

## Lab 6 – Classification & Prediction #2

*Data Mining, Spring 2018*

- Remember the deadline: Monday April 2<sup>nd</sup>
- Feel free to use today's lab on the assignment
- Also feel free to ask questions
  - Either now at the lab
  - On the Q&A forum on learnIT
  - Or via email: [berm@itu.dk](mailto:berm@itu.dk) / [dafr@itu.dk](mailto:dafr@itu.dk) / [jams@itu.dk](mailto:jams@itu.dk) (email both of us)

*Today's Lab: Neural Networks and  
Backpropagation*

# Classification & Prediction #2

- Today you will create a digital brain capable of learning!
- Two options
  - A perceptron
  - Neural network with backpropagation (advanced, but well explained in the book)



# Overview of the two approaches

## Perceptron

- One of the simplest neural networks
- Binary input/output
- Topology:
  - One node which is fed variable amount of inputs, and produces a single output
- Activation function = step function
- Best for two class label problems
- Ignored by the book
  - Info available at: <https://en.wikipedia.org/wiki/Perceptron>

$$f(x) = \begin{cases} 1 & \text{if } \sum(\text{weight} \cdot \text{input}) + \text{bias} > 0 \\ 0 & \text{otherwise} \end{cases}$$

# Overview of the two approaches

## Neural Network with Backpropagation

- Advanced neural network
- More topology choices decided by you
  - Input/output
  - Number of nodes
  - Activation function
- Can be used in many different scenarios
- Detailed walkthrough of the algorithm in the book, chapter 9.2

# Plan of attack

## Perceptron

- Implement necessary data structures to build the perceptron
- Construct your perceptron
  - Try first with two inputs
  - Then implement weight updating
  - Example with results available in help slides

# Plan of attack

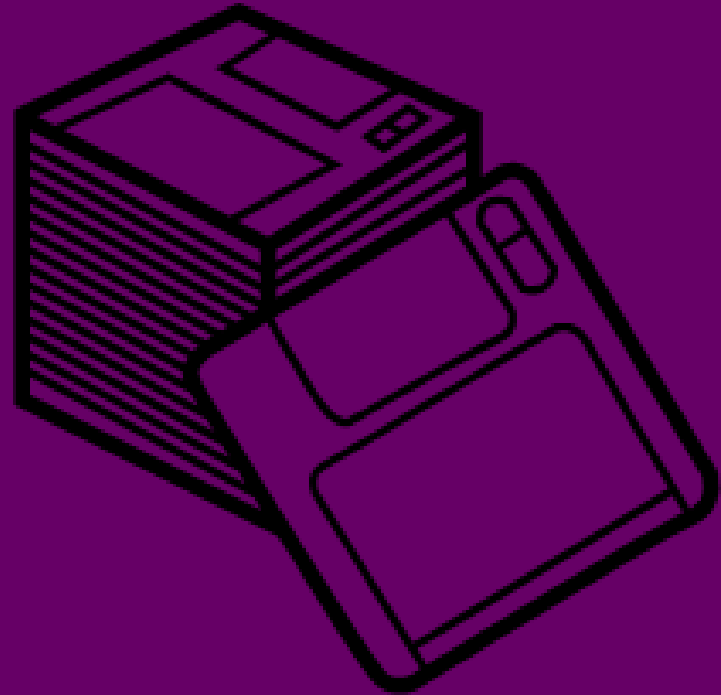
## Neural Network with Backpropagation

- You will be building a digital brain to predict which class different Iris flowers belong to.
- Code is provided to help you load in the data.
- Then start working on your neural network
  - First step: Normalize data!
  - Then construct the neural network:
    - Data structure to store layers
    - Topology?
- Then implement Backpropagation
- After implementing the perceptron use k-fold cross validation (chapter 8.5.3) with  $k = 10$  to measure prediction accuracy. (*optional*)



# The Data

- Same data as clustering #1
- The iris data can be found in the iris.csv file in the java-project.
- Attributes:
  - Sepal length
  - Sepal width
  - Petal length
  - Petal width
  - Class
- Possible values: Iris-setosa, Iris-versicolor and Iris-virginica



# Code Provided

- Iris class used to store data for each Iris flower in data.
- Data loading and conversion to Iris-objects
  - Done by the CSVFileReader and DataLoader class.
- Main-class contains Main-function
  - Currently it calls the LoadData method of the DataLoader which returns an ArrayList of all Iris objects loaded in from the data file.

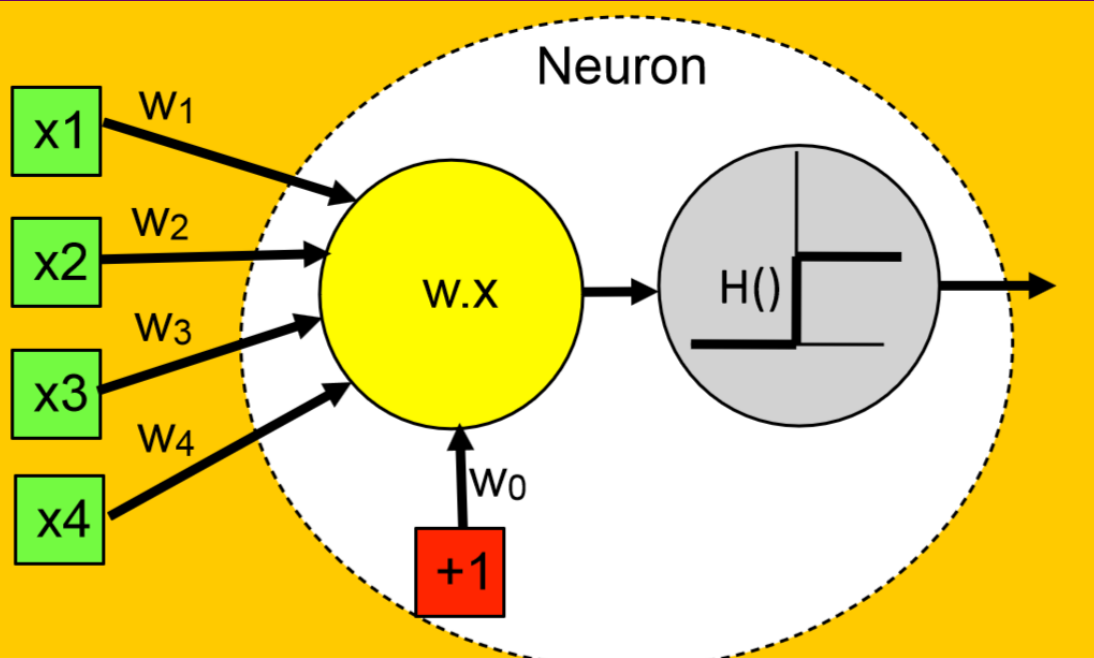
*Thanks for listening!*  
*Help Slides available below*

*Help!!!*

# Perceptron

- To simplify the topology in regards to the output layer consider making two perceptrons instead of one.
- E.g.
  - One to classify between Class 1 and Class 2
  - Another to classify between Class 2 and Class 3.
- If you do this, you will only need one output neuron.
- Requires you to split data

# Neural networks 101



- Inputs (X)
- Connection Weights (W)
- Threshold/bias
- Weighted Sum
- Activation function
- Output

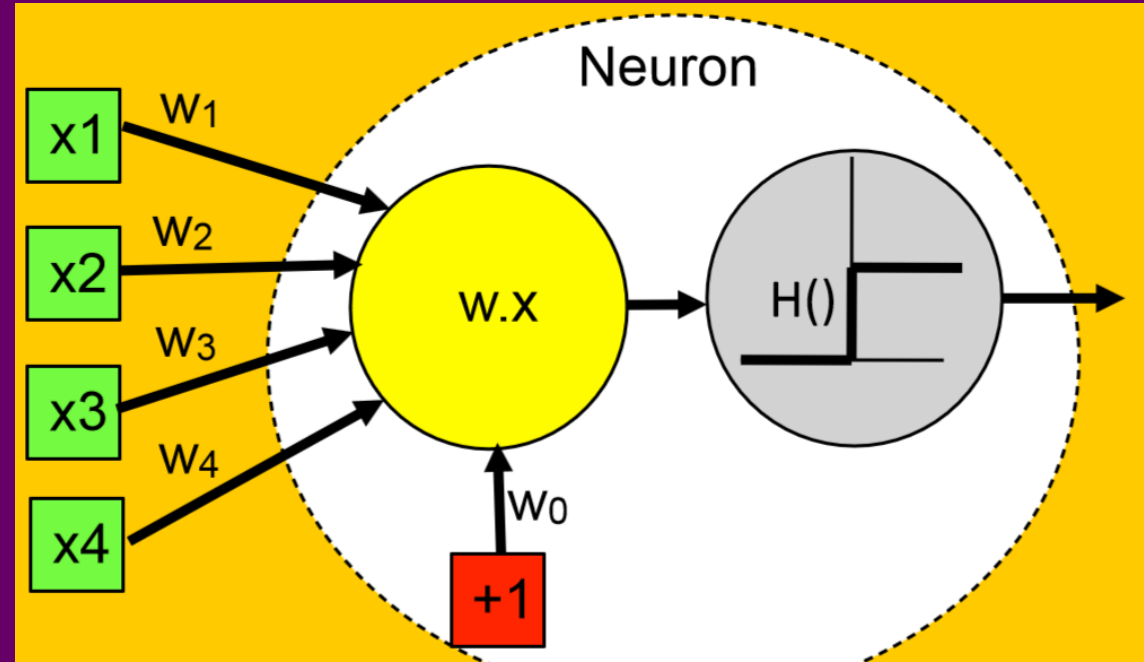
Slide from a lecture given by Hector P. Martinez in the course Modern AI for Games, Fall 2012

# Perceptron algorithm

1. Initialize perceptron with random weights  $[0...1]$  and bias value  $[0...1]$ , or with your own values (e.g. bias = 0)
2. For each set of inputs
  - Compute actual output,  $a_p$ , from perceptron using the activation function (more on next slide)
  - Update all weights with  $\Delta w_j$
3. If no changes to weights, then stop
4. Otherwise go back to 2

# Perceptron algorithm – weight update

- $\Delta w_j = \text{Learning rate} \cdot x_j \cdot (d_j - a_j)$
- Example with  $w_1$ , learning rate = 0.5,  $d_j = 0$ ,  $a_j = 1$ :
  - $\Delta w_1 = 0.5 \cdot x_1 \cdot (0 - 1)$
- $x_j$  = output from neuron  $x_j$
- $d_j$  = desired output
- $a_j$  = actual output





# Perceptron example

Threshold = Bias

Slide from a lecture given by Hector P. Martinez in the course Modern AI for Games, Fall 2012

Epoch	Inputs		Desired output $d^p$	Initial weights		Actual output $a^p$	Error $E^p$	Final weights	
	$x_1$	$x_2$		$w_1$	$w_2$			$w_1$	$w_2$
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	-1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	-1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	-1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
Threshold: $w_0 = -0.2$ ; learning rate $\eta = 0.1$									