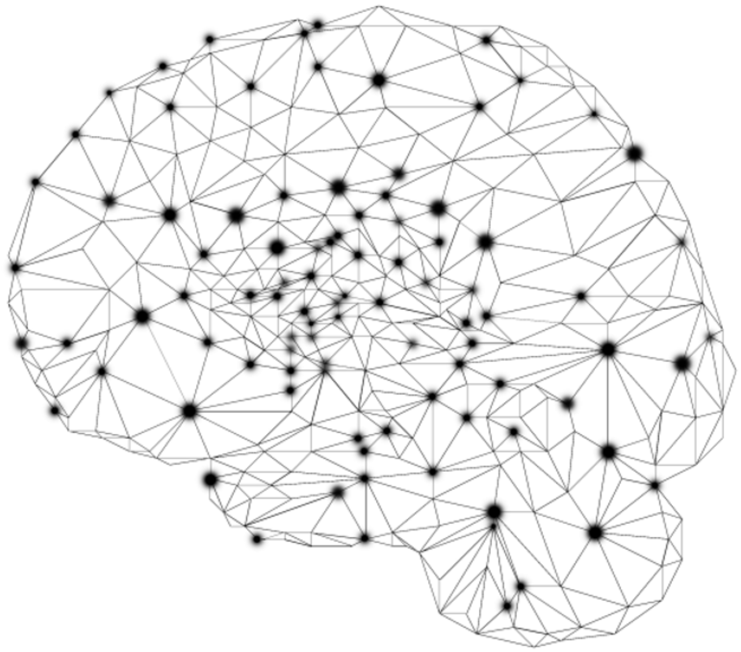
NEURAL NETWORKS

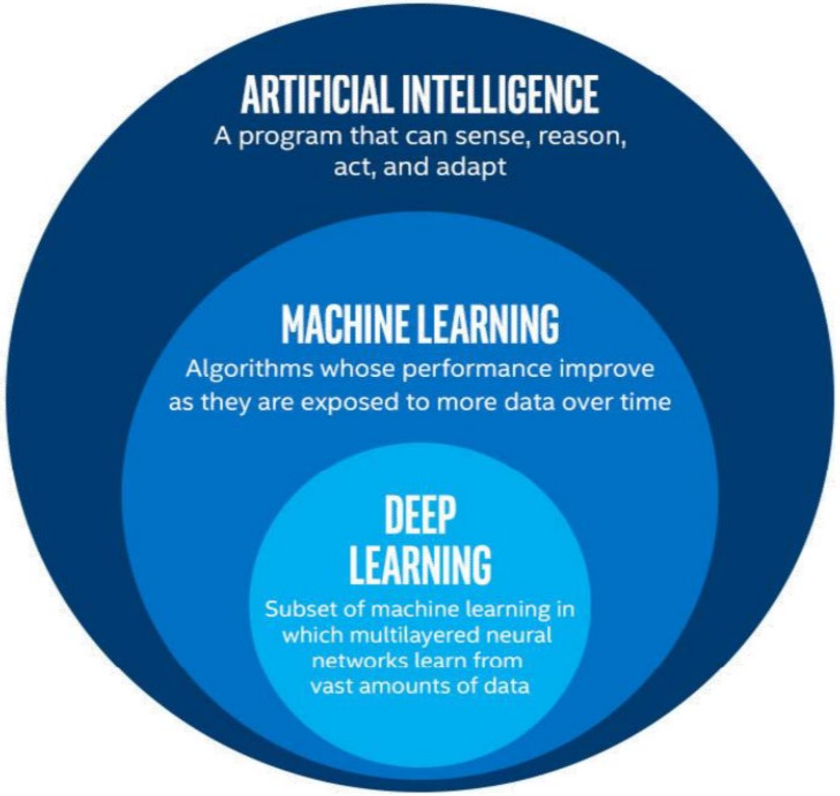
Lecture 6: A Neural Network From Scratch



Marcel Völschow
Hochschule für Angewandte Wissenschaften Hamburg
15.11.2023

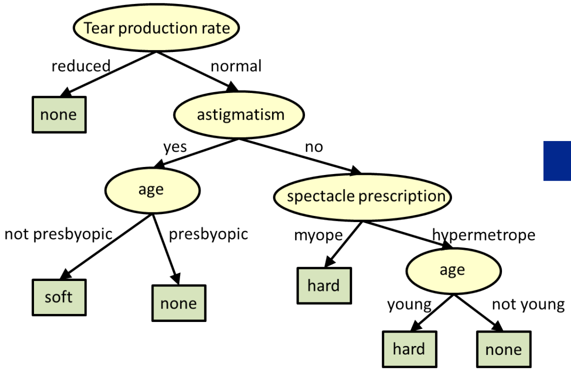
HAW HAMBURG

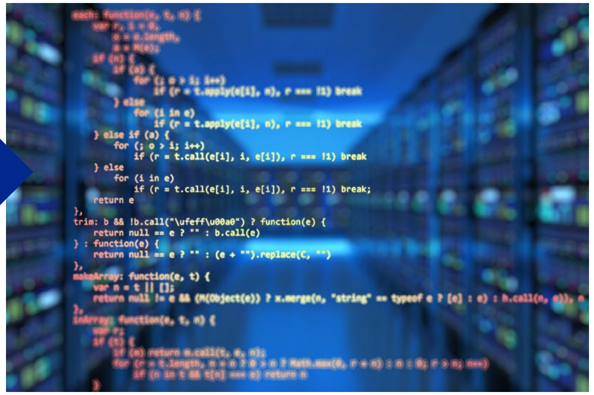
ARTIFICIAL INTELLIGENCE

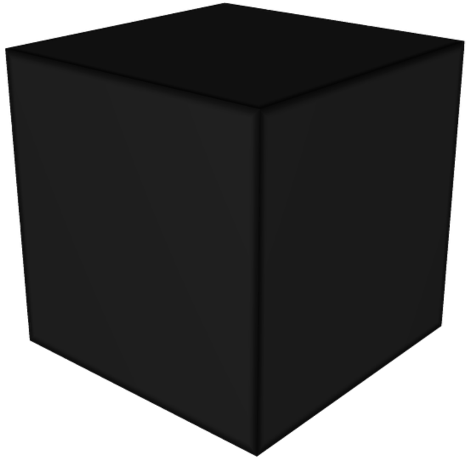


HAW HAMBURG

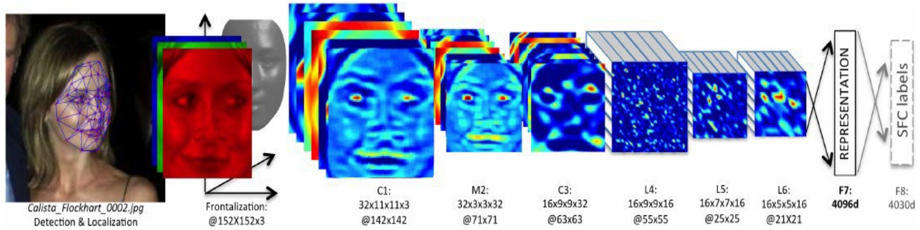
EXPLICIT VERSUS IMPLICIT INTELLIGENCE

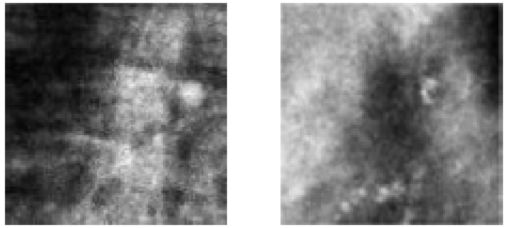


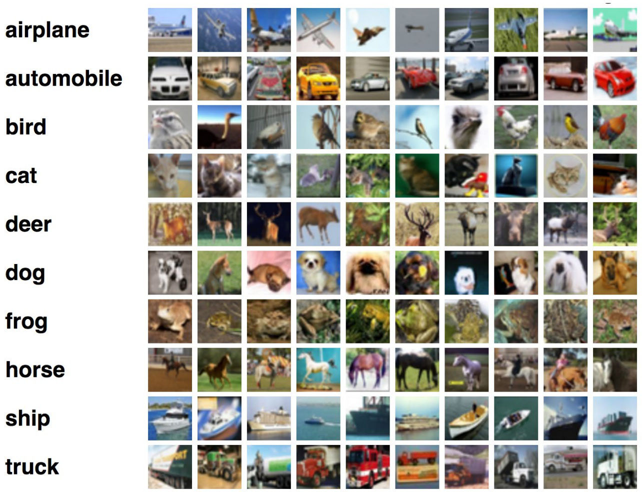


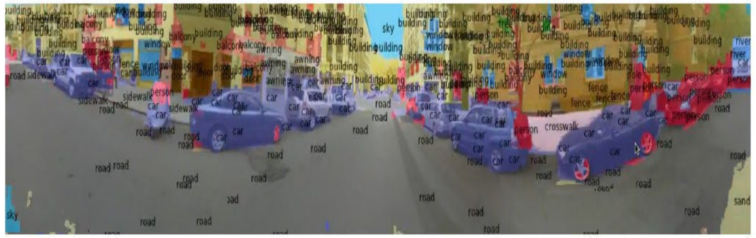


CLASSIFYING STUFF WITH NN

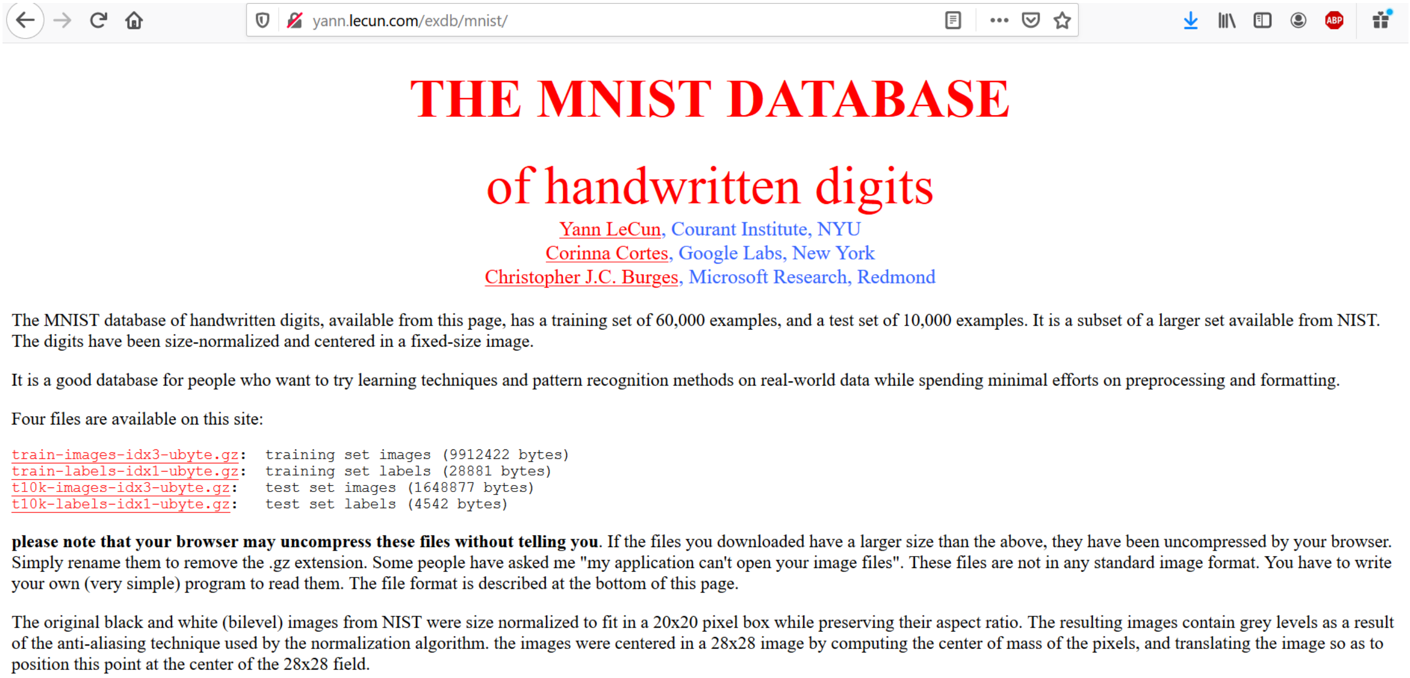








THE MNIST DATASET



yann.lecun.com/exdb/mnist/

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

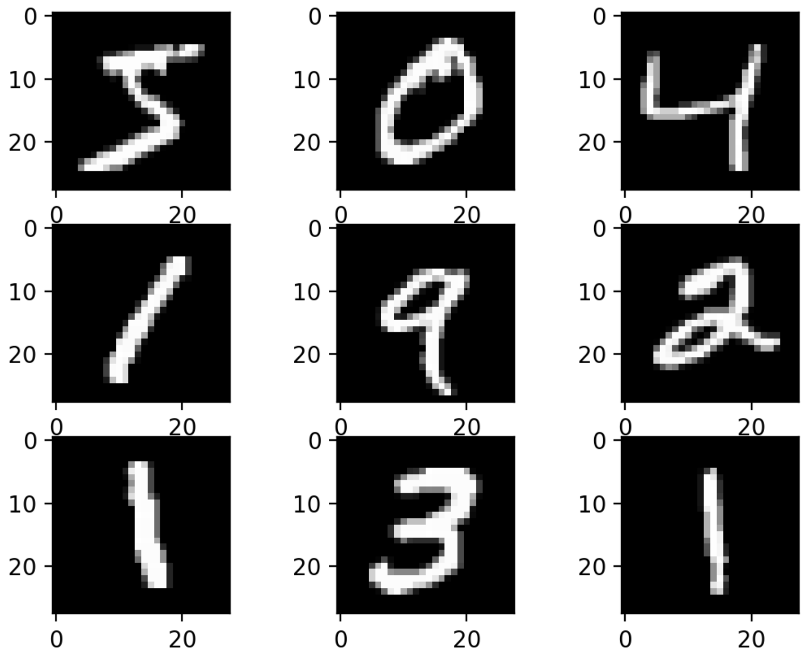
It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

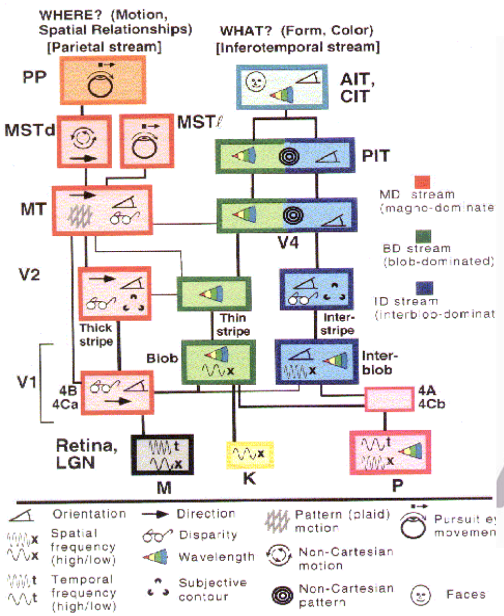
please note that your browser may uncompress these files without telling you. If the files you downloaded have a larger size than the above, they have been uncompressed by your browser. Simply rename them to remove the .gz extension. Some people have asked me "my application can't open your image files". These files are not in any standard image format. You have to write your own (very simple) program to read them. The file format is described at the bottom of this page.

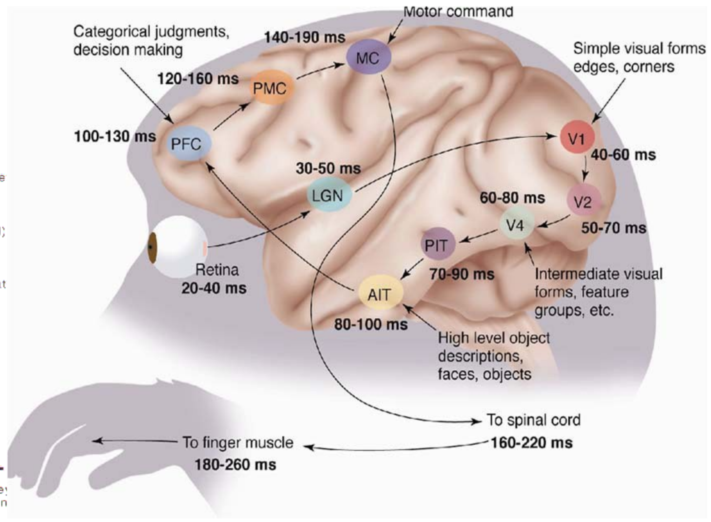
The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.



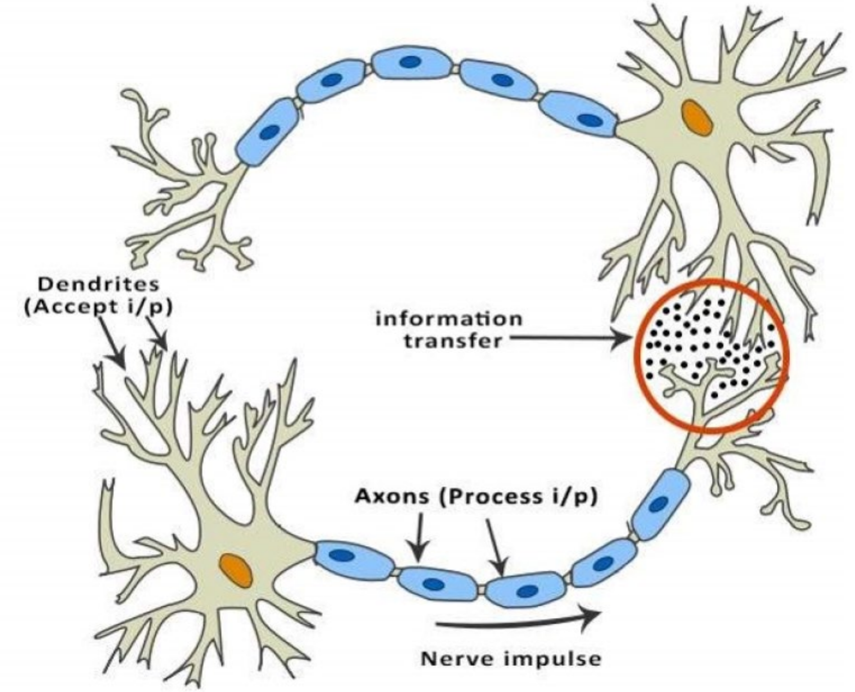
HAW HAMBURG

ORGANIC NEURAL NETWORKS





[Gallant & Van Essen]



1. **Input from m nodes (e.g., „eye pixels“)**
2. **Transmitted to n neurons**
3. **Neuron response is a function of signal strength**
4. **Neuron output represents a decision (e.g., classification)**

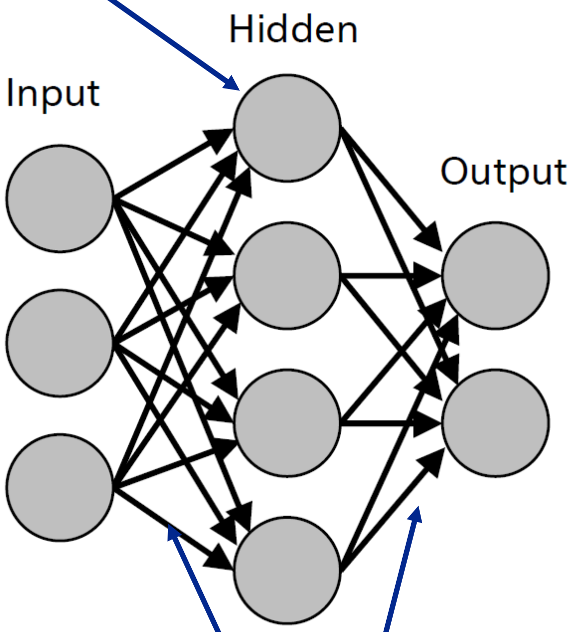
HAW HAMBURG

ARTIFICIAL NEURAL NETWORKS

1. Vector/Matrix with 28x28 entries

28x28

2. N hidden nodes/neurons



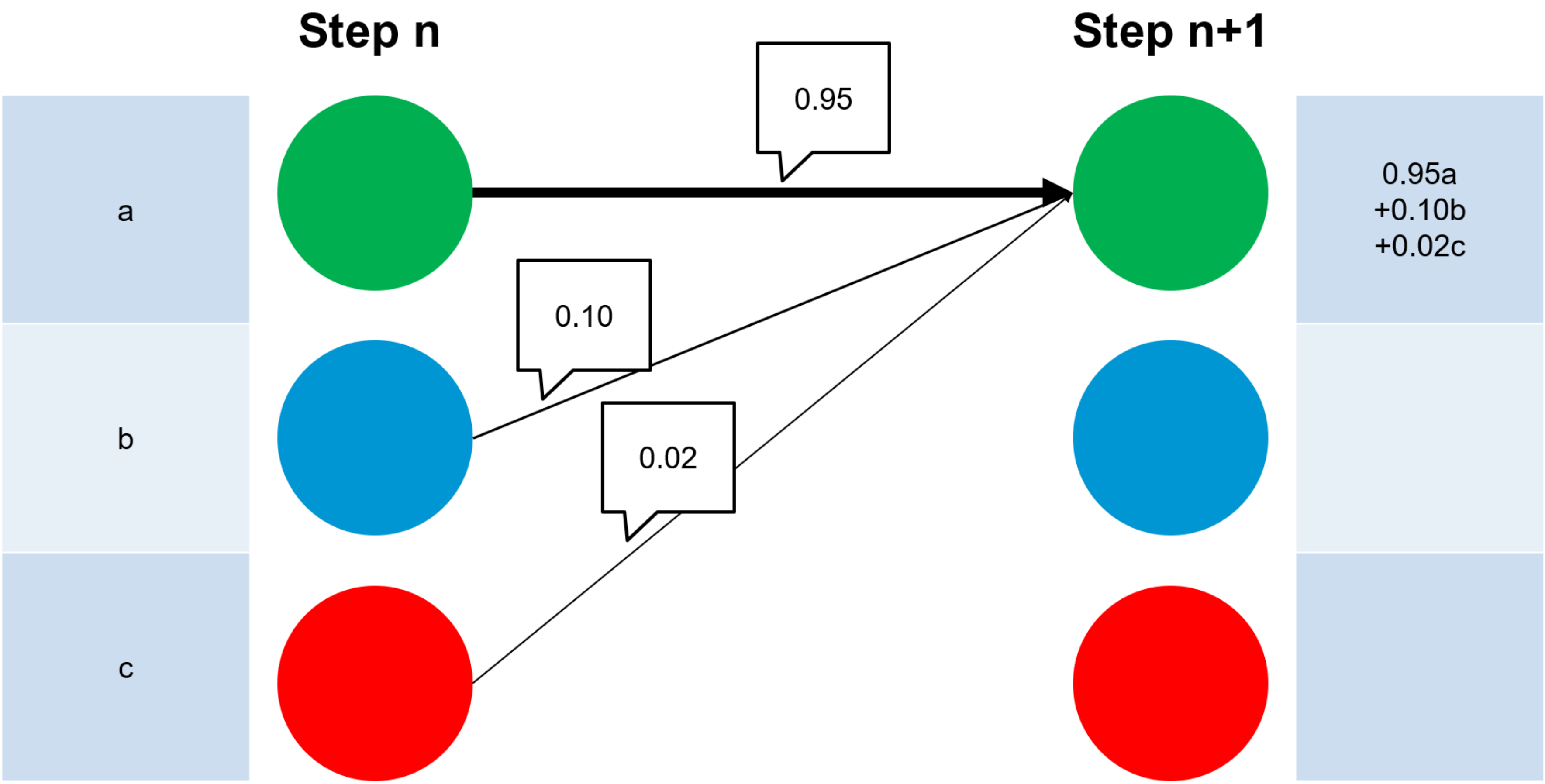
3. Matrices communicate signals between nodes/neurons

Output vector (10x1)

0.0
0.1
0.0
0.3
0.1
0.1
0.0
0.0
0.4
0.0

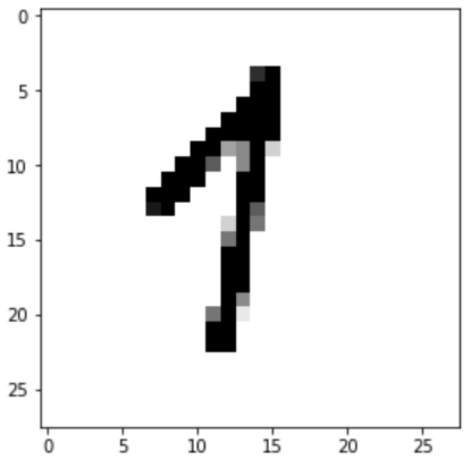
4. Output vector quantifies the result

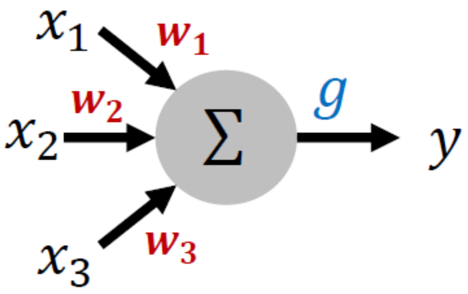
COMMUNICATING SIGNALS BETWEEN NODES



HAW HAMBURG

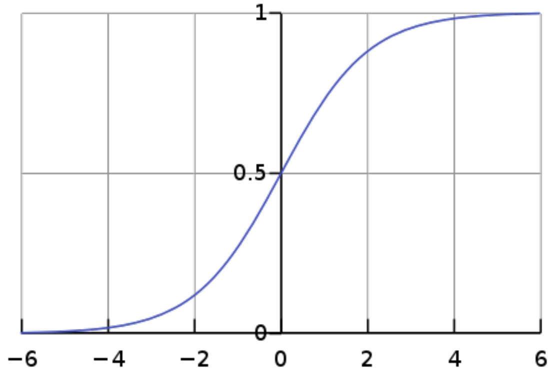
ACTIVATION FUNCTIONS



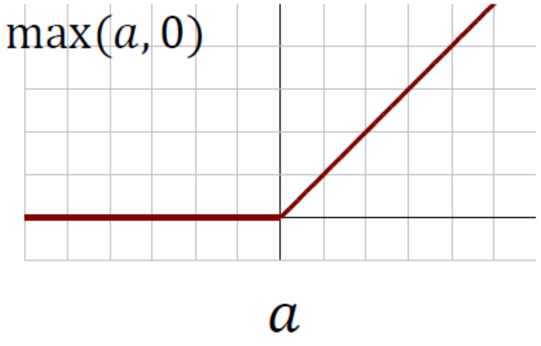


Activation functions g model the response of a neuron

Sigmoid function



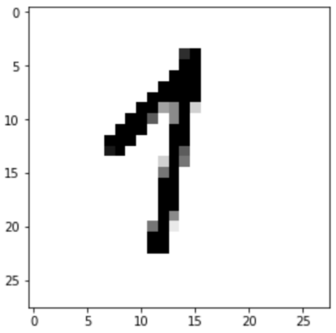
Rectified linear (ReLu)

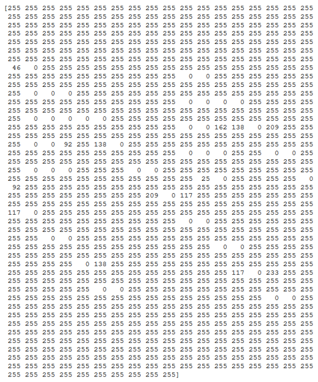


HAW HAMBURG

BASIC NN INGREDIENTS

1. Define input

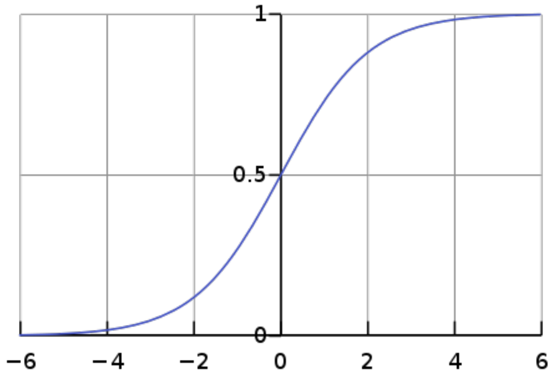




784 input nodes (uint8, 0-255)
10 output nodes (int, 0-9)

2. Set number of hidden nodes/neurons → **100 neurons** → **(100x784) input-hidden (wih)**
(10x100) hidden-output (who)

3. Choose an activation function

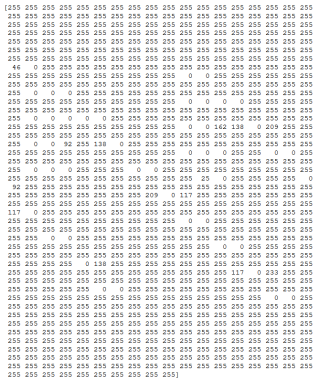


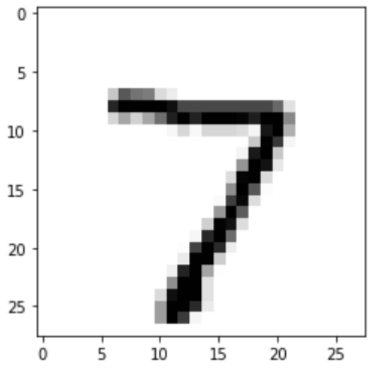

```python
def sigmoid(X):
    return 1/(1+np.exp(-X))
```


HAW HAMBURG

SIGNAL FEED-FORWARD

I. Input layer (784 nodes)





Dot product: wih * input

Apply sigmoid function

II. Hidden layer (100 nodes)

```
[2.41792527e-03 1.10299319e-12 9.58926222e-05 2.41395445e-11
 7.78628311e-05 4.44162273e-20 9.99909568e-01 2.67383949e-12
 1.24303873e-01 5.91268930e-05 1.23644027e-01 6.15685354e-11
 6.33027791e-07 1.22016810e-07 2.98387247e-07 9.45567933e-05
 8.68749020e-05 9.99730834e-01 3.28582482e-11 5.89070143e-05
 5.13603744e-06 3.99916687e-08 3.21972056e-06 1.00000000e+00
 4.15461563e-03 2.69911130e-04 1.62828675e-09 4.39410565e-07
 3.94994661e-04 9.99022659e-01 2.96670603e-06 3.06577608e-06
 2.82313871e-09 9.99533298e-01 2.19969831e-07 1.18015670e-08
 3.58296737e-14 4.35269474e-13 1.47678576e-07 6.95528880e-09
 1.23659513e-12 1.44085848e-04 5.48810614e-06 2.04256443e-10
 1.16948194e-06 2.31804989e-07 2.48182532e-07 2.24920806e-13
 4.84913342e-06 4.30006189e-06 4.13803950e-07 1.11251392e-05
 3.31332897e-08 1.93942672e-01 1.65626312e-09 1.78564511e-08
 3.30632034e-08 3.91382983e-06 2.05142528e-11 2.14213678e-04
 5.44112858e-08 1.20251474e-07 8.05326951e-04 3.84056809e-10
 9.72551232e-01 2.97104772e-11 4.09035111e-13 7.18237324e-11
 6.22399765e-08 2.13573010e-08 6.16334160e-05 6.53225739e-13
 1.45819673e-08 9.99204220e-02 1.38881809e-04 7.86723727e-09
 9.99999834e-01 1.89808632e-15 1.42010176e-02 1.08437044e-04
 8.00671644e-10 1.10047033e-05 3.00049192e-17 3.12103408e-06
 4.93474020e-09 8.88113435e-02 1.02655128e-09 2.73271118e-06
 9.99707382e-01 7.41835912e-14 1.20566212e-06 1.05372275e-07
 2.89354798e-06 9.70455447e-01 2.12703667e-18 3.64019170e-15
 3.38688270e-05 5.79111423e-06 8.22853229e-09 9.99999991e-01]
```

Dot product: who * hidden

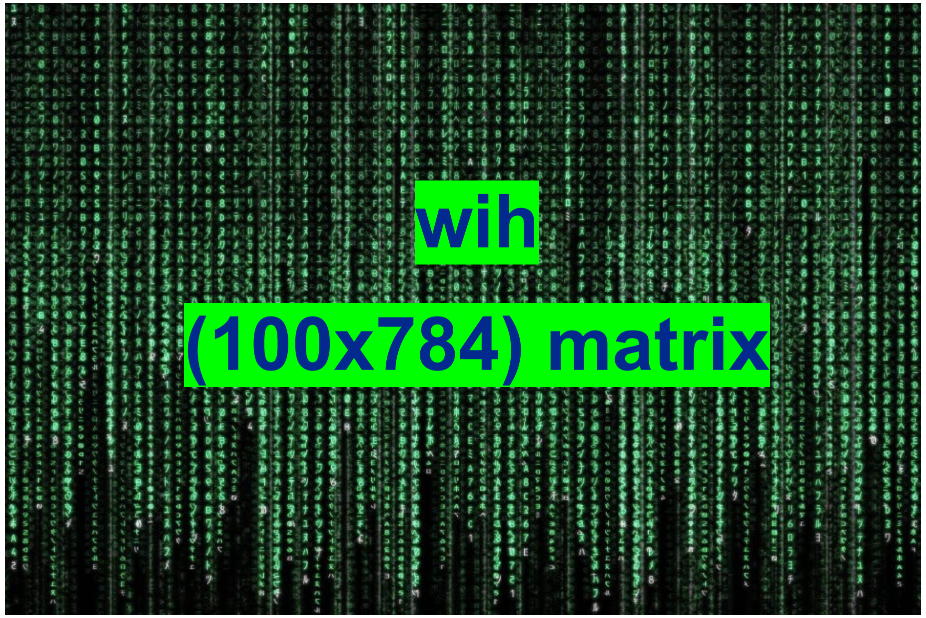
Apply sigmoid function

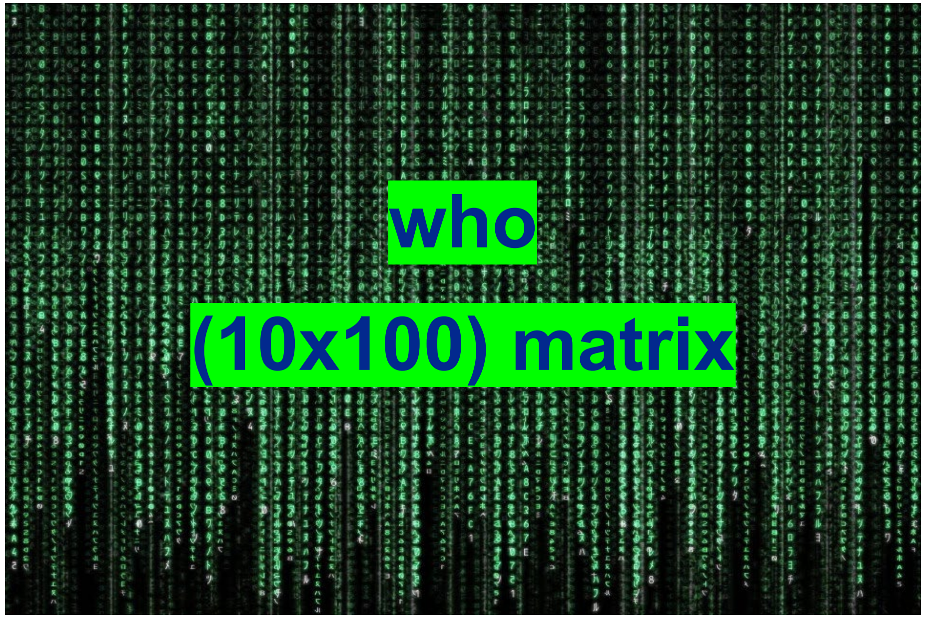
III. Output layer (10 nodes)

0.0114
0.0066
0.0010
0.0050
0.0007
0.0012
0.0004
0.9941
0.0070
0.0002

WHAT’S THE CATCH?

We need to find all entries of the weight matrices which represent the NNs memory:





That‘s a total of 78400 + 1000 = 79400 unknown parameters …

=> Backpropagation (next week…)

IF YOU WANNA LEARN MORE …