Data Mining

Lecture 5

Classification with Supervised Neural Networks

http://www.informatik.uni-hamburg.de/WTM/

Why Learning? Some Quotes

- “Artificial Intelligence is realised only when a computer can ‘discover’ ***for itself*** new techniques for problem solving” Fogel (1966)

- “Intelligent agents must be able to ***change*** through the course of their interactions with the world“ Luger (2002)

- “A machine or software tool would not be viewed as intelligent if it could not ***adapt to*** changes in its environment“ Callan (2003)

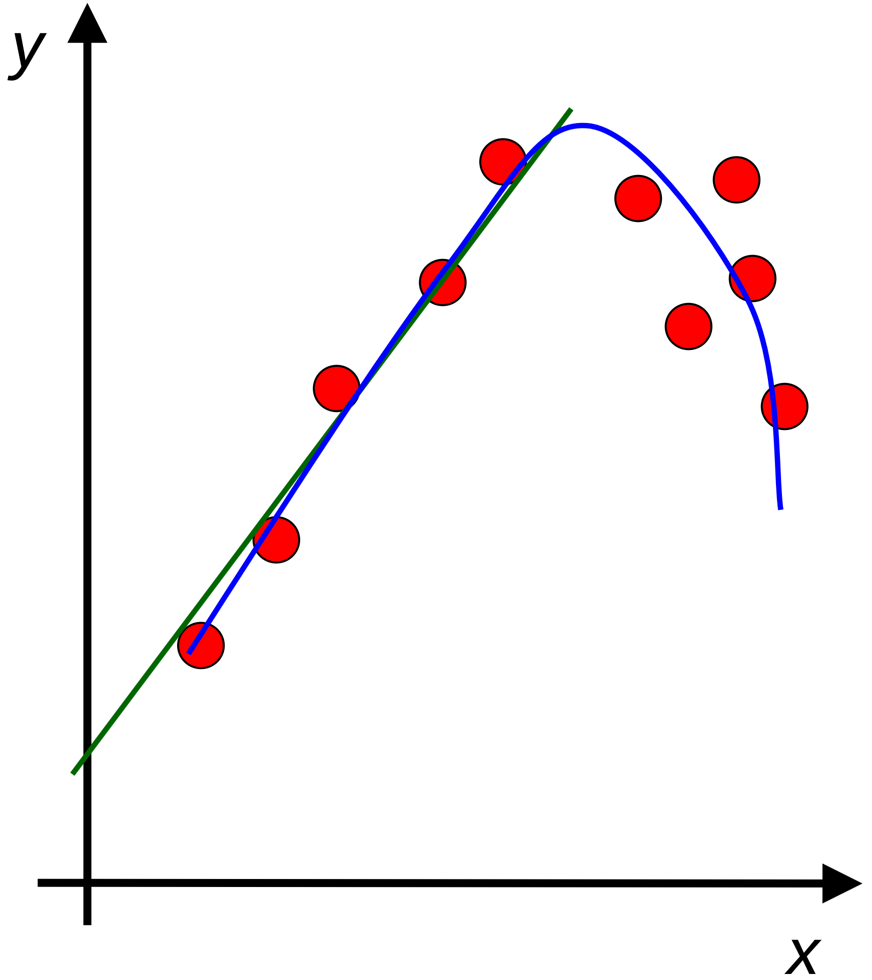
What is Neural Learning?

- Modify and improve behaviour by past experience
- How does the brain learn?
  - Strengths of synaptic connections vary

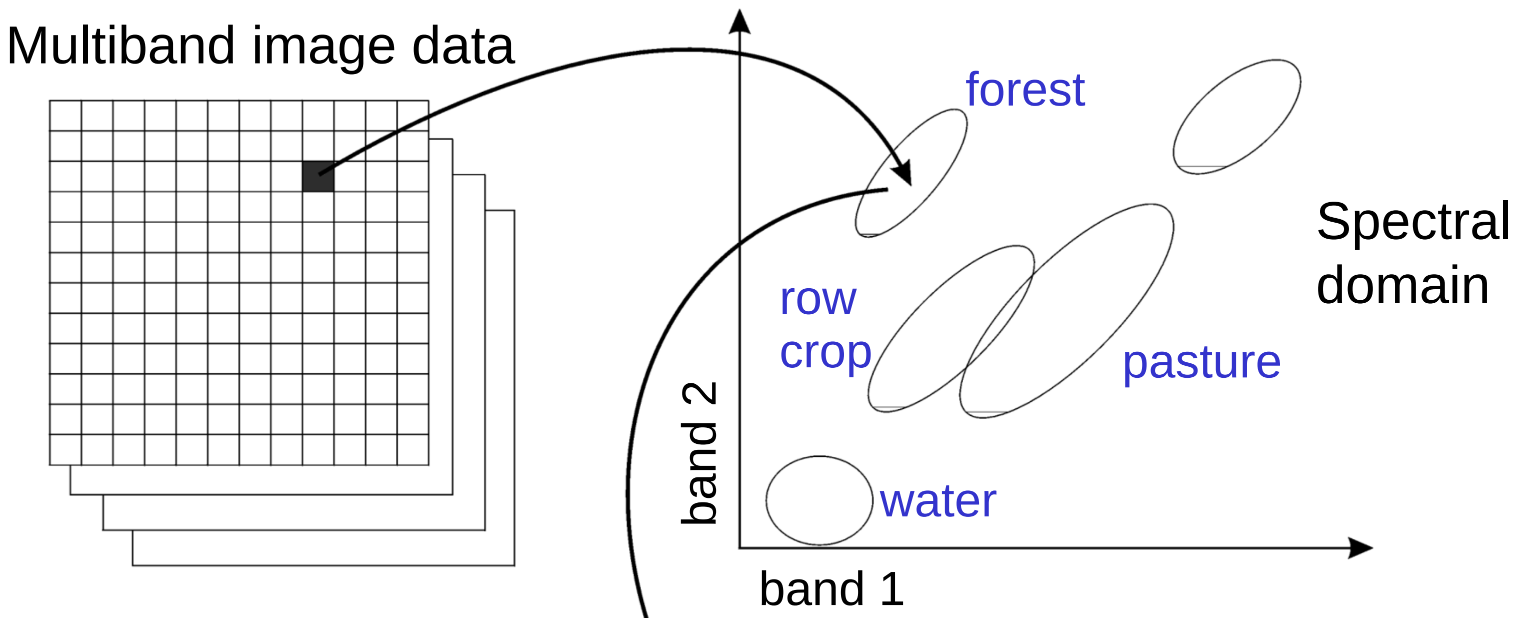
- Hebb's rule
  - If two neurons connected by a synapse fire simultaneously then the synapse strengthens
  - If two neurons connected by a synapse do not fire simultaneously then the synapse weakens
  - "fire together, wire together"

Learning Regression Problems

- Curve Fitting (with ***noise***)
- Function Approximation
- Many other functions could fit the data



Learning Classification Problems



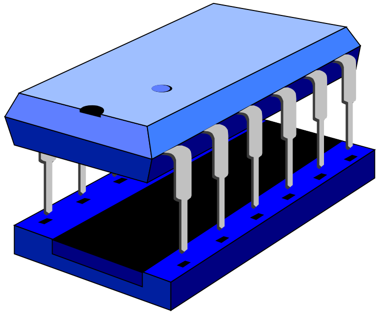
Classified image

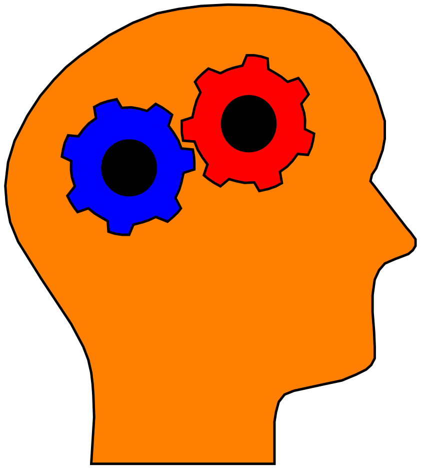
s	s	f	p	p	p	r	f	f	f	f	f
s	s	w	p	p	p	r	f	f	f	f	f
s	w	w	w	p	r	r	r	f	f	f	f
s	w	w	w	p	r	r	r	r	f	f	f
s	w	w	w	p	r	r	r	r	f	r	r
w	s	w	s	p	p	r	r	r	r	r	r
w	w	s	s	f	p	r	r	r	p	p	p
w	w	w	s	s	s	p	p	p	p	p	p
w	w	w	w	w	w	s	p	p	p	p	p
w	w	w	w	w	w	s	p	p	p	f	f
w	w	w	w	w	w	w	w	p	f	f	f
w	w	w	w	w	f	f	f	f	f	f	f

f = forest
p = pasture
w = water
r = row crop
s = sand

Computer versus Brain

- The ***von Neumann architecture*** uses a single processing unit
  - Floating Point Operations Per Second (typical today: 1 Tera FLOPS, 10^{12})
  - Absolute arithmetic precision

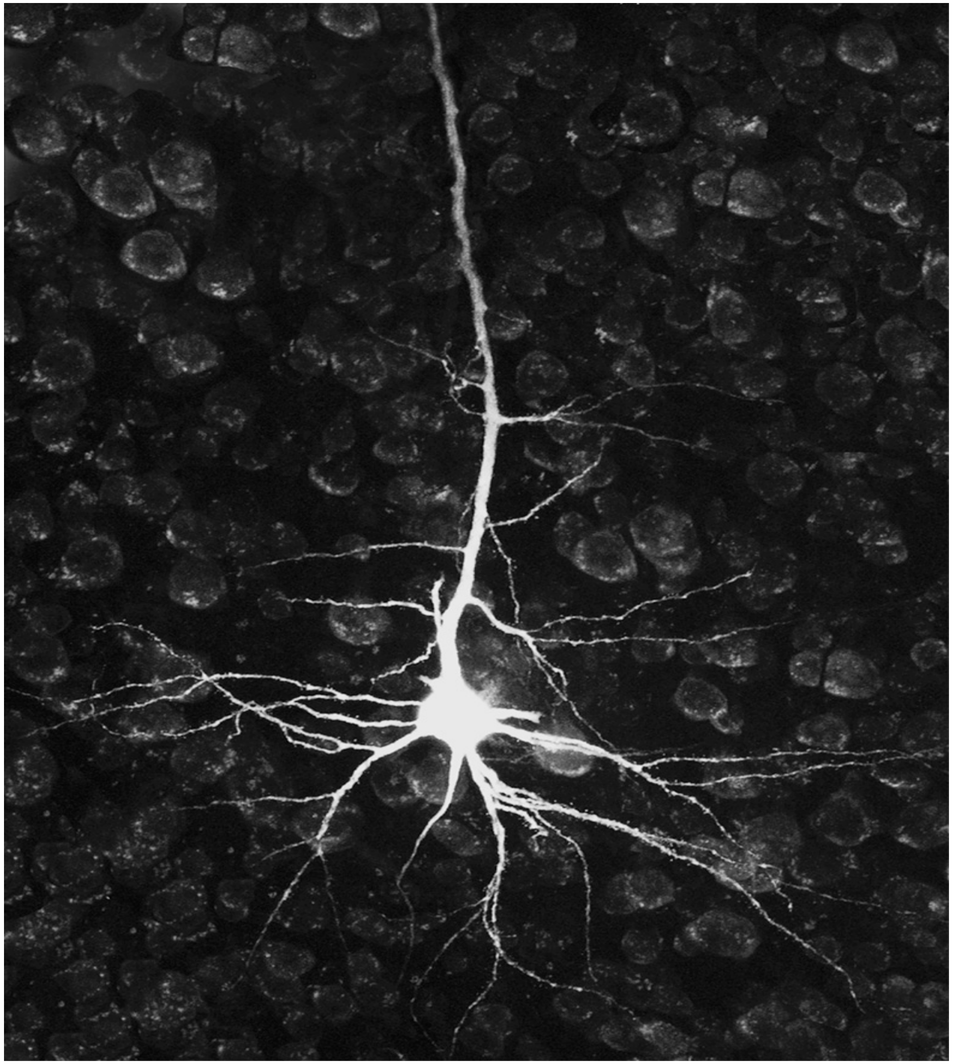




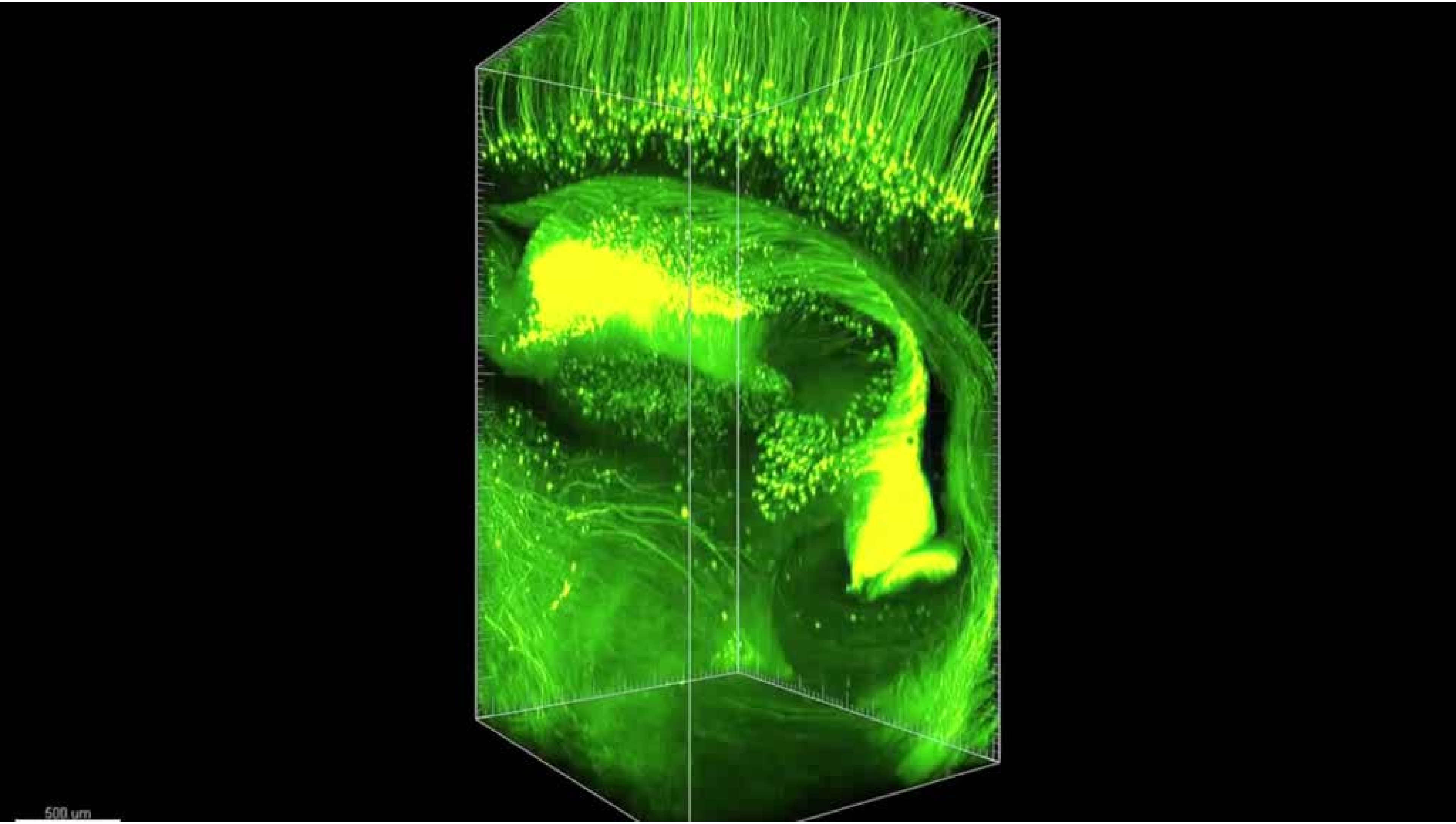
The ***brain***

- Uses many but slow, unreliable processors acting in parallel but they produce robust learned behaviour

A Real Neuron

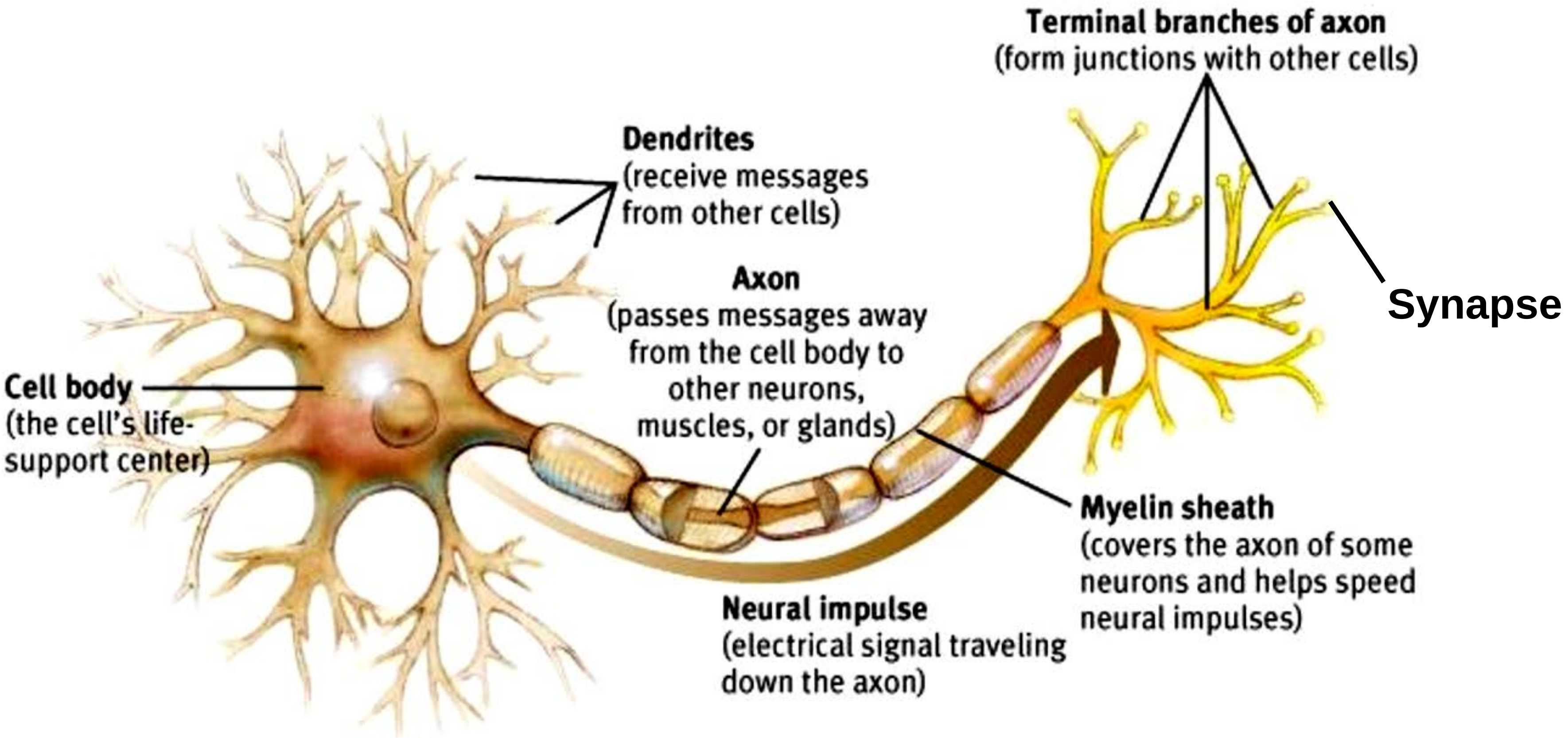


3D-View of Neurons in the Brain



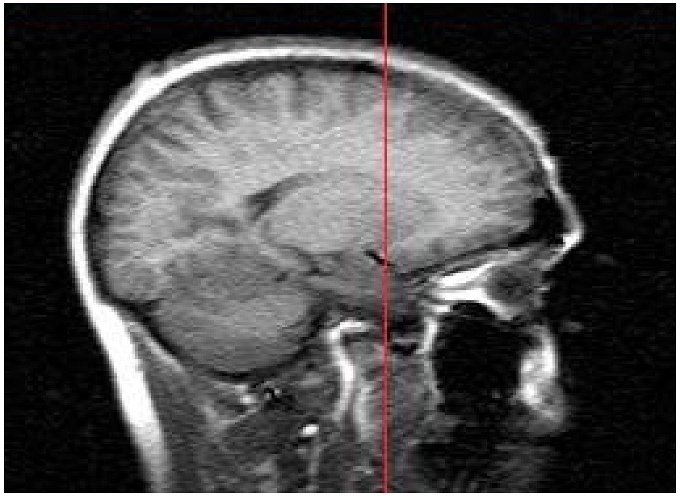
Shen, H. See-through brains clarify connections. Nature, vol. 496, pp. 151, Macmillan Publishers Limited, 11 April 2013. Video online

The Neuron


Terminal branches of axon
(form junctions with other cells)
Dendrites
(receive messages
from other cells)
Axon
(passes messages away
from the cell body to
other neurons,
muscles, or glands)
Synapse
Cell body
(the cell's life-
support center)
Myelin sheath
(covers the axon of some
neurons and helps speed
neural impulses)
Neural impulse
(electrical signal traveling
down the axon)

A Neuron`s Firing

Noninvasive Inspections of the Brain



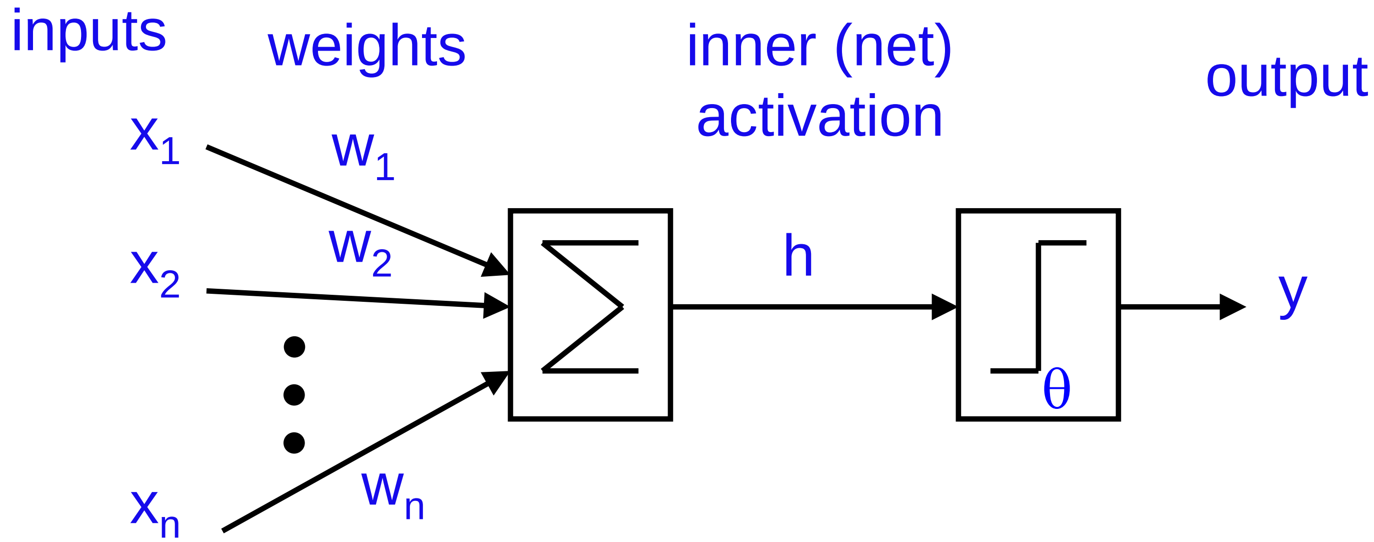
Parallel Processing in the Brain

The human brain:

- Weight on average 1.4kg
- Contains around ***10^{11} neurons***
  - Many ***different types***
  - In computational terms, 10^{11} simple processors
    - Each takes a few milliseconds to do a computation
    - But the whole brain is very fast
- Has about ***10^{14} synapses***
- ***Highly connected***
  - Things done massively in parallel
  - Robust to faults

Neuron Activity

Perceptron Neurons



Greatly simplified biological neurons

- Sum the inputs x_j each being weighted with weight w_j
- The total sum is h
- If h is more than some threshold θ
  - then neuron fires: $y = 1$,
  - else not: $y = 0$ (sometimes also used: $y = -1$)

Perceptron Neurons

n input neurons

$$h = \sum_{j=1}^{n} x_j w_j$$

$$y = \begin{cases} 1 & h \geq \theta \\ 0 & h < \theta \end{cases}$$

for some threshold θ

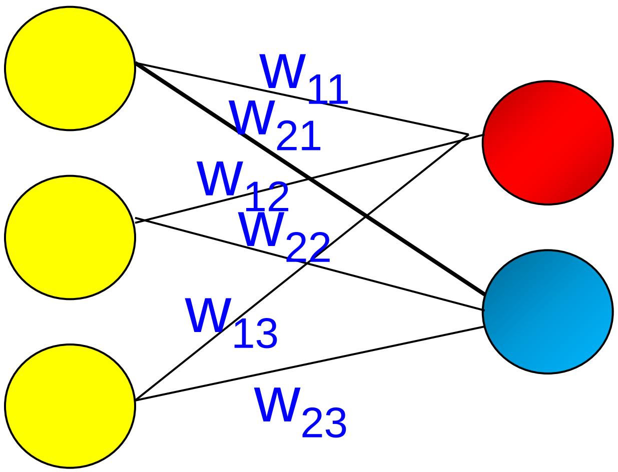
How biologically (un)realistic?

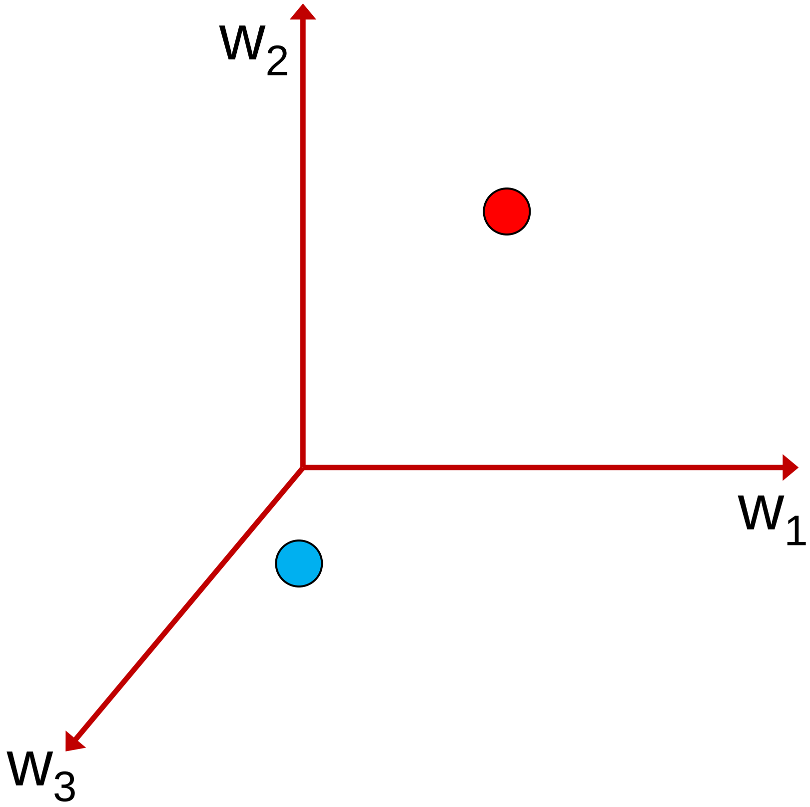
- The weight w_j can be positive or negative
- A unit can become inhibitory or excitatory, or both
- Use only a linear sum of inputs
- Use a simple output instead of a pulse (spike train)
- No refractory period

Some Terminology

Term	*Typical Symbol*	*Alternate Term(s)*
Input vector	$\boldsymbol{x}$	*input activation*
Weights	w_{ij}	*synaptic weights (from j to i)*
Inner activation	h	*net activation*
Activation function	g	*transfer function; threshold function*
Output	y	*(outer) activation; prediction*
Target	t	*teacher value*
Error	E	cost

Weight Space: Represent a Unit with its Incoming Weights

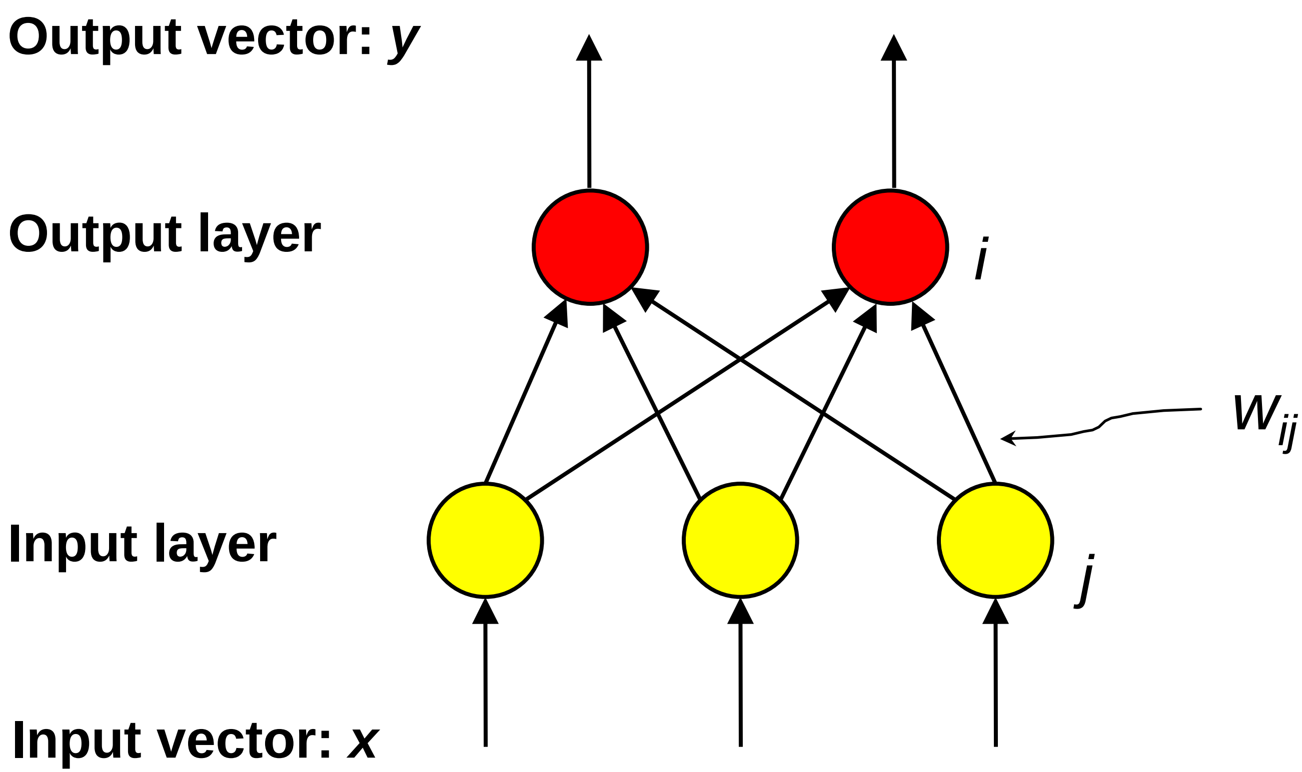




Neural Networks

- Started by psychologists and neurobiologists as computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a ***weight*** associated with it
- During supervised learning, the ***network learns by adjusting the weights*** so as to be able to predict the correct class label of the input tuples
- Also referred to as ***connectionist learning*** due to the connections between units

Perceptron Network



Updating the Weights

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

- We want to change the values of the weights
- Aim: ***minimize the error*** at the output
- Let error $E = t - y$. We want E to be 0
- Use:

Learning rate

Input

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

Change of weight

Error

Perceptron Algorithm

- Initialisation: set all weights to small positive and negative random numbers
- For #iterations
  - Chose a new data point (***x***, ***t***)
  - Compute the output activation y_i of each neuron i

$$h_i = \sum_{j=1}^{n} w_{ij} x_j \qquad y_i = \begin{cases} 1 & h_i \geq \theta \\ 0 & h_i < \theta \end{cases}$$

  - Update each of the weights according to

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

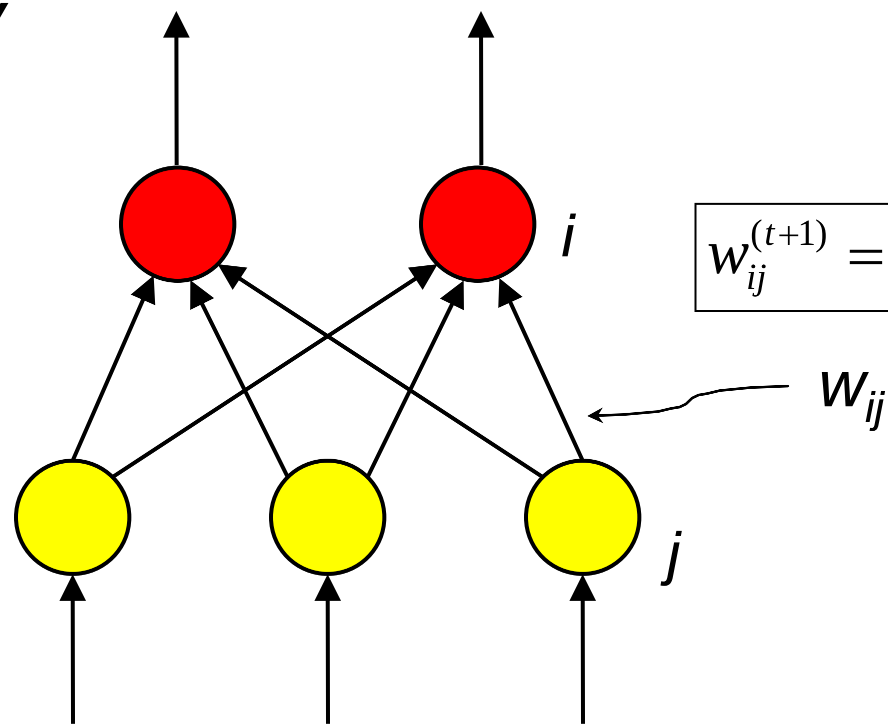
Perceptron Network

Output vector: *y*

Output layer

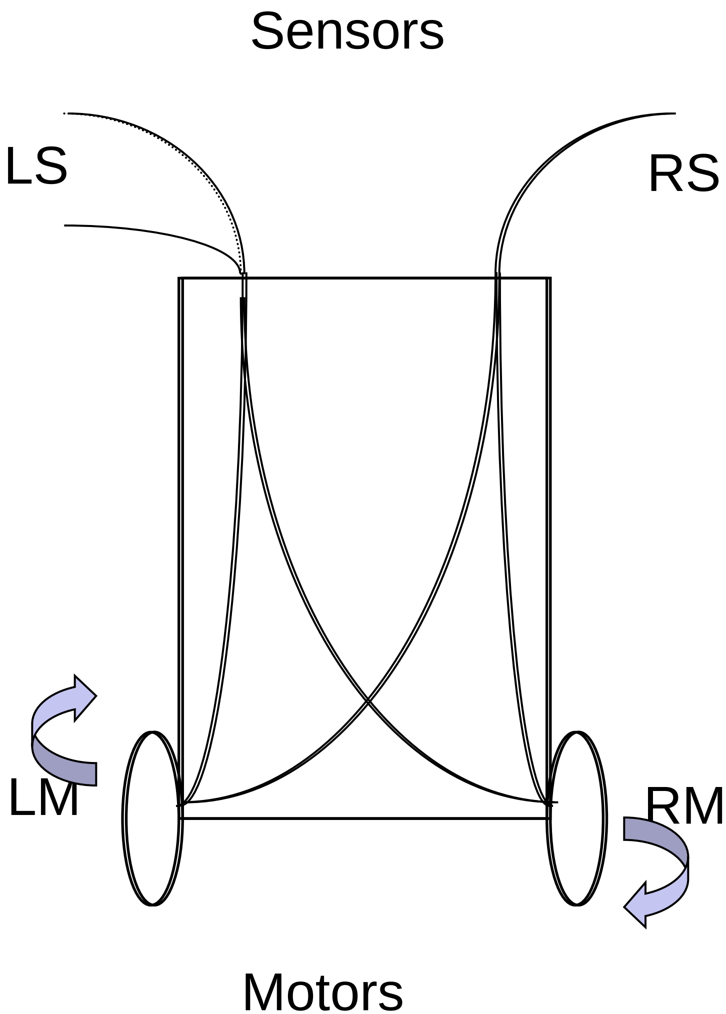
Input layer

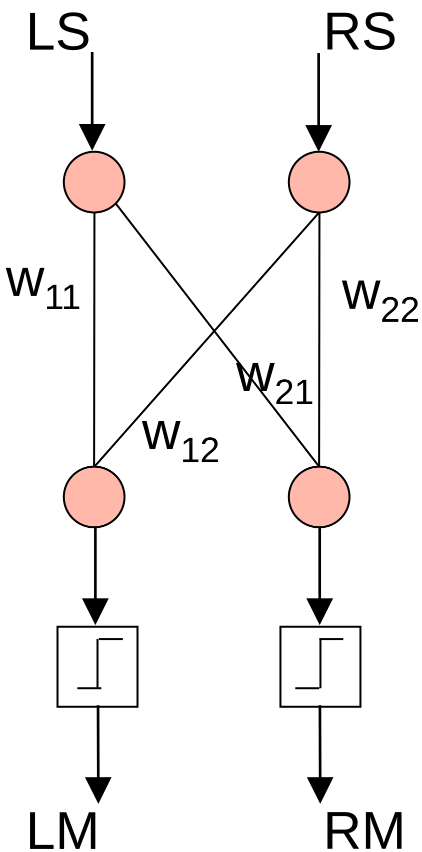
Input vector: *x*



$$\boxed{w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta(t_i - y_i)x_j}$$

Obstacle Avoidance with the Perceptron





$\eta = 0.3$

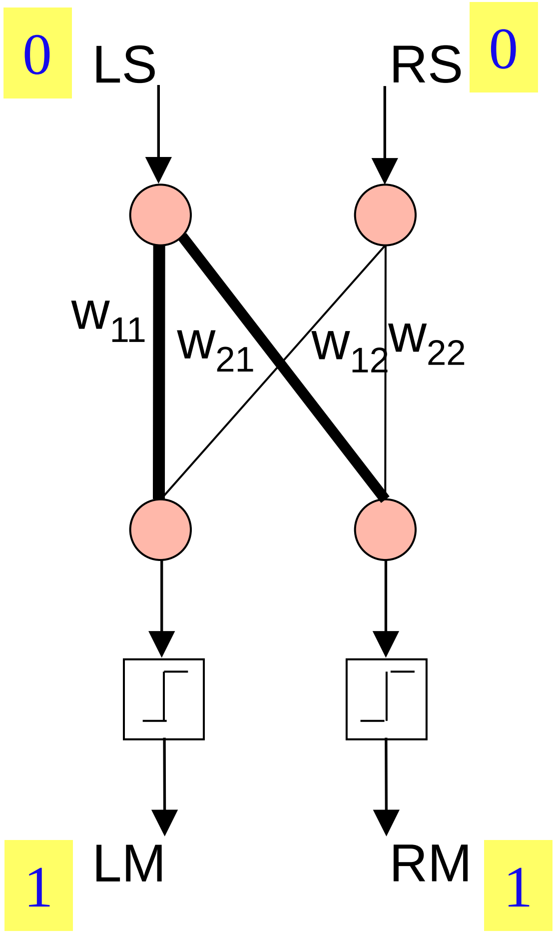
$\theta = -0.01$

$w_{ij} = 0$

Obstacle Avoidance with the Perceptron: Behaviour we want

LS	RS	LM	RM
0	**0**	**1**	**1**
0	1	-1	1
1	0	1	-1
1	1	X	X

Obstacle Avoidance with the Perceptron



Assume initial weights are 0
No update if target = actual computed

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$w_{11} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

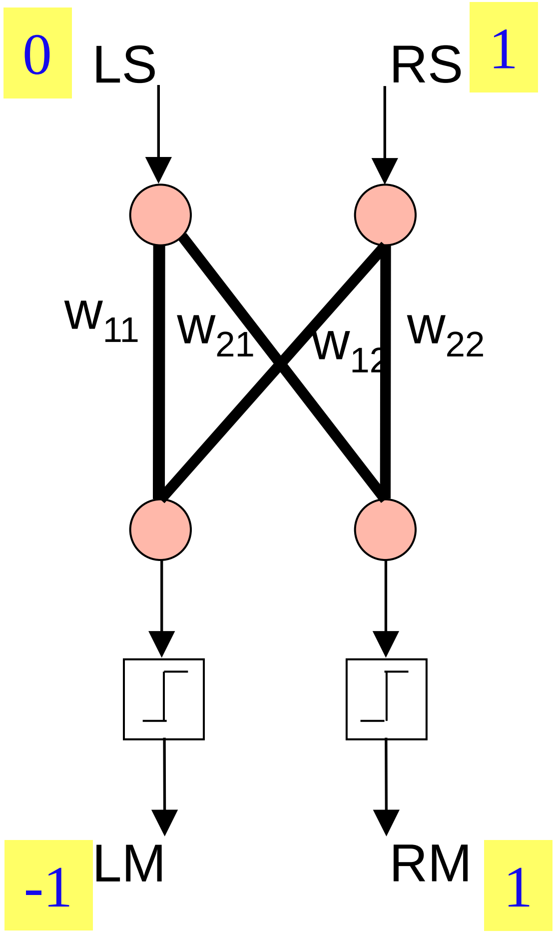
$$w_{21} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

And the same for w_{12}, w_{22}

Obstacle Avoidance with the Perceptron

LS	RS	LM	RM
0	0	1	1
0	**1**	**-1**	**1**
1	0	1	-1
1	1	X	X

Obstacle Avoidance with the Perceptron



No update if input = 0

$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$w_{11} = 0 + 0.3 \cdot (-1 - 1) \cdot 0 = 0$$

$$w_{21} = 0 + 0.3 \cdot (1 - 1) \cdot 0 = 0$$

$$w_{12} = 0 + 0.3 \cdot (-1 - 1) \cdot 1 = -0.6$$

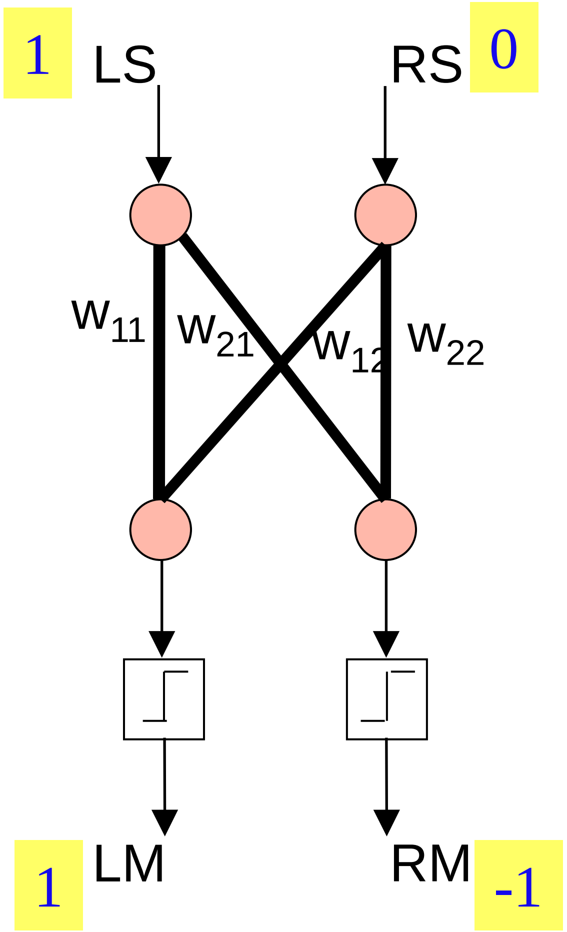
$$w_{22} = 0 + 0.3 \cdot (1 - 1) \cdot 1 = 0$$

w_{12}: the robot turns left by reversing the left motor

Obstacle Avoidance with the Perceptron

LS	RS	LM	RM
0	0	1	1
0	1	-1	1
1	**0**	**1**	**-1**
1	1	X	X

Obstacle Avoidance with the Perceptron



$$\Delta w_{ij} = \eta \cdot (t_i - y_i) \cdot x_j$$

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

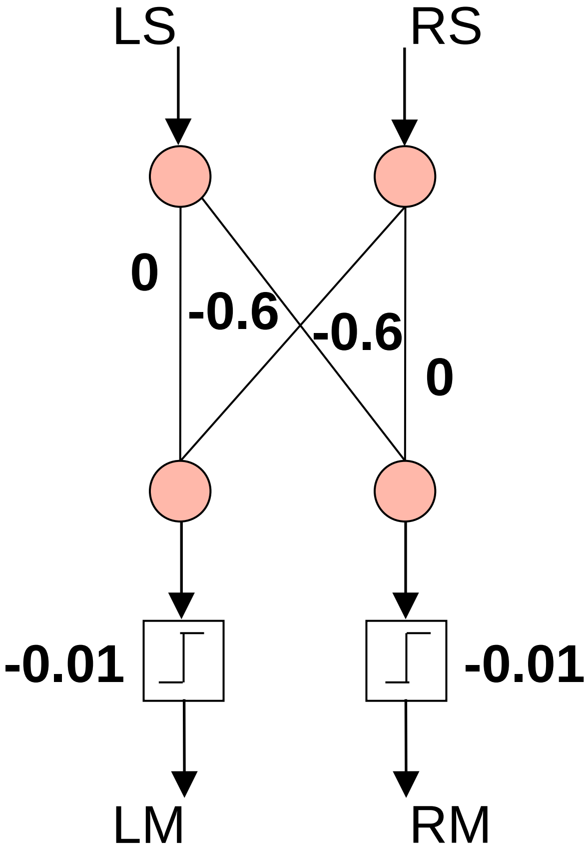
$$w_{11} = 0 + 0.3 \cdot (1 - 1) \cdot 1 = 0$$

$$w_{21} = 0 + 0.3 \cdot (-1 - 1) \cdot 1 = -0.6$$

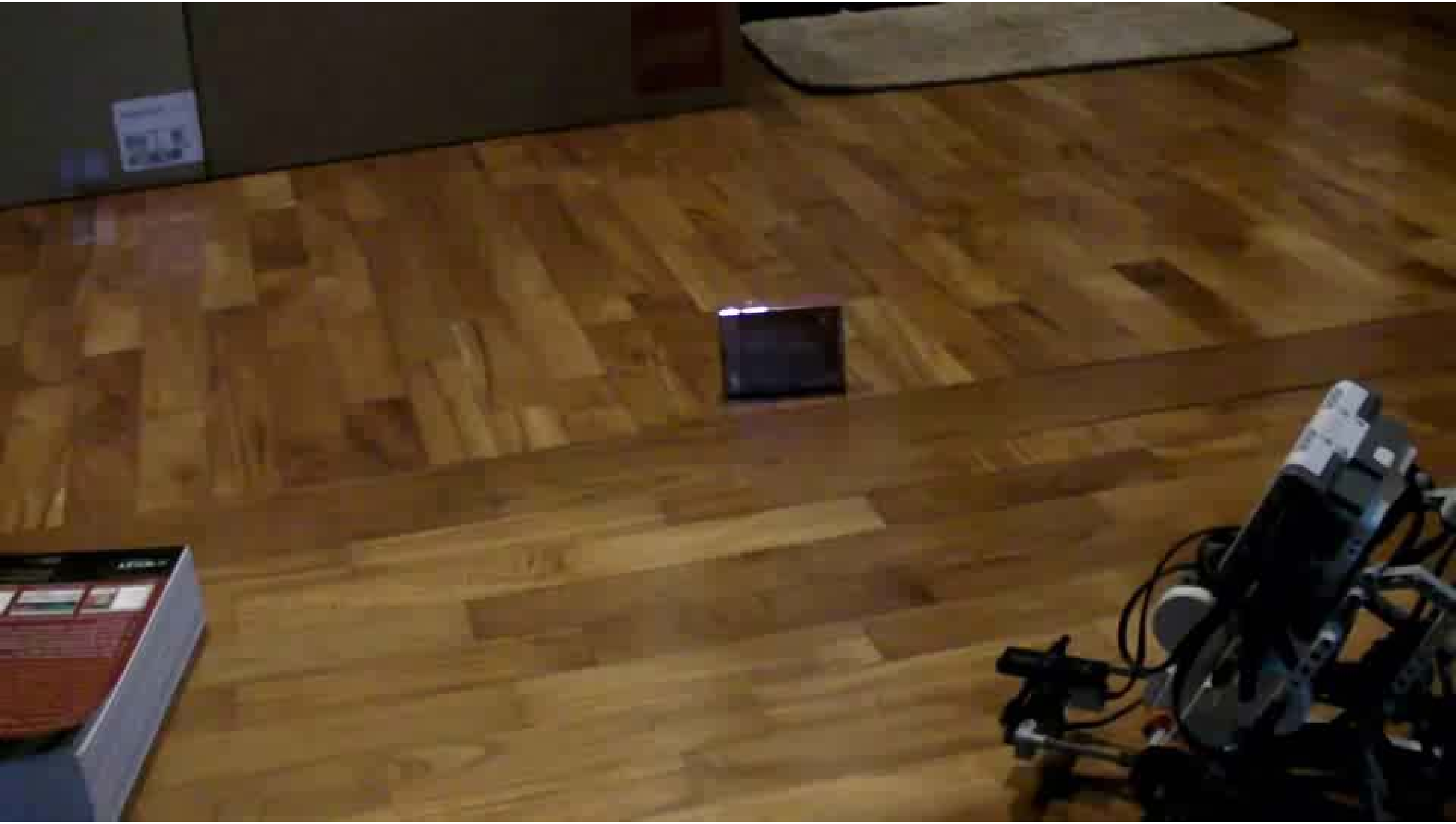
$$w_{12} = -0.6 + 0.3 \cdot (1 - 1) \cdot 0 = -0.6$$

$$w_{22} = 0 + 0.3 \cdot (-1 - 1) \cdot 0 = 0$$

Obstacle Avoidance with the Perceptron



Obstacle Avoidance with a Mindstorm Vehicle



Linear Separability

- Outputs are: $y_i = \text{sign}\left(\sum_{j=1}^{n} w_{ij} x_j\right)$
  - Positive output +1 if:

    $$w_i \cdot x \geq 0$$

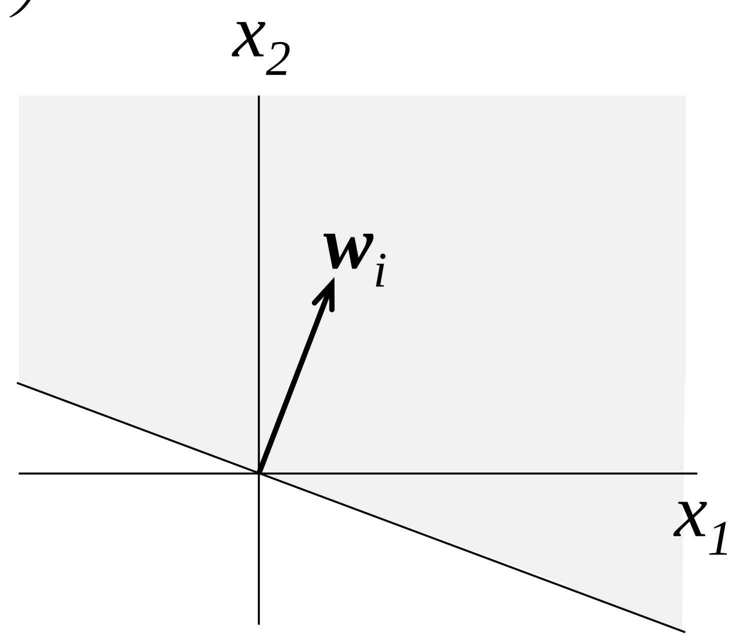
  - Negative output -1 (or 0) if:

    $$w_i \cdot x < 0$$

    weight vector

    dot product

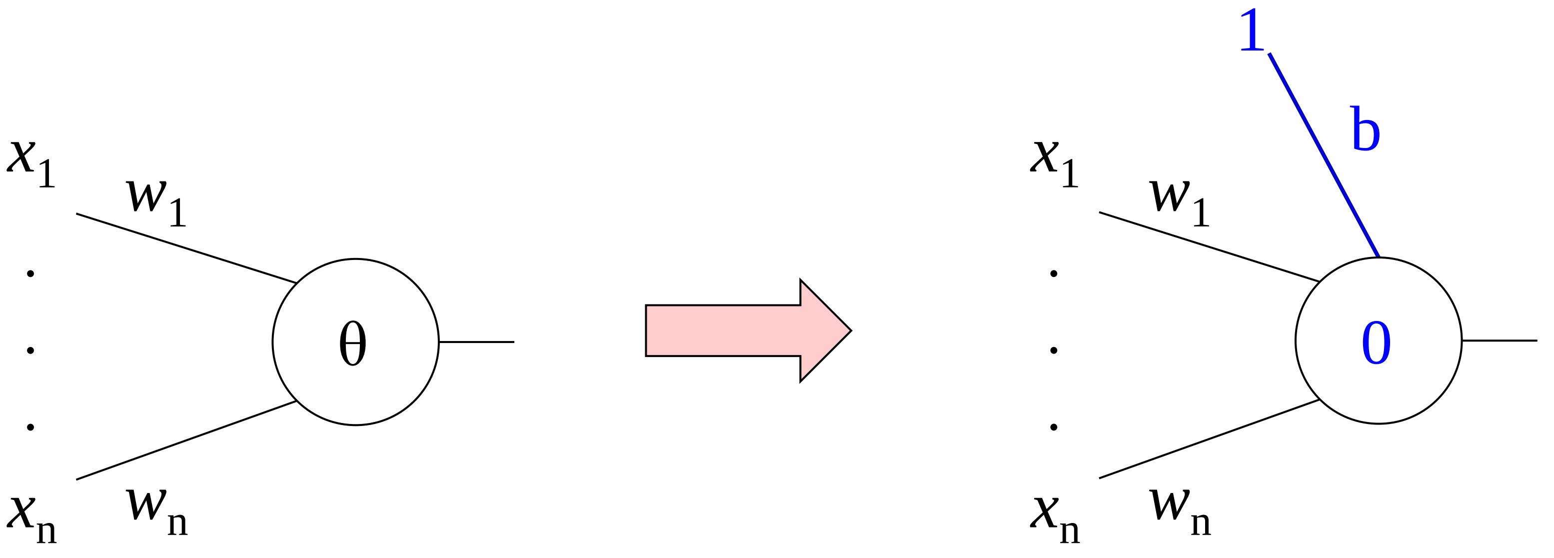
    input vector



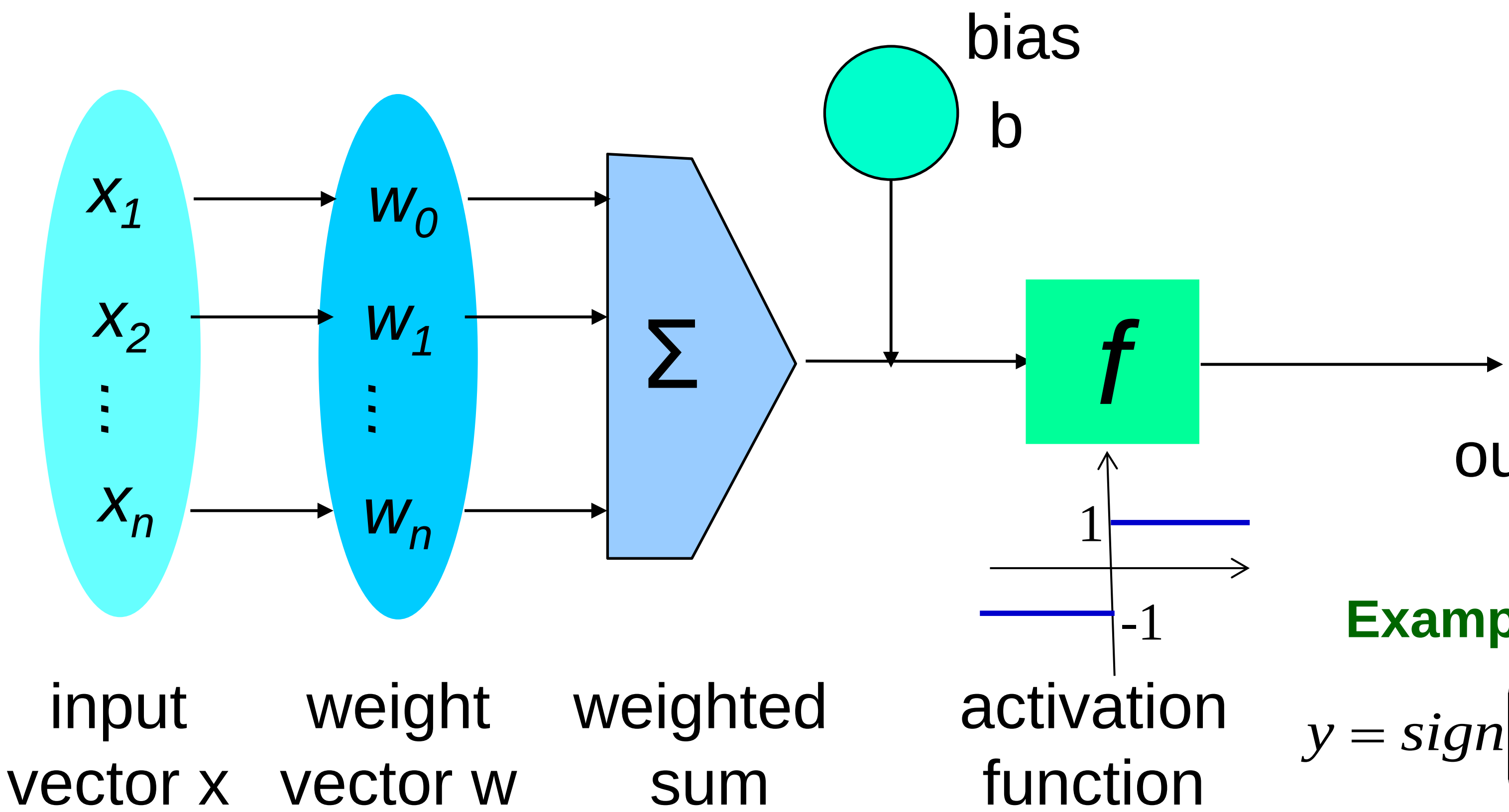
The region in input space where $\mathbf{x}$ yield positive output y is a half-plane.

Bias

- An extra input – increases or lowers the net input (depending on its sign)
- Can be regarded as a weight connected to a ***constant of 1***
  - Then bias learning is similar to weight learning
  - Can ***convert*** a threshold into an additional weight.



Perceptron with Bias

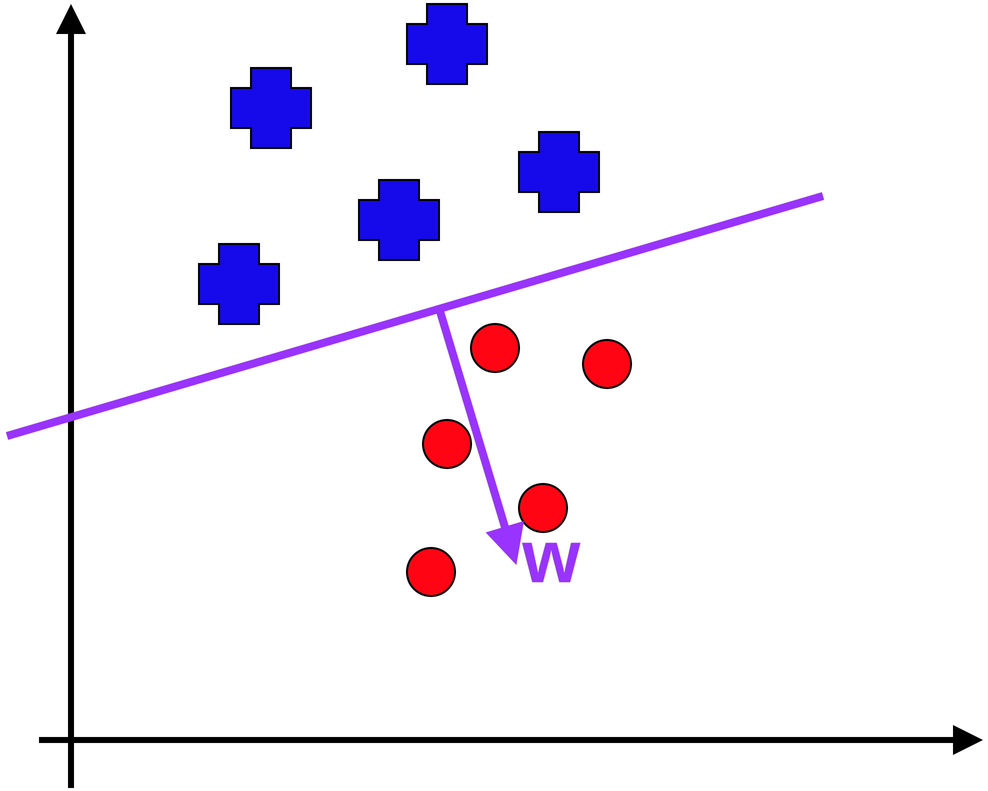


Example:

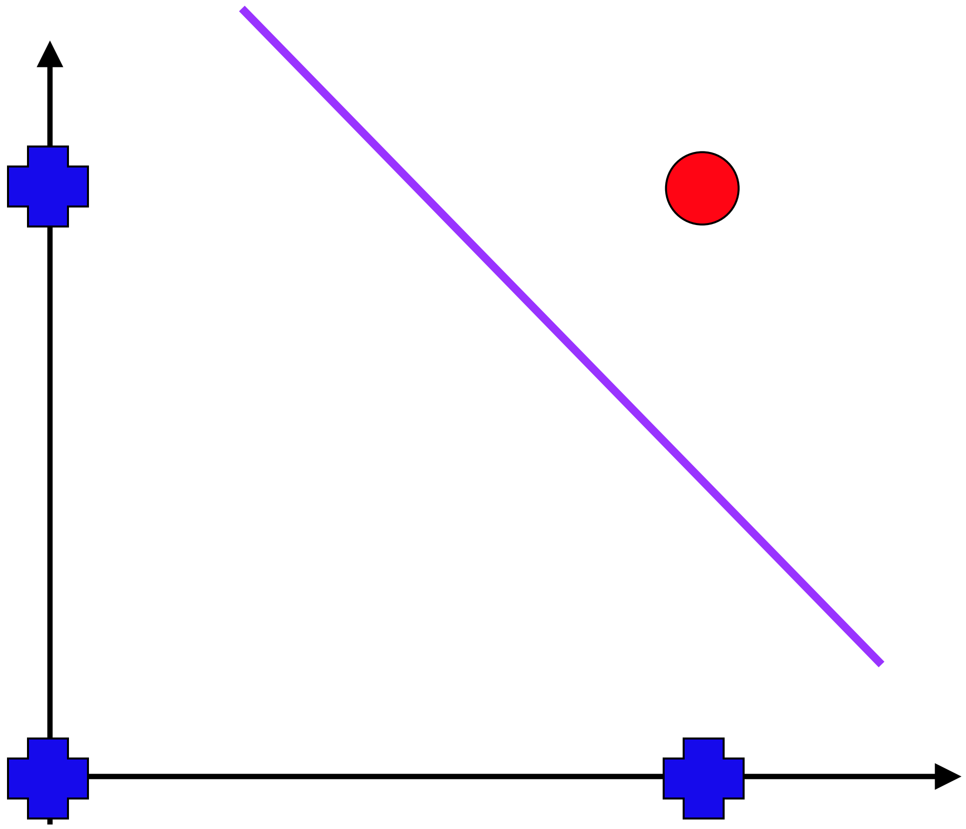
$$y = sign\left(b + \sum_{j=1}^{n} w_j x_j\right)$$

- The *n*-dimensional input vector ***x*** together with bias *b* is mapped into variable *y* by means of the scalar product and a nonlinear function mapping
- negative *bias* ~ (positive) *threshold*

Linear Separability



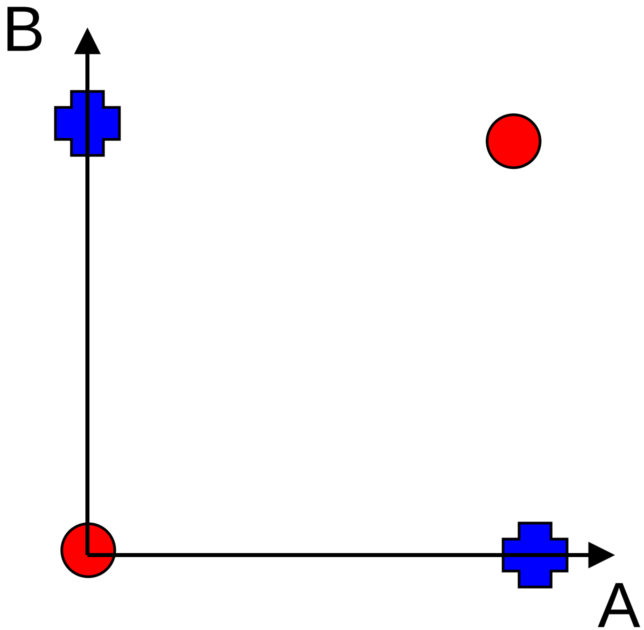
Linear Separability



The Binary
AND Function

Limitations of the Perceptron

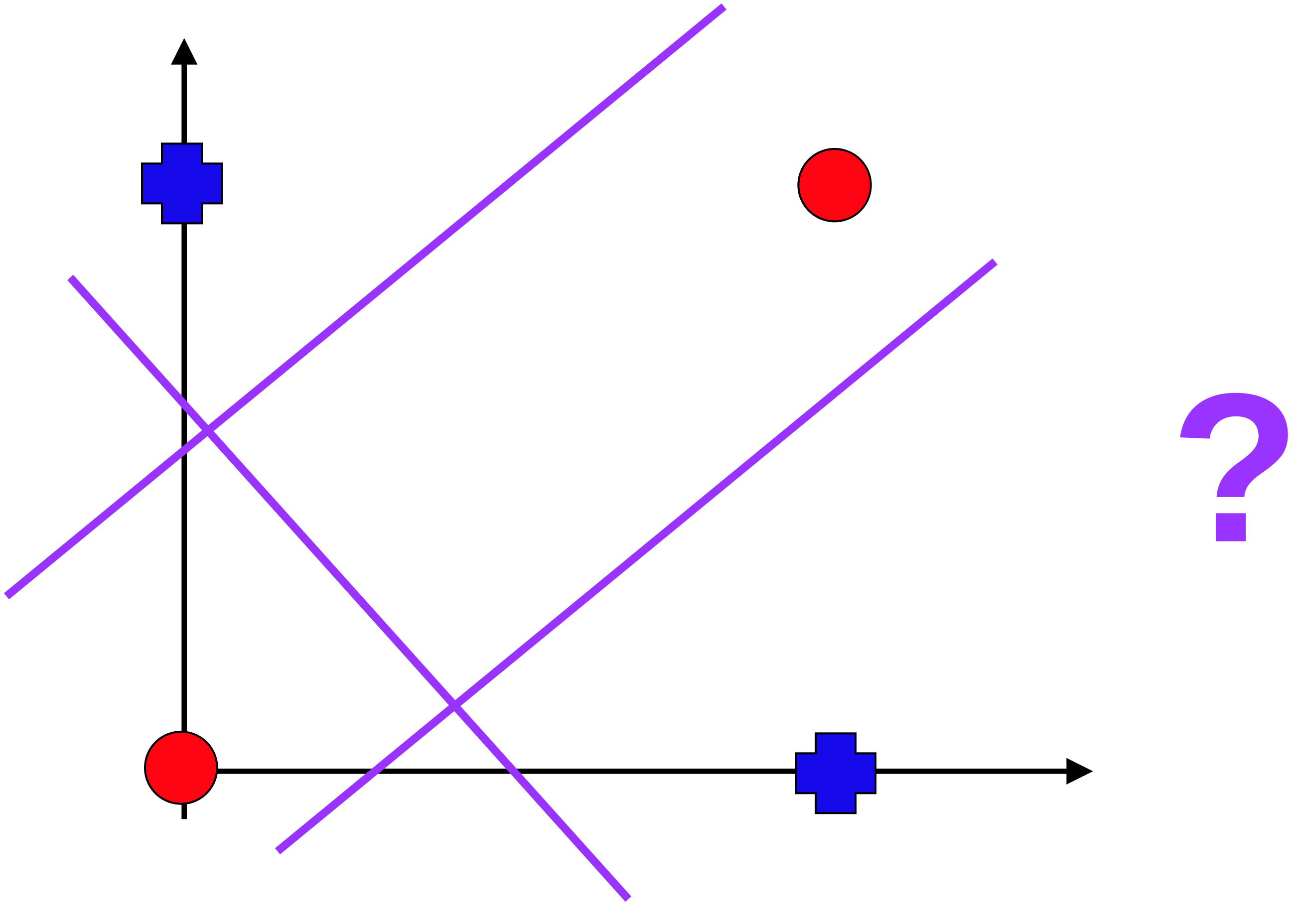
Linear Separability?



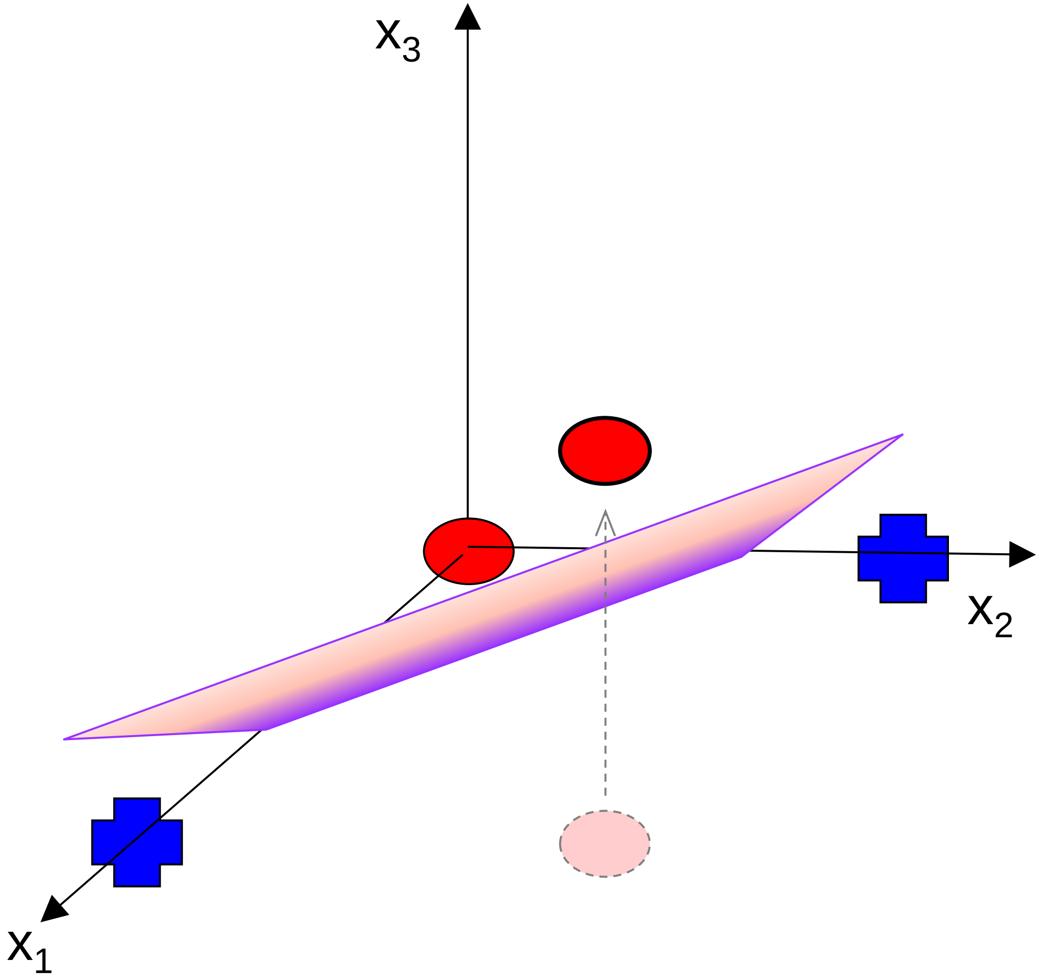
Exclusive Or (XOR) function

A	**B**	**Out**
0	0	0
0	1	1
1	0	1
1	1	0

Limitations of the Perceptron



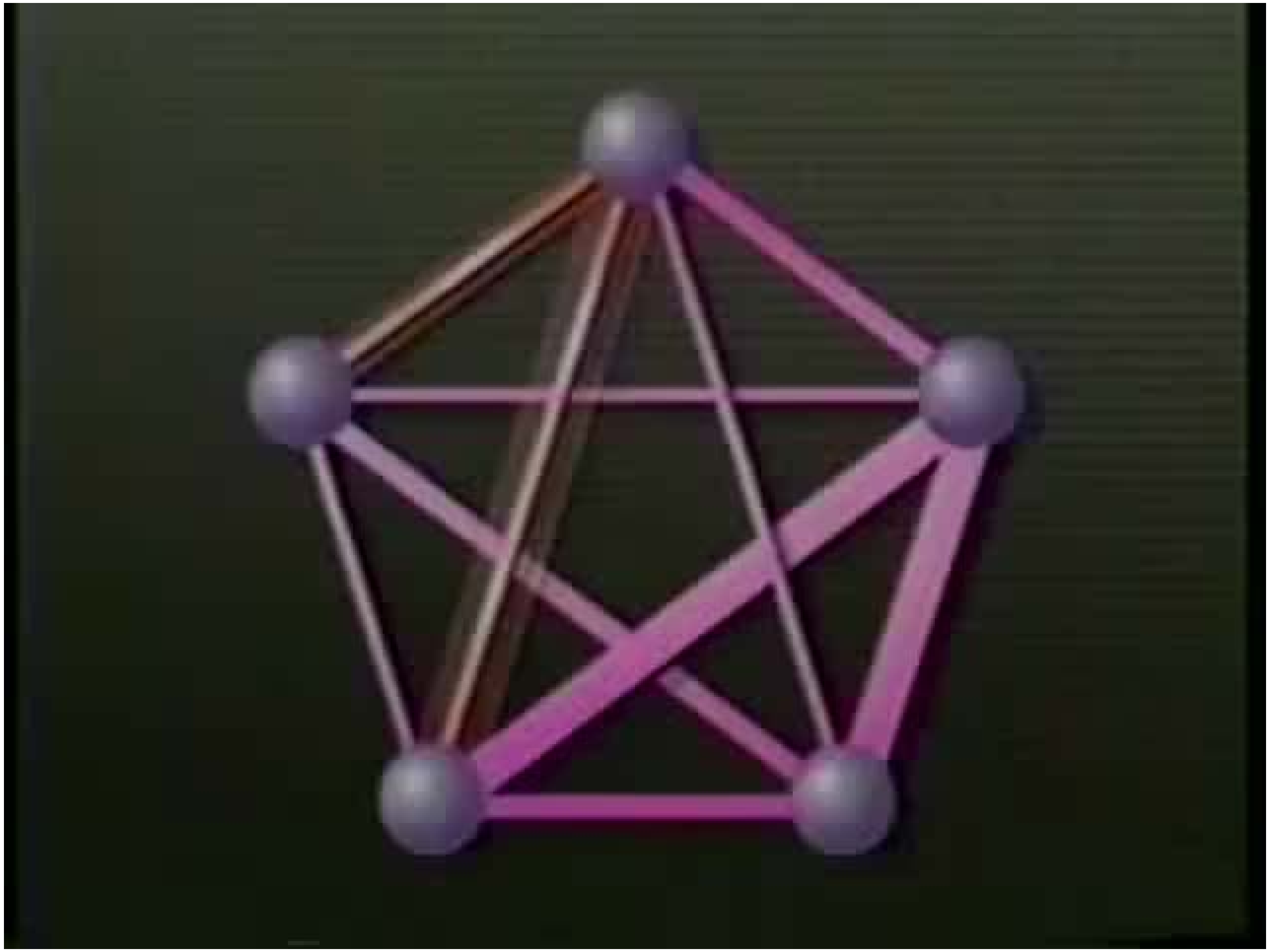
Limitations of the Perceptron



One way around the problem is to use a more complex input set (e.g., three-dimensional: $x_3 = x_1 \cdot x_2$).

Another is to make the network more complex.

Perceptrons – Early Successes(?)



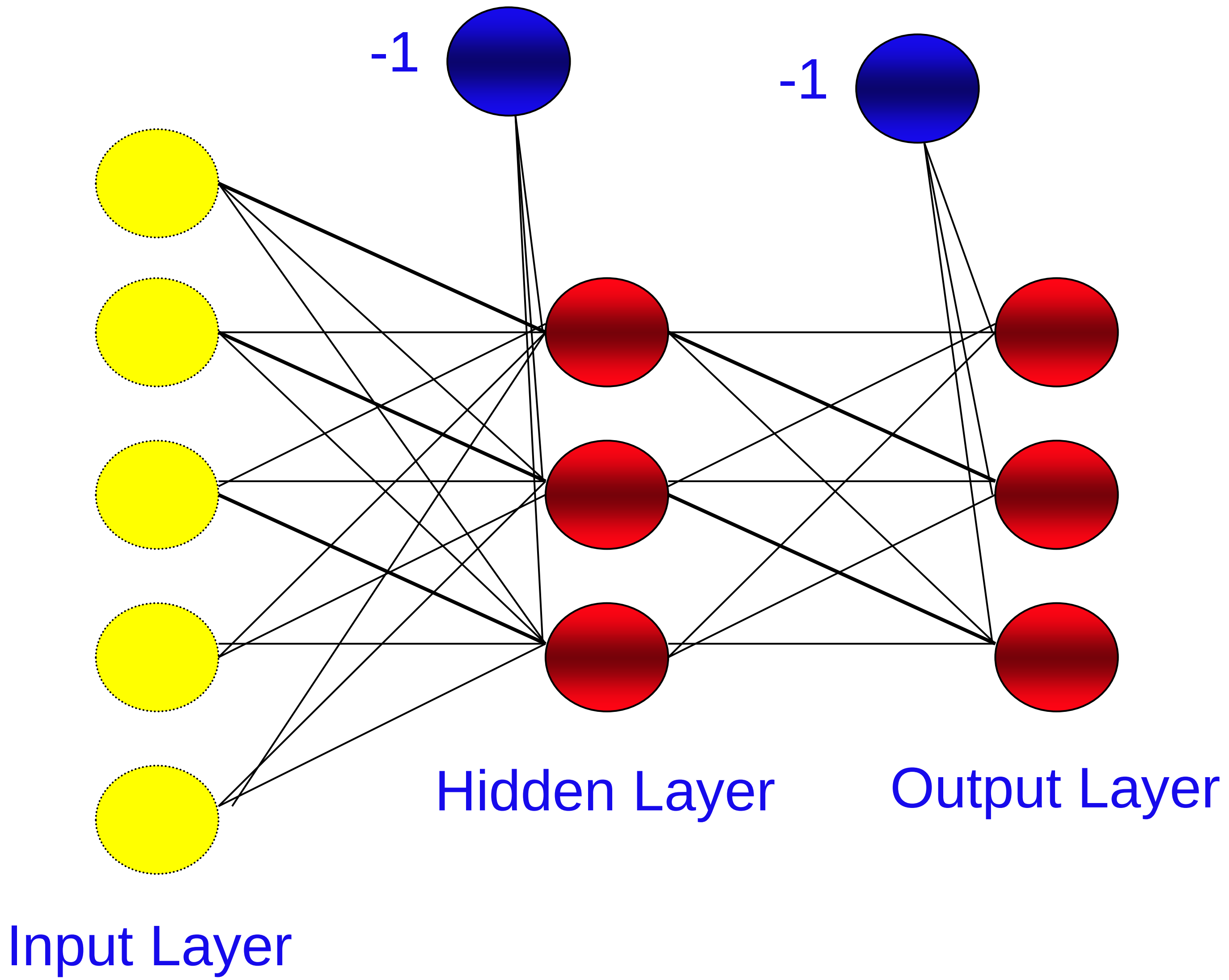
Perceptron

- How can we make the perceptron more powerful?

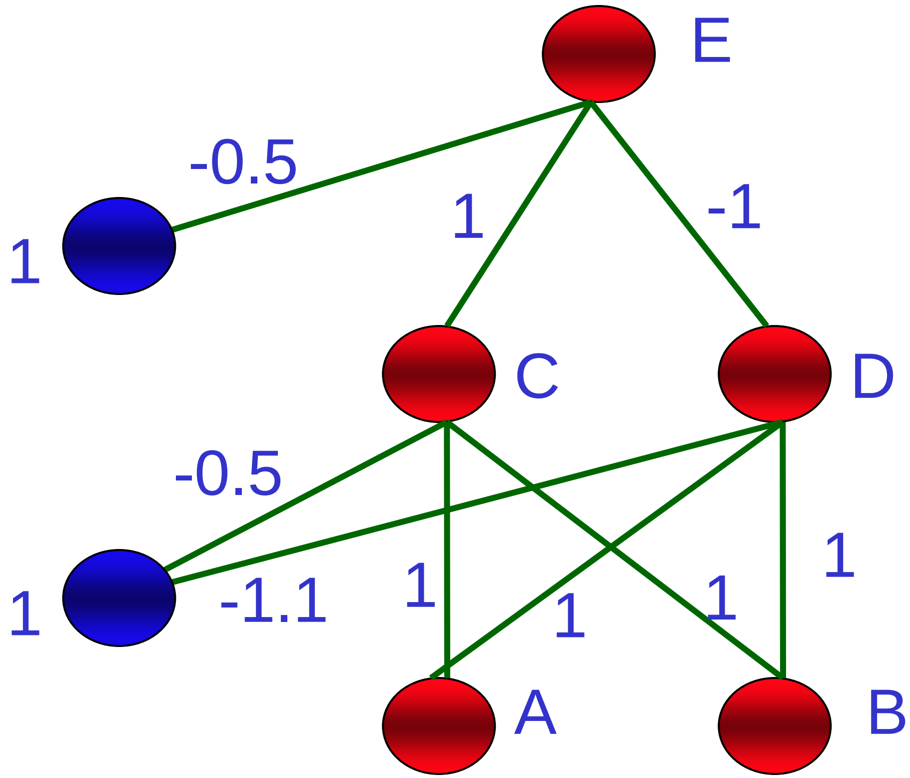
- More layers in the networks?
- More connections?

- Perceptron: one layer of weights
- Multi-layer perceptron: at least 2 layers of weights

The Multi-Layer Perceptron



XOR Again

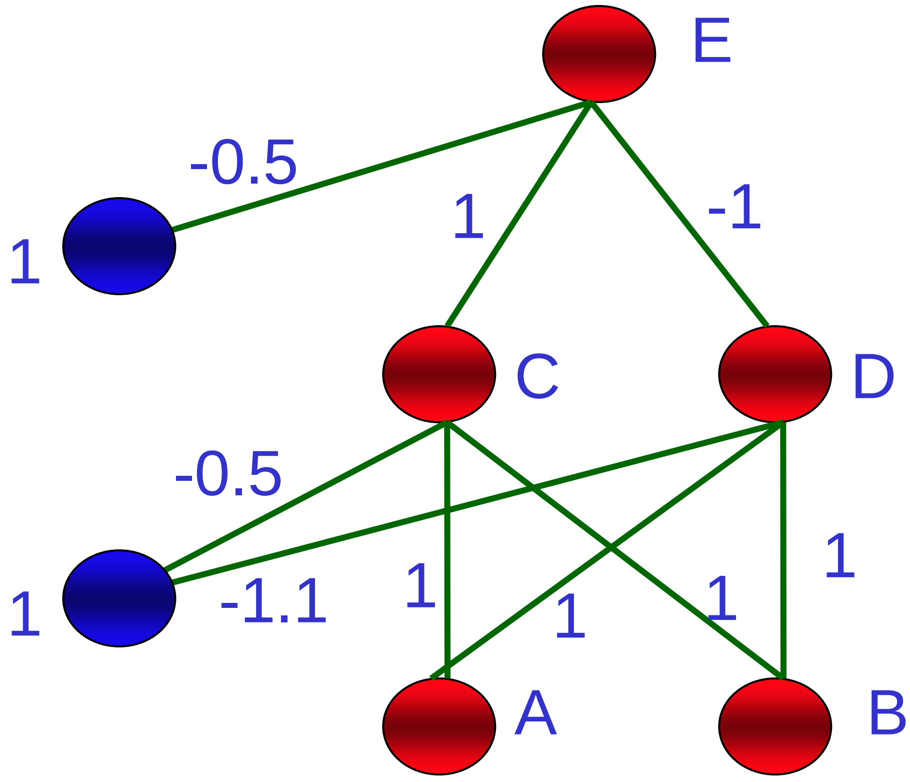


So E does not fire

C does not fire,
D does not fire

Apply input 0 0

XOR Again



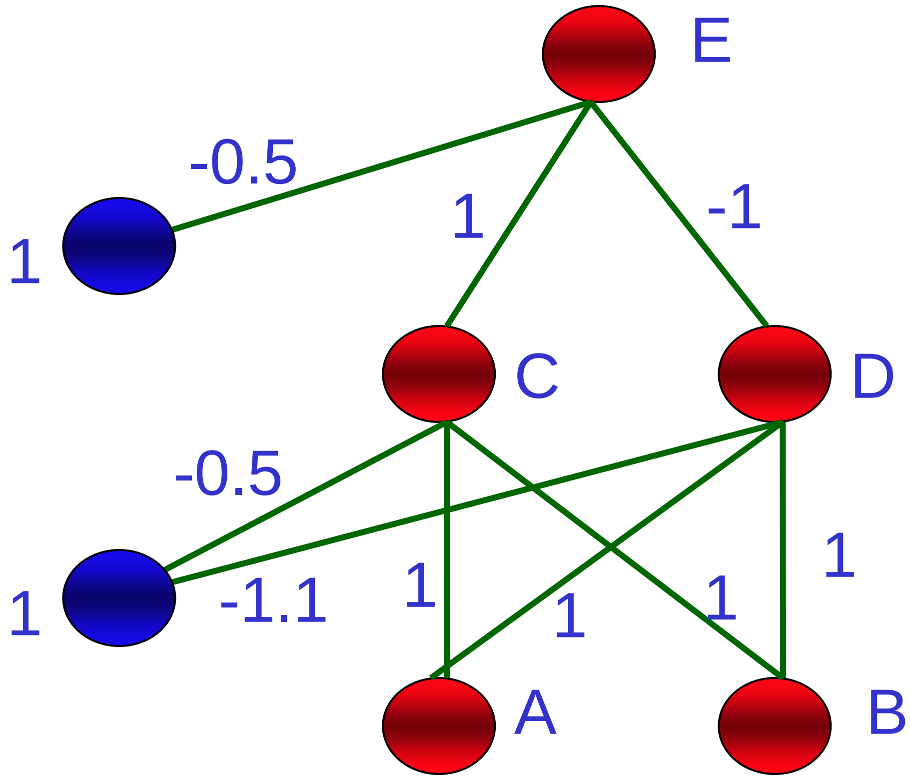
So overall E fires

C fires
D does not fire

Apply input 1 0

Same for input 0 1

XOR Again

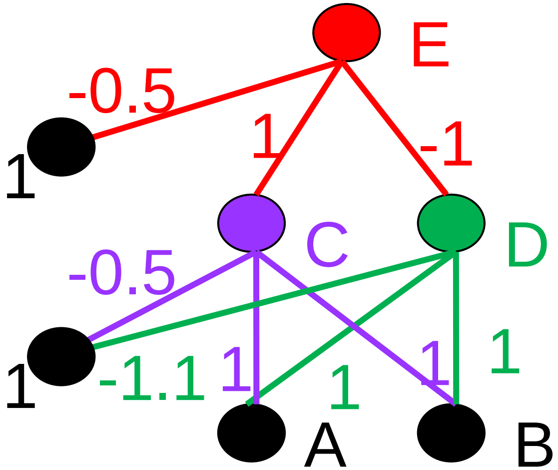


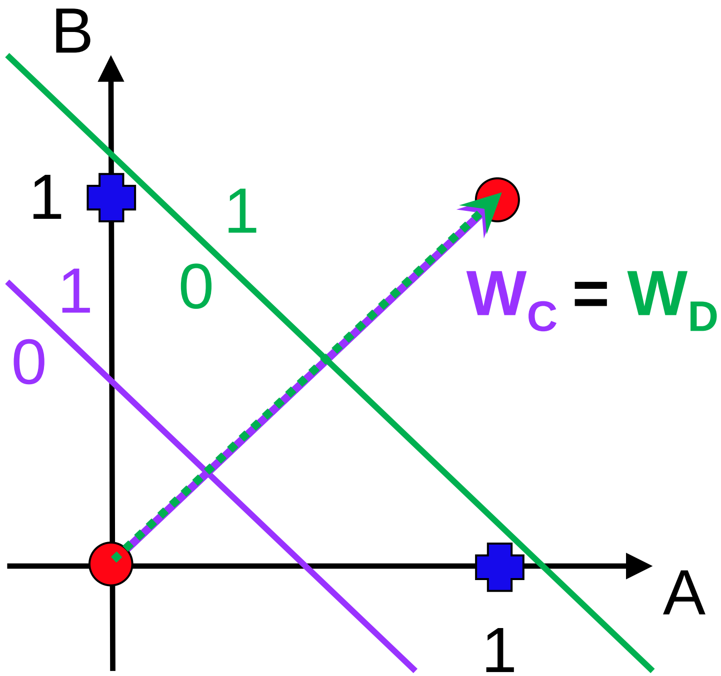
So E does not fire

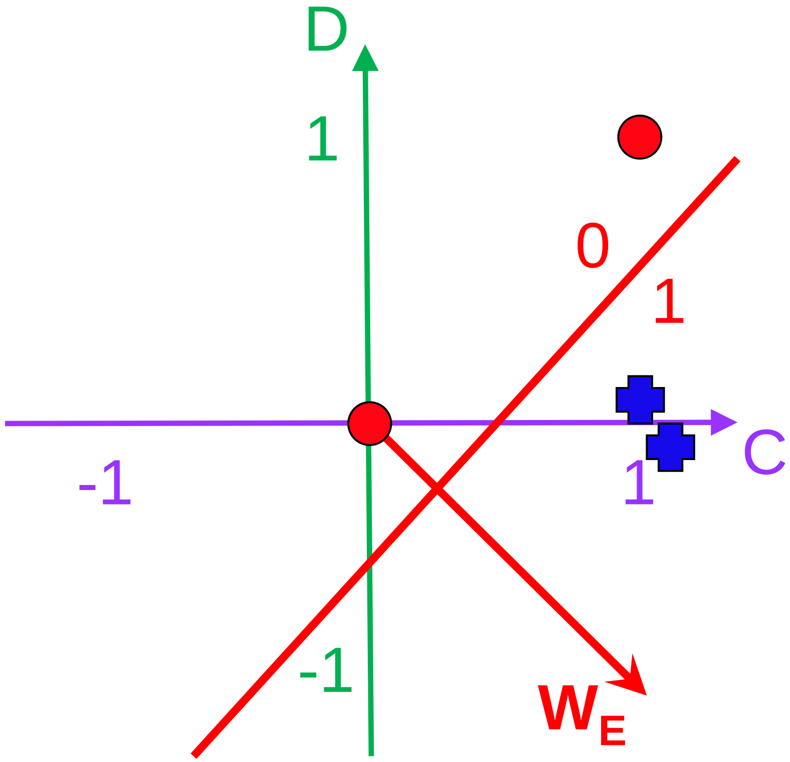
C fires
D fires

Apply input 1 1

XOR Again



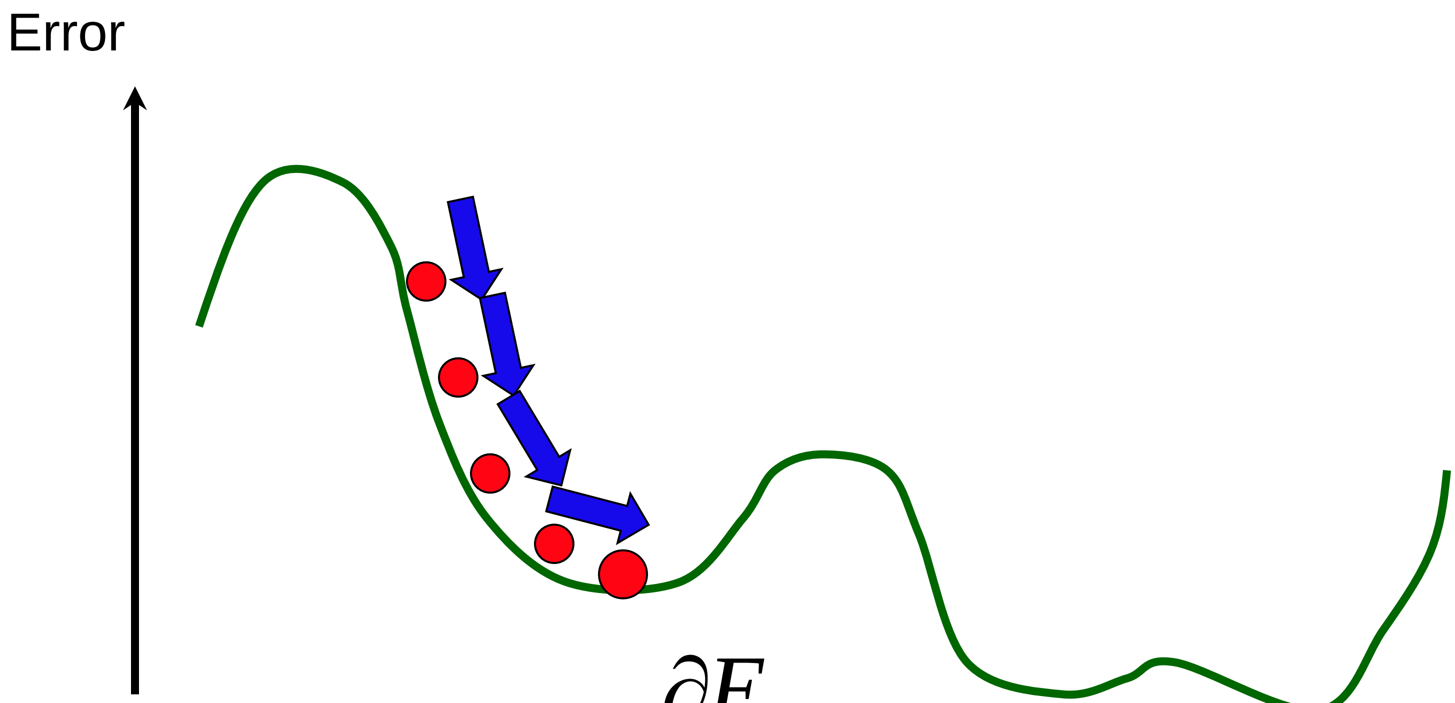




Gradient Descent

- The MLP ***can*** solve XOR
- How do we choose the weights?
- Harder than for the perceptron
  - More weights
  - Which weights are wrong? Input-hidden or hidden-output?
- Use gradient descent learning
- Compute gradient ⇨ differentiation

Gradient Descent



$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

If we differentiate function E, we get the negative gradient of the function (direction of change)

An Error Function

- If E=*(t−y)* → pos. and neg. errors would cancel out
- Better: ***sum-of-squares error***

$$E(\mathrm{w}) = \frac{1}{2}\sum_i (t_i - y_i)^2 = \frac{1}{2}\sum_i \left(t_i - \sum_j w_{ij} x_j \right)^2$$

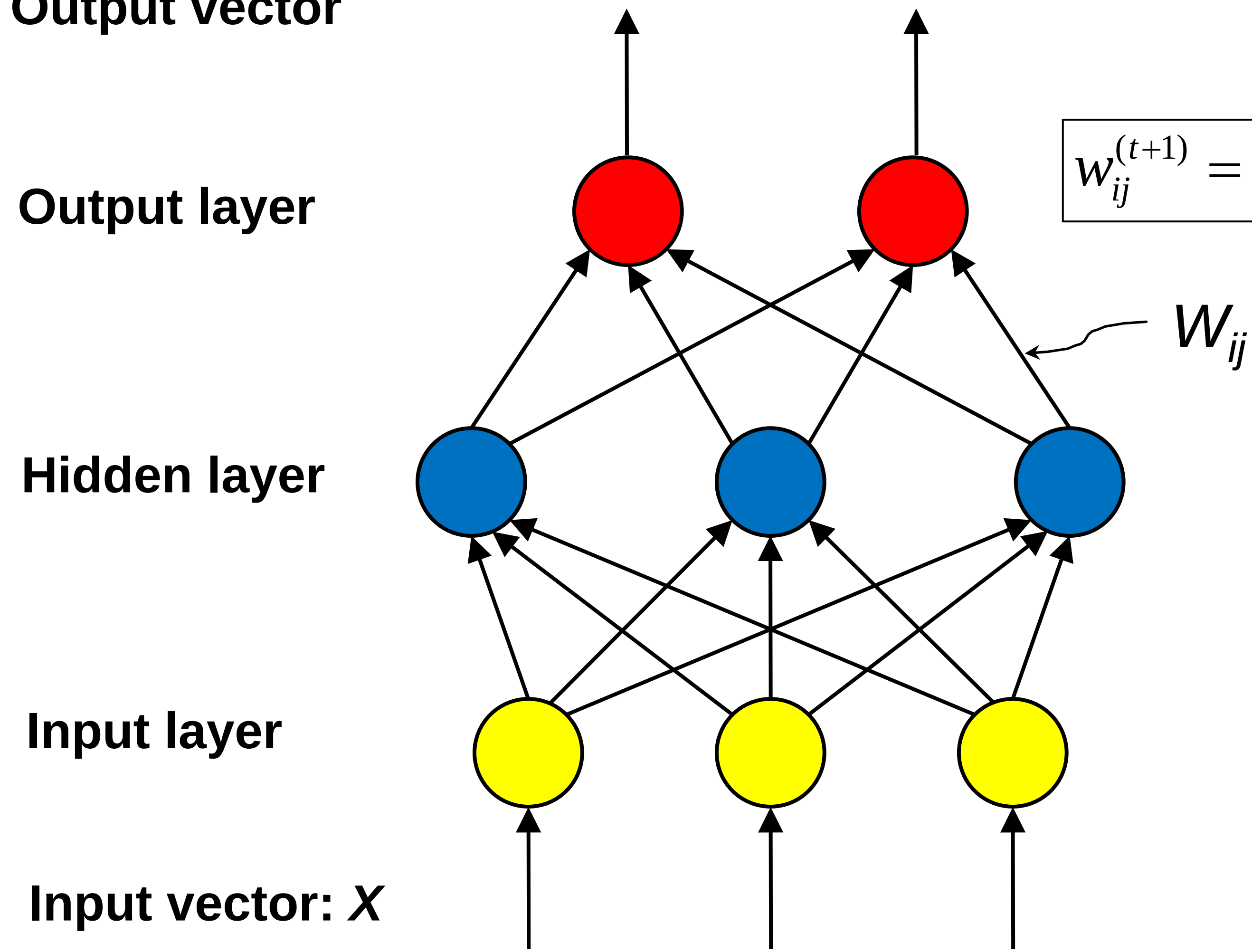
- We will ignore the threshold function in the output neurons

$$\Rightarrow \qquad -\frac{\partial E}{\partial w_{ij}} = (t_i - y_i) \cdot x_j$$ ***Gradient descent***

- Rule for the weights to the output layer (also for perceptron)

A Multi-Layer Feed-Forward Neural Network

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta(t_i - y_i)x_j$$



How a Multi-Layer Neural Network Works

- The ***inputs*** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units of the ***input layer***
- They are then weighted and fed simultaneously to a ***hidden layer***
- The weighted outputs of the last hidden layer are input to units making up the ***output layer***, which emits the network's prediction
- The network is ***feed-forward***, i.e. none of the weights cycles back to a unit in the same or a previous layer
- From a statistical point of view, networks perform ***nonlinear regression***: given enough hidden units and enough training samples, they can closely approximate any function

Decide on the Network Topology

- # of units in the *input layer* ←fixed through the application
  - One input unit per domain value
- # of *hidden layers* (if > 1)
  - Complex function – transformations? Hierarchical features?
- # of units in *each hidden layer*
  - Complex function – many features?
- # of units in the *output layer* ←fixed through the application
  - One output unit for each variable in regression
  - A single output unit for two-class classification
  - For more than two classes, one output unit per class
    - output values may be coupled by a softmax function
- *May repeat training with different network topologies*

Weight Initialisation & Data Preprocessing

- Initialize the network with small random weights
  - All-same weights would lead to symmetry-breaking problem: all hidden units will have same activations and learn the same
  - Large weights could lead to saturation of the transfer function

- Normalizing the input values for each attribute measured in the training tuples, e.g.
  - shift & scale attribute values to be in the interval [0.0 .. 1.0], or
  - shift & scale them to have mean=0, variance=1;
  - this may be done per attribute or over all attributes

all attributes have same importance

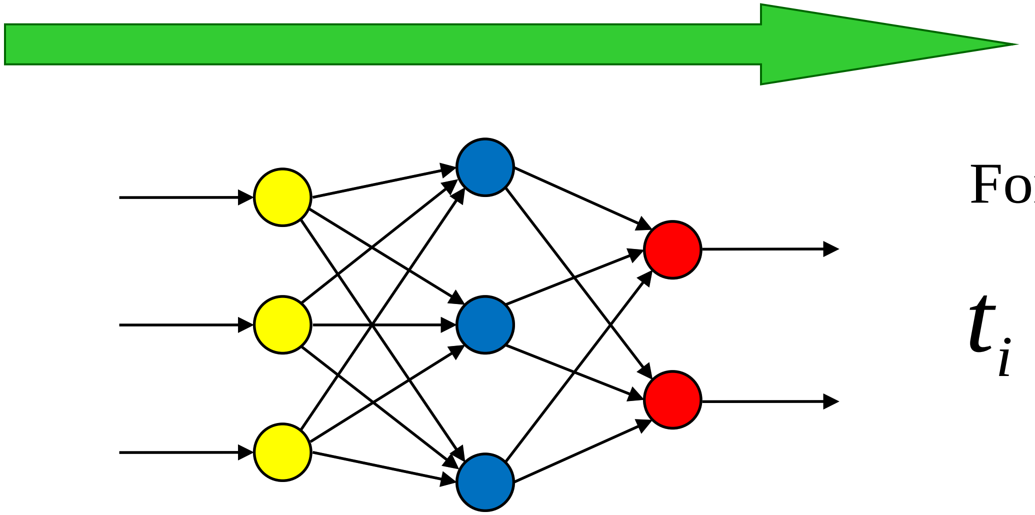
attributes keep their relative importance

- *May repeat training with a different set of initial weights*

Training MLP

(1) Forward Pass

- Put the input values in the input layer
- Calculate the activations of the hidden nodes
- Calculate the activations of the output nodes
- Calculate the errors using the targets



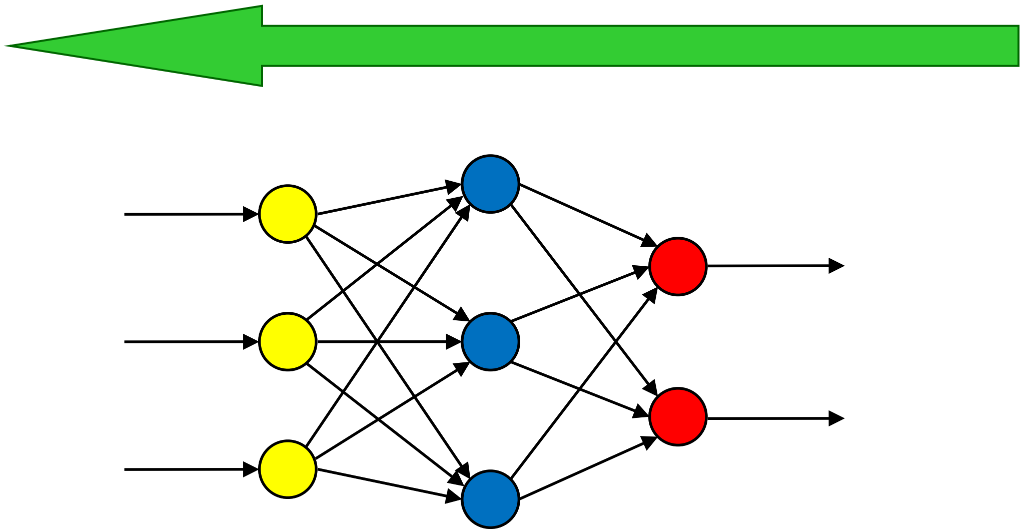
For example

$$t_i - y_i$$

Training MLPs

(2) Backward Pass

- Using output errors, update last layer of weights
- Calculate hidden-layer errors, update hidden-layer weights
- Work backwards through the network
- Error is backpropagated through the network



Error Backpropagation

- Iteratively process training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to ***minimize*** the ***squared error*** between network's prediction and actual target value
  - This minimizes the ***mean*** square error over the entire data set
- Errors are computed "***backwards***": from the output layer, through each hidden layer down to the first hidden layer, hence "***backpropagation***"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - For each data point:
    - Propagate the inputs forward (by applying activation function)
    - Propagate the error backwards (backpropagation)
    - Update the weights and biases (using inputs and errors)
  - Terminating condition (when error small; test error increases; etc.)

Activation Function

- In the analysis we ignored the activation function
  - The threshold function is not differentiable

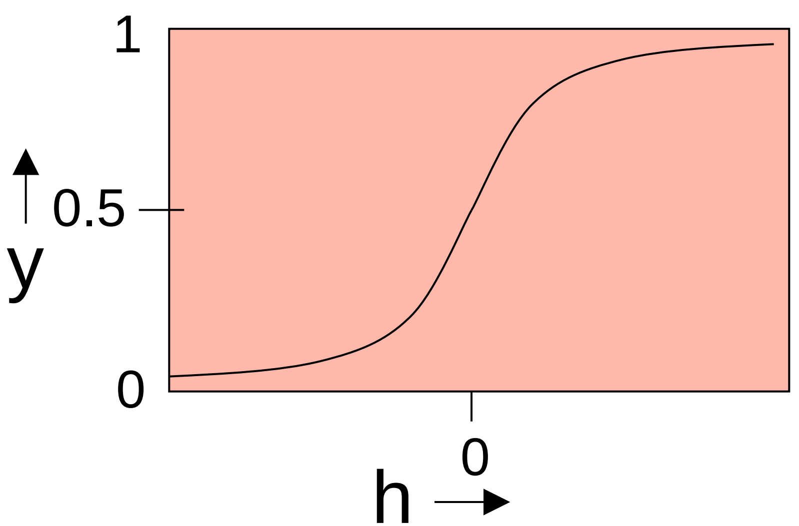
- What do we want in an activation function?
  - Differentiable
  - Should saturate (become constant at ends)
  - Change between saturation values quickly

Sigmoid Neurons

- Sigmoidal / logistic transfer function:
  - gives a real-valued, positive output
  - bounded in interval [0,1]
  - easily differentiable, positive derivative
  - output can be interpreted as a ***probability*** of a binary output to be =1 (or of producing a spike) → stochastic binary neurons

$$h = b + \sum_j x_j w_j$$

$$\boxed{y = g(h) = \frac{1}{1+e^{-h}}}$$



Sigmoid Activation Function for a Neuron

Transfer function:

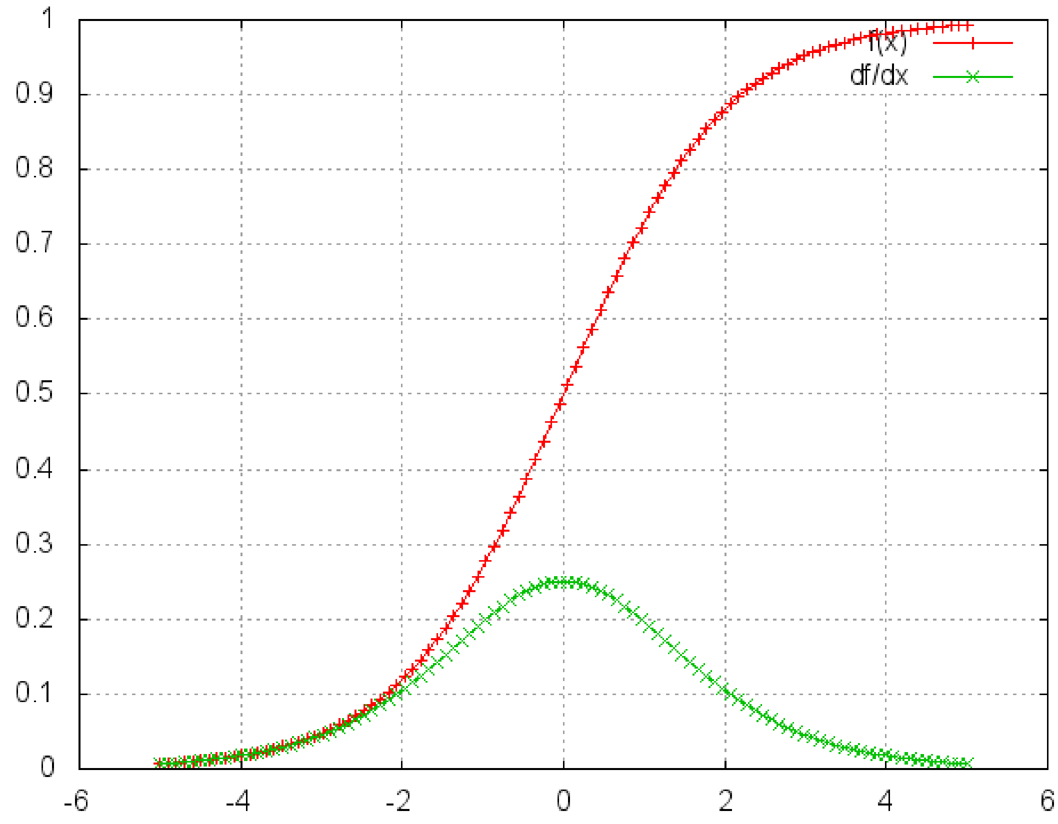
$$g(h) = \frac{1}{1+\exp(-h)}$$

Derivative:

$$g'(h) = \frac{\partial g(h)}{\partial h}$$

$$= \dots$$

$$= g(h) \cdot (1 - g(h))$$



The derivative can be expressed as a function of the ***outputs***.

Overview of Transfer Functions

Transfer function:	Corresponding derivative:
• sigmoid $g(h) = \dfrac{1}{1+\exp(-h)}$	$g'(h) = g(h) \cdot (1 - g(h))$
• linear $g(h) = h$	$g'(h) = 1$
• threshold function $g(h) = \begin{cases} 1 & h \geq \theta \\ 0 & h < \theta \end{cases}$	no useful derivative
• sign $g(h) = \begin{cases} 1 & h \geq \theta \\ -1 & h < \theta \end{cases}$	no useful derivative

Error Terms

- Need to differentiate the sigmoid function
- Gives us the following ***error terms*** (deltas)
  - For the outputs

$$\delta_i = (t_i - y_i)\underbrace{y_i(1 - y_i)}_{\text{derivative}}$$

  - For the hidden nodes (with activations y_j^{hid})

$$\delta_j = \underbrace{y_j^{hid}(1 - y_j^{hid})}_{\text{derivative}}\sum_i w_{ij}\delta_i$$

Update Rules

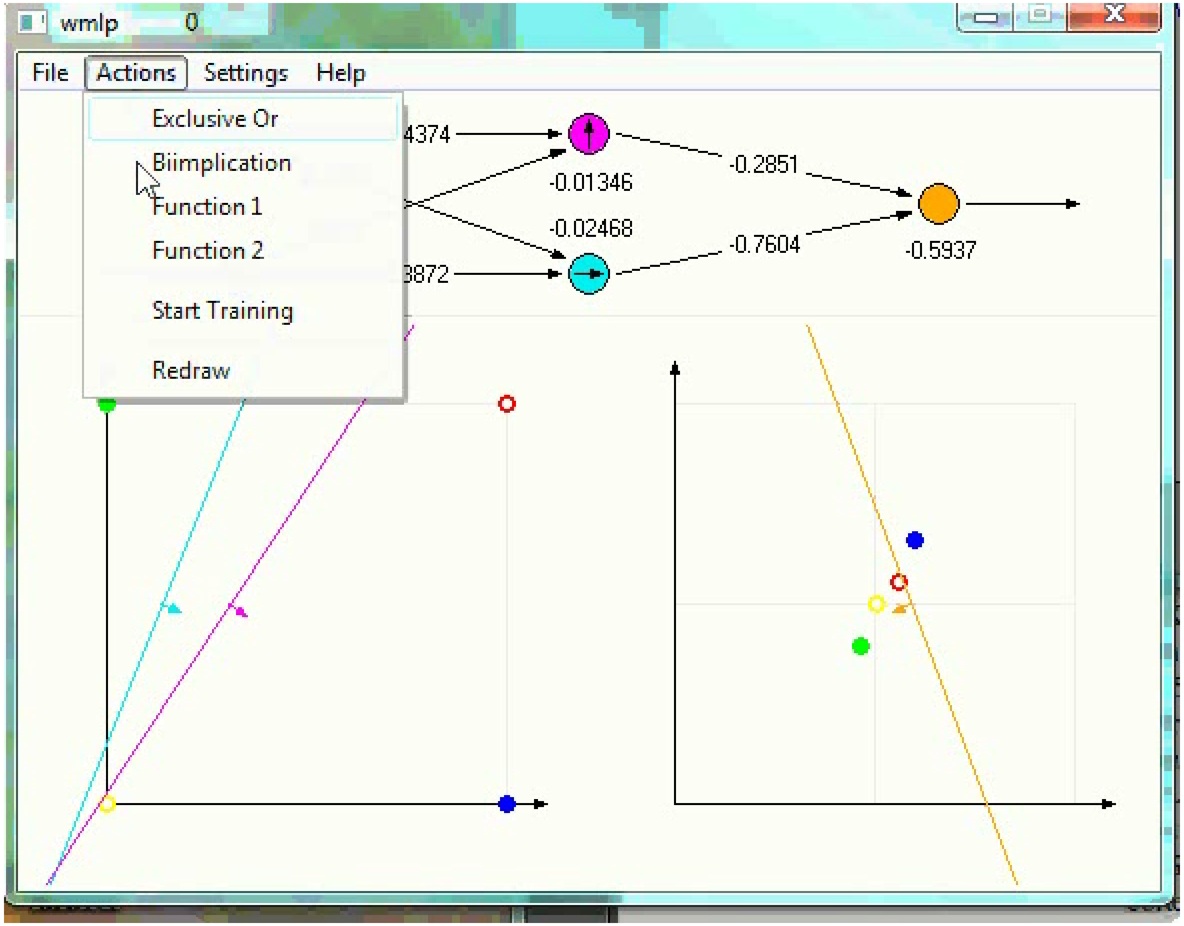
- This gives us the necessary update rules
  - For the weights connected to the outputs:

$$w_{ij} \leftarrow w_{ij} + \eta \delta_i y_j^{\mathrm{hid}}$$

  - For the weights connected to the hidden nodes:

$$v_{jk} \leftarrow v_{jk} + \eta \delta_j x_k$$

MLP training a XOR problem



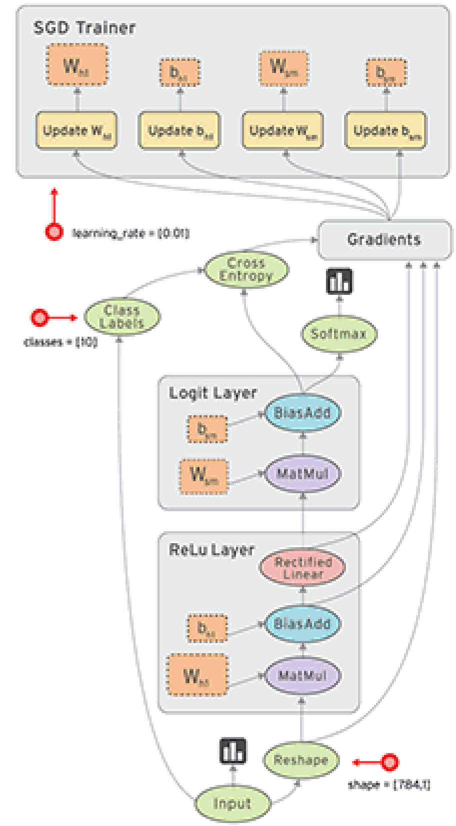
[http://www.borgelt.net/mlpd.html]

Tensorflow

- Open source package for deep MLP learning by google
- Given a network structure and cost function:

  → does automatic differentiation and learning

- Online demo for small networks:
  - http://playground.tensorflow.org



Network Topology

- How many layers?
- How many neurons per layer?
- Experiments
  - Often two or three hidden layers (but new research into deep learning networks…)
  - Determine size of layers (usually get smaller)
  - Test several different networks

Deep Learning (MLP with many Layers)

- Enabled by GPUs, multi-core CPUs and large data sets
- Convolutional filtering layers (only lower layers)
  - Use small, replicated (shared) weights on large inputs
  - Lowest-layer filters sometimes not learned

    → e.g. Sobel-, Gabor-, centre-surround/whitening filters
- MAX-Pooling layers (over convolutional layers)
  - Create invariances (e.g. to shift/scale); reduce dimensionality

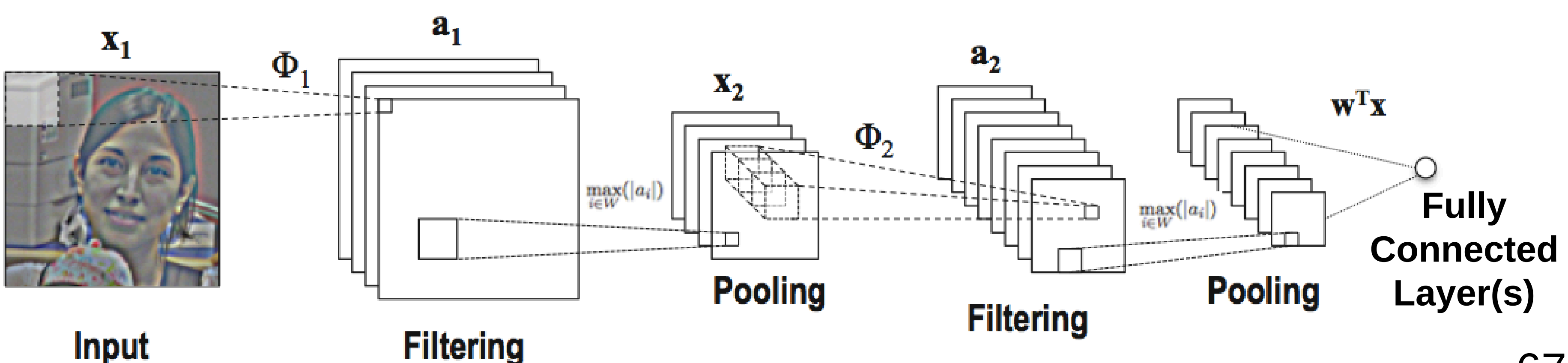
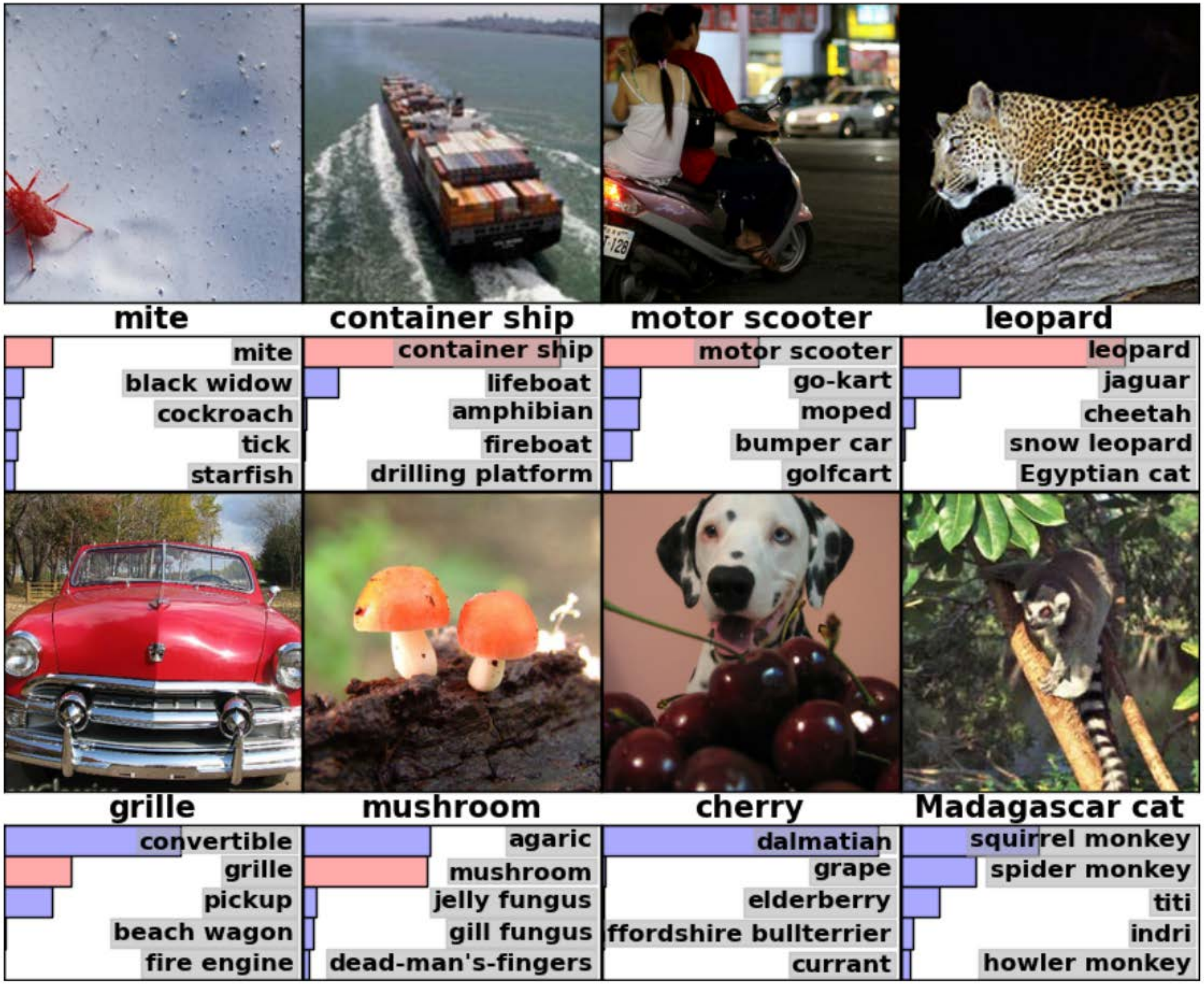


Figure modified from: https://www.ece.nus.edu.sg/stfpage/eleqiz/deep_saliency.html

Deep Learning Multi-Class Classification



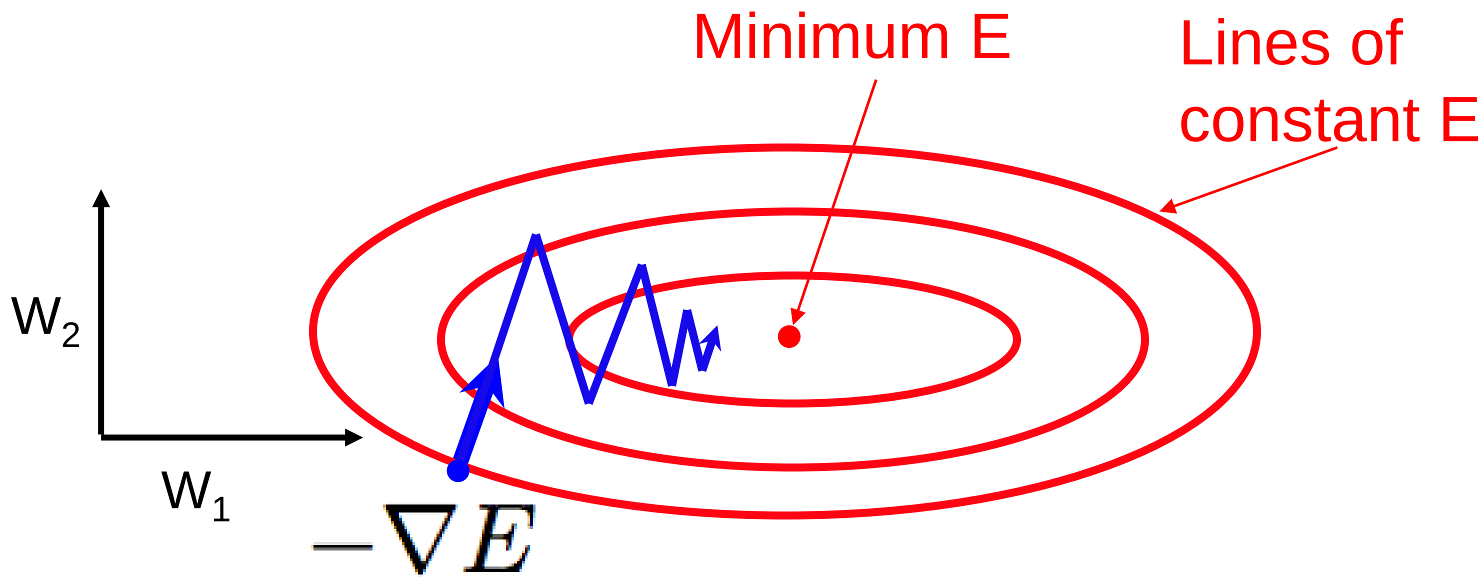
ImageNet Classification with Deep Convolutional Neural Networks. Krizhevsky, Sutskever, Hinton. NIPS, 2012

Batch and Incremental Learning

- When should the weights be updated?
  - After all inputs seen (***batch***)
    - Accurate estimate of gradient
    - Converges systematically to the (local) minimum
    - Requires many epochs (passes through the whole dataset)
  - After each input is seen (***incremental, online***)
    - Simpler to program
    - Handles infinite amount of data (continual learning)
    - Noise may help escaping from saddle points in the energy landscape, or even from local minima
    - Pitfall: data distribution may drift (also within batch data).

      Remedy: randomize order of presentation

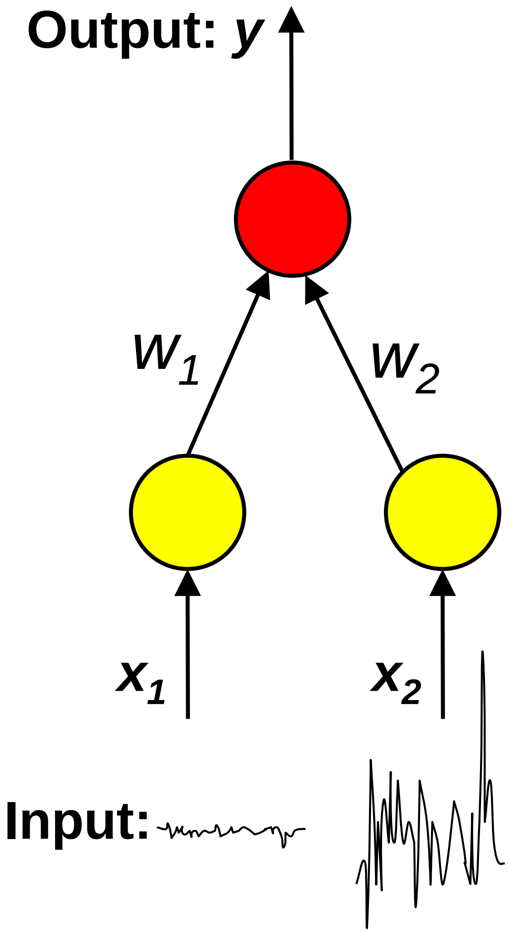
Gradient Descent



- Local gradient does not point towards minimum
- Gradient descent with large learning rate $\longrightarrow$ oscillations
- Long learning time!

Gradient Descent

- Learning rule (for the simple case of the perceptron):



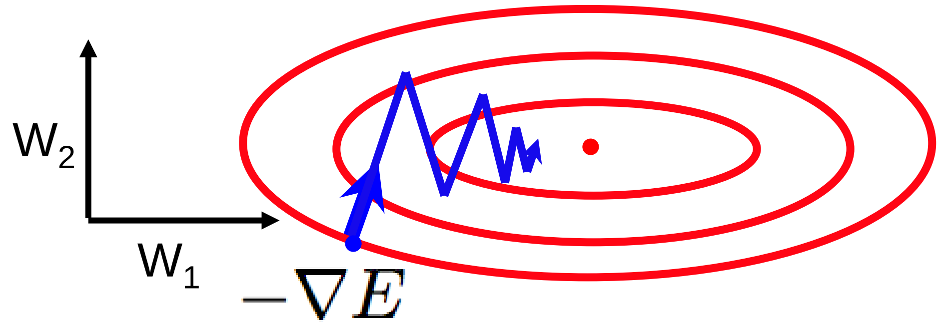
$$-\frac{\partial E}{\partial w_{ij}} = \left(t_i - y_i\right)\cdot x_j$$

- Assume: inputs x_1 and x_2 are of similar importance for classification
- both have mean zero: $\mu(x_1) = \mu(x_2) = 0$
- but std. deviations differ: $\sigma(x_1) < \sigma(x_2)$

$\rightarrow$ weights should be: $w_1 > w_2$

but average updates: $|\Delta w_1| < |\Delta w_2|$

Momentum Term

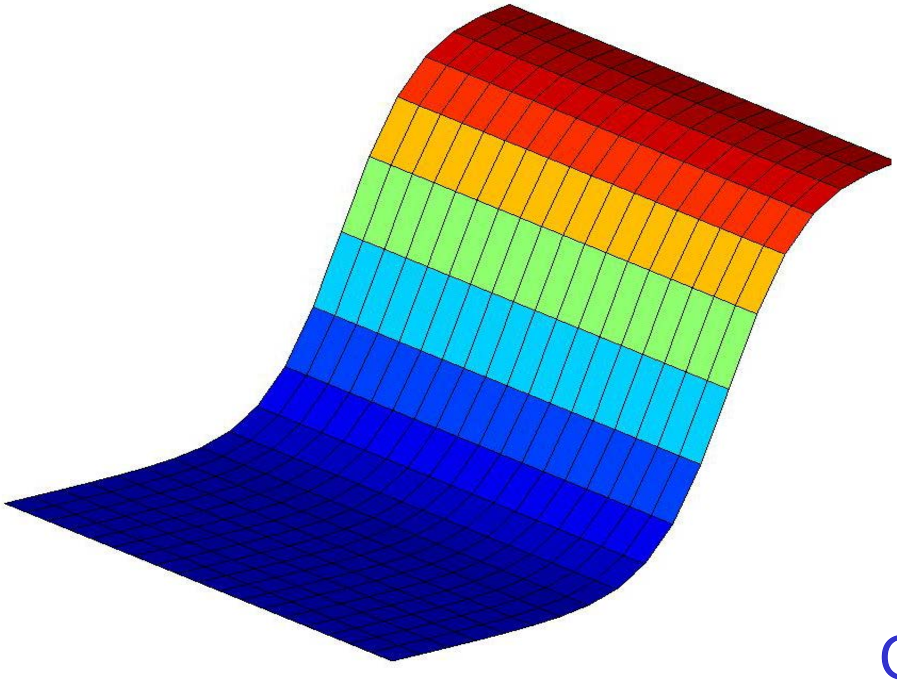


$$w_{ij}^{\tau} \leftarrow w_{ij}^{\tau-1} + \eta \delta_i y_j^{\text{hid}} + \underbrace{\alpha \Delta w_{ij}^{\tau-1}}$$

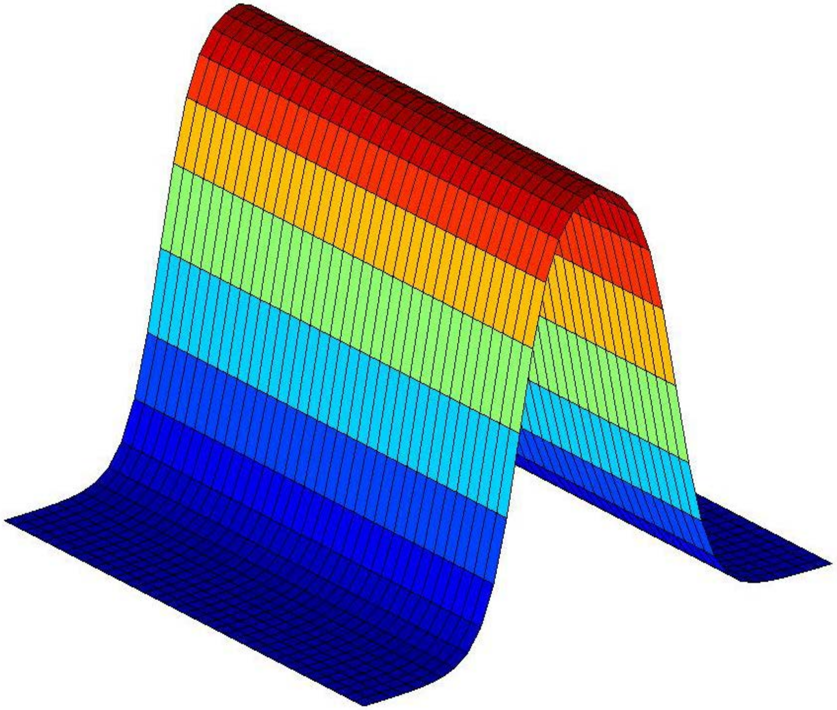
Add contribution from previous weight change (momentum)

- Counteracts oscillations, by averaging previous and current updates (relevant for batch learning)
- Averages out noise (relevant for on-line learning)
- More stable, leads to faster learning

Learning Capacity

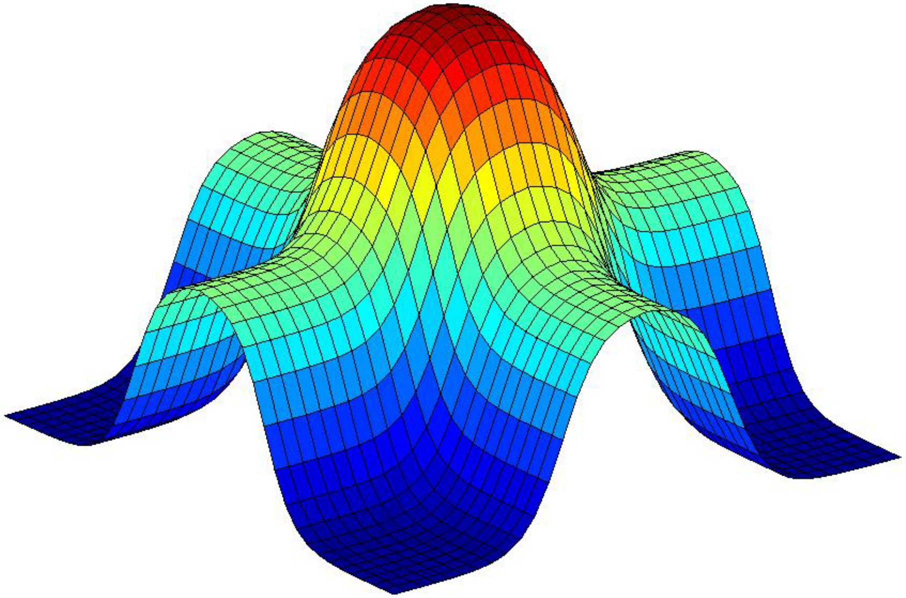


Output of one sigmoid



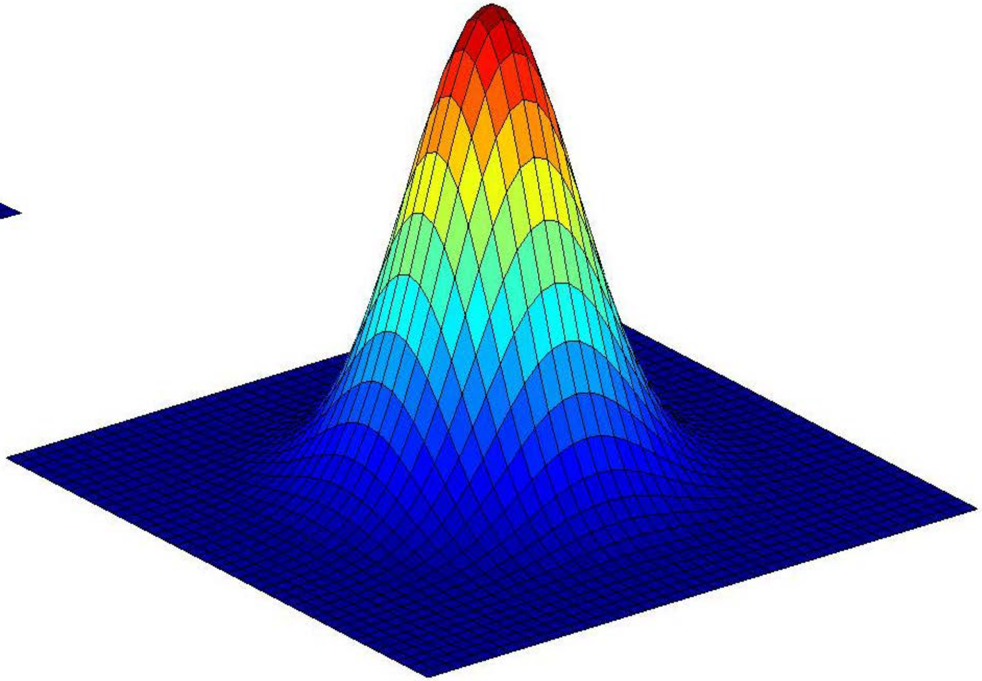
Output of two sigmoids

Learning Capacity

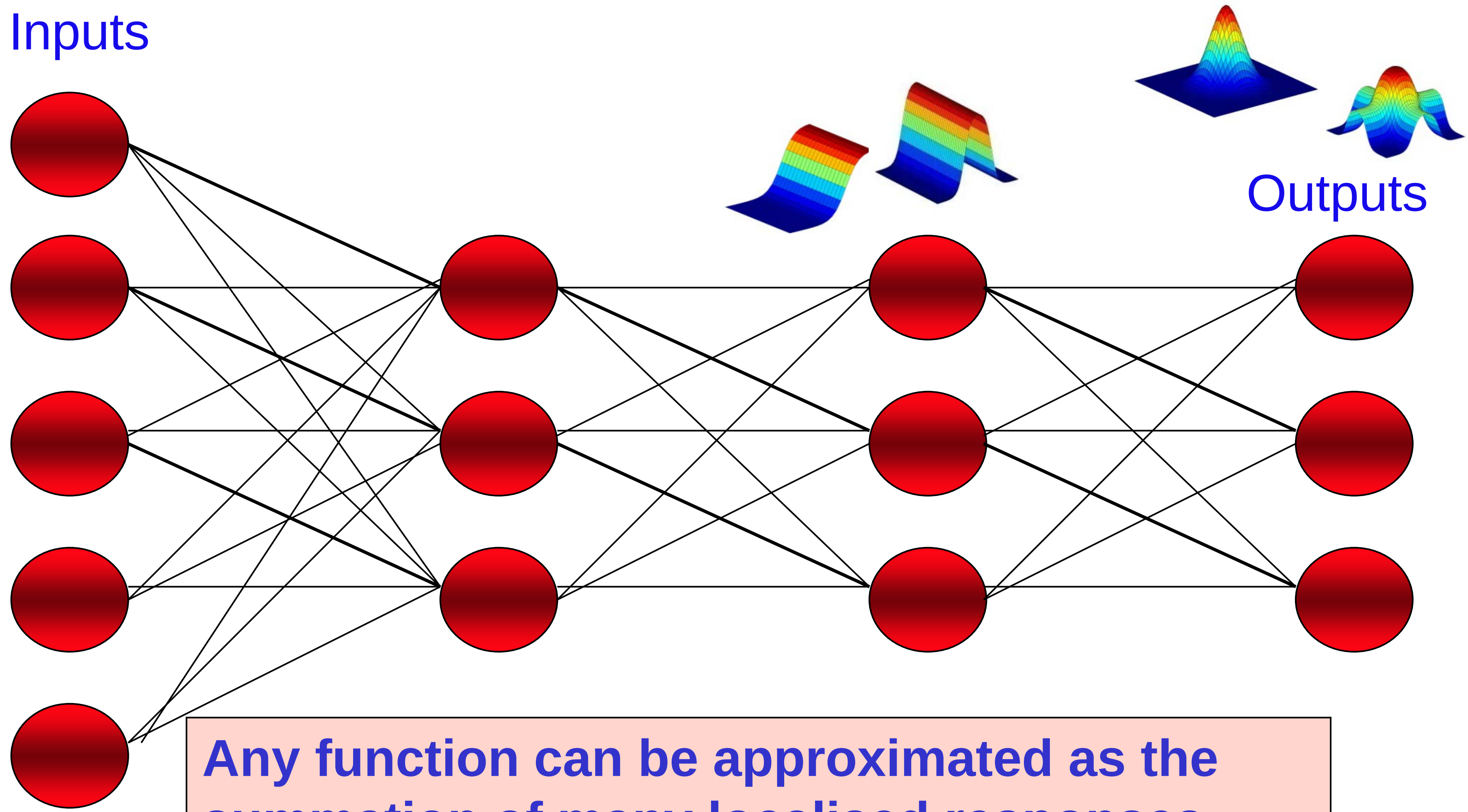


Addition of two ridges
Unique maximum

Addition of more ridges
and transformation with
another sigmoid
Localised response

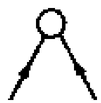
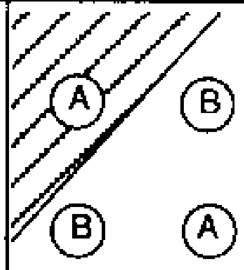
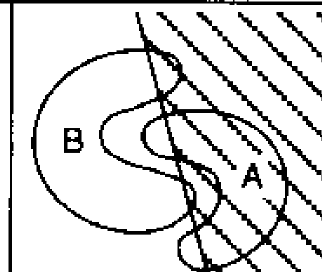
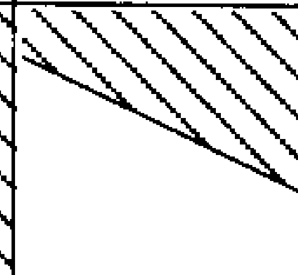
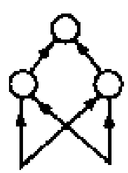
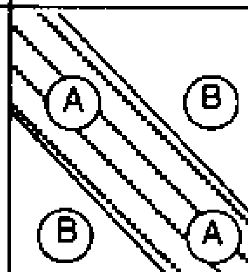
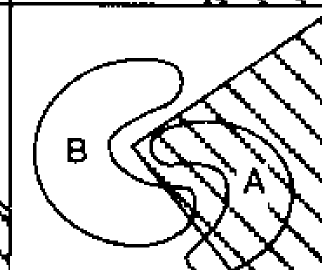
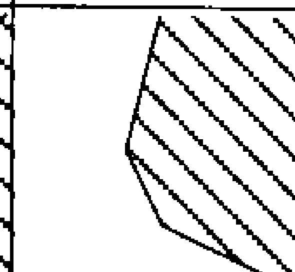
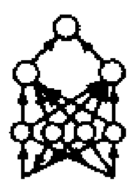
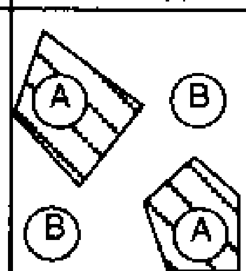
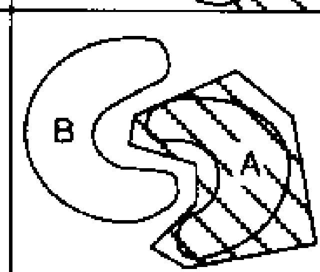
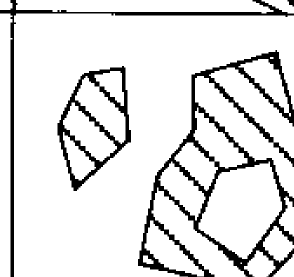


Learning Capacity



Any function can be approximated as the summation of many localised responses

Decision Boundaries (Lippmann)

Structure	Types of Decision Regions	Exclusive OR Problem	Classes with Meshed Regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded by Hyperplane			
Two-Layer 	Convex Open or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by Number of Nodes)			

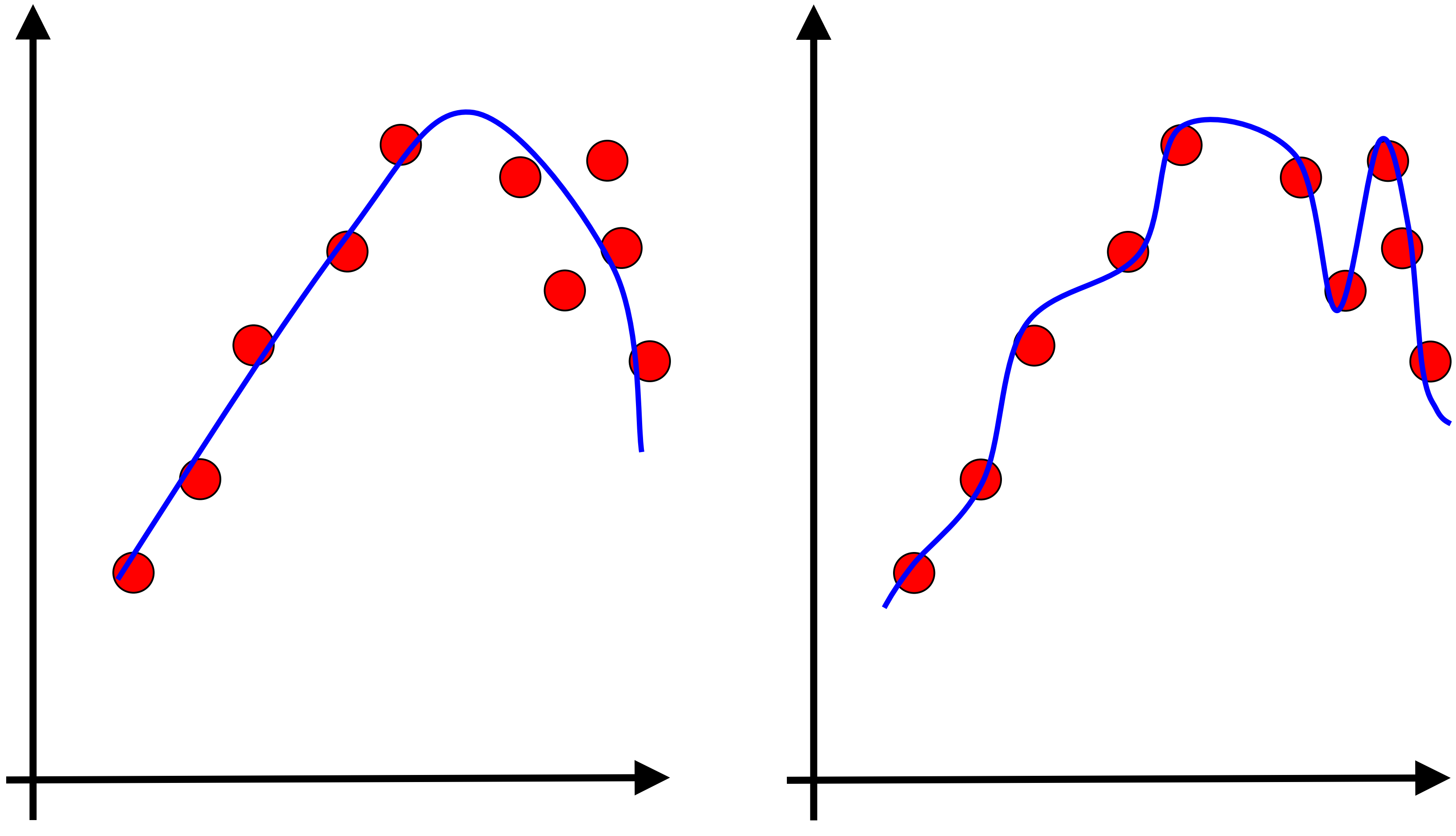
Generalisation

- Aim of neural network learning:

  Generalise from training examples to all possible inputs

- Undertraining is bad
- Overtraining is worse

Overfitting

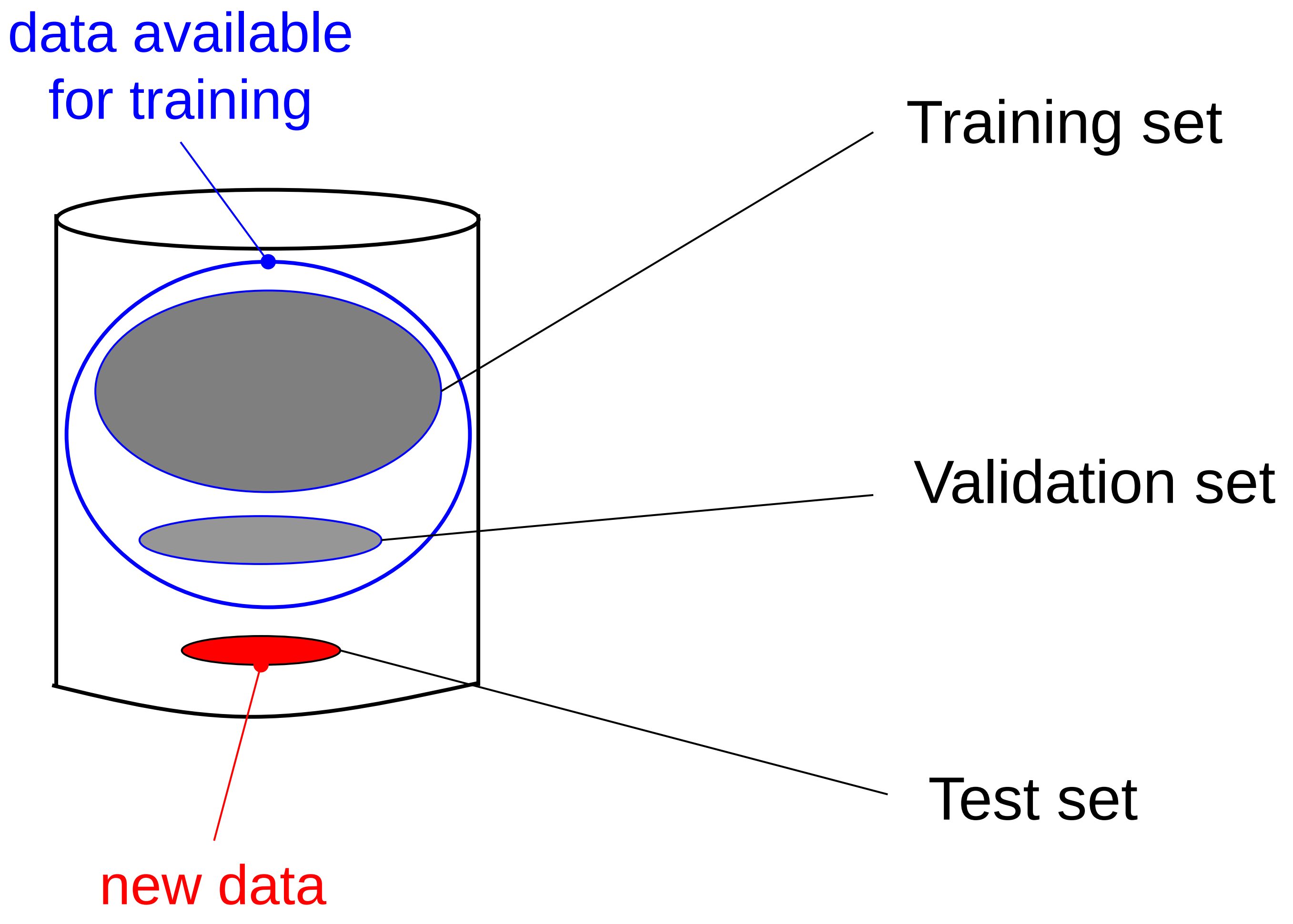


Testing

How do we evaluate our trained network?

- The error on the training data is biased and hides overfitting
- Validate on a separate validation set
  - evaluate periodically on this validation set during training (while training only on the training set)
  - indicator of overfitting: the validation error increases
- After training, test the final model on the test set

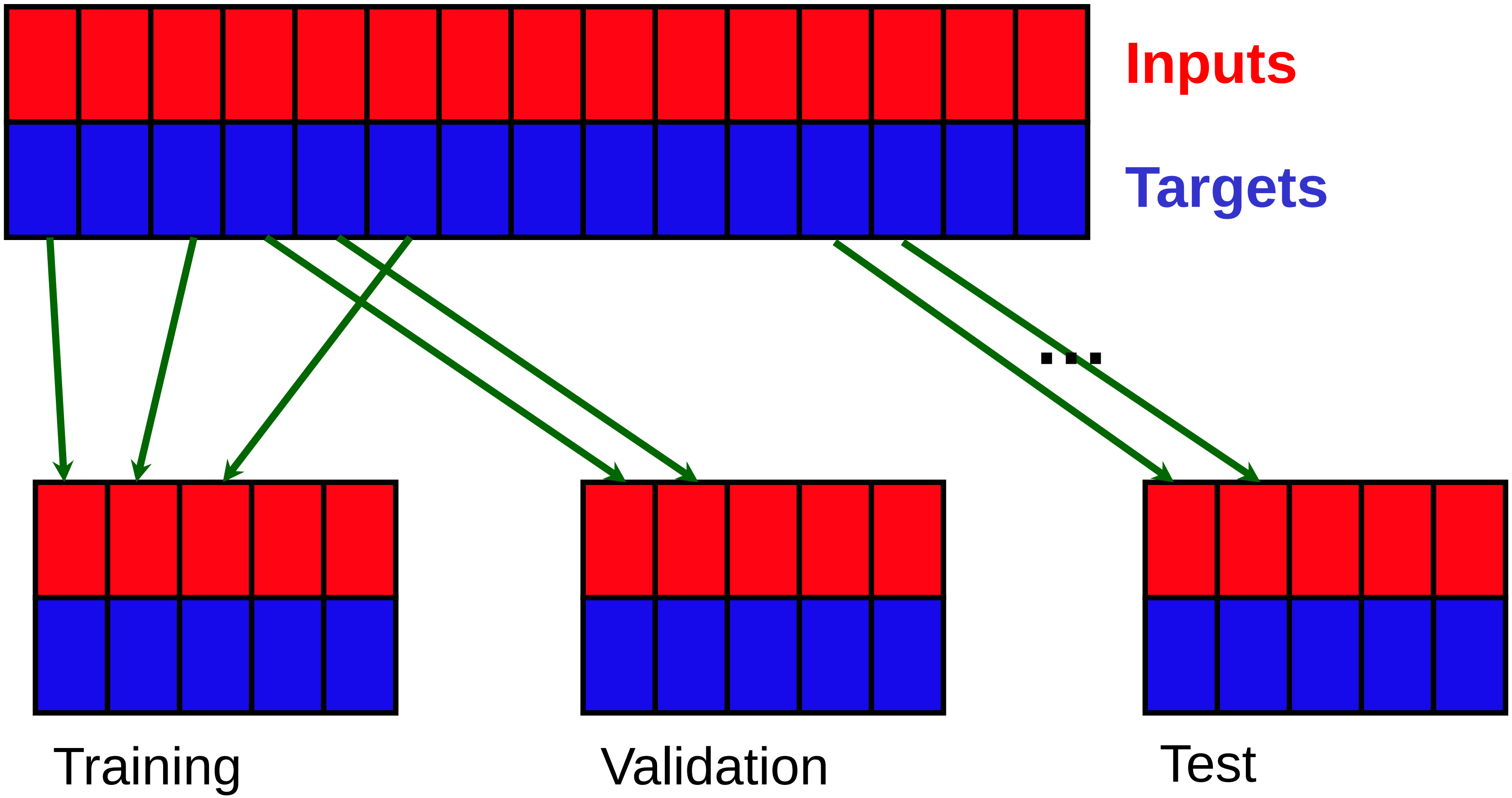
Using Training, Validation and Test Data



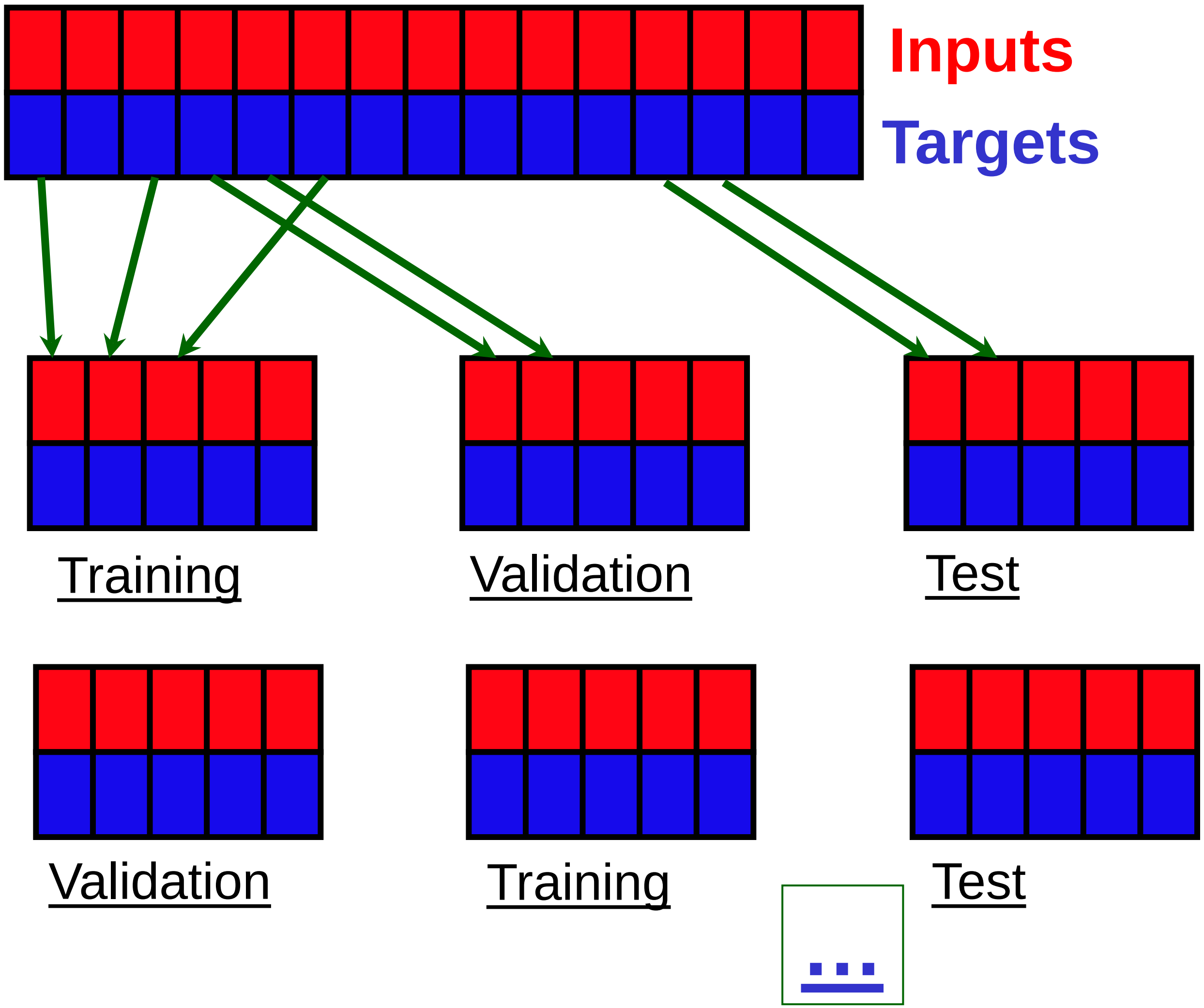
Validation

- Of the data that is available during training, keep a subset for validation
- Train the network on training data
- Periodically, stop and evaluate on validation set
- After training has finished, test on test set
- This is coming expensive on data!

Hold Out Cross Validation



Multifold Cross Validation



Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

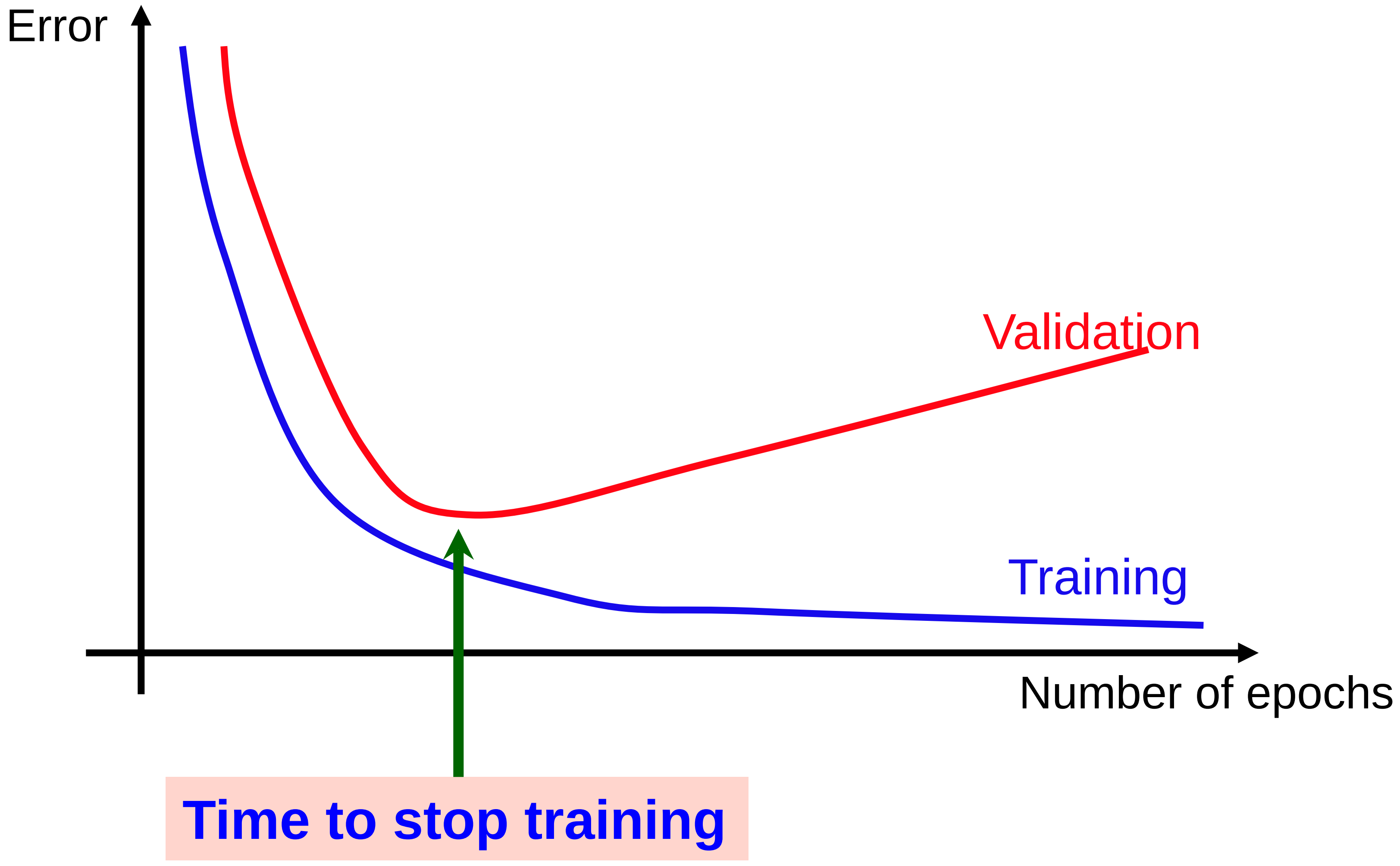
- ***Holdout method***
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test ("hold-out") set (e.g., 1/3) for accuracy estimation
  - Expensive on the data
- ***Cross-validation*** (*k*-fold, where k = 10 is popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - Random subsampling: *k* folds, with *random split* between training and test set each time; accuracy = avg. of the accuracies obtained

Early Stopping

When should we stop training?

- Could set a minimum training error
  - Danger of overfitting
- Could set a number of epochs
  - Danger of underfitting or overfitting
- Can use the validation set
  - Measure the error on the validation set during training

Early Stopping



Backpropagation and Interpretability

- ***Rule extraction*** from networks: network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers

- ***Sensitivity analysis***: assess the impact that a given input variable has on a network output.
  - The knowledge gained from this analysis can be represented in rules

Summary: Neural Networks as a Classifier

- Weaknesses
  - Long training time (but same with humans …)
  - Some parameters have to be determined empirically, e.g.
    - network topology, transfer functions, learning rate, etc.
  - Challenging to interpret the symbolic meaning behind the learned weights and of “hidden units”
  - Cannot handle well missing values
- Strengths
  - Well-suited for continuous-valued inputs and outputs
  - High tolerance to noisy data
  - Generalisation ability: classify untrained patterns
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Recent techniques extract rules from trained neural networks
  - Relationship to brain

WTM Student Project

(a) Learner and teacher

(b) Objects

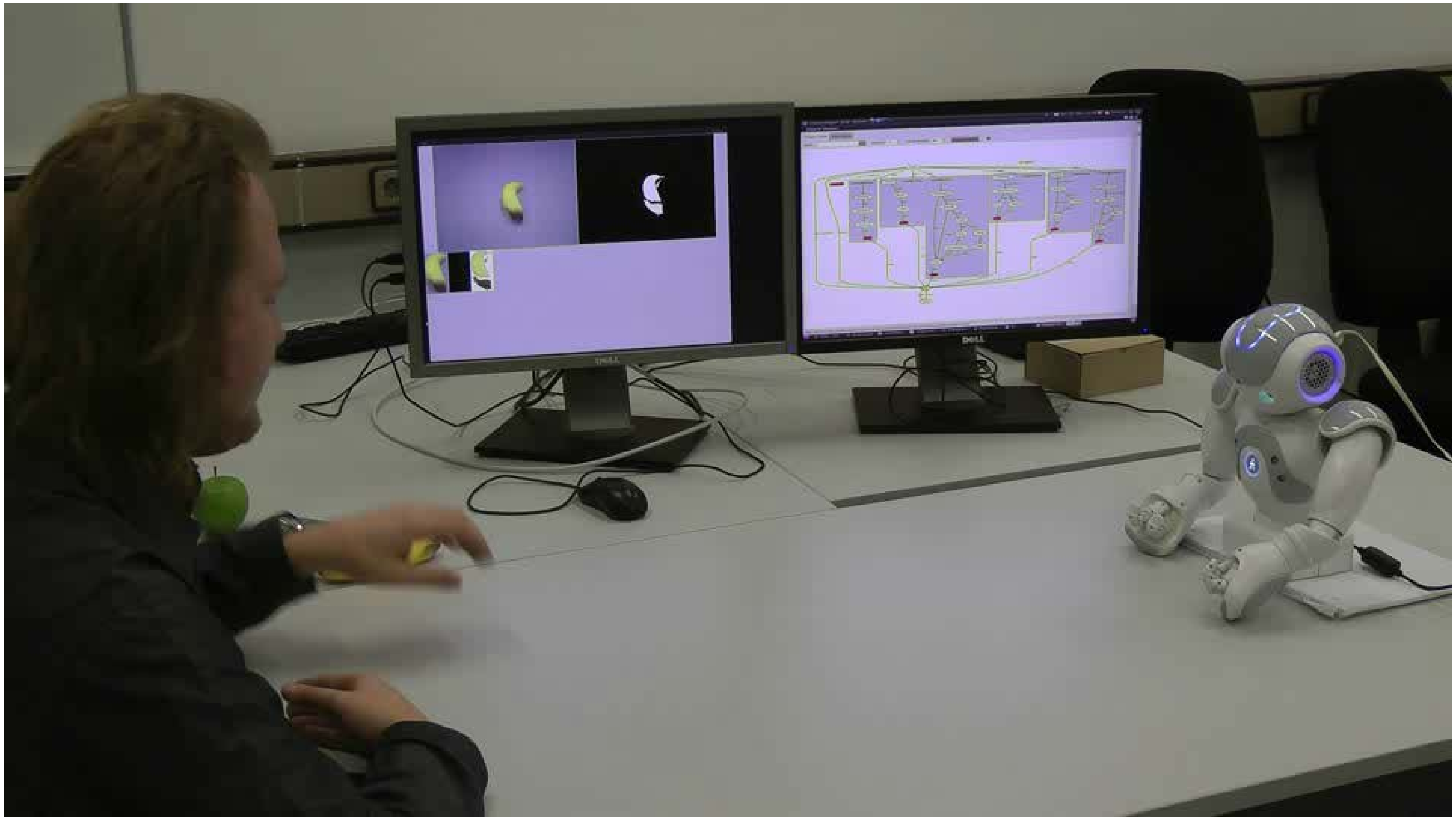
In: IEEE First International Conference on Cognitive Systems and Information Processing (CSIP 2012), in press (Springer). Beijing, December 15-17, 2012.

Object Learning with Natural Language in a Distributed Intelligent System – A Case Study of Human-Robot Interaction

Stefan Heinrich, Pascal Folleher, Peer Springstübe, Erik Strahl, Johannes Twiefel, Cornelius Weber, and Stefan Wermter

University of Hamburg, Department of Informatics, Knowledge Technology
Vogt-Kölln-Straße 30, D - 22527 Hamburg, Germany
{heinrich,6follehe,3springs,strahl}@informatik.uni-hamburg.de
{7twiefel,weber,wermter}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/WTM/

Student Project: Classifying Objects with MLPs



The robot perceives visual features of the objects and learns the objects' names

http://www.informatik.uni-hamburg.de/WTM/teaching/WiSe11_HumanRobotInteraction_Pj.shtml