Handwritten Digit Recognition

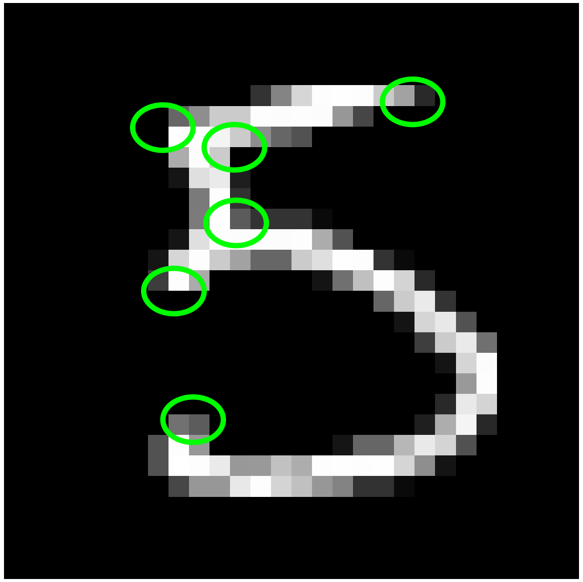
Problem

- How can you make a computer recognize single handwritten digits?

(Pictures from MNIST-database; [1])

Problem

- ‚Basic‘ Approach: Using Image processing for detection of special features: edges, corners, angles, etc.

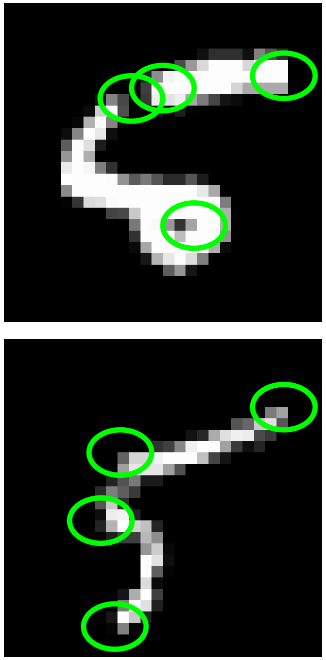


For example: This features might be detected and their relative positioning and alignment to each other might me used for classification

(possible outcome of SIFT-features; [2])

Problem

- But what if a 5 looks like that:



It‘s very difficult (you might even say impossible) to create a set of rules, which define the concept of a 5

Introduction

- Is there a possibility to make a program understand the concept of given data itself?

  → Yes, there is!

  – Machine Learning Algorithms

  – Used Here: Artificial Neural Network (NN)

[3]

Excursion: NN

- Inspired by the brain: A huge amount of simple computing units (brain-cells/neurons) heavily interconnected (synapses)
- Impressively good at classification, learning and memory; rather bad at precise computation

[4]

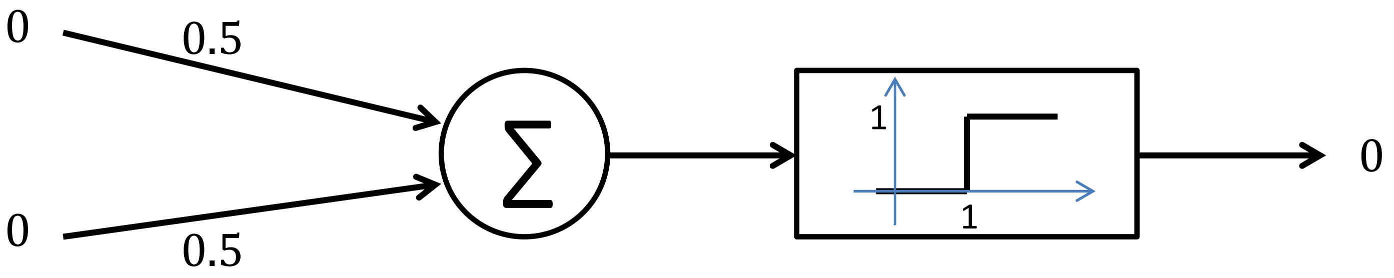
Excursion: NN

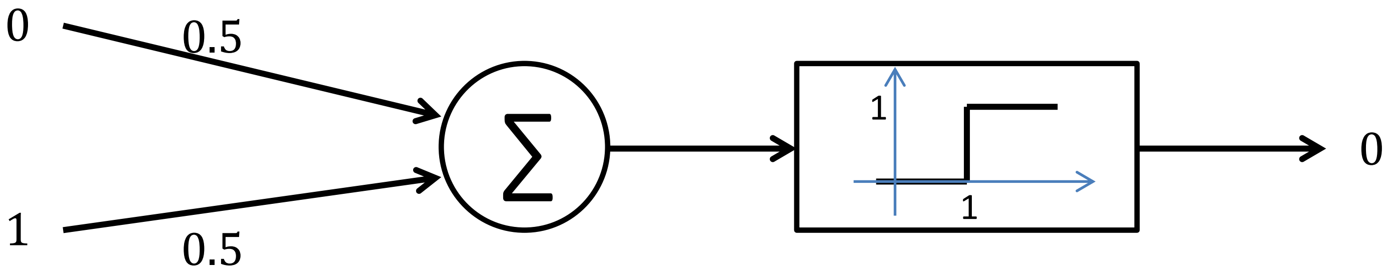
Concept of a single (artificial) neuron:

- A number of inputs, each multiplicated with a special weight, are added up in the neuron
- If the sum reaches a special value, the neuron is being activated and ‚fires‘ a signal

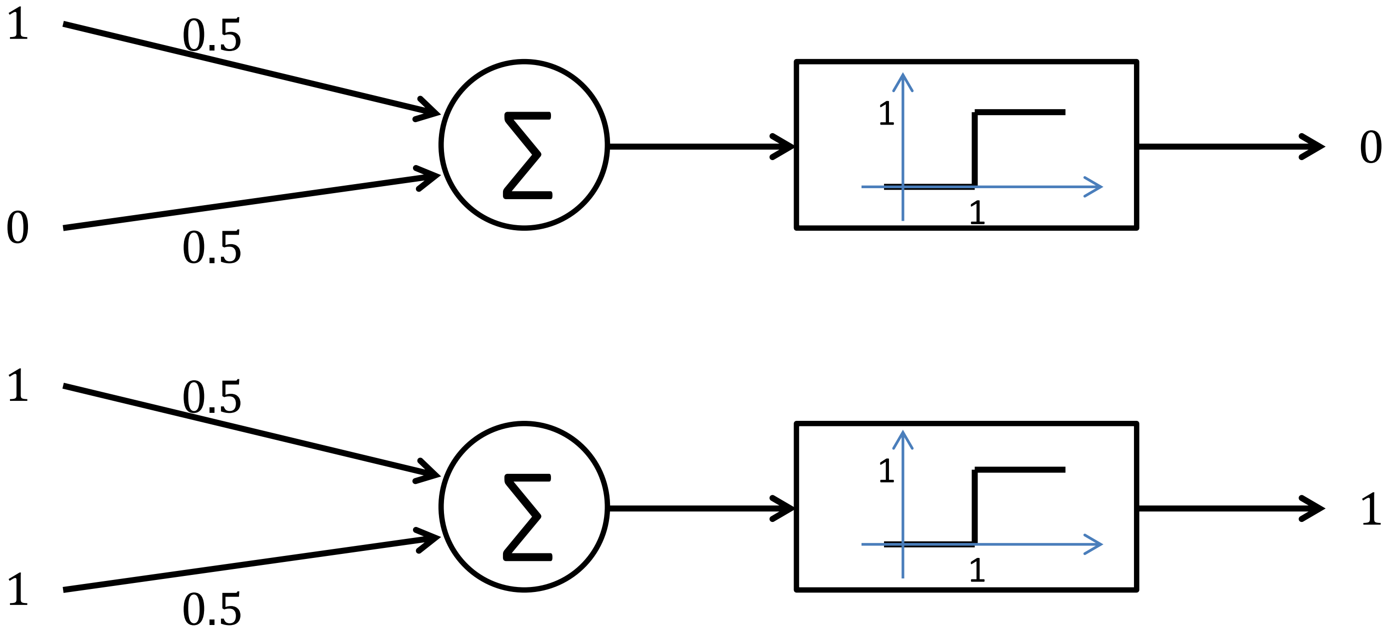
Excursion: NN

Concept of a single (artificial) neuron: Example





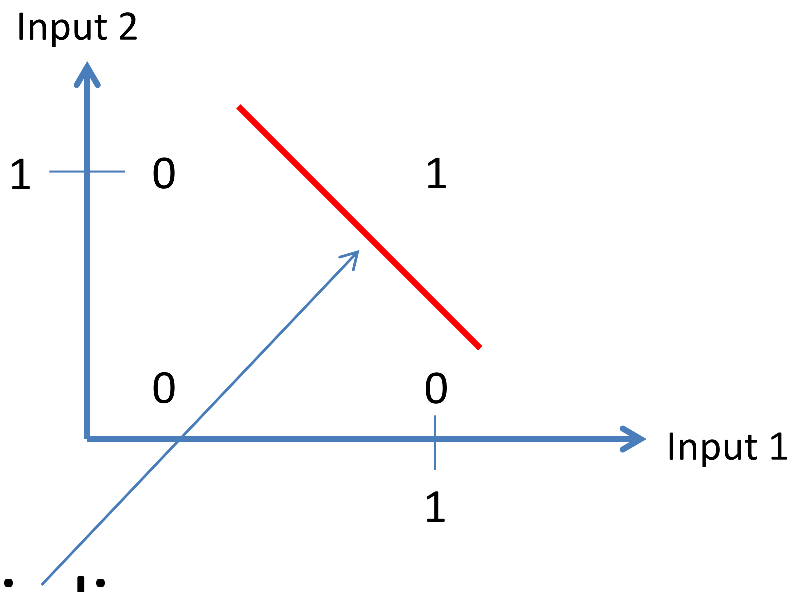
Excursion: NN



[5]

Excursion: NN

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

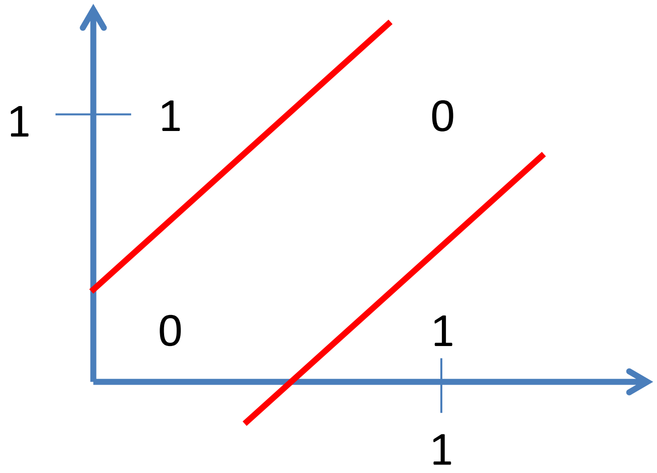


The weights define this line

(boolean AND)

Excursion: NN

- A single neuron can‘t handle XOR



- Multiple neurons in several layers can handle much more complex problems

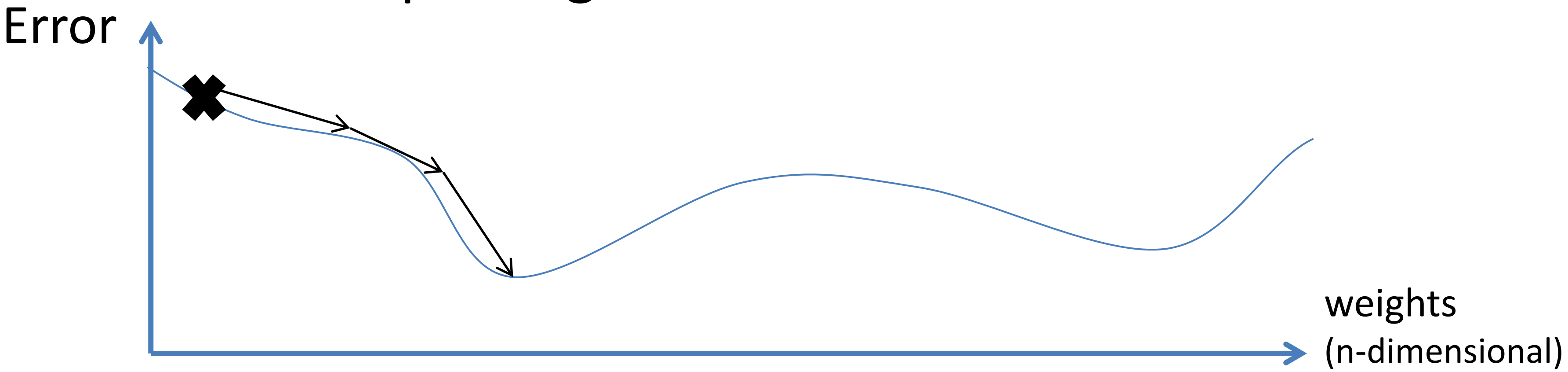
Excursion: NN

- How are the weights chosen, to solve a problem?

→ put in data example

→ measure error at output(s)

→ adapt weights to reduce error

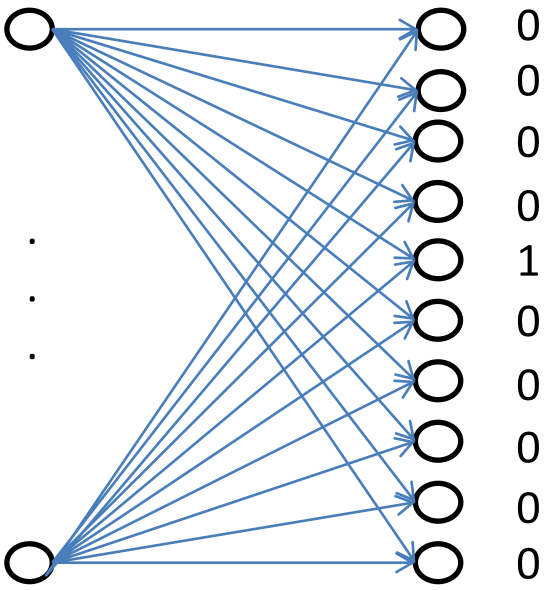


Methods

- MNIST-database-files contain 1000 pictures of a single digit
- Each picture is 28x28 pixels; greyscale
- For input pixels are put in row
  - Creating a vector of 1x784
- Pixel-values are normalized to be between 0 and 1

Methods

- 784 inputs (+bias input) are fully connected to 10 output neurons, representing the 10 different digits (0, …, 9)



→ input image is a 4

Methods

Bias Unit:

„The use of biases in a neural network increases the capacity of the network to solve problems by allowing the hyperplanes that separate individual classes to be offset for superior positioning. “

[6]

Methods

Feed-forward:

- One sample is propagated trough the net and produces output vector
- Error for every output is calculated:

$$error_i = \ out_i(1 - out_i)(target_i - out_i)$$

- Why not simpler: $error = abs(target - out)$?
  - Euclidean distance not applicable here, due to non-linear problem

Methods

Adapt weights:

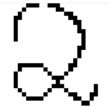
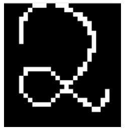
- According to the error every weight - connected to current output (i) - is changed:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$
$$with$$
$$\Delta w_{ji} = \eta * error_i * x_j$$

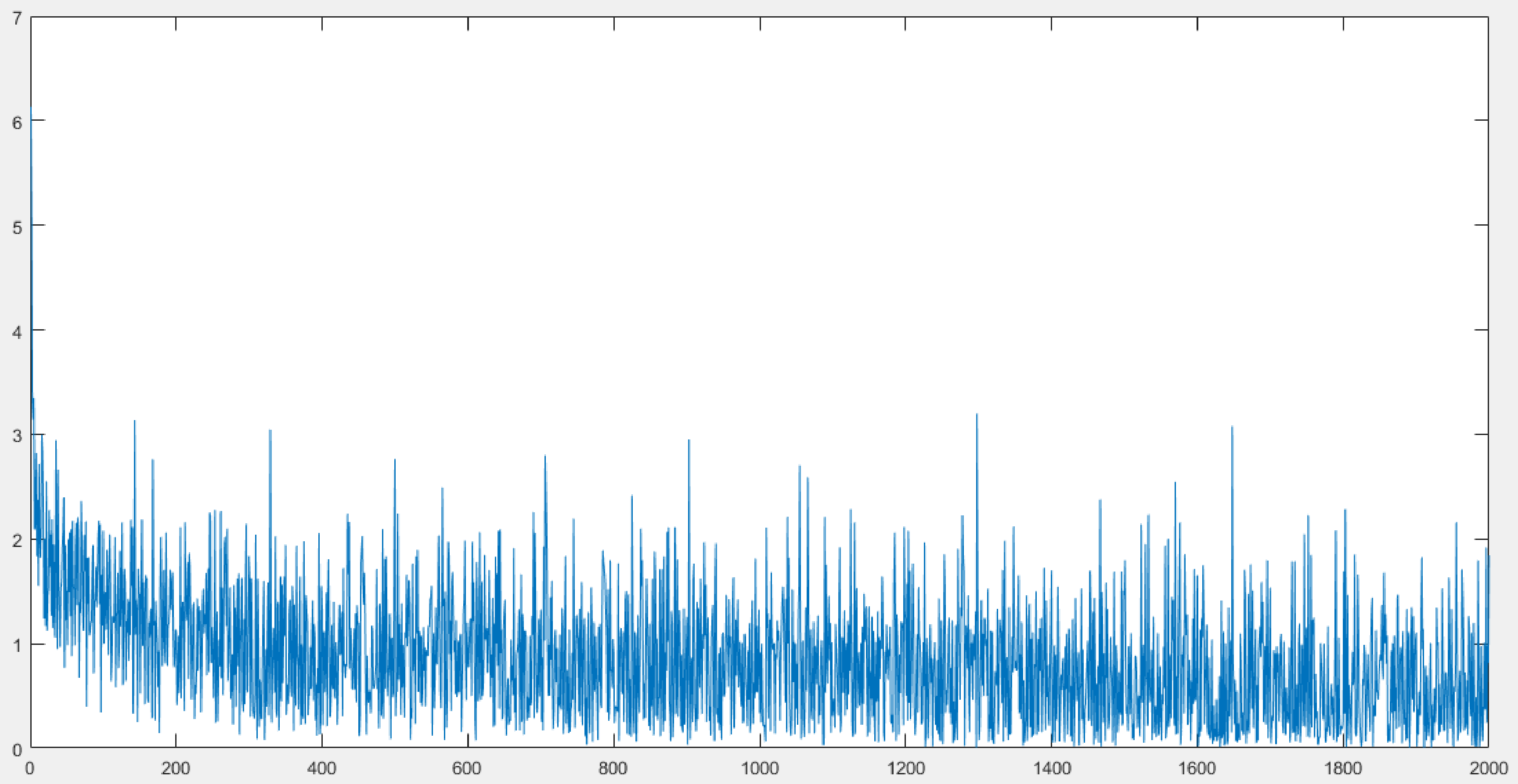
$$\eta = 0.2; learning\ rate$$
$$x_j : input\ j\ (pixel\ j)$$

[7]

Methods

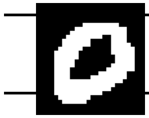
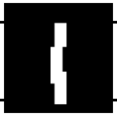
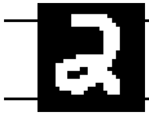
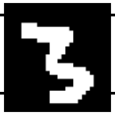
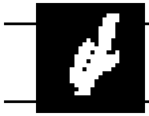
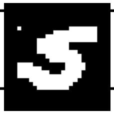
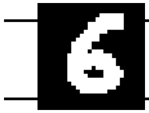
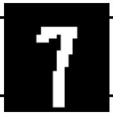
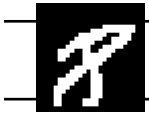
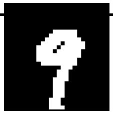
- After a few thousand examples the network learned the ‚concept‘ of handwritten digits
- Now the outcomes should be tested:
  - Loading picture of digit, written by student
  - Scaling picture to 28x28
  - Apply threshold → binary image 
  - Invert colors: now white on black 
  - Put through network; see results

Results



Error-Plot

Results

Picture	Classification	Certainty
	0	0.9990
	1	0.9309
	2	0.9997
	3	0.1275
	4	0.6142
	5	0.2406
	6	0.9956
	7	0.8898
	8	0.7294
	9	0.9750

Not implemented

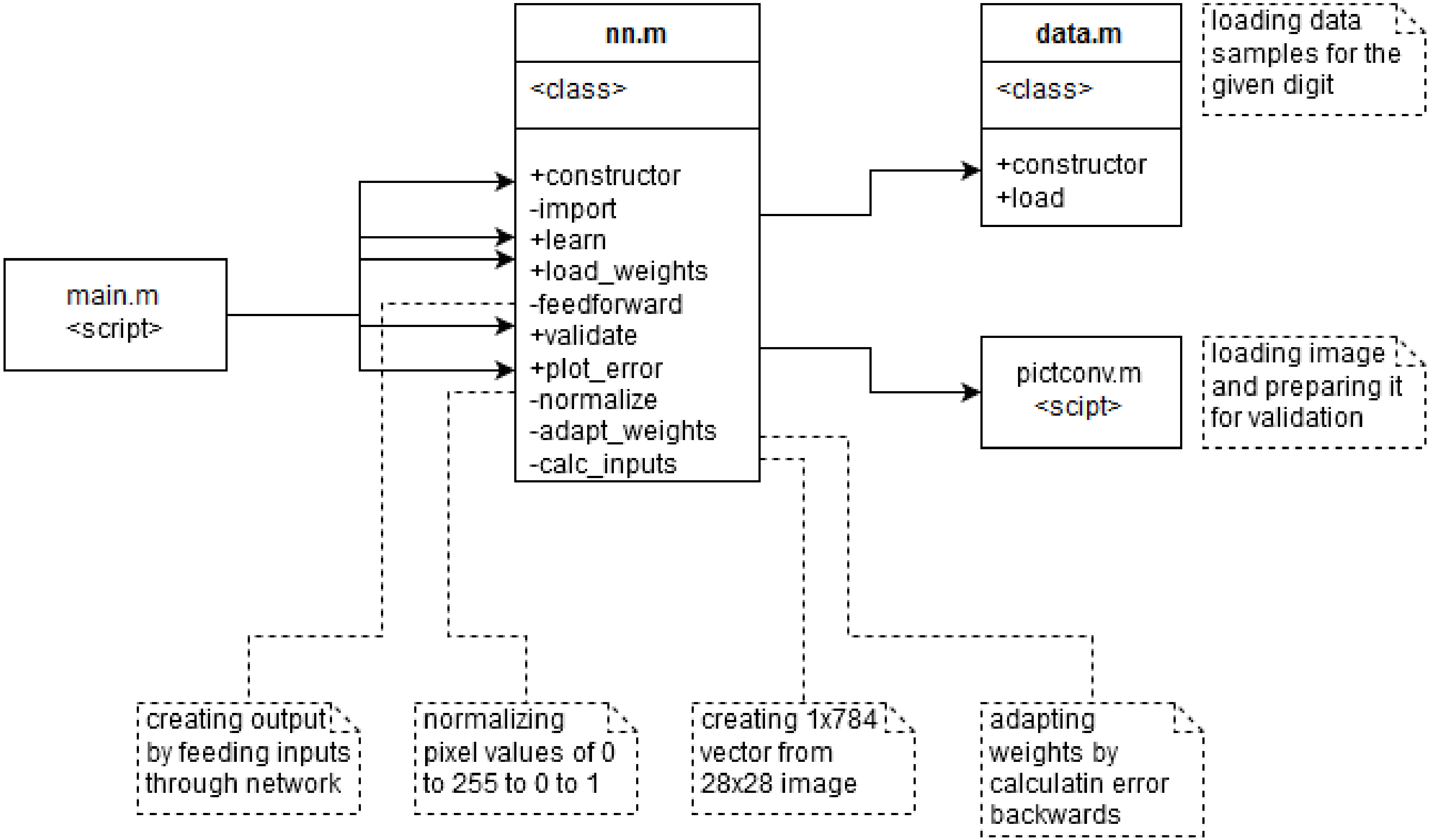
- Using Test-Data
  - Adding complexity to code
  - Unsatisfying results
  - Concept is hard to understand (learning theory)

Add source here

Summary

- Training with only 2000 samples shows good learning behavior (8500 planned)
- Test-data doesn‘t show significant results
- Verification satisfying
- Working with actual photographs also shows good results

Code structure



nn.m

<class>

+constructor
-import
+learn
+load_weights
-feedforward
+validate
+plot_error
-normalize
-adapt_weights
-calc_inputs

data.m

<class>

+constructor
+load

loading data samples for the given digit

main.m
<script>

pictconv.m
<scipt>

loading image and preparing it for validation

creating output by feeding inputs through network

normalizing pixel values of 0 to 255 to 0 to 1

creating 1x784 vector from 28x28 image

adapting weights by calculatin error backwards

matlab functions

`randn`	-creating (array) of small random values; ca. between -4 and 4
`squeeze`	- removing unnecessary array dimensions: 1x1x28x28 to 28x28
`csvread`	- loading .csv file (comma-separated values)
`ceil`	- rounding decimals up to next integer value: 0.3 -> 1
`imread`	- loading image file and returning array
`imresize`	- resizing image to given tuple
`imcomplement`	- returns complement image (“negative”)
`imshow`	- displays image
`gray2ind`	- converts the binary image to grayscale
`imwrite`	- writing image (array) to file

Sources

[1] http://yann.lecun.com/exdb/mnist/

[2] https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[3] https://en.wikipedia.org/wiki/Machine_learning

[4] https://en.wikipedia.org/wiki/Brain#Cellular_structure

[5] http://www.mind.ilstu.edu/curriculum/mcp_neurons/mcp_neuron_1.php

[6] http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

[7] Tom M. Mitchell, Machine Learning, McGraw Hill, ISBN 0-07-115467-1