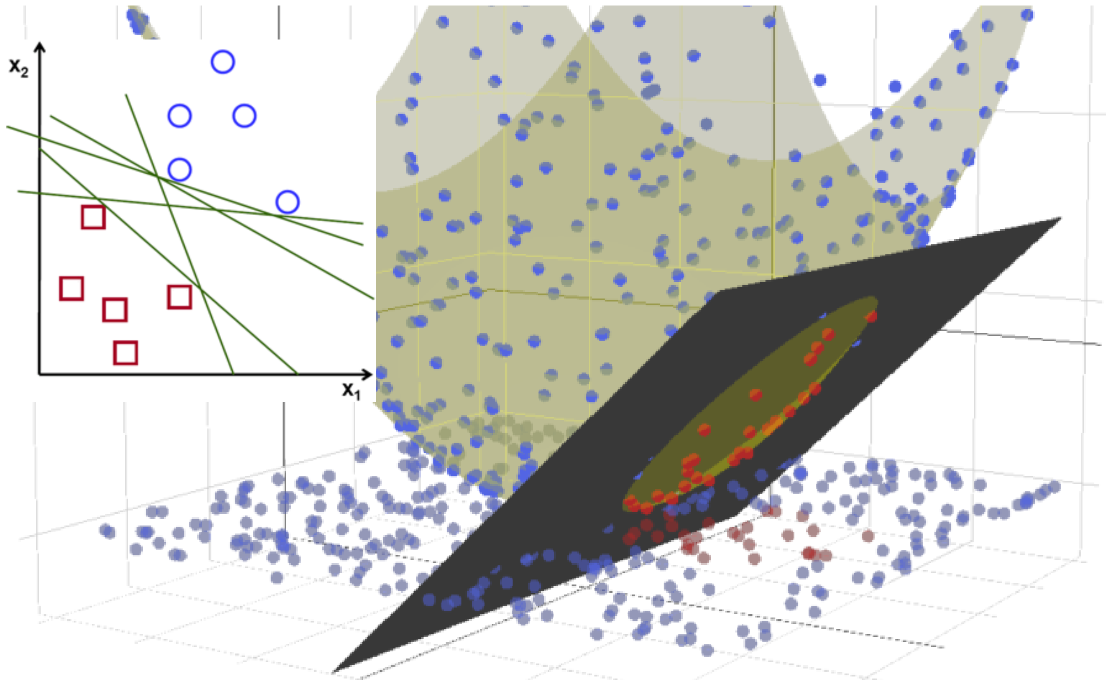
Machine learning
Lecture 4



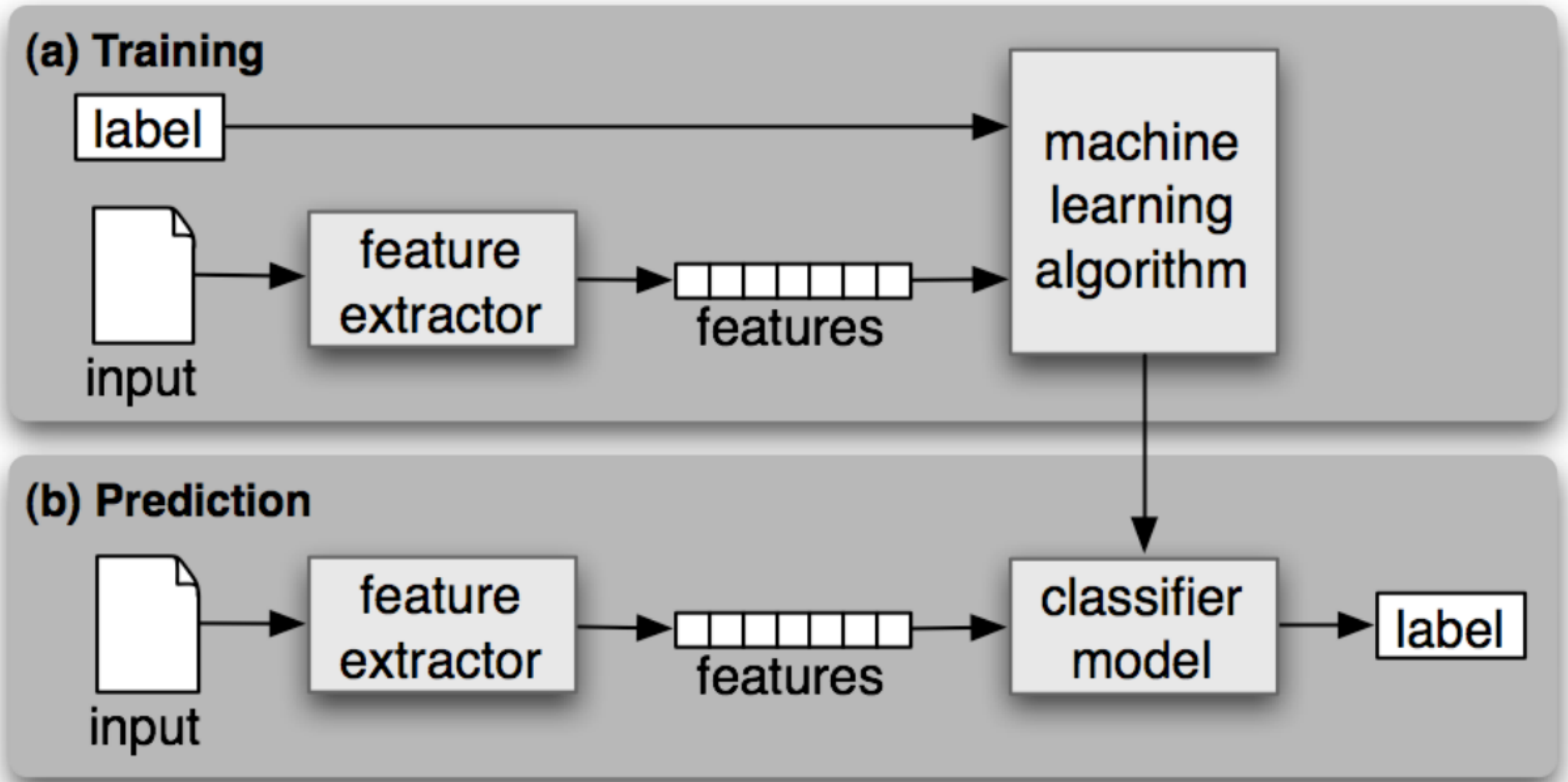
Marcin Wolter

IFJ PAN

21 March 2017

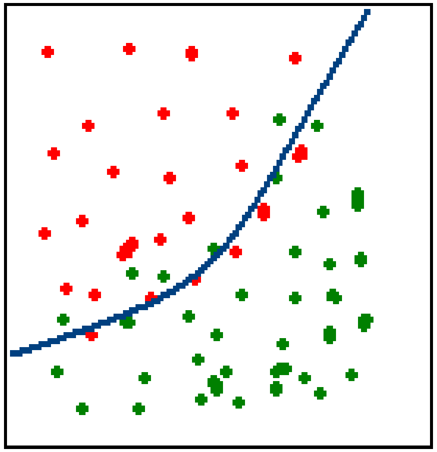
- Training - cross-validation.
- Optimalization of hyperparameters.
- Deep learning

Supervised training

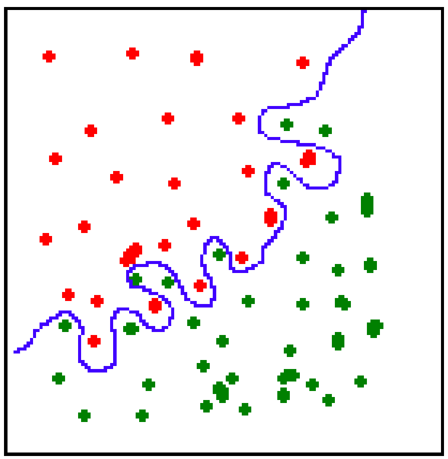


Overtraining

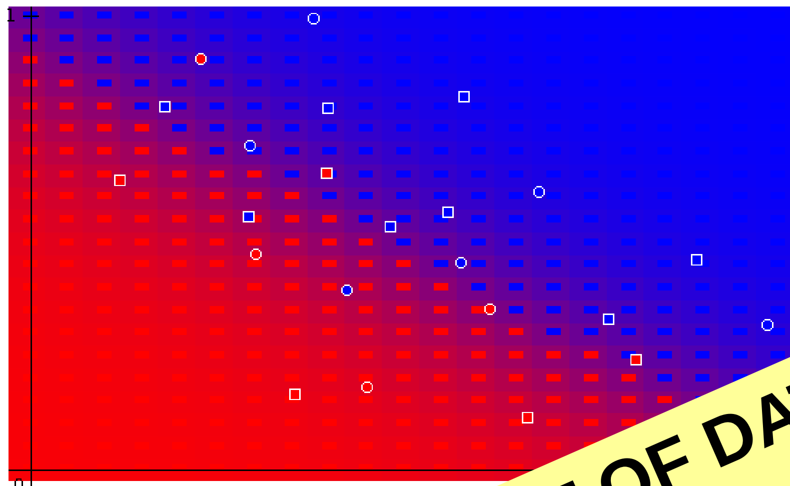
- **Overtraining** – algorithm “learns” the particular events, not the rules.
- <u>This effect appears for all ML algorithms.</u>
- Remedy – checking with another, independent dataset.

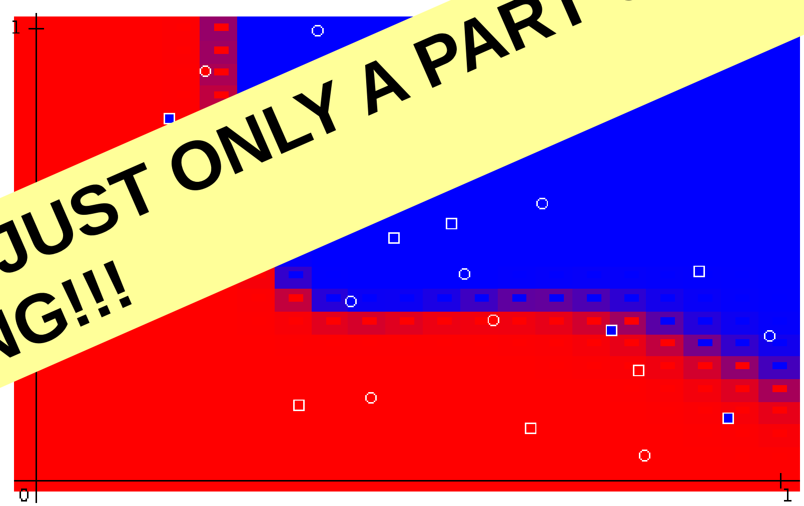


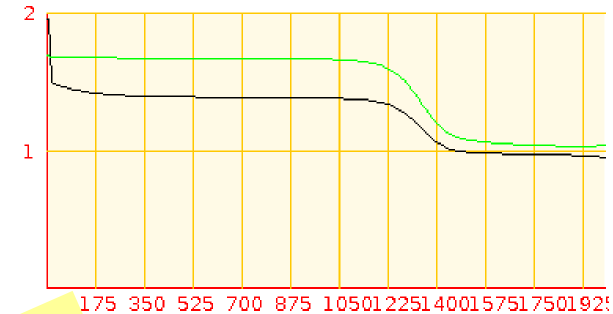
Correct



Overtraining

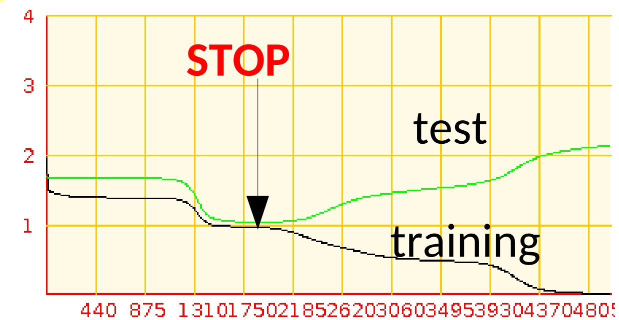






Training sample

Test sample



Example of using Neural Network.

BUT WE USE JUST ONLY A PART OF DATA FOR TRAINING!!!

How to train a ML algorithm?

- How to avoid overtraining while learning?
- Should we use one sample for training and another for validating?
- Then we increase the error – we use just a part of data for training.

- Second remark: to avoid ovetraining and find the performance of the trained algorithm we should use one more, third data sample to measure the final performance of the ML algorithm.

- How to optimize the hyperparameters of the ML algorithm (number of trrees and their depth for BDT, number of hidden layers, nodes for Neural Network)?

Validation

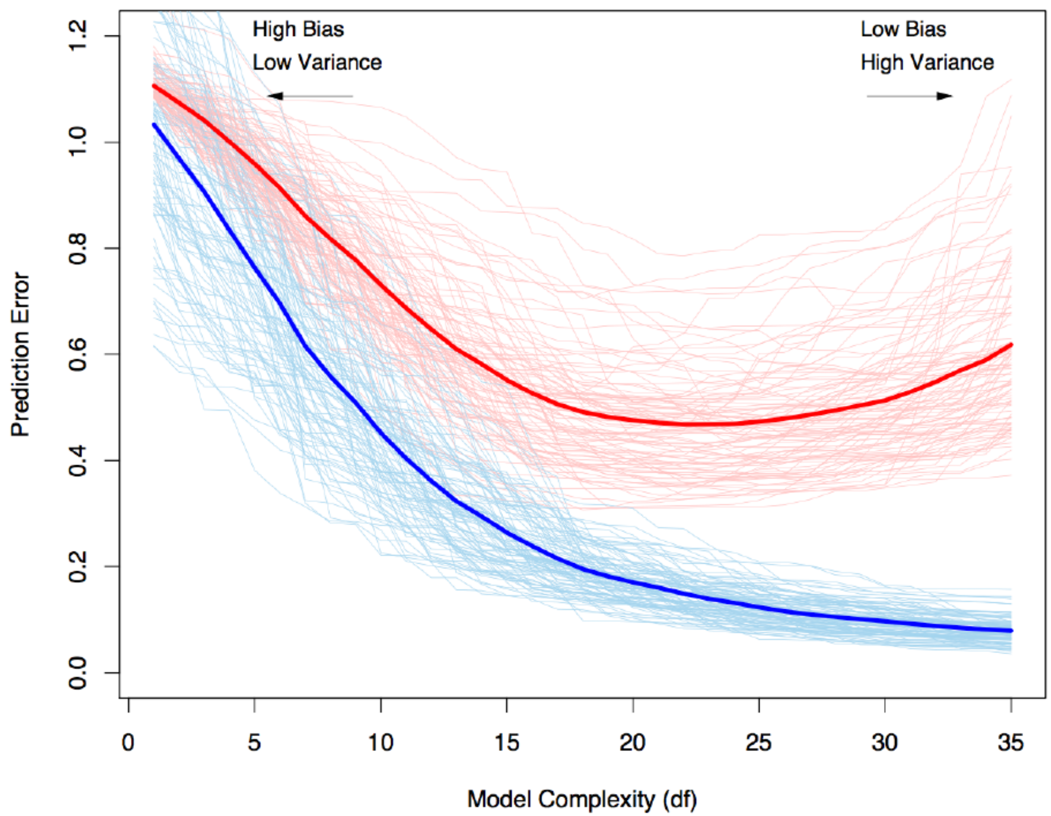


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\mathrm{err}}$*, while the light red curves show the conditional test error* $\mathrm{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\mathrm{E}[\overline{\mathrm{err}}]$.

Source: Elements of Statistical Learning

Cross-validation

- We have independant training sample L_n and a test sample T_m.
- Error level of the classifier $\hat{d}(\boldsymbol{x}) = \hat{d}(\boldsymbol{x};\ \mathcal{L}_n)$ built on the training sample L_n

$$\hat{e}_T = \frac{1}{m}\sum_{j=1}^{m} I\left(\hat{d}(\boldsymbol{X}_j^t; \mathcal{L}_n) \neq Y_j^t\right)$$

- Estimator using "recycled data" (the same data for training and for error calculation) is biased.
- Reduction of bias: for example division of data into two parts (training & validation) we use just a part of information only.
- **Cross-validation** – out of sample L_n we remove just one event j, train classifier, validate on single event j. We repeat n times and get the estimator:

$$\hat{e}_{CV} = \frac{1}{n}\sum_{j=1}^{n} I\left(\hat{d}(\boldsymbol{X}_j; \mathcal{L}_n^{(-j)}) \neq Y_j\right)$$

- We get an estimator, which is unbiased (in limit of huge n), but CPU demanding, with bigger variation than e_T.

Cross-validation

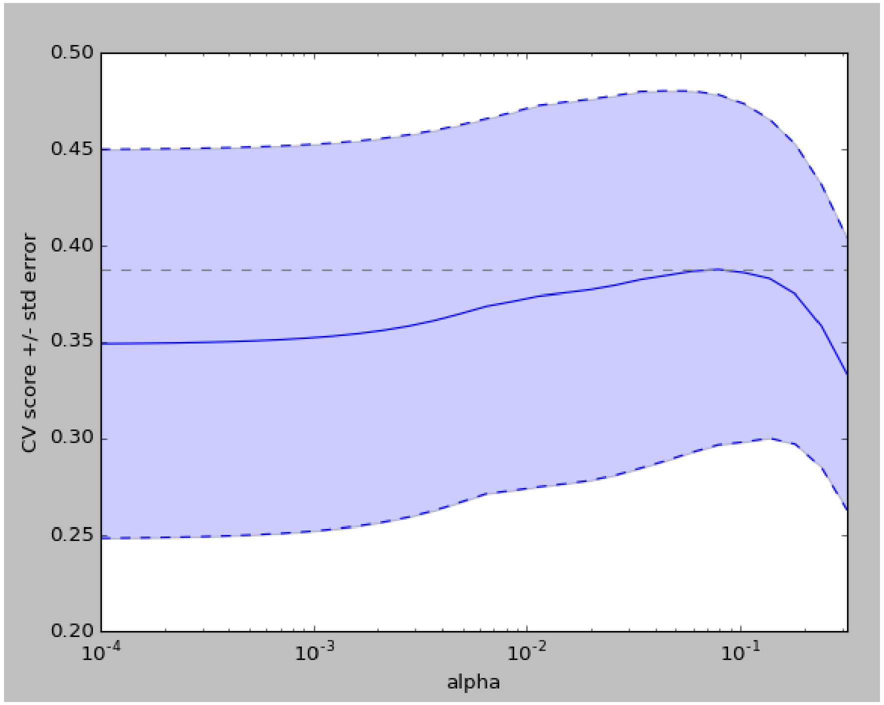
- Intermediate solution – *v-fold cross-validation*
- The sample is divided into v subsamples, v-1 of them we use for training, the one for validation. Then the procedure is repeated with other subsamples and the procedure is repeated v tir

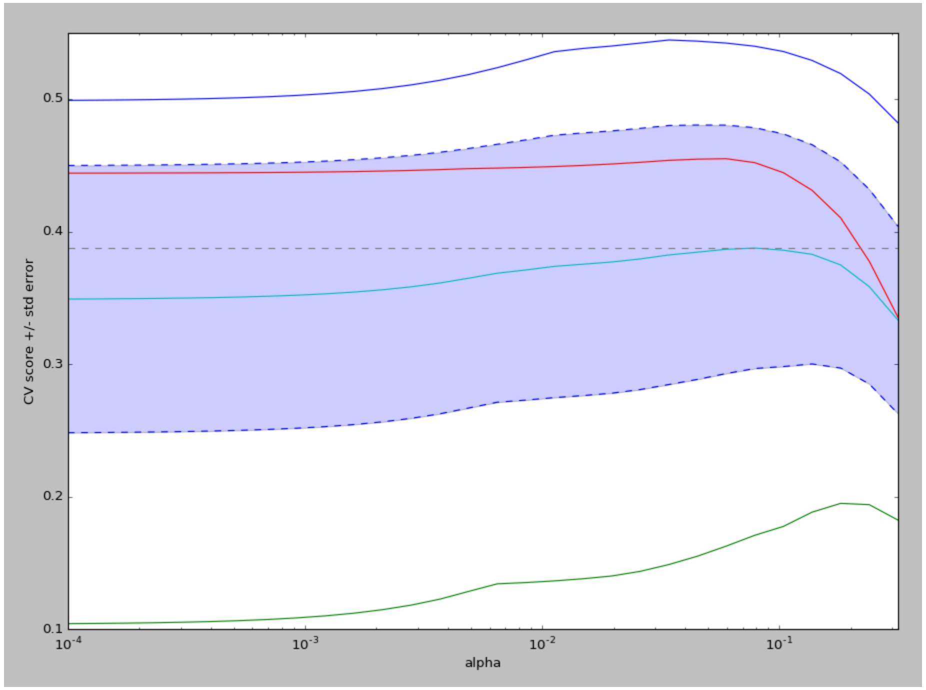
$$\hat{e}_{vCV} = \frac{1}{n}\sum_{i=1}^{v}\sum_{j=1}^{n} I(\boldsymbol{Z}_j \in \tilde{\mathcal{L}}_n^{(i)}) I\left(\hat{d}(\boldsymbol{X}_j; \tilde{\mathcal{L}}_n^{(-i)}) \neq Y_j\right)$$

- Smaller CPU usage comparing to ross-validation.
- Recommended v ~ 10 .
- While choosing the classifier (for example tuning hyperparameters), we should choose the classifier, which gives the smallest classification error.W

Walidacja krzyżowa

- 4-times folding
- Finding a dependence of CV from alpha (medical data)
- We can draw the mean and a standard deviation. In the next plot we draw the dependence for each folding.
- As a result we can estimate an error of CV.





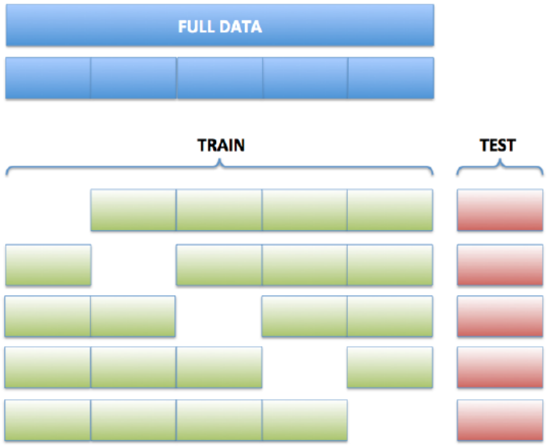
Model performance

Test Error

- Partition the original data (randomly) into a training set and a test set. (e.g. 70/30)
- Train a model using the training set and evaluate performance (a single time) on the test set.

K-fold Cross-validation

- Train & test K models as shown.
- Average the model performance over the K test sets.
- Report cross-validated metrics.



Performance Metrics

- Regression: R^2, MSE, RMSE
- Classification: Accuracy, F1, H-measure, Log-loss
- Ranking (Binary Outcome): AUC, Partial AUC

https://www.slideshare.net/0xdata/top-10-data-science-practitioner-pitfalls

Train vs Test vs Valid

Training Set vs.
Validation Set vs.
Test Set

- If you have “enough” data and plan to do some model tuning, you should really partition your data into three parts — Training, Validation and Test sets.
- There is **no general rule** for how you should partition the data and it will depend on how strong the signal in your data is, but an example could be: 50% Train, 25% Validation and 25% Test

Validation is for
Model Tuning

- The validation set is used **strictly for model tuning** (via validation of models with different parameters) and the test set is used to make a final estimate of the generalization error.

Hyperparameter optimization

- Nearly each ML method has few hyperparameters (structure of the Neural Net etc).
- They should be optimized for a given problem.
- **Task: for a given data sample find a set of hyperparameters, that the estimated error of the given method is minimized.**
- Looks like a typical minimization problem (fitting like), but:
  - Getting each measurement is costly
  - High noise
  - We can get the value of the minimized function (so our error) in the pont **x** of the hyperparameter space, but we can't get the differential.

Optimization of hyperparameters

- How to optimize:
  - „Grid search” - scan over all possible values of parameters.
  - „Random search”
  - Some type of fitting…
- Popular method is the **„bayesian optimization”**
  - Build the probability model
  - Take „a priori” distributions of parameters
  - Find, for which point in the hyperparameter space you can maximally improve your model
  - Find the value of error
  - Find the „a posteriori” probability distribution
  - Repeat

How does it work in practice?

- Straight line fitting

  $y(x, w) = w_0 + w_1 x$ fit to the data.

1) Gaussian prior, no data used
2) First data point. We find the likelihood based on this point (left plot)and multiply: priori*likelihood. We get the posterior distribution (right plot).
3) We add the second point and repeat the procedure.
4) Adding all the points one by one.

Remark: data are noisy.

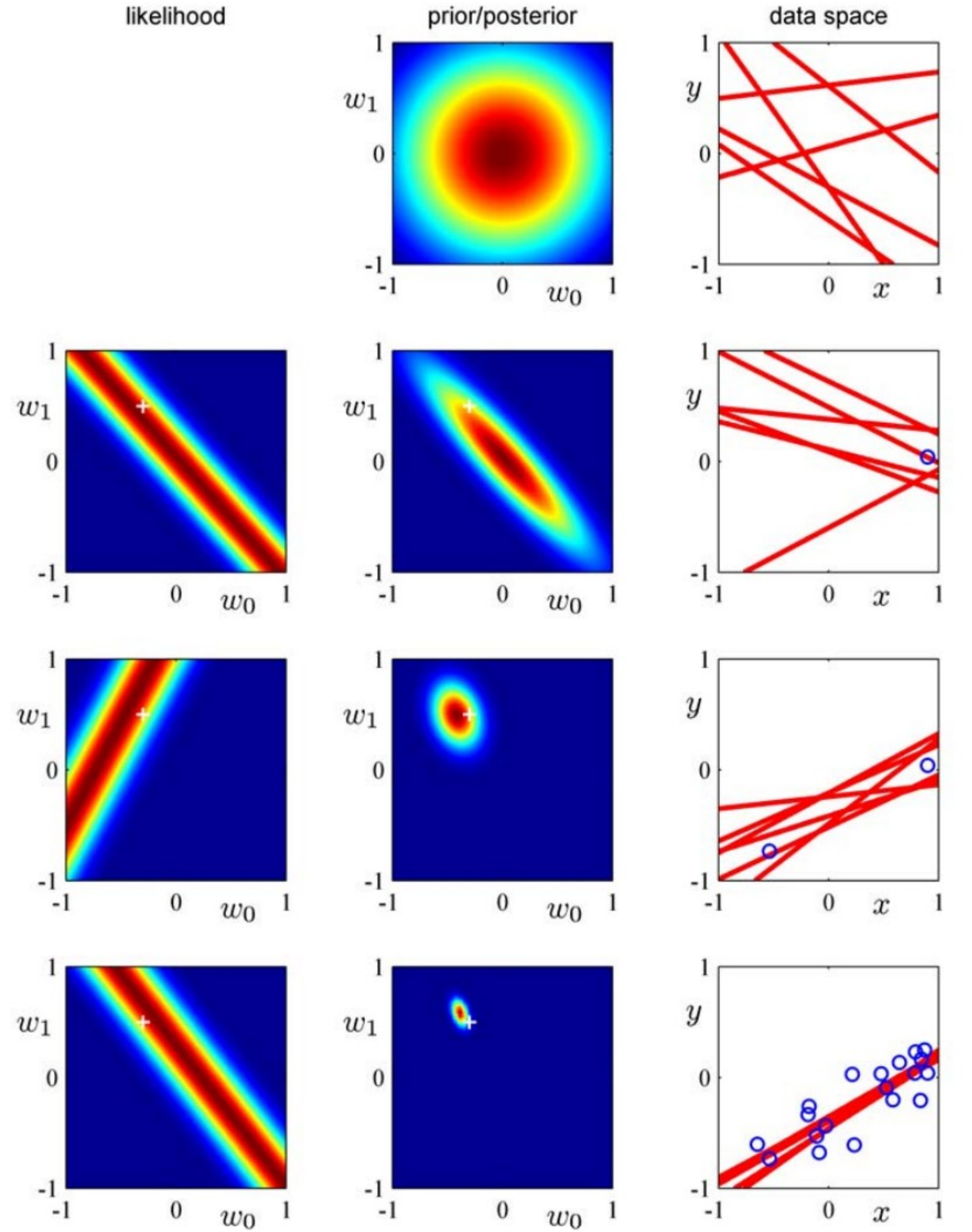
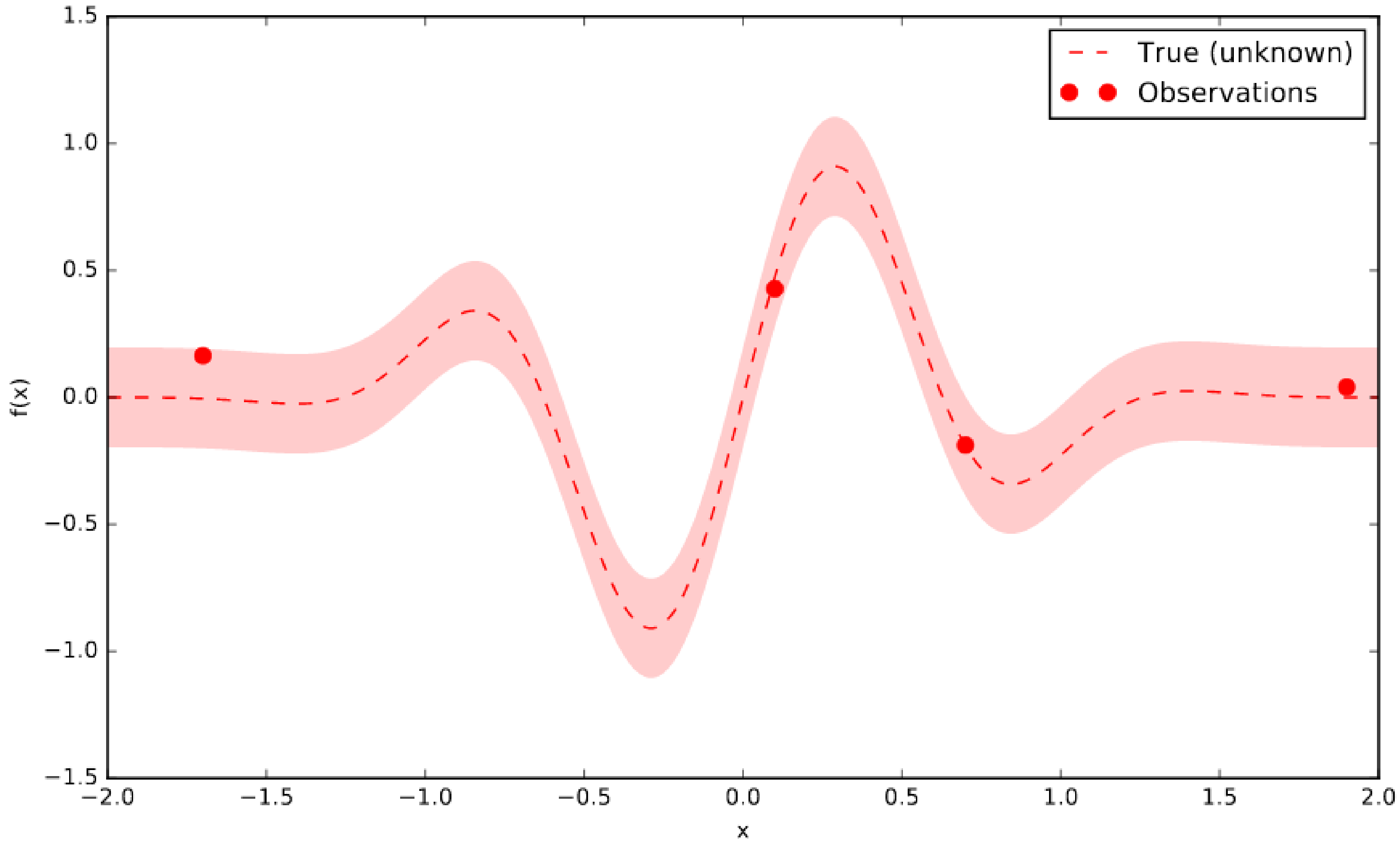


Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, \mathbf{w}) = w_0 + w_1 x$. A detailed description of this figure is given in the text.

Starting point

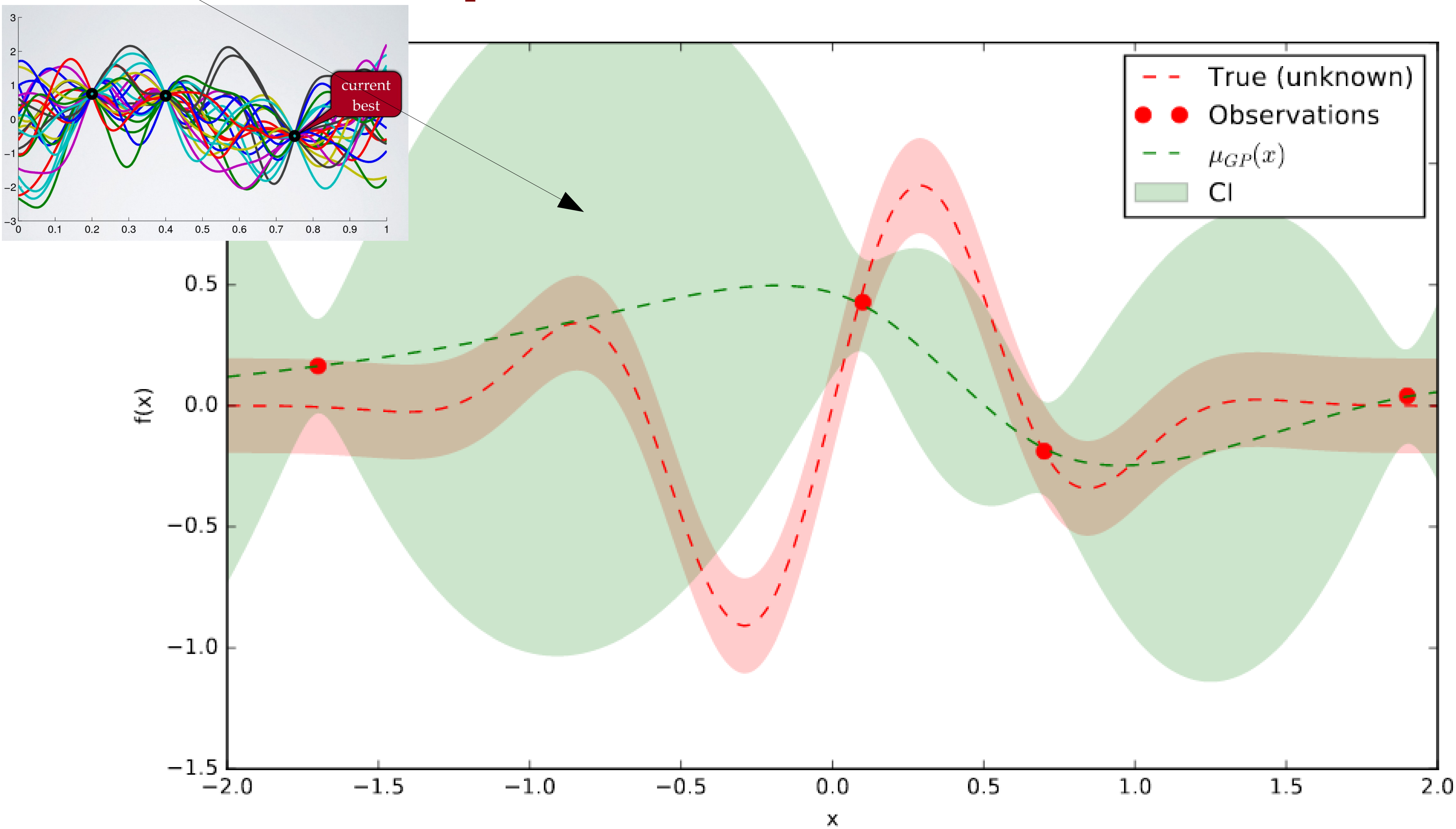


Unknown function (with noise), four observations.
Where should we do the next costly probing?

See this tutorial

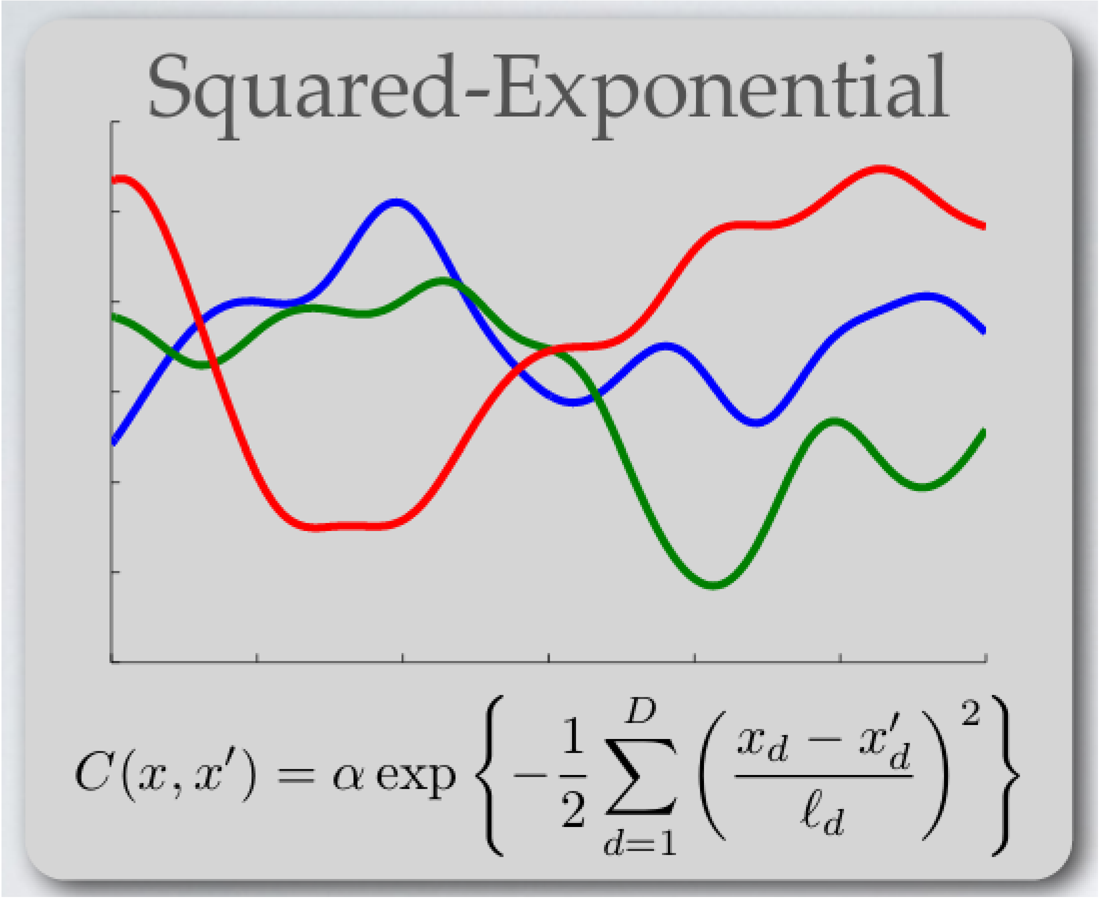
A posteriori distribution

A set of functions



The a posteriori distribution of possible functions, which could generate the observed data points.

A posteriori functions – Gaussian Processes (GP)



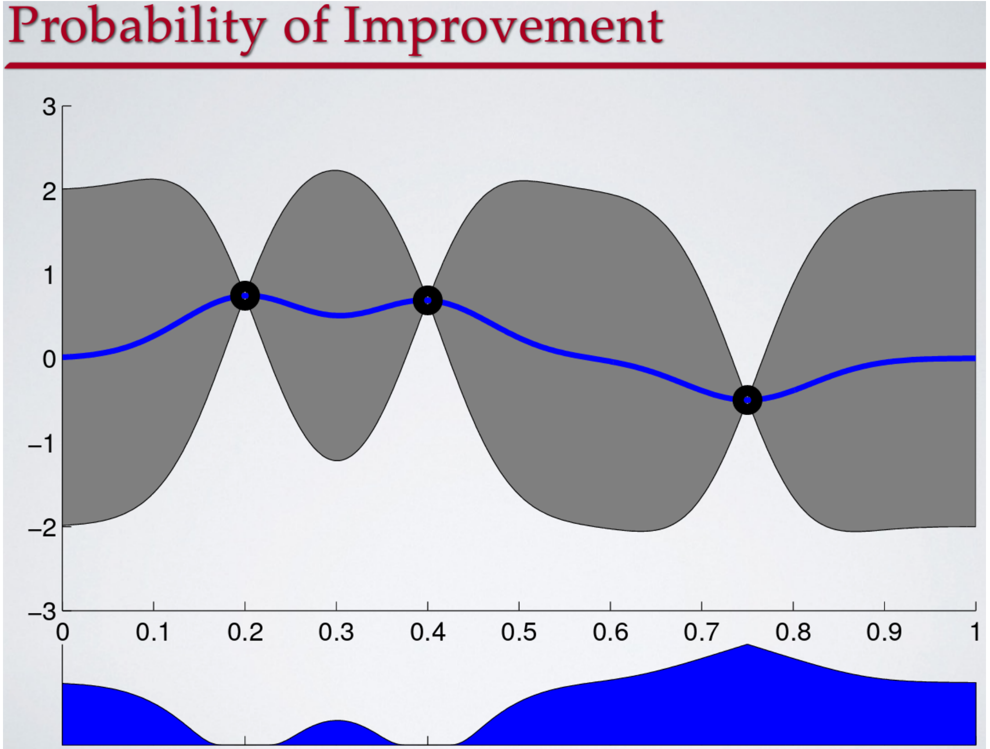
These functions should be somehow parametrized, for example these could be Gaussian functions.

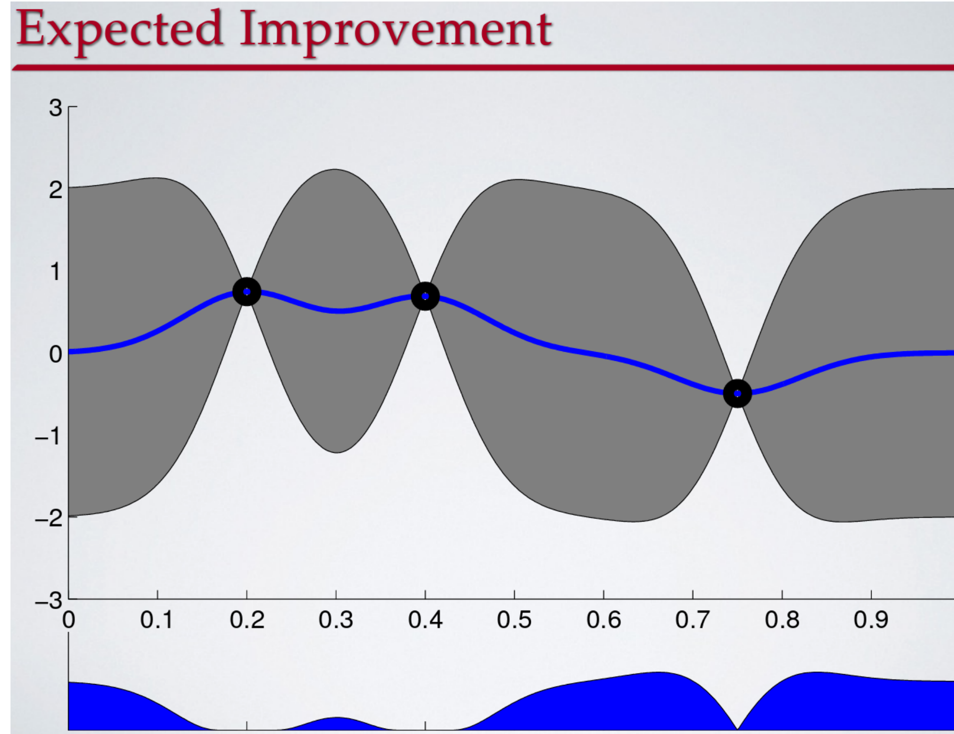
Acquisition function

- Posterior GP (Gaussian Processes) give us the mean of GP functions $\mu(x)$ and their expected variation $\sigma^2(x)$.
  - Exploration – searching for huge variation
  - Exploitation – search for a smallest/greatest (depends on sign and convention) value of mean $\mu(x)$
- The acquisition policy has to balance these two approaches:
  - Probability of Improvement (Kushner 1964):
    - $a_{PI}(x) = \Phi(\gamma(x))$

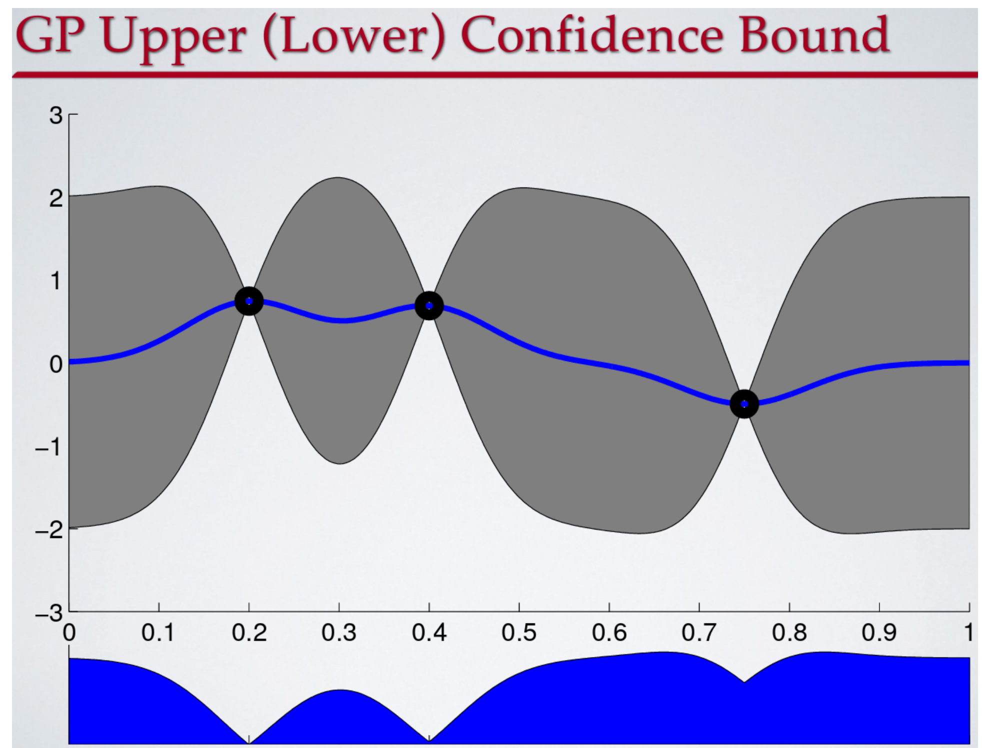
$$\gamma(x) = \frac{f(x_{\text{best}}) - \mu(x)}{\sigma(x)}$$

  - Expected Improvement (Mockus 1978)
  - GP Upper Confidence Bound (Srinivas et al. 2010):
    - $a_{LCB}(x) = \mu(x) - \kappa\sigma(x)$



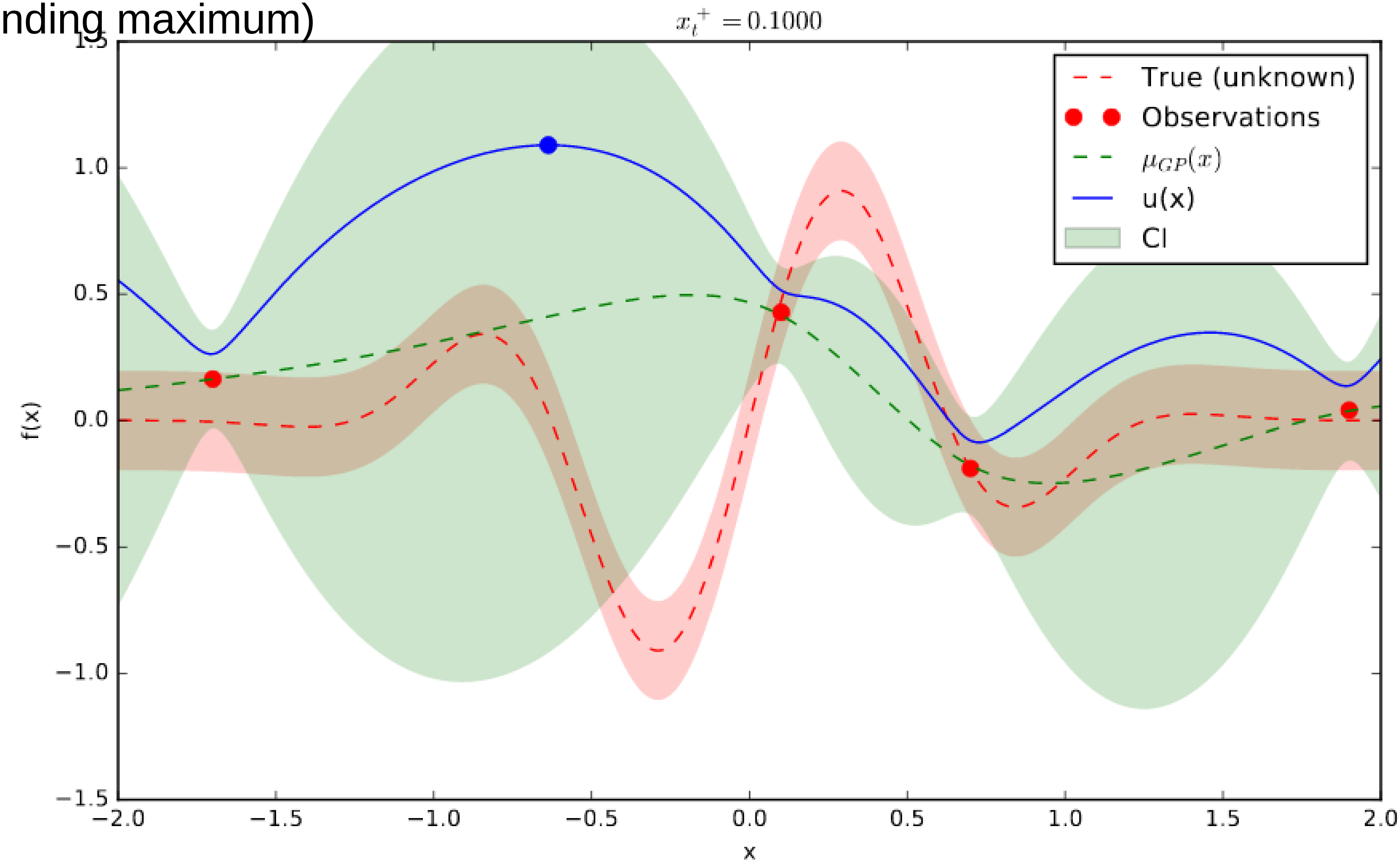


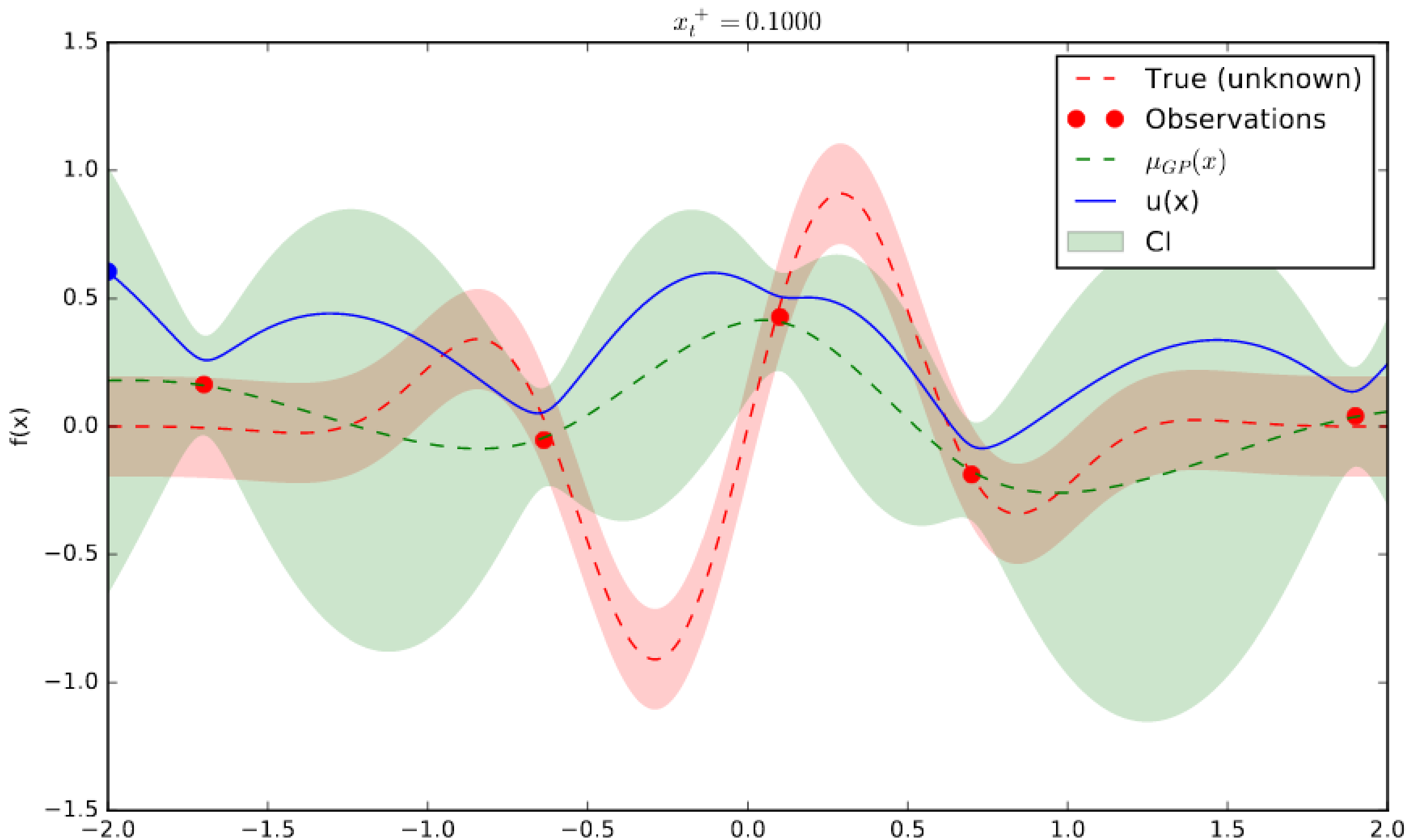
These functions are quite similar...



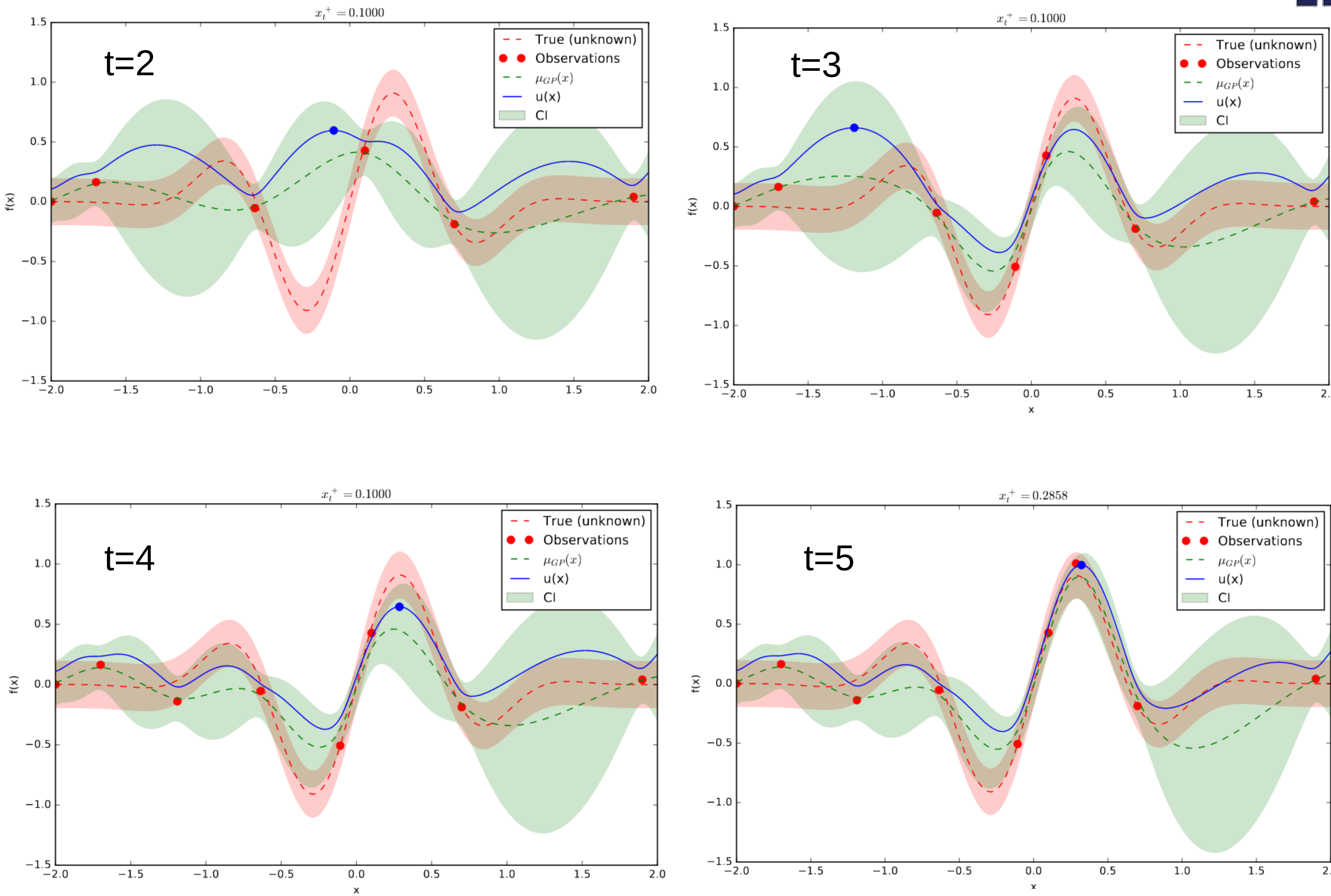
We choose next x

u(x) – acqusition function
(finding maximum)





Dokonujemy próbkowania i powtarzamy procedurę...



$x_t^+ = 0.1000$
t=2
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
$x_t^+ = 0.1000$
t=3
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x
$x_t^+ = 0.1000$
t=4
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x
$x_t^+ = 0.2858$
t=5
True (unknown)
Observations
$\mu_{GP}(x)$
u(x)
CI
f(x)
x

Limitations

- Bayesian optimization depends on the parameters chosen
- On the acquisition function
- On the prior selected....
- It’s sequential.
- There are alternative methods, which can be done in parallel (like Random Search or Tree of Parzen Estimators (TPE) used by the HyperOpt package https://github.com/hyperopt/hyperopt).

Implementations

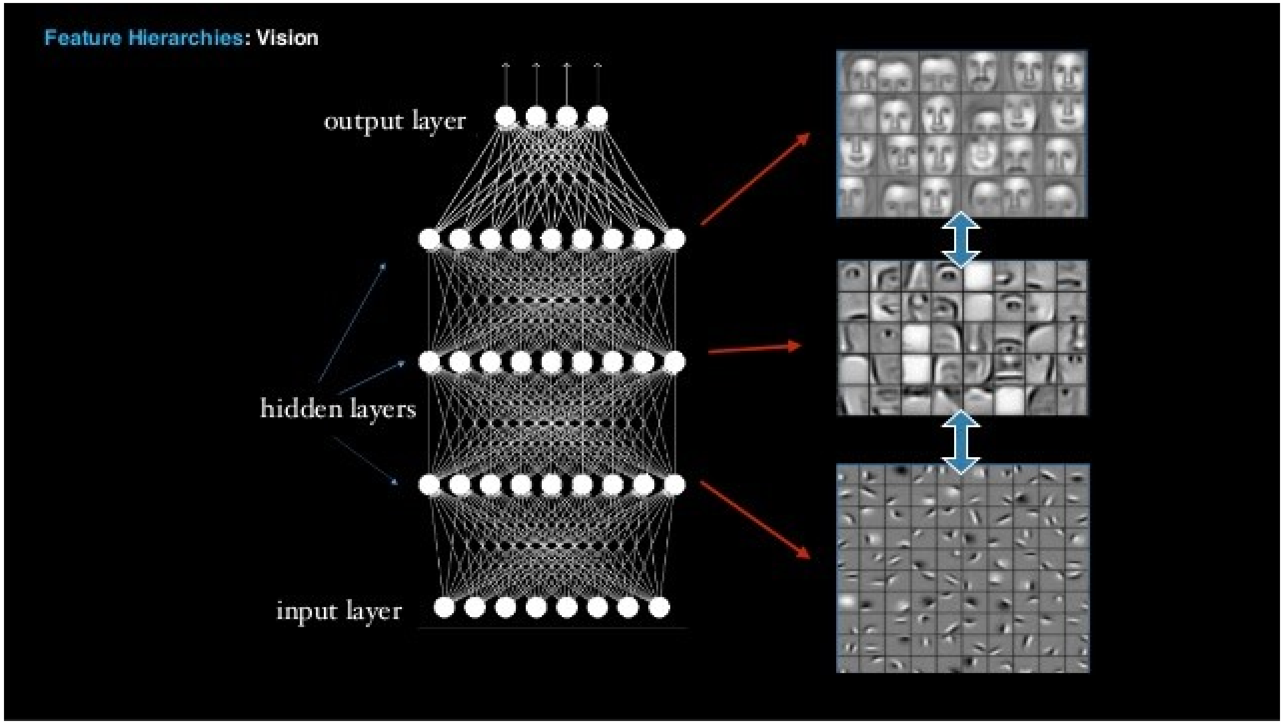
- **Python**
  - Spearmint https://github.com/JasperSnoek/spearmint
  - GPyOpt https://github.com/SheffieldML/GPyOpt
  - RoBO https://github.com/automl/RoBO
  - Scikit-optimize https://github.com/MechCoder/scikit-optimize
- **C++**
  - MOE https://github.com/yelp/MOE

Articles

- Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599.
- Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE, 104(1):148–175.
- Nice tutorial:

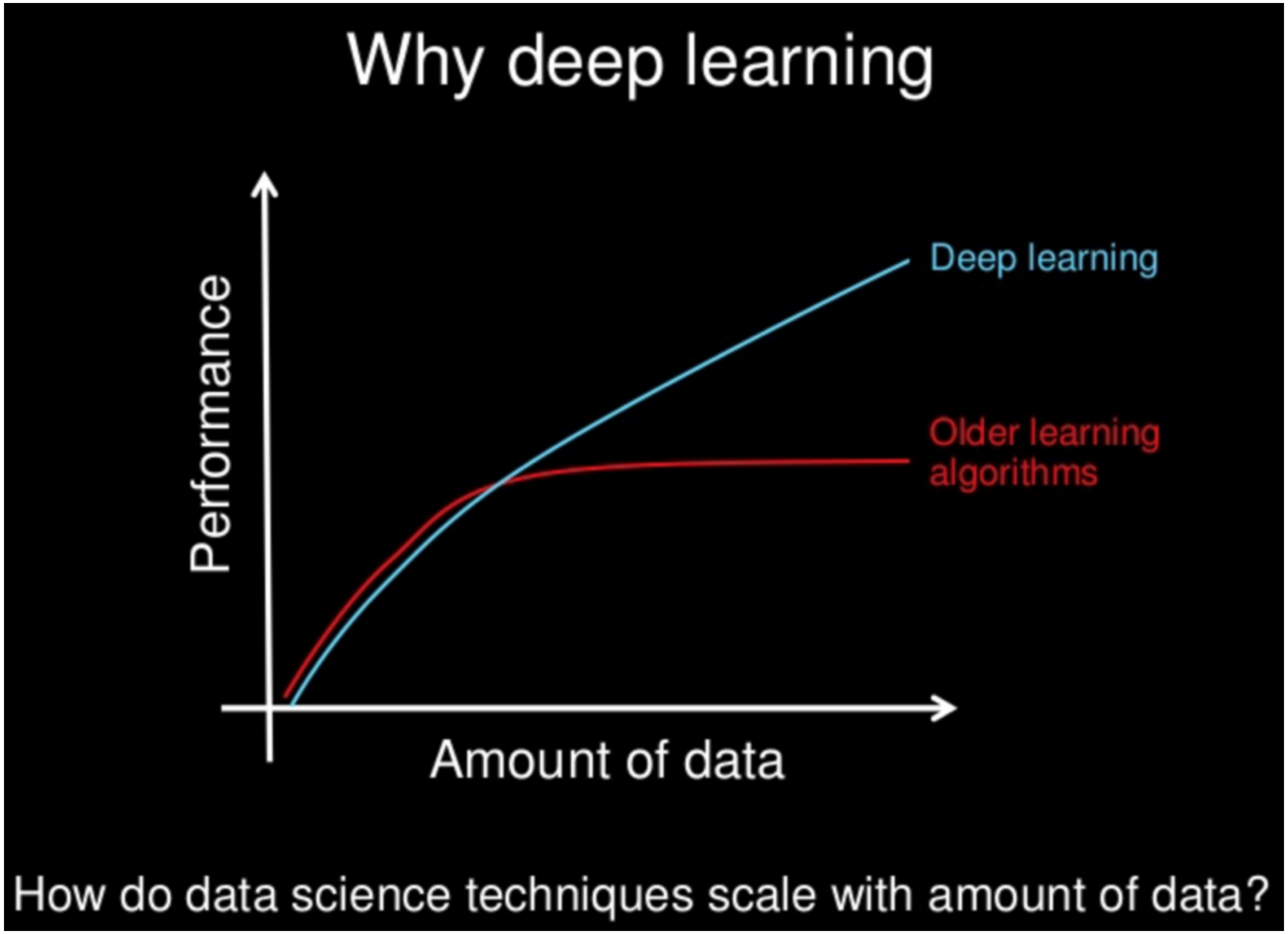
  https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

Deep Learning



1. What does “deep learning” mean?

2. Why does it give better results than other methods in pattern recognition, spech recognition and others?



Short answer:

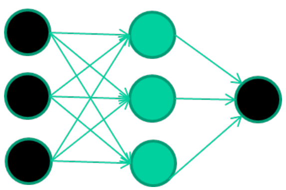
‘Deep Learning’ - using a neural network with many hidden layers

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???

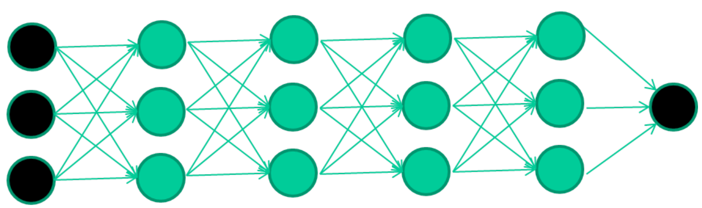
We always had good algorithms to train NN with one or two hidden layers.

But they were failing for more layers



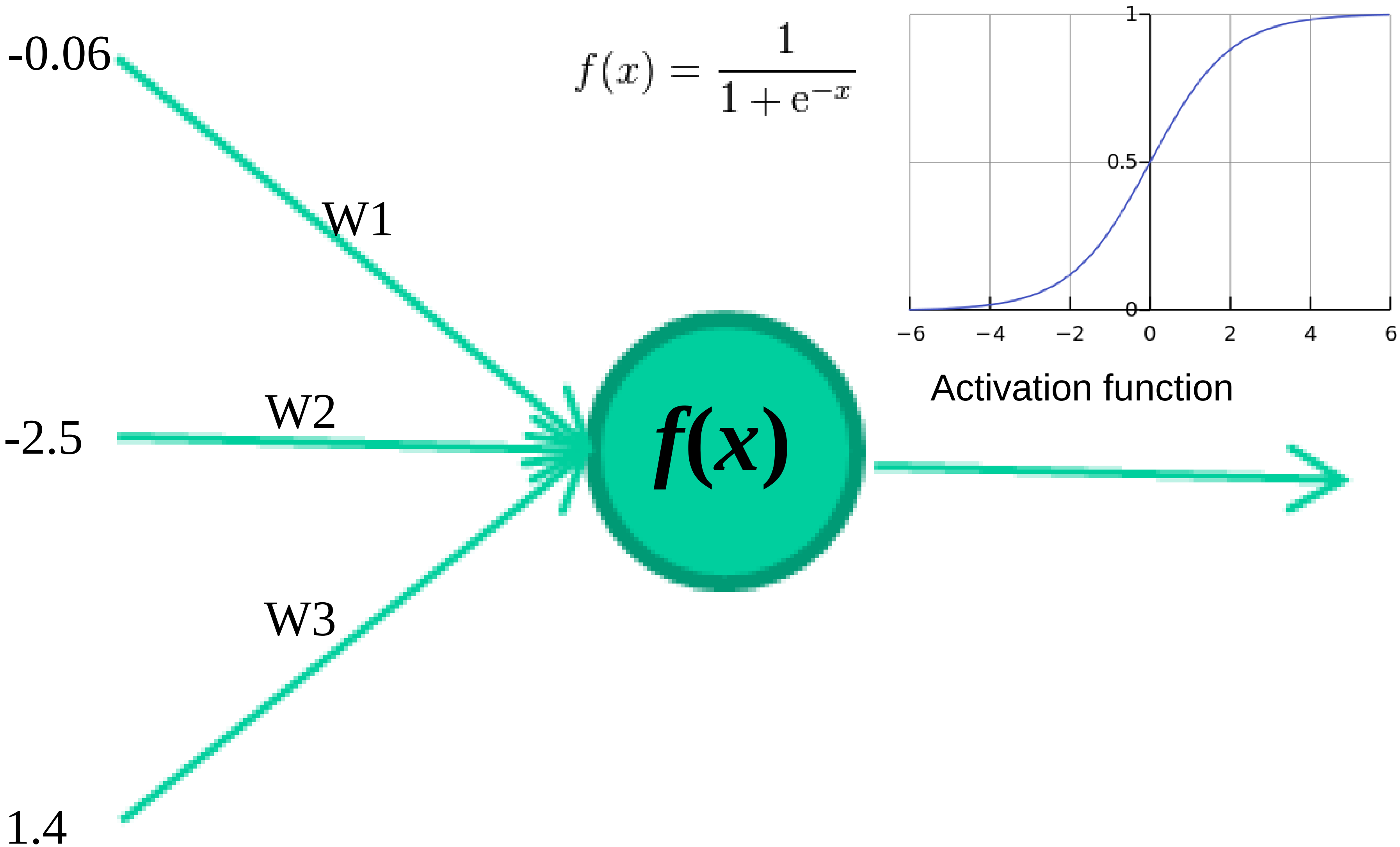
NEW: algorithms for training deep networks

huge computing power



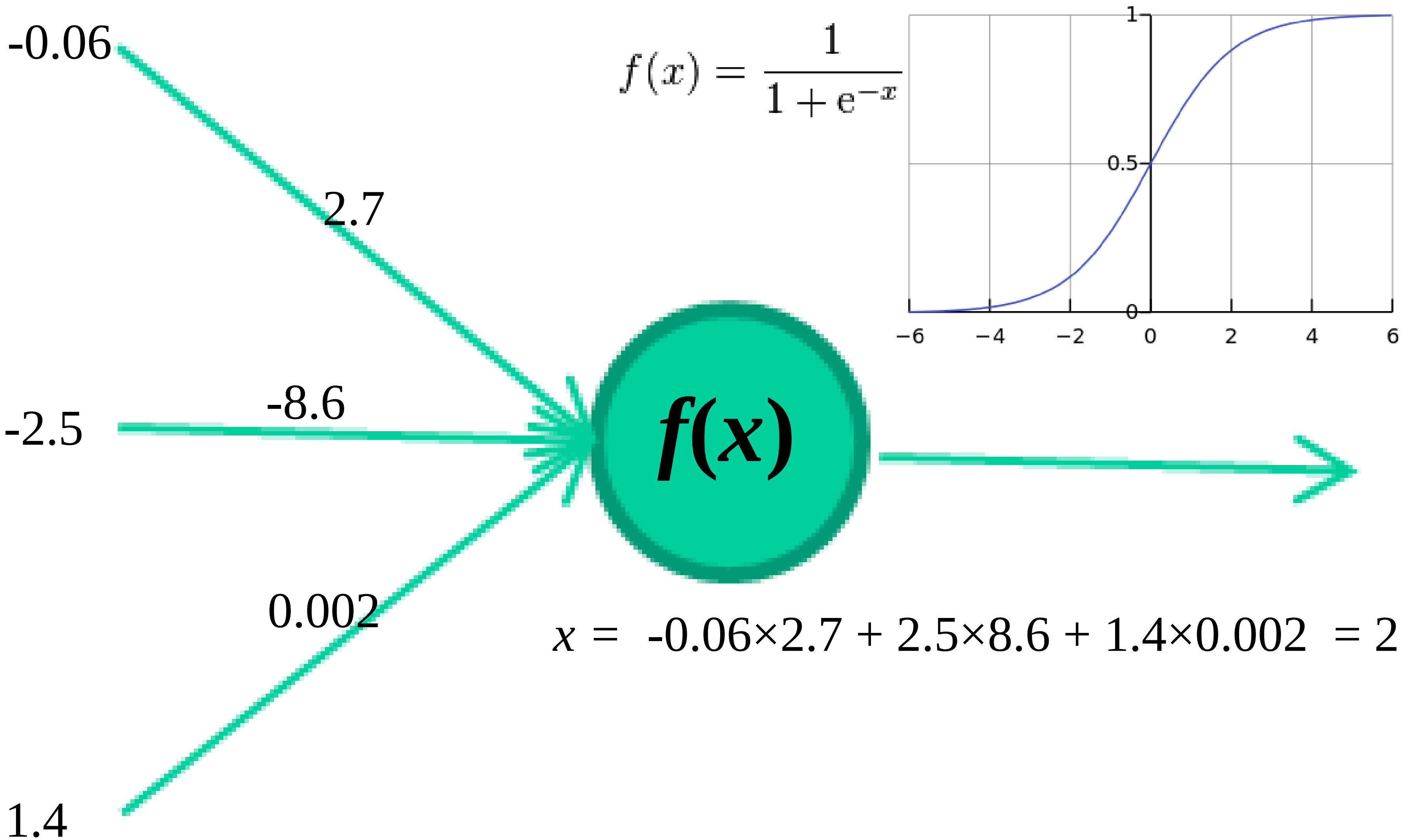
How do we train a NN

$$f(x) = \frac{1}{1+\mathrm{e}^{-x}}$$



Activation function

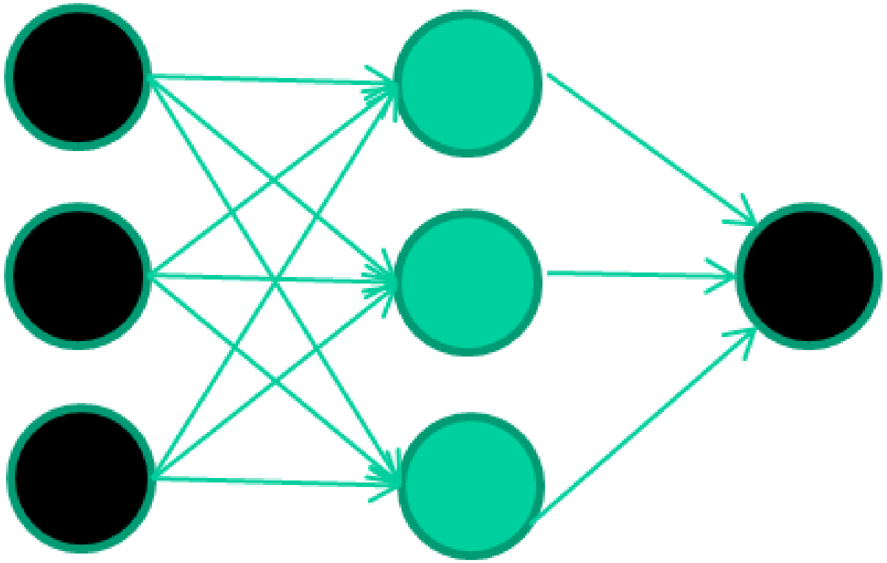
$$f(x) = \frac{1}{1+e^{-x}}$$



$$x = -0.06\times2.7 + 2.5\times8.6 + 1.4\times0.002 = 21.34$$

NN training

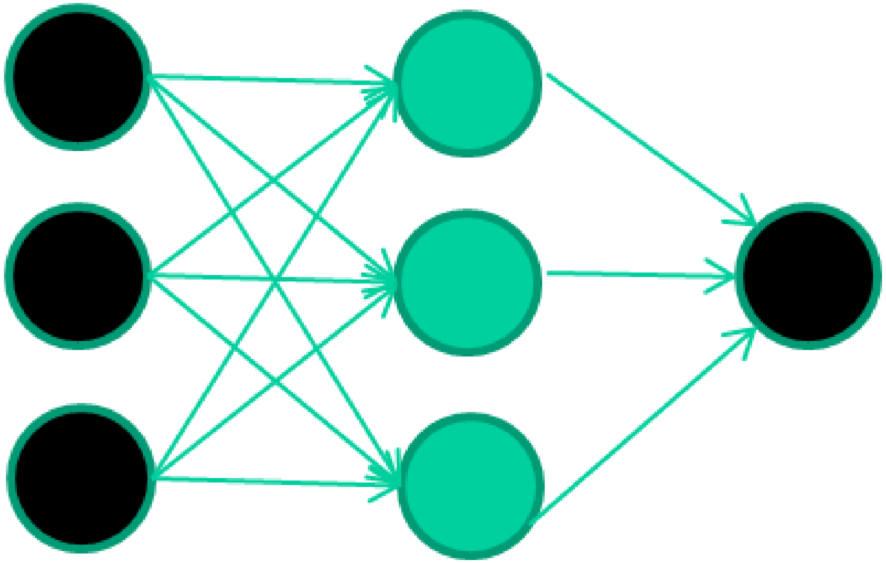
Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



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 …			

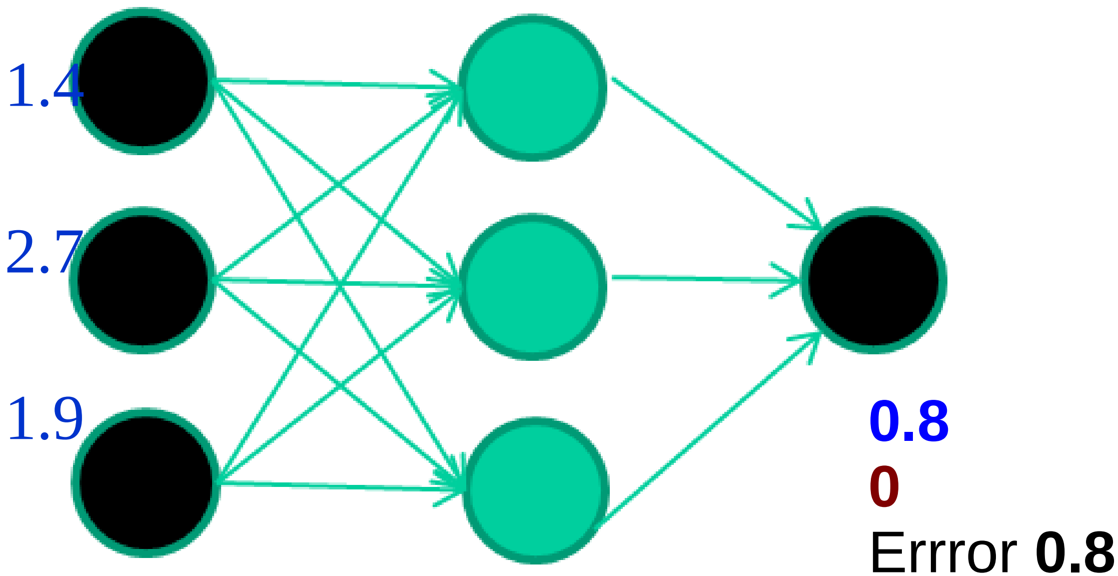
Initialization with random weights



Training data

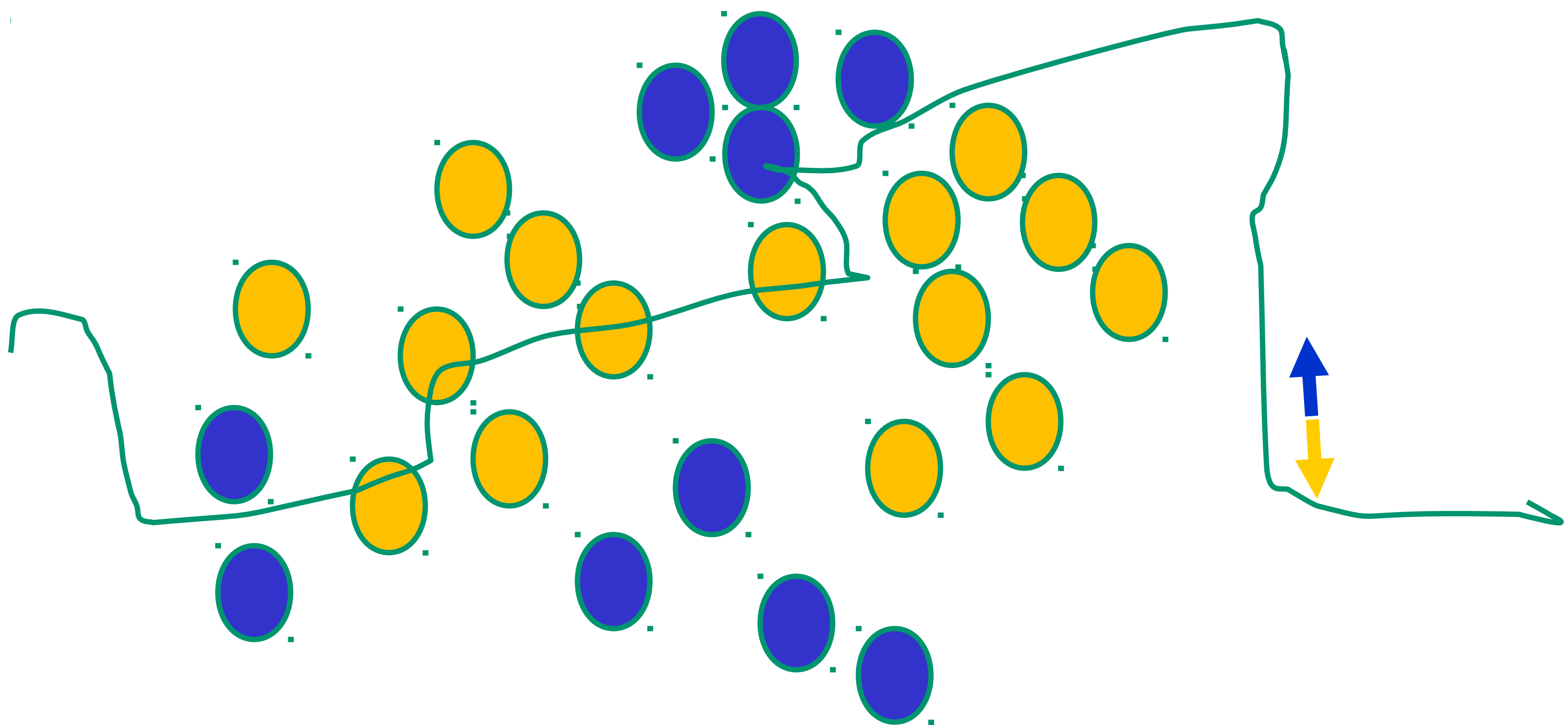
Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

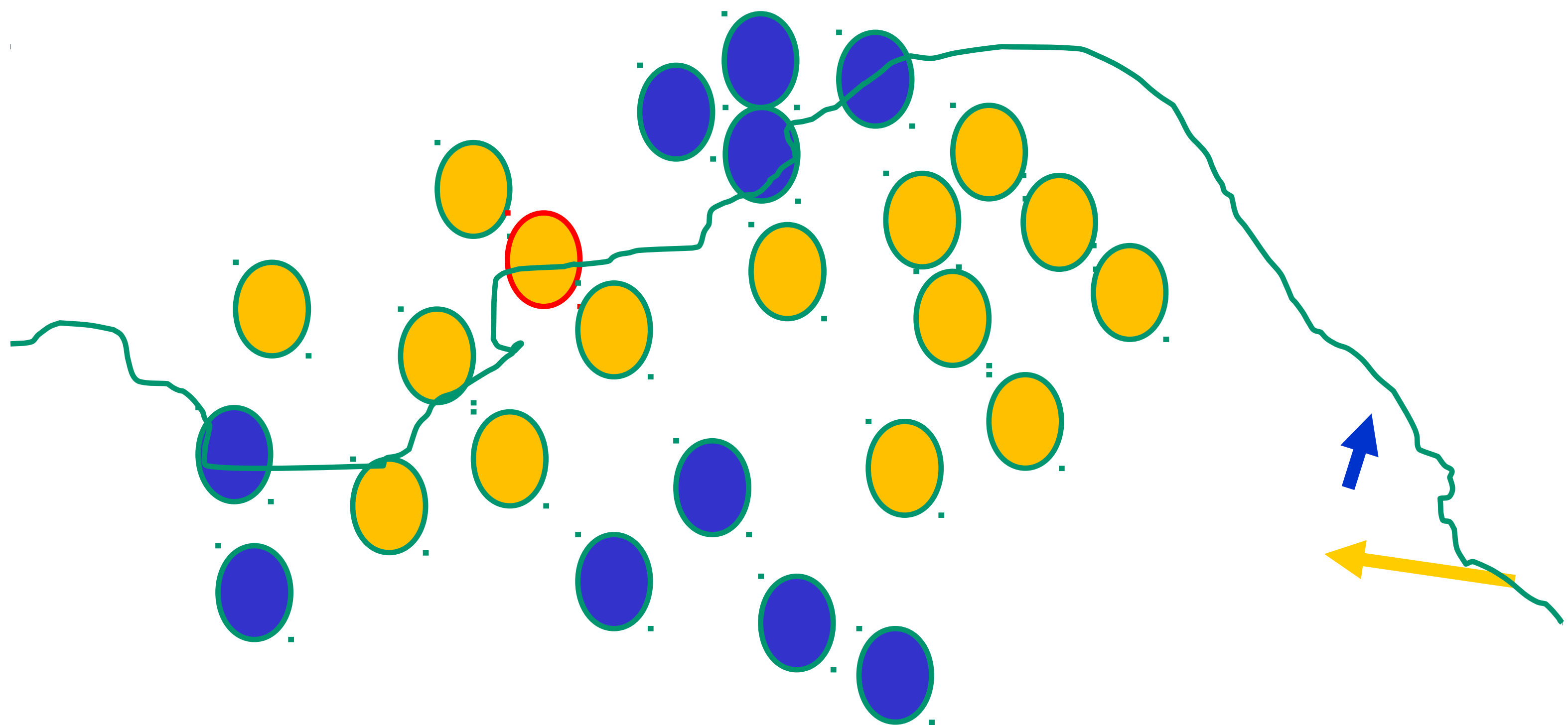
Reading data
Processing them by network
Result compared with the true value

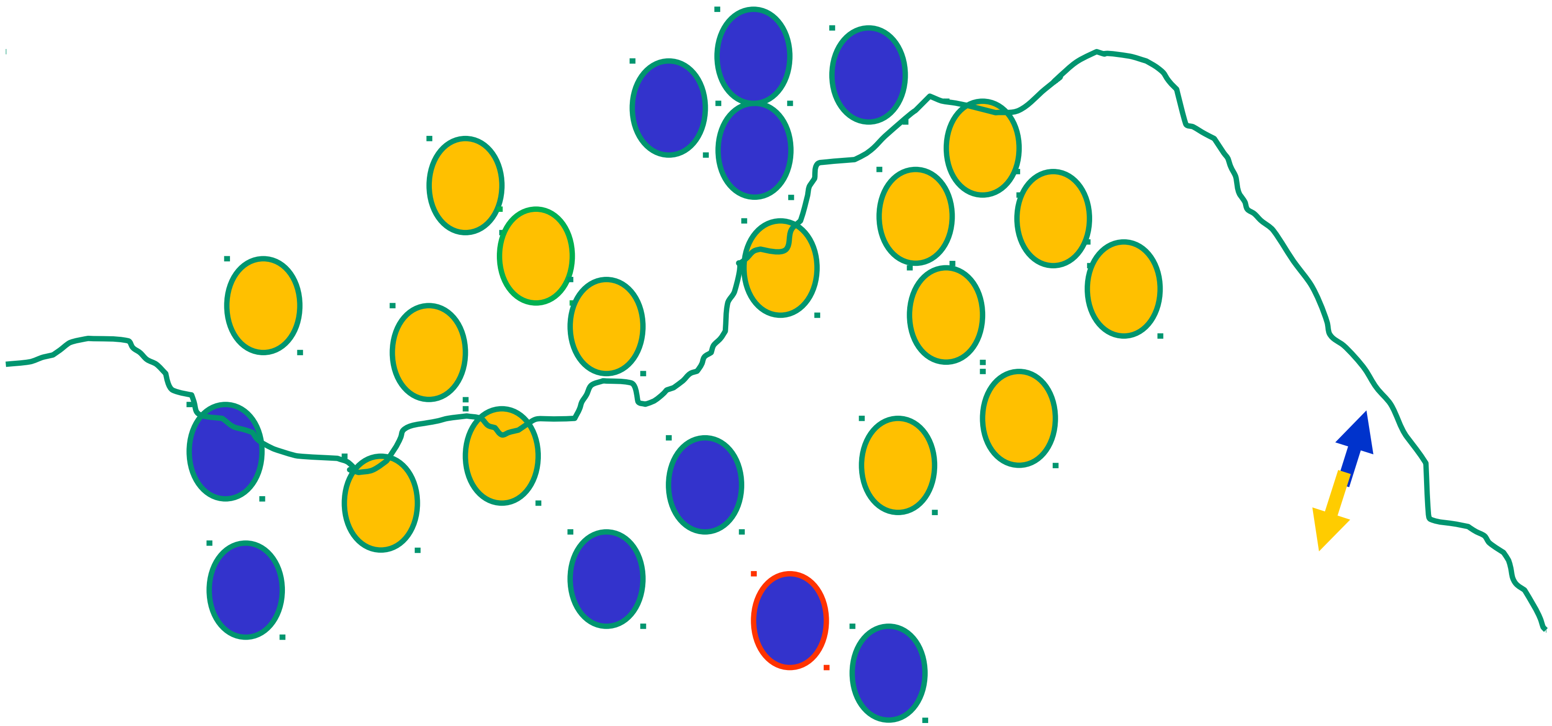


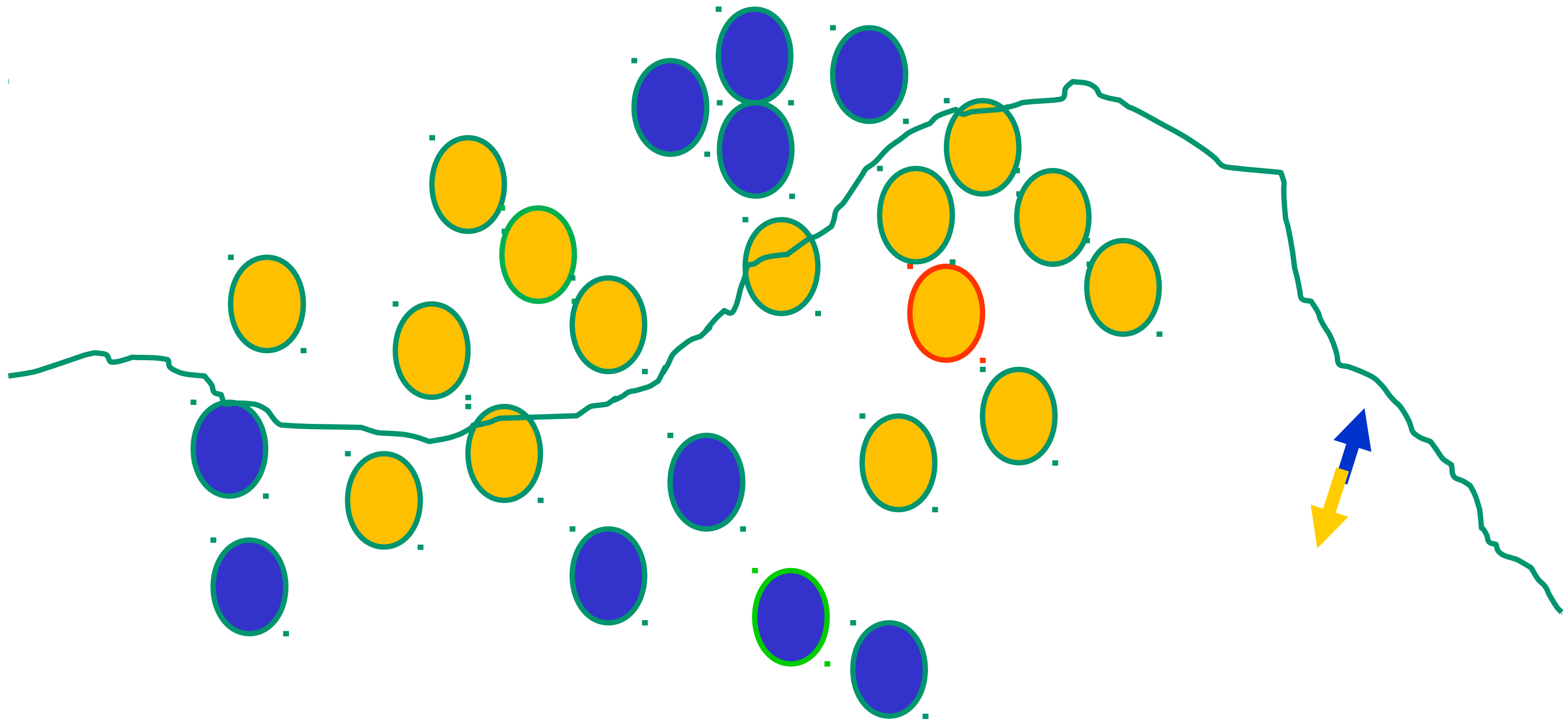
The weights are modified. Modification Based on this error.

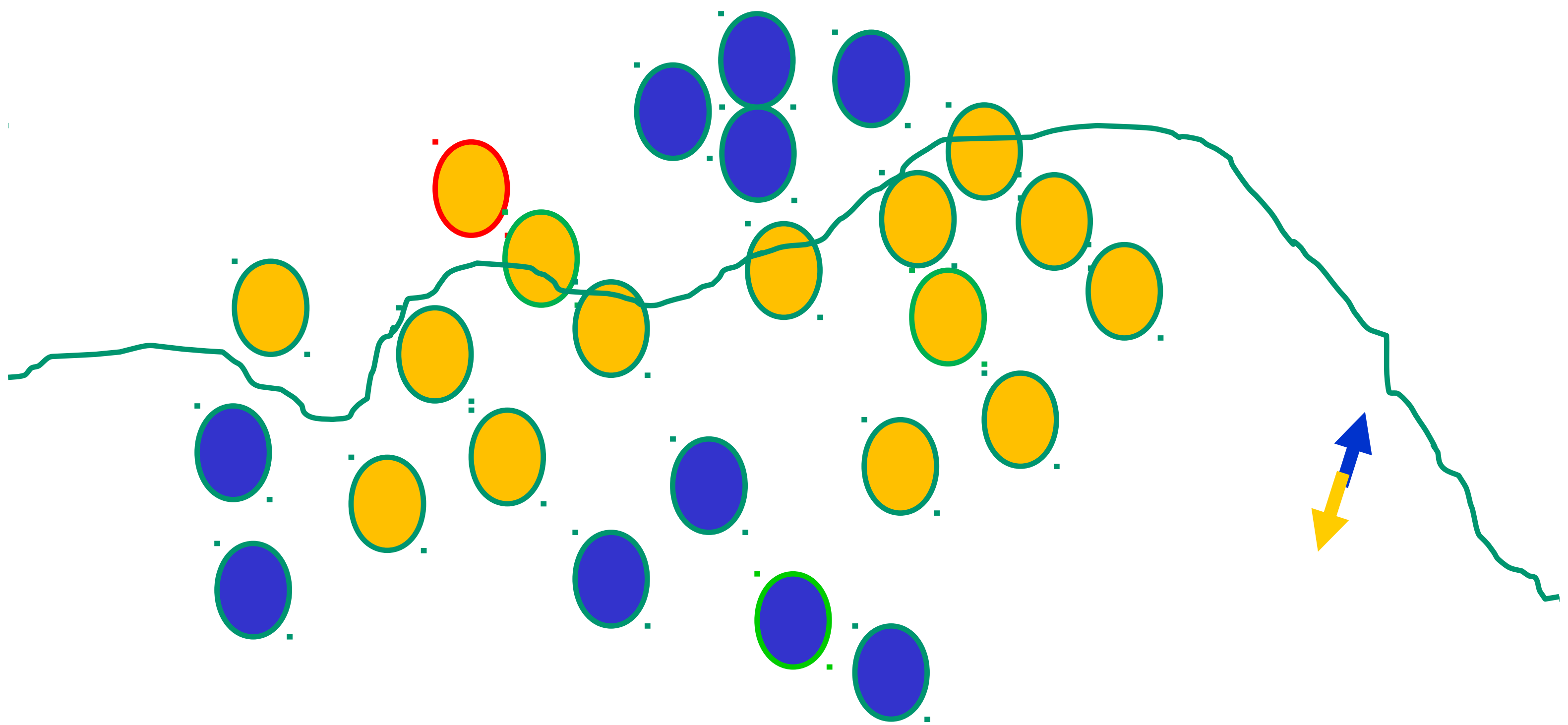
We repeat many times, each time modifying the weghts
Training algorithms take care, that the error is smaller and smaller

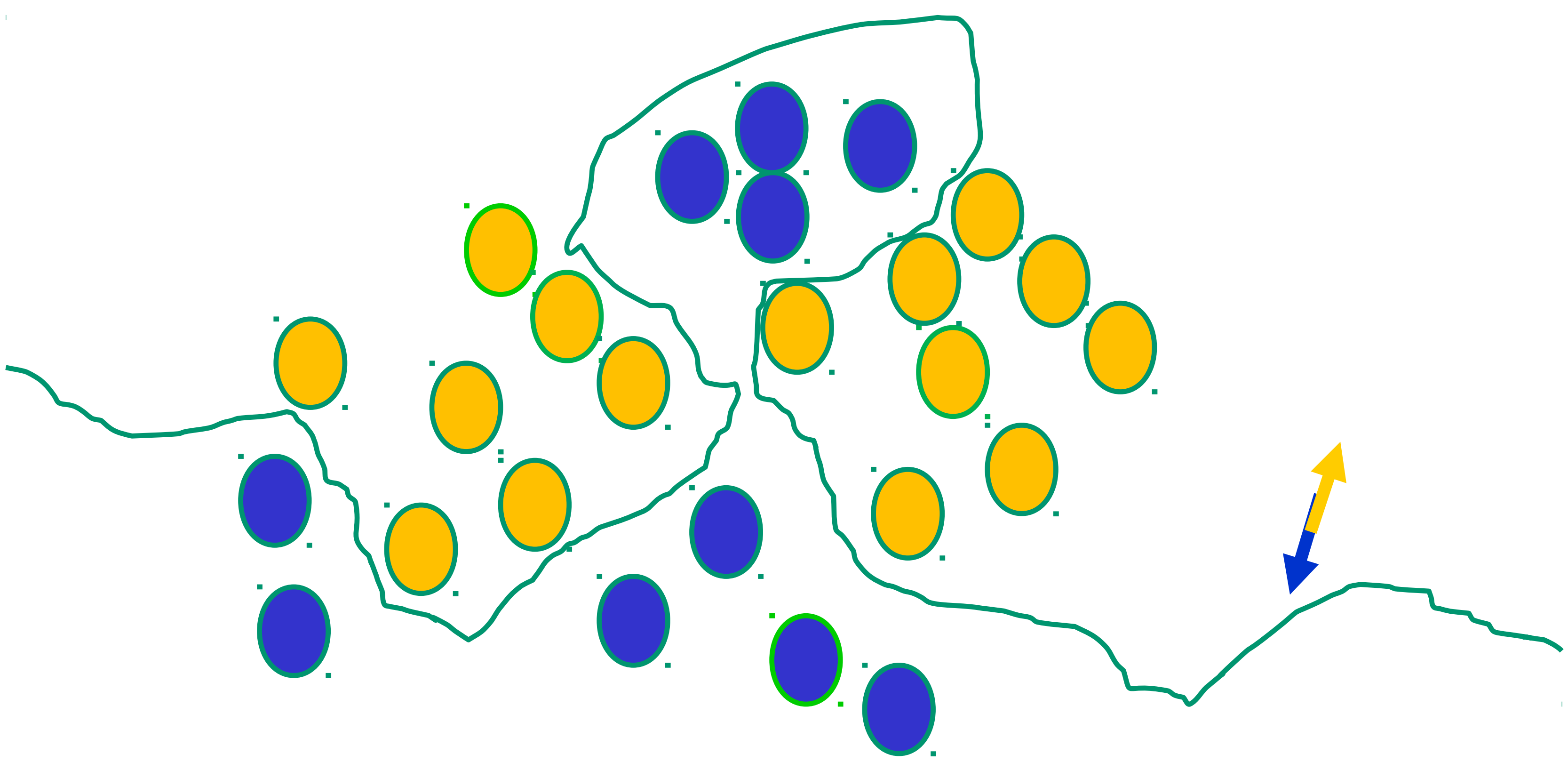












Remark

- 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 it...

How to identify the features?

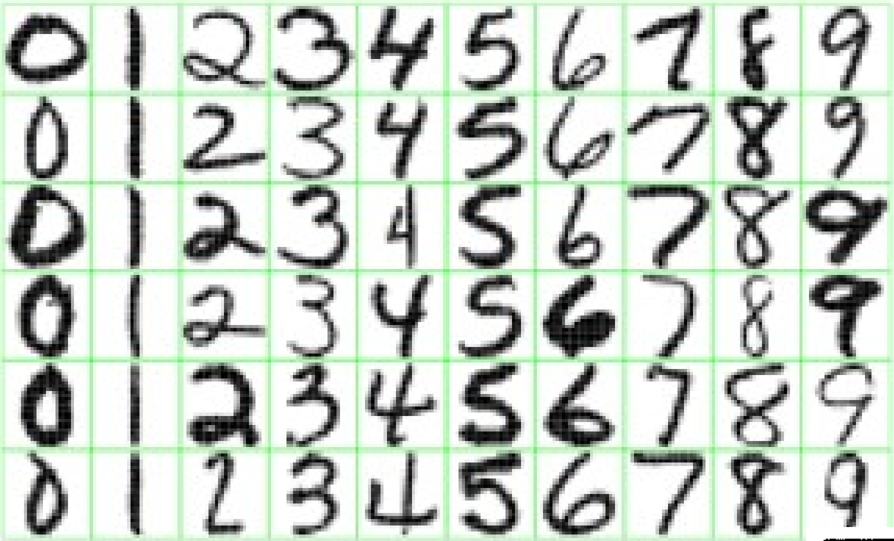
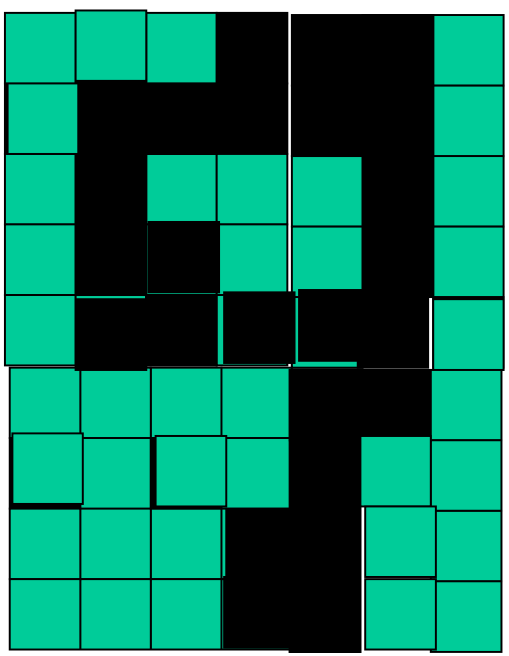
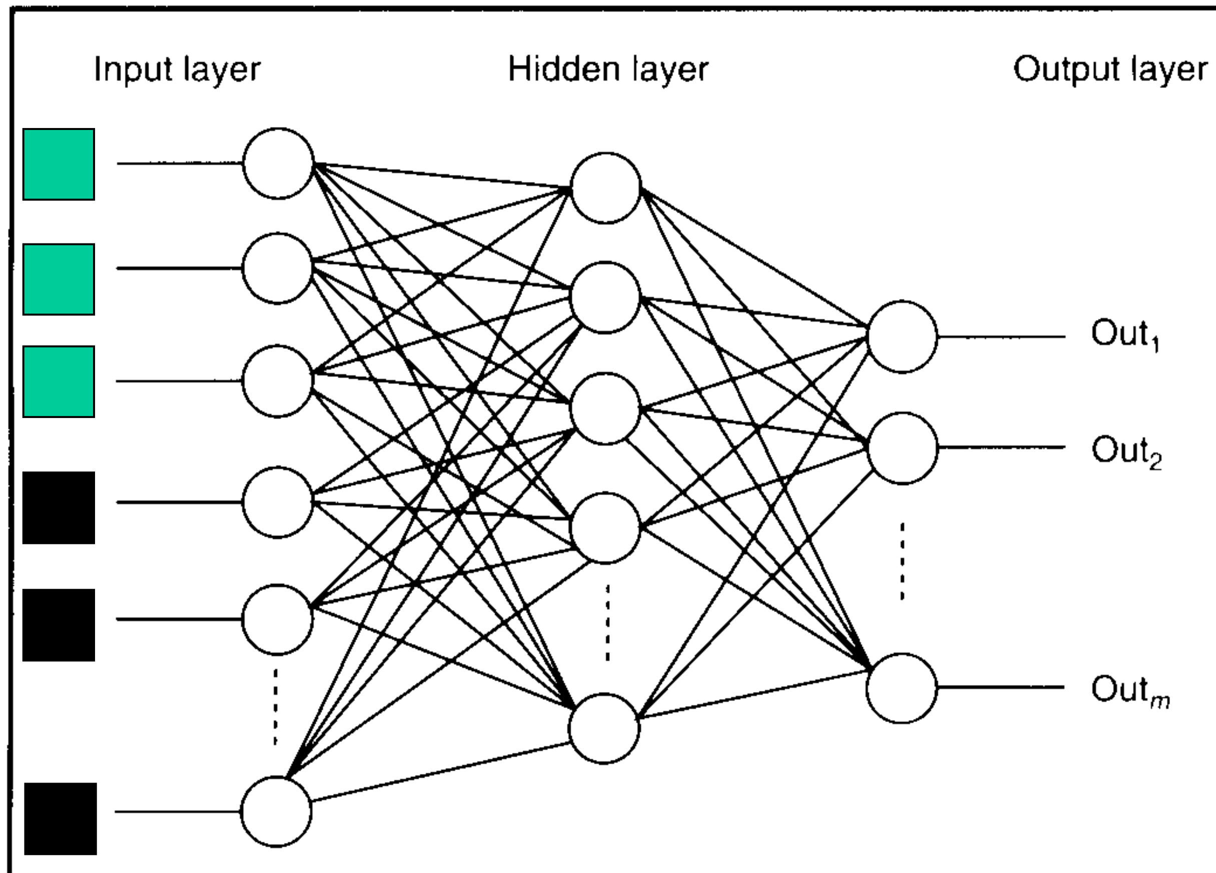


Figure 1.2: *Examples of handwritten digits from postal envelopes.*





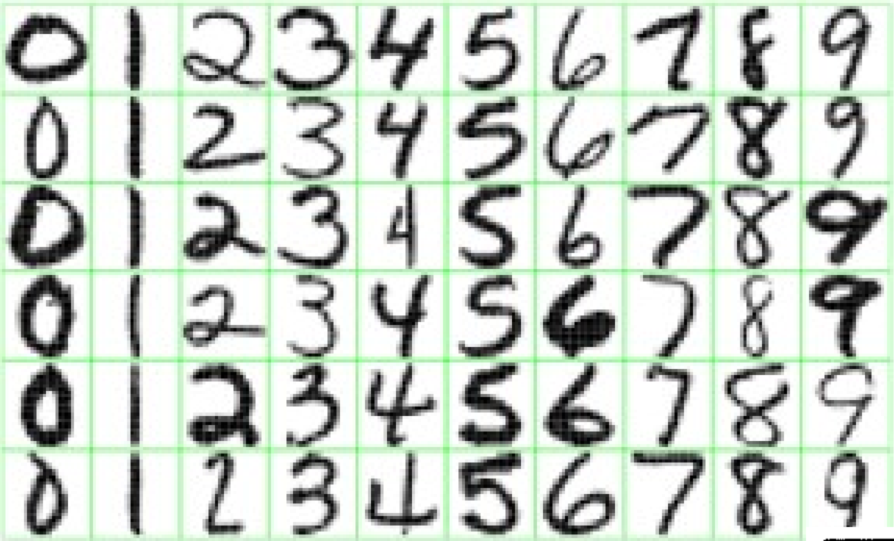
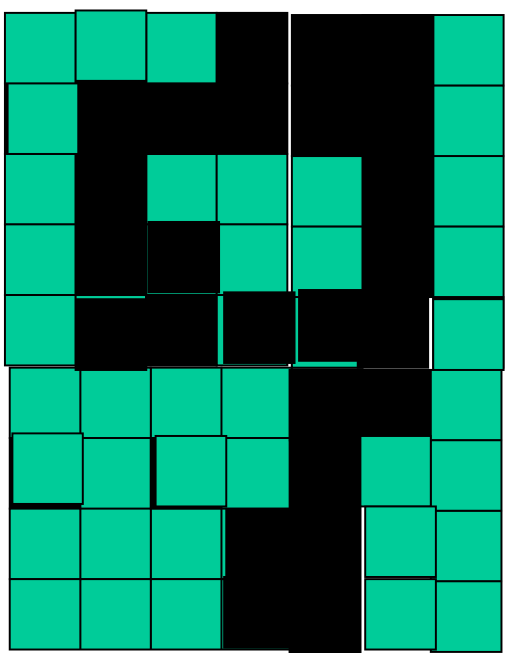
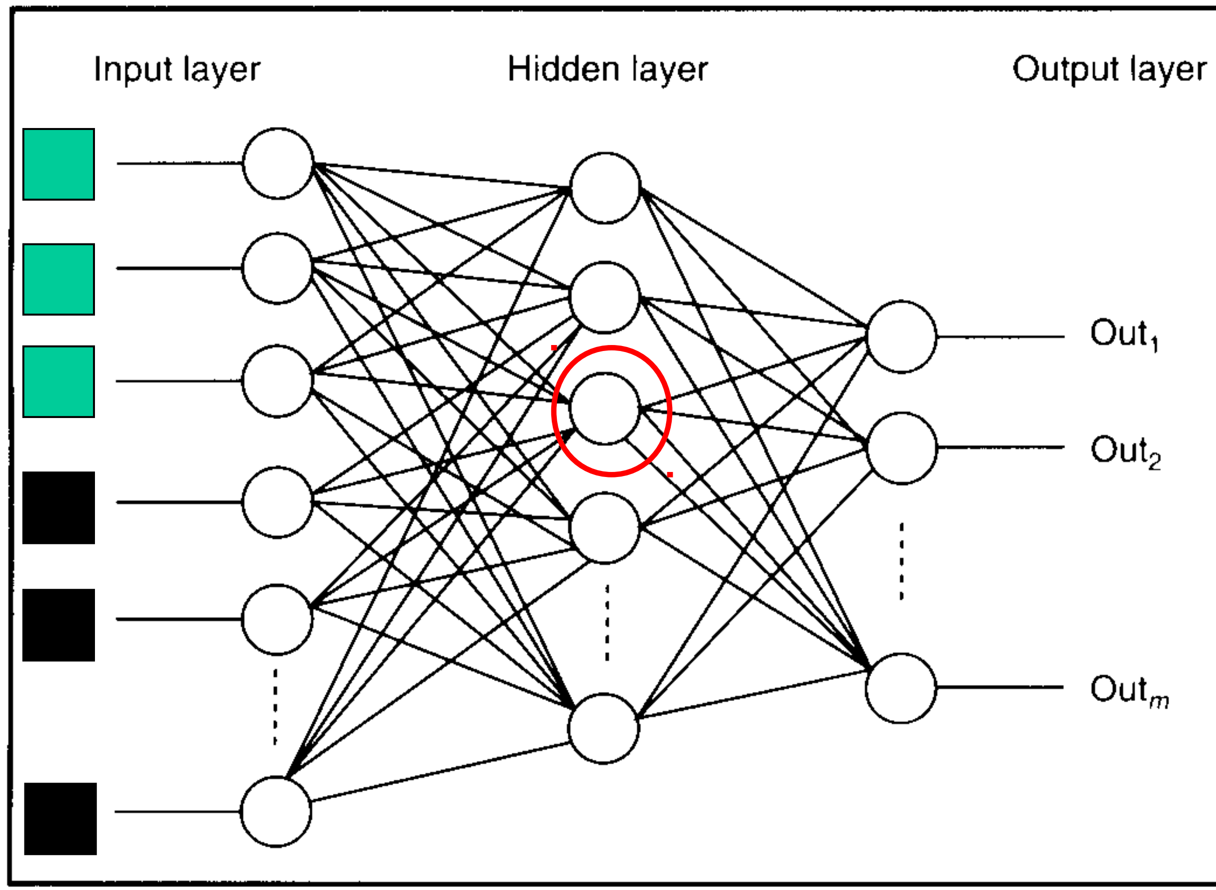


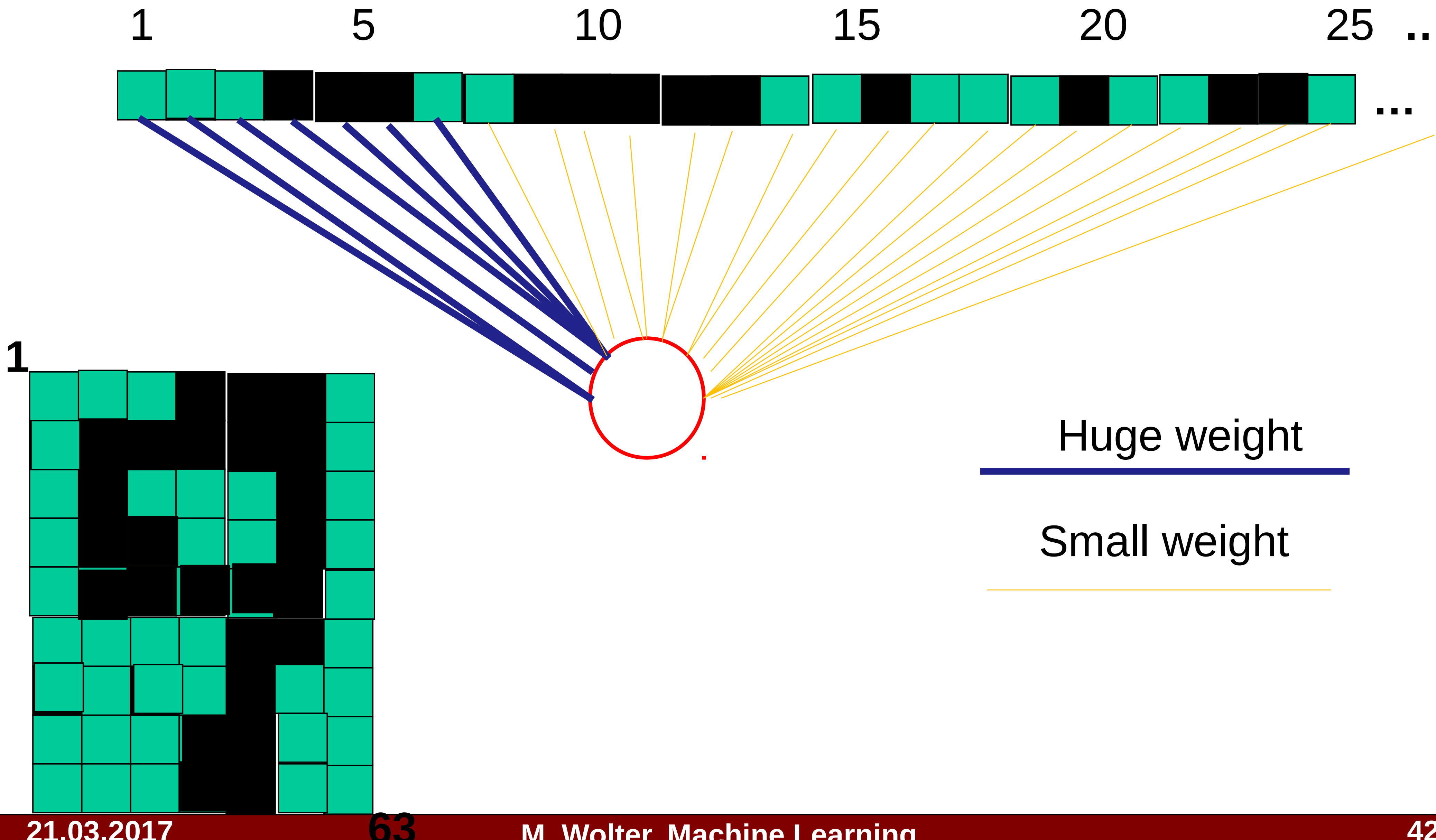
Figure 1.2: *Examples of handwritten digits from postal envelopes.*

What is this neuron doing?

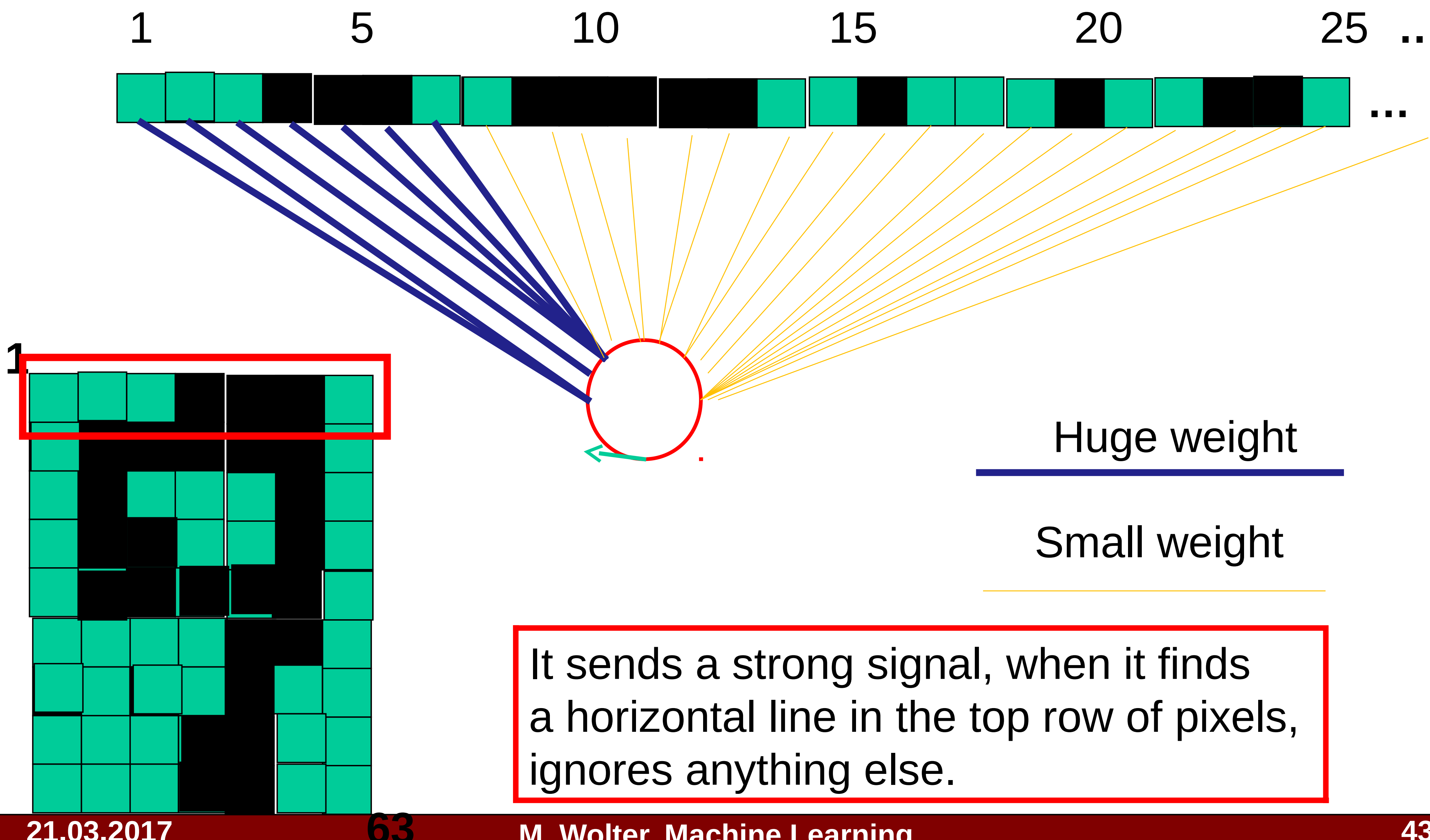




Neurons in the hidden layers are the self-organizing feature detectors

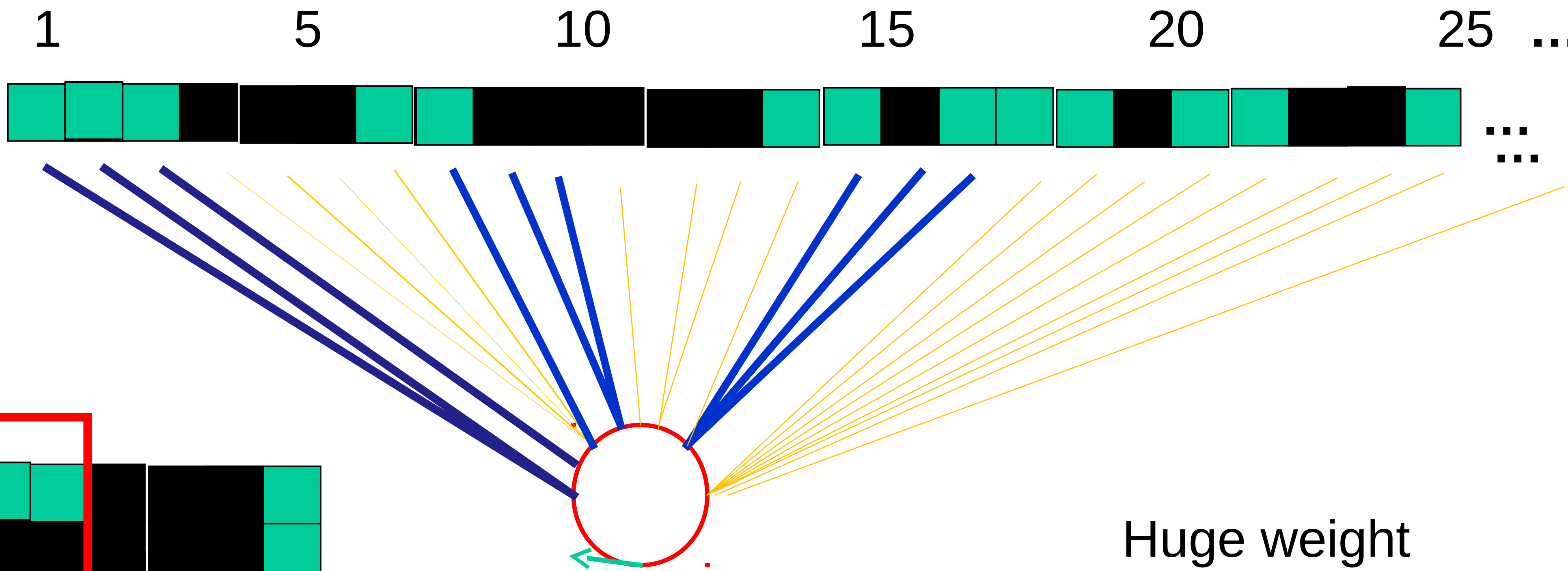


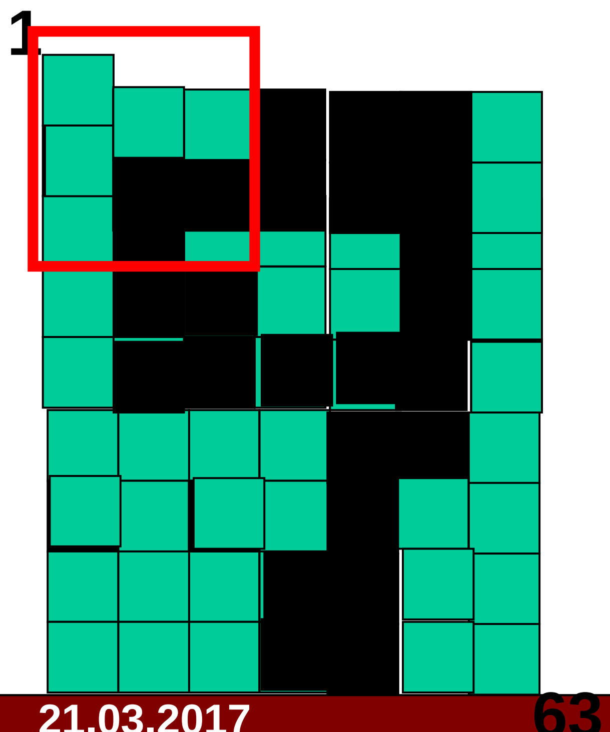
What can it detect?



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.

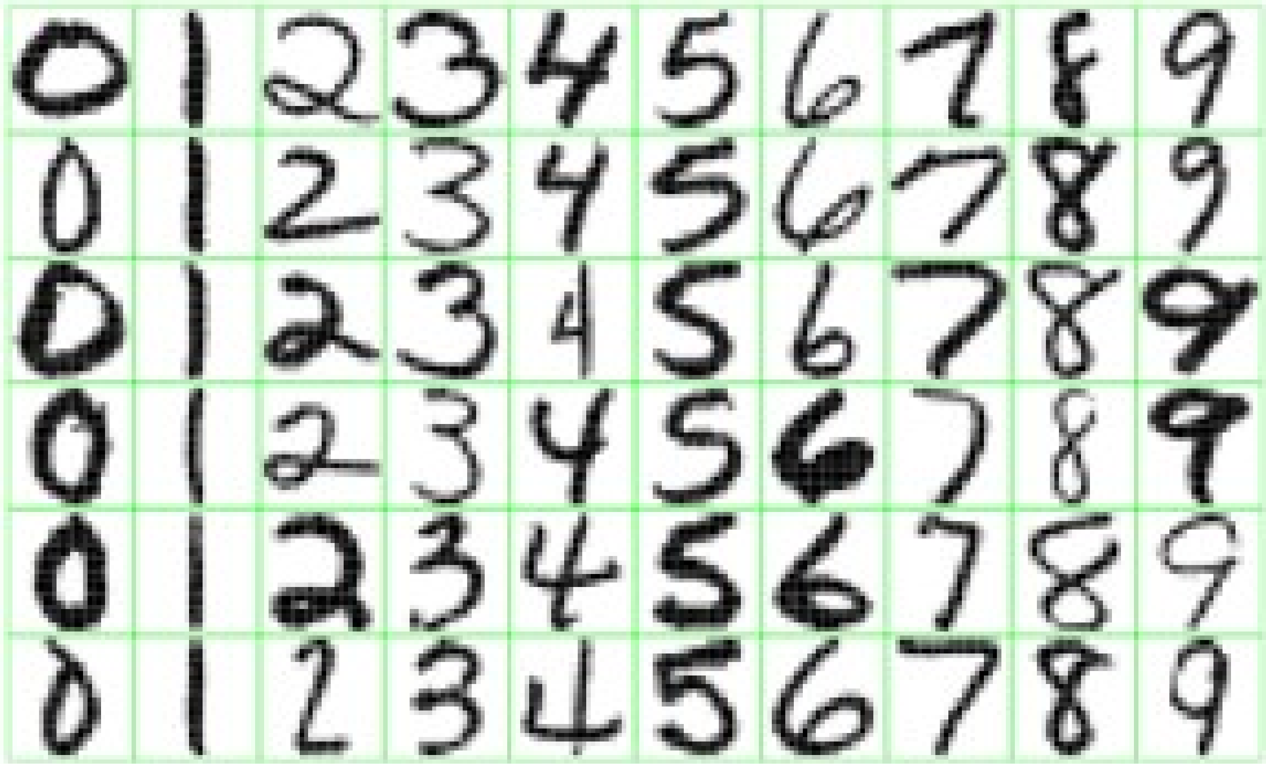
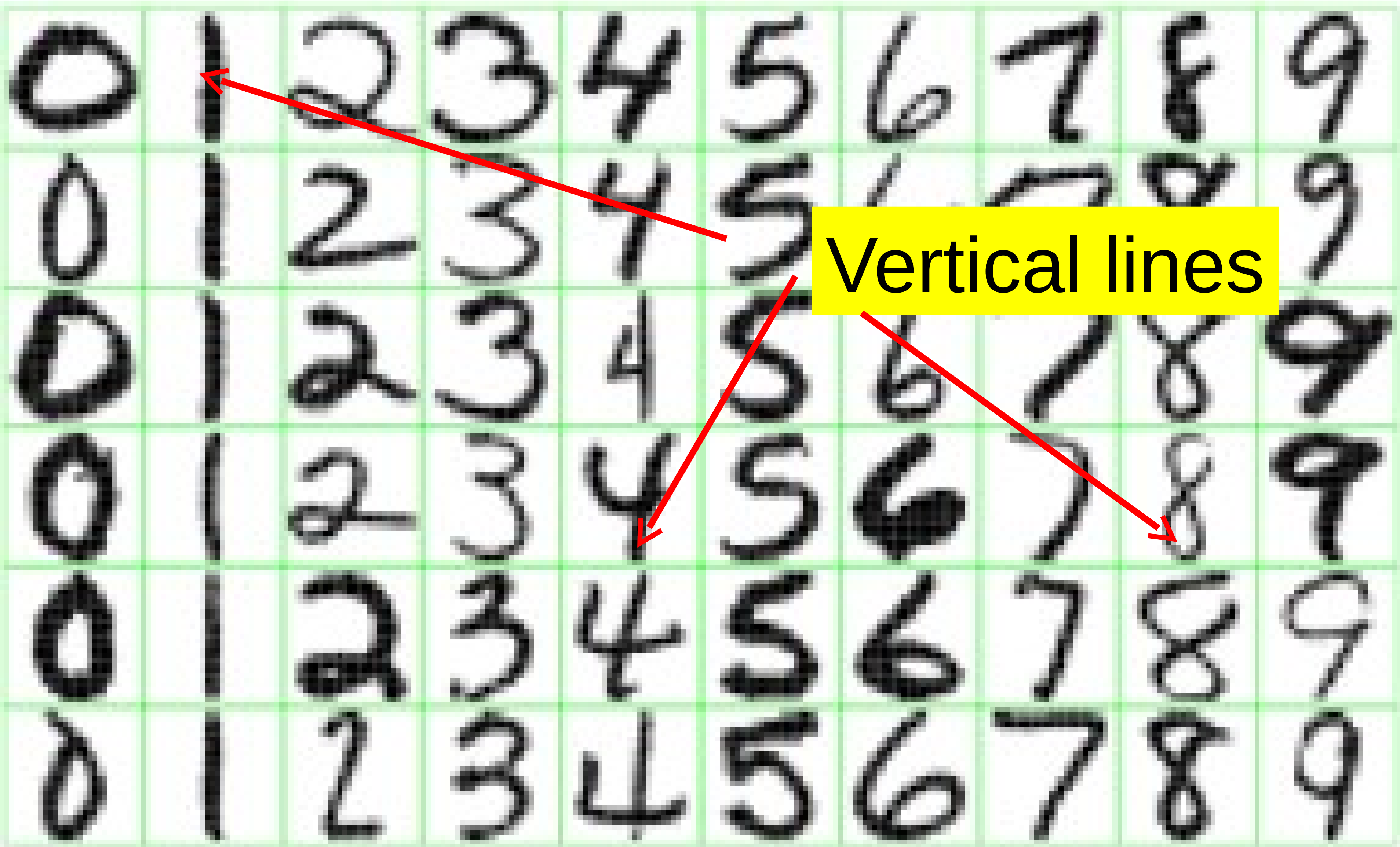


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What feature should detect a neural network recognizing the handwriting?



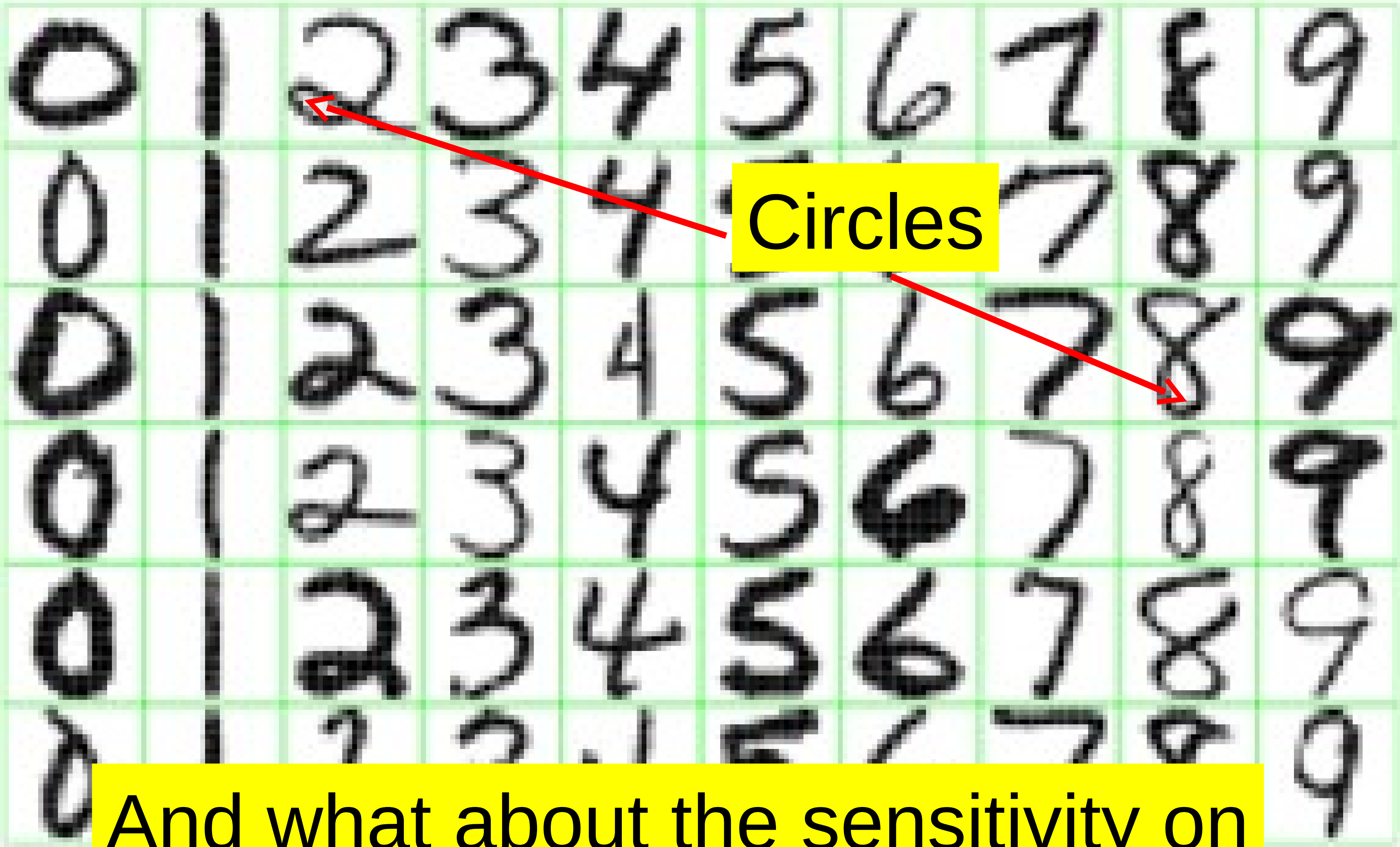
1

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

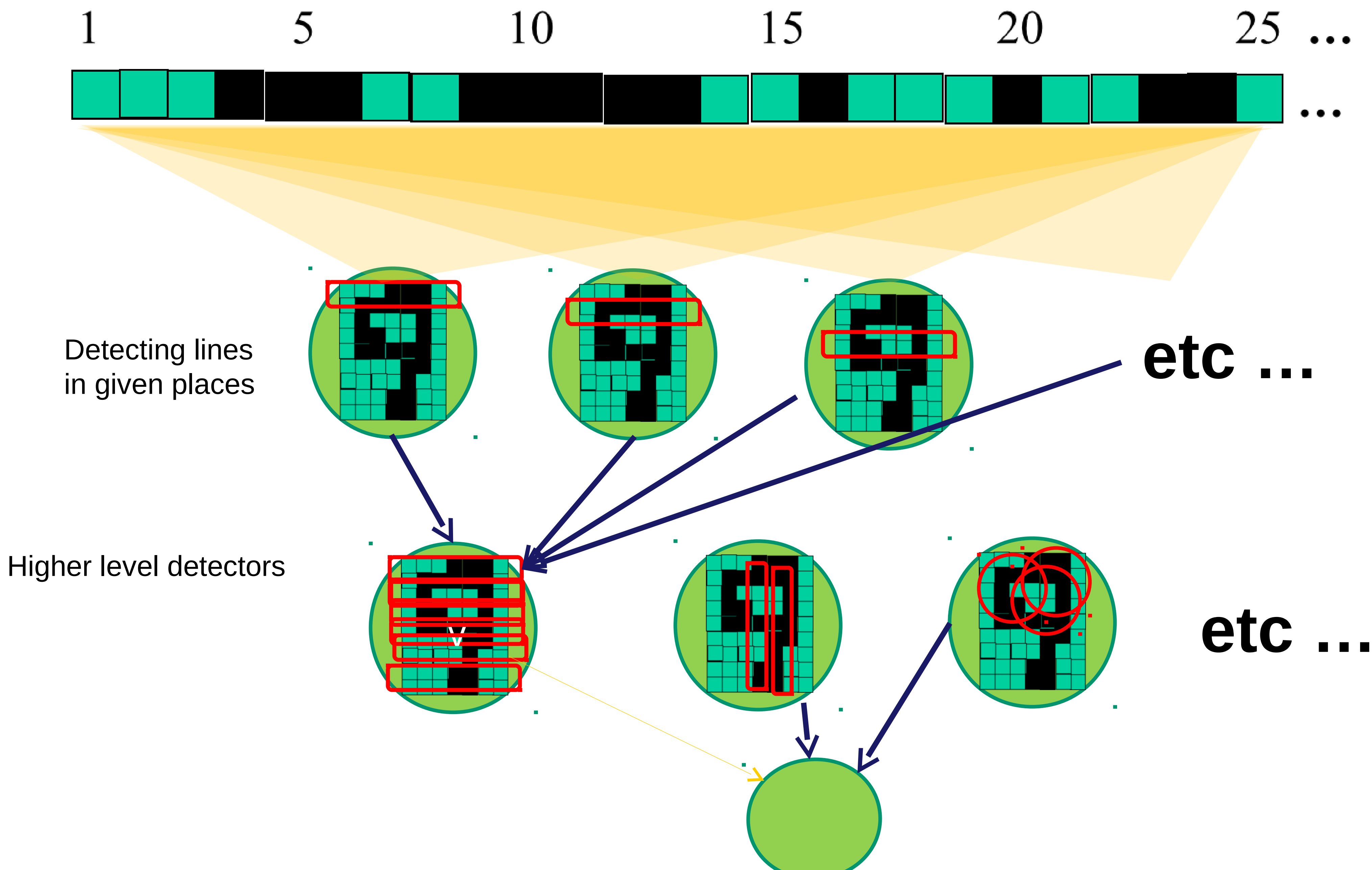
1



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

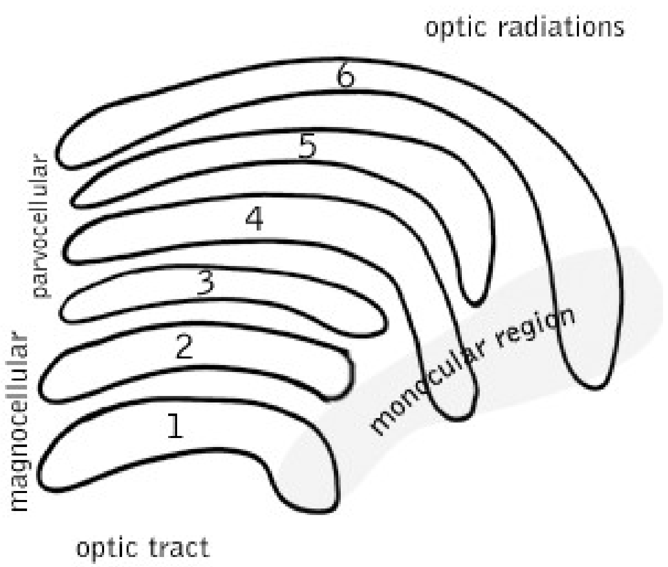
Fig ... U.S.
postal envelopes.

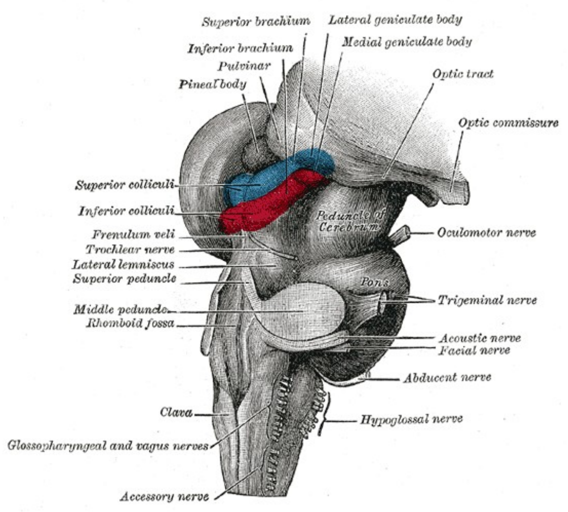
Next layers can learn the higher level features



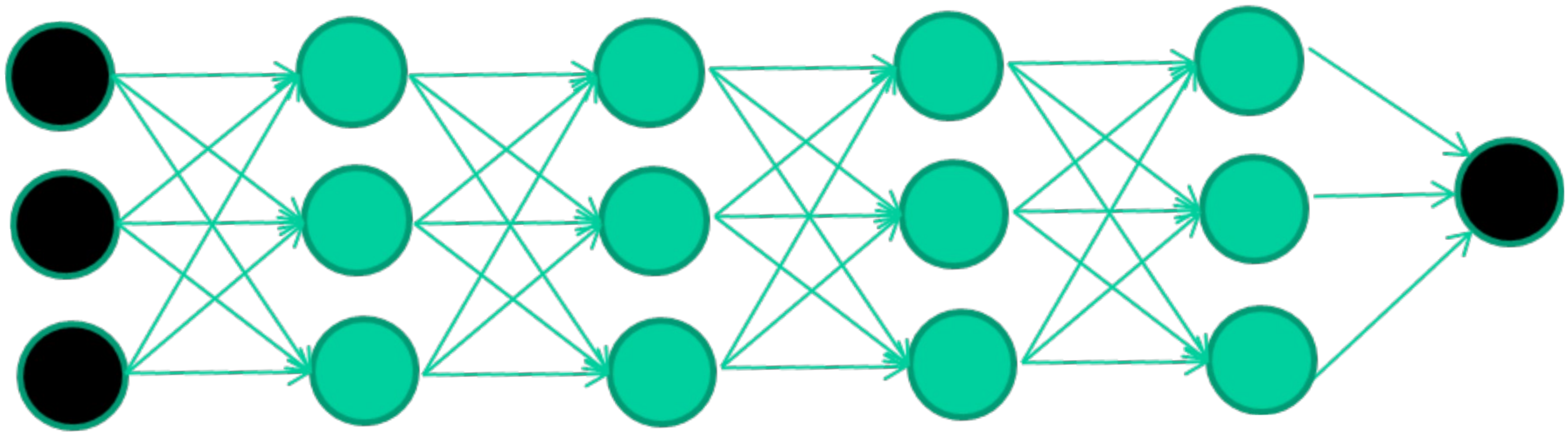
Such an organised network makes sense...

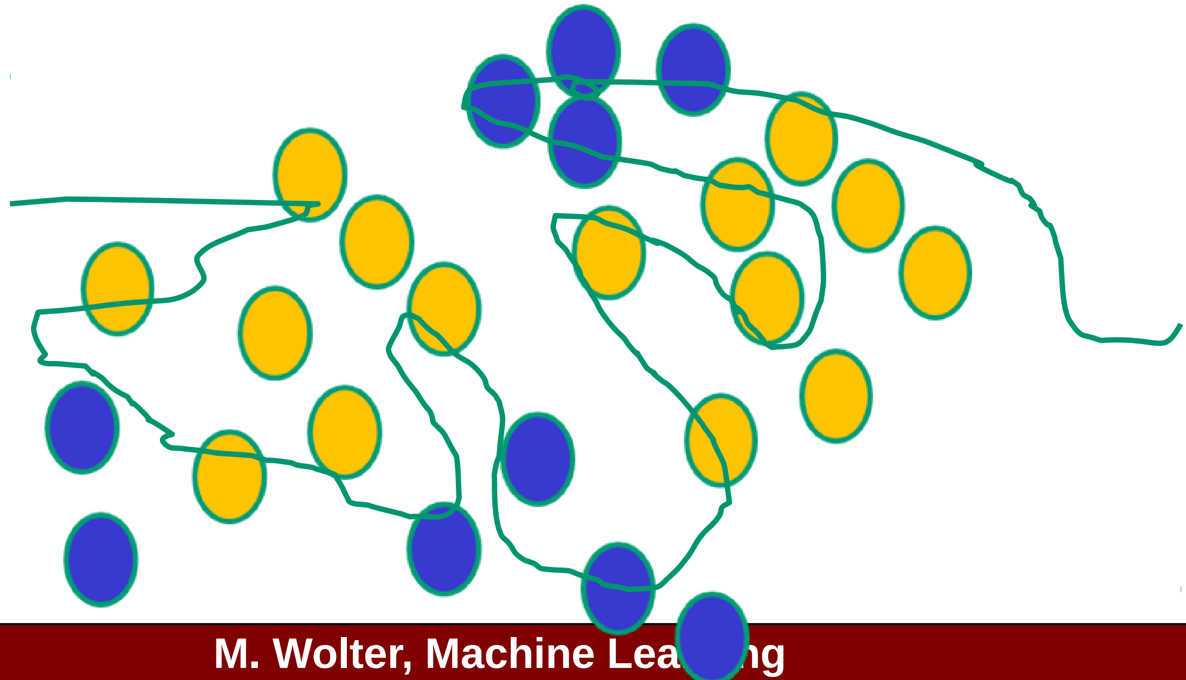
Our brains probably work in a similar way



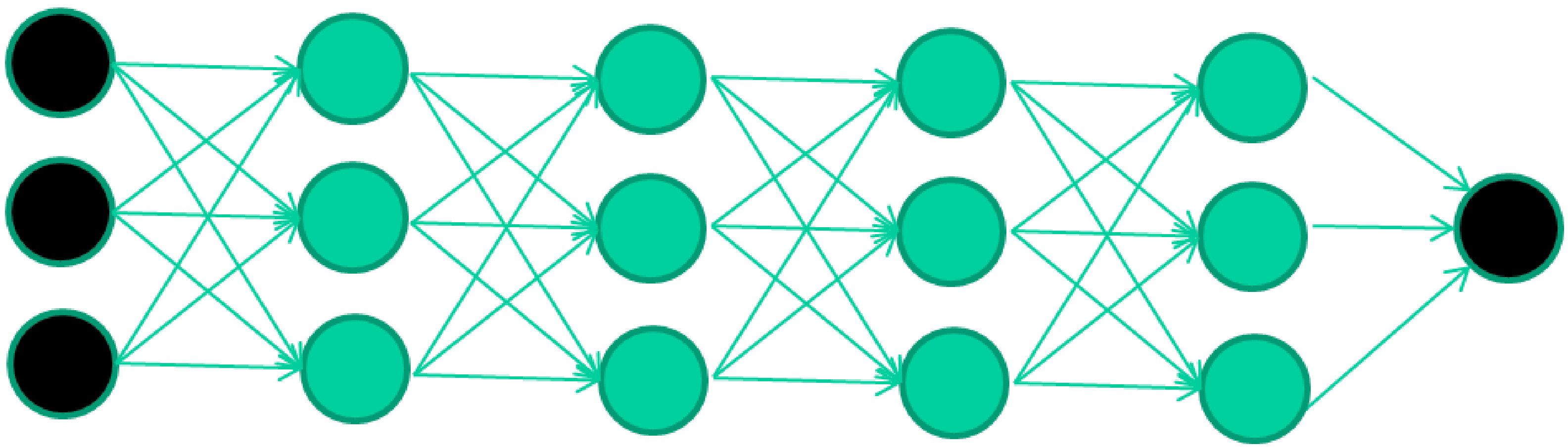


Unfortunately, until recent years we didn't know how to train a deep network

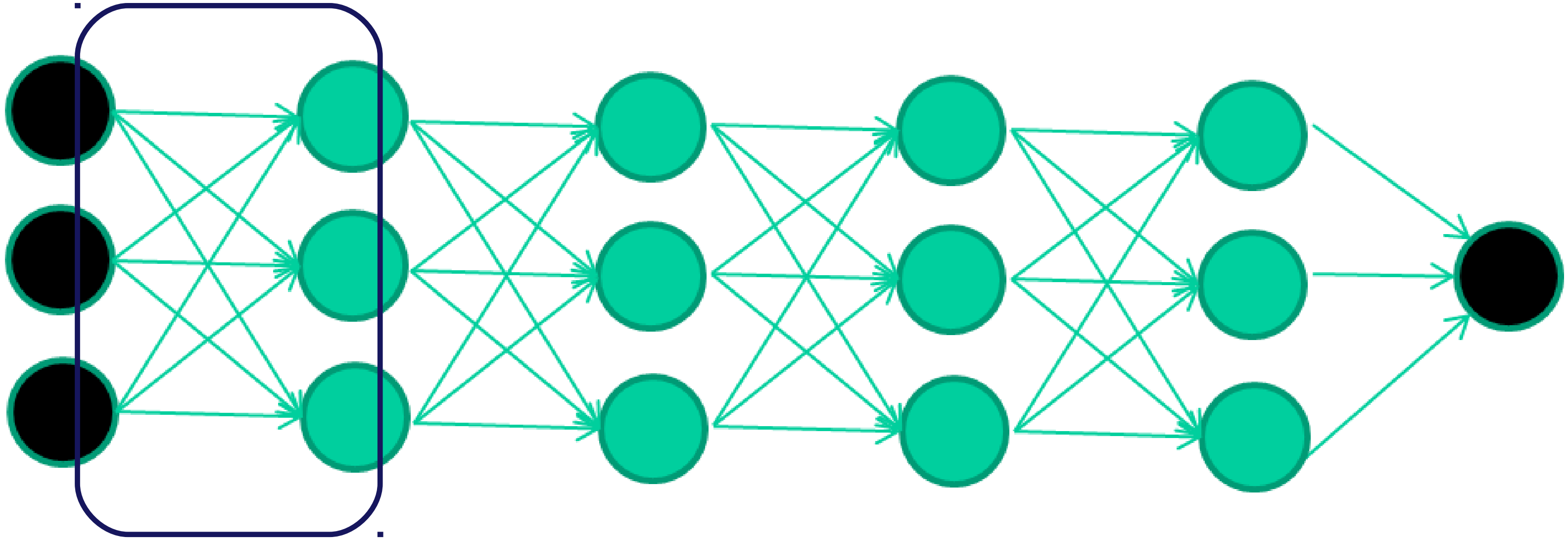




New method of training (basic info)

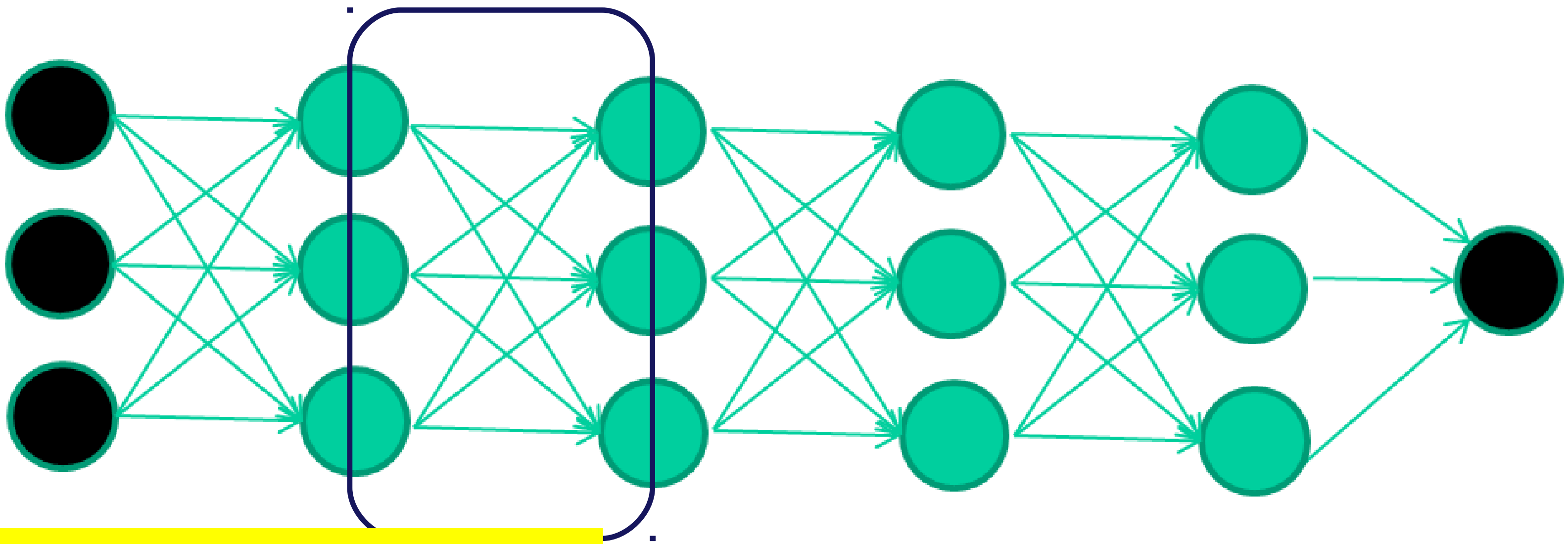


New method of training (basic info)



Train this layer

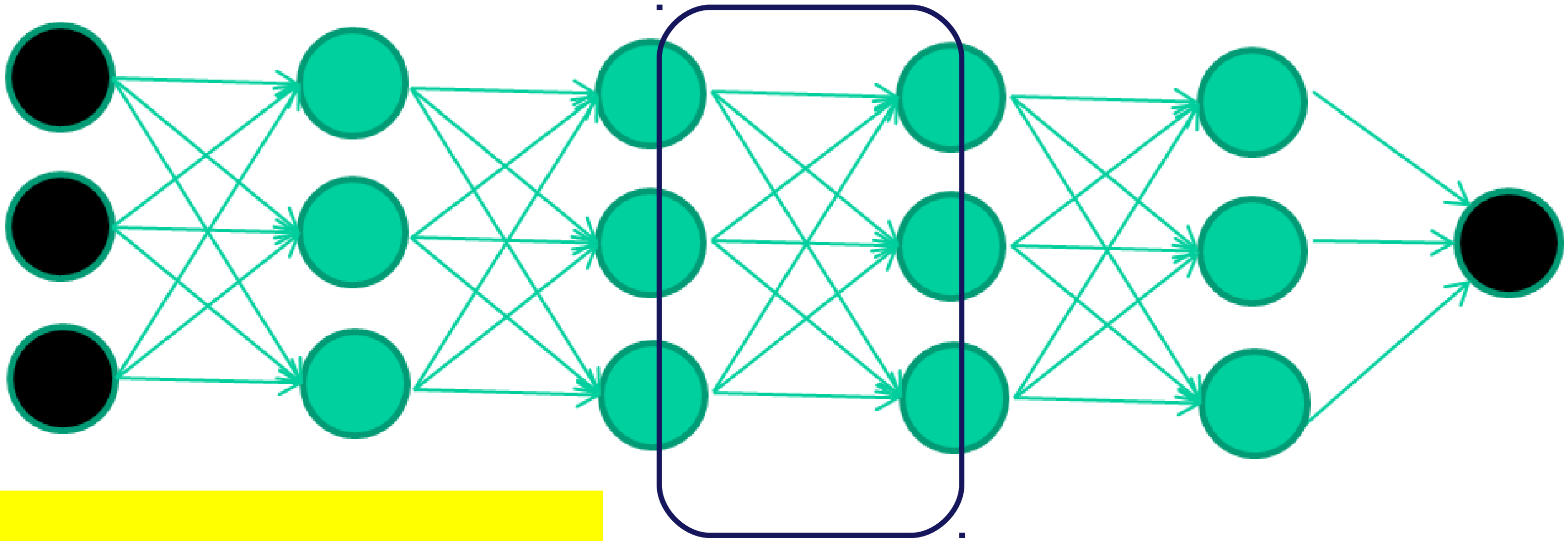
New method of training (basic info)



Train this layer

Then this

New method of training (basic info)

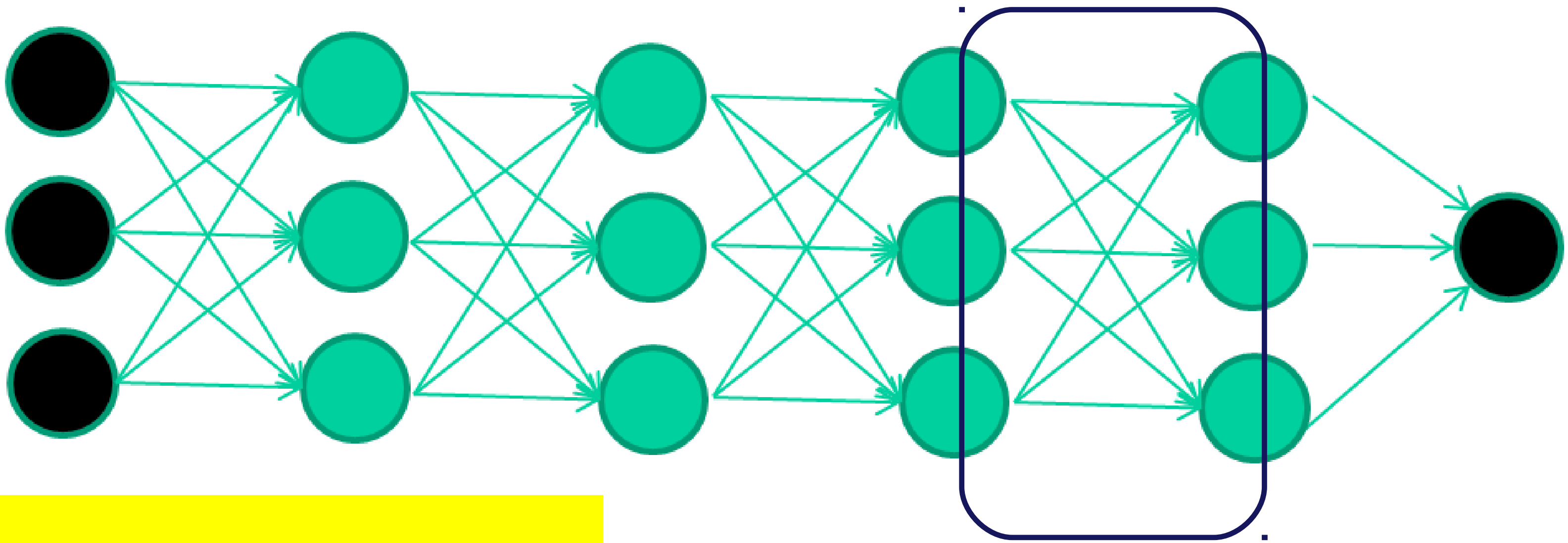


Trenuj tę warstwę

Then this

Then this

New method of training (basic info)



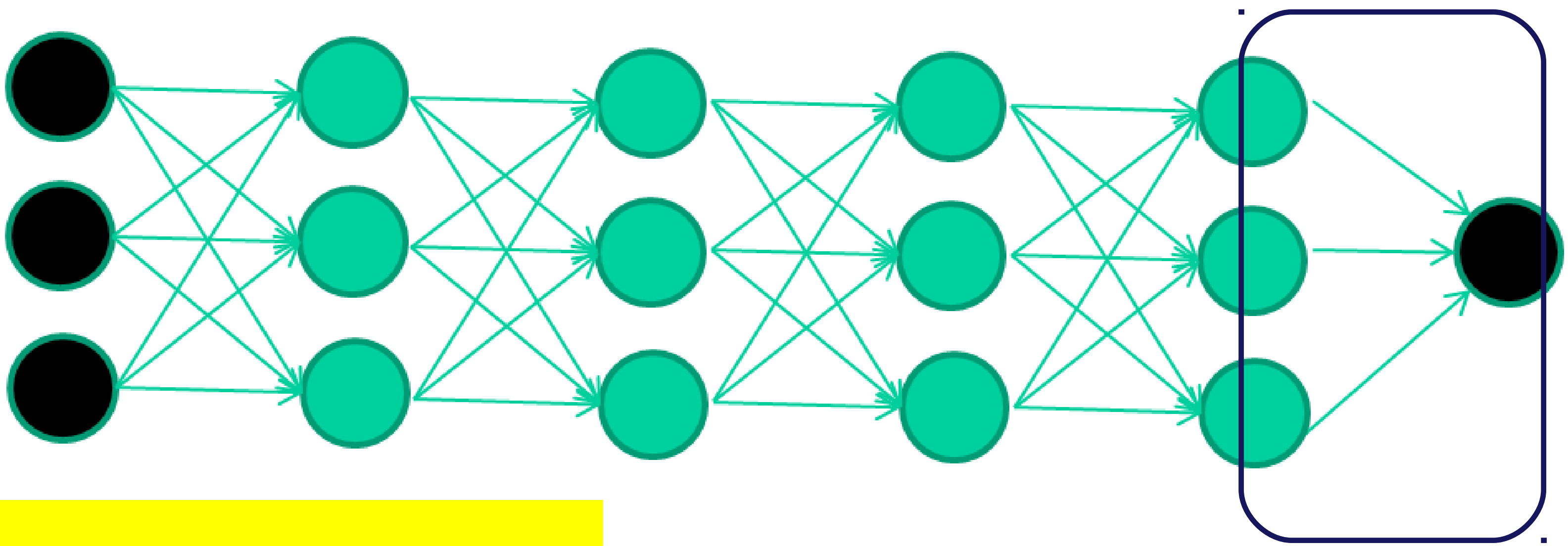
Train this layer

Then this

Then this

Then this

New method of training (basic info)



Train this layer

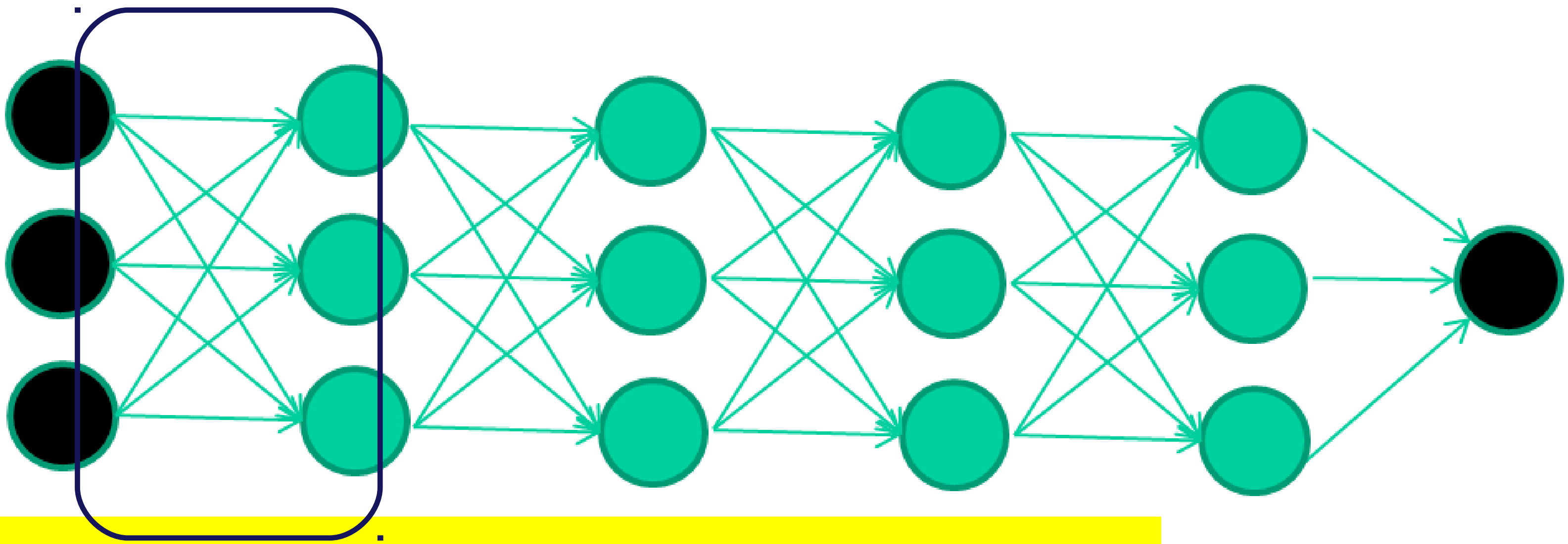
Then this

Then this

Then this

At the end this

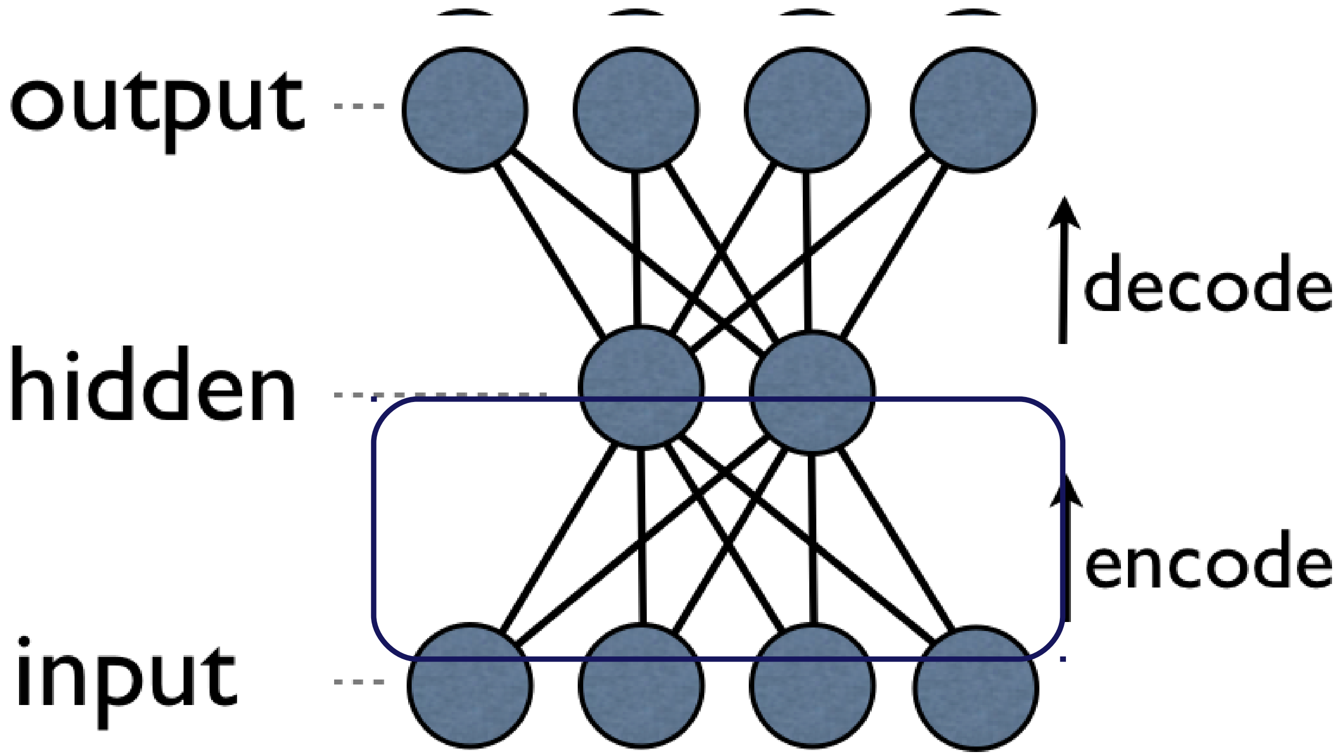
New method of training (basic info)



Each hidden layer is an automatic feature detector

an auto-encoder

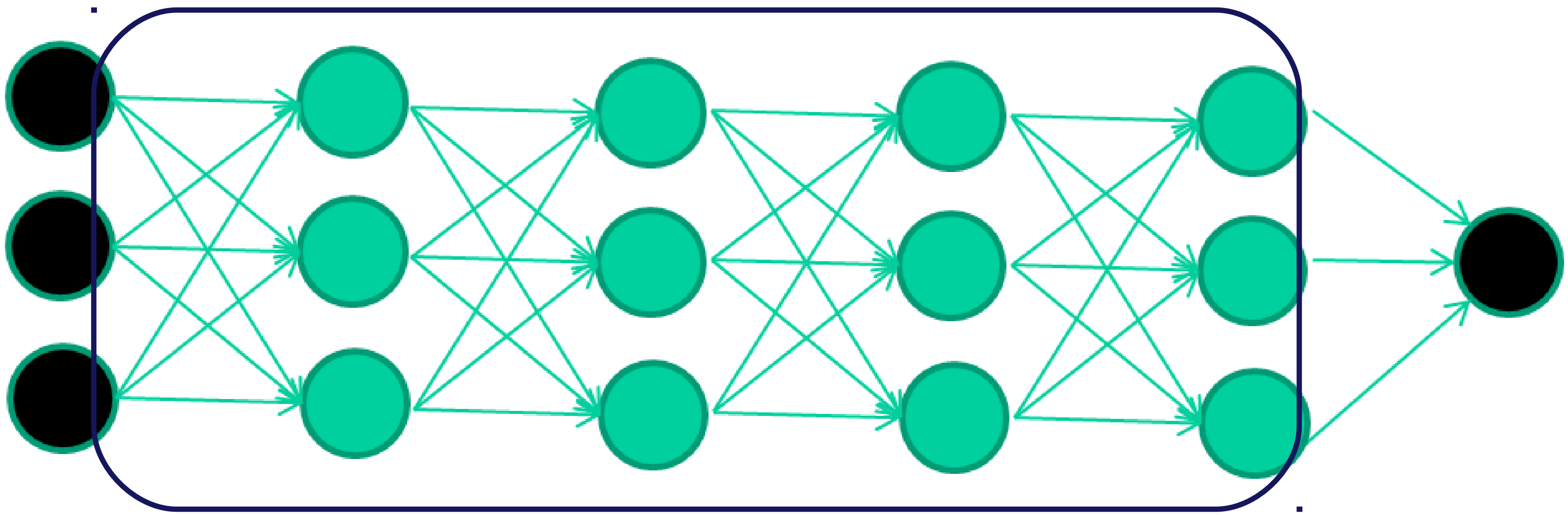
An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.



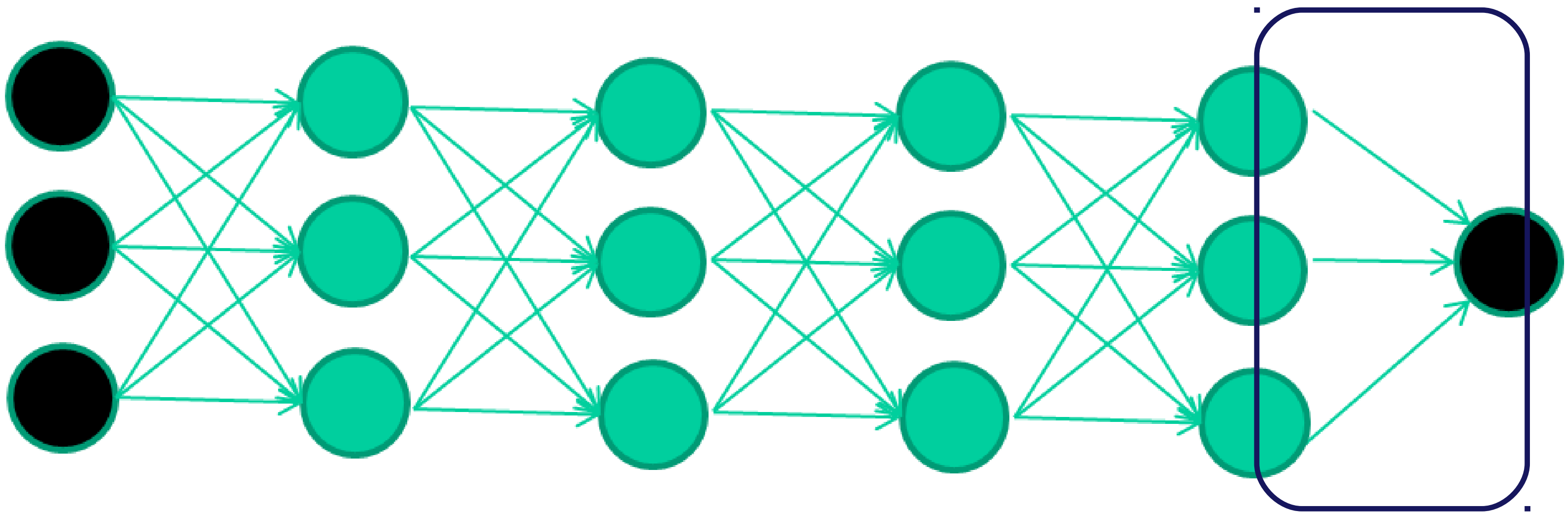
The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.

If there is a structure in the data, than it should find features.

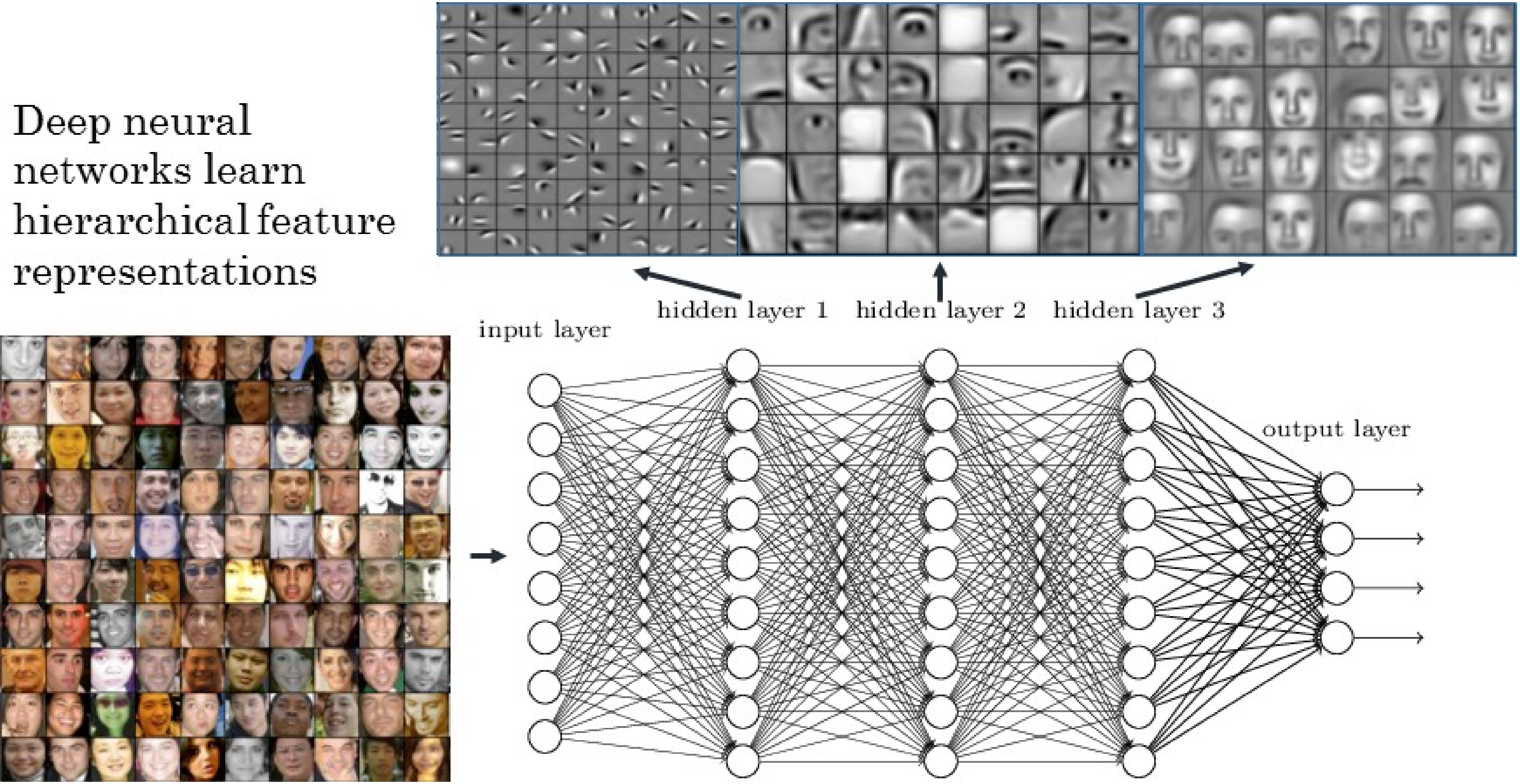
Hidden layers are trained to identify features



The last layer performs the actual classification

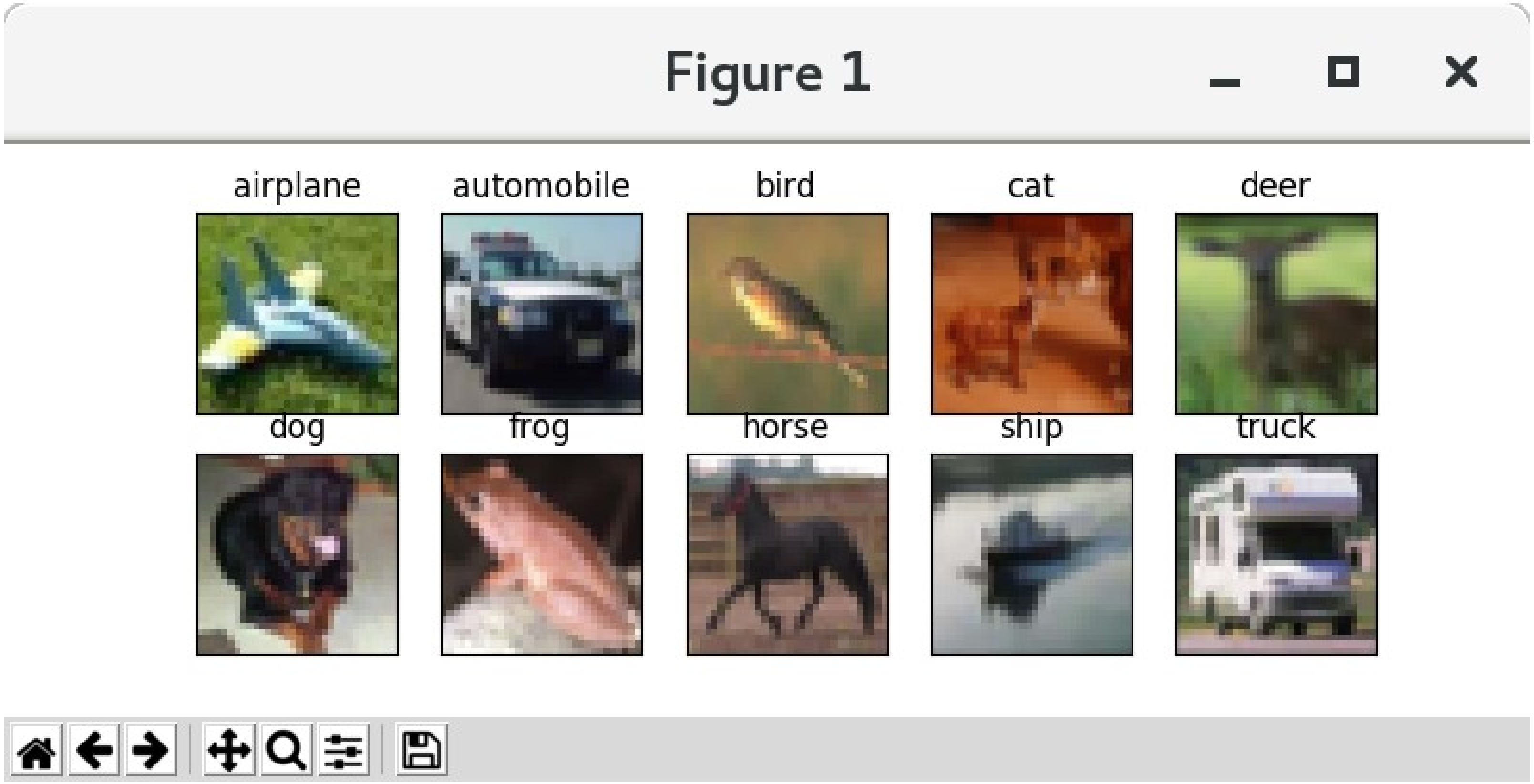


Deep neural networks learn hierarchical feature representations



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

Training:
~24 h on 12 core machine on our Cloud cluster.

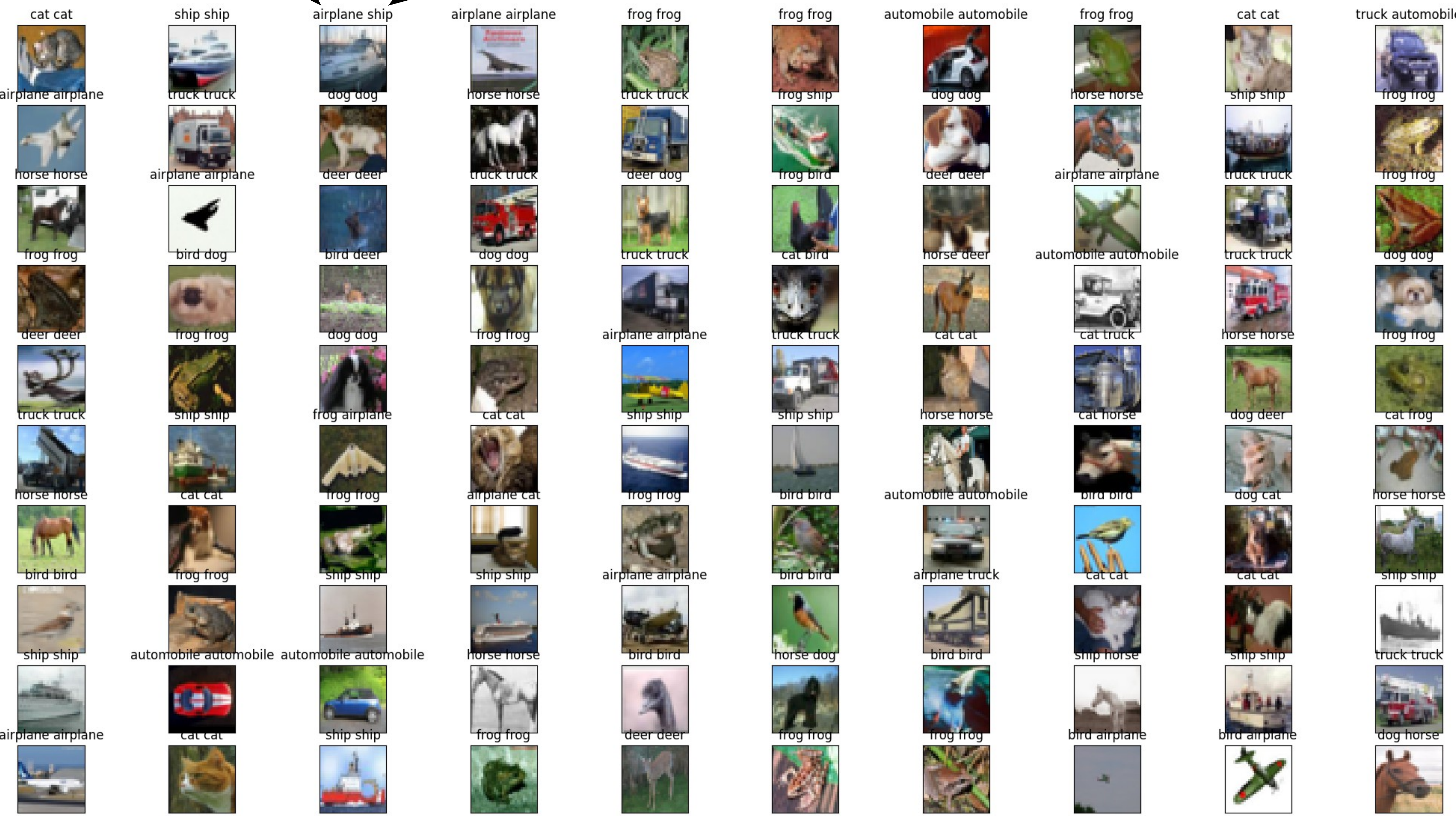
200 epochs

```
================================================================
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
```

Results

Recognized as

Really was



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

Applet showing the performance of deep NN:

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

Tab. 1.1. Wartości błędów klasyfikacji dla różnych liczebności próby uczącej

	$\hat{d}_{100}$	$\hat{d}_{200}$	$\hat{d}_{500}$	$\hat{d}_{1000}$
$\hat{e}_R$	0.0065	0.0110	0.0148	0.0164
$\hat{e}_{CV}$	0.0115	0.0150	0.0242	0.0200
$\hat{e}_{10CV}$	0.0138	0.0157	0.0239	0.0197
$\hat{e}_{J'}$	0.0143	0.0151	0.0247	0.0198
$\hat{e}_{B_1}$	0.0125	0.0224	0.0362	0.0332
$\hat{e}_{B_2}$	0.0194	0.0325	0.0323	0.0288
$\hat{e}_{.632}$	0.0147	0.0246	0.0280	0.0250
$\hat{e}_{.632+}$	0.0149	0.0248	0.0282	0.0251
$\hat{e}_T$	0.0262	0.0215	0.0197	0.0187

Bootstrap

Ostatni wiersz tab. 1.1 zawiera estymator $\hat{e}_T$ aktualnego poziomu błędu klasyfikatora uzyskany z niezależnie wygenerowanej próby testowej o liczbie elementów $m = 100\,000$. Nie będzie więc dużym nadużyciem (ze względu na liczebność próby testowej), jeżeli ten błąd klasyfikacji przyjmiemy za aktualny poziom błędu klasyfikatorów $\hat{d}_n$.

Wobec powyższej uwagi zauważmy, że dla $n = 100$ wszystkie estymatory zaniżają (niedoestymowują) aktualny poziom błędu. To znaczne obciążenie wszystkich ocen jest efektem zbyt małej próby (pamiętajmy, że obserwacje pochodzą z przestrzeni 8-wymiarowej). Dla $n = 200$ niedoszacowanie aktualnego poziomu błędu obserwujemy dla estymatora ponownego podstawienia, sprawdzania krzyżowego oraz dla estymatora typu jackknife, natomiast w przypadku większych prób – dla wszystkich ocen z wyjątkiem estymatora resubstytucji.

Unfortunately Polish...