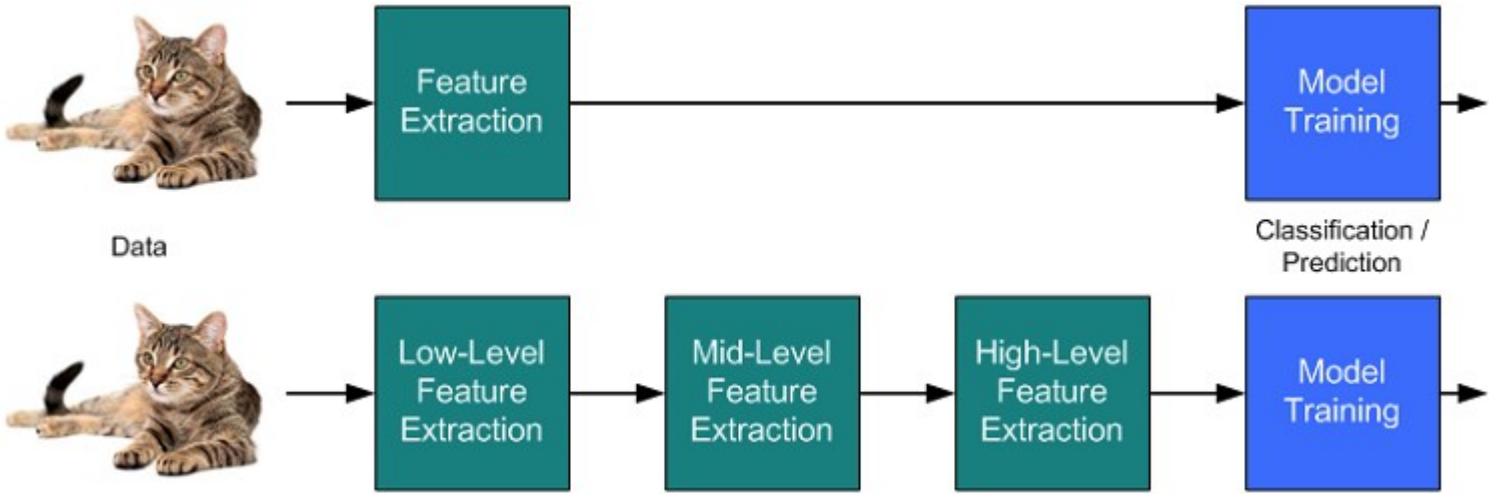
Machine learning

Lecture 5



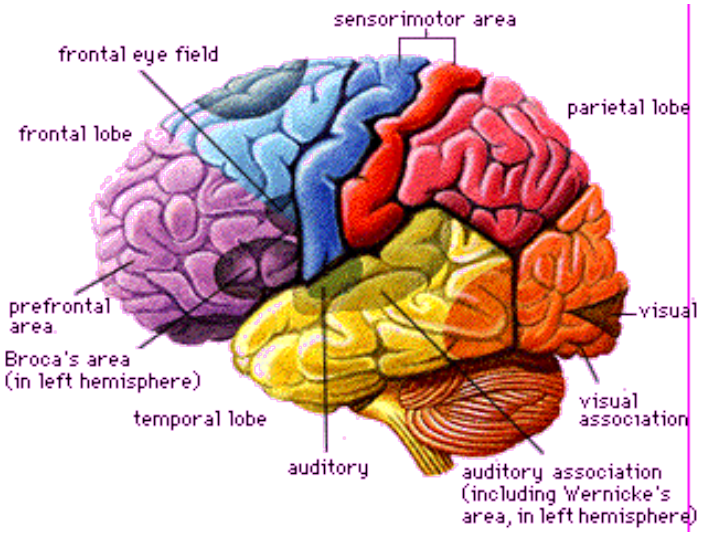
- Radial Base Function (RBF) Networks
- Bayesian Neural Networks
- Deep Neural Networks

Marcin Wolter

IFJ PAN

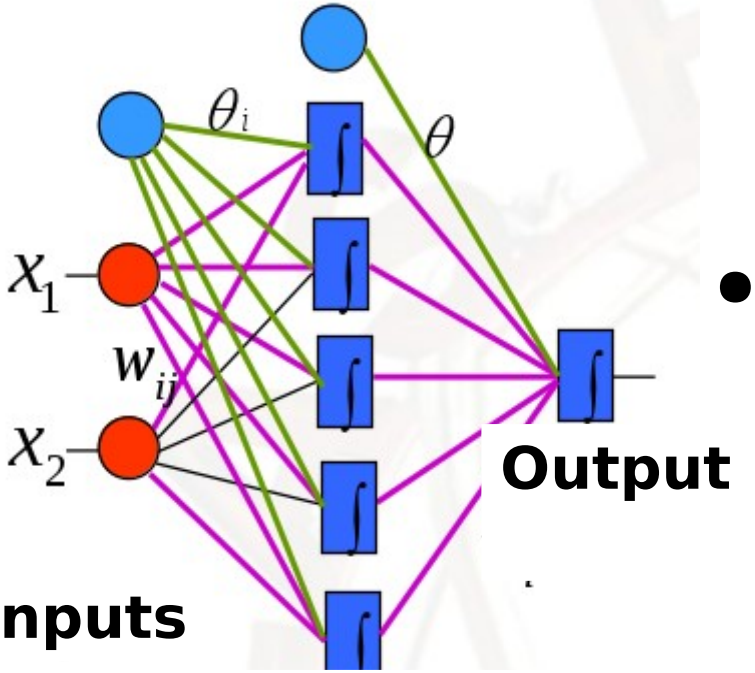
10 January 2020

Inspired by human brain



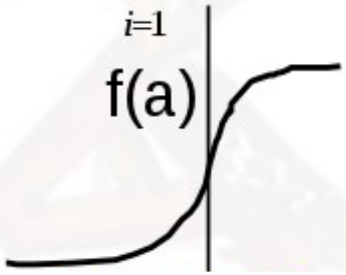
- **Human brain:**
  - 10^{14} neurons, frequency 100 Hz
  - Parallel processing of data (complex pattern recognition in 100 ms – 10 steps only!!!)
  - Learns on examples
  - Resistant for errors and damaged neurons
- **Neural Network:**
  - Just an algorithm, which might not reflect the way the brain is working.

Neural networks



$$a_i = \sum_{j=1}^{2} w_{ij} x_j + \theta_i \quad \rightarrow f(a_i)$$

$$n(x,w) = f\left(\sum_{i=1}^{5} w_i f(a_i) + \theta\right)$$

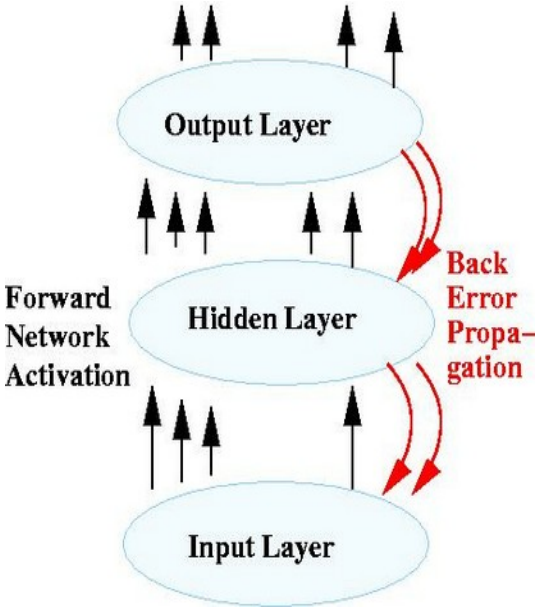


Activation function

- At the input of each node a weighted sum of inputs is given. It is transformed by the activation function (typically sigmoid) and later send to the output.

  How to train the multilayer network? How to tune the weights in the hidden layers? This problem was unsolved for a long time...

- Solution – **backpropagation**. An error y-*f(x,w)* is propagated backward through the net using the actual weights („revolution" of '80'ies).



Network regularization

- Both MLPRegressor and MLPClassifier use parameter alpha for regularization (L2 regularization) term which helps in avoiding overfitting by penalizing weights with large magnitudes.

- Example with different regularizations:

  https://github.com/marcinwolter/MachineLearnin2019/blob/master/simple_classifier_comparison.ipynb

Neural network examples

- Simple comparison of many classifiers
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/simple_classifier_comparison.ipynb

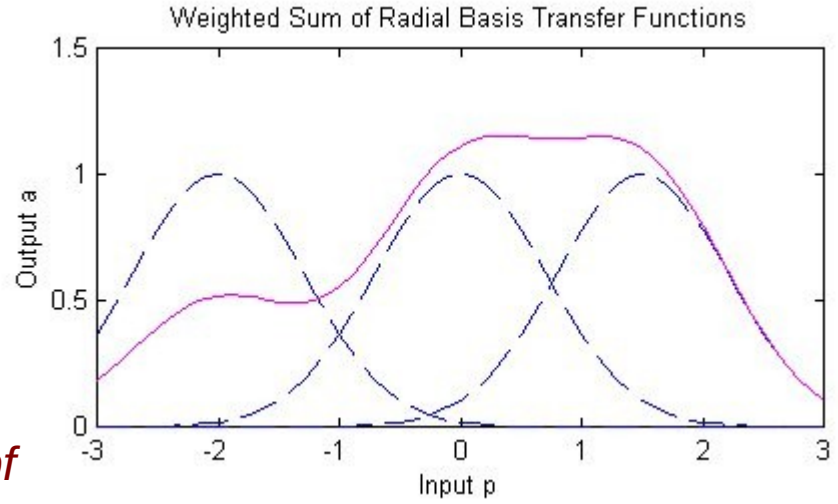
- Neural network for hand-written digits classification
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/plot_digits_classif_mlp.ipynb

- Visualization of MLP weights
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/plot_mnist_filters.ipynb

Radial Base Functions (RBF)

- A neural network that uses radial basis functions as activation functions. The output of the network is a linear combination of radial basis functions of the inputs and neuron parameters. Formulated in a 1988 paper by Broomhead and Lowe.
- Neuron in a hidden layer – the radial function, which is non-zero around the center c only:

  $f_i(x) = f_i(||x - c||)$ - a radial base function.



Applet rbf

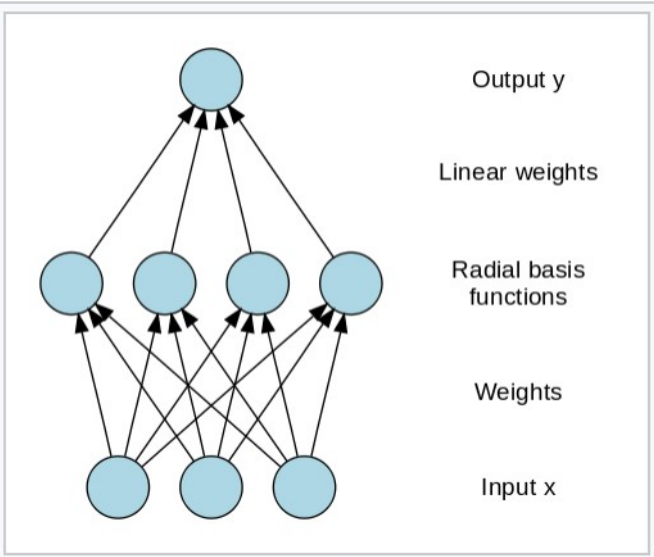


Figure 1: Architecture of a radial basis function network. An input vector x is used as input to all radial basis functions, each with different parameters. The output of the network is a linear combination of the outputs from radial basis functions.

RBF functions

1. **Gaussian Functions:**

$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \text{width parameter } \sigma > 0$$

2. **Multi-Quadric Functions:**

$$\phi(r) = \left(r^2 + \sigma^2\right)^{1/2} \qquad \text{parameter } \sigma > 0$$

3. **Generalized Multi-Quadric Functions:**

$$\phi(r) = \left(r^2 + \sigma^2\right)^{\beta} \qquad \text{parameters } \sigma > 0,\ 1 > \beta > 0$$

4. **Inverse Multi-Quadric Functions:**

$$\phi(r) = \left(r^2 + \sigma^2\right)^{-1/2} \qquad \text{parameter } \sigma > 0$$

5. **Generalized Inverse Multi-Quadric Functions:**

$$\phi(r) = \left(r^2 + \sigma^2\right)^{-\alpha} \qquad \text{parameters } \sigma > 0,\ \alpha > 0$$

6. **Thin Plate Spline Function:**

$$\phi(r) = r^2 \ln(r)$$

7. **Cubic Function:**

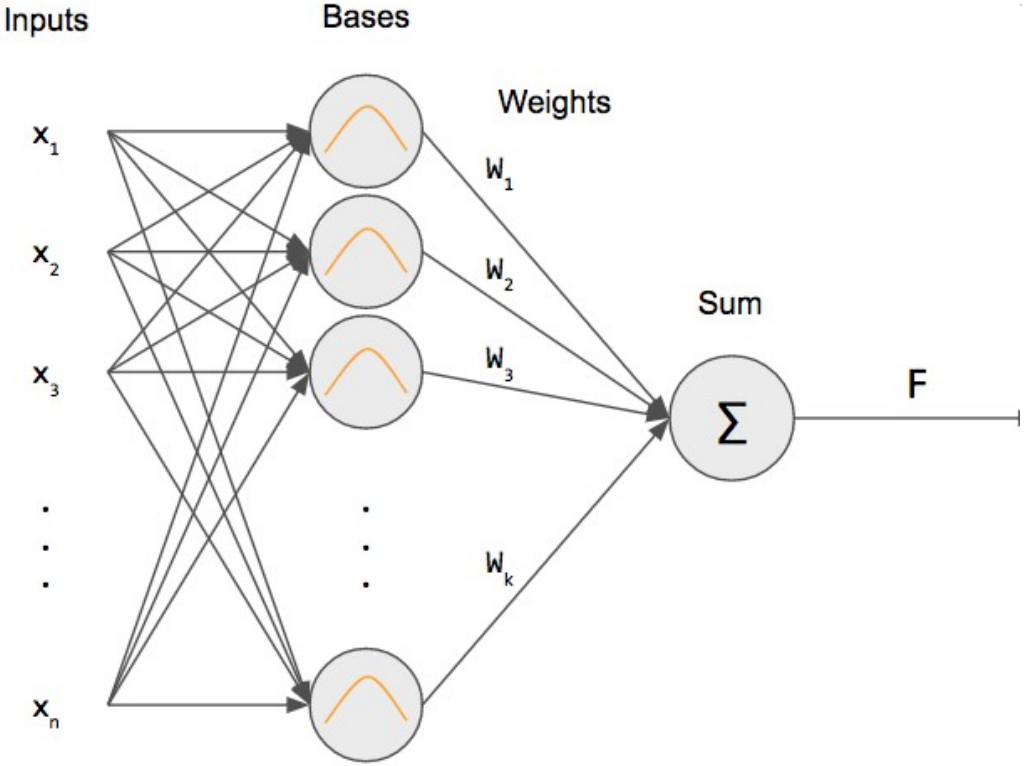
$$\phi(r) = r^3$$

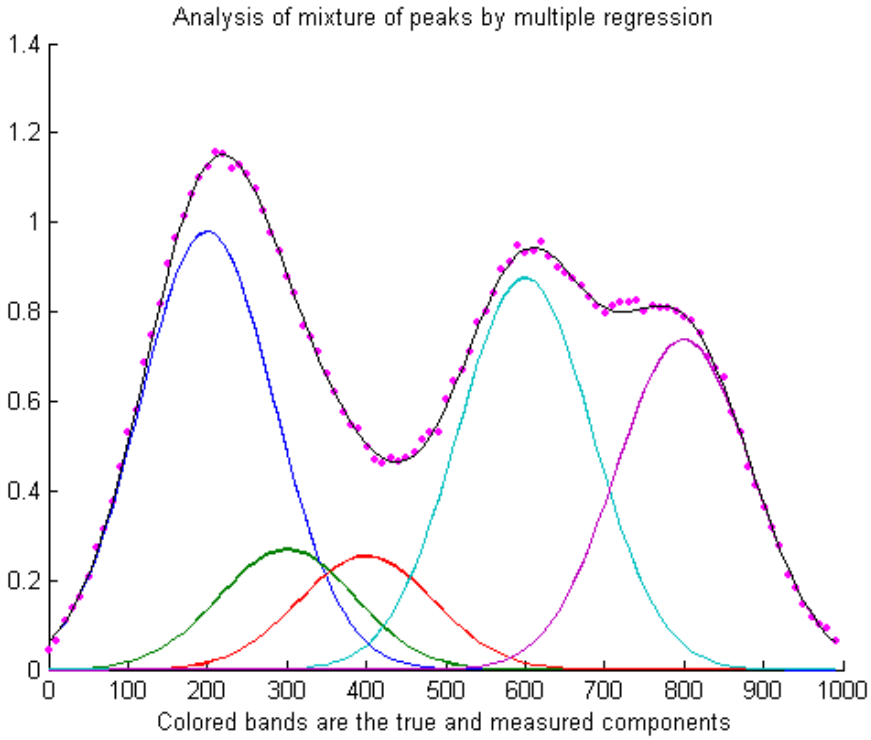
8. **Linear Function:**

$$\phi(r) = r$$

Example of RBF network
RBF for REGRESSION

- Radial Basis Function Networks (RBF nets) are used for exactly this scenario: regression or function approximation. We have some data that represents an underlying trend or function and want to model it. RBF nets can learn to approximate the underlying trend using many Gaussians/bell curves.
- We have an input that is fully connected to a hidden layer. Then, we take the output of the hidden layer perform a weighted sum to get our output.



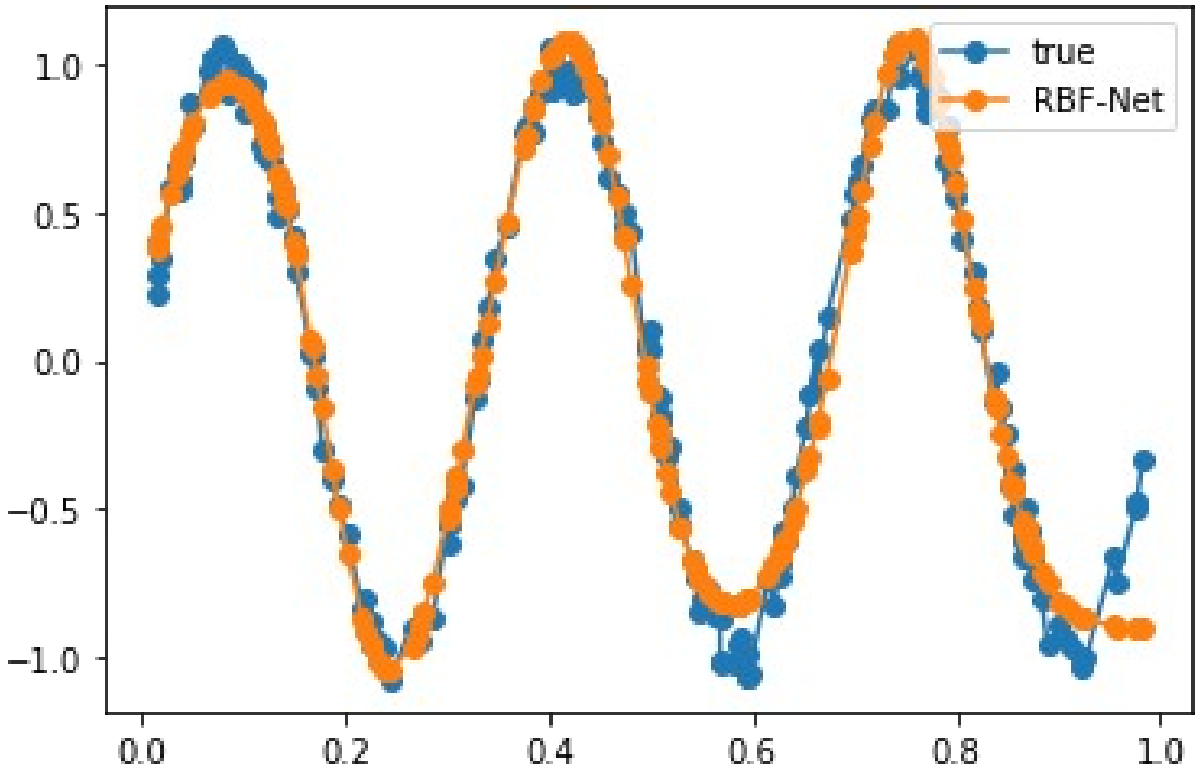


RBF network example

- We need to know where to place the Gaussian centers and find their standard deviations.
- We use *k-means* clustering on our input data to figure out where to place the Gaussians. The reasoning behind this is that we want our Gaussians to "span" the largest clusters of data since they have that bell-curve shape.
- What the standard deviations should be?:
  - set the fixed standard deviation
  - set it to that of the points assigned to a particular cluster
  - use a single standard deviation for all clusters $\sigma = \frac{d_{max}}{\sqrt{2k}}$ where d_{max} is the maximum distance between any two cluster centers
- Set k - the number of cluster centers.

RBF example

- Run the code:
  - https://github.com/marcinwolter/MachineLearnin2019/blob/master/rbfnet.ipynb
- Play with the parameters



Machine learning vs Bayesian learning

Machine learning

Teaching the function $y = f(x)$ giving **training data** $\boldsymbol{T} = (\boldsymbol{x}, \boldsymbol{y}) = (x,y)_1, (x,y)_2, \ldots (x,y)_N$ and **bonds** by giving a class of this function.

Bayesian learning

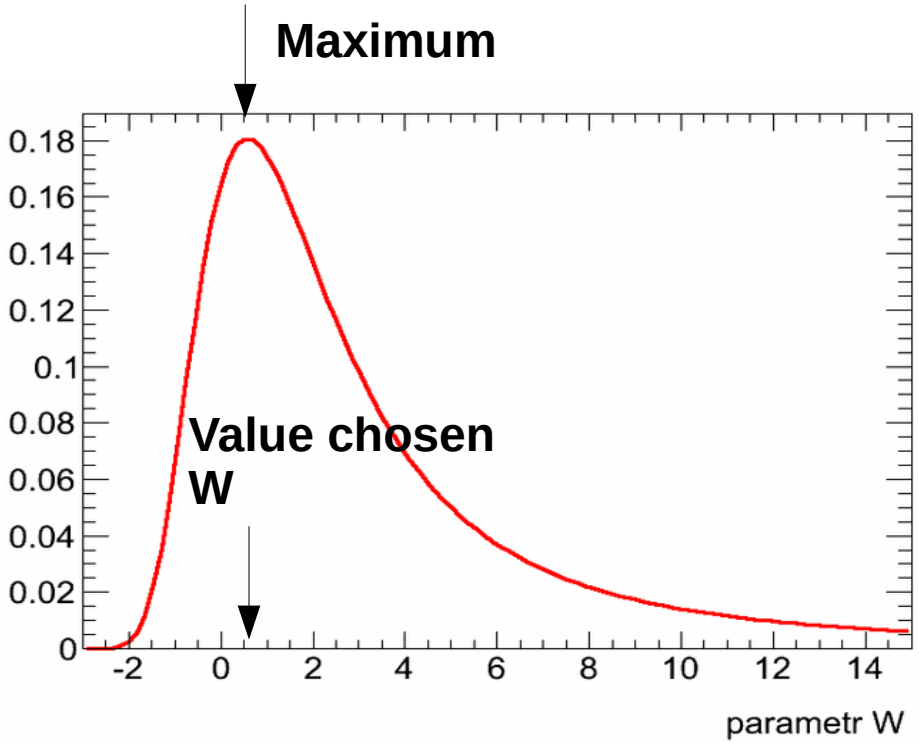
For each function f(x) from the function space F we find an *a posteriori* probability $\boldsymbol{p(f \mid T)}$ using the training sample **T= (*x*, *y*).**

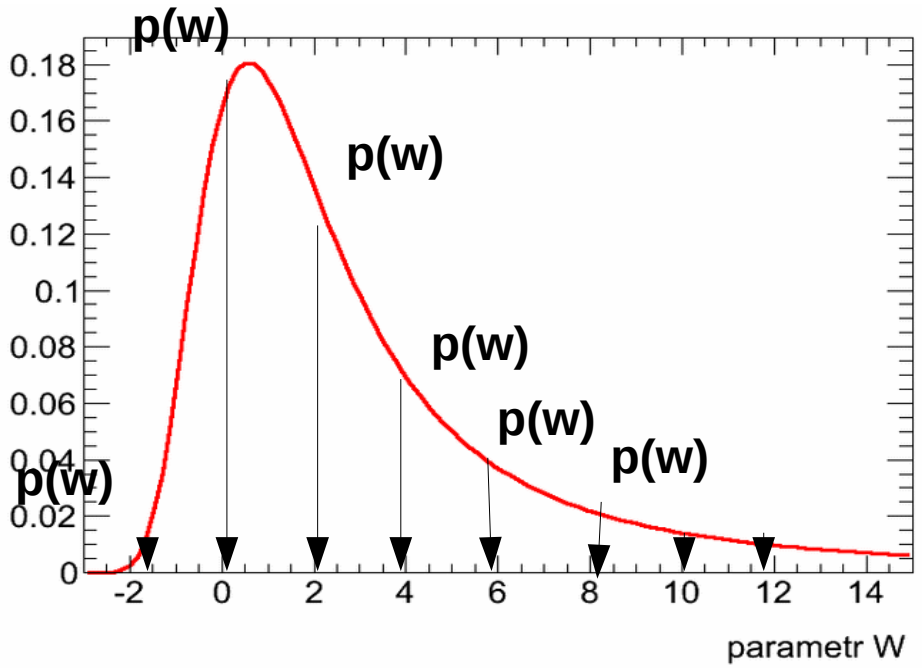
In the bayesian learning we DO NOT find the one, best function, but we use many functions weighted by their probability.

Probability *a posteriori* – probability calculated using the results of the experiment.

Training sample $\boldsymbol{T} = (\boldsymbol{x}, \boldsymbol{y})$: a set of input vectors ***x*** and results ***y.***

Machine and bayesian learning





Machine learning

We chose one function (or a value of a parameter describing the function).

Bayesian learning

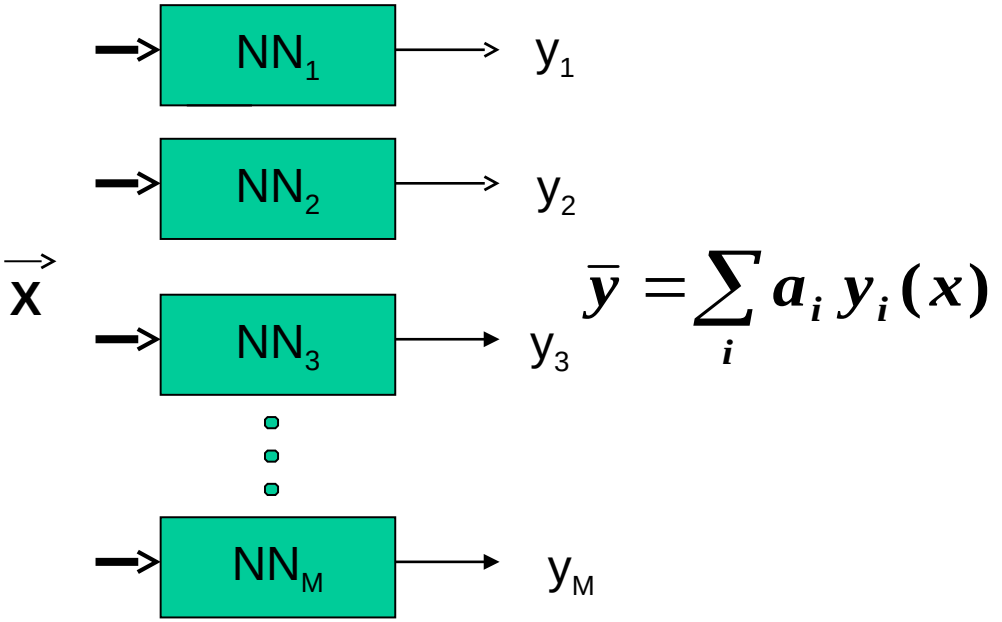
Each function (or a parameter value) is given some probability (weight).

Implementation of bayesian networks:

Instead of choosing a single set of weights describing the NN we should find the probability density for the entire space of weights.

Many neural networks.

Having many NN we can get the weighted mean or he most probable network and also the estimation error



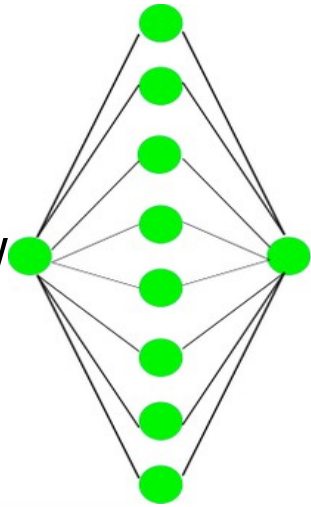
C.M. Bishop
"Neural Networks for Pattern Recognition",
Oxford 1995

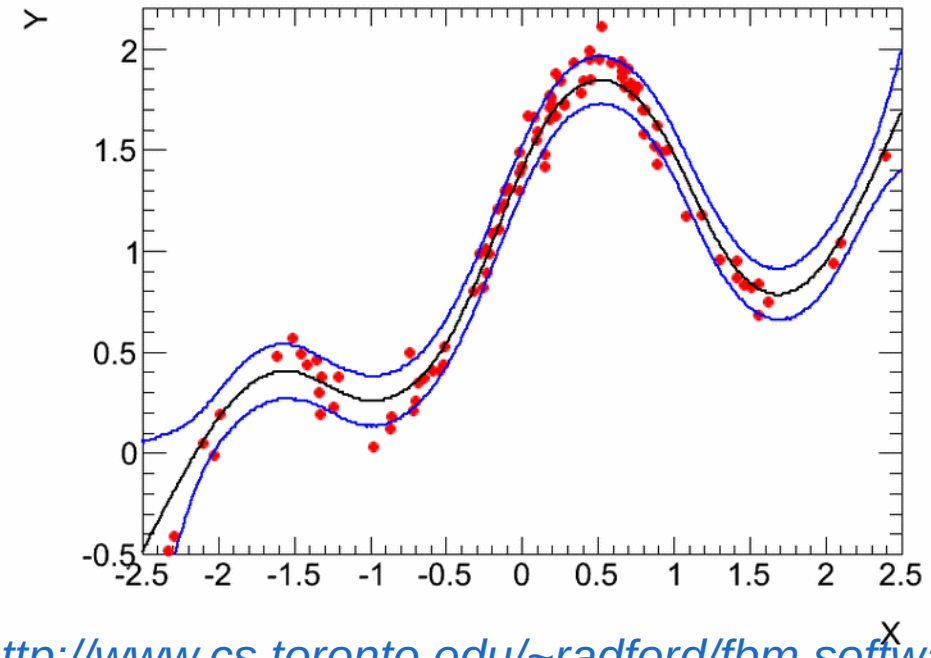
Free software:
Radford Neal, http://www.cs.toronto.edu/~radford/fbm.software.html

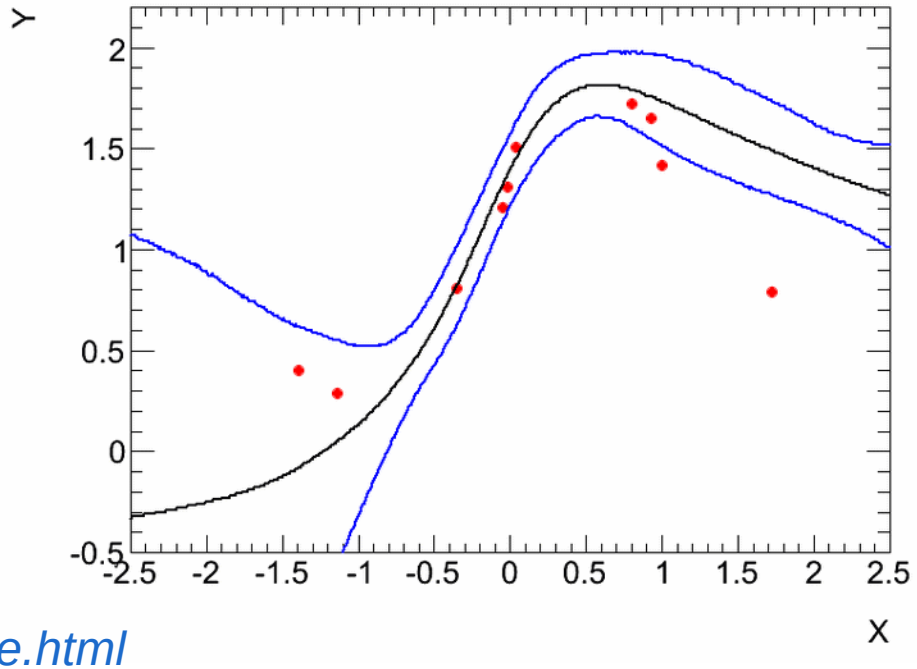
Example of BNN

How does the bayesian network with 8 neurons works for different amount of data?

- Data generated using function: $y = 0.3 + 0.4x + 0.5\sin(2.7x) + 1.1/(1+x^2)$ with a gaussian noise (sigma = 0.1)
- 400 neural networks, from the distribution of answers we get: median and 10% Qnt i 90% Qnt (10% of networks gave answers below lower blue line and 10% above the upper line).
- When only 10 training points used, we got much bigger errors (as they should be).







http://www.cs.toronto.edu/~radford/fbm.software.html

BNN example

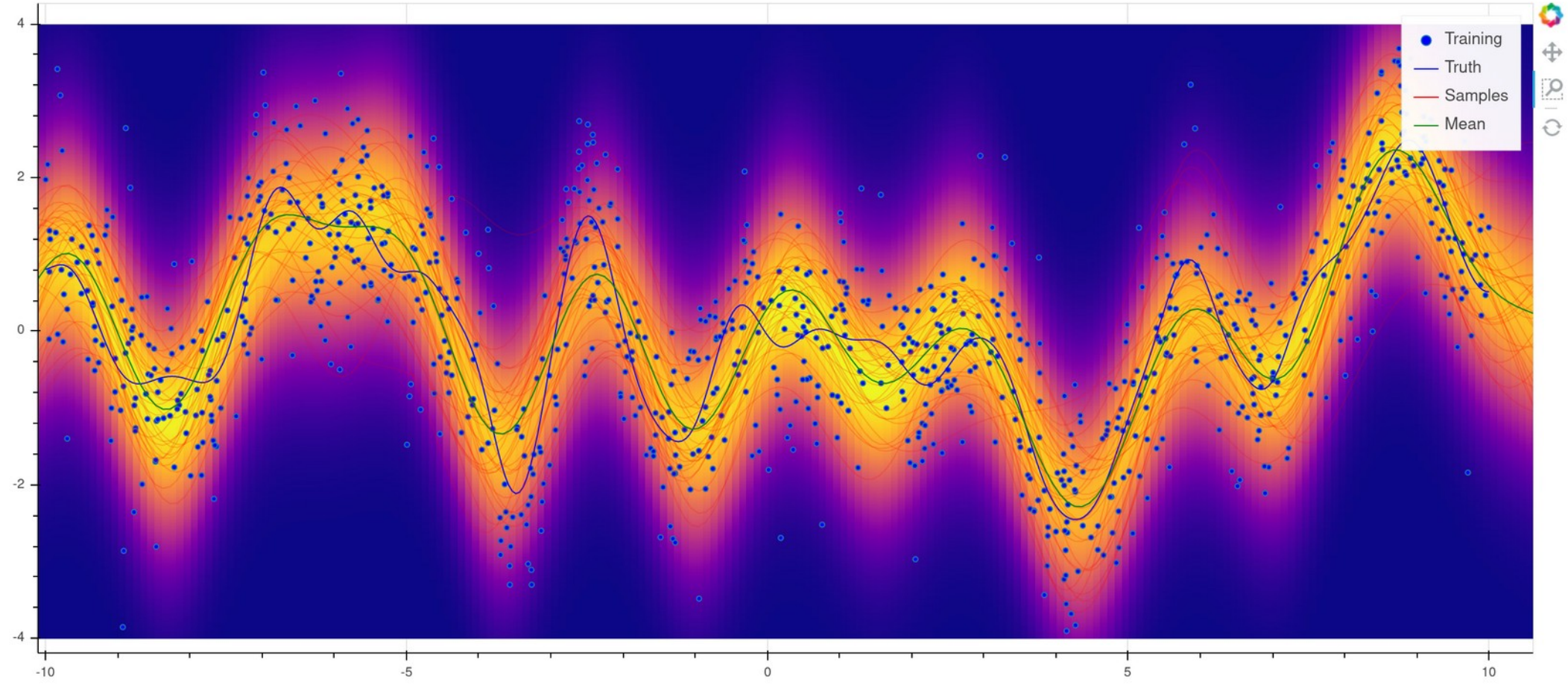
- MNIST hand written digits recognition with torch package:

  https://towardsdatascience.com/making-your-neural-network-say-i-dont-know-bayesian-nns-using-pyro-and-pytorch-b1c24e6ab8cd

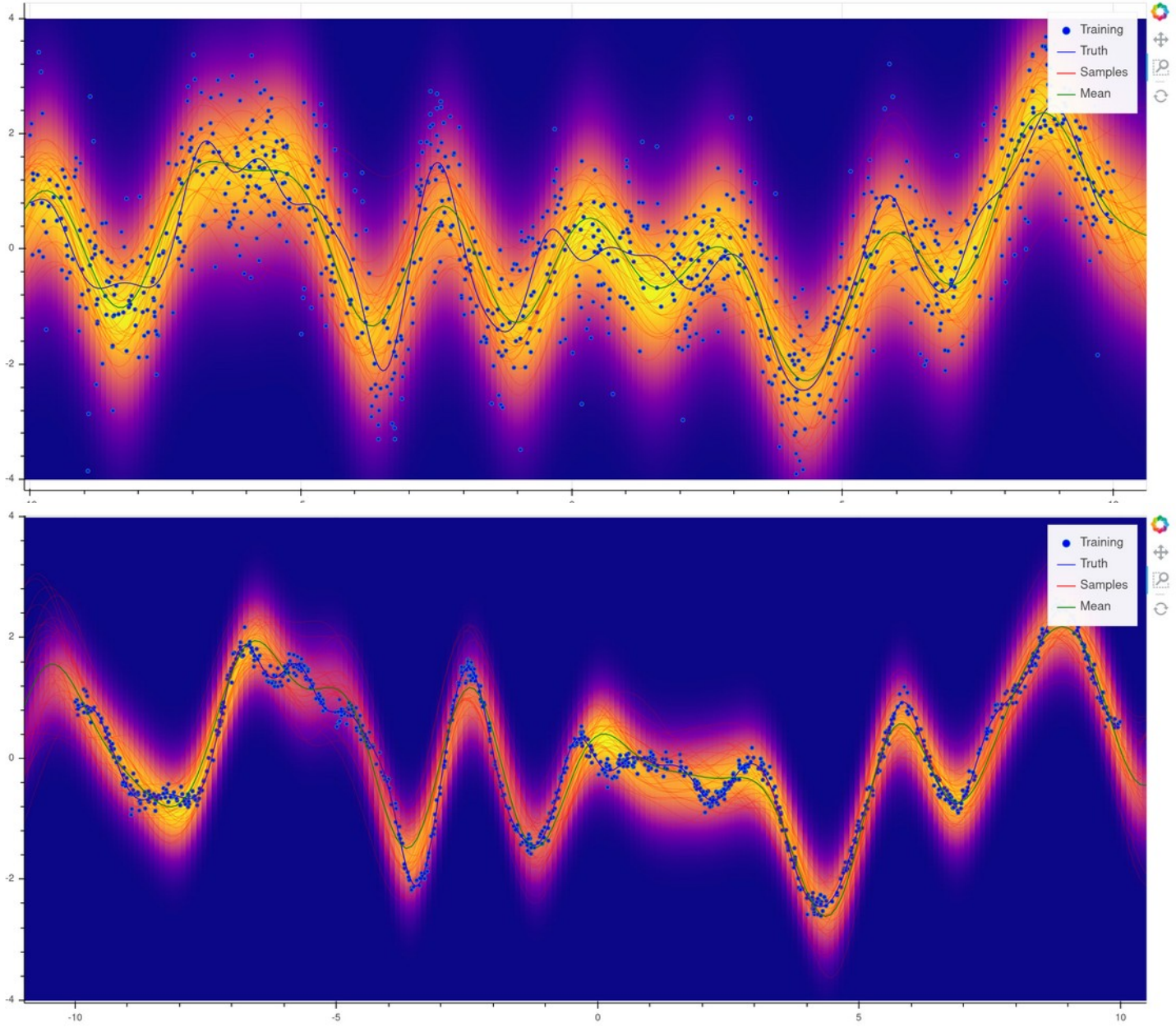
- Essentially, what we're doing is this:
  - For an input image, take 100 samples of neural networks to get 100 different output values from the last layer
  - Convert those outputs (which are logsoftmaxed) into probabilities by exponentiating them
  - Now, given the input image, for each digit we have 100 probability values
  - We take median (50th percentile) of these 100 probability values as the threshold probability for each digit
  - If the threshold probability is greater than 0.2, we select the digit as a classification output from the network
- **NETWORK REFUSES TO CLASSIFY SOME PSEUDO "DIGITS"!!!**
- https://github.com/marcinwolter/MachineLearnin2019/blob/master/bnn.ipynb

Regression BNN example

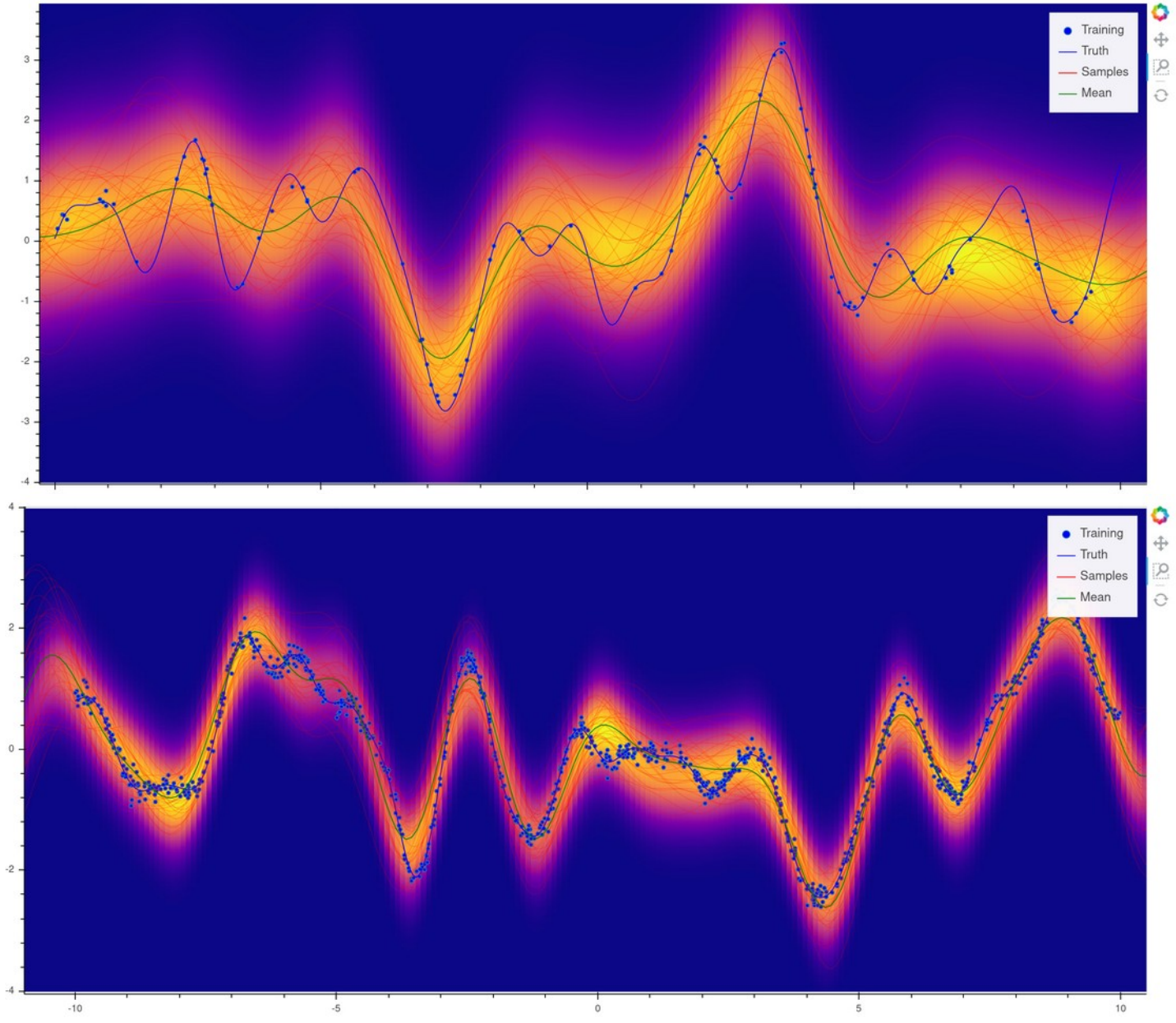
- Fit using Neural Network – example codes from https://github.com/gradientinstitute/aboleth/tree/master/demos
- Fit to the points drawn from the true function (blue) https://github.com/marcinwolter/MachineLearnin2019/blob/master/regression



Different noise



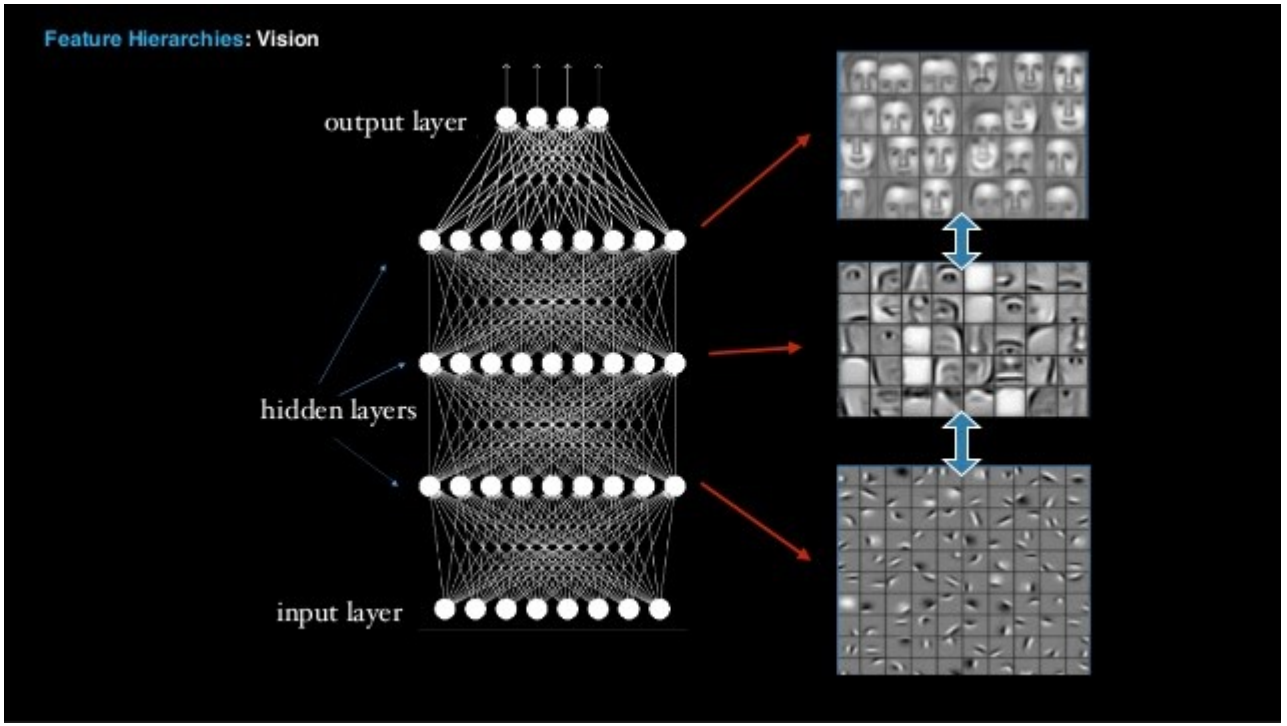
Different number of data points



Bayesian Neural Network Revival (some recent papers)

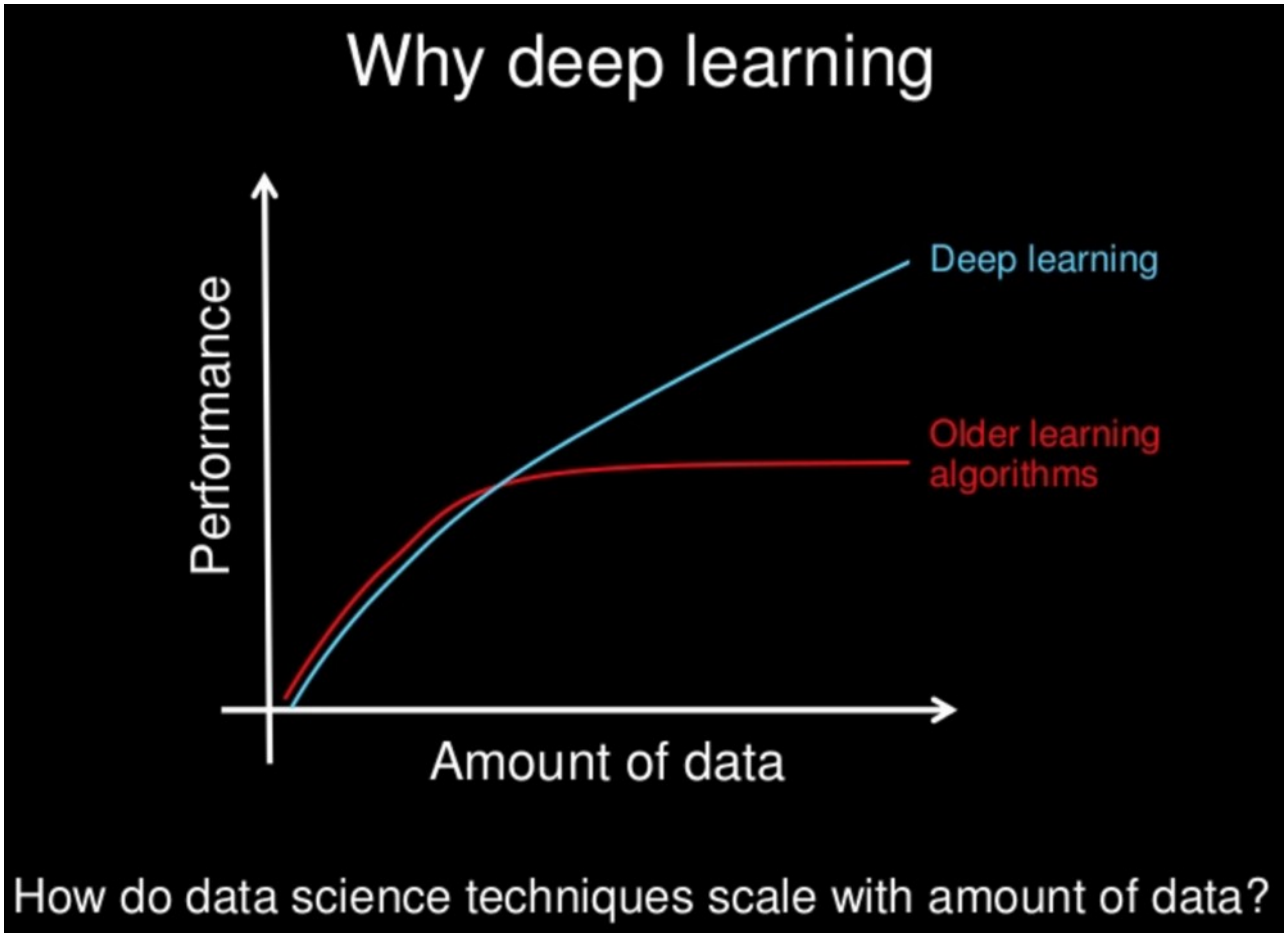
- A. Honkela and H. Valpola. Variational learning and bits-back coding: An information-theoretic view to Bayesian learning. IEEE Transactions on Neural Networks, 15:800-810, 2004.
- Alex Graves. Practical variational inference for neural networks. In NIPS 2011.
- Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In ICML, 2015.
- José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In ICML, 2015.
- José Miguel Hernández-Lobato, Yingzhen Li, Daniel Hernández-Lobato, Thang Bui, and Richard E Turner. Black-box alpha divergence minimization. In Proceedings of The 33rd International Conference on Machine Learning, pages 1511-1520, 2016.
- Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. ICML, 2016.
- Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. NIPS, 2016.

Deep Learning



Deep Learning

- What does “deep learning” mean?
- Why does it give better results than other methods in pattern recognition, speech recognition and others?



Short answer:

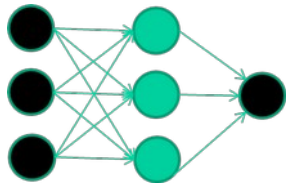
‘Deep Learning’ - using a neural network with many hidden layers

A series of hidden layers makes the feature identification first and processes them in the chain of operations: feature identification → further feature identification → ……. → selection

But NN are well known starting from 80-ties???

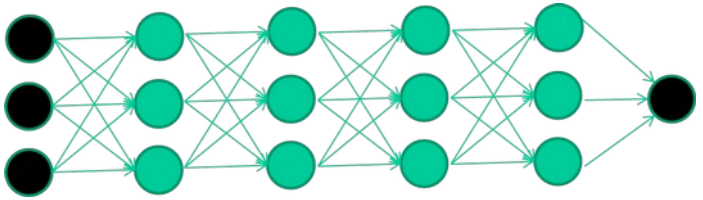
We always had good algorithms to train NN with one or two hidden layers.

But they were failing for more layers

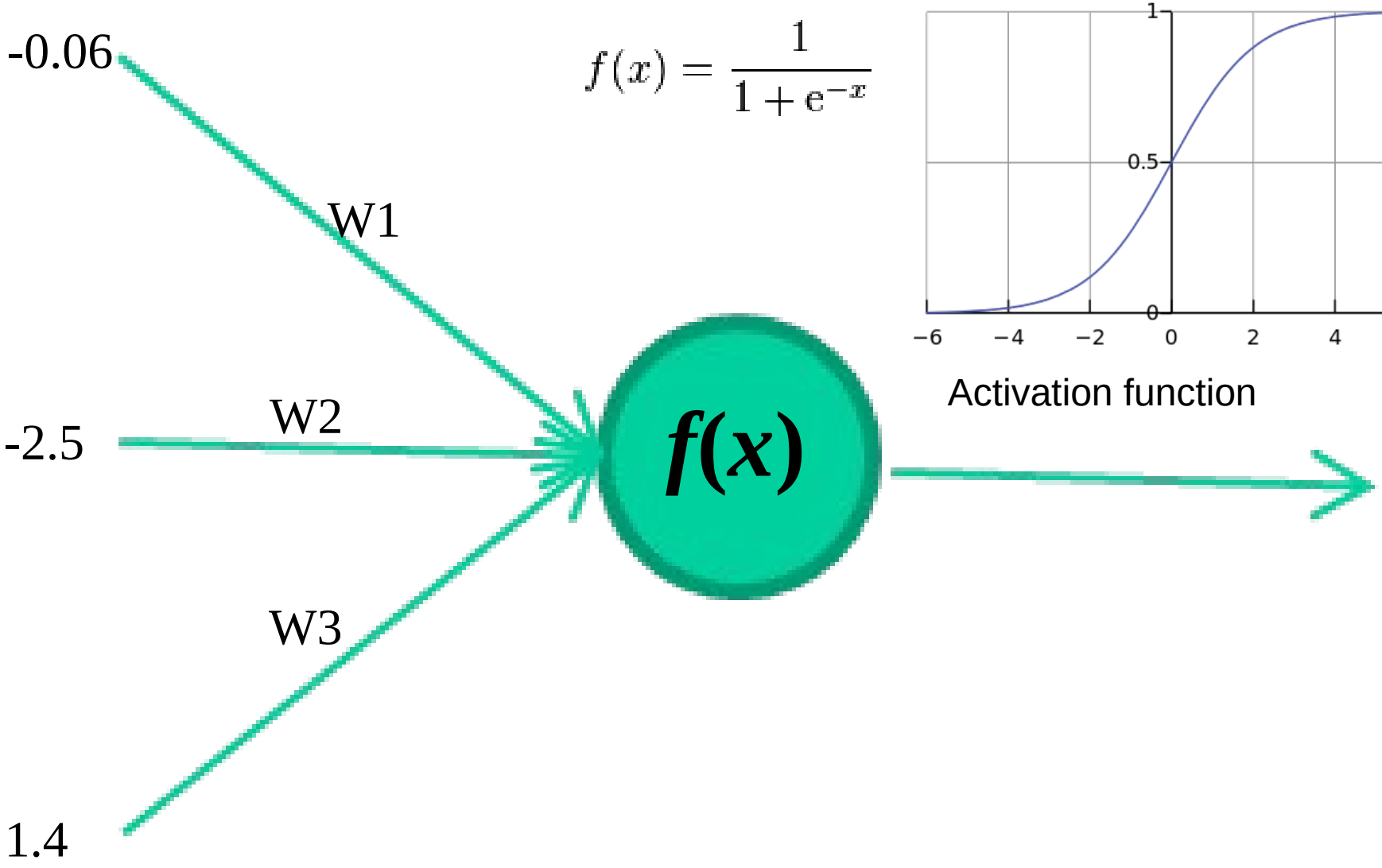


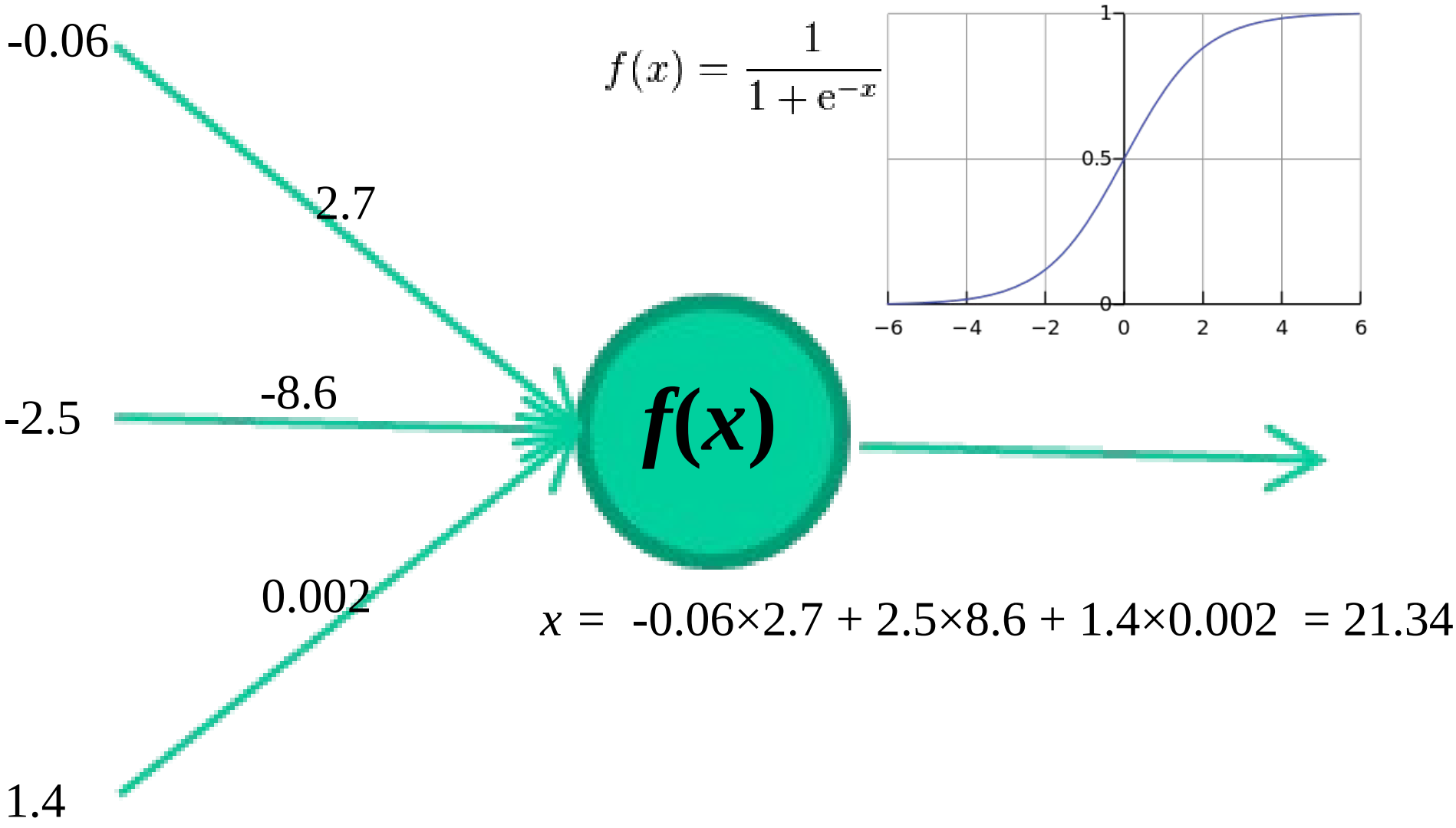
NEW: algorithms for training deep networks

huge computing power



How do we train a NN

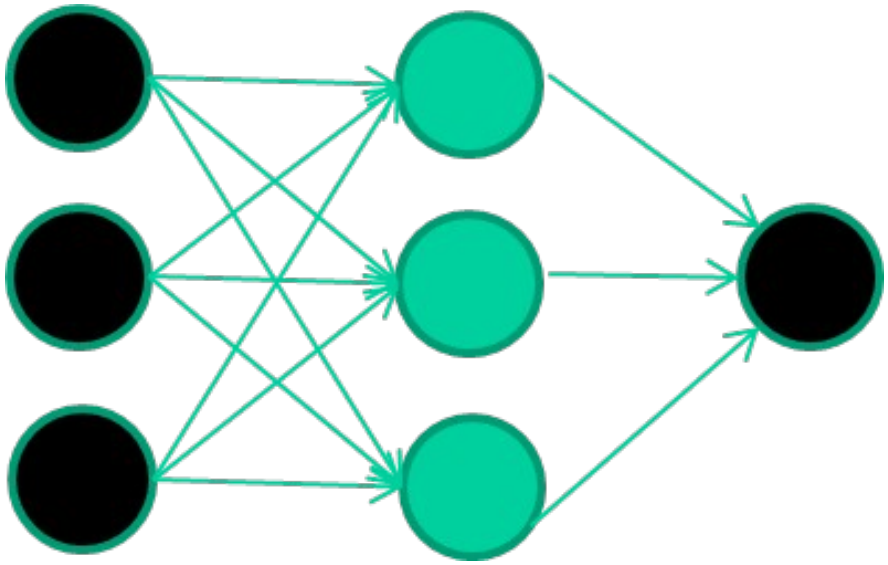



-0.06
2.7
-2.5
-8.6
0.002
1.4
f(x)
$f(x) = \frac{1}{1+e^{-x}}$
1
0.5
0
−6
−4
−2
0
2
4
6
$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

NN training

Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

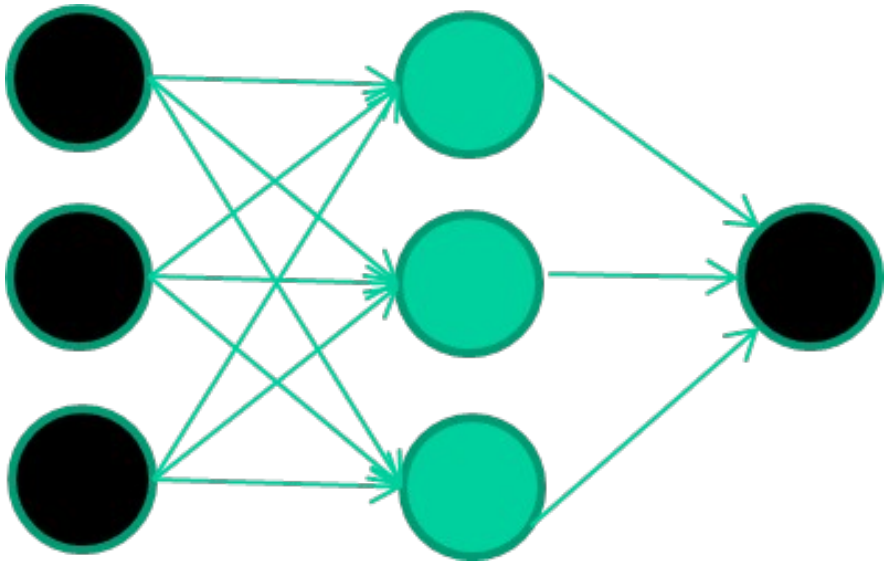


Training data

Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Initialization with random weights

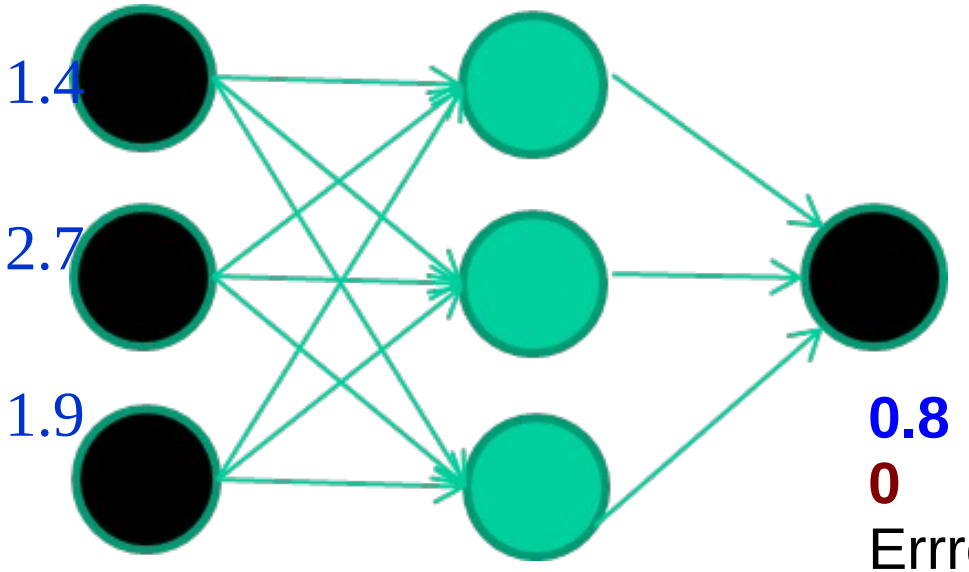


Training data

Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

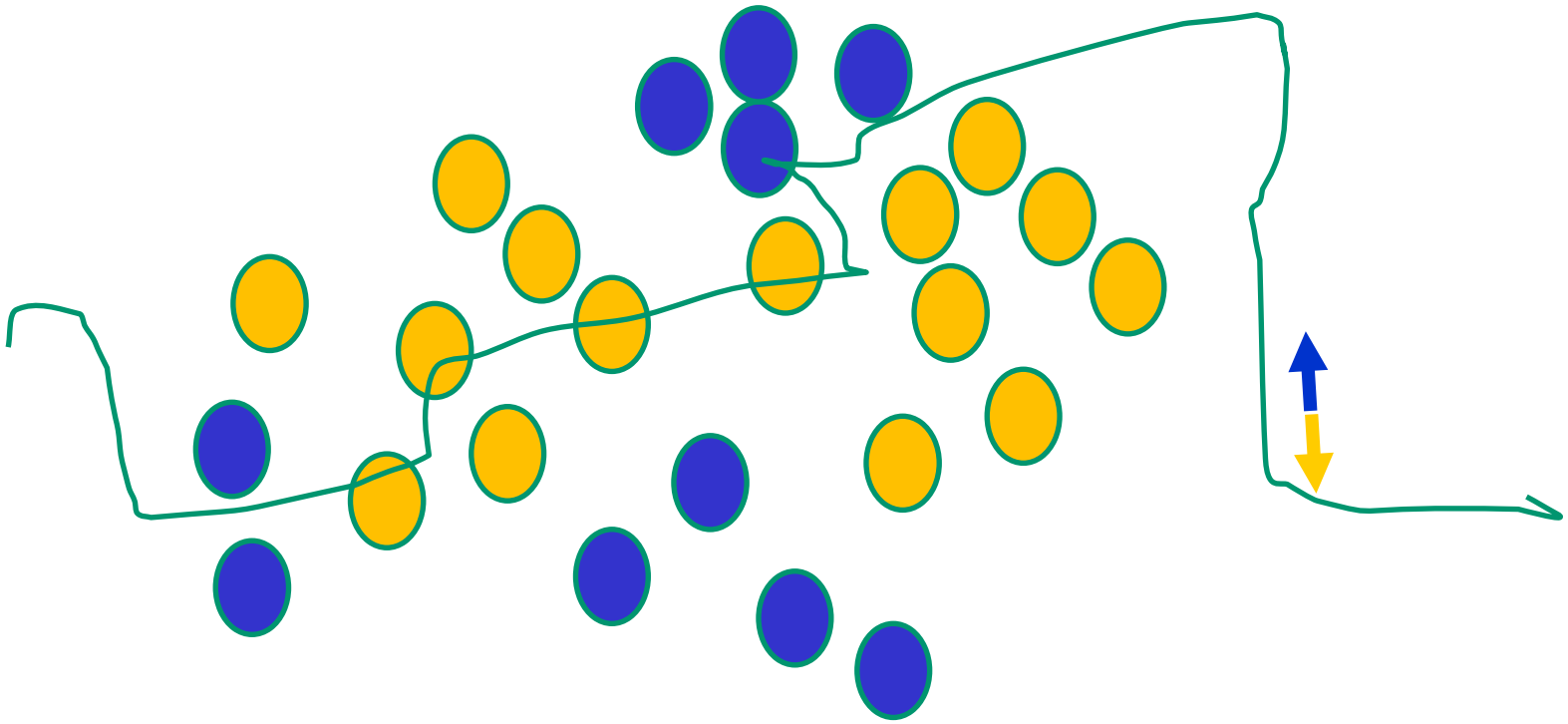
etc …

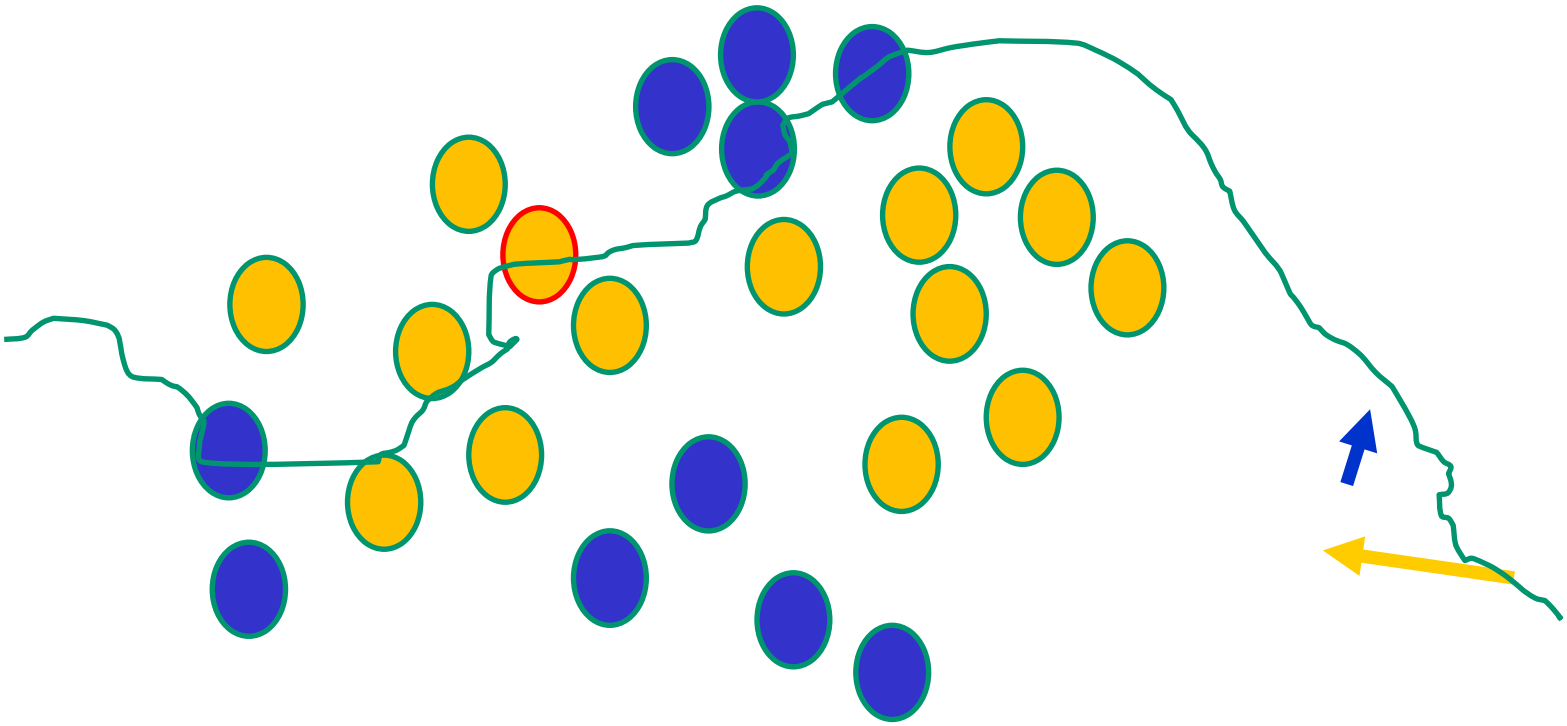
Reading data
Processing them by network
Result compared with the true value

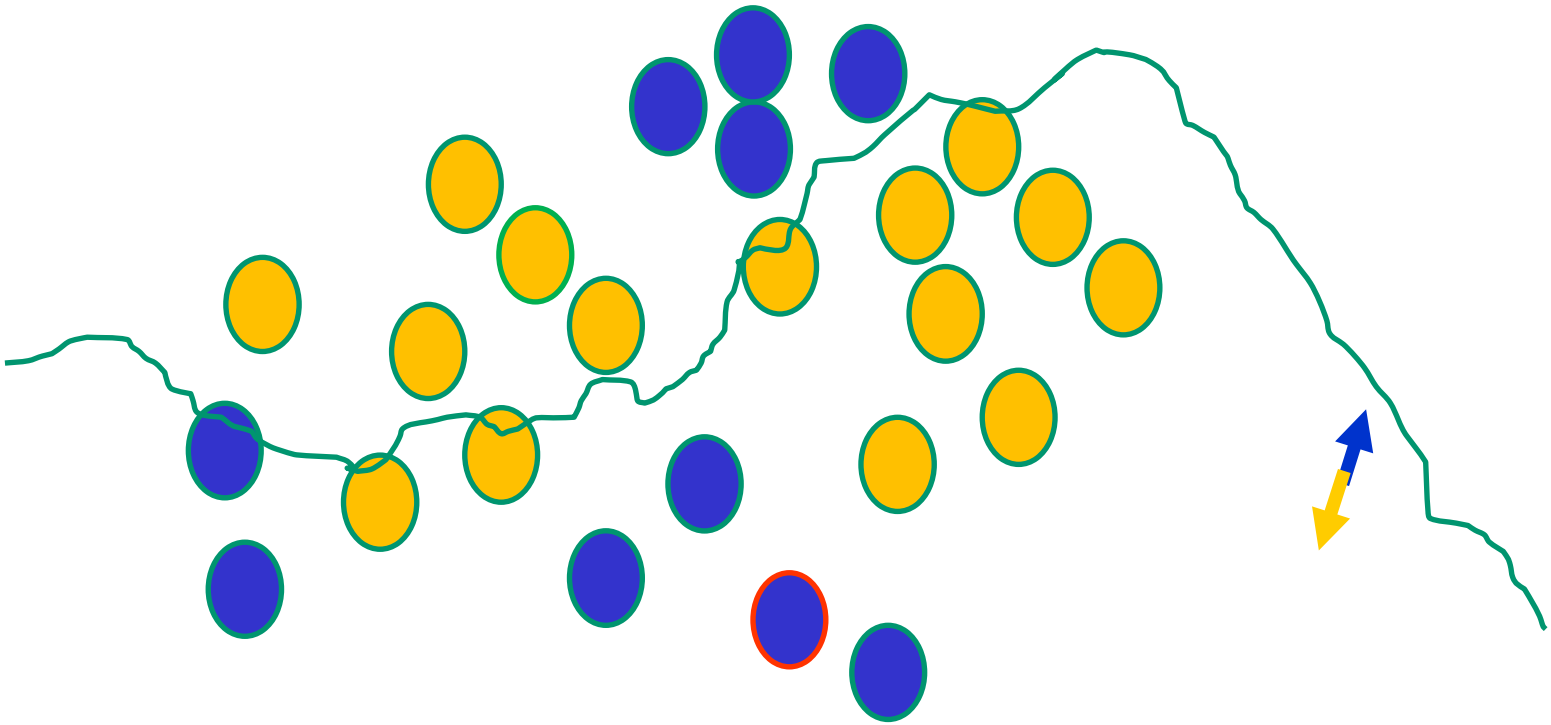


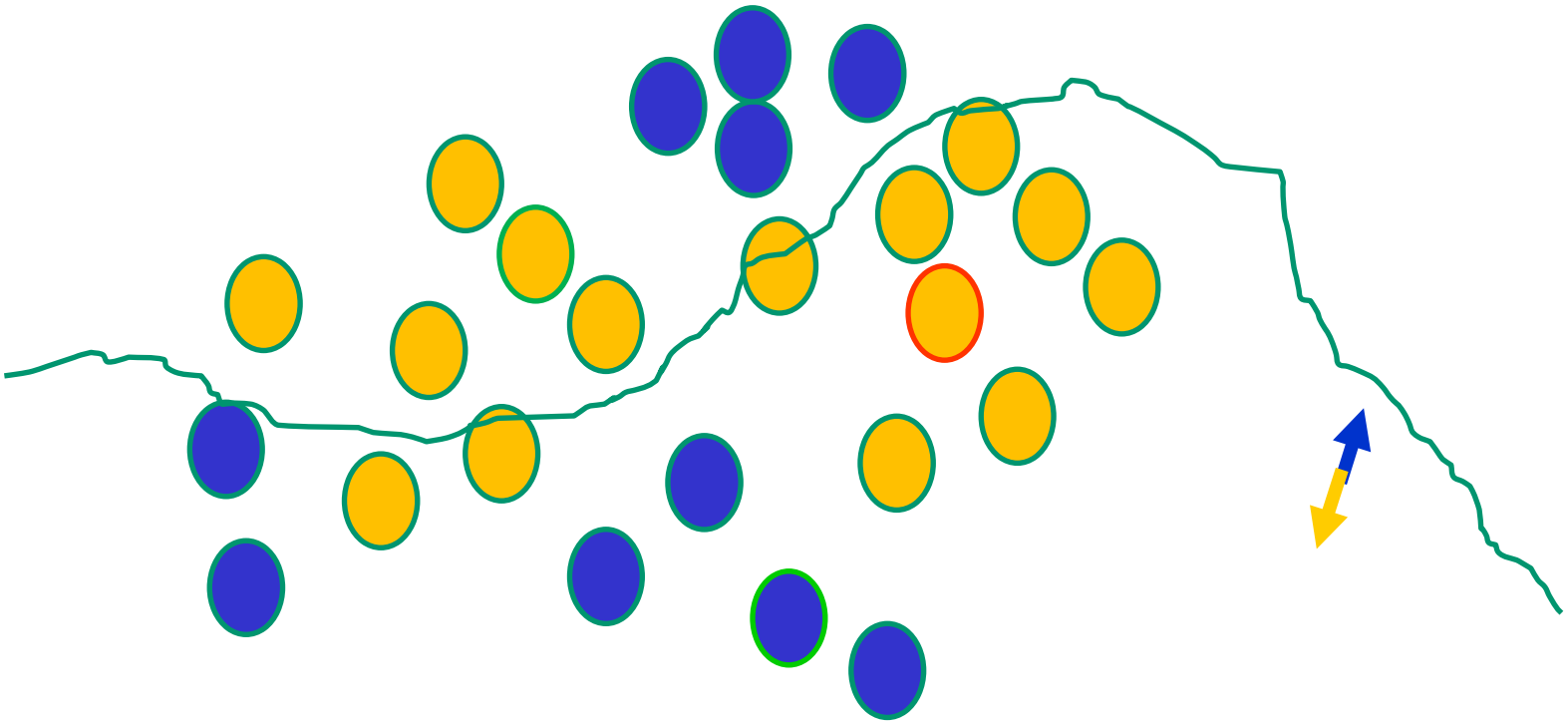
The weights are modified. Modification Based on this error.

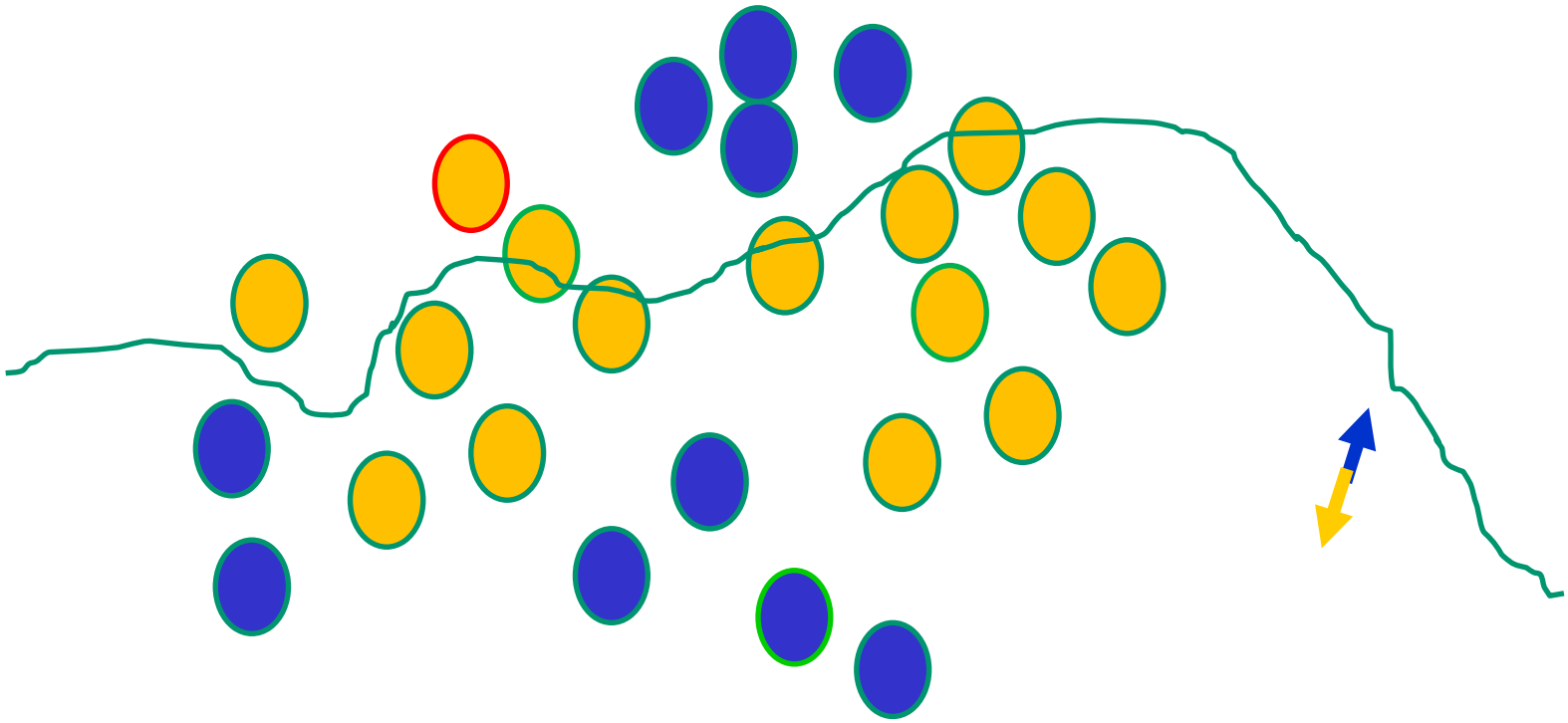
We repeat many times, each time modifying the weights
Training algorithms take care, that the error is smaller and smaller

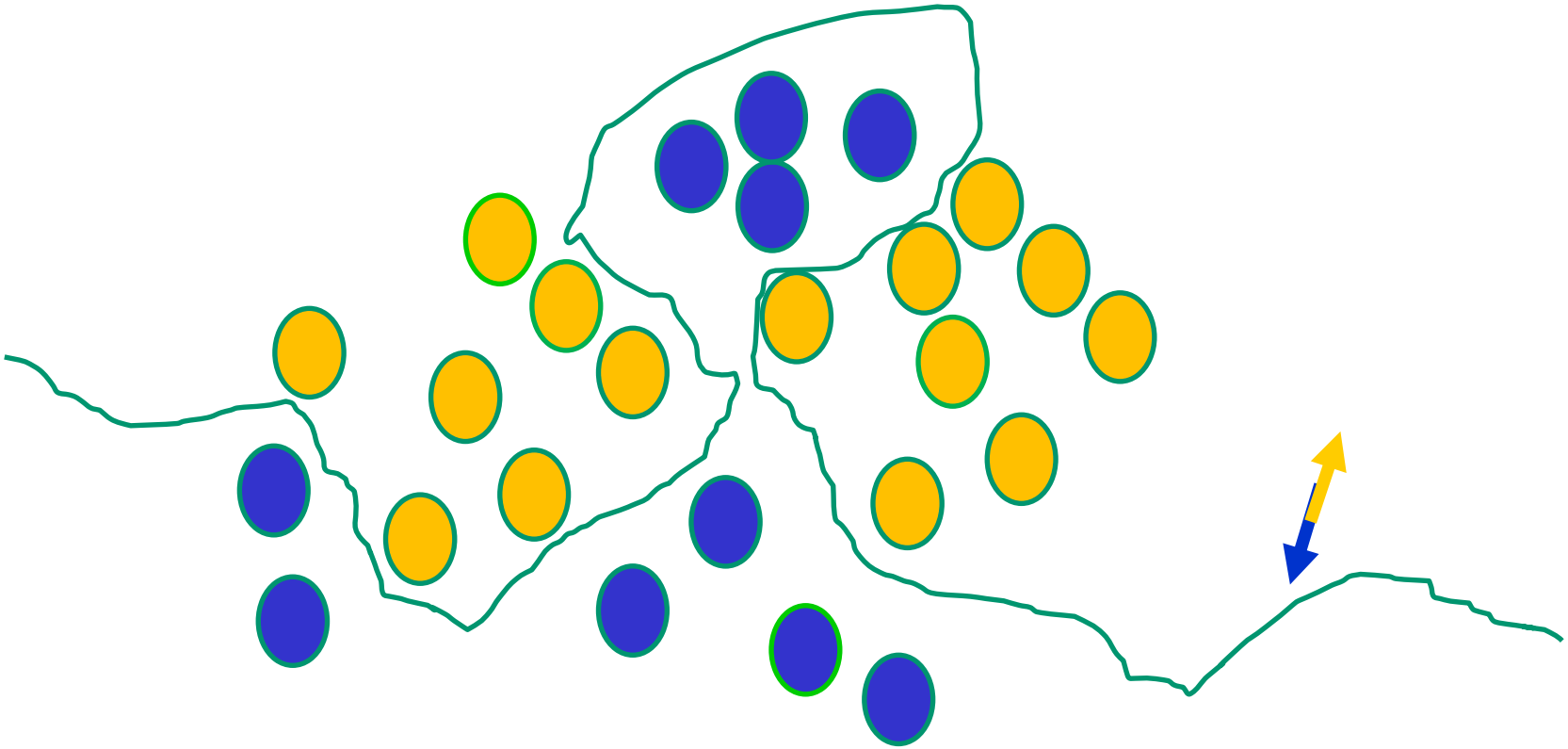












Remark

- If the activation function is non-linear, than a neural network with one hidden layer can classify any problem (fits any function).
- There exists a set of weights, which allows to do that. However, the problem is to find this set of weights...

How to identify the features?

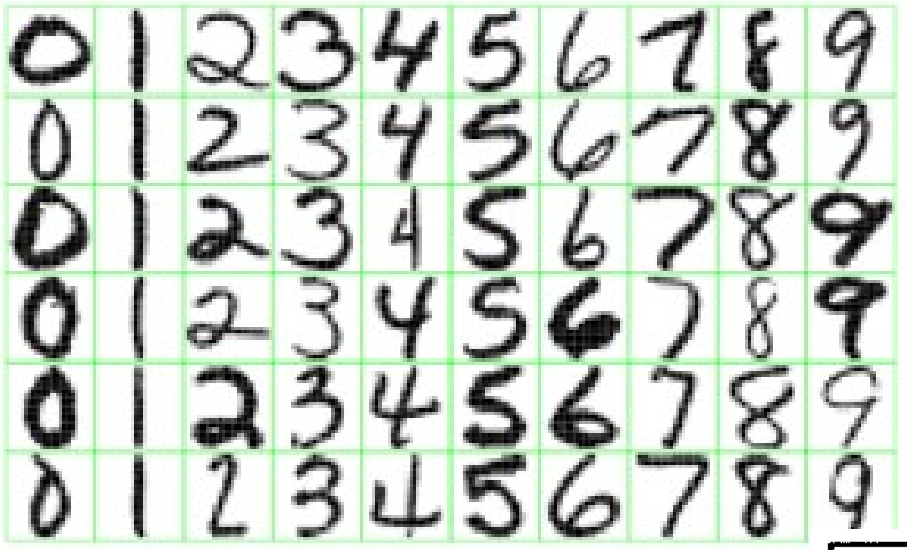
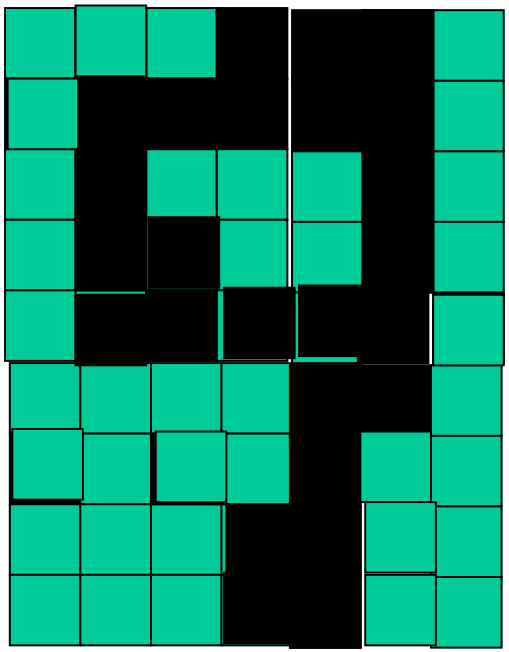
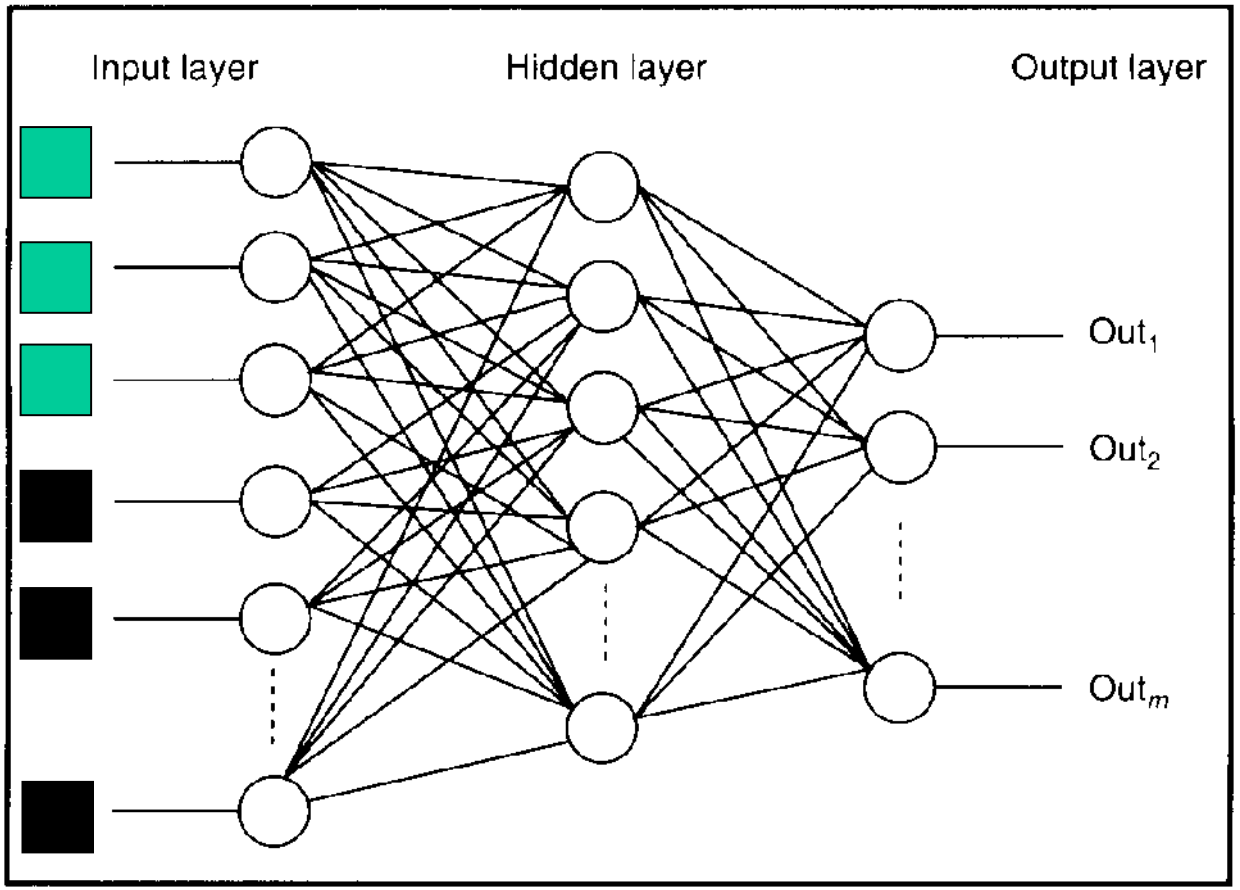


Figure 1.2: *Examples of handwritten digits from postal envelopes.*





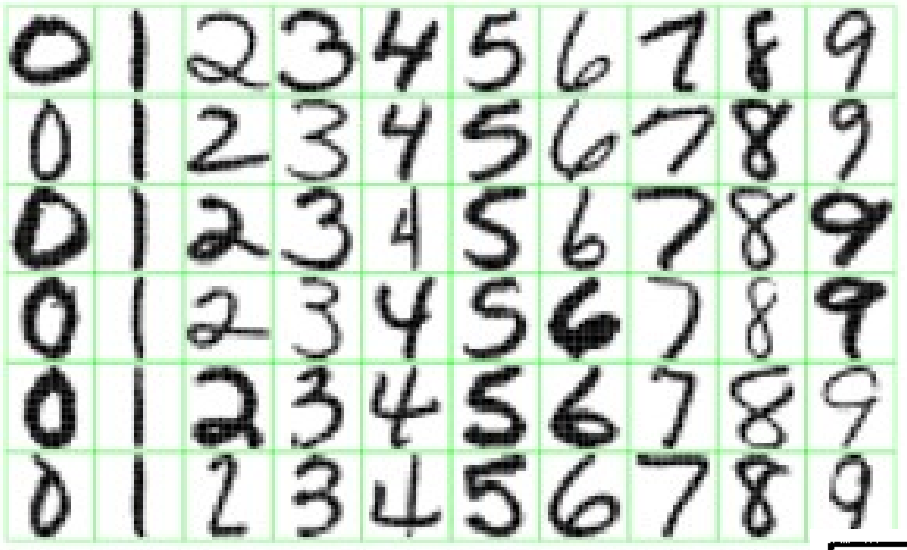
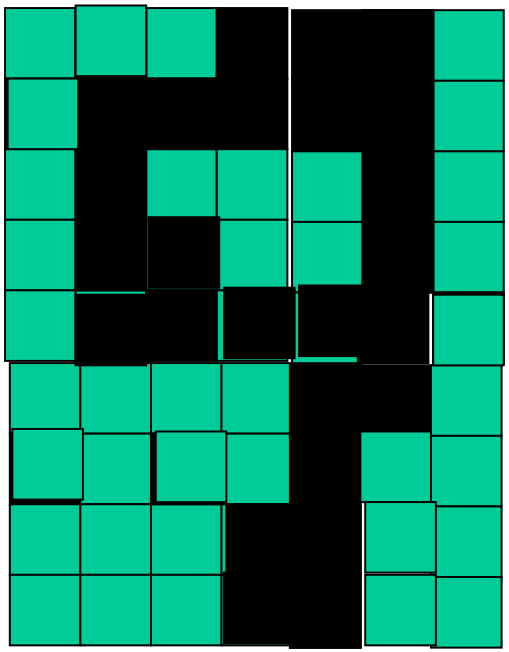
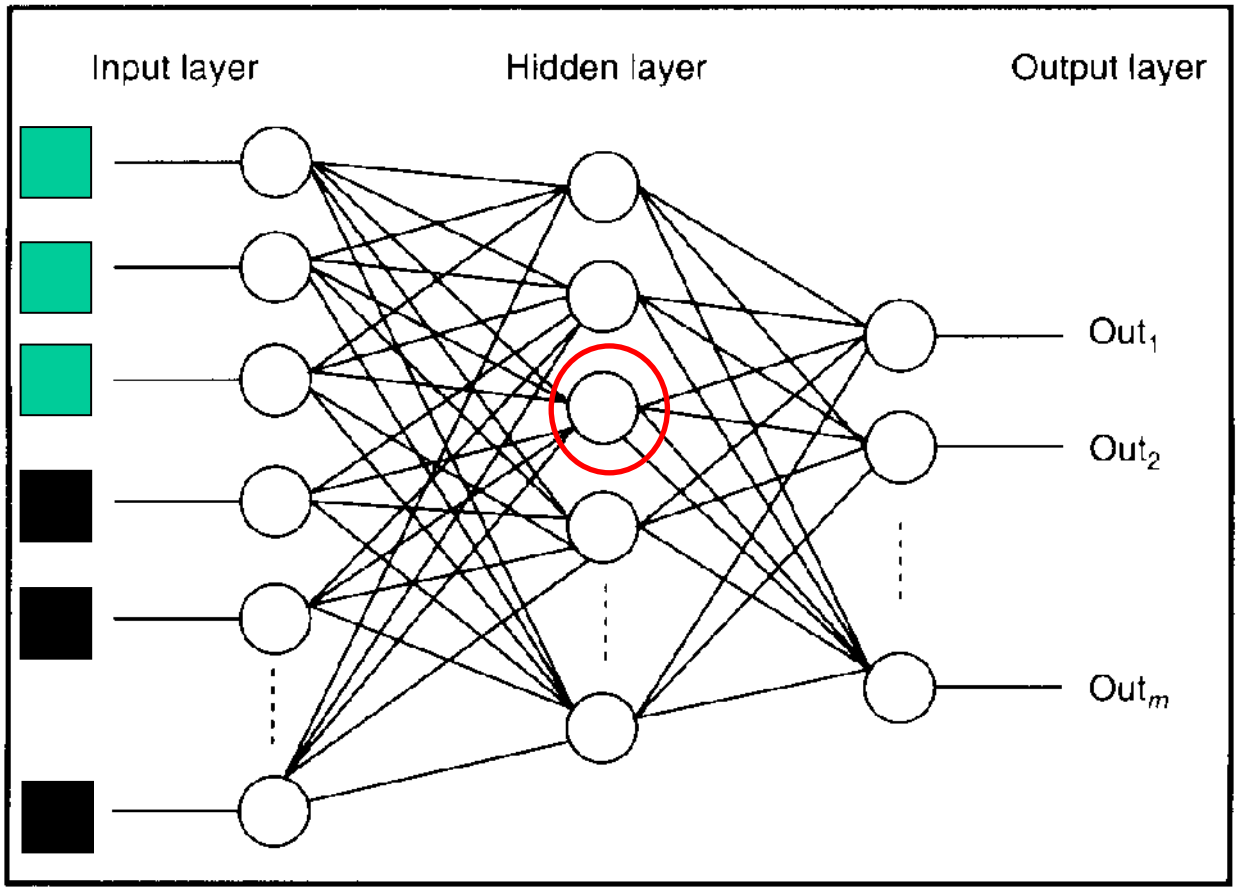


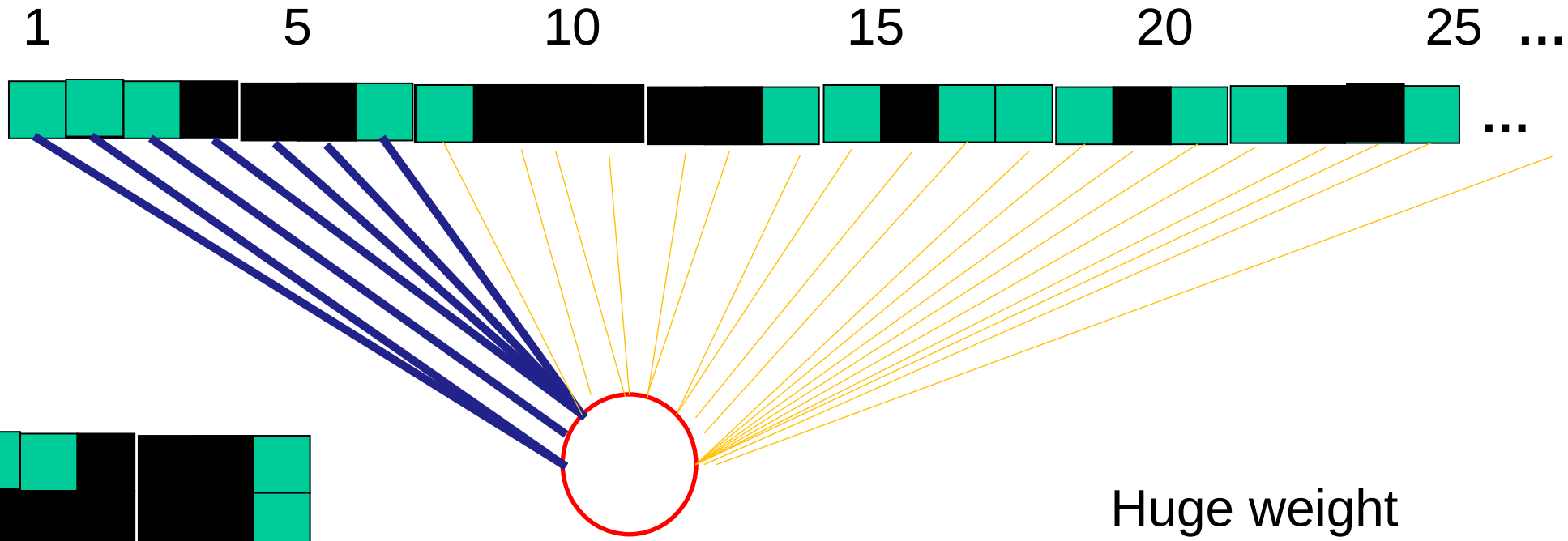
Figure 1.2: *Examples of handwritten digits from postal envelopes.*

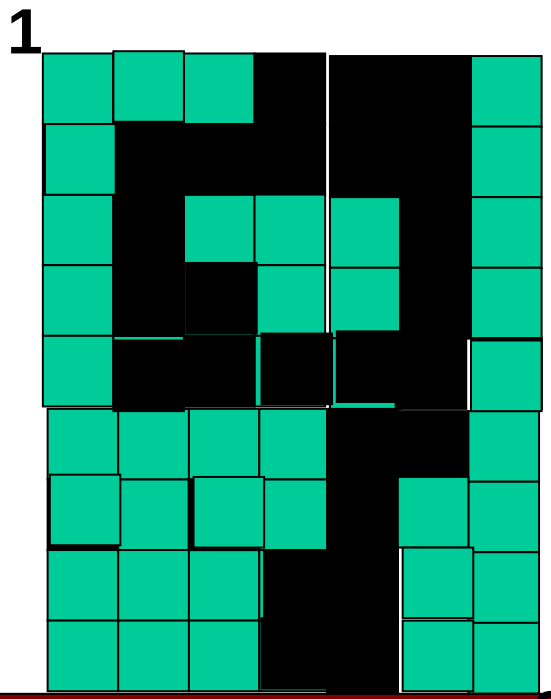


What is this neuron doing?



Neurons in the hidden layers are the self-organizing feature detectors

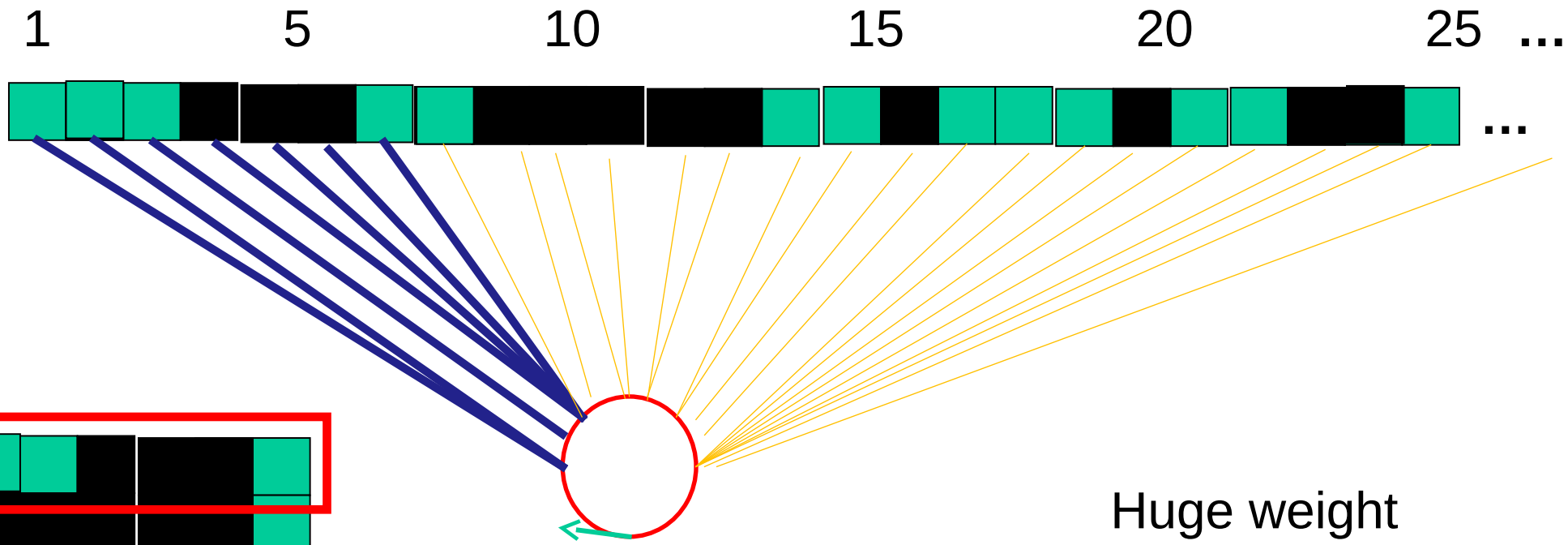


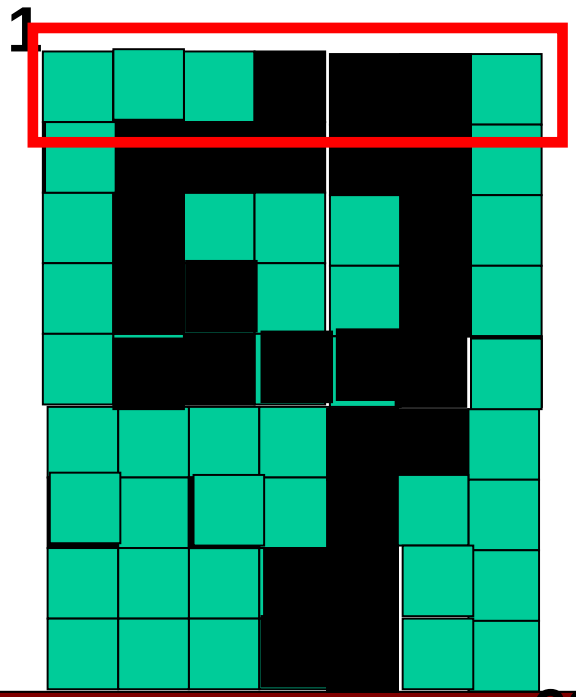


Huge weight

Small weight

What can it detect?



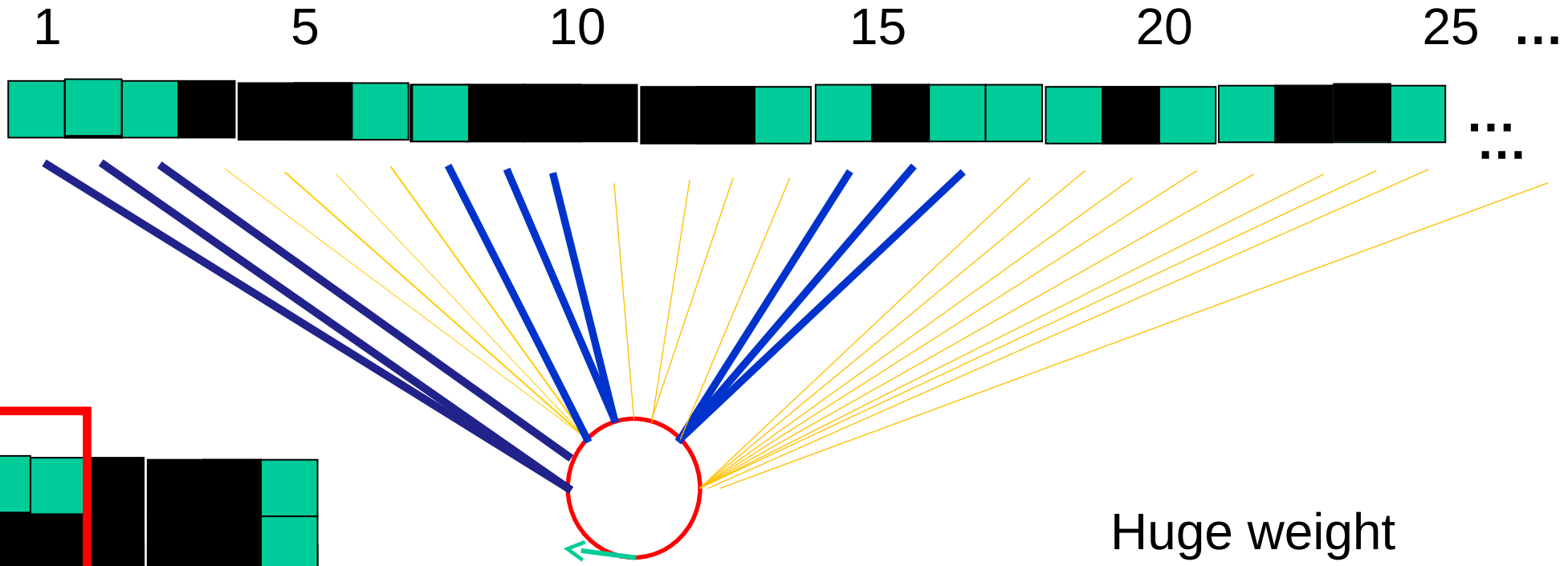


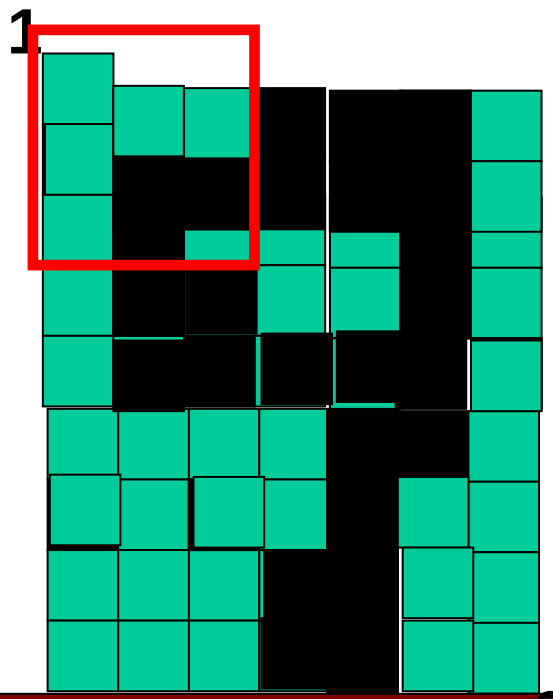
Huge weight

Small weight

It sends a strong signal, when it finds a horizontal line in the top row of pixels, ignores anything else.

What can it detect?





Huge weight

Small weight

Sends strong signal, if a dark region in the left upper corner is found.

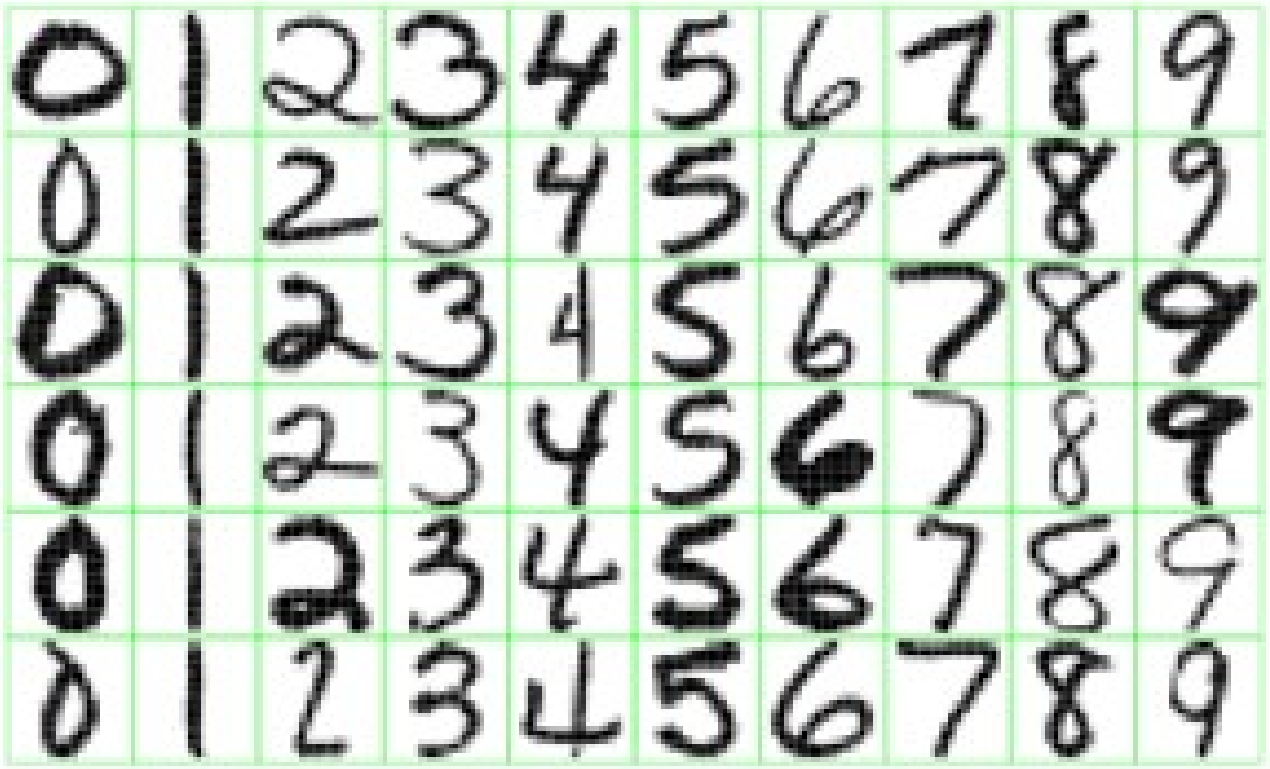


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What feature should detect a neural network recognizing the handwriting?

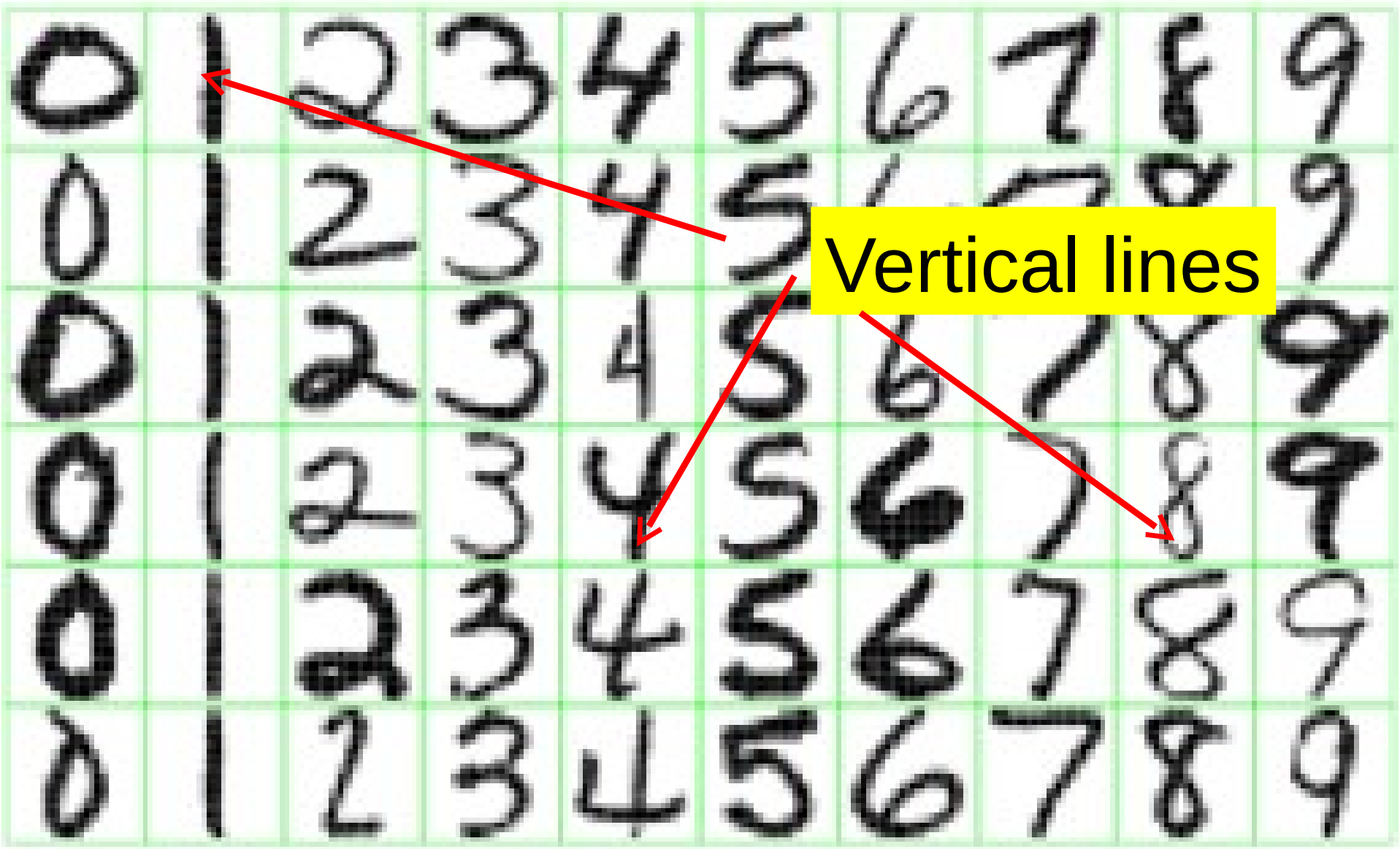
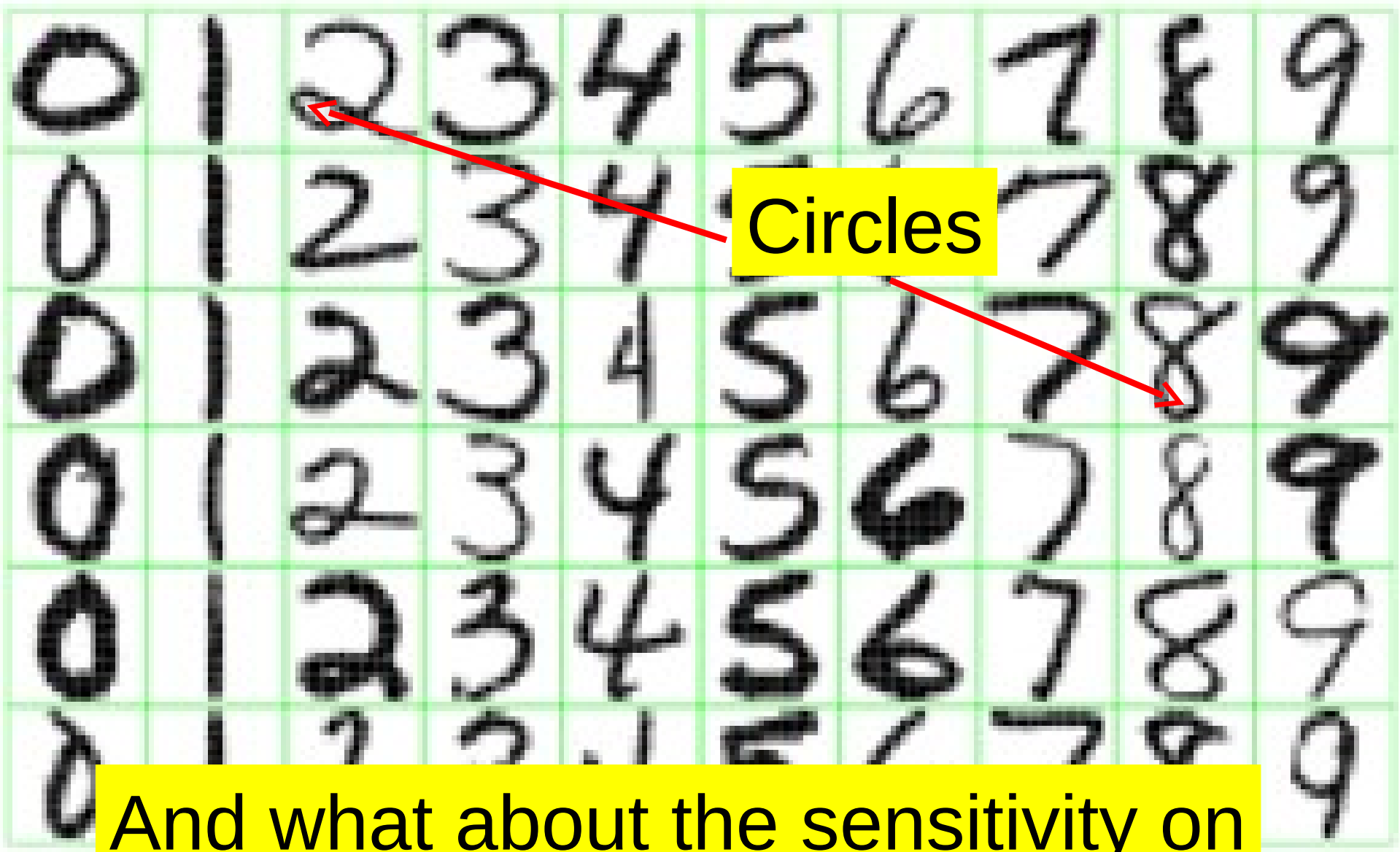


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1



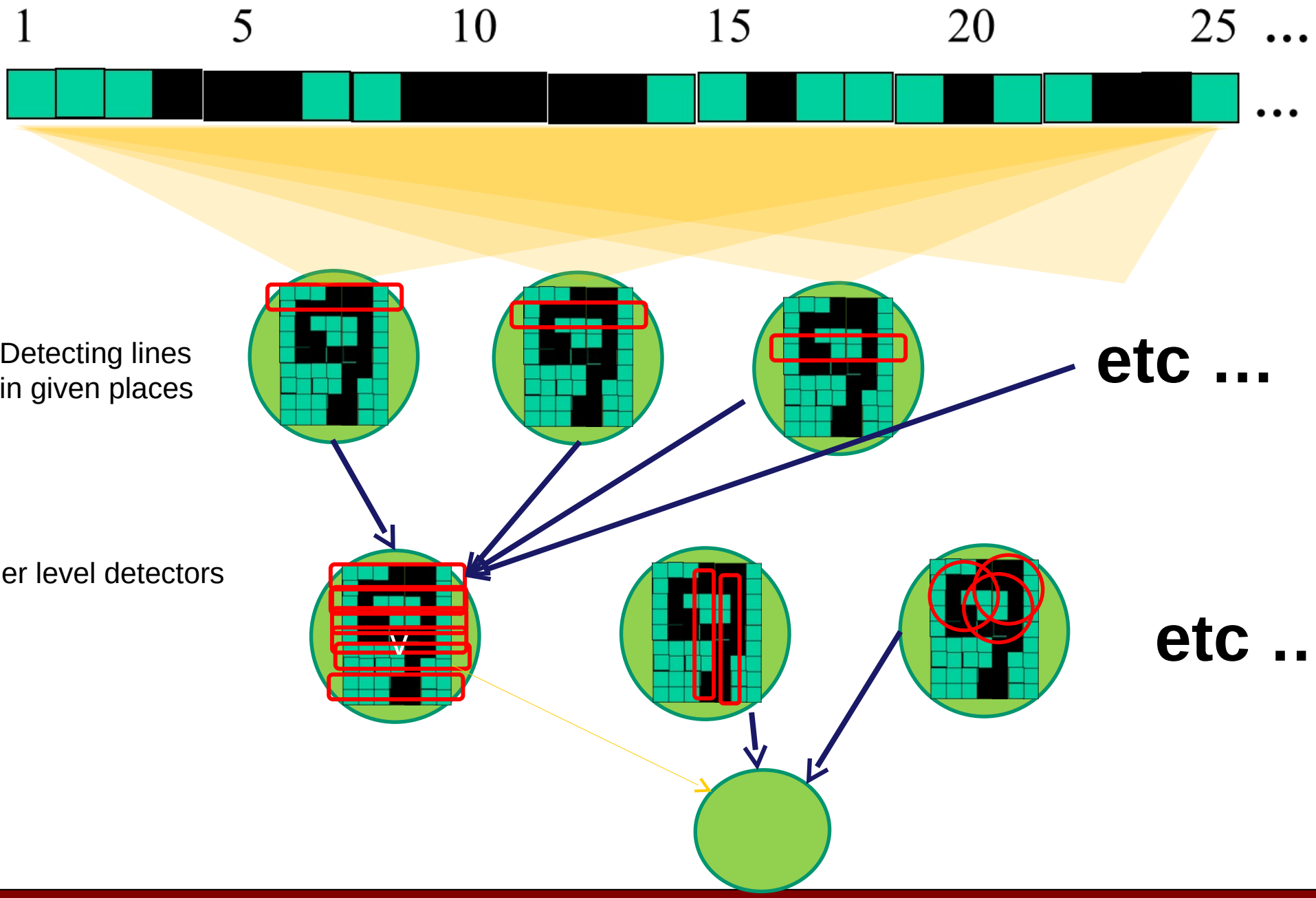
1

And what about the sensitivity on position of these features???

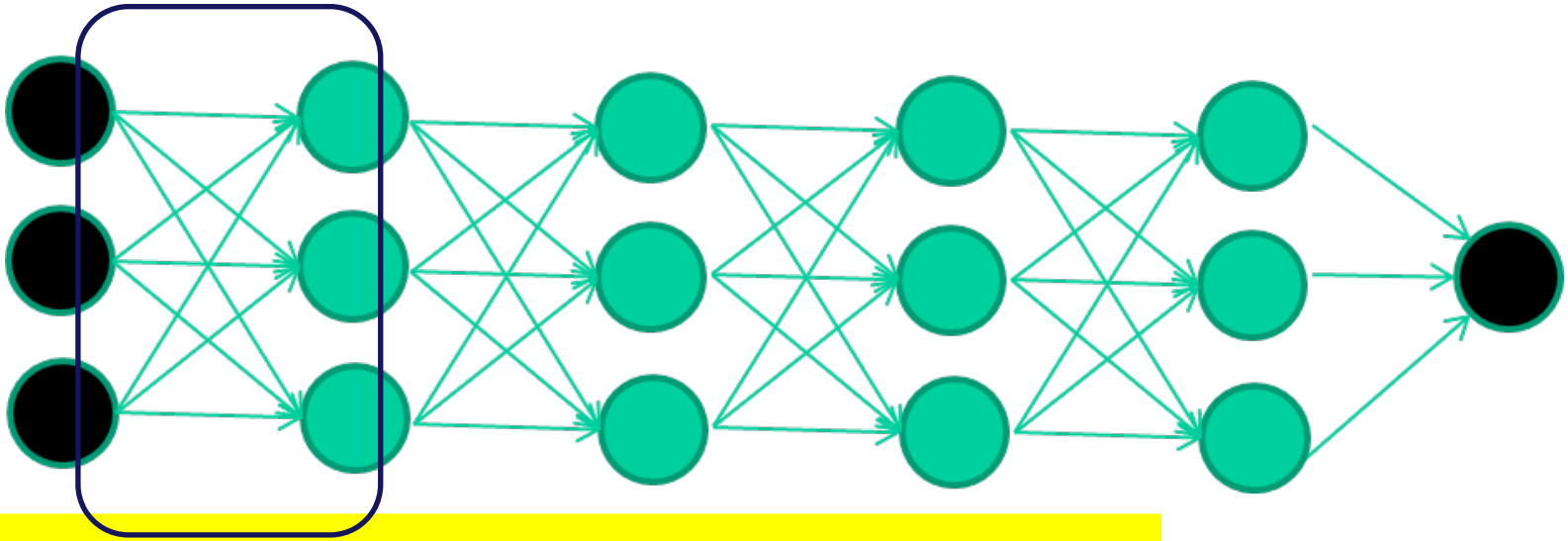
Fig. U.S.

possess characteristics.

Next layers can learn the higher level features



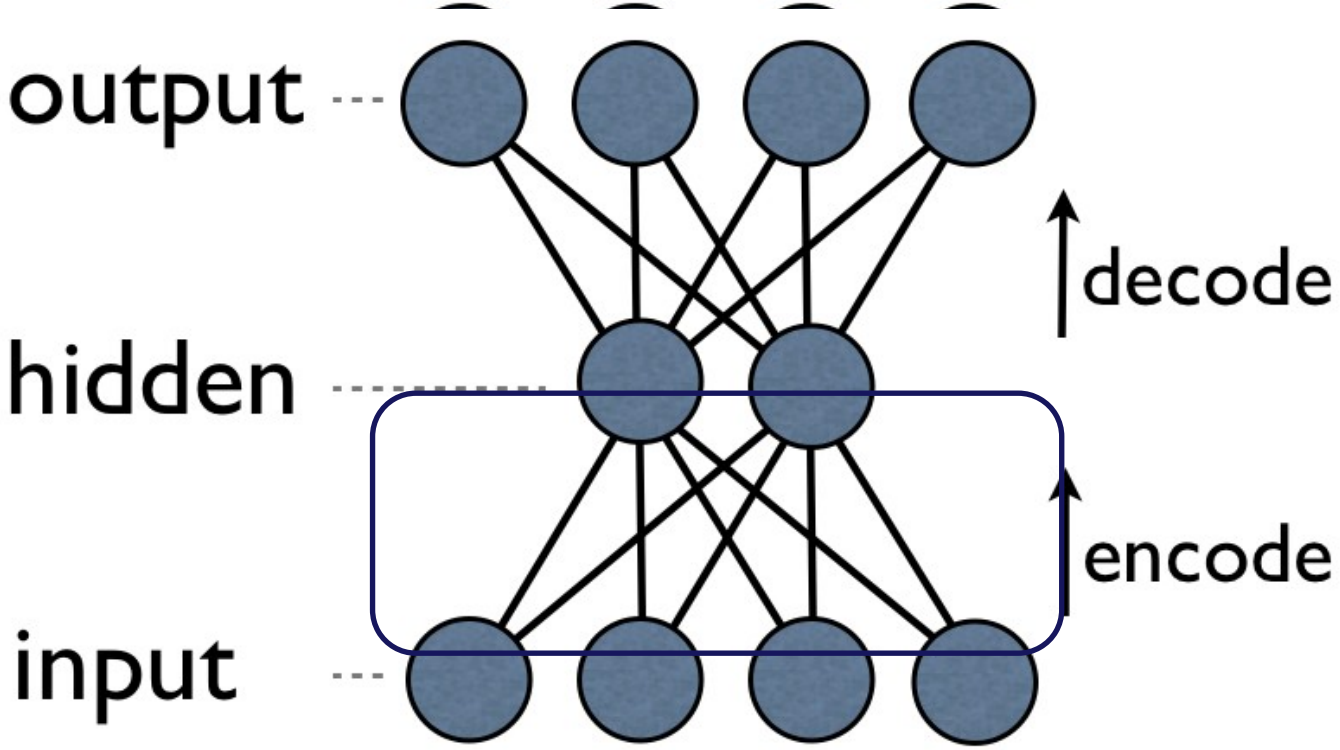
Deep network



Each hidden layer is an automatic feature detector

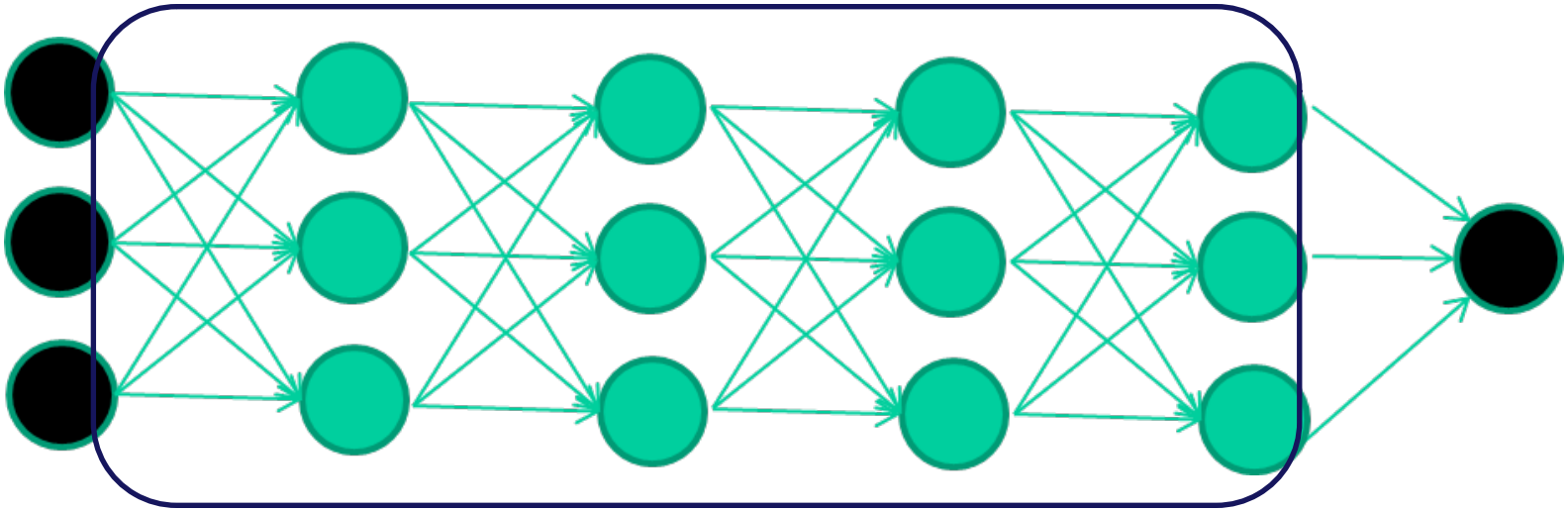
an auto-encoder

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

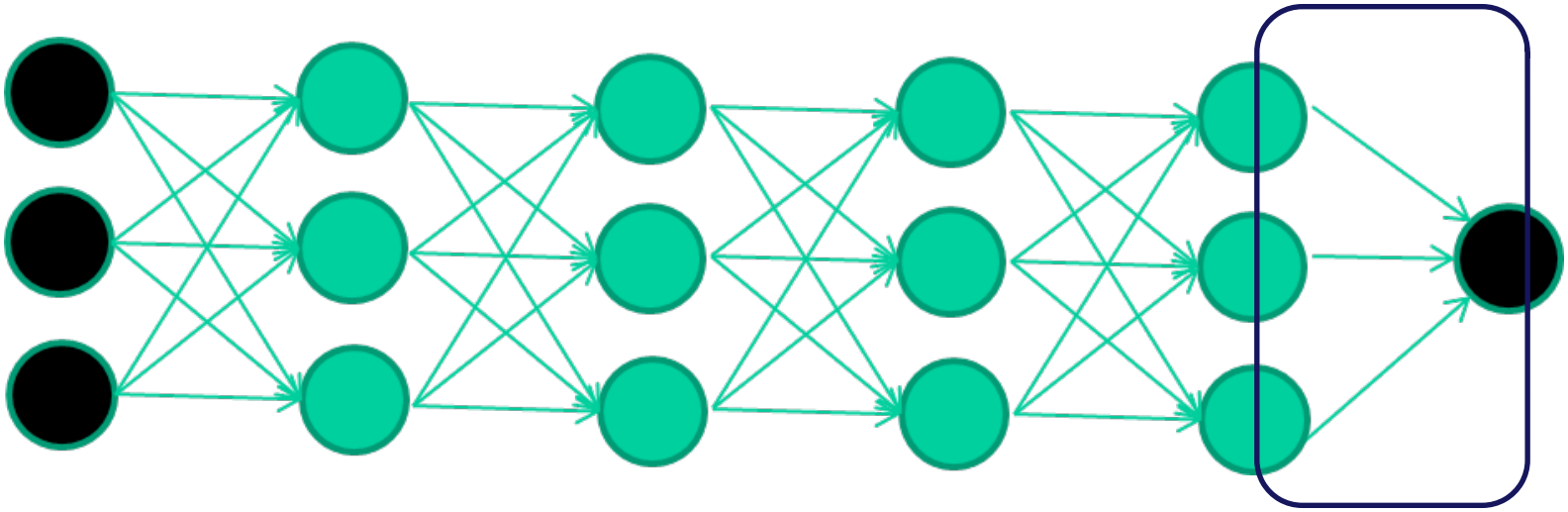


The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
If there is a structure in the data, than it should find features.

Hidden layers are trained to identify features

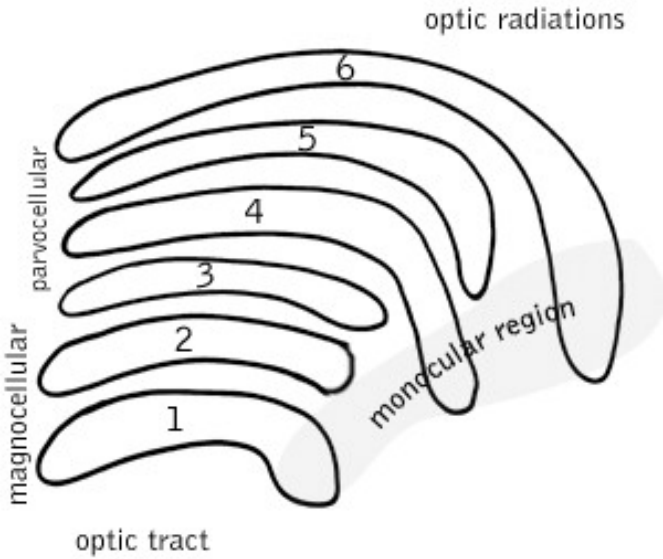


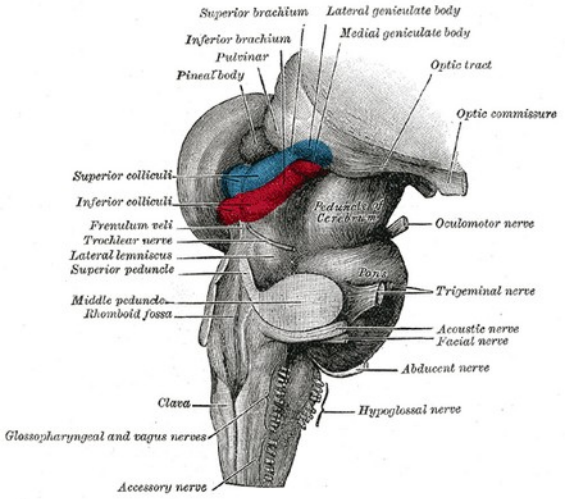
The last layer performs the actual classification



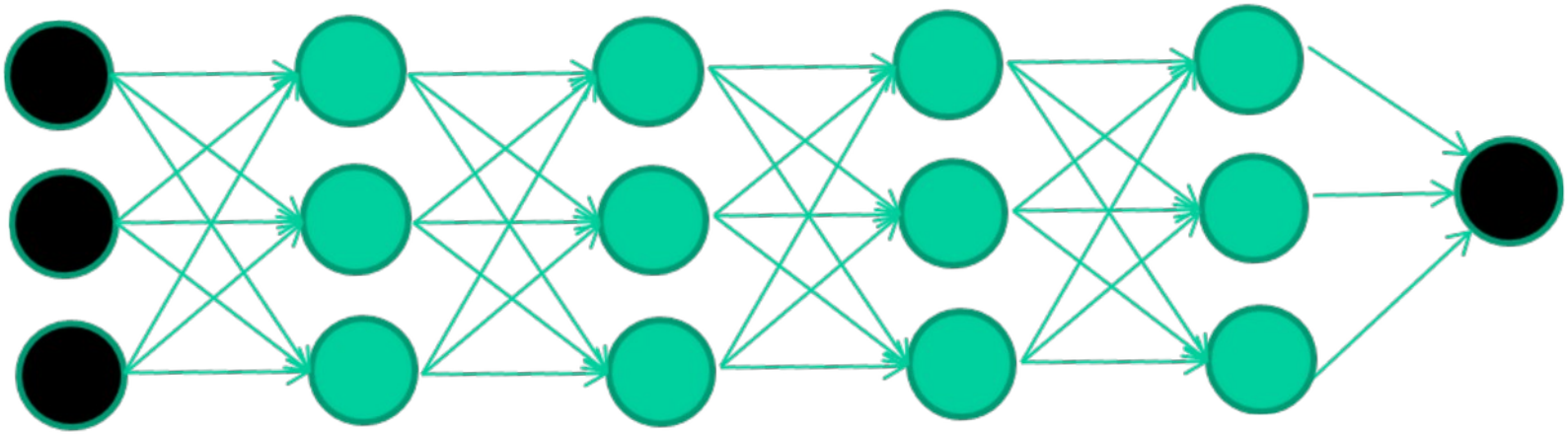
Such an organised network makes sense...

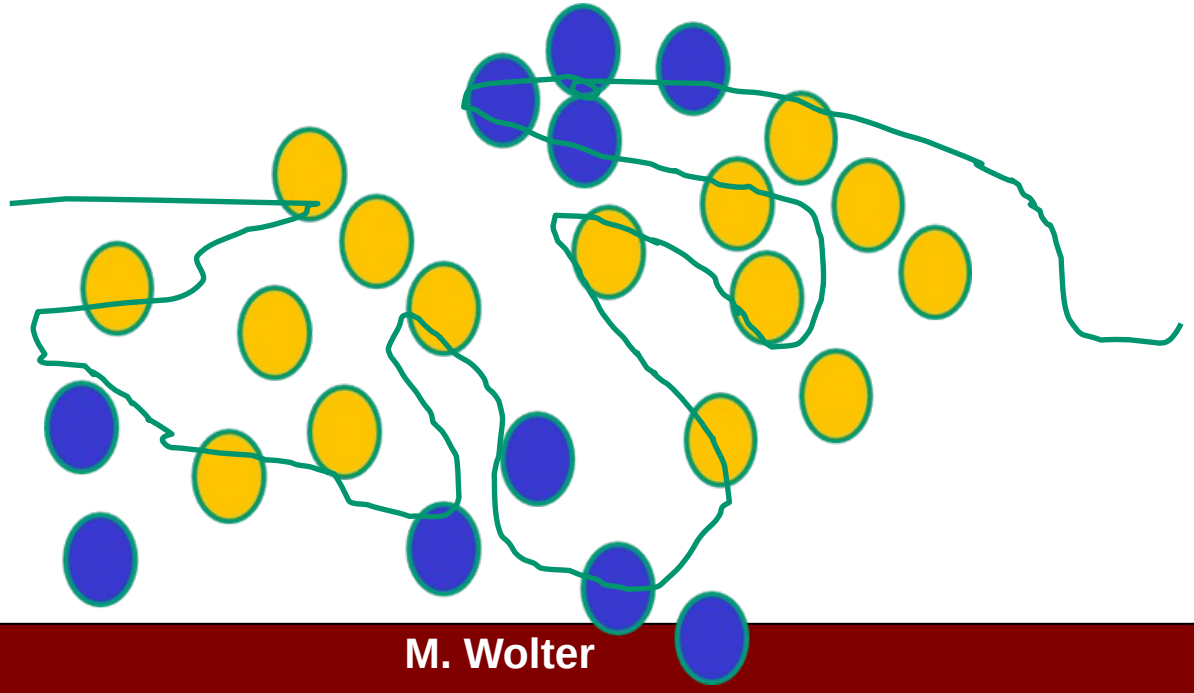
Our brains probably work in a similar way





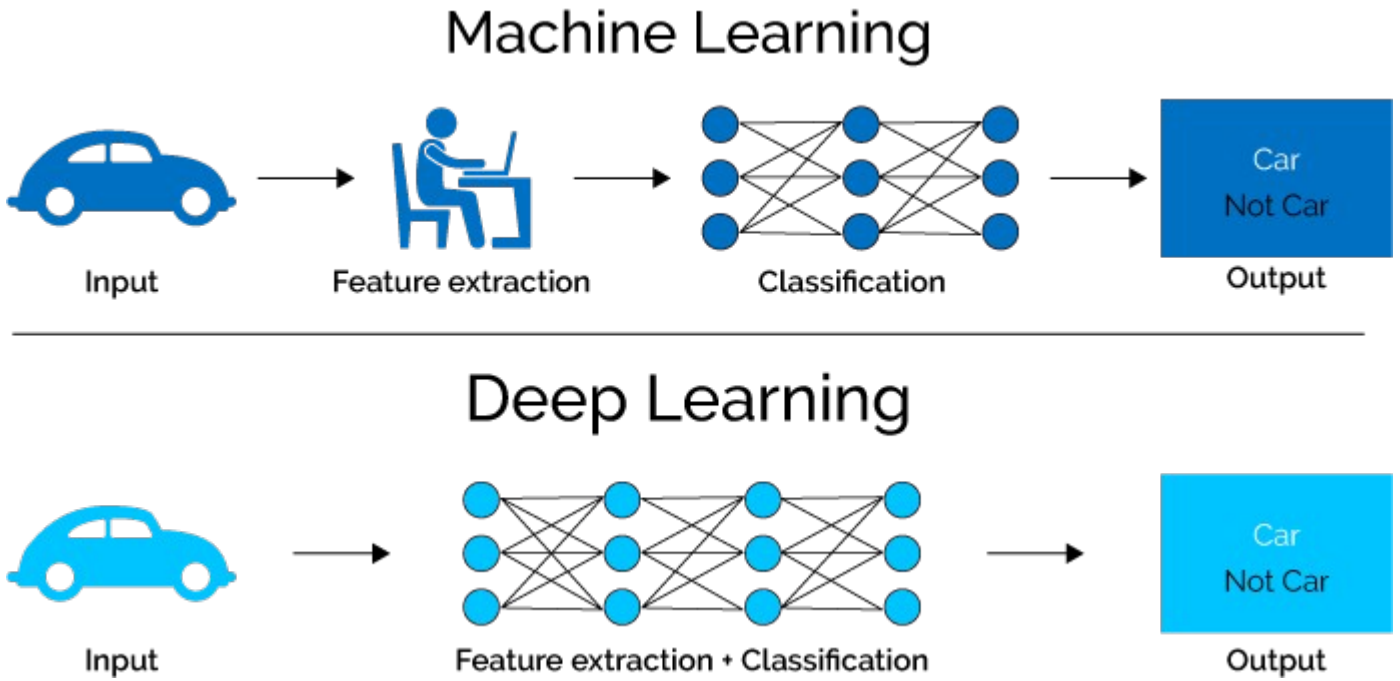
Unfortunately, until recent (10?) years we didn't know how to train a deep network





Machine Learning and Deep Learning

- **Traditional ML (BDT, NN etc)** – the scientist finds good, well discriminating variables (~10), called “features”, and performs classification using them as inputs for the ML algorithm.
- **Deep Learning** – thousands or millions of input variables (like pixels of a photo), the features are *automagically* extracted during training.

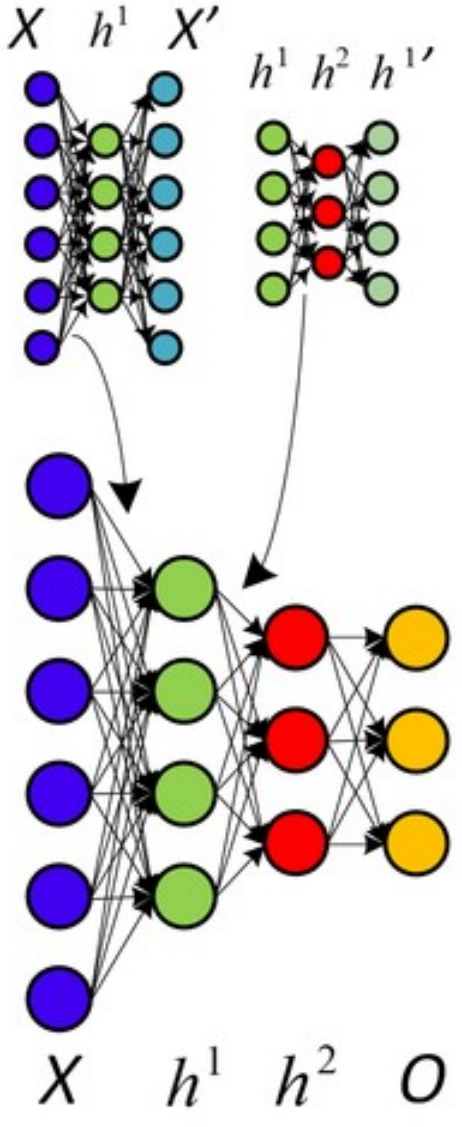


Deeper network?

Traditional Neural Networks have one or two hidden layers.

Deep Neural Network: a stack of sequentially trained **auto encoders**, which recognize different features (more complicated in each layer) and automatically prepare a new representation of data. This is how our brains are organized.

But how to train such a stack?



http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

Training a Deep Neural Network

- In the early 2000s, attempts to train deep neural networks were frustrated by the apparent failure of the well known back-propagation algorithms (backpropagation, gradient descent). Many researchers claimed NN are gone, only Support Vector Machines and Boosted Decision Trees should be used!

- In 2006, Hinton, Osindero and Teh[1] first time **succeeded in training a deep neural network** by first initializing its parameters sequentially, layer by layer. Each layer was trained to produce a representation of its inputs that served as the training data for the next layer. Then the network was tweaked using gradient descent (standard algorithm).

- **There was a common belief that Deep NN training requires careful initialization of parameters and sophisticated machine learning algorithms.**

[1]Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets, Neural Computation 18, 1527-1554.

Training with a brute force

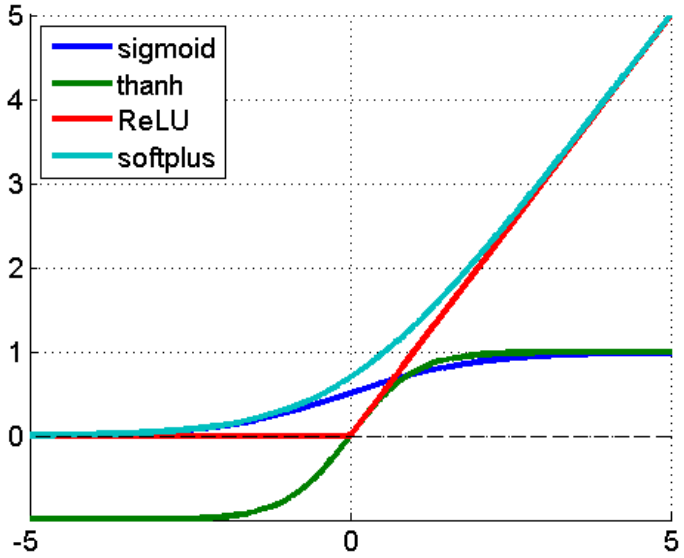
- In 2010, a surprising counter example to the conventional wisdom was demonstrated[1].
- Deep neural network was trained to classify the handwritten digits in the MNIST[2] data set, which comprises 60,000 28 × 28 = 784 pixel images for training and 10,000 images for testing.
- They showed that a plain DNN with architecture (784, 2500, 2000, 1500, 1000, 500, 10 – **HUGE!!!**), trained using standard stochastic gradient descent (Minuit on steroids!), outperformed all other methods that had been applied to the MNIST data set as of 2010. The error rate of this ∼12 million parameter DNN was 35 images out of 10,000.

  The training images were randomly and slightly deformed before every training epoch. The entire set of 60,000 undeformed images could be used as the validation set during training, since none were used as training data.

[1] Cireşan DC, Meier U, Gambardella LM, Schmidhuber J. ,Deep, big, simple neural nets for handwritten digit recognition. Neural Comput. 2010 Dec; 22 (12): 3207-20.
[2] http://yann.lecun.com/exdb/mnist/

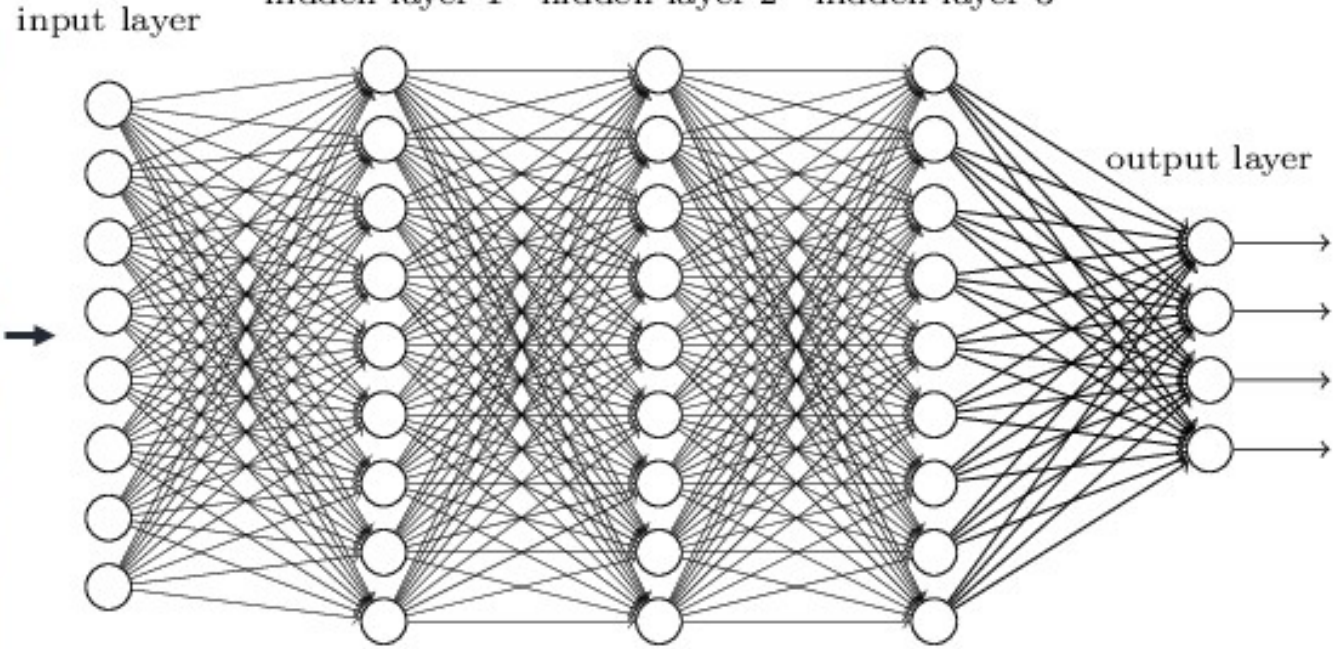
Why it didn’t work before?

- More data, clusters of GPU/CPU (computing power!)
- The particular non-linear activation function chosen for neurons in a neural net makes a big impact on performance, and the one often used by default is not a good choice.
- The old vanishing gradient problem happens, basically, because backpropagation involves a sequence of multiplications that invariably result in smaller derivatives for earlier layers. That is, unless weights are chosen with difference scales according to the layer they are in - making this simple change results in significant improvements.



Deep neural networks learn hierarchical feature representations

hidden layer 1 hidden layer 2 hidden layer 3



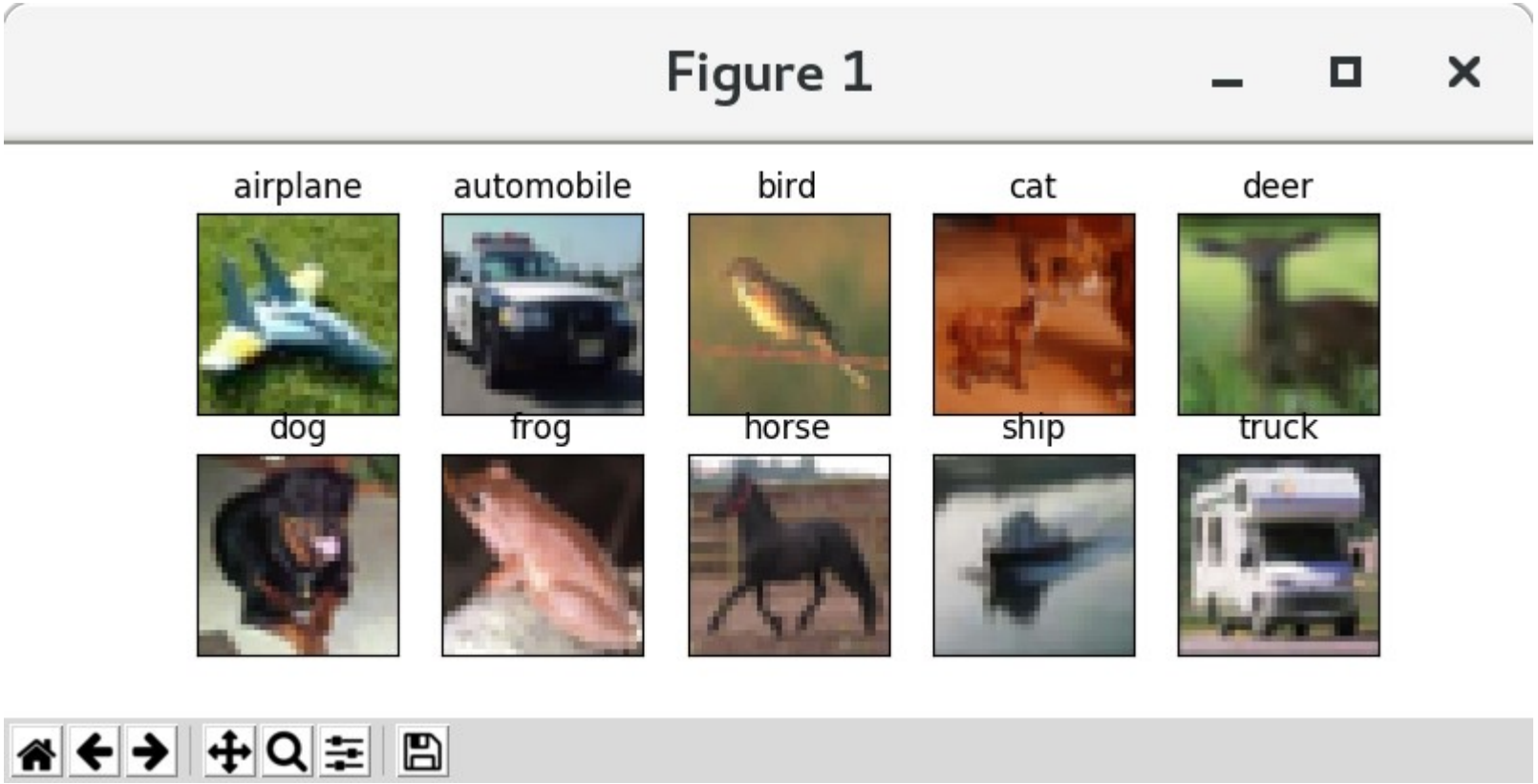
A Deep Neural Network Applet

Applet showing the performance of deep NN:

http://cs.stanford.edu/people/karpathy/convnetjs/

An example – pattern recognition with KERAS and TensorFlow

- CIFAR10 small image classification. Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.



Deep Neural Network

```
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0

_________________________________________________________________
          OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

              Input   #####      3   32   32
             Conv2D    \|/  -------------------       896     0.0%
               relu   #####     32   32   32
             Conv2D    \|/  -------------------      9248     0.0%
               relu   #####     32   30   30
       MaxPooling2D   Y max -------------------         0     0.0%
                      #####     32   15   15
            Dropout    | || -------------------         0     0.0%
                      #####     32   15   15
             Conv2D    \|/  -------------------     18496     1.0%
               relu   #####     64   15   15
             Conv2D    \|/  -------------------     36928     2.0%
               relu   #####     64   13   13
       MaxPooling2D   Y max -------------------         0     0.0%
                      #####     64    6    6
            Dropout    | || -------------------         0     0.0%
                      #####     64    6    6
            Flatten   ||||| -------------------         0     0.0%
                      #####         2304
              Dense   XXXXX -------------------   1180160    94.0%
               relu   #####          512
            Dropout    | || -------------------         0     0.0%
                      #####          512
              Dense   XXXXX -------------------      5130     0.0%
            softmax   #####           10
Train on 50000 samples, validate on 10000 samples
```

Training:
~24 h on 12 core machine on our Cloud cluster. Much faster while using GPU!

200 epochs

Results


Recognized as
Really was
cat cat
ship ship
airplane ship
airplane airplane
frog frog
frog frog
automobile automobile
frog frog
cat cat
truck automobile
airplane airplane
truck truck
dog dog
horse horse
truck truck
frog ship
dog dog
horse horse
ship ship
frog frog
horse horse
airplane airplane
deer deer
truck truck
deer dog
frog bird
deer deer
airplane airplane
truck truck
frog frog
frog frog
bird dog
bird deer
dog dog
truck truck
cat bird
horse deer
automobile automobile
truck truck
dog dog
deer deer
frog frog
dog dog
frog frog
airplane airplane
truck truck
cat cat
cat truck
horse horse
frog frog
truck truck
ship ship
frog airplane
cat cat
ship ship
ship ship
horse horse
cat horse
dog deer
cat frog
horse horse
cat cat
frog frog
airplane cat
frog frog
bird bird
automobile automobile
bird bird
dog cat
horse horse
bird bird
frog frog
ship ship
ship ship
airplane airplane
bird bird
airplane truck
cat cat
cat cat
ship ship
ship ship
automobile automobile
automobile automobile
horse horse
bird bird
horse dog
bird bird
ship horse
ship ship
truck truck
airplane airplane
cat cat
ship ship
frog frog
deer deer
frog frog
frog frog
bird airplane
bird airplane
dog horse

Confusion matrix

```
[[754   8  74  47  32   4   9   4  50  18]
 [ 10 875   6  24   3   1  10   0  19  52]
 [ 38   2 678  79  64  52  54  18  11   4]
 [ 14   3  52 647  70 121  51  30   7   5]
 [  7   2  49  85 753  15  39  43   3   4]
 [  6   0  44 192  45 650  26  30   3   4]
 [  3   1  31  70  35   8 837   4   9   2]
 [ 10   0  31  70  57  31   6 790   3   2]
 [ 54  13  15  30  15   4  14   1 831  23]
 [ 31  50   6  36  11   4  12   4  18 828]]
```

That’s all for today

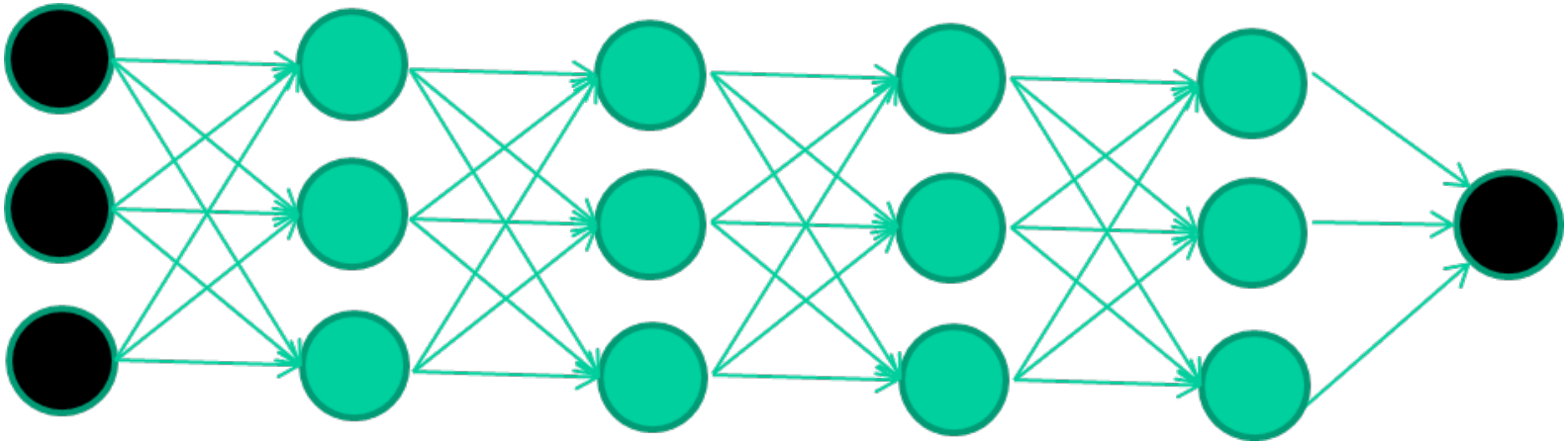
Next week we make a tutorial:

“*How to implement your first deep neural network using Keras*”,

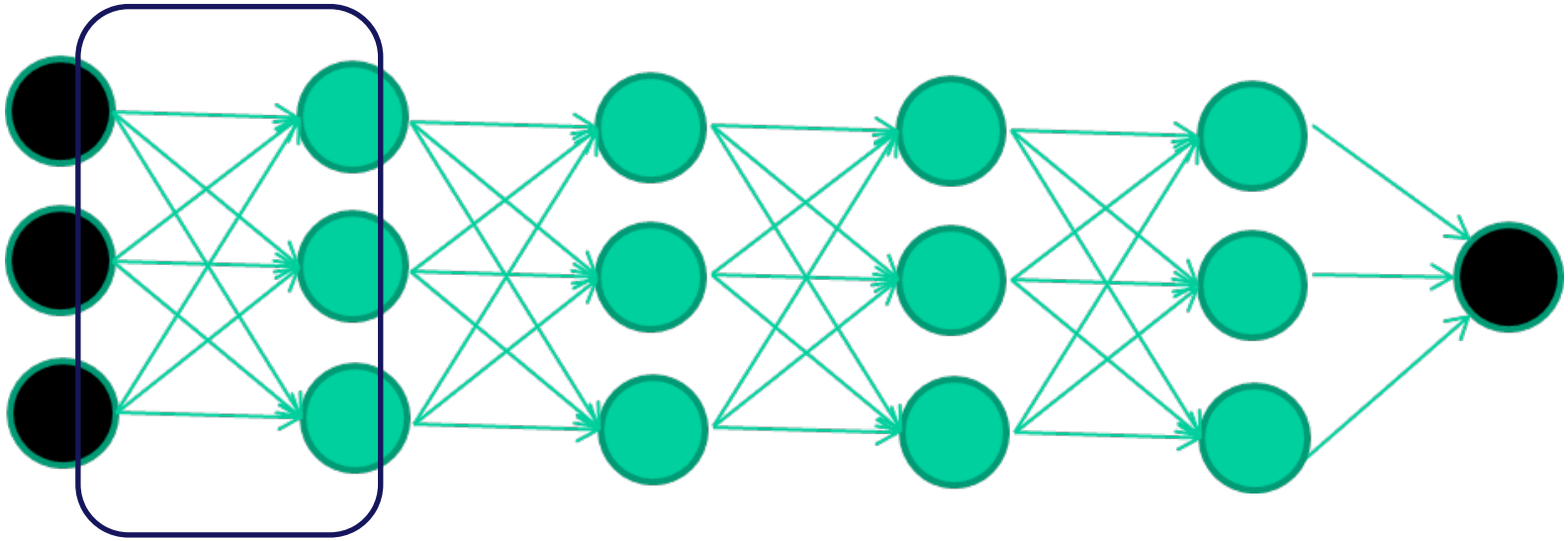
so please bring your laptops!

Backup

New method of training (basic info)

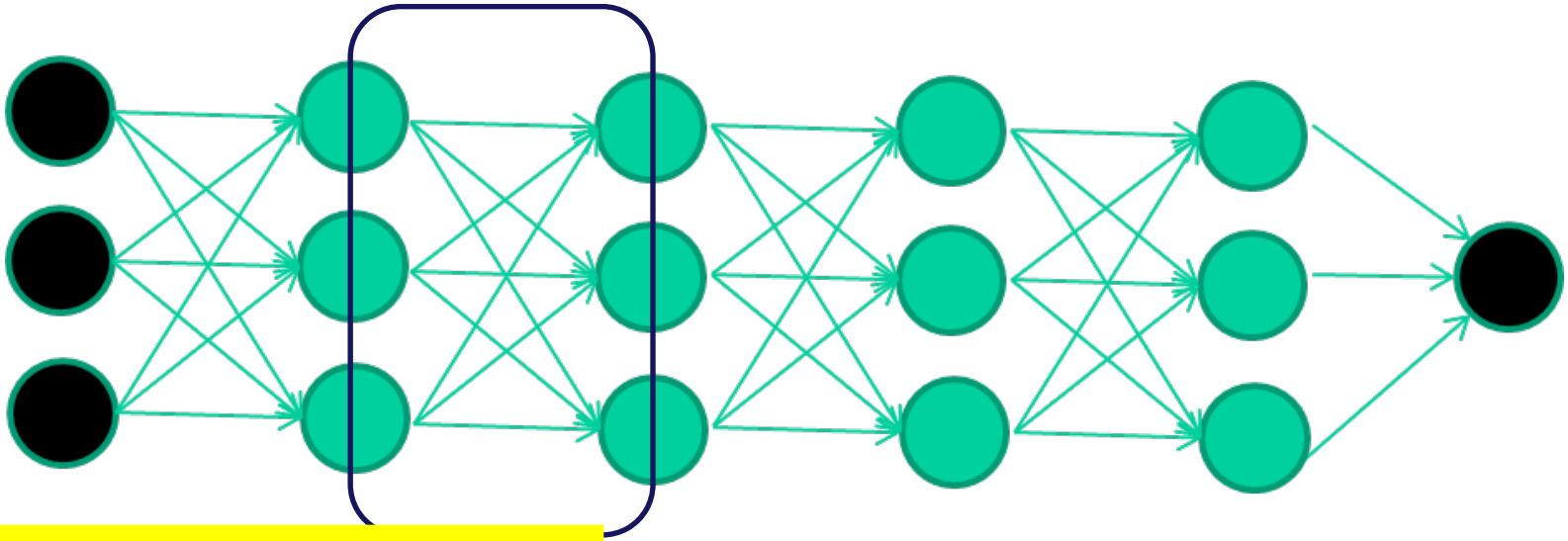


New method of training (basic info)



Train this layer

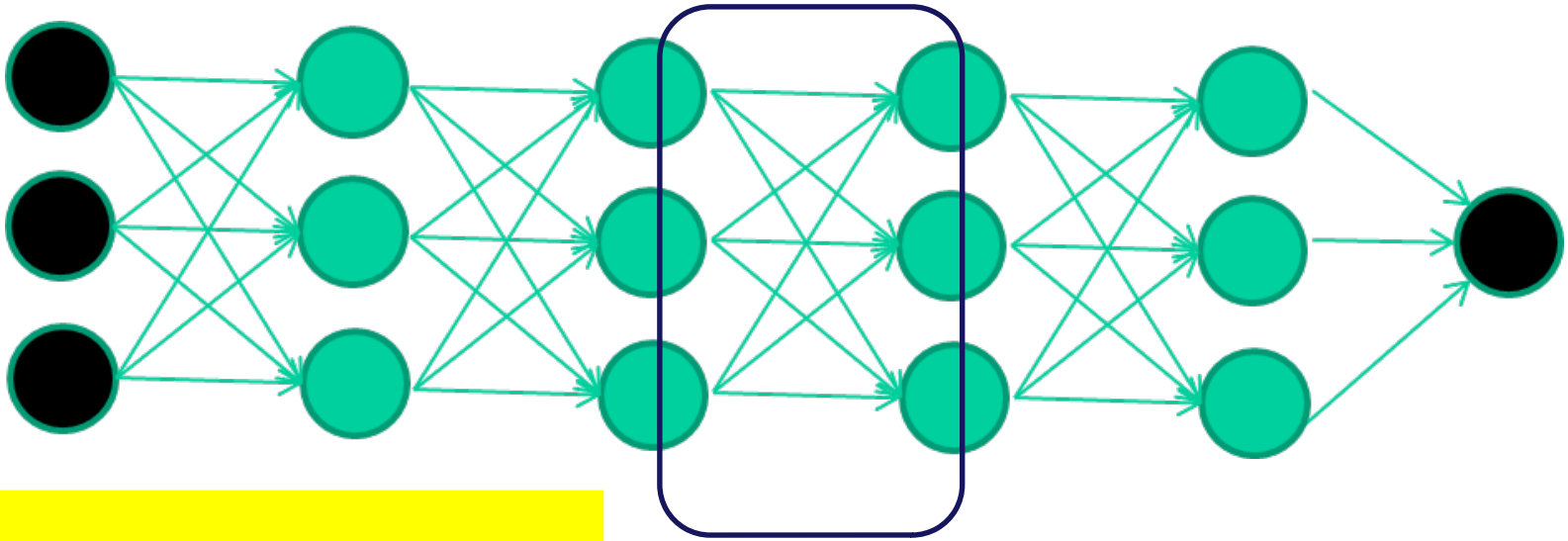
New method of training (basic info)



Train this layer

Then this

New method of training (basic info)

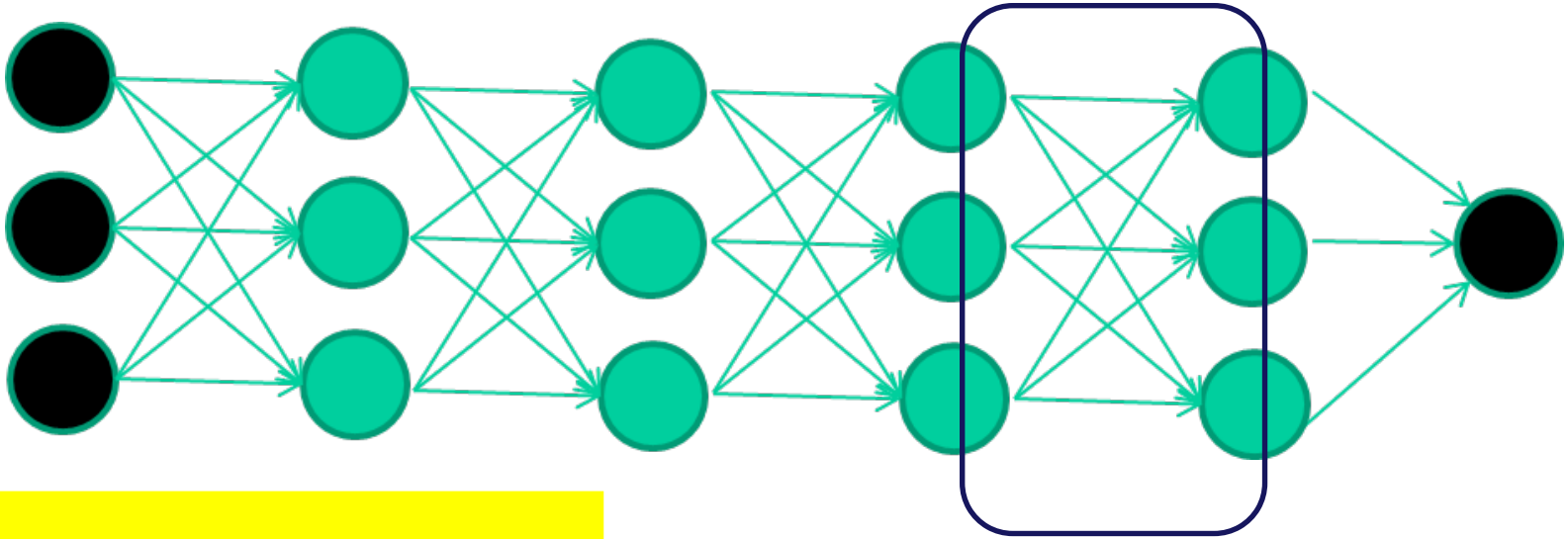


Trenuj tę warstwę

Then this

Then this

New method of training (basic info)



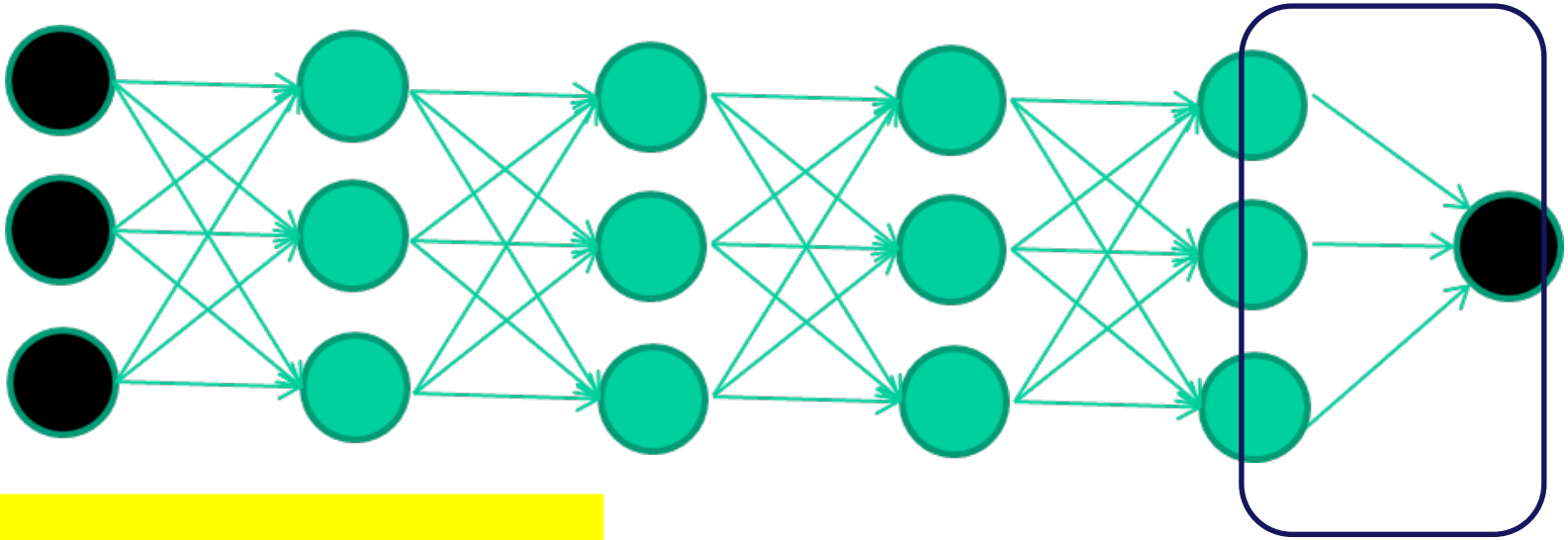
Train this layer

Then this

Then this

Then this

New method of training (basic info)



Train this layer

Then this

Then this

Then this

At the end this