LAB 6 – CLASSIFICATION & PREDICTION #2

DATA MINING SPRING 2014 | ANDERS HARTZEN (ANDERSHH@ITU.DK) & JENS ANDERSSON GRØN (JANG@ITU.DK)

TODAY'S LAB

Brains!

CLASSIFICATION & PREDICTION #2

- Today you will create a digital brain capable of learning!
- Two options
  - A perceptron
  - Neural network (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
  - $f(x) = \begin{Bmatrix} 1\ if\ \sum(weight \cdot input) + bias > 0 \\ 0\ otherwise \end{Bmatrix}$
- Best for two class label problems
- Ignored by the book
  - Info available at: https://en.wikipedia.org/wiki/Perceptron

Neural Network with backpropagation

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

PLAN OF ATTACK

Perceptron

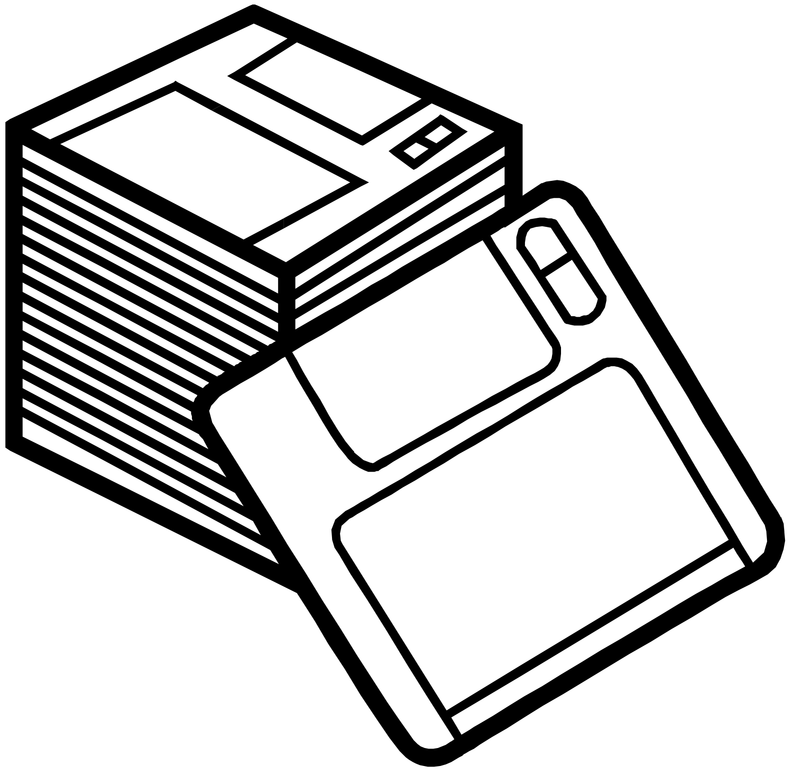
- Implement necessary data structures to build the perceptron
- Construct your perceptron
  - Try first with two inputs
- Then implement weight updating
  - Example with results available in help slides

Neural Network

- 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)

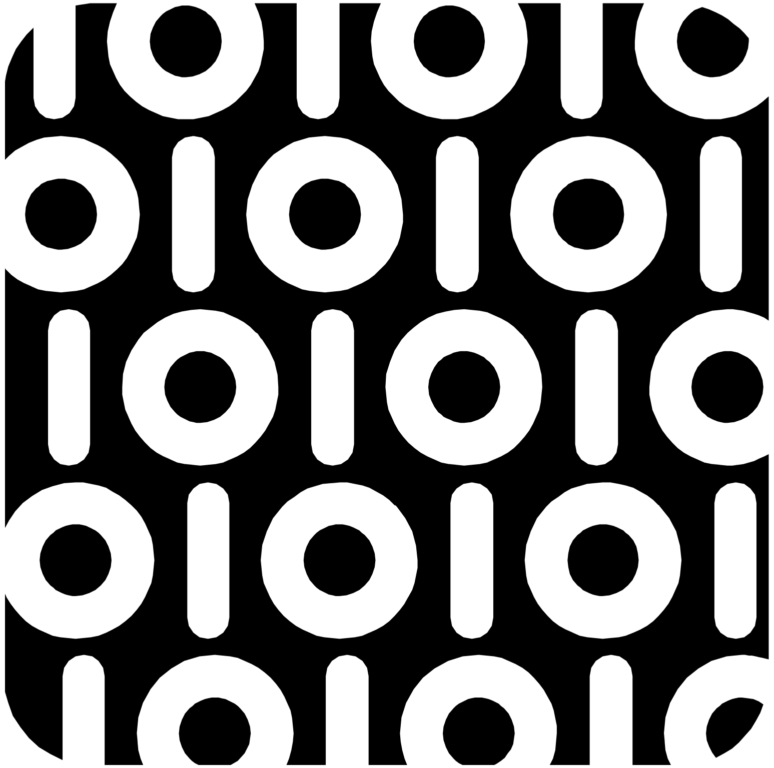
THE DATA (NEURAL NETWORK)

- 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.



☺ ☺
☺ ☺

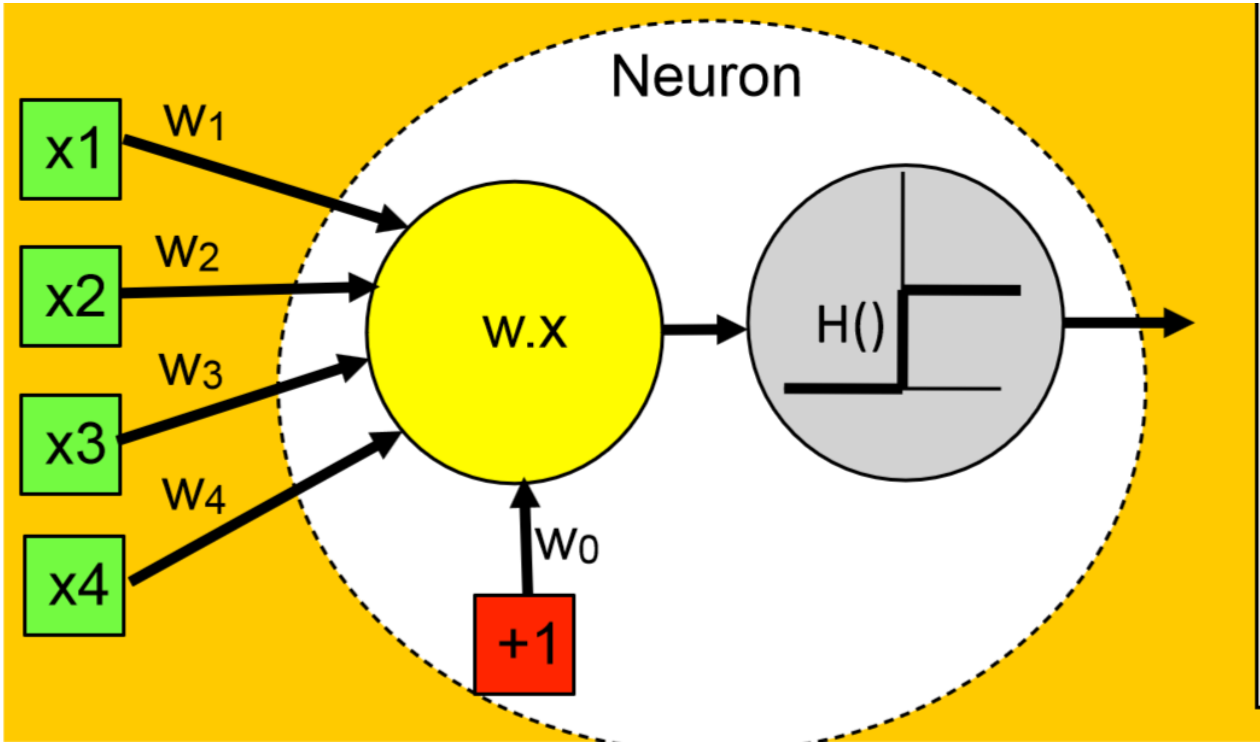
THANK YOU FOR LISTENING!

HELP SLIDES

NEURAL NETWORK – SMALL TIP

- 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 neurode (or node).
- Requires you to split data

PERCEPTRON/NEURAL NETWORK BASICS



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

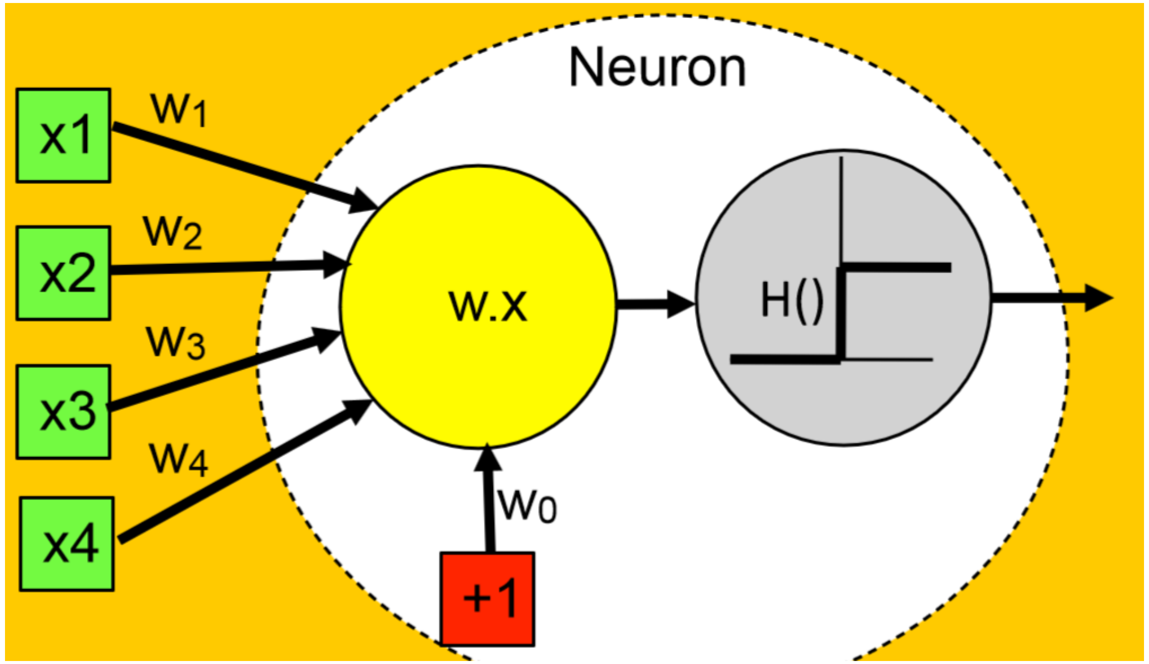
Slide from a lecture given by Hector P. Martinez in the course *Modern AI for Games*, Fall 2012

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 ΔW_j
3. If no changes to weights, then stop
4. Otherwise go back to 2

PERCEPTRON ALGORITHM – UPDATING WEIGHTS

- ΔW_j = Learning rate * Xj * (desired output – actual output)
- Example with W_1, learning rate = 0.5, desired output = 0, actual output = 1:
- $\Delta W_1 = 0.5 * X1 * (0 - 1)$



PERCEPTRON EXAMPLE

Threshold = Bias

Epoch	Inputs x_1	Inputs x_2	Desired output d^p	Initial weights w_1	Initial weights w_2	Actual output a^p	Error E^p	Final weights w_1	Final weights 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 $\eta = 0.1$

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