Classification and Prediction 2
Perceptrons and MLPs

Sebastian Risi
IT University of Copehage

Group Project Proposals

- March 26th ➔ You are responsible for forming groups (3 or 4 people, not 1 or 2)

- https://docs.google.com/document/d/1VPTm3rbUIlq3DohIEcGXvw28QJzxiSn1FPDe6JcMnpo/edit?usp=sharing

Group Projects

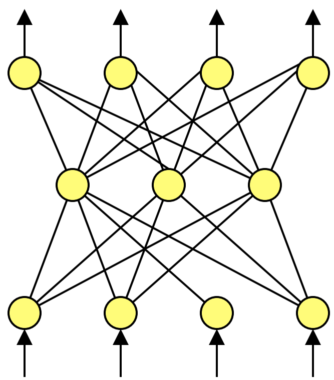
The report should be 6-8 pages (double column format0, excluding source code but including figures, and should be well written. The report should answer the following questions:

- What knowledge are you trying to extract from what data?
- What methods will you be using?
- Why did you choose this data set and this data?
- How do the algorithms work? (Only a brief overview necessary.)
- How did you extract, stor and manage the data?
- Which parameters were used with the algorithms? Did you do any tuning? Why/why not?
- How good are the results, and how valid? What do they really show? (Remember to compare with mode/mean prediction as a baseline, i.e. ZeroR)
- If applicable (in most cases it is), what is the societal impact and what are the privacy aspects of your work?
- Why did it work / did it not work?
- The following technical tasks should be undertaken:

- Preprocessing (except if obviously unnecessary)
- Descriptive statistics
- Appropriate visualization
- Appropriate validation of results
- You should use algorithms from at least two of the following groups (except if you have a very good reason to use something else, needs pre-approval from me):
  - Frequent pattern mining (association mining, sequence pattern mining etc.)
  - Clustering
  - Supervised learning (classification/prediction)
- There will be a 15-minute supervision slot for each group each week, where the group will receive supervision by one of us.

WHAT IS AN ARTIFICIAL NEURAL NETWORK?

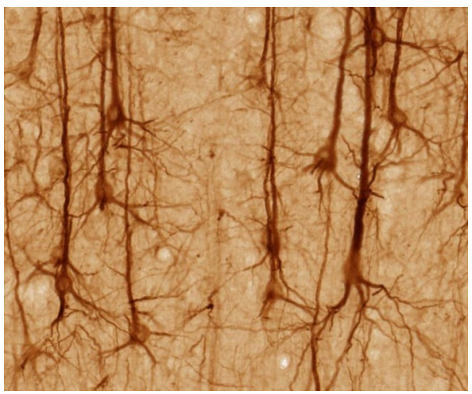
Artificial Neural Networks



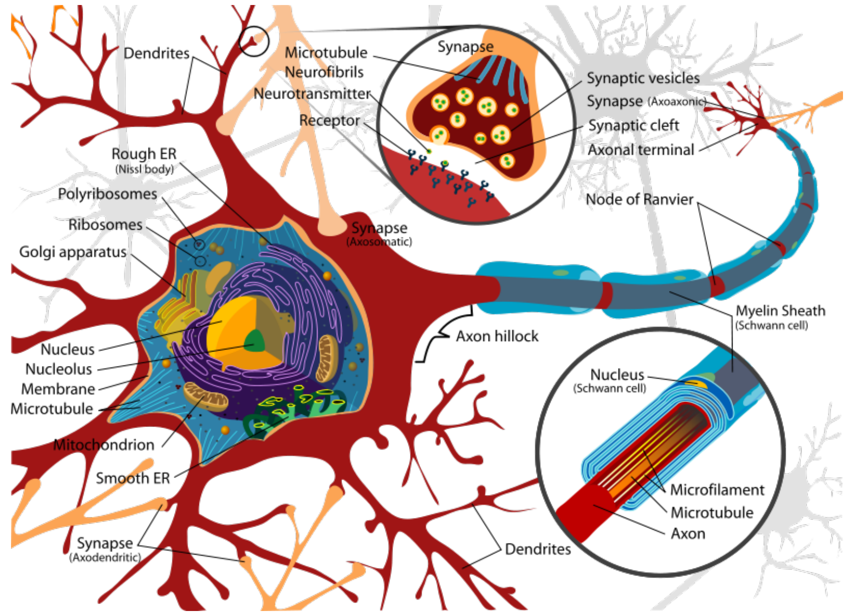
- Biological brains are made out of *neurons*
- Neurons send signals to each other
- Idea (*Connectionism*): Simulate neurons, attach them to each other, and create an artificial brain!

Neurons

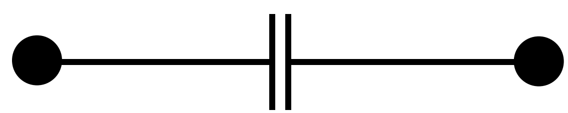
- A neuron is a cell in the brain that stores and transmits information



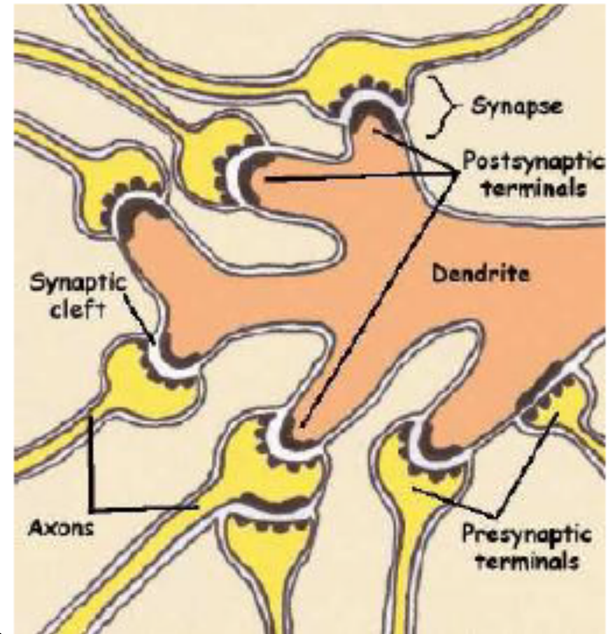
Cortical Neurons (UC Davis)



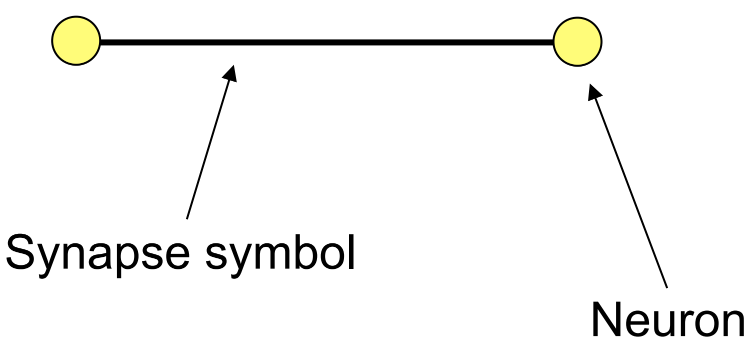
- Neurons connect through a *synapse*
  - Chemicals called neurotransmitters send information across the synapse



Networks of Neurons



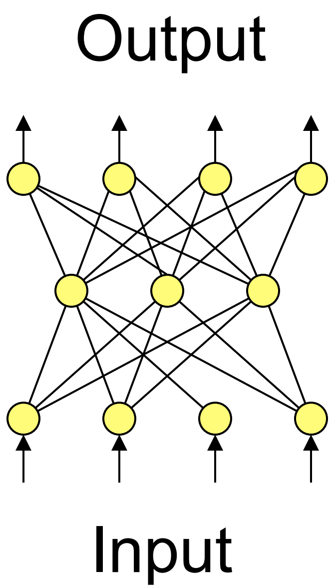
- Many neurons connecting to each other form a *neural network*
- More than one neuron can feed into another through a synapse
- The ANN symbol for a synapse is a connecting line

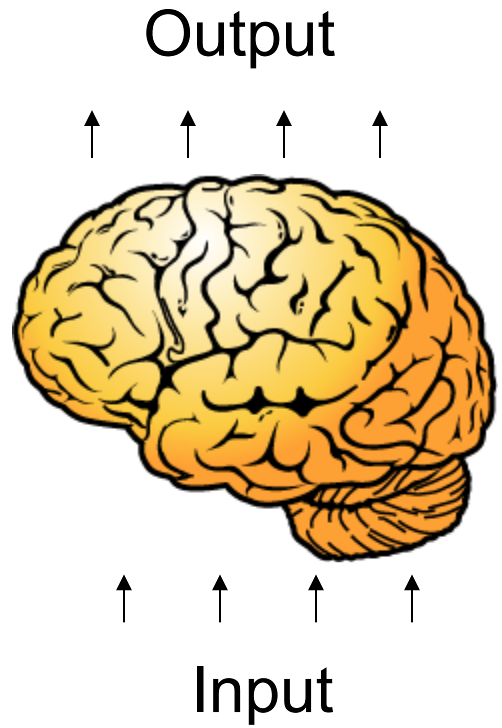


ANN

- What can they do?
  - Supervised Learning (needs data)
  - Classification/Prediction
- Some can do clustering (unsupervised)
  - Self-organizing map

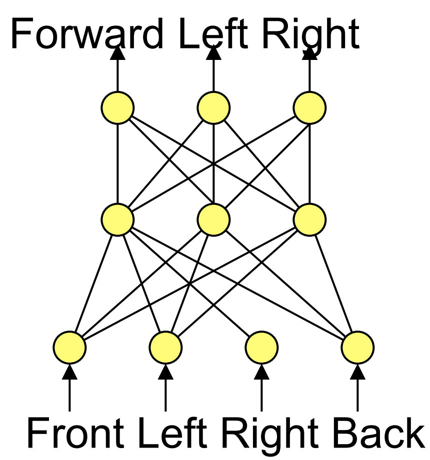
How Do ANNs Work?



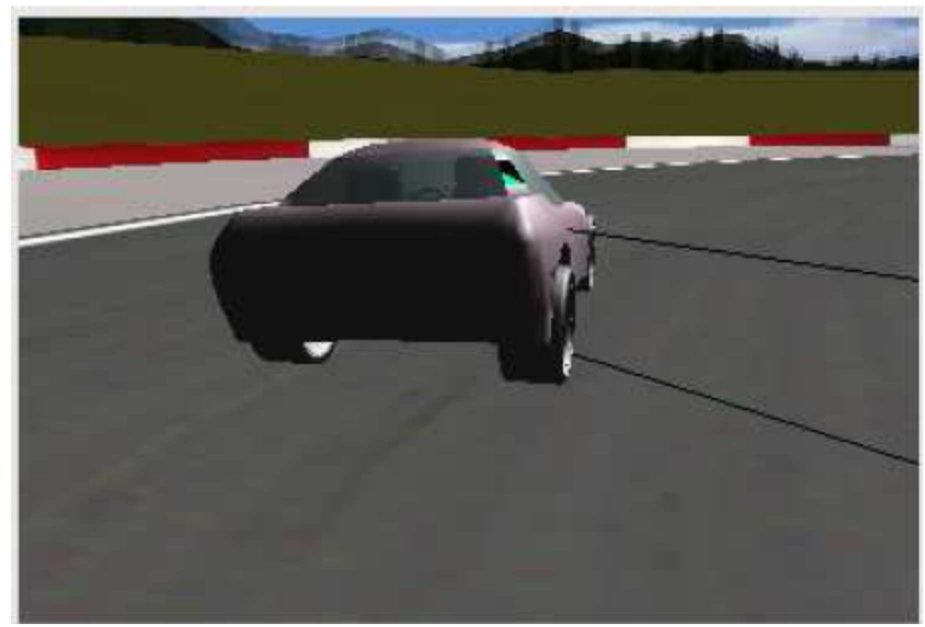


How do NNs Work?
Example

Outputs (effectors/controls)

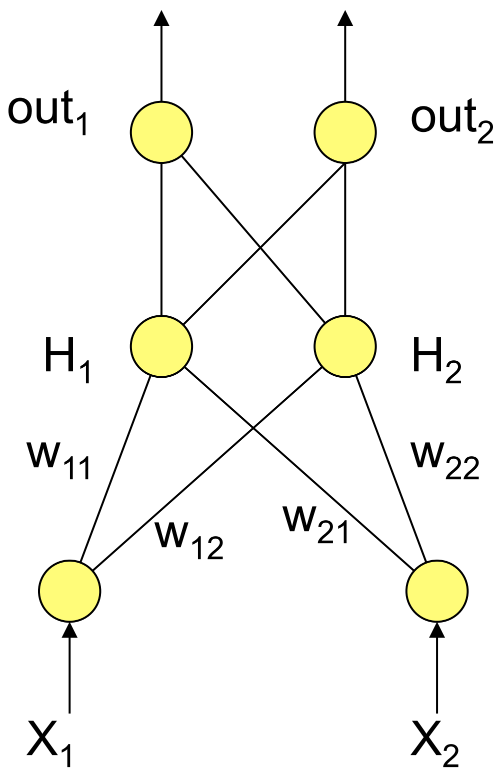


Inputs (Sensors)



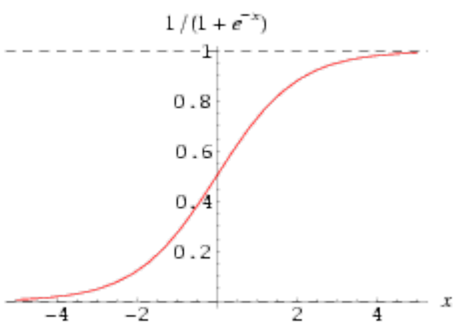
What Exactly Happens Inside the Network?

- *Network Activation*



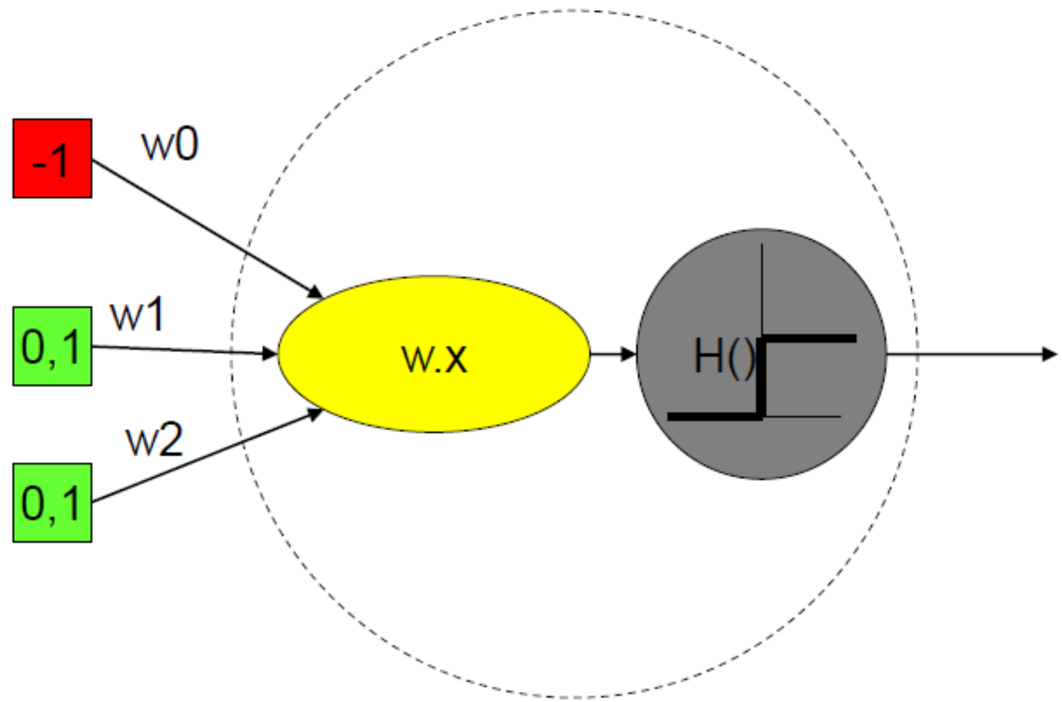
Neuron *j* activation:

$$H_j = \sigma\left(\sum_{i=1}^{n} x_i w_{ij}\right)$$



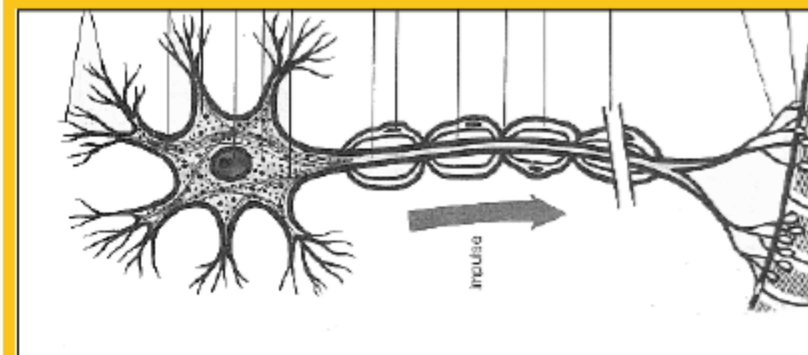
Perceptron (1943)

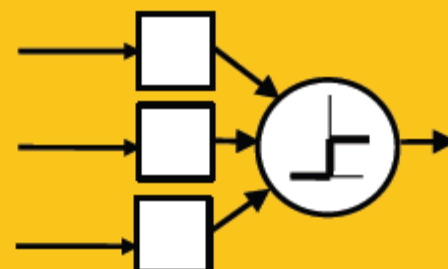
- McCulloch & Pitts‘ neuron model:
  - Inputs and Output are Binary
  - Activation function is Heavyside (step) function



Previously...

1943 1949 1958 1959 1962 1969





one layer
only solves
linear
problems

**learning algorithm
for an arbitrary
topology unavailable !**

Training Perceptron Algorithm

- Gradient-search algorithm
- Goal: minimize error (*E*) between actual (*a*) and desired/target (*d*) output by adjusting the weights (**w**)
- Do that by computing the partial derivative of the Error relative to each connection weight. Apply the Delta-rule to adjust the weights

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta x_i \left(d^{(p)} - a^{(p)} \right)$$

$$w_i \leftarrow w_i + \Delta w_i$$

Perceptron Training Algorithm

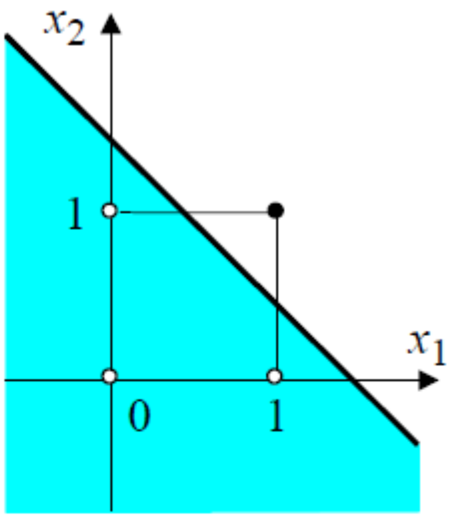
Algorithm's structure

- Given a set of input patterns $\left(x^{(p)}\right)$ and desired outputs: $\left(d^{(p)}\right)$
  1. Initialize Perceptron with random weights
  2. For each pattern p
     - Compute actual output $\left(a^{(p)}\right)$
     - Update all weights i = 1,... by Δw_i
  3. If no changes to the weights, then STOP
  4. Otherwise loop to step 2

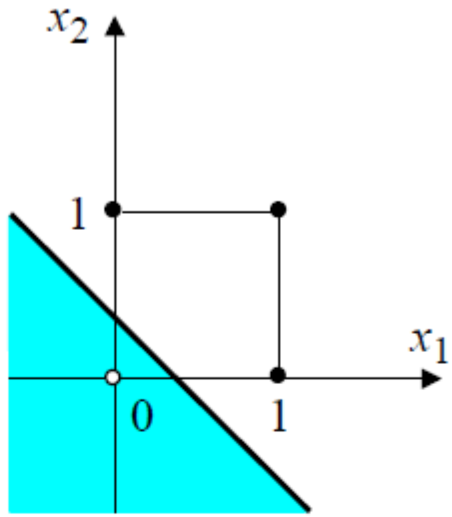
Perceptron Training Algorithm

- What can it learn? What can it not learn?
- Can we train networks with more than one layer?

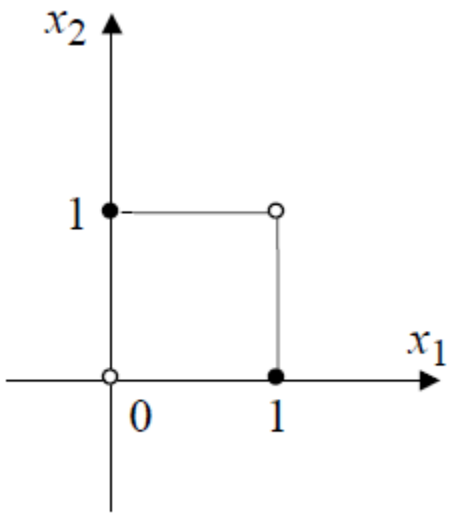
Perceptron (two inputs)



(a) *AND* $(x_1 \cap x_2)$

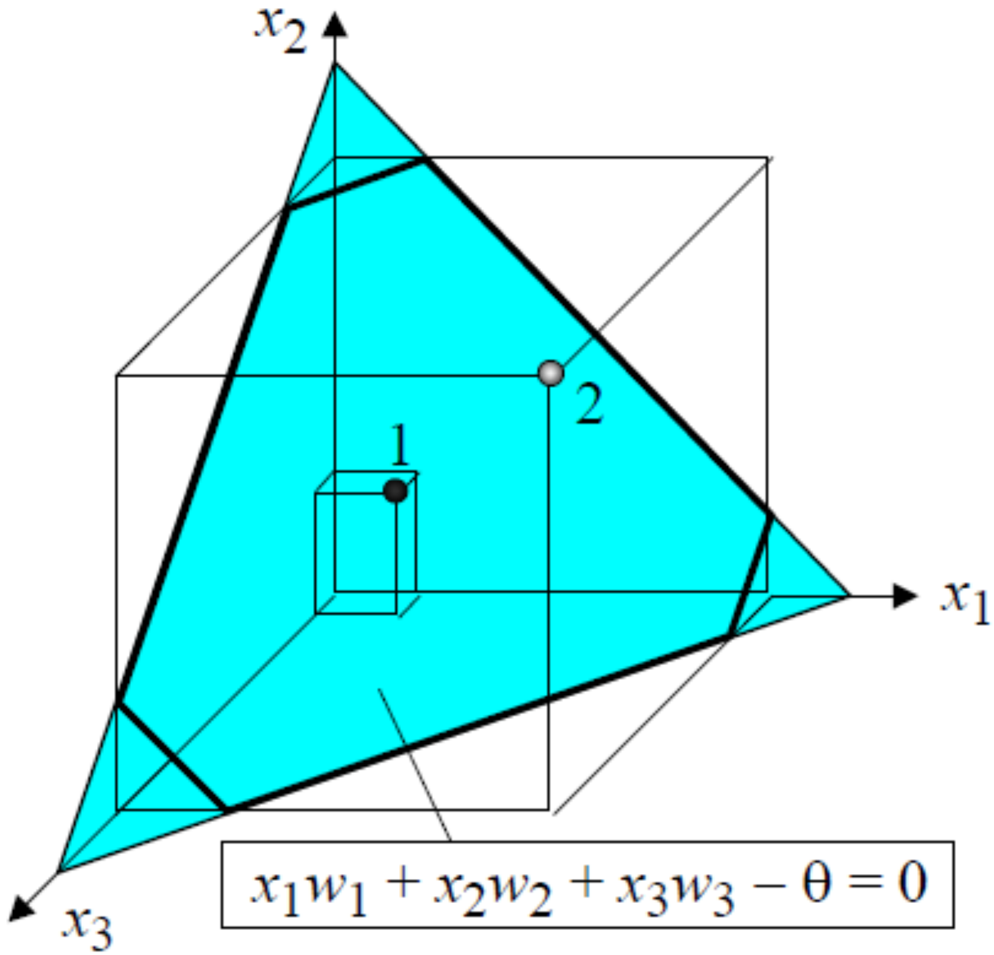


(b) *OR* $(x_1 \cup x_2)$



(c) *Exclusive-OR* $(x_1 \oplus x_2)$

Perceptron (three inputs)



(*b*) Three-input perceptron.

Example: Logical Operation

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1		

Example: Logical Operation

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

What can Perceptrons Learn

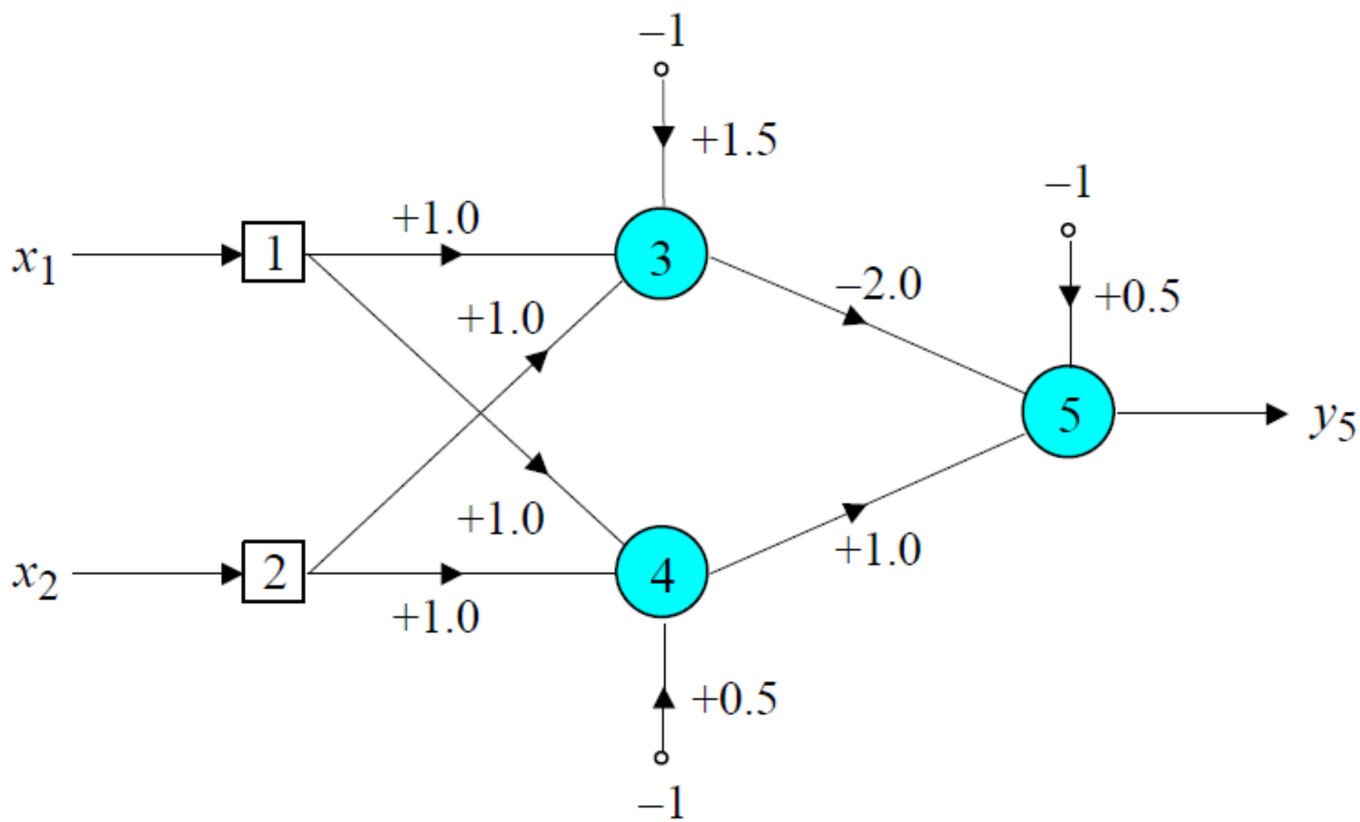
- Restrict to single layer
  - Linearly separable problems
- Multi-Layer → more complex classification
  - But no training algorithm... (Why?)

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta x_i \left(d^{(p)} - a^{(p)} \right)$$

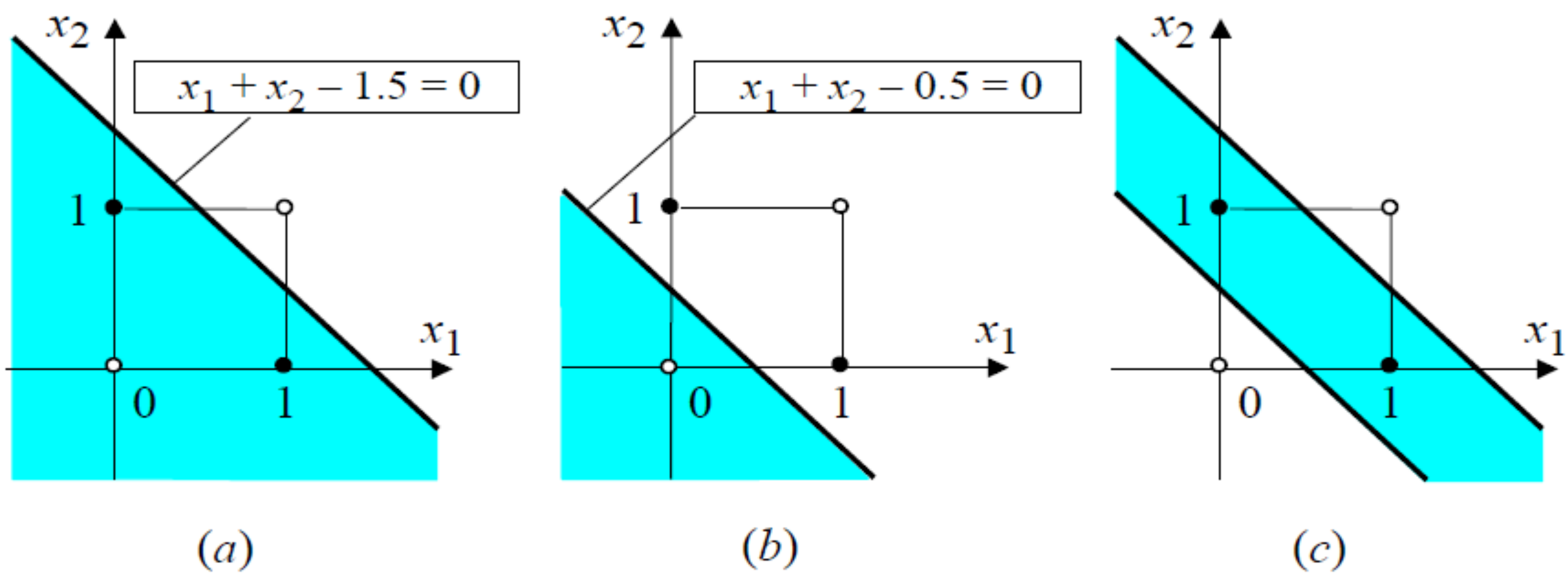
$$w_i \leftarrow w_i + \Delta w_i$$

XOR Problem Perceptron

- Not trained – Weights found ad/hoc
  - Activation Function is Heavyside/Step (not Sigmoid!)
  - What does this mean graphically?



XOR Problem – Decision Boundaries



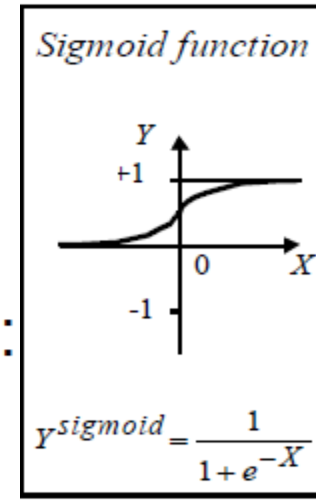
(a) Decision boundary constructed by hidden neuron 3;
(b) Decision boundary constructed by hidden neuron 4;
(c) Decision boundaries constructed by the complete Network (linear combination)

So what can we do?

- Change activation function:
  - Differentiable everywhere
  - Bounded Output
  - Monotonic, ideally
  - Easy to calculate derivative
- The sigmoid (Fermi function, logistic function):

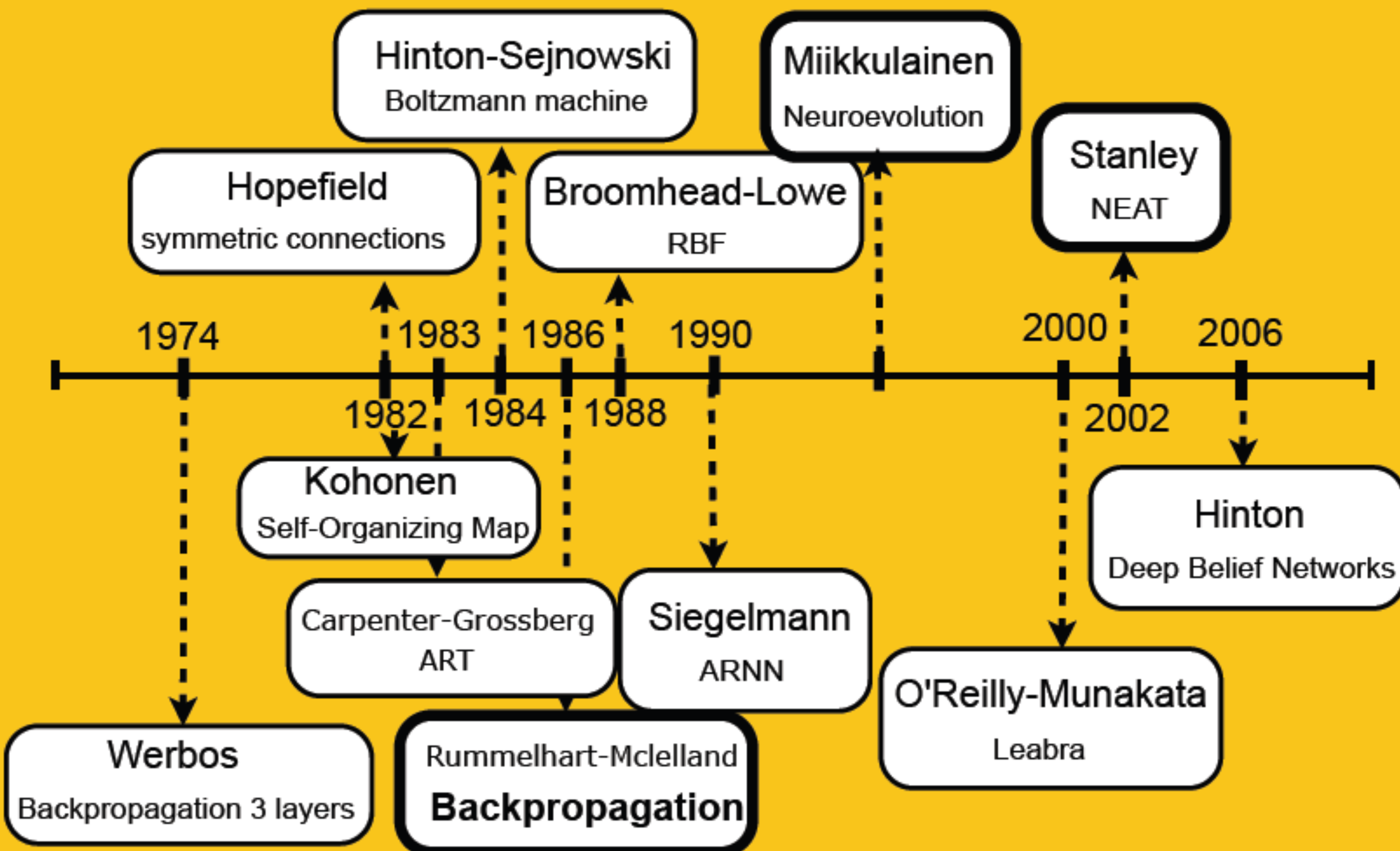
$$g(x) = \frac{1}{1+e^{-Dx}}$$

- With such an activation function, multi-layer training is possible



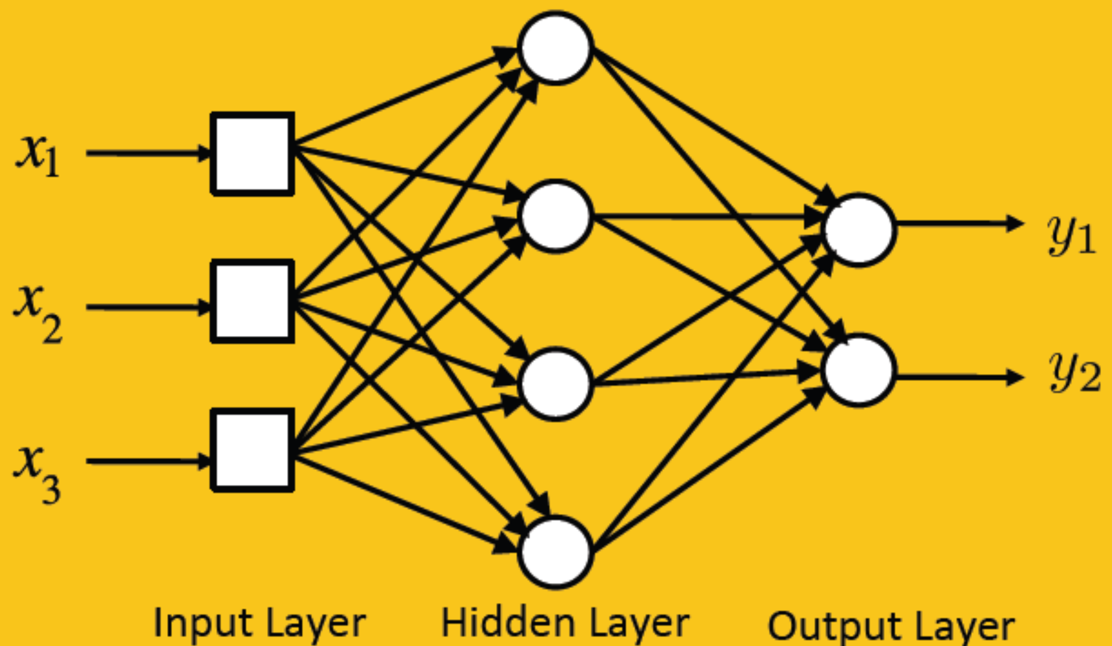
MULTI-LAYER PERCEPTRONS (MLPS)

History from 1970's to today



Multi-Layer Perceptrons (MLPs)

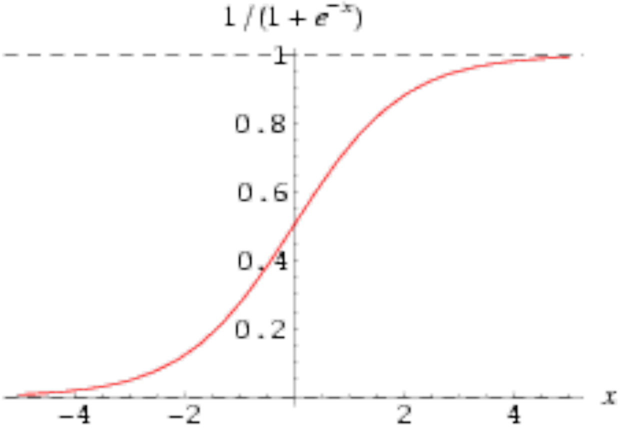
- Layered feed-forward **fully** connected network topology
- Single or multiple output
- Neurons with *smooth* activation function



Trainable using Backpropagation

Backpropagation

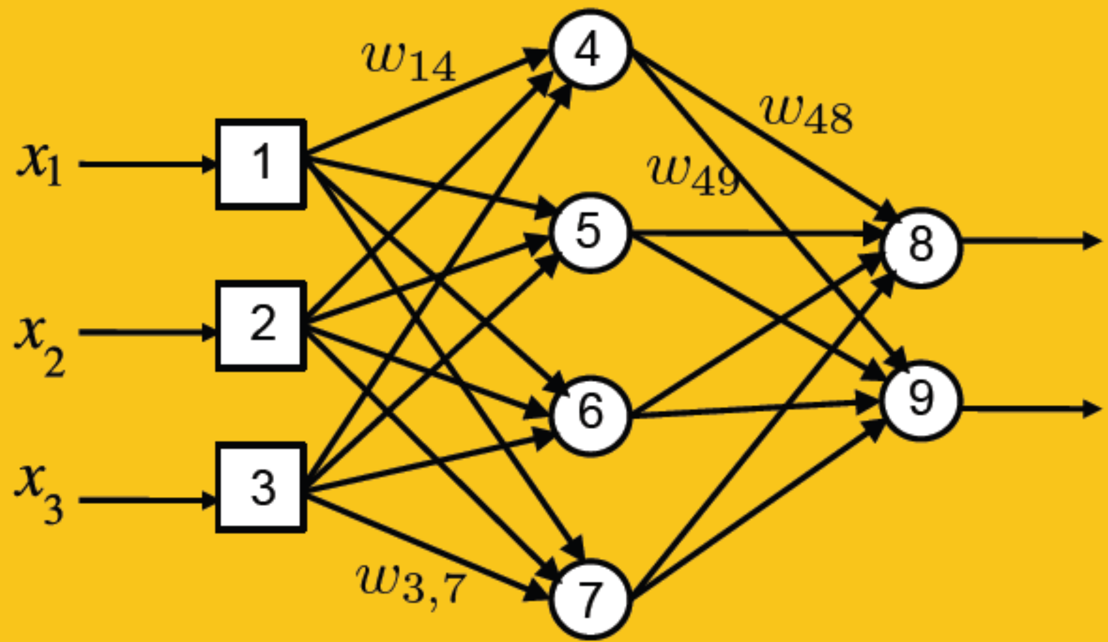
- Designed for at least one hidden layer
- First, activation propagates to outputs
- Then, errors are computed and assigned
- Finally, weights are updated
- Sigmoid is a common activation function



MLP Forward Operation

- Label and order units (neurons)
- Assume w_{ij} is weight from i to j unit
- Apply an input pattern $\vec{x}^{(p)}$
- For each unit j
  - Compute $s_j = \sum_i w_{ij} a_i$
  - Bias values included as w_{0j} as usual
  - Apply activation function $a_j = g(s_j)$

MLP Forward Operation



$$S_4 = w_{14}a_1 + w_{24}a_2 + w_{34}a_3 + w_{04} = w_{14}x_1 + w_{24}x_2 + w_{34}x_3 + w_{04}$$

$$a_4 = g(S_4) = \frac{1}{1+e^{-S_4}} = \frac{1}{1+e^{-(w_{14}x_1 + w_{24}x_2 + w_{34}a_3 + w_{04})}}$$

$$a_8 = g(S_8) = \frac{1}{1+e^{-S_8}} = \frac{1}{1+e^{-(w_{48}\boldsymbol{a_4} + w_{58}a_5 + w_{68}a_6 + w_{78}a_7 + w_{08})}}$$

MLP Backward Operation

- Choose a (differentiable) error function:
  Sum of Squared Deviations

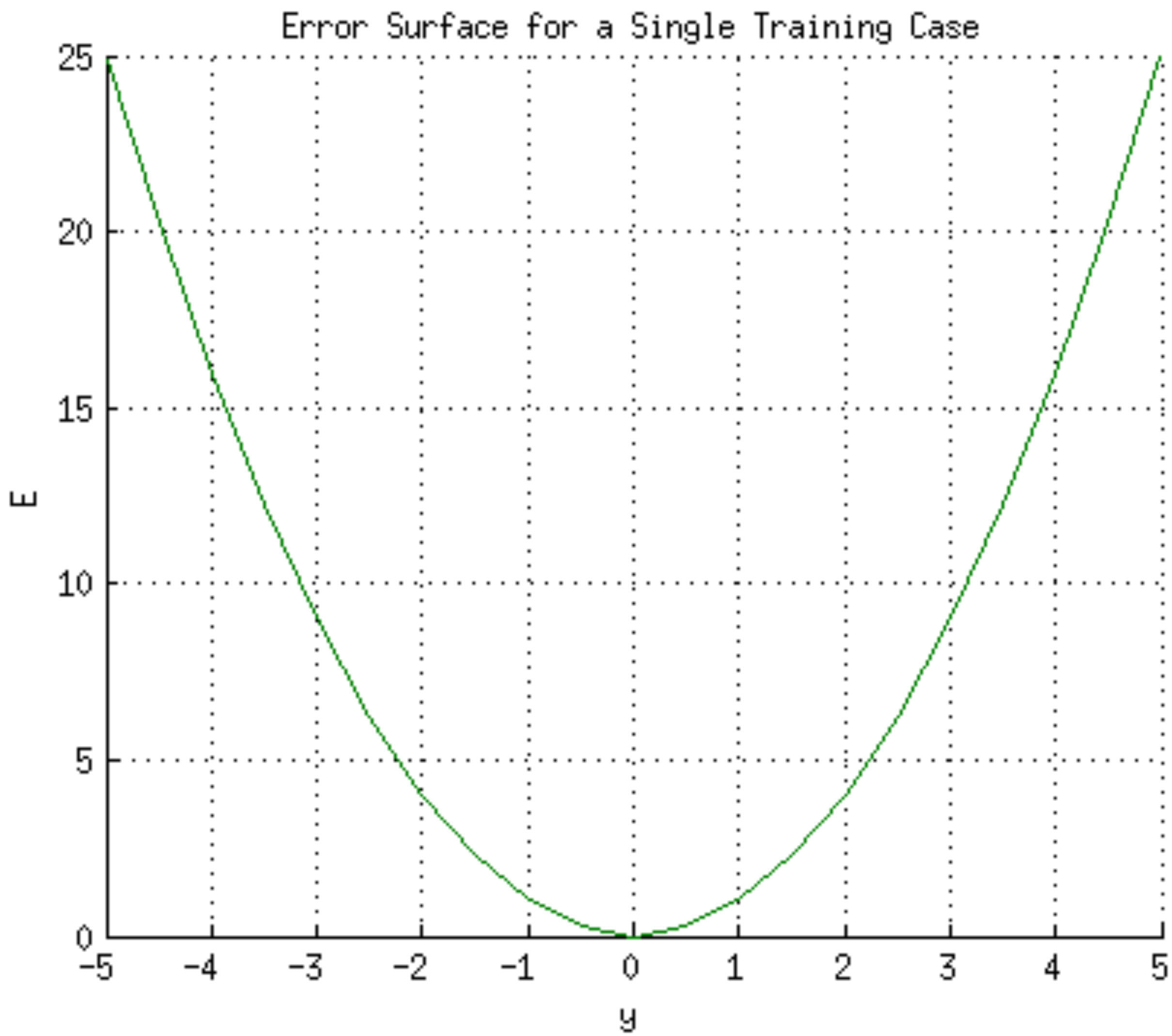
  For each output: $E_k = \frac{1}{2}(d_k - a_k)^2$
  (and one input pattern)

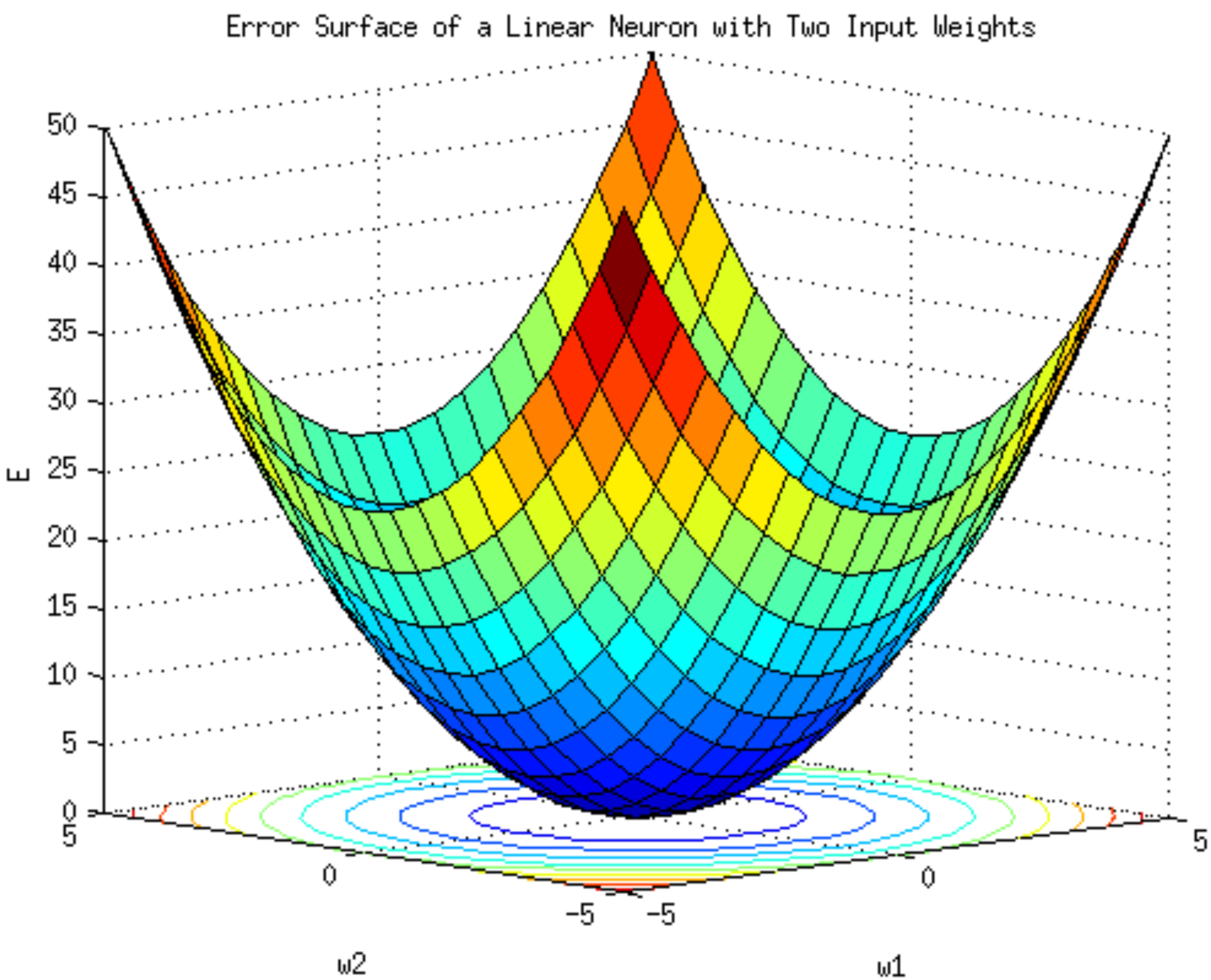
  For all outputs: $E = \frac{1}{2}\sum_{\forall k \in out}(d_k - a_k)^2$
  (and one input pattern)

- Minimize it…

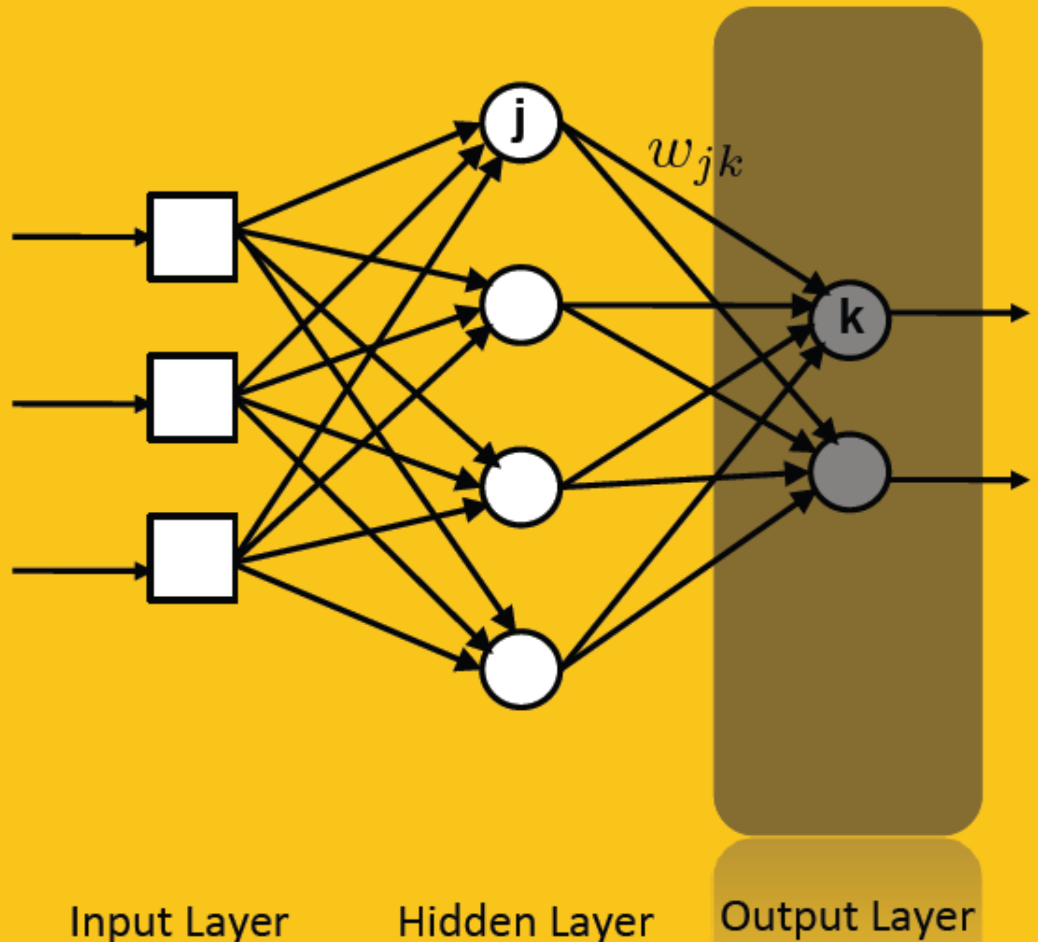
- Error is an implicit function of w_{ij}

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

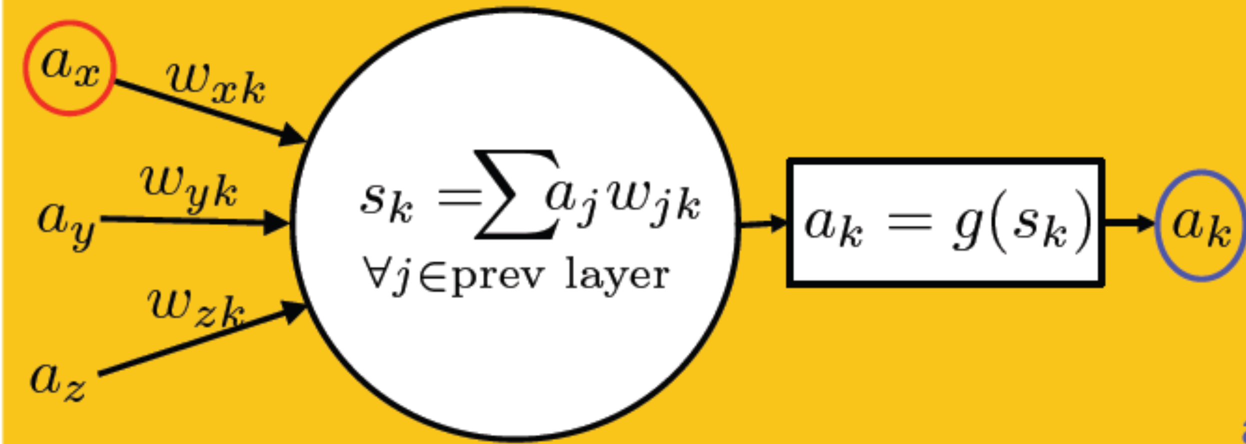

Error Surface for a Single Training Case
25
20
15
10
5
0
E
-5
-4
-3
-2
-1
0
1
2
3
4
5
y


Error Surface of a Linear Neuron with Two Input Weights
E
50
45
40
35
30
25
20
15
10
5
0
5
0
-5
-5
0
5
w2
w1

Output layer



Output layer



$$E = \frac{1}{2} \sum_{\forall k \in out} (d_k - a_k)^2$$

output (calculated at the neuron)

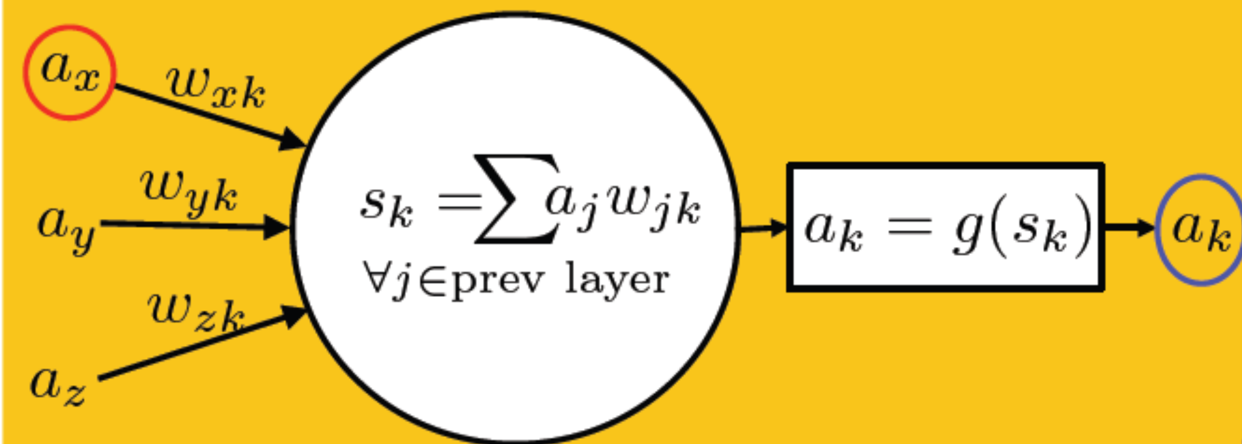
$$\boxed{\frac{\partial E}{\partial w_{jk}}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{jk}} \boxed{= -(d_k - a_k) \cdot a_k \cdot (1 - a_k) \cdot a_j}$$

error (backpropagated)

input

$$\frac{\partial s_k}{\partial w_{jk}} = a_j \qquad \frac{\partial a_k}{\partial s_k} = a_k \cdot (1 - a_k) \qquad \frac{\partial E}{\partial a_k} = -(d_k - a_k)$$

Output layer



$$E = \frac{1}{2} \sum_{\forall k \in out} (d_k - a_k)^2$$

output
(calculated
at the neuron)

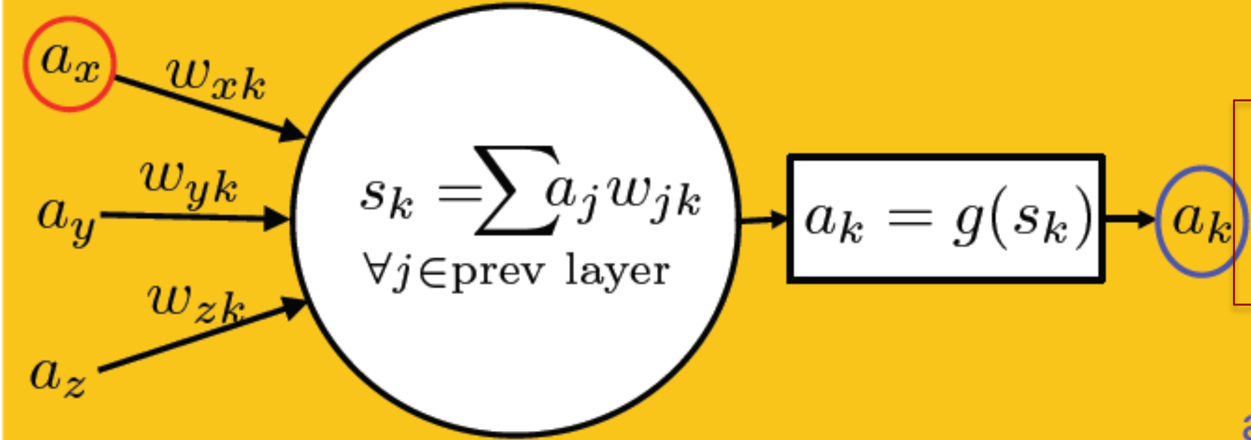
$$\boxed{\frac{\partial E}{\partial w_{jk}}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{jk}} \boxed{= -(d_k - a_k) \cdot a_k \cdot (1 - a_k) \cdot a_j}$$

error
(backpropagated)

input

$$\frac{\partial s_k}{\partial w_{jk}} = a_j \qquad \frac{\partial a_k}{\partial s_k} = a_k \cdot (1 - a_k) \qquad \frac{\partial E}{\partial a_k} = -(d_k - a_k)$$

Output layer



$$E = \frac{1}{2} \sum_{\forall k \in out} (d_k - a_k)^2$$

output
(calculated
at the neuron)

$$\boxed{\frac{\partial E}{\partial w_{jk}}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{jk}} \boxed{= -(d_k - a_k) \cdot a_k \cdot (1 - a_k) \cdot a_j}$$

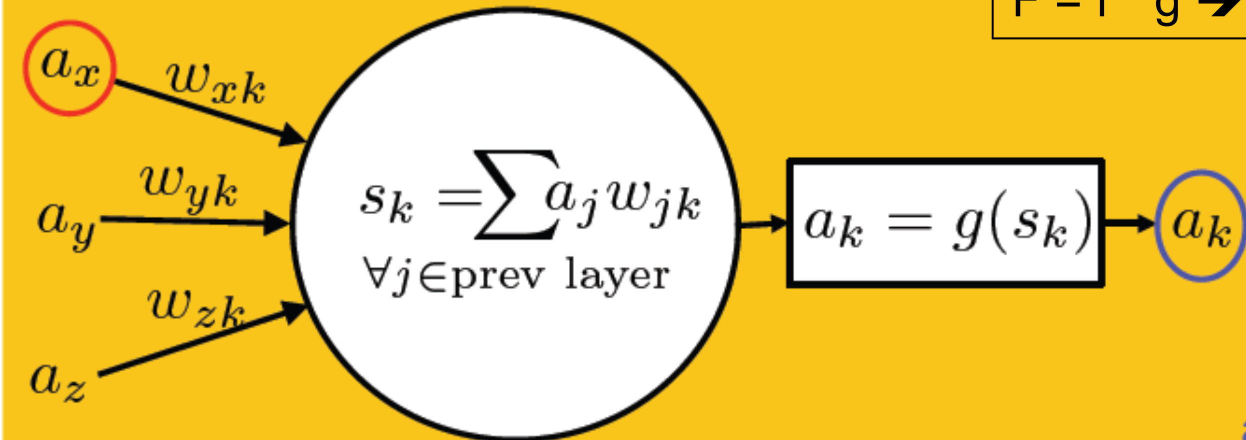
error
(backpropagated)

input

$$\frac{\partial s_k}{\partial w_{jk}} = a_j \qquad \frac{\partial a_k}{\partial s_k} = a_k \cdot (1 - a_k) \qquad \frac{\partial E}{\partial a_k} = -(d_k - a_k)$$

Output layer

Remember the Chain Rule?
F = f * g ➔ F'(x)=f'(g(x))g'(x)



$$E = \frac{1}{2} \sum_{\forall k \in out} (d_k - a_k)^2$$

output
(calculated
at the neuron)

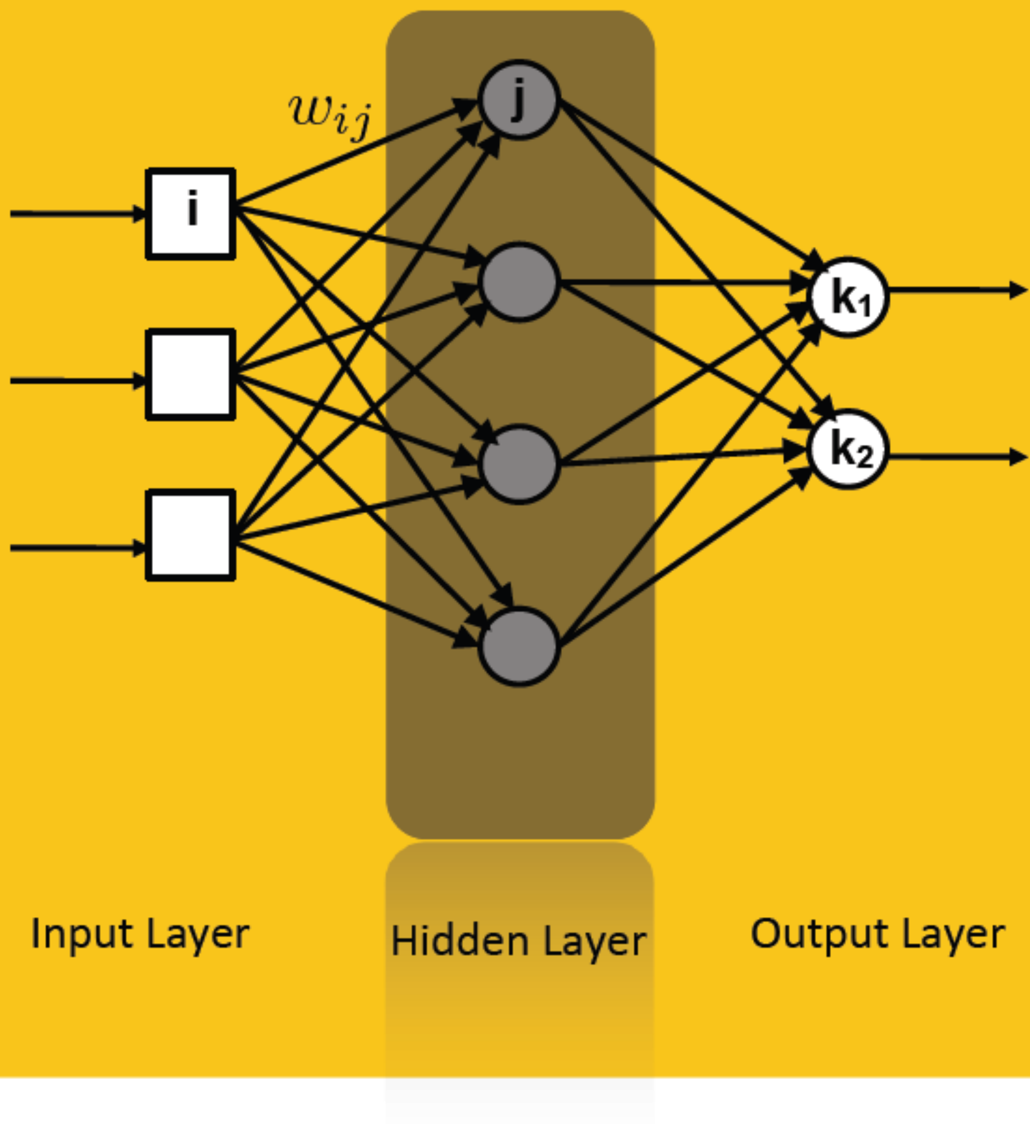
$$\boxed{\frac{\partial E}{\partial w_{jk}}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial w_{jk}} \boxed{= -(d_k - a_k) \cdot a_k \cdot (1 - a_k) \cdot a_j}$$

error
(backpropagated)

input

$$\frac{\partial s_k}{\partial w_{jk}} = a_j \qquad \frac{\partial a_k}{\partial s_k} = a_k \cdot (1 - a_k) \qquad \frac{\partial E}{\partial a_k} = -(d_k - a_k)$$

Hidden layer



Hidden layer

Partial derivative of the error for hidden neurons:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_j} \cdot \frac{\partial a_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{ij}} = \left(\sum_{k \in out} \left(\frac{\partial E_k}{\partial a_k} \cdot \frac{\partial a_k}{\partial s_k} \cdot \frac{\partial s_k}{\partial a_j} \right) \right) \cdot \frac{\partial a_j}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{ij}}$$

$$\frac{\partial s_j}{\partial w_{ij}} = a_i \qquad \frac{\partial a_j}{\partial s_j} = a_j \cdot (1 - a_j)$$

$$\frac{\partial s_k}{\partial a_j} = w_{jk} \qquad \frac{\partial a_k}{\partial s_k} = a_k \cdot (1 - a_k) \qquad \frac{\partial E_k}{\partial a_k} = -(d_k - a_k)$$

$$\boxed{\frac{\partial E}{\partial w_{ij}} = \left(\sum_{k \in out} \left(-(d_k - a_k) \cdot a_k(1 - a_k) \cdot w_{jk} \right) \right) \cdot a_j(1 - a_j) \cdot a_i}$$

Hidden layer

backpropagated error output input

Output layer update

$$\frac{\partial E}{\partial w_{jk}} = -(d_k - a_k) \cdot a_k \cdot (1 - a_k) \cdot a_j$$

backpropagated error

error next layer

connection weights to next layer

output (calculated at the neuron)

input

Hidden layer update

$$\frac{\partial E}{\partial w_{ij}} = \Big(\sum_{k \in out} \big(-(d_k - a_k) \cdot a_k(1 - a_k)\big) \cdot w_{jk} \Big) \cdot a_j(1 - a_j) \cdot a_i$$

What you need to know to implement it

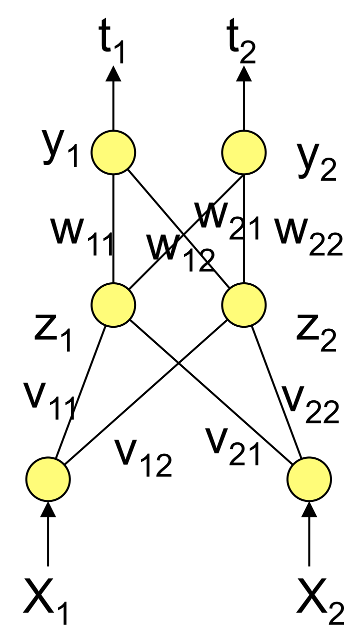
Backpropagation Algorithm

(note slightly different notation)

1) Initialize weights
2) While stopping condition is false, for each training pair
   1) Compute outputs by forward activation
   2) Backpropagate error:
      1) For each output unit, error $\delta_k = (t_k - y_k) f'(yin_k)$ (*target minus output times slope*) $y_k(1-y_k)$
      2) Weight correction $\Delta w_{jk} = \alpha \delta_k z_j$ (*Learning rate times error times hidden output*)
      3) Send error back to hidden units
      4) Calculate error contribution for each hidden unit:

$$\delta_j = \left(\sum_{k=1}^{m} \delta_k w_{jk} \right) f'(zin_j)$$

      5) Weight correction $\Delta v_{ij} = \alpha \delta_j x_i$
3) Adjust weights by adding weight corrections



x' s are inputs, z' s are hidden units, y' s are outputs

t' s are targets, v' s are layer 1 weights, w' s are layer 2 weights

Back-Propagation: Parameters & Variants

- Stopping condition
  - after a number of epochs or after error below a given threshold
- Batch or non-batch
  - weight update after each pattern or after all patterns
- Activation function and topology of the network
- Learning rate

Example Backprop Applications

- Learn mouse gesture recognition
- Learn to drive by observation
- Learn to control anything by observation
- Learn to diagnose medical conditions based on past examples
- For games: Learn to control the NPC by example

Classic Applications

- Anything with a set of examples and known targets
- XOR
- Character recognition
- NETtalk: reading English aloud
- Failure prediction
- (Disadvantages: trapped in local optima)

What can MLPs represent?

- Any continuous real-valued functions can be approximated using a neural network **to any desired degree of accuracy** with **a single hidden layer of logistic sigmoid activation functions**.
  [Kolmogorov (1957); Cybenko (1989); Hornik (1989)]

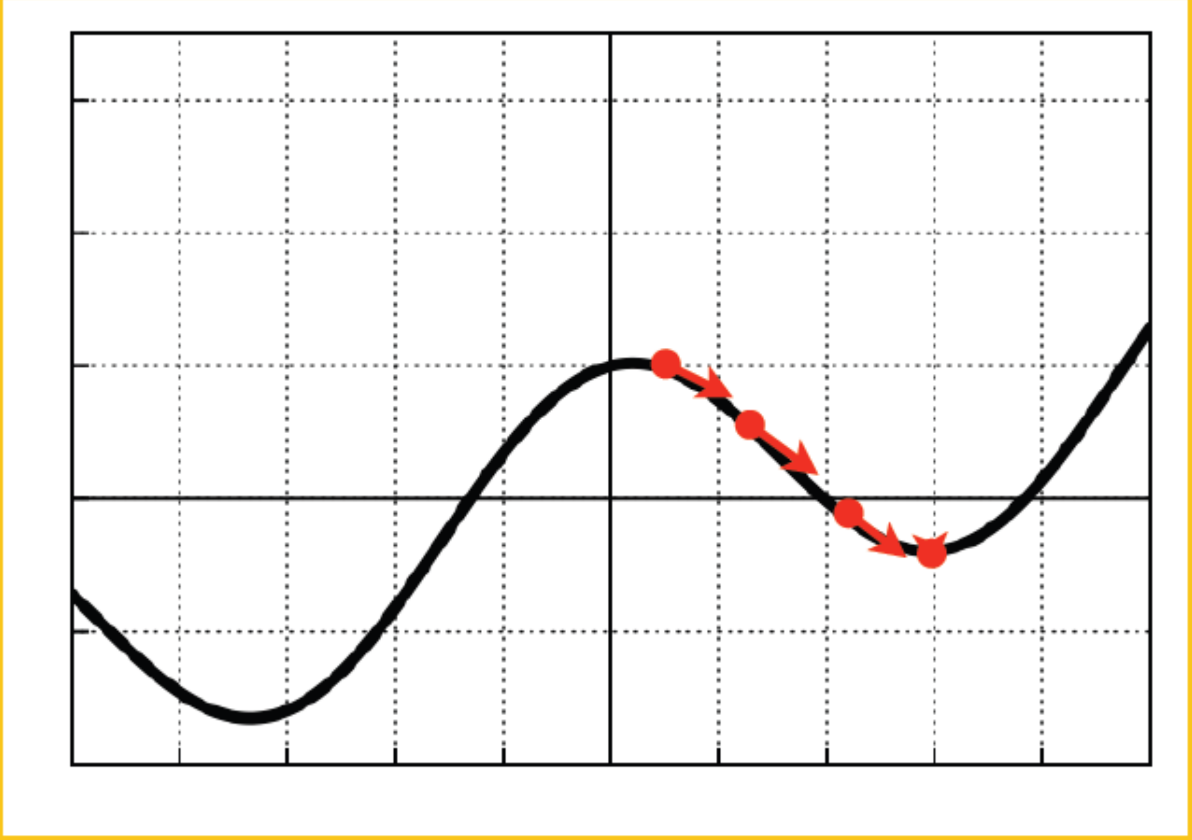
- But how to choose the topology?
- Can gradient descent find the correct weights?
- Training is done, have I found the function I was looking for?

Topology

- The size of the network should match the complexity of problem
- In theory, one hidden layer would be enough for any problem
- In practice, more layers may lead to less neurons
- Basic algorithms will rarely train correctly more than three layers

Gradient Descent Problems

- Problem: Local minima
- ~~Solution~~ Patch: several experiments with different initial weights



Generalization

- Training finished with error equal to zero
  - perfect result?
  - perfectly wrong?
- Another set of data is required to evaluate whether the ANN has learned the underlying function or simply memorized the training samples (***overfitting***)

- Divide your data in training and testing set

Reducing Overfitting: Early stopping

- Divide data into three sets:
  - Training
  - Validation
  - Test
- Train model of training set
- Stop when the error on the validation set increases
- Evaluate accuracy of model on test set

ANNs Pros & Cons

- **Ideal for well-defined problems**
- **Simple to implement**
- **Wide variety of training algorithms to choose from!**
- **Work well with noise**
- **Work well with different input and output data types**
- **Can be used for effective real-time learning**
- **Adaptive**
- **Universal Approximators**

- **Expressiveness** (Black Box)
- **Experimentation effort!** (pre and post data processing, training algorithm parameters, appropriate algorithm)

Thank you!

Questions?