\text{CH} = \text{'paid'} \mid fr)$	=	0.1666	$P(\text{CH} = \text{'paid'} \mid \neg fr)$	=	0.2857
$P(\text{CH} = \text{'current'} \mid fr)$	=	0.5	$P(\text{CH} = \text{'current'} \mid \neg fr)$	=	0.2857
$P(\text{CH} = \text{'arrears'} \mid fr)$	=	0.1666	$P(\text{CH} = \text{'arrears'} \mid \neg fr)$	=	0.4286
$P(\text{GC} = \text{'none'} \mid fr)$	=	0.8334	$P(\text{GC} = \text{'none'} \mid \neg fr)$	=	0.8571
$P(\text{GC} = \text{'guarantor'} \mid fr)$	=	0.1666	$P(\text{GC} = \text{'guarantor'} \mid \neg fr)$	=	0
$P(\text{GC} = \text{'coapplicant'} \mid fr)$	=	0	$P(\text{GC} = \text{'coapplicant'} \mid \neg fr)$	=	0.1429
$P(\text{ACC} = \text{'own'} \mid fr)$	=	0.6666	$P(\text{ACC} = \text{'own'} \mid \neg fr)$	=	0.7857
$P(\text{ACC} = \text{'rent'} \mid fr)$	=	0.3333	$P(\text{ACC} = \text{'rent'} \mid \neg fr)$	=	0.1429
$P(\text{ACC} = \text{'free'} \mid fr)$	=	0	$P(\text{ACC} = \text{'free'} \mid \neg fr)$	=	0.0714

CREDIT HISTORY	GUARANTOR/COAPPLICANT	ACCOMODATION	FRAUDULENT
paid	none	rent	?

Example

Naive Bayes' Classifier

$$\mathbb{M}(\mathbf{q}) = \underset{l \in levels(t)}{\operatorname{argmax}} \left(\prod_{i=1}^{m} P(\mathbf{q}[i] \mid t = l) \right) \times P(t = l)$$

$P(fr)$	$=$	0.3	$P(\neg fr)$	$=$	0.7
$P(\text{CH} = \text{'paid'} \mid fr)$	$=$	0.1666	$P(\text{CH} = \text{'paid'} \mid \neg fr)$	$=$	0.2857
$P(\text{GC} = \text{'none'} \mid fr)$	$=$	0.8334	$P(\text{GC} = \text{'none'} \mid \neg fr)$	$=$	0.8571
$P(\text{ACC} = \text{'rent'} \mid fr)$	$=$	0.3333	$P(\text{ACC} = \text{'rent'} \mid \neg fr)$	$=$	0.1429

$$\left(\prod_{k=1}^{m} P(\mathbf{q}[k] \mid fr) \right) \times P(fr) = 0.0139$$

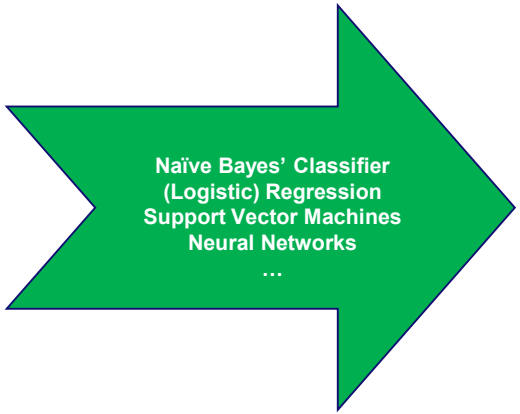
$$\left(\prod_{k=1}^{m} P(\mathbf{q}[k] \mid \neg fr) \right) \times P(\neg fr) = 0.0245$$

argmax

CREDIT HISTORY	GUARANTOR/COAPPLICANT	ACCOMODATION	FRAUDULENT
paid	none	rent	*'false'*

Supervised learning

descriptive features



target feature

training instances vs unseen instances

© PADS (use only with permission & acknowledgements)

Conclusion

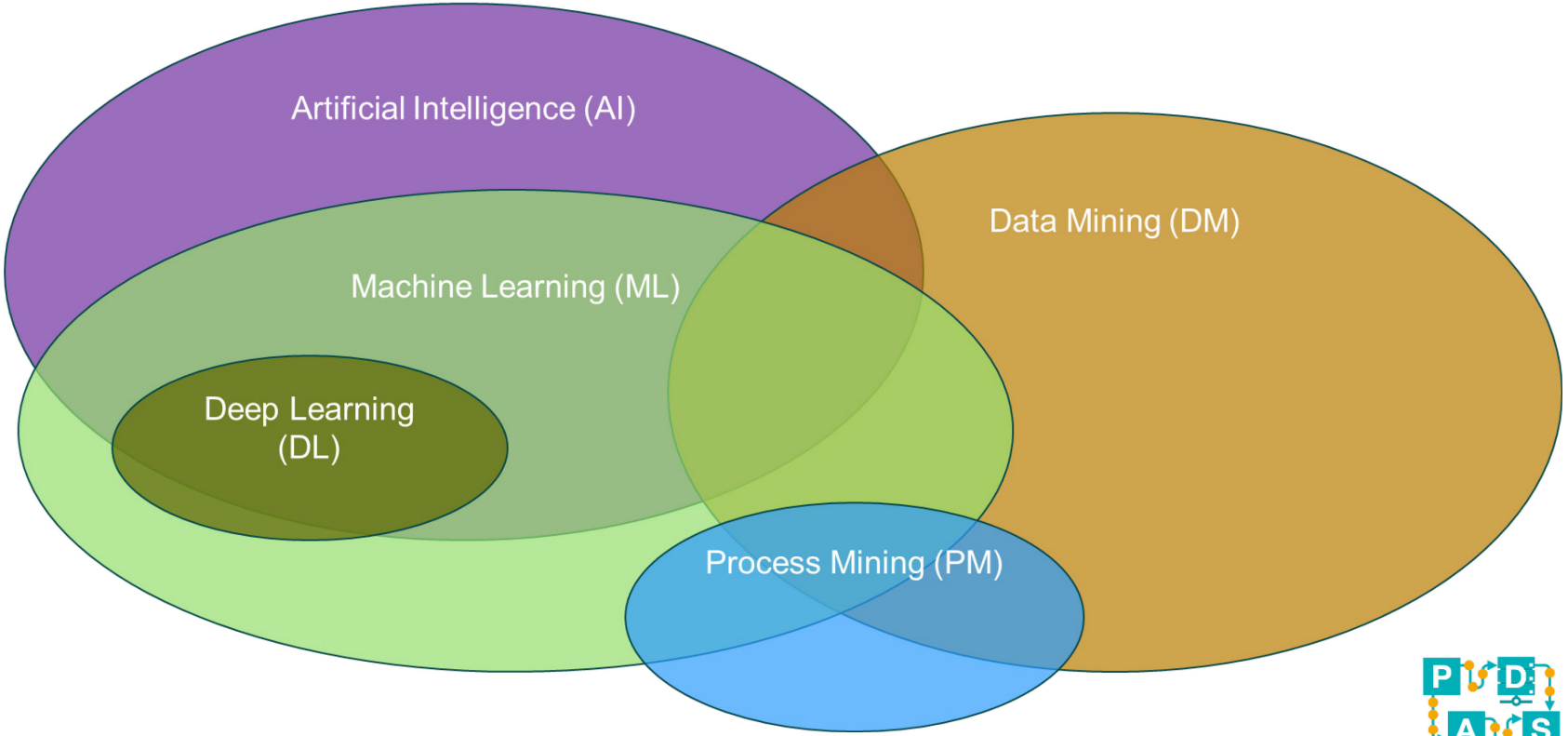
© PADS (use only with permission & acknowledgements)

Summary

- Dealing with non-linear problems with a large number of features to consider
- ANNs are useful in many data science problems
- Backpropagation as an learning approach in NNs
- Performance and amount of training data should be considered
- No silver bullet!

© PADS (use only with permission & acknowledgements)

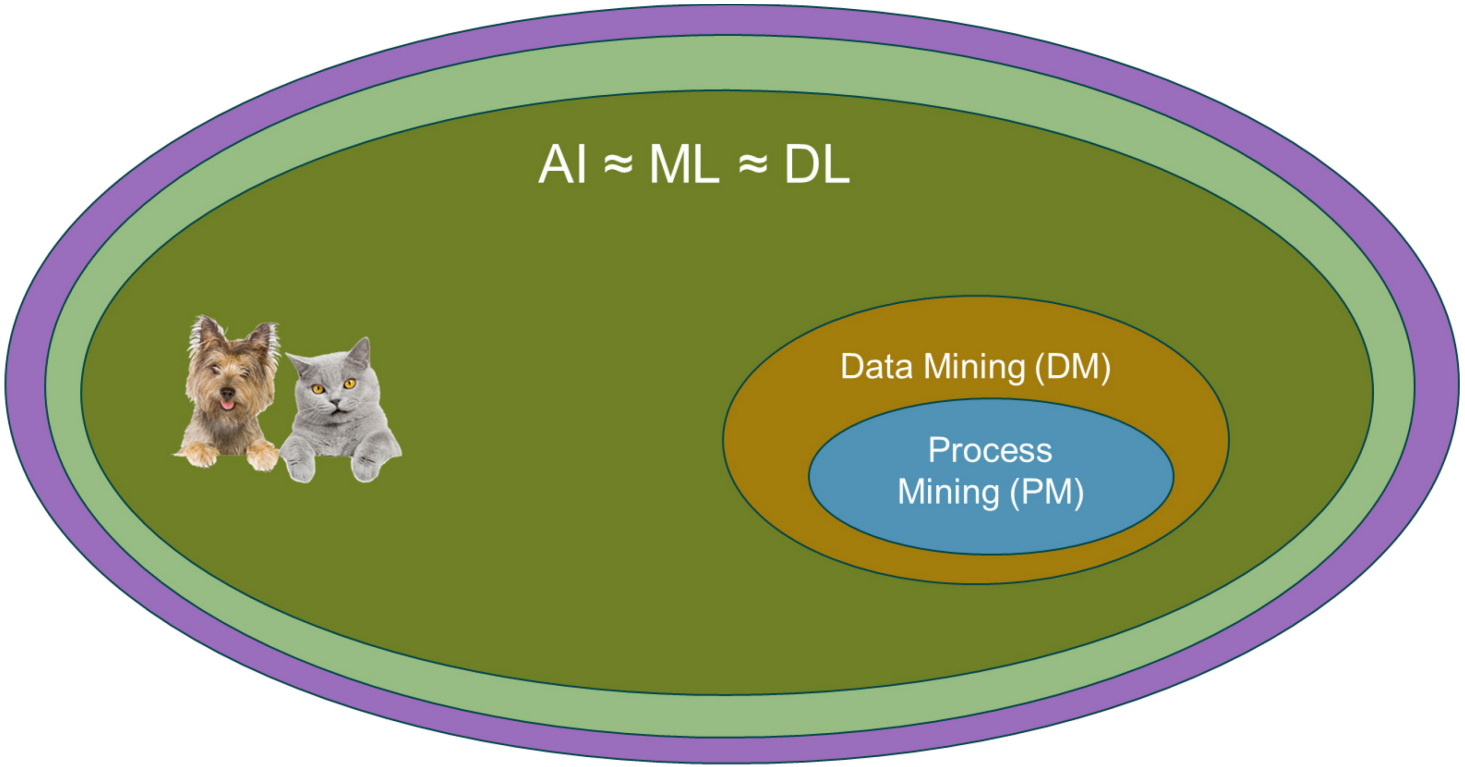
Recall: A more balanced view



© PADS (use only with permission & acknowledgements)

Perception by outsiders

Now you know this view is very misleading and naïve (AI/DL involves lots of engineering, parameters, heuristics, network choices, “voodoo”, …)



© PADS (use only with permission & acknowledgements)

Chair of Process and Data Science

#	Lecture	date	day
Lecture 1	Introduction	10/10/2018	Wednesday
Lecture 2	Crash Course in Python	11/10/2018	Thursday
Instruction 1	*Python*	*12/10/2018*	*Friday*
Lecture 3	Basic data visualisation/exploration	17/10/2018	Wednesday
Lecture 4	Decision trees	18/10/2018	Thursday
Instruction 2	*Decision trees and data visualization/exploration*	*19/10/2018*	*Friday*
Lecture 5	Regression	24/10/2018	Wednesday
Lecture 6	Support vector machines	25/10/2018	Thursday
Instruction 3	*Regression and support vector machines*	*26/10/2018*	*Friday*
Lecture 7	Neural networks (1/2)	31/10/2018	Wednesday
Instruction 4	*Neural networks and supervised learning*	*02/11/2018*	*Friday*
Lecture 8	Neural networks (2/2)	07/11/2018	Wednesday
Lecture 9	Evaluation of supervised learning problems	08/11/2018	Thursday
Instruction 5	*Neural networks and supervised learning*	*09/11/2018*	*Friday*
Lecture 10	Clustering	14/11/2018	Wednesday
Lecture 11	Frequent items sets	15/11/2018	Thursday
Lecture 12	Association rules	21/11/2018	Wednesday
Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association rules*	*23/11/2018*	*Friday*
Instruction 7			
Lecture 1			
Instruction 8			
Lecture 1			
Lecture 1			
Lecture 1			
backup			
Instruction 9			
Lecture 2			
Lecture 2			
Instruction 10			
Lecture 2			
Lecture 2			
Instruction 11			
Lecture 2			
backup			
Instruction 12	*Example exam questions*	*25/01/2018*	*Friday*
backup		30/01/2019	Wednesday
backup		31/01/2019	Thursday
extra	*Question hour*	*01/02/2019*	*Friday*