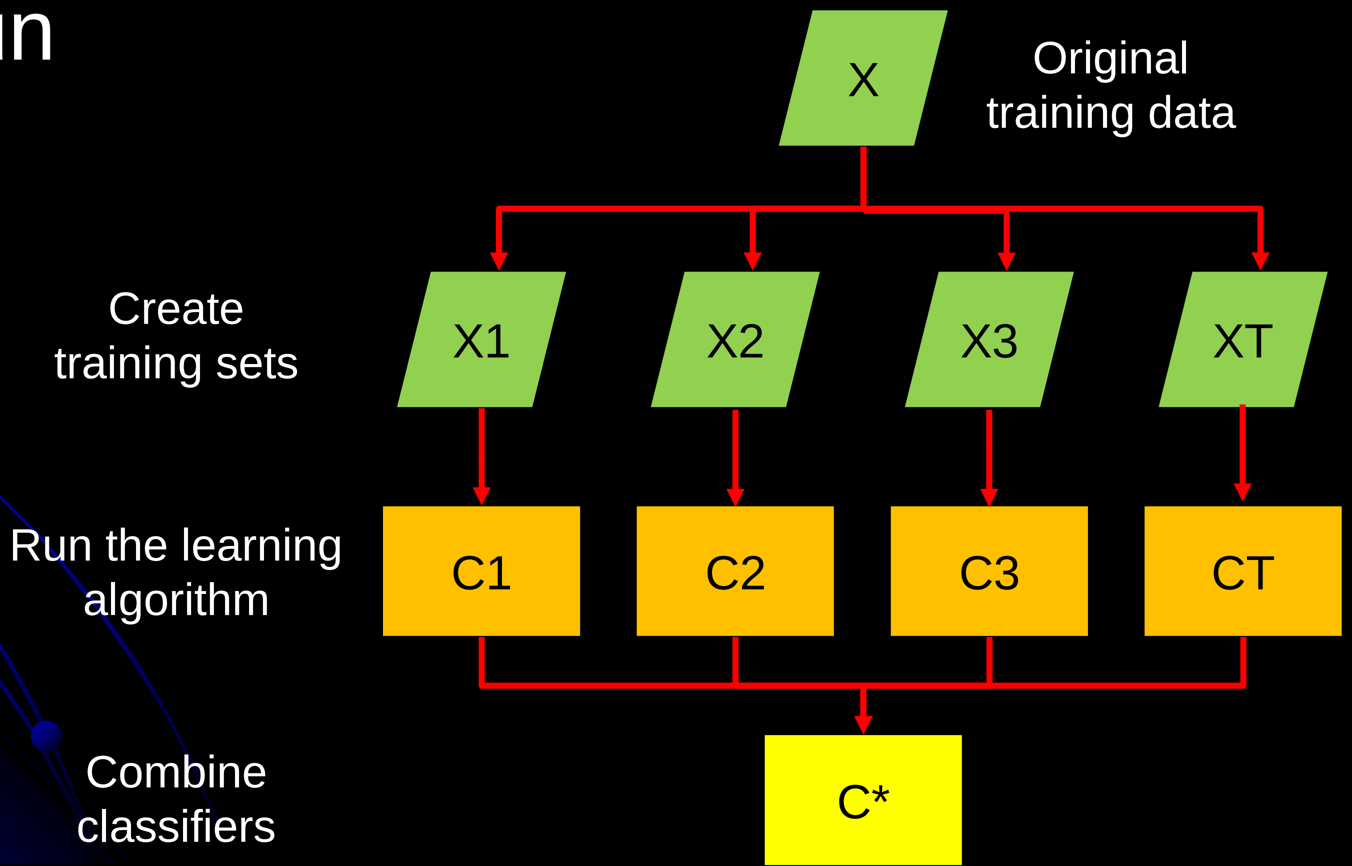
Machine learning in computer vision

Lesson 8

Independently Constructed Ensembles

Run the learning algorithm several times and provide it with somewhat different data in each run



Independently Constructed Ensembles

- Bagging
- Randomness Injection
- Feature-Selection Ensembles

Bagging

- Bootstrap aggregating
- We introduced the bootstrap as a way of assessing the accuracy
- Here, bootstrap is used to create the diverse training sets
- Each bootstrap sample is drawn with replacement, so each one contains some duplicates of certain training points and leaves out other training points completely

Bagging

Accuracy is increased if the prediction method is unstable, i.e. if small changes in the training set or in the parameters used in construction can result in large changes in the resulting predictor

Trees, neural nets are unstable, as are other well-known prediction methods. Other methods such as nearest neighbors, are stable

Bagging

Classifier generation

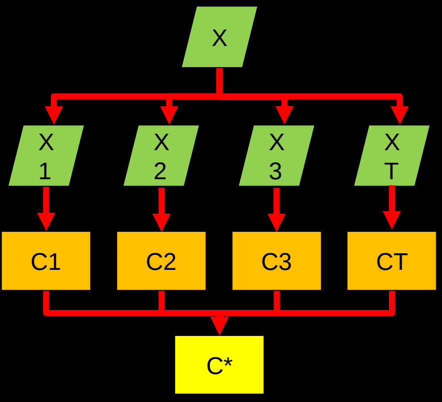
```
Let N be the size of the training set
For each of T iterations:
  Sample N instances with replacement from the
training set
  Apply the learning algorithm to the sample
  Store the resulting classifier
```

Classification

```
For each of the T classifiers:
  Predict class of instance using classifier
Return class that was predicted most often
```

Out-of-bag error estimation

Cross-validating bagged predictors may lead to large computing efforts



OOB error:

For each **x** in the training set

find base classifiers, where the training set does not contain **x**

use these classifiers to make a prediction

Average the error over all samples

Randomization Injection

Inject some randomization into a standard learning algorithm (usually easy):

- Neural network: random initial weights
- Decision tree: when splitting, choose one of the top N attributes at random (uniformly)

Feature-Selection Ensembles

Key idea: Provide a different subset of the input features in each call of the learning algorithm

Bagged trees

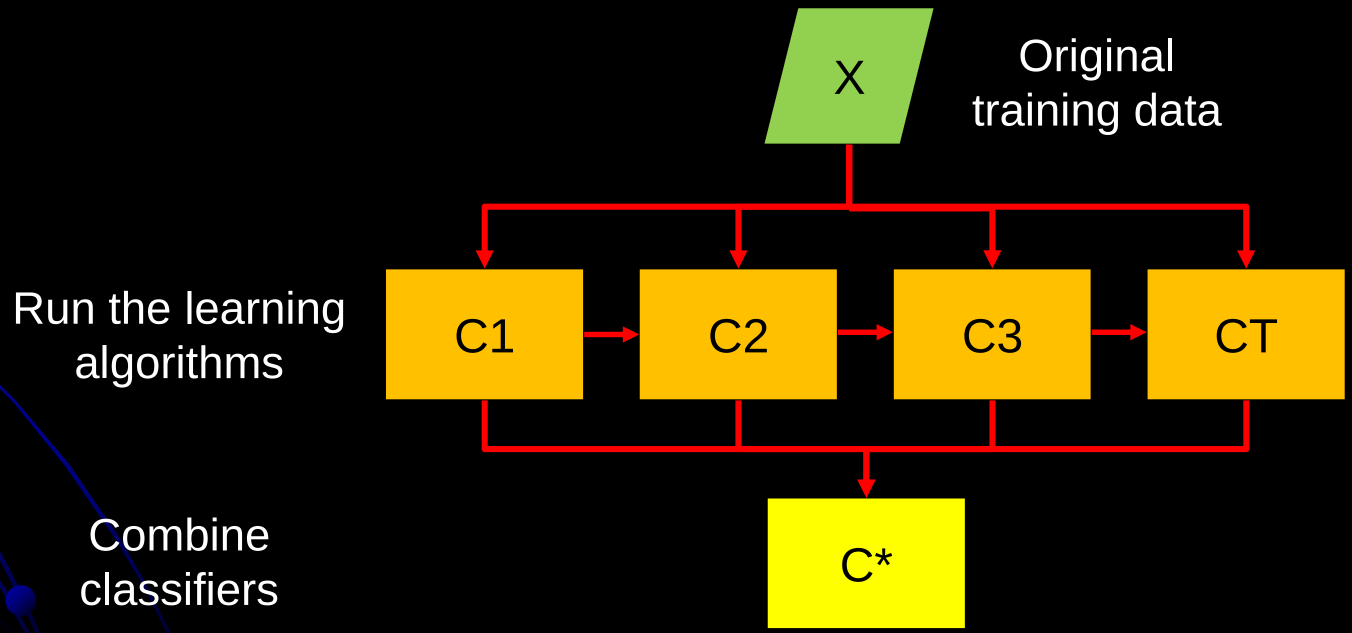
Ensemble of trees – decision forest

Improvement: random forests

Less correlated trees

Coordinated Construction of Ensembles

- Learn complementary classifiers
- Instance classification is realized by taking an weighted sum of the classifiers



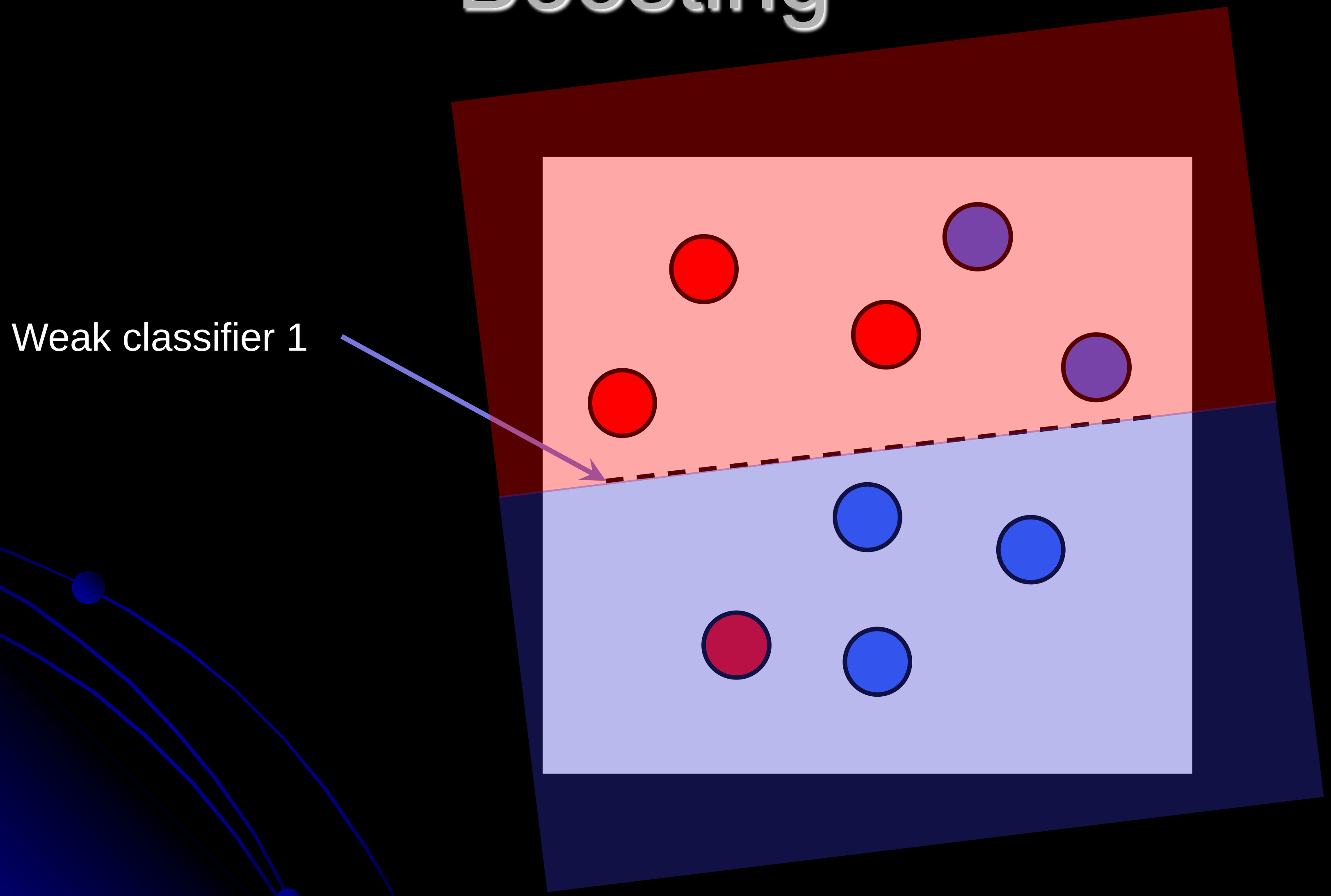
Coordinated Construction of Ensembles

- Adaptively change distribution of training data by focusing more on previously misclassified records
- Boosting
- Stacking

Boosting

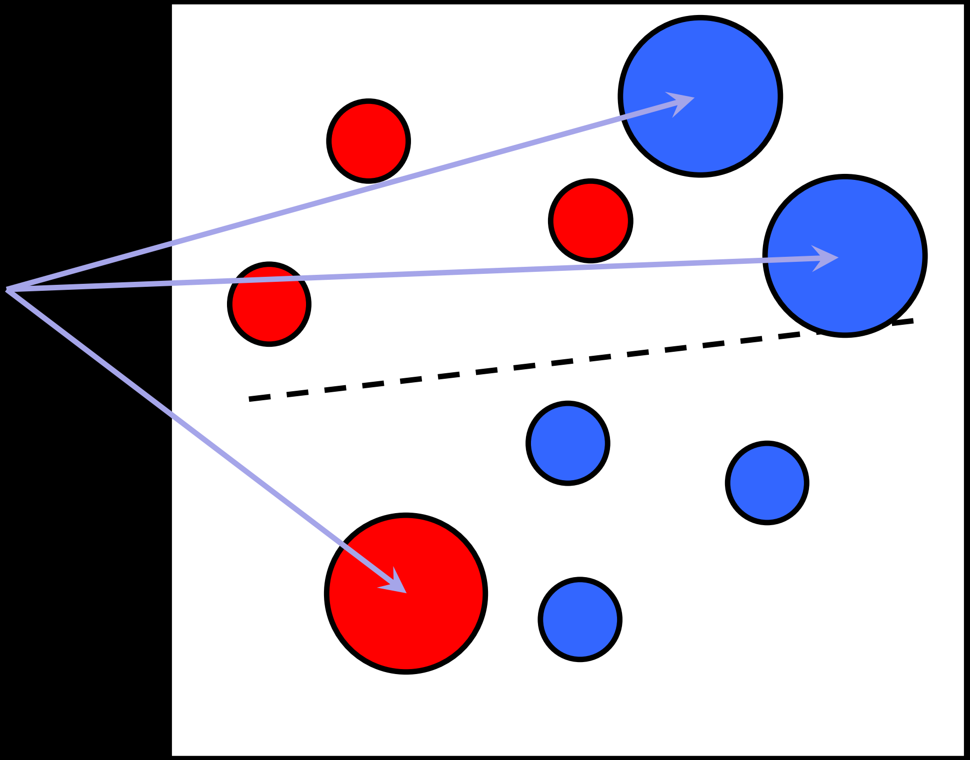
- Also uses voting but models are weighted according to their performance
- Iterative procedure: new models are influenced by performance of previously built ones
  - New model is encouraged to become expert for instances classified incorrectly by earlier models
  - Intuitive justification: models should be experts that complement each other
- There are several variants of this algorithm

Boosting

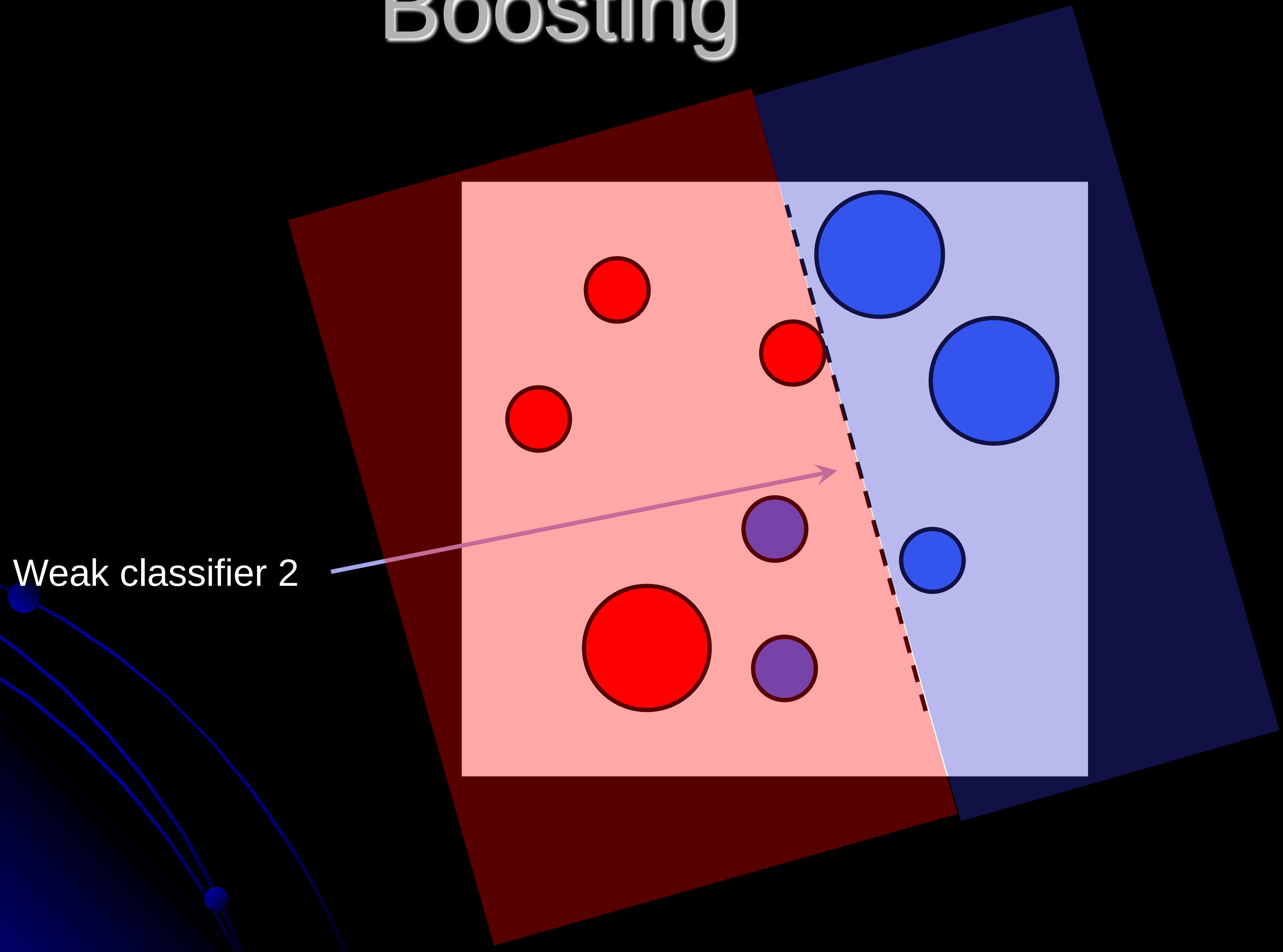


Boosting

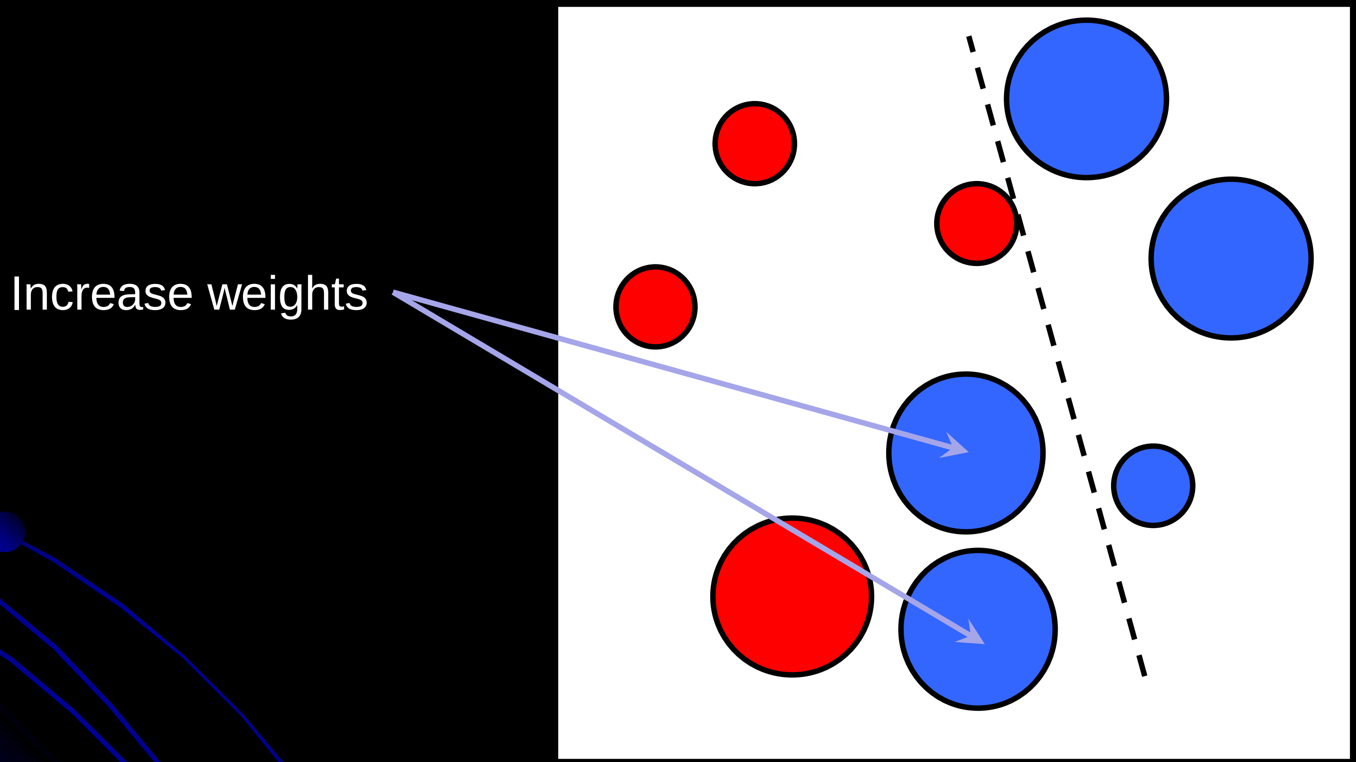
Increase weights



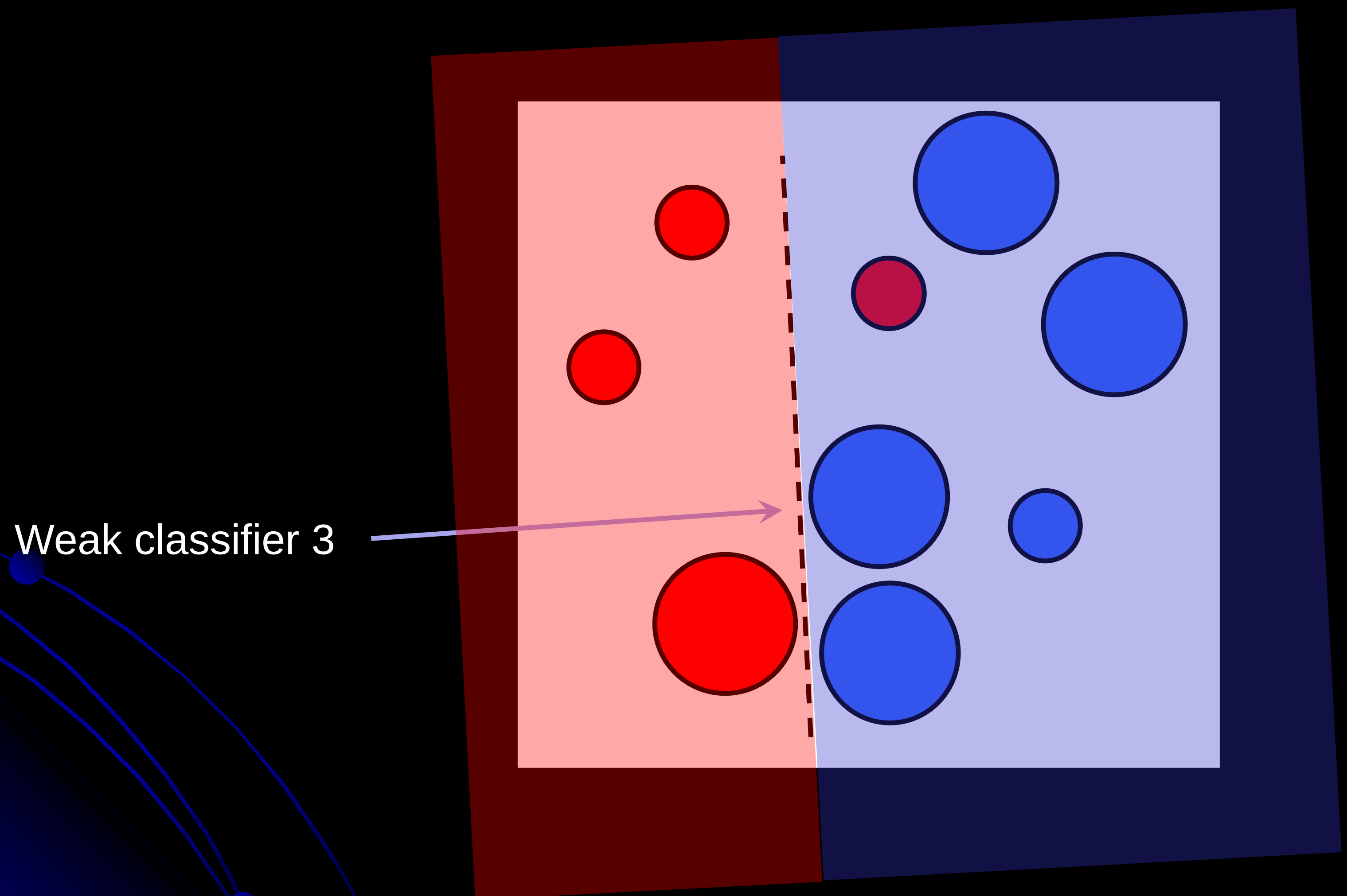
Boosting



Boosting

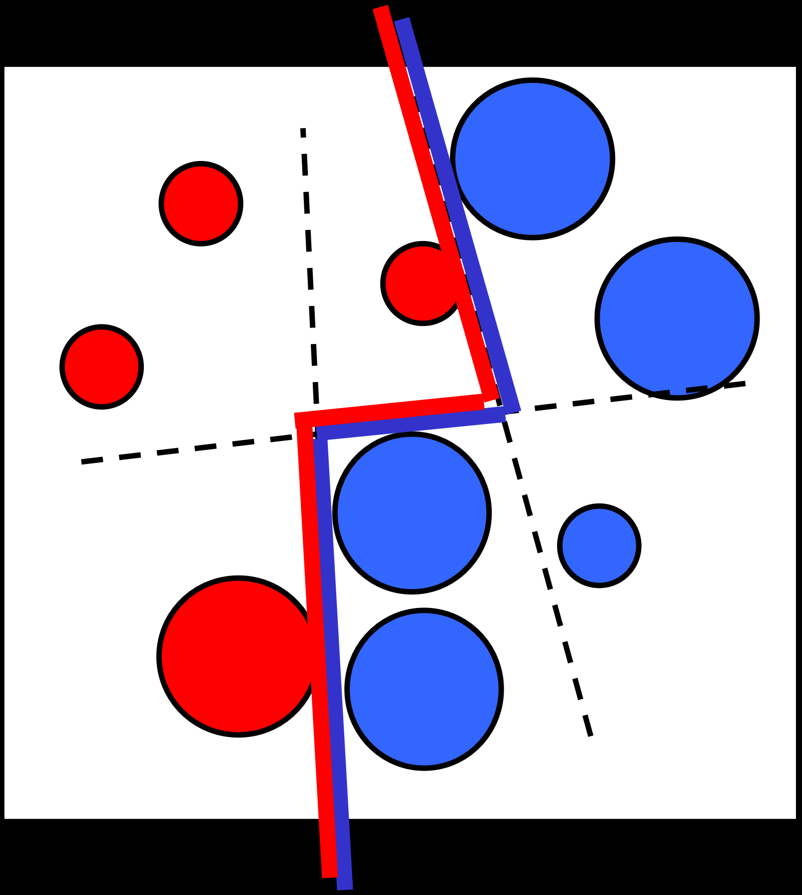


Boosting



Boosting

Strong classifier – linear combination of weak classifiers



AdaBoost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathscr{X}$, $y_i \in \{-1, +1\}$.
Initialize $\boxed{D_1(i) = 1/m \text{ for } i = 1, \ldots, m.}$ Initial Distribution of Data
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathscr{X} \to \{-1, +1\}$.

Train model

- Aim: select h_t with low weighted error:

$$\boxed{\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].}$$

Error of model

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

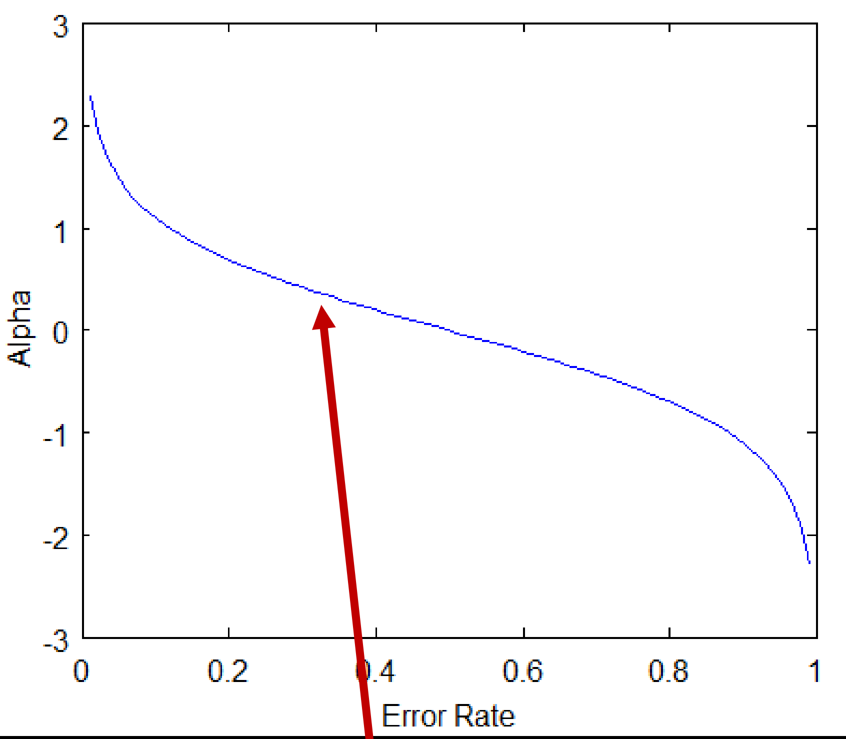
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf



aBoost

$_i \in \{-1,+1\}$.

.

$r_{i\sim D_t}\left[h_t(x_i) \neq y_i\right]$.

- Choose $\boxed{\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)}$. ← Coefficient of model
- Update, for $i = 1,\ldots,m$:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

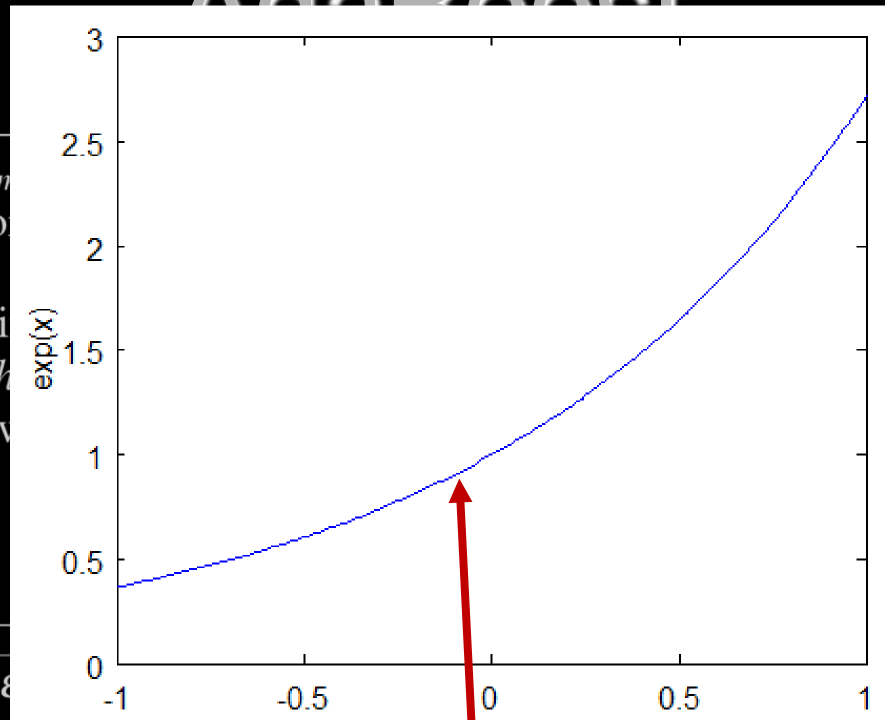
$$H(x) = \mathrm{sign}\left(\sum_{t=1}^{T}\alpha_t h_t(x)\right).$$

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

AdaBoost

Given: $(x_1, y_1), \ldots, (x_m, y_m$
Initialize: $D_1(i) = 1/m$ fo
For $t = 1, \ldots, T$:

- Train weak learner usi
- Get weak hypothesis h
- Aim: select h_t with lov



- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 -}{\varepsilon} \right.$
- Update, for $i = 1, \ldots, m$:

$$\boxed{D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}}$$

Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$\boxed{H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x) \right).}$$

Final average

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

AdaBoost

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathscr{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathscr{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

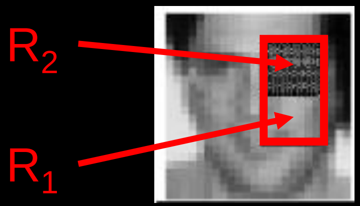
$$H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

AdaBoost in CV

Viola-Jones face detector

Image features: $f(Im) = \sum_{x \in R_1} I(x) - \sum_{x \in R_2} I(x)$



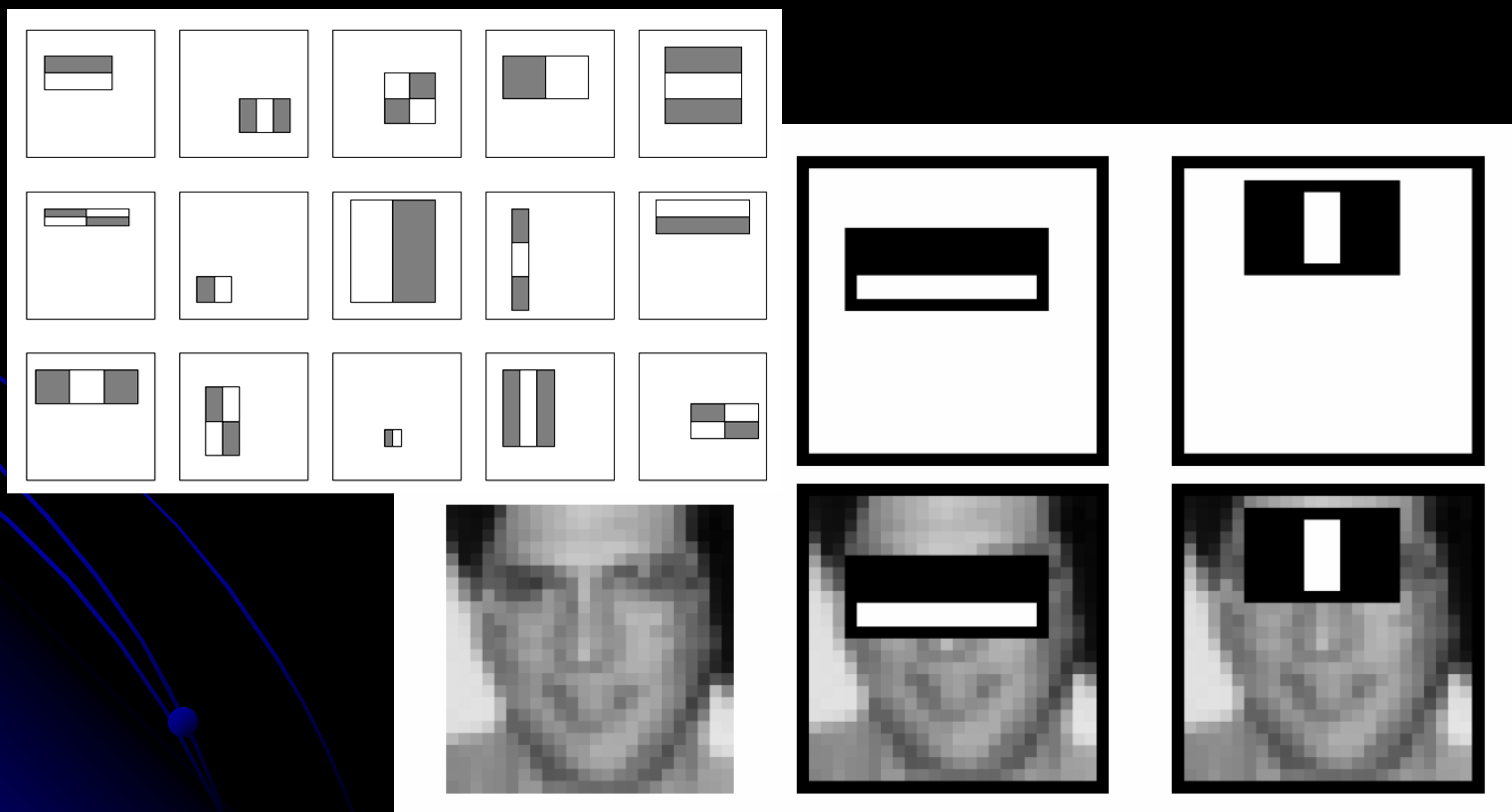
Features – Haar wavelets

Weak classifier: $h(Im) = \begin{cases} 1, & \text{if } f(Im) > \theta \\ -1, & \text{otherwise} \end{cases}$

AdaBoost in CV

Most discriminative features

Cascade of boosted classifiers (1, 10, 25, 25, 50…)



Remarks on Boosting

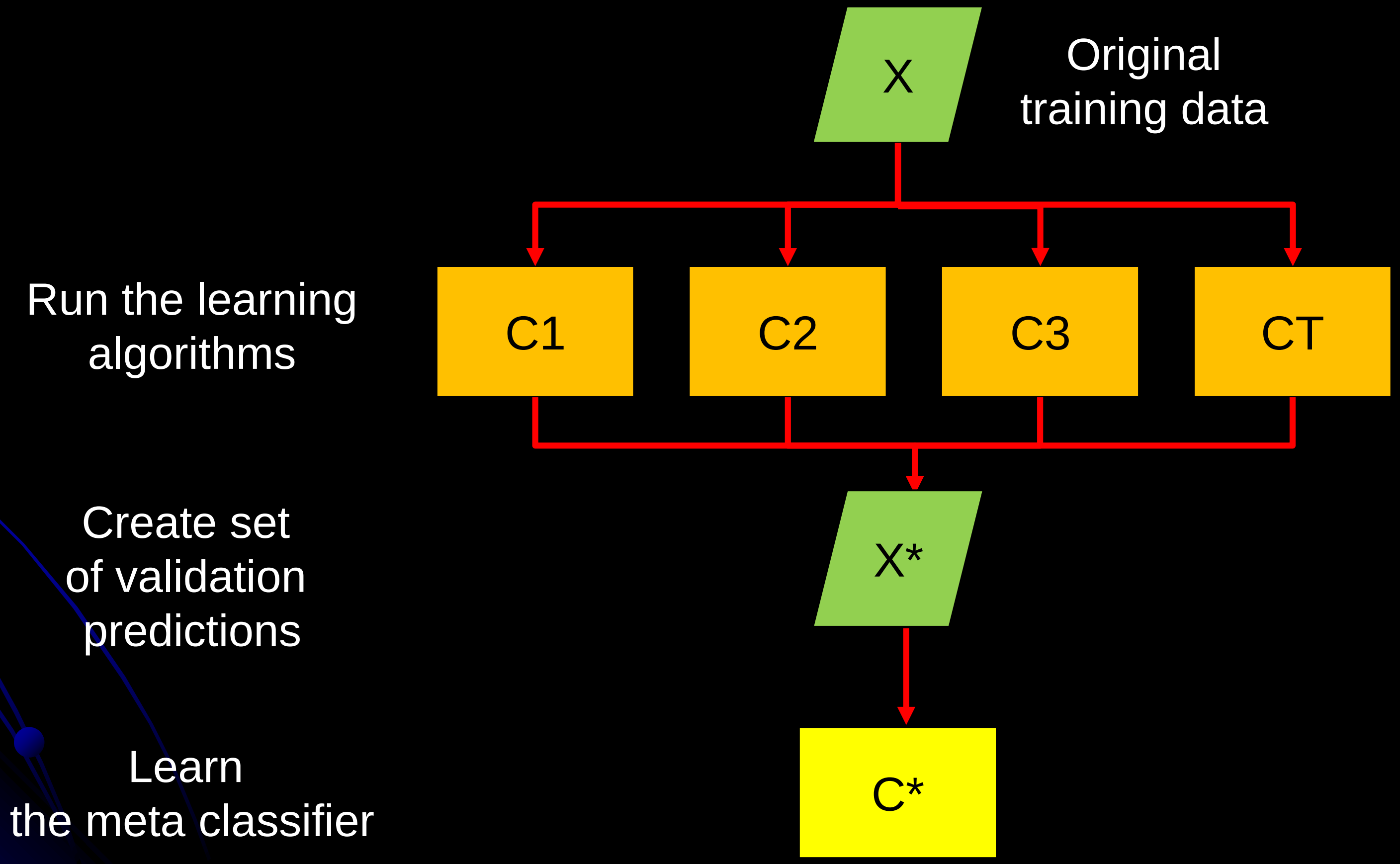
- Boosting can be applied without weights using re-sampling with probability determined by weights
- Boosting decreases exponentially the training error in the number of iterations
- Boosting works well if base classifiers are not too complex and their error doesn’t become too large too quickly
- Boosting reduces the bias component of the error of simple classifiers

Stacking

- Uses meta learner instead of voting to combine predictions of base classifiers
- Predictions of base classifiers (*level-0 models*) are used as input for meta classifier (*level-1 model)*
- Method for generating base classifiers usually apply different learning schemes

Stacking

If the base classifiers provide class probabilities, it is better to use them



Artificial Neural Networks

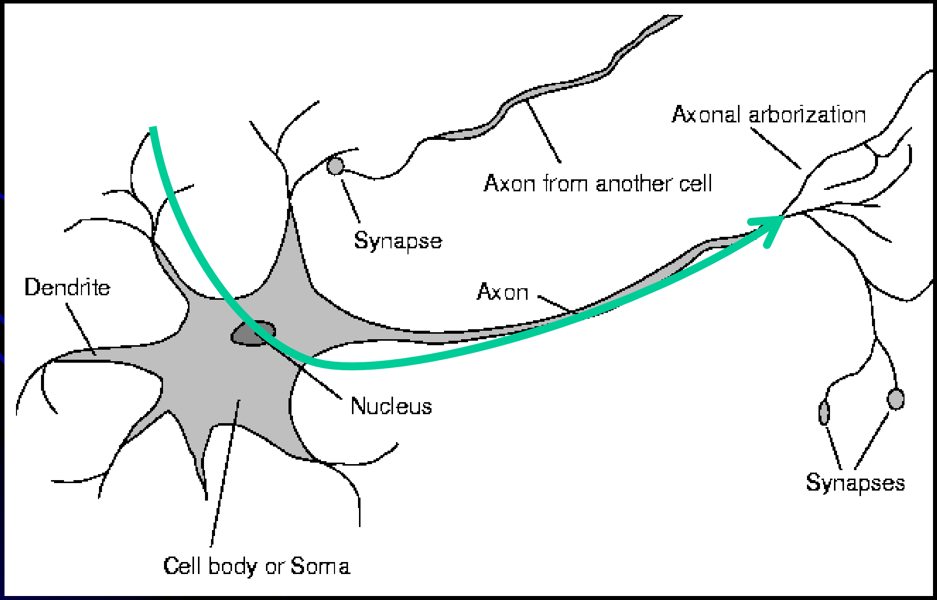
Neurons

Biological inspiration: A neuron

Dendrite – accepts signal from other neurons

Soma – integrates the signals

Axon – outputs the signal to other neurons



Neurons

The soma may give rise to numerous dendrites, but never to more than one axon.

A synapse is a contact between the axon of one neuron and a dendrite or soma of another.

Synaptic signals may be excitatory or inhibitory. If the net excitation received by a neuron over a short period of time is large enough, the neuron generates a brief pulse called an action potential, which originates at the soma and propagates rapidly along the axon, activating synapses onto other neurons as it goes.

McCulloch and Pitts logic neuron (1943)

Inputs and output are binary

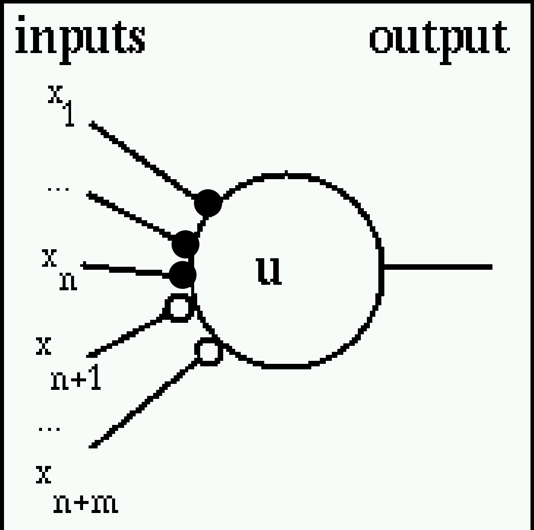
A set of n excitatory inputs, x_i

A set of m inhibitory inputs, x_{n+j}

A threshold, u

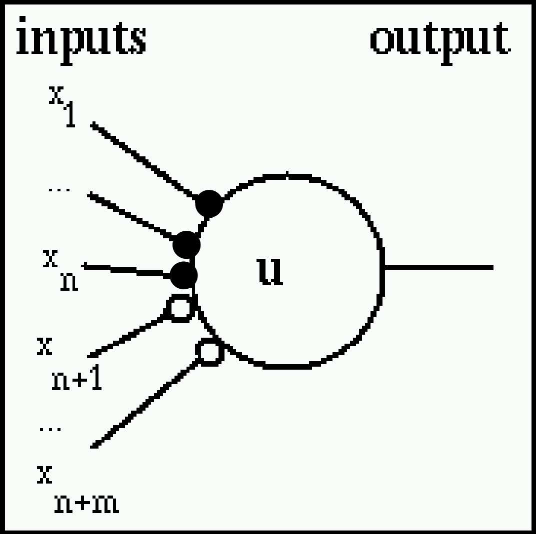
A unit step activation function

A single neuron output, y



McCulloch and Pitts model

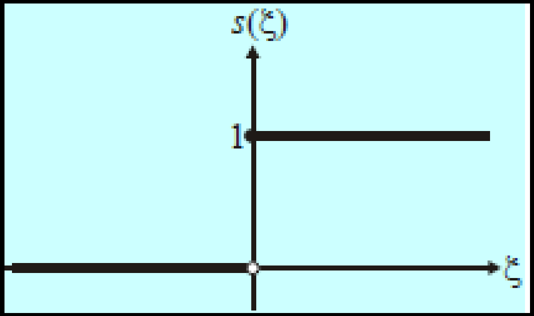
$$y = \begin{cases} 1, & \sum_{i=1}^{n} x_i - \sum_{j=1}^{m} x_{n+j} \geq u \\ 0, & \sum_{i=1}^{n} x_i - \sum_{j=1}^{m} x_{n+j} < u \end{cases}$$



McCulloch and Pitts model

A unit step activation function

$$s(\xi) = \begin{cases} 1, \xi \geq 0 \\ 0, \xi < 0 \end{cases}$$

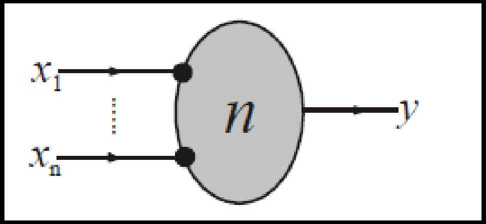


$$y = s\left(\sum_{i=1}^{n} x_i - \sum_{j=1}^{m} x_{n+j} - u\right)$$

Alternatively we can express the neuron activity by ± 1 weight coefficients

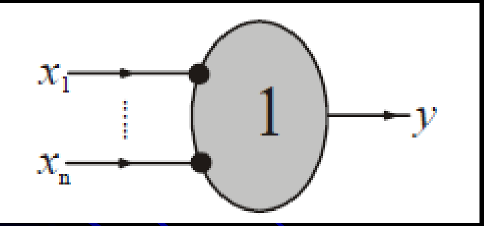
$$y = s(\sum_{i=1}^{n+m} w_i x_i - u)$$

Boolean functions by M-P neurons



AND

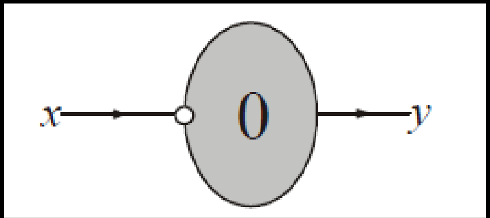
$$\sum_{i=1}^{n} x_i \geq n$$



OR

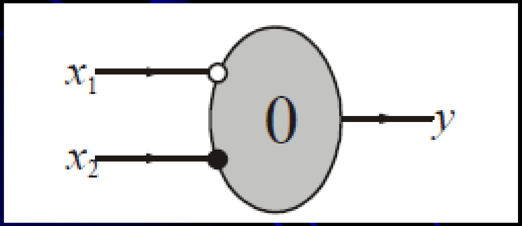
$$\sum_{i=1}^{n} x_i \geq 1$$

Boolean functions by M-P neurons



NOT

$$-x \geq 0$$



implication

$$-x_1 + x_2 \geq 0$$

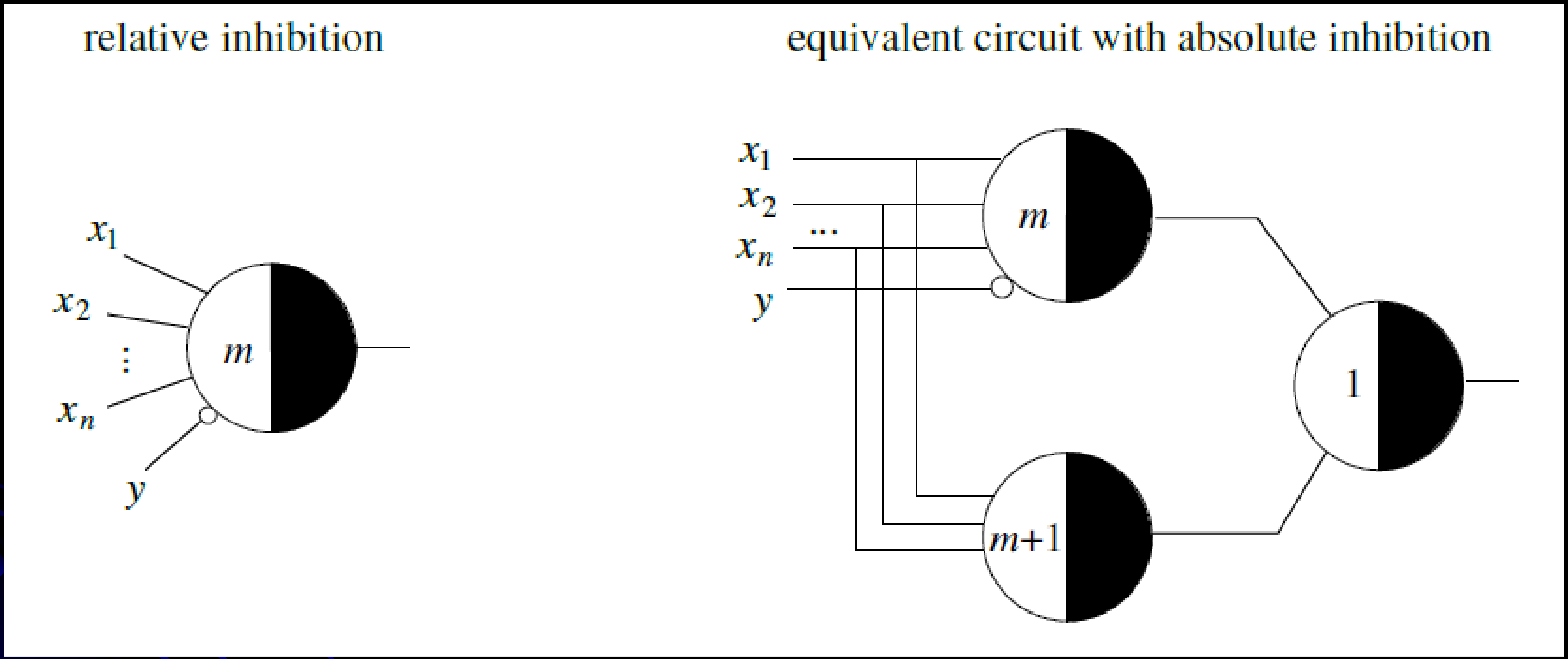
x_1	x_2	$-x_1 + x_2$	y
0	0	0	1
0	1	1	1
1	0	-1	0
1	1	0	1

M-P inhibition

- Original M-P units: absolute inhibition
- If at least one of the inhibitory signals is 1, the unit is inhibited and the result of the computation is 0.
- Networks with absolute inhibition are equivalent to networks with relative inhibition.

Raul Rojas Neural Networks - A Systematic Introduction, Springer-Verlag, Berlin, New-York, 1996
https://page.mi.fu-berlin.de/rojas/neural/chapter/K2.pdf

M-P inhibition



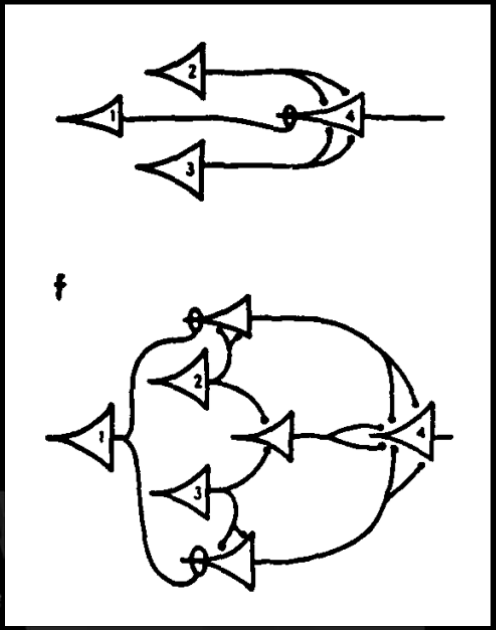
Relative ⇨ Absolute?

McCulloch and Pitts networks

- Weights and thresholds in neurons are given to compute a certain Boolean operation – no possibility to learn
- The networks are designed to compute arbitrary Boolean operation
- All logical functions can be implemented with a network composed of units which exclusively compute the AND, OR, and NOT functions

McCulloch and Pitts networks

- The networks are built from fixed building blocks – neurons with specified properties
- The networks do not learn to complete a given task



Hebbian learning (1949)

Two neurons which are simultaneously active should develop a degree of interaction higher than those neurons whose activities are uncorrelated

1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.

2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

activity product rule $\Delta w_{ij} = \eta x_i y_j$

Rosenblatt perceptron (1957)

Mark 1 Perceptron

a vision machine

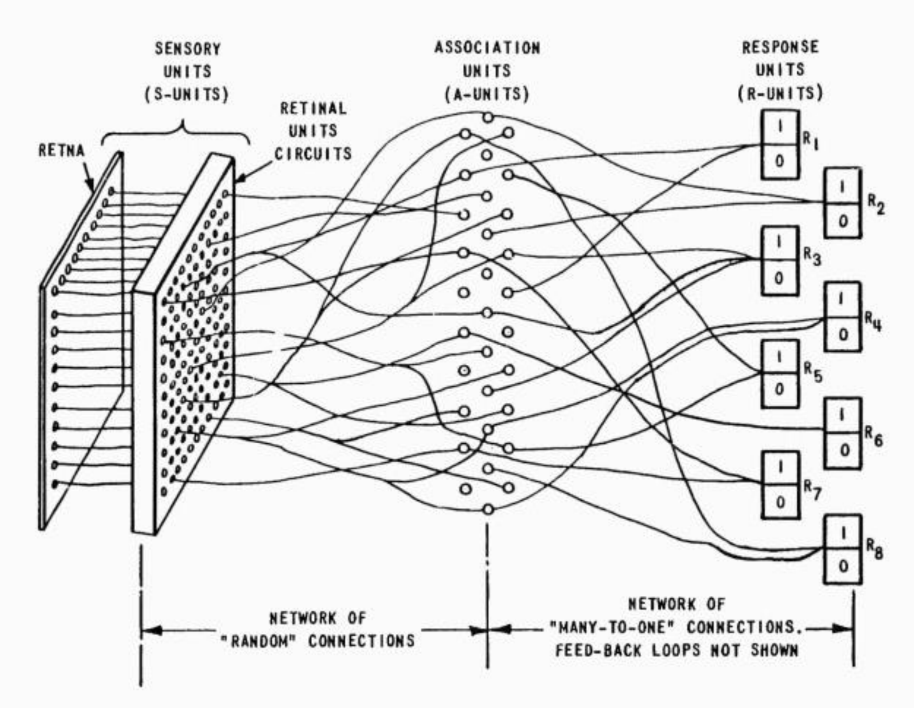
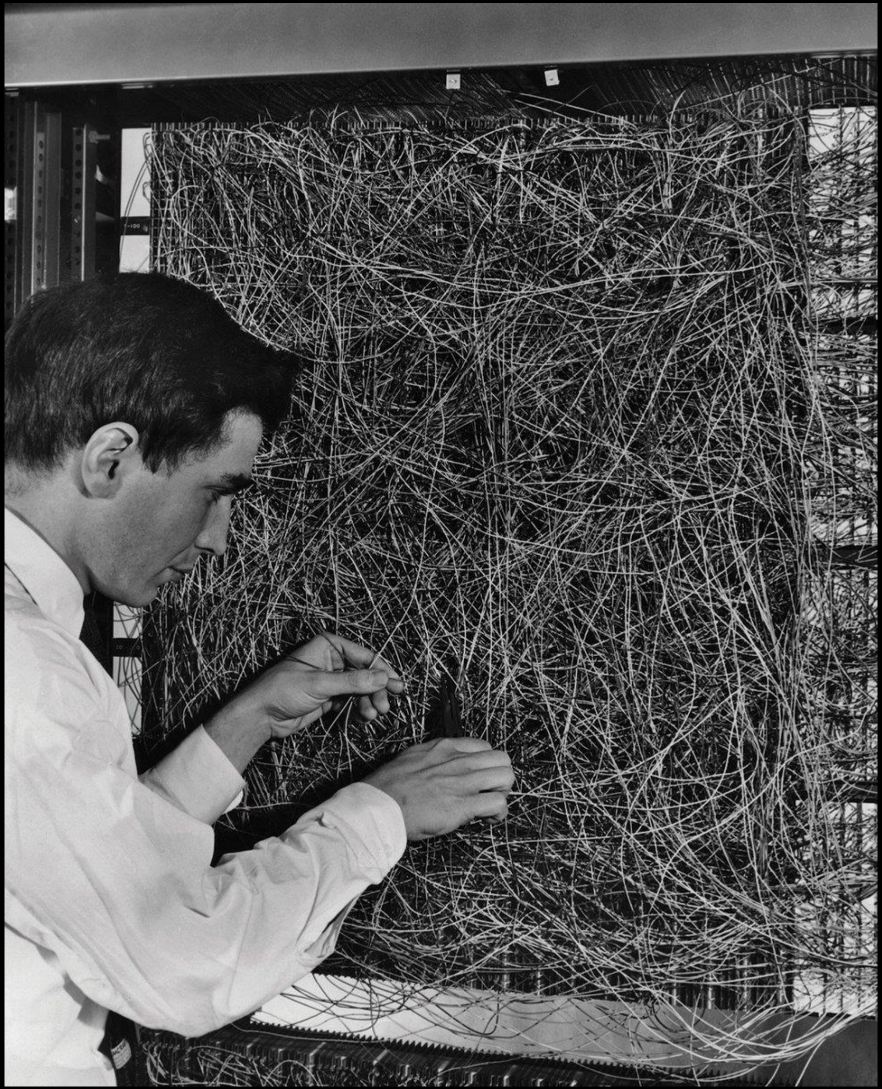


Figure I ORGANIZATION OF THE MARK I PERCEPTRON

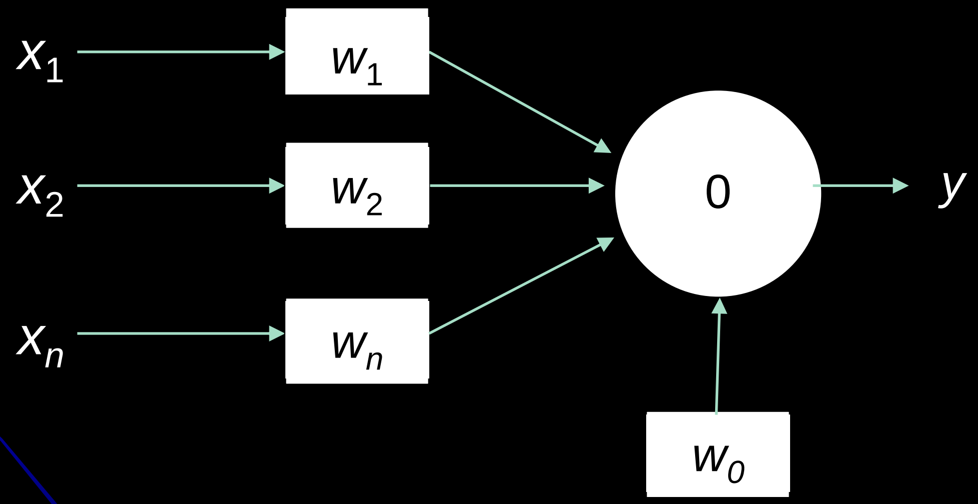


http://www.glass-bead.org/article/machines-that-morph-logic/?lang=enview

Rosenblatt perceptron (1957)

Adjustable weights

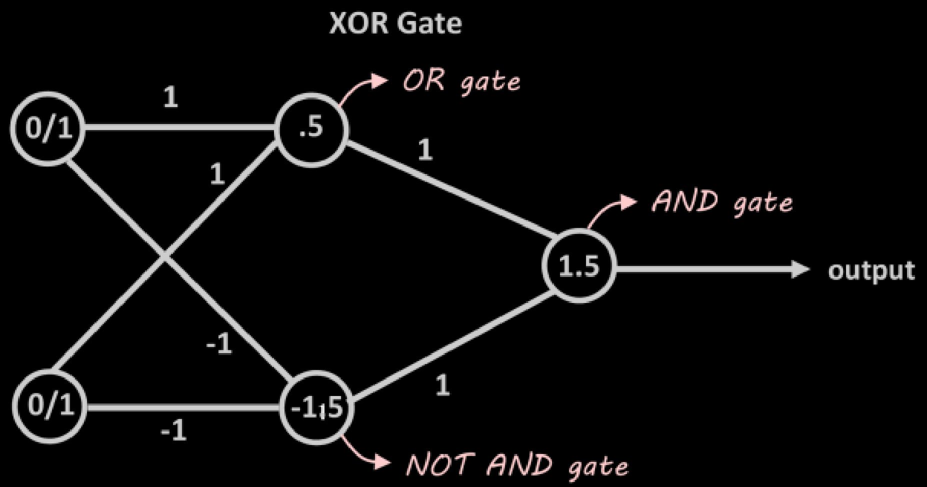
Linear classifier (previous class)



Nonseparable data?

1969 Minsky - XOR problem

Solution: more layers



No learning algorithm for multilayer network

Problem too complex

Multilayer network learning

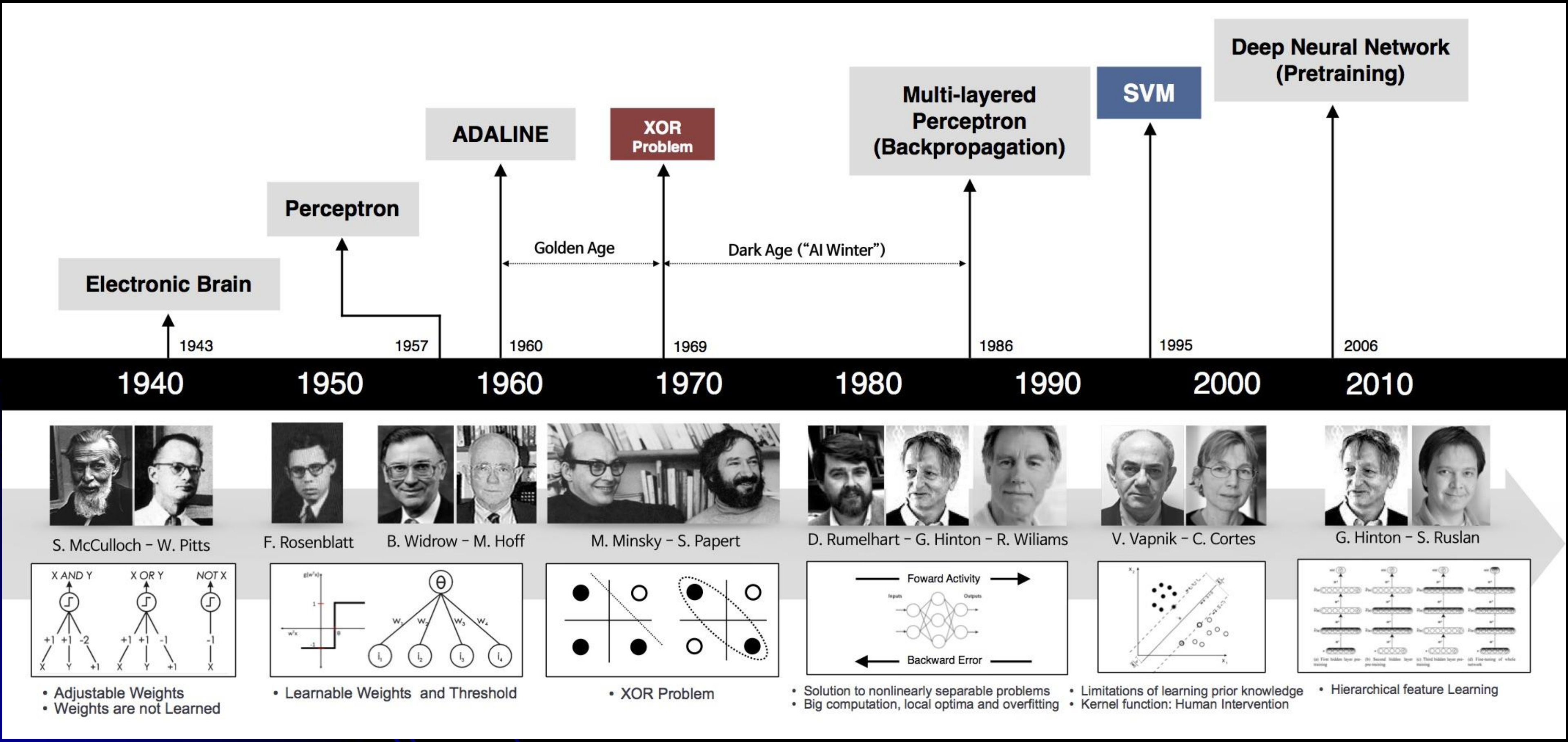
Error backpropagation

Worked on since 60-ies

80-ies – used in NN

(Rumelhart et al., 1986) popularization

A bit of history



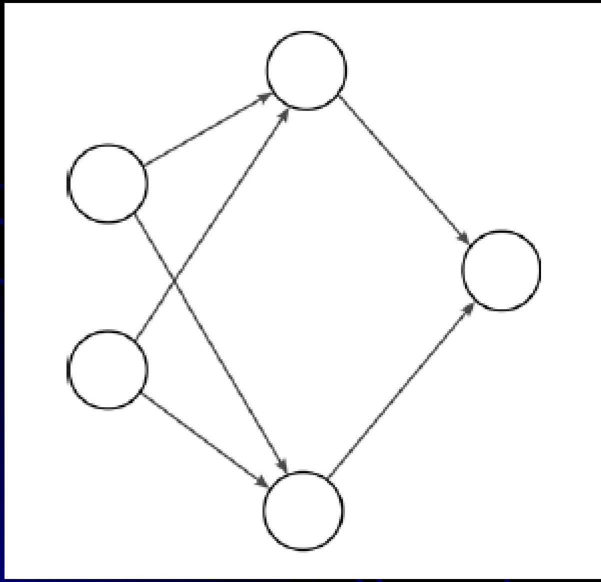
https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Artificial neural networks

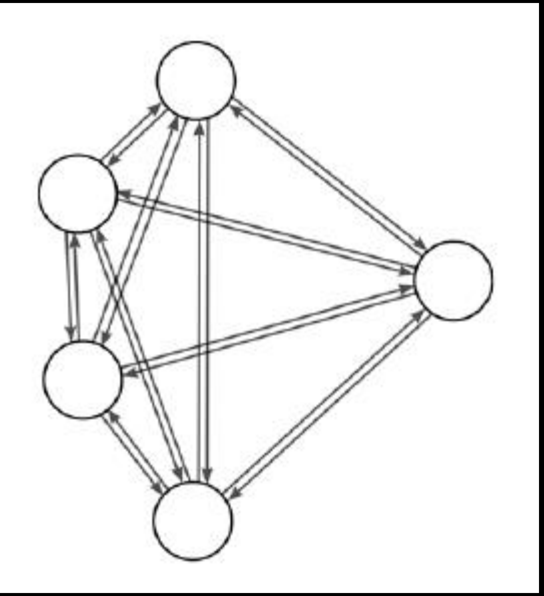
Networks of connected nodes

Oriented connections

Can have various topologies



Feed forward

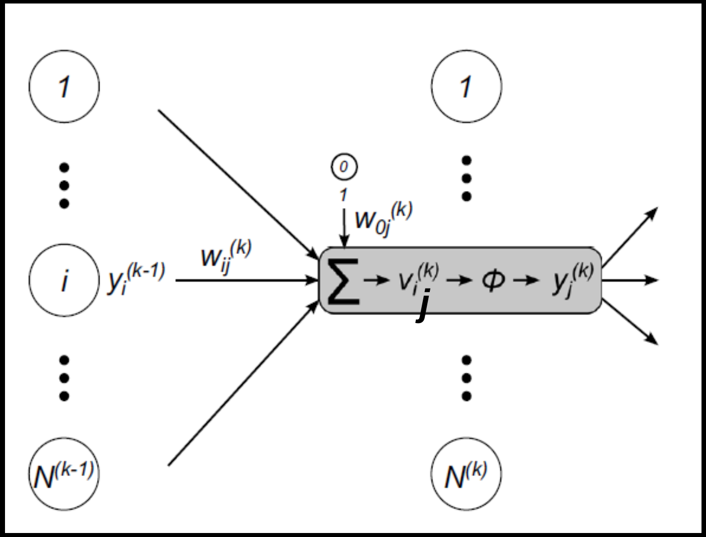


Feed forward ANN procedure

The outputs from the previous layer are weighted and summed

Nonlinear activation function φ is applied on the sum

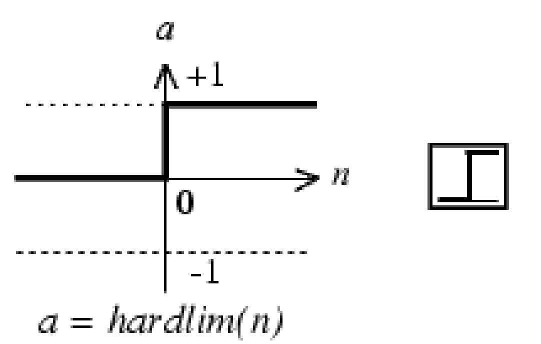
The output is sent to the next layer



Activation function

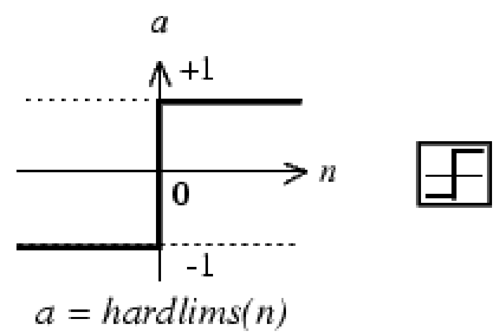
Step function

$$\varphi(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$



Hard-Limit Transfer Function

$$\varphi(x) = \begin{cases} 1, x \geq 0 \\ -1, x < 0 \end{cases}$$



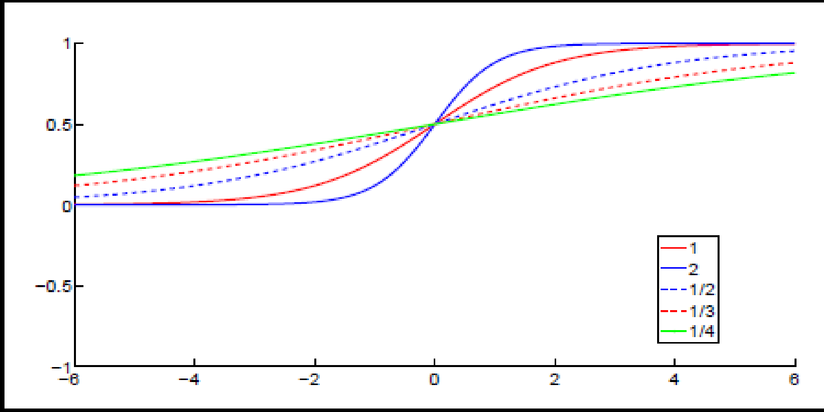
Symmetric Hard-Limit Transfer Function

Activation function

Continuous (and quick) change

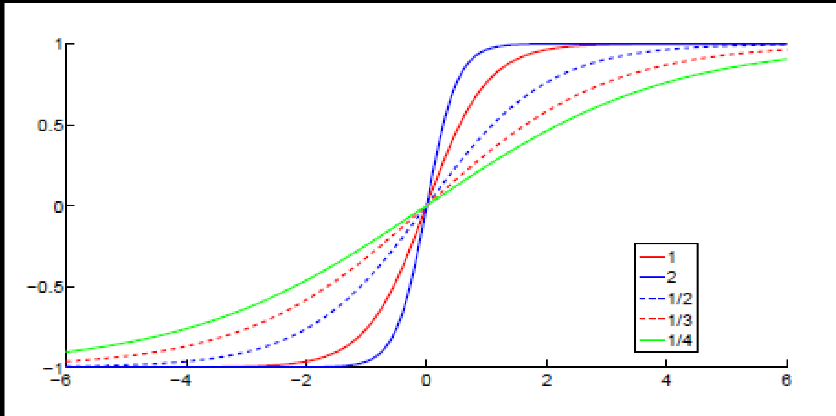
Logistic fn (sigmoid for $\alpha = 1$)

$$\varphi(\alpha, x) = \frac{1}{1+e^{-\alpha x}}$$

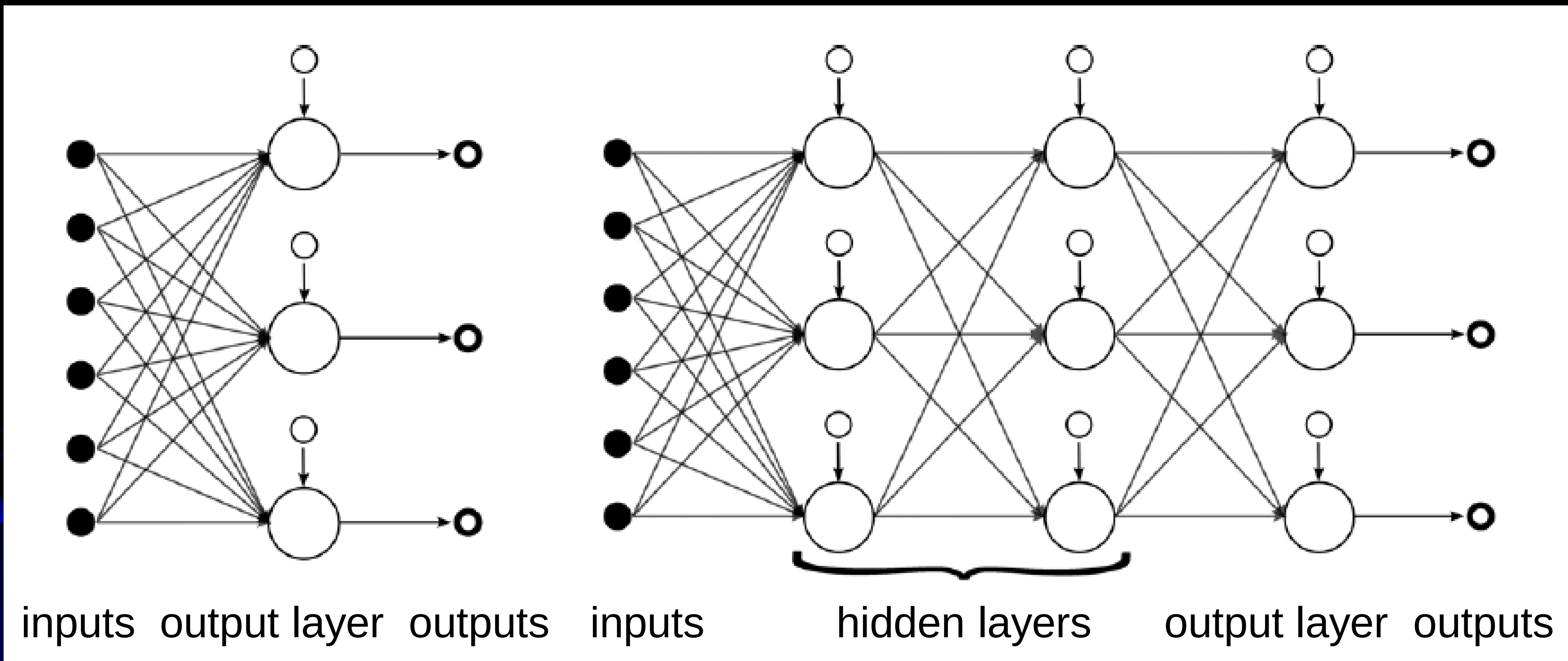


hyperbolic tangent

$$\varphi(\beta, x) = \tanh(\beta x) = \frac{e^{\beta x} - e^{-\beta x}}{e^{\beta x} + e^{-\beta x}}$$

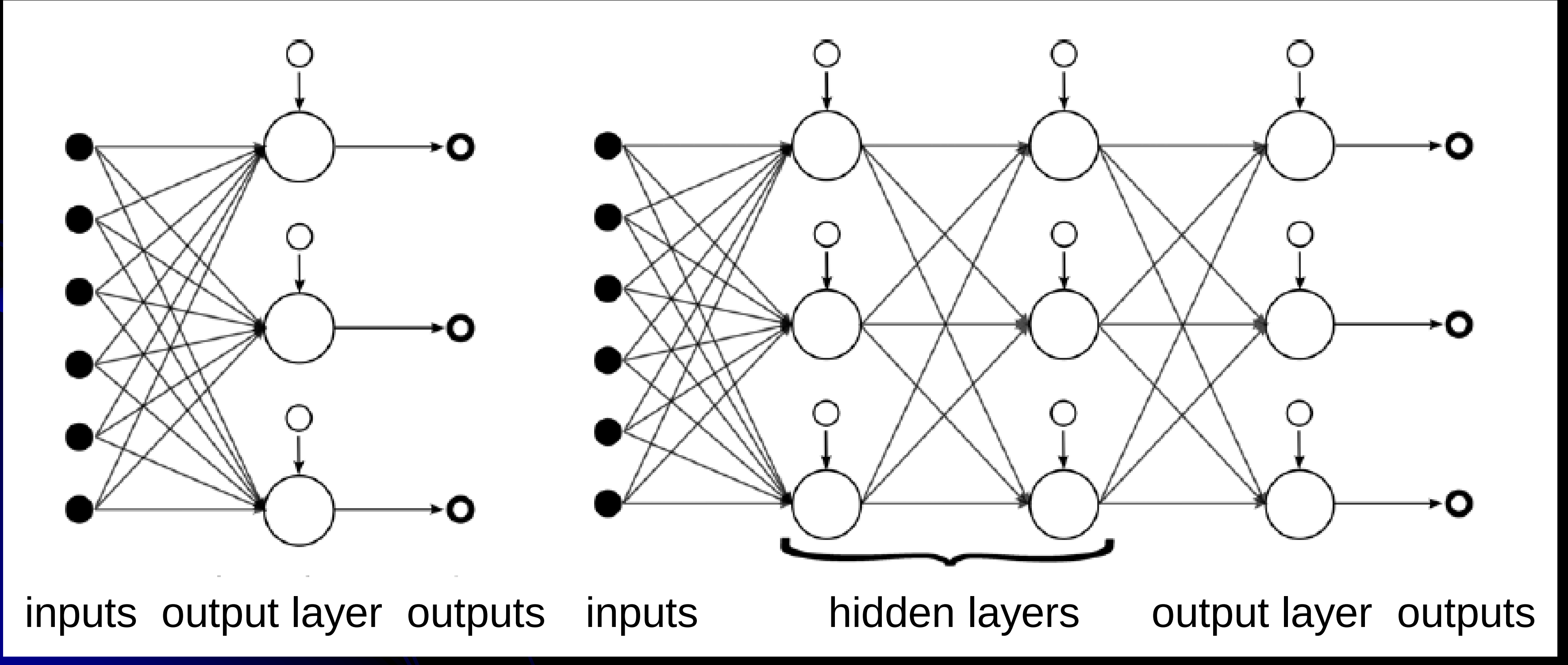


Multilayer perceptron



ANN learning

Deriving the weight matrix
(weight vectors for neurons in all layers)

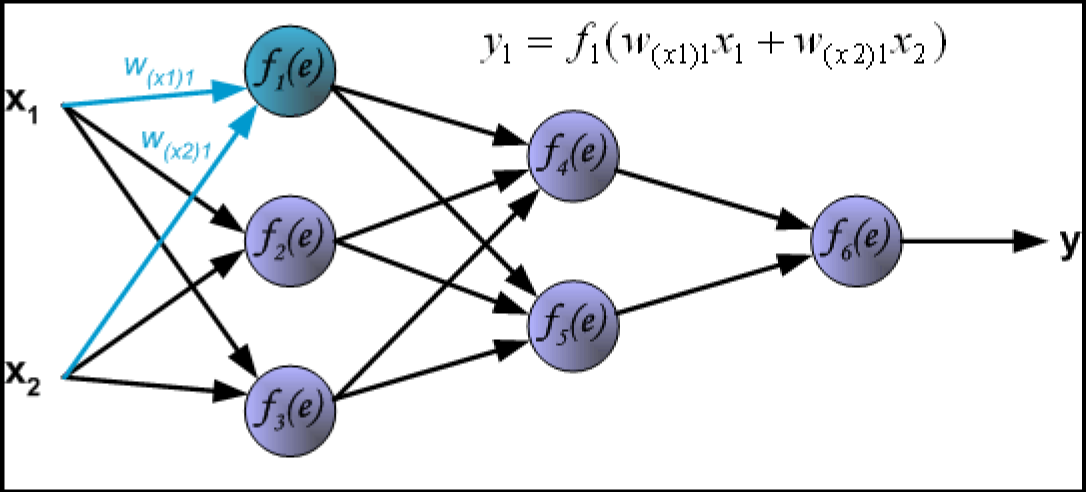


ANN learning

1. Random initialization of $\mathbf{W}$
2. Feature vectors arrive at the input layer, bubble through the network to the output layer
3. Output vector is compared against the expected classification output and the value of the objective function E is computed
4. The error is back-propagated through the network and the weights are adjusted

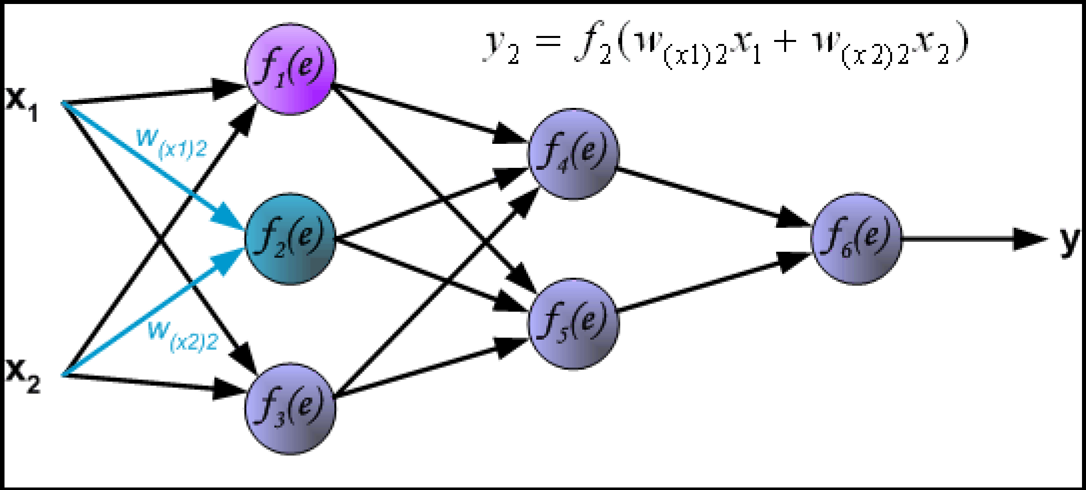
$$\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W}$$
$$\Delta\mathbf{W} = -\eta \nabla E\Big|_{\mathbf{W}}$$

ANN learning

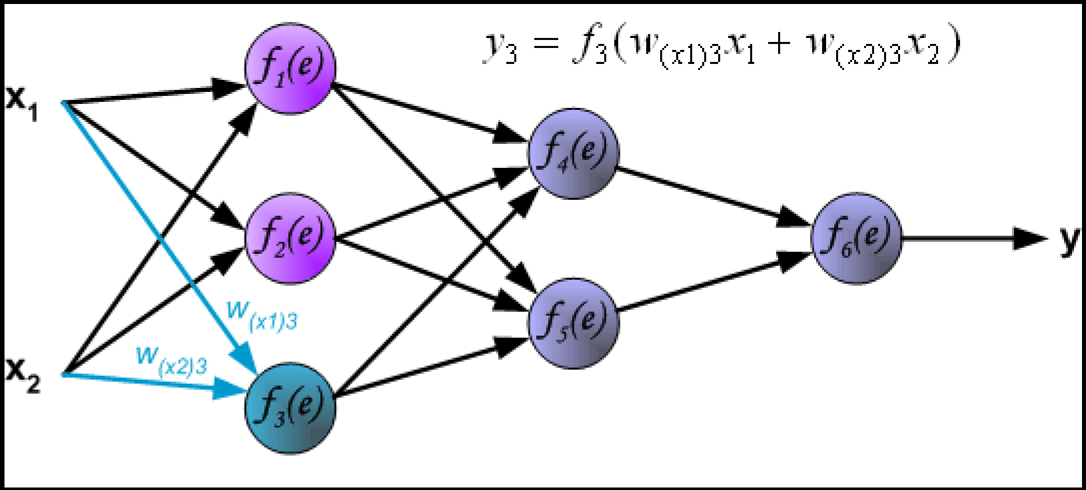


http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

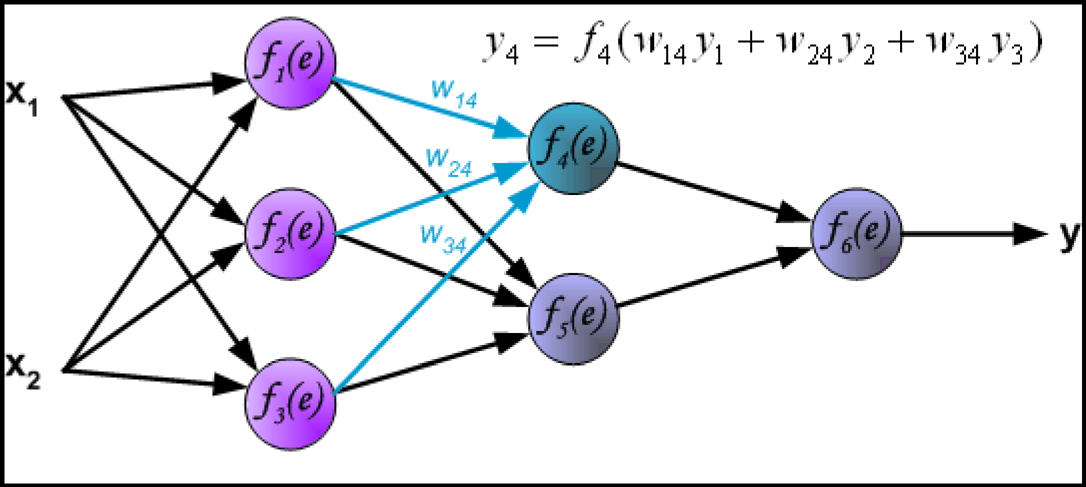
ANN learning



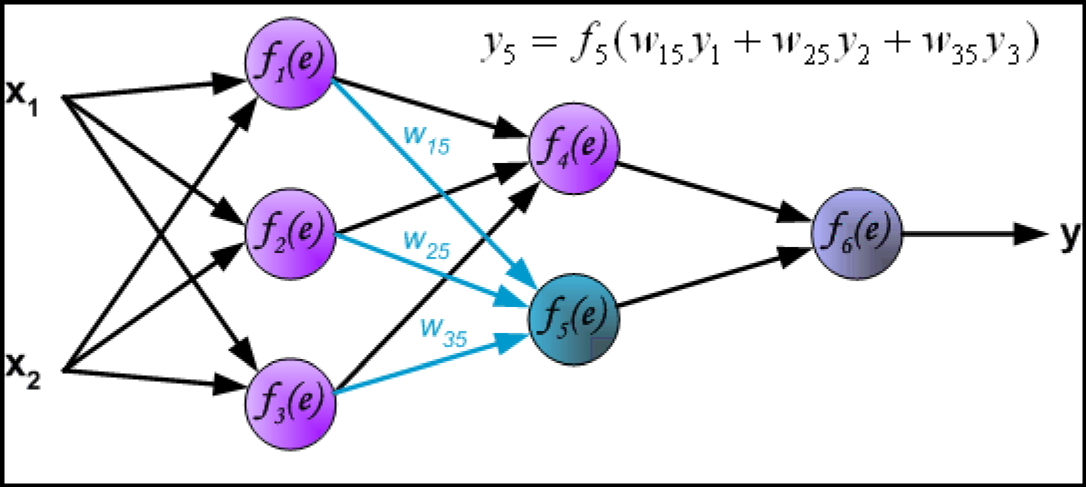
ANN learning



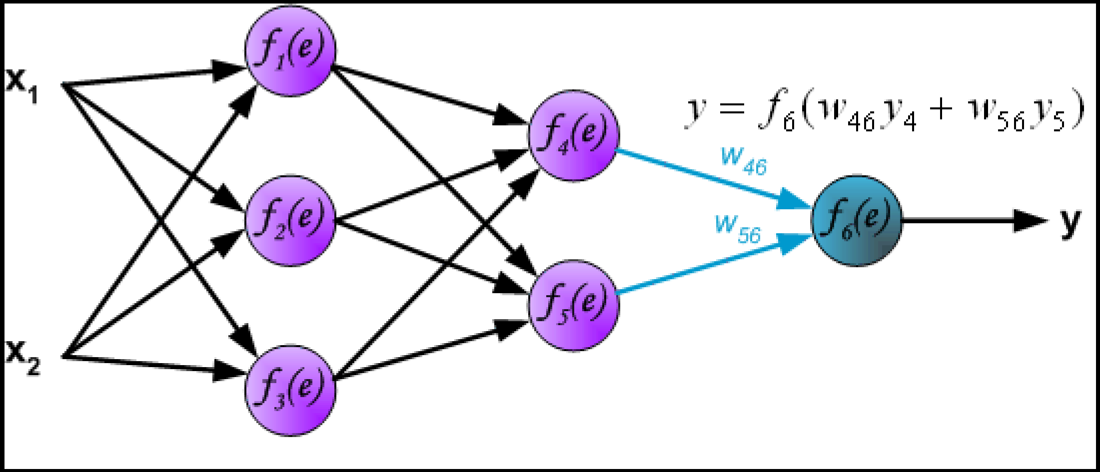
ANN learning



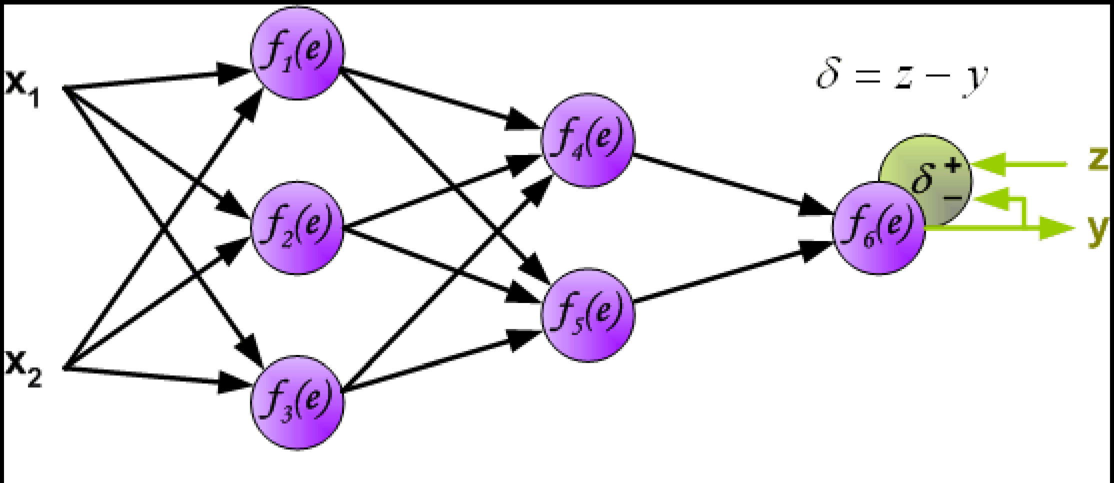
ANN learning



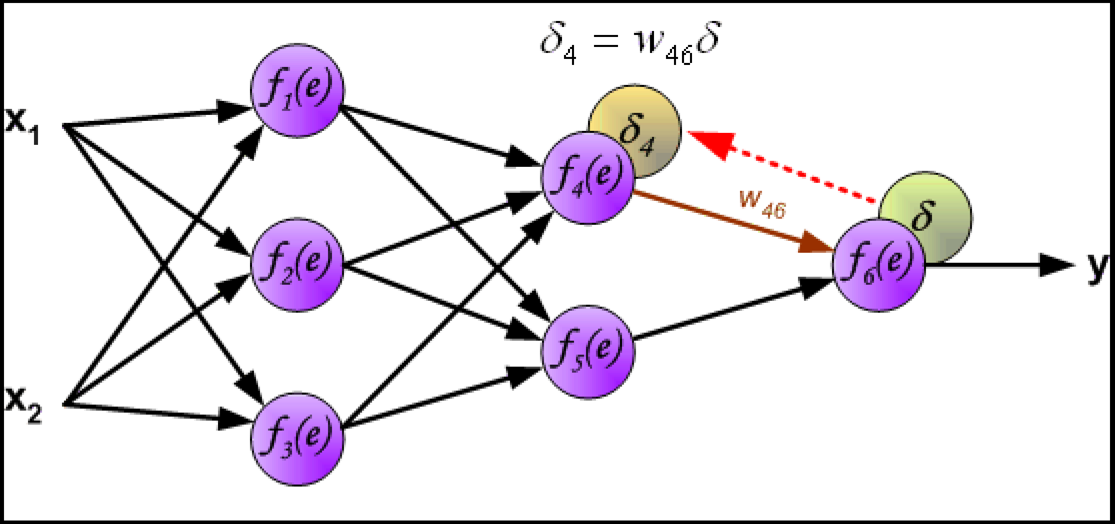
ANN learning



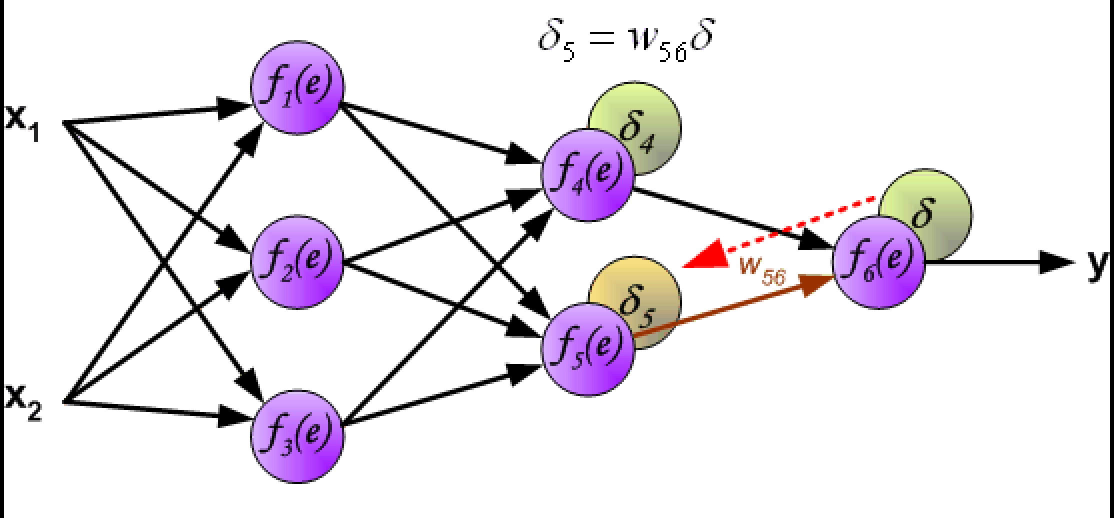
ANN learning



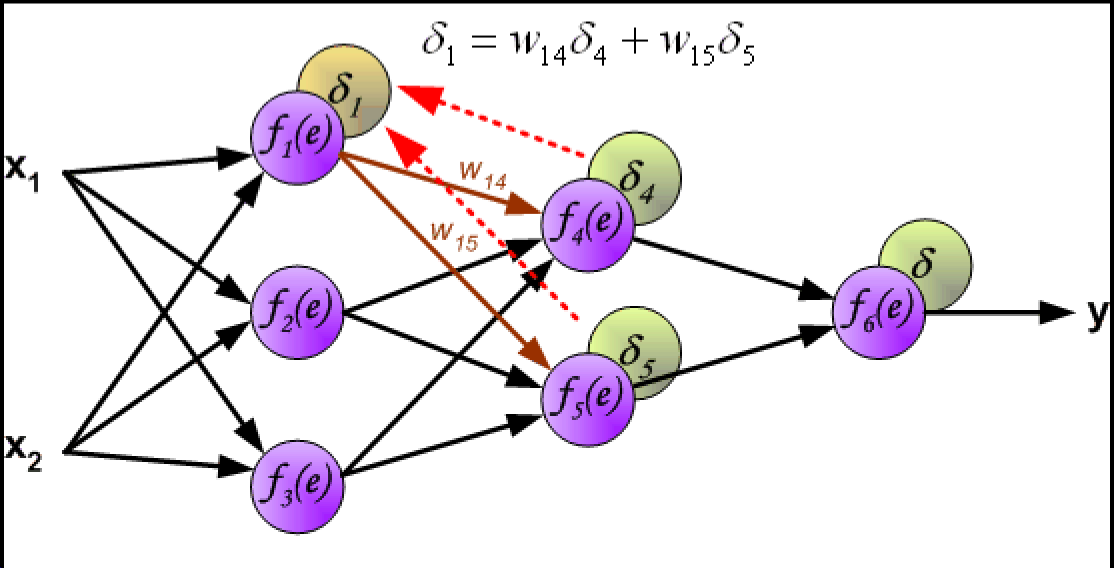
ANN learning



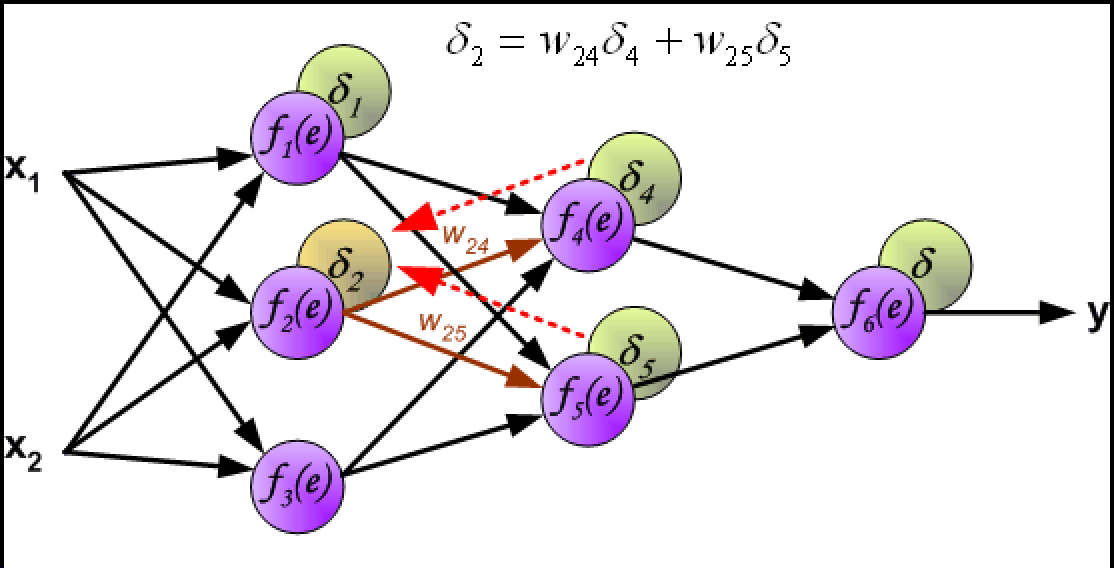
ANN learning



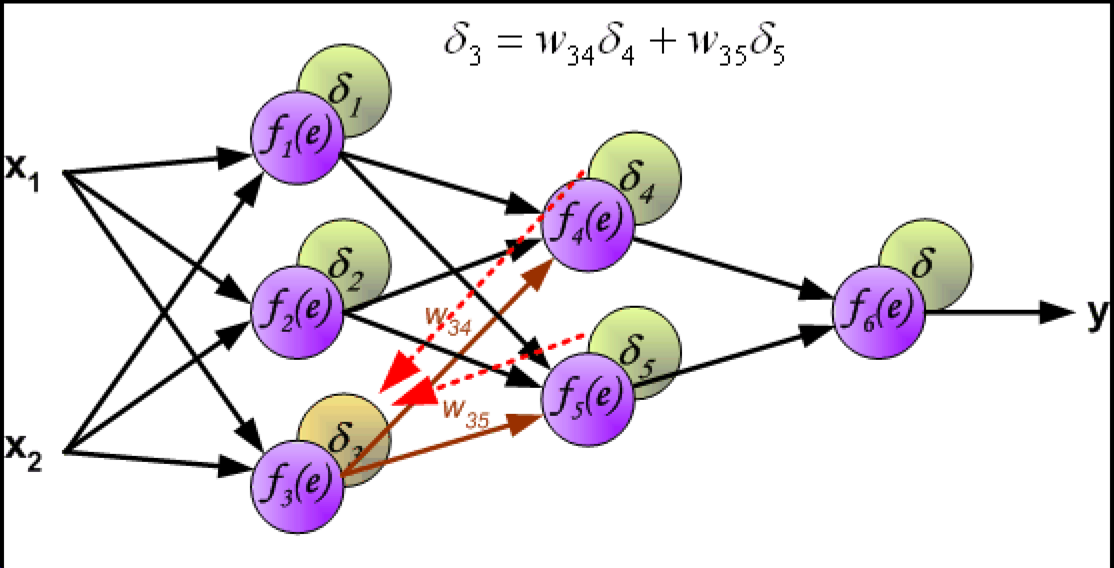
ANN learning



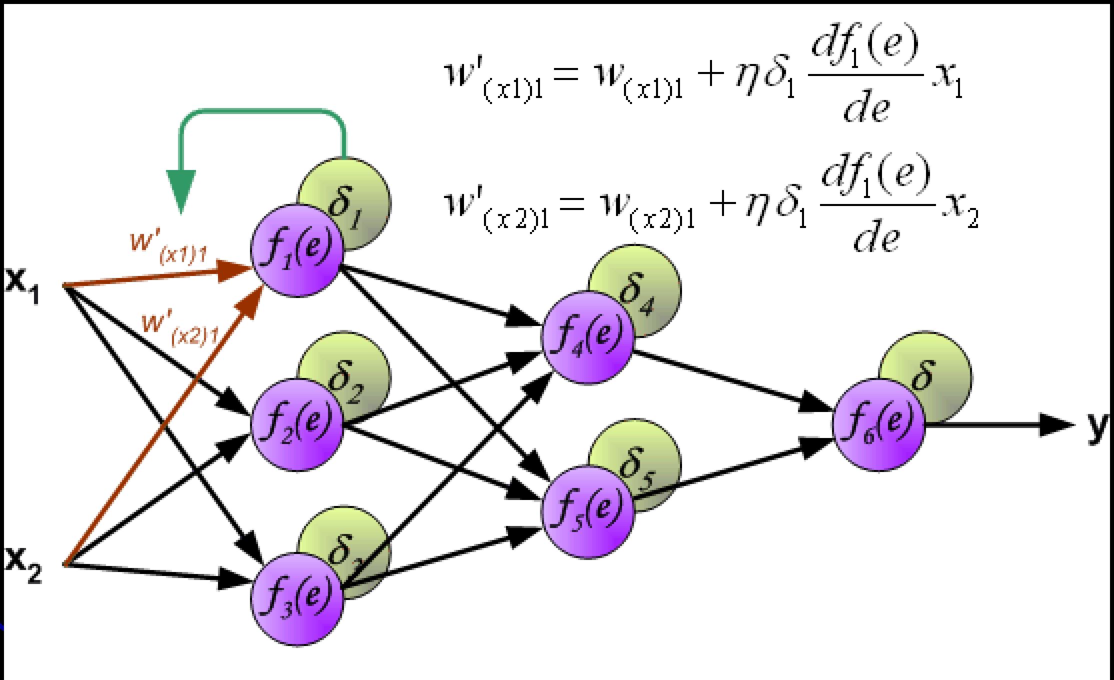
ANN learning



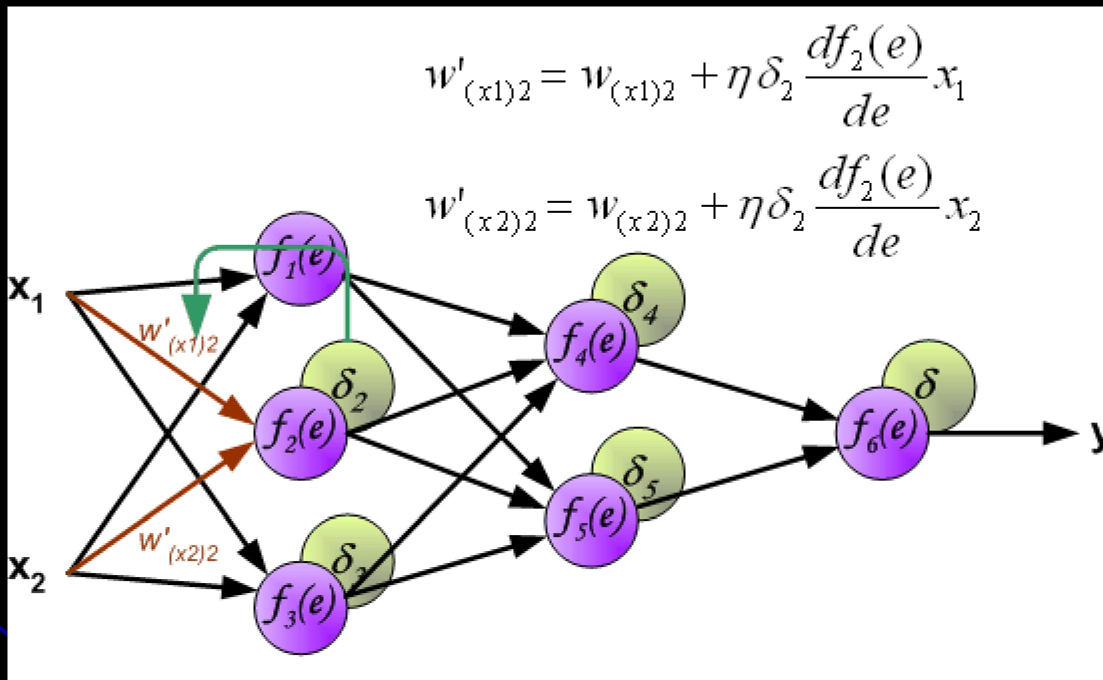
ANN learning



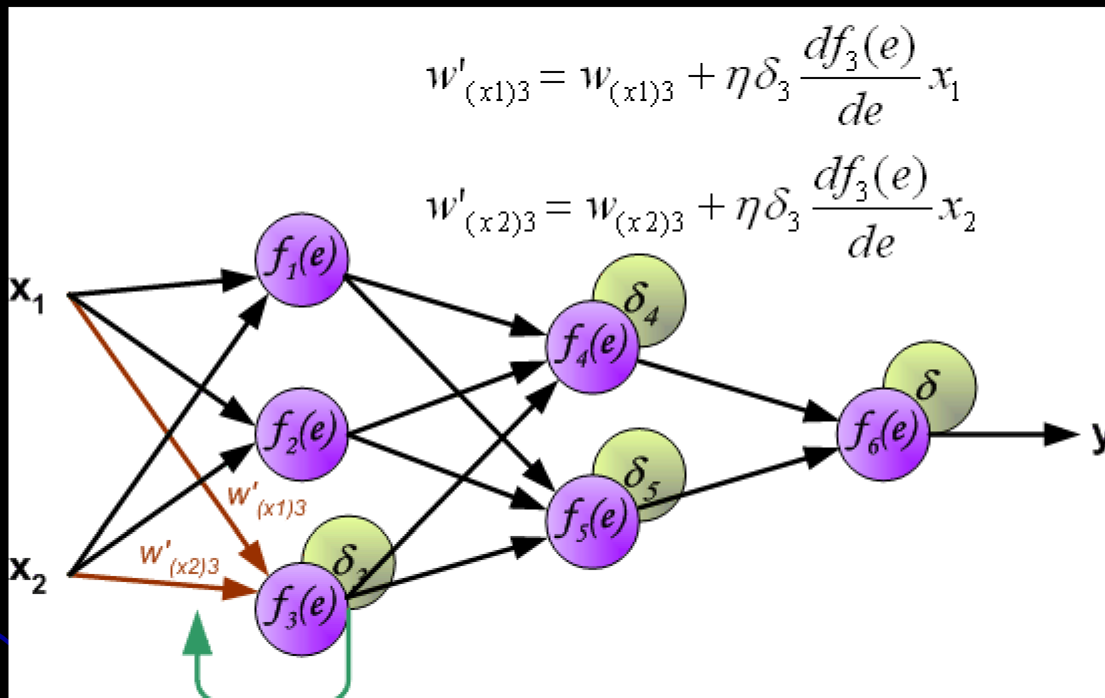
ANN learning



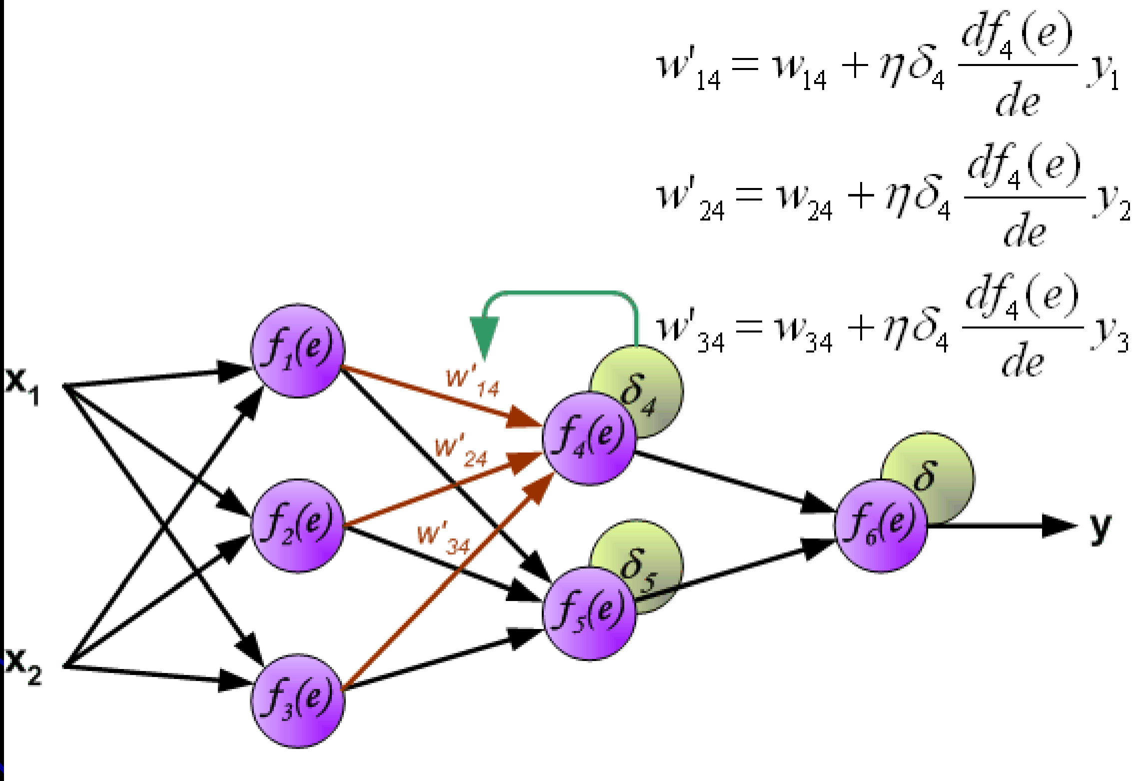
ANN learning



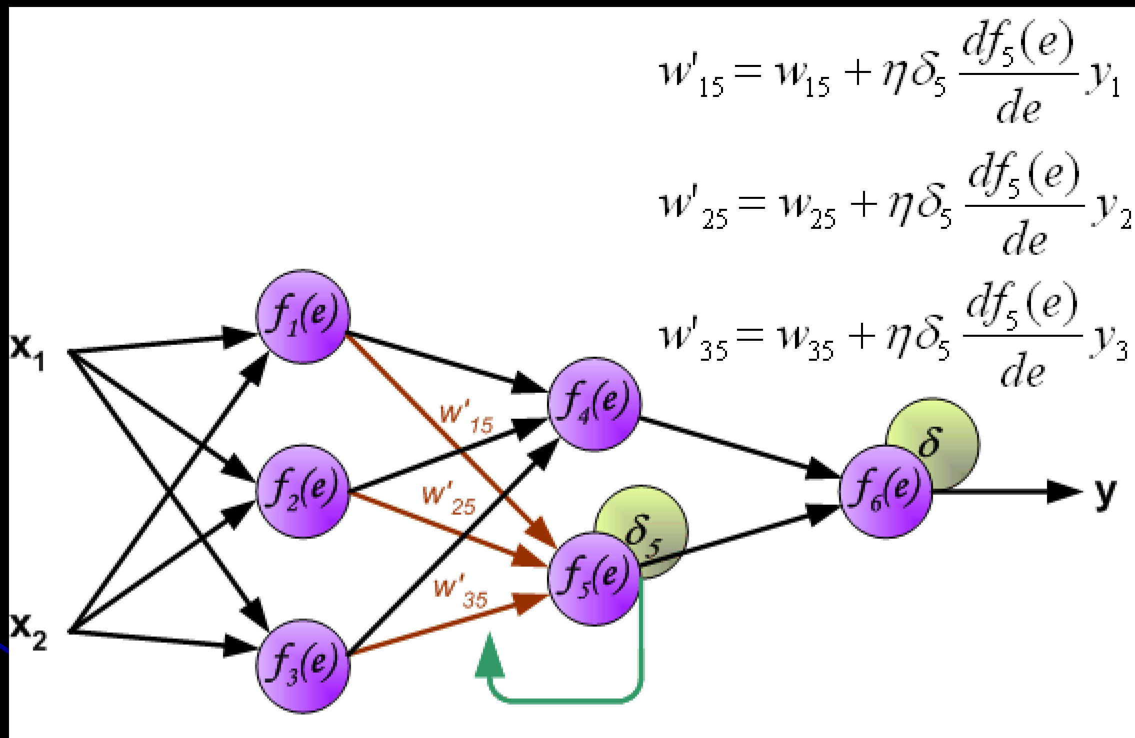
ANN learning



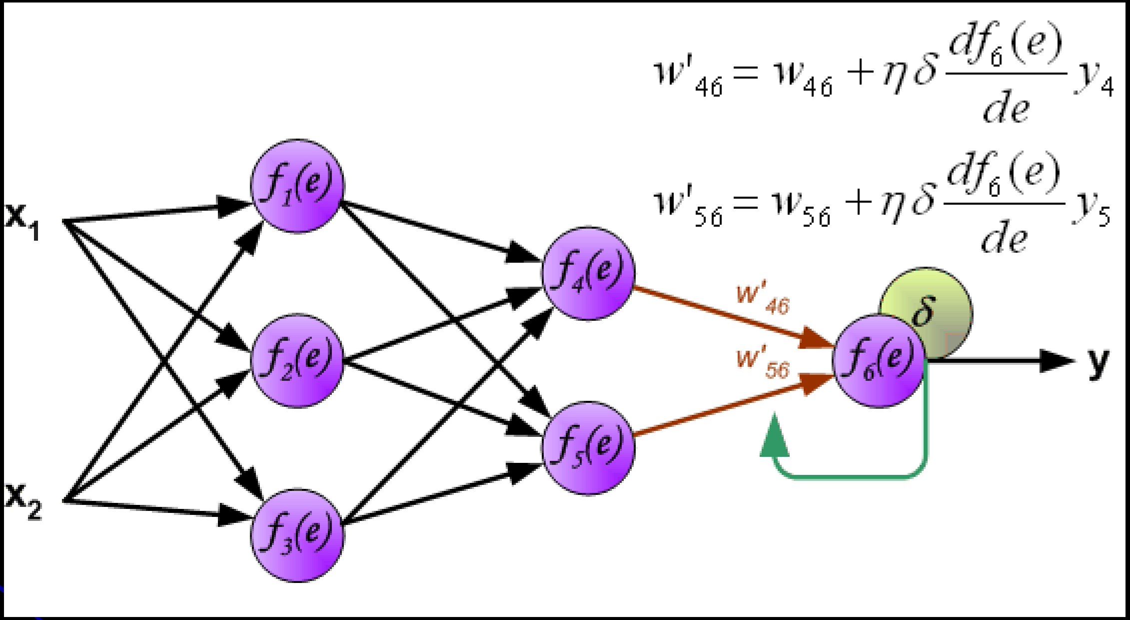
ANN learning



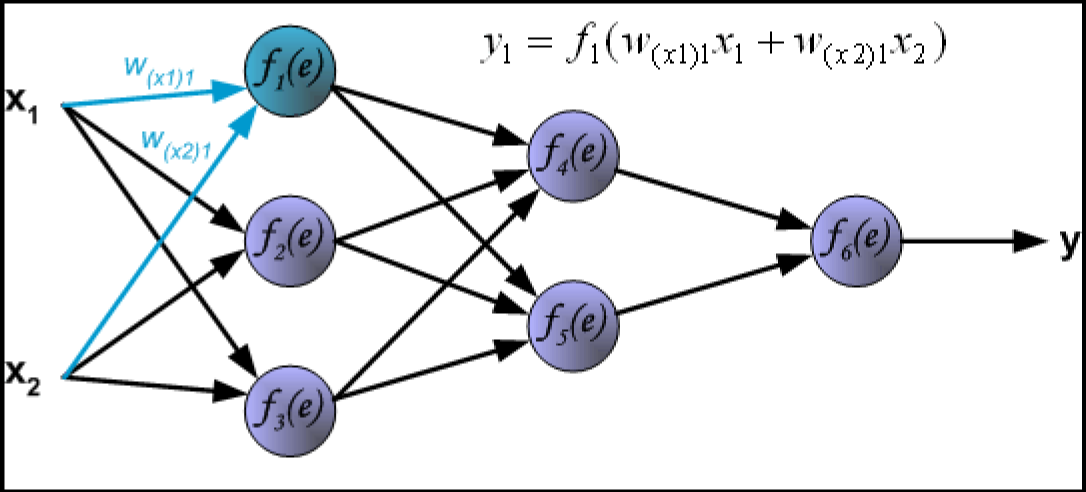
ANN learning



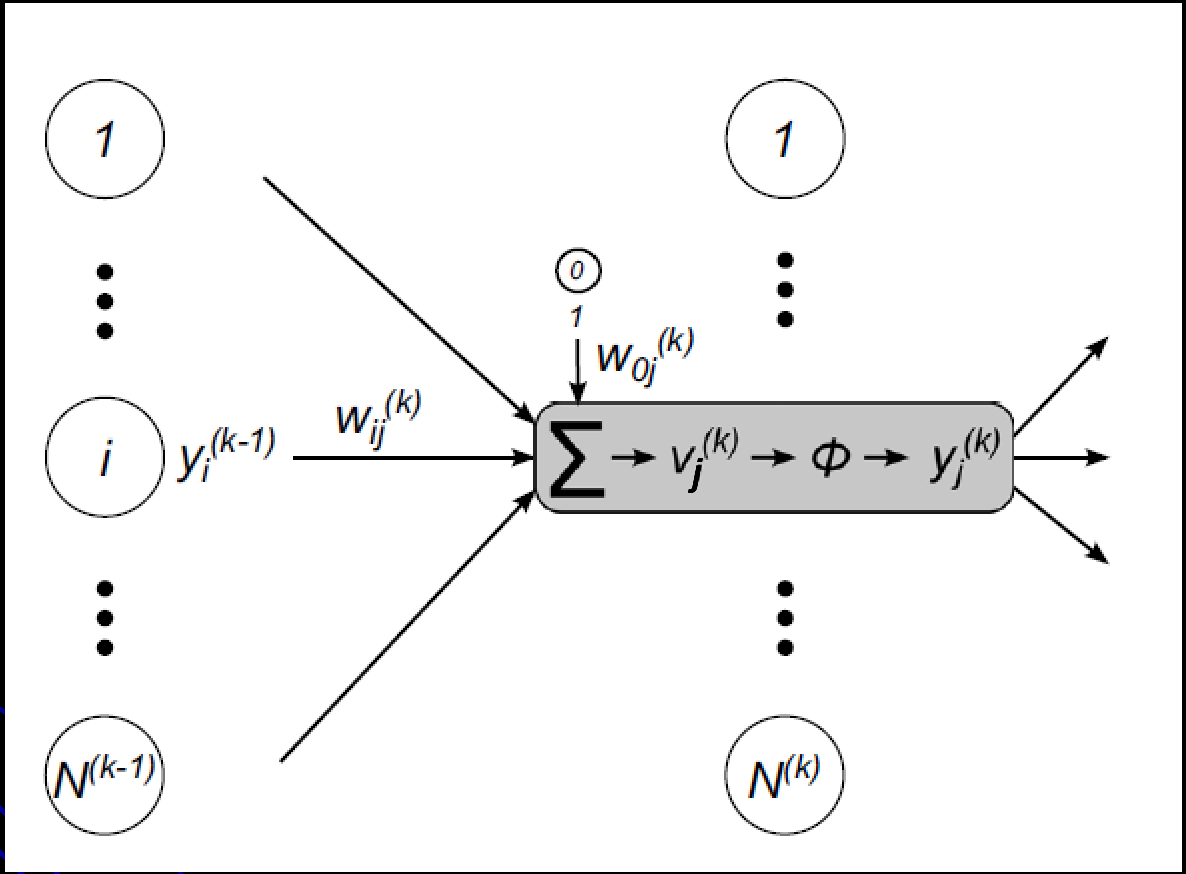
ANN learning



ANN learning



ANN learning



$$v_j^{(k)} = \sum_{i=1}^{N^{(k-1)}} w_{ij}^{(k)} y_i^{(k-1)} + w_{0j}^{(k)},$$

$$y_j^{(k)} = \phi^{(k)}\left(v_j^{(k)}\right)$$

ANN learning

$$\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W},$$

$$\Delta\mathbf{W} = -\eta \, \nabla E\big|_{\mathbf{W}}.$$

$$\Delta w_{ij}^{(k)} = -\eta \, \frac{\partial E}{\partial w_{ij}^{(k)}}.$$

$$\Delta w_{ij}^{(k)} = -\eta \, \frac{\partial E}{\partial v_j^{(k)}} \, \frac{\partial v_j^{(k)}}{\partial w_{ij}^{(k)}}$$

$$\frac{\partial v_j^{(k)}}{\partial w_{ij}^{(k)}} = y_i^{(k-1)}$$

$$\delta_j^{(k)} = -\frac{\partial E}{\partial v_j^{(k)}} = -\frac{\partial E}{\partial y_j^{(k)}} \, \frac{\partial y_j^{(k)}}{\partial v_j^{(k)}}$$

$$\Delta w_{ij}^{(k)} = \eta \, \delta_j^{(k)} \, y_i^{(k-1)}$$

Backpropagation – output layer

Mean square error (MSE)

$$E(\mathbf{W}) = \frac{1}{2}\sum_{j=1}^{M}(y_j^{(m)} - o_j)^2$$

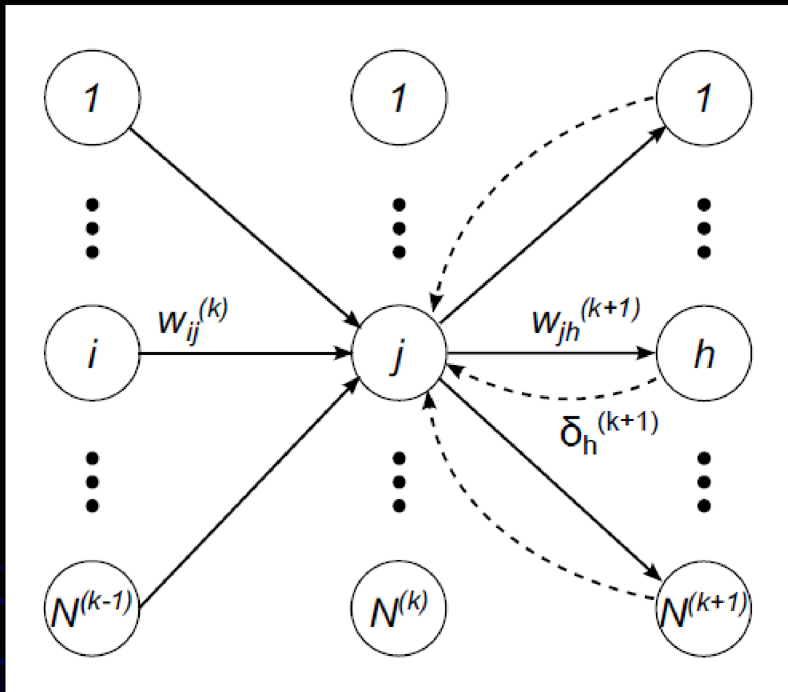
$$\delta_j^{(m)} = -\frac{\partial E}{\partial v_j^{(m)}} = -\frac{\partial E}{\partial y_j^{(m)}} \frac{\partial y_j^{(m)}}{\partial v_j^{(m)}}$$

$$\frac{\partial E}{\partial y_j^{(m)}} = y_j^{(m)} - o_j$$

$$\frac{\partial y_j^{(m)}}{\partial v_j^{(m)}} = \phi'(v_j^{(m)})$$

$$\boxed{\delta_j^{(m)} = (o_j - y_j^{(m)})\phi'(v_j^{(m)})}$$

Backpropagation – hidden layers



$$\frac{\partial E}{\partial y_j^{(k)}} = \sum_{h=1}^{N^{(k+1)}} \frac{\partial E}{\partial v_h^{(k+1)}} \frac{\partial v_h^{(k+1)}}{\partial y_j^{(k)}} =$$

$$= - \sum_{h=1}^{N^{(k+1)}} \delta_h^{(k+1)} w_{jh}^{(k+1)}$$

$$\frac{\partial y_j^{(k)}}{\partial v_j^{(k)}} = \phi'(v_j^{(k)})$$

$$\boxed{\delta_j^{(k)} = \phi'(v_j^{(k)}) \sum_{h=1}^{N^{(k+1)}} \delta_h^{(k+1)} w_{jh}^{(k+1)}}$$

ANN learning

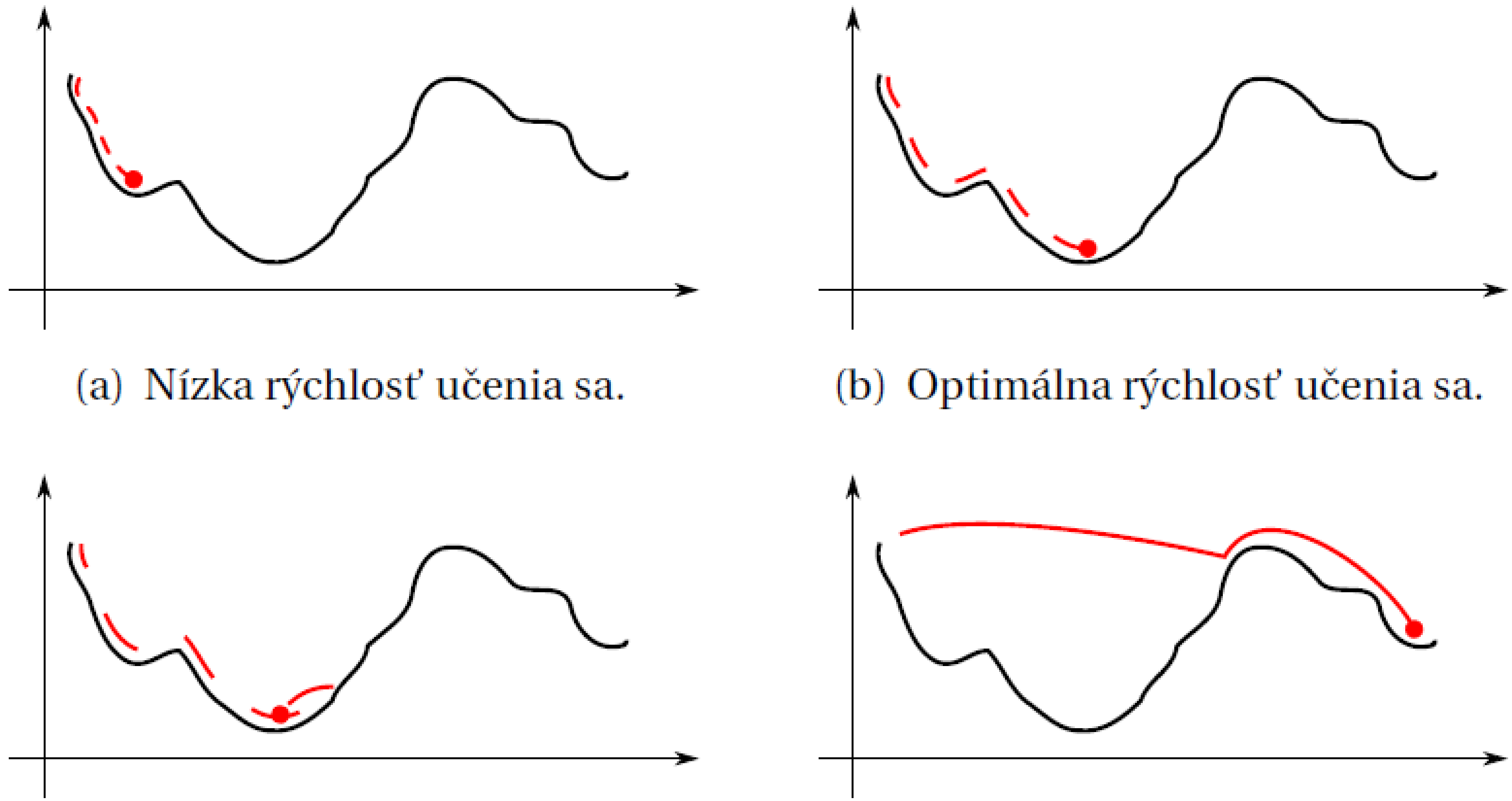
$$\delta_j^{(m)} = (o_j - y_j^{(m)})\phi'(v_j^{(m)})$$

$$\delta_j^{(k)} = \phi'(v_j^{(k)}) \sum_{h=1}^{N^{(k+1)}} \delta_h^{(k+1)} w_{jh}^{(k+1)}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W},$$

$$\Delta w_{ij}^{(k)} = \eta\, \delta_j^{(k)} y_i^{(k-1)}$$

Learning rate



(a) Nízka rýchlosť učenia sa.

(b) Optimálna rýchlosť učenia sa.

(c) Vysoká rýchlosť učenia sa.

(d) Veľmi vysoká rýchlosť učenia sa.

Learning rate

Low learning rate – very slow training, can end up in local minimum

Optimal learning rate – slow training, without oscillations, ends up in global minimum

High learning rate – faster training, possible oscillations, can end up in local minimum

Learning rate

Heuristics – variable learning rate:

When the new error is higher than the previous and the difference is bigger than a threshold, we follow the wrong path. New weights are discarded and learning rate is decreased.

We allow small increase in the error in order to be able to leave a local minimum. If the error increase is less than a threshold, we accept the new weights.

If the new error is lower than the previous, we are heading to the minimum, we can increase the learning rate.

Other learning algorithms

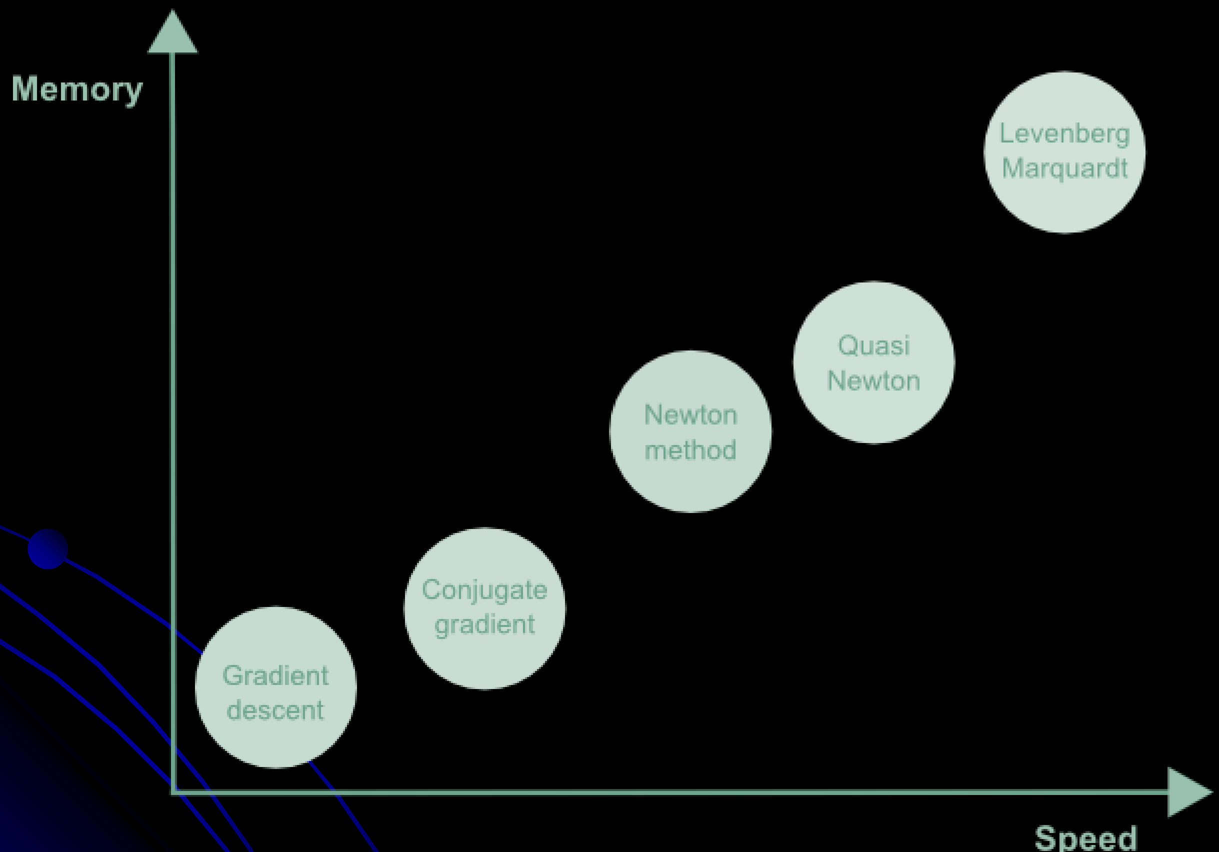
Newton's method
Uses Hessian

Quasi-Newton method
Approximation of Hessian

Conjugate gradient
new gradient and the previous search direction

Levenberg-Marquardt
Hessian approximated by Jacobian

Other learning algorithms



https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network