

Learning

- Learning agents
- Inductive learning
 - Different Learning Scenarios
 - Evaluation
- Neural Networks
 - Perceptrons
 - Multilayer Perceptrons
- Reinforcement Learning
 - Temporal Differences
 - Q-Learning
 - SARSA

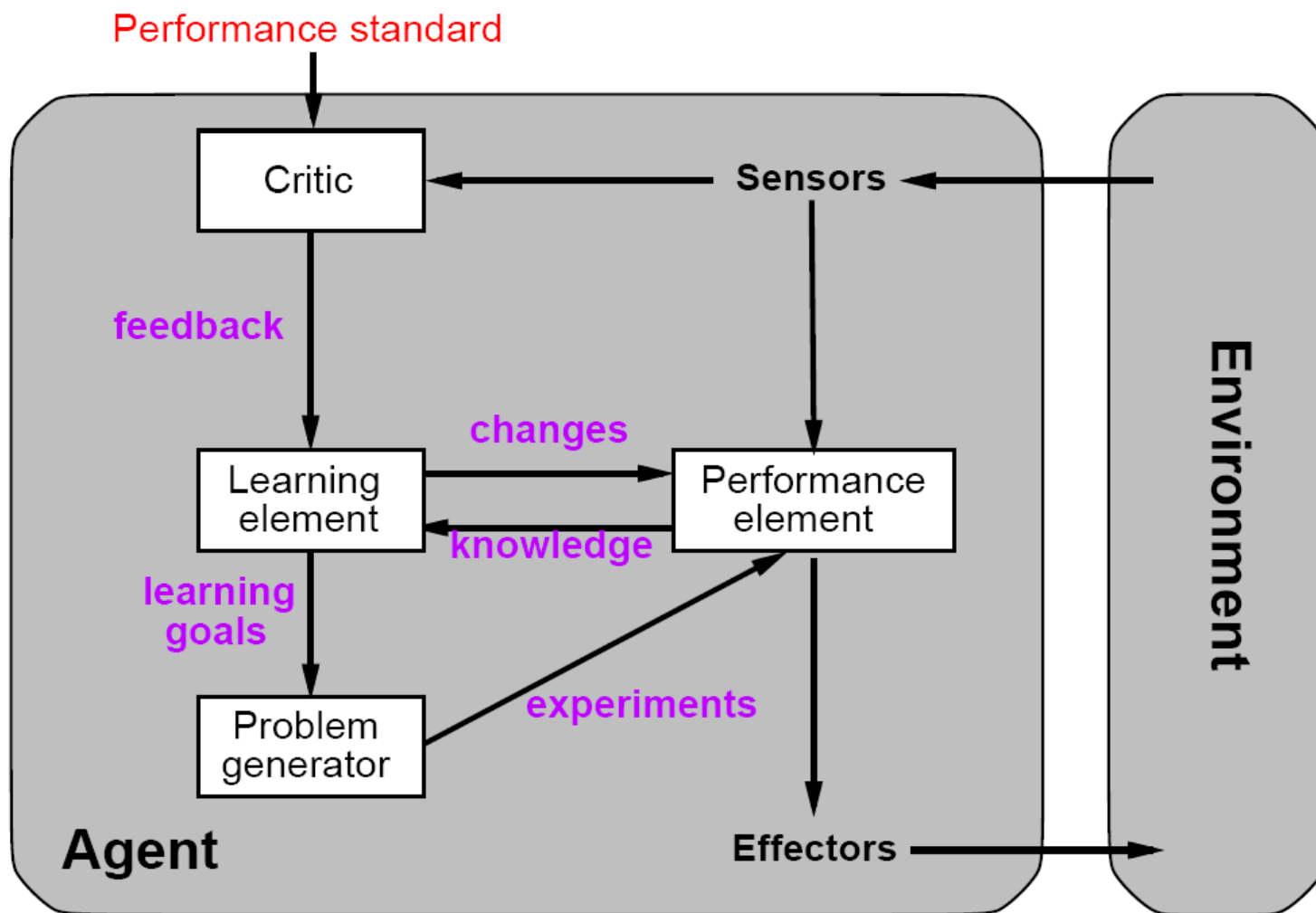
Material from
Russell & Norvig,
chapters 18.1,
18.2, 20.5 and 21

Slides based on Slides
by Russell/Norvig,
Ronald Williams,
and Torsten Reil

Learning

- Learning is essential for **unknown environments**,
 - i.e., when designer lacks omniscience
- Learning is useful as a **system construction method**,
 - i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to **improve performance**

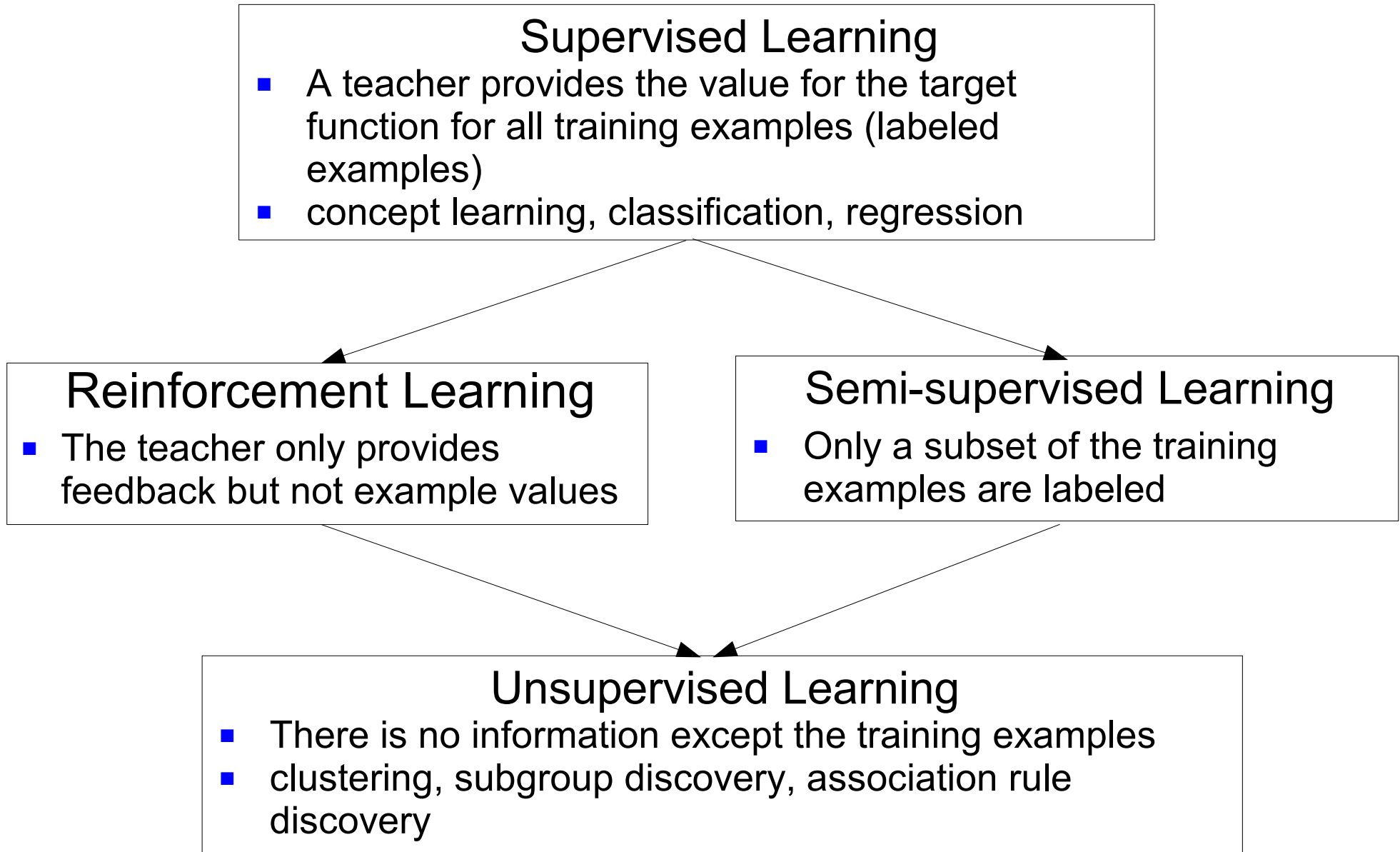
Learning Agents



Learning Element

- Design of a learning element is affected by
 - Which components of the performance element are to be learned
 - What feedback is available to learn these components
 - What representation is used for the components
- Type of feedback:
 - Supervised learning:
 - correct answers for each example
 - Unsupervised learning:
 - correct answers not given
 - Reinforcement learning:
 - occasional rewards for good actions

Different Learning Scenarios



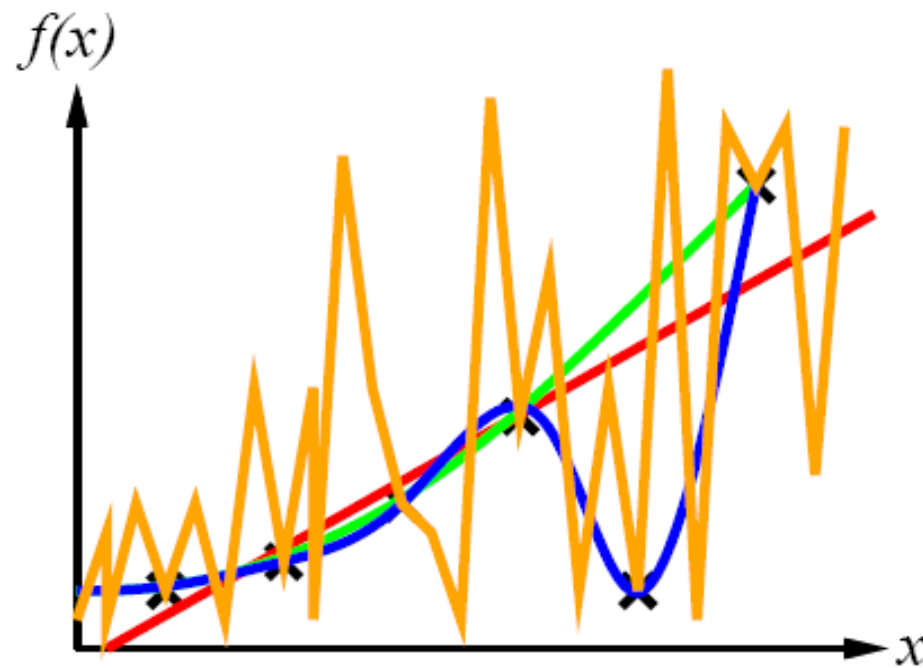
Inductive Learning

Simplest form: learn a function from examples

- f is the **target function**
- An **example** is a pair $(x, f(x))$
- Problem: find a **hypothesis** h
 - given a **training set** of examples
 - such that $h \approx f$
 - on *all* examples
 - i.e. the hypothesis must **generalize** from the training examples
- This is a highly simplified model of real learning:
 - Ignores prior knowledge
 - Assumes examples are given

Inductive Learning Method

- Construct/adjust h to agree with f on training set
 - h is **consistent** if it agrees with f on all examples
- Example:
 - curve fitting

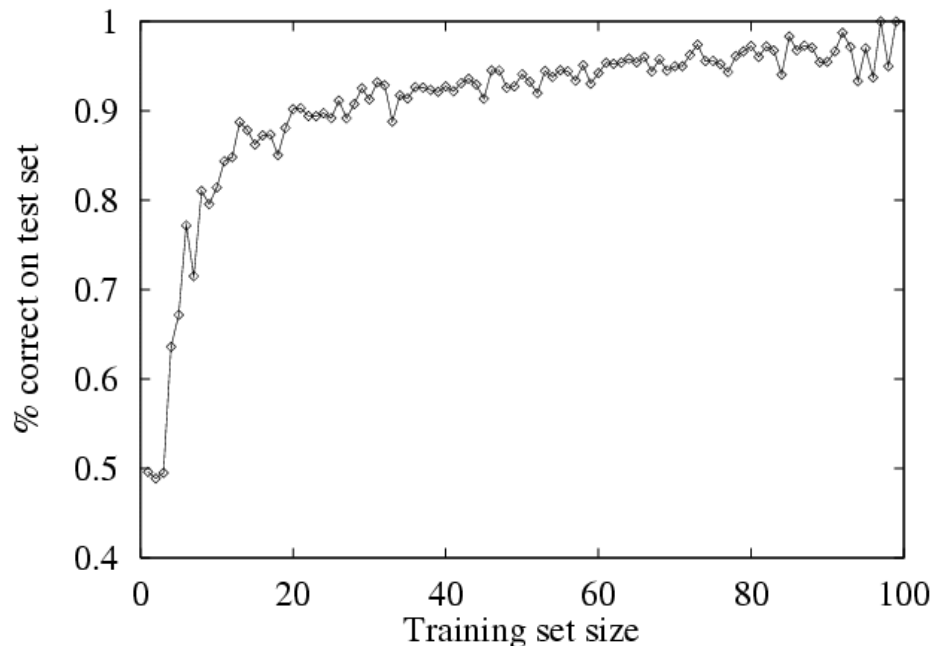


- **Ockham's Razor**
 - The best explanation is the simplest explanation that fits the data
- **Overfitting Avoidance**
 - maximize a combination of consistency and simplicity

Performance Measurement

- How do we know that $h \approx f$?
 - Use theorems of computational/statistical learning theory
 - Or try h on a new **test set** of examples where f is known (use **same distribution** over example space as training set)

Learning curve = % correct on test set over training set size



What are Neural Networks?

- Models of the brain and nervous system
- Highly parallel
 - Process information much more like the brain than a serial computer
- Learning

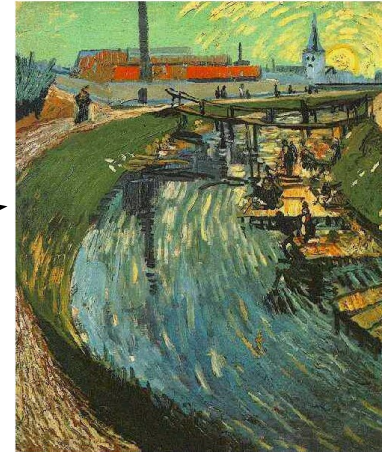
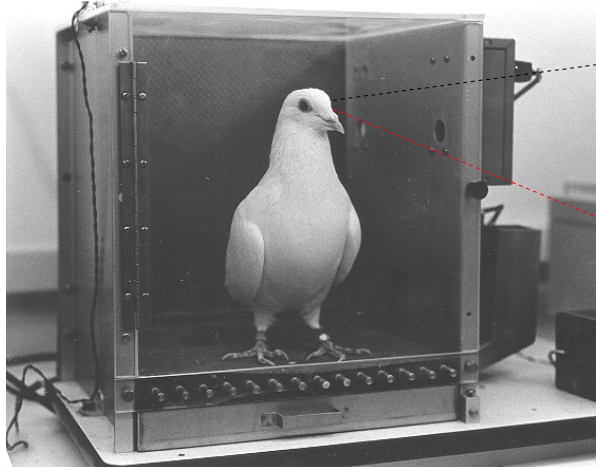
- Very simple principles
- Very complex behaviours

- Applications
 - As powerful problem solvers
 - As biological models

Pigeons as Art Experts

Famous experiment (Watanabe *et al.* 1995, 2001)

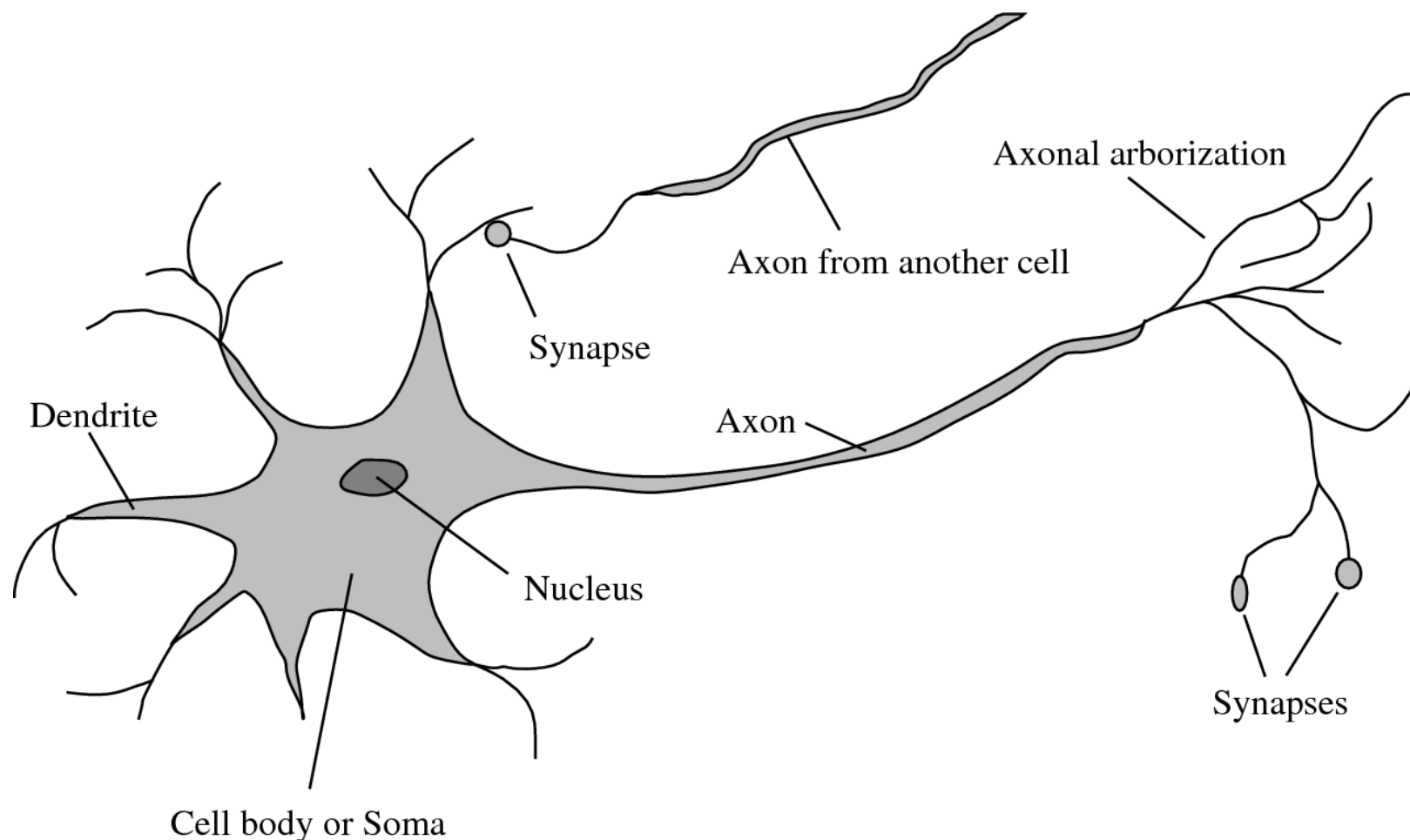
- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist



Results

- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy
 - when presented with pictures they had been trained on
- Discrimination still 85% successful for previously unseen paintings of the artists
- Pigeons do not simply memorise the pictures
- They can extract and recognise patterns (the 'style')
- They generalise from the already seen to make predictions
- This is what neural networks (biological and artificial) are good at (unlike conventional computer)

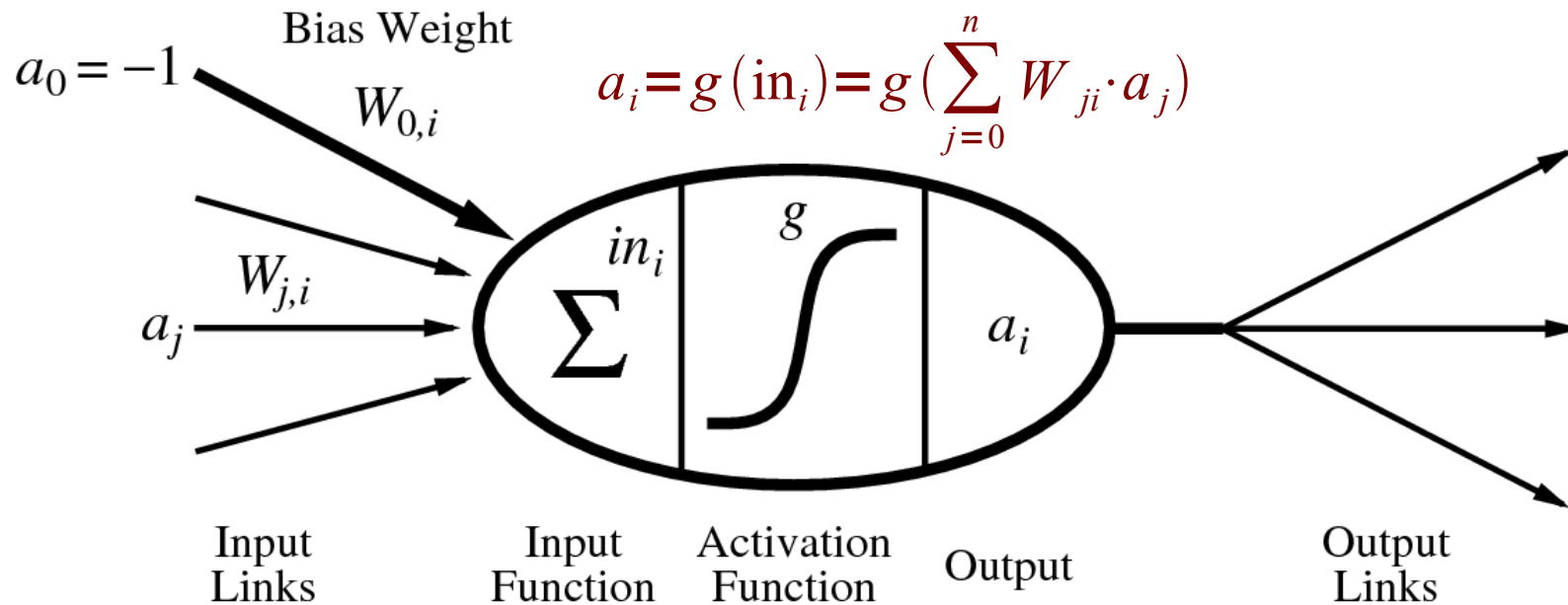
A Biological Neuron



- Neurons are connected to each other via synapses
- If a neuron is activated, it spreads its activation to all connected neurons

An Artificial Neuron

(McCulloch-Pitts, 1943)



- Neurons correspond to nodes or **units**
- A **link** from unit i to unit j propagates activation a_j from j to i
- The **weight** $W_{i,j}$ of the link determines the strength and sign of the connection
- The total **input activation** is the sum of the input activations
- The **output activation** is determined by the activation function g

Perceptron

(Rosenblatt 1957, 1960)

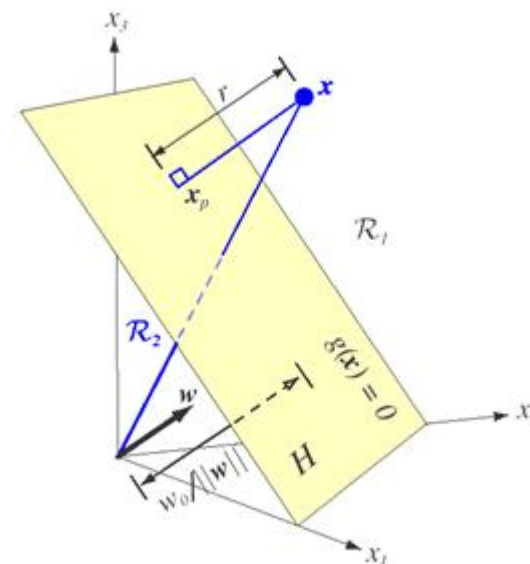
- A single node
 - connecting n input signals with one output signal

- Activation function

- A simple threshold function:

$$a_i = \begin{cases} -1 & \text{if } \sum_{j=0}^n W_{ij} \cdot a_j \leq 0 \\ 1 & \text{if } \sum_{j=0}^n W_{ij} \cdot a_j > 0 \end{cases}$$

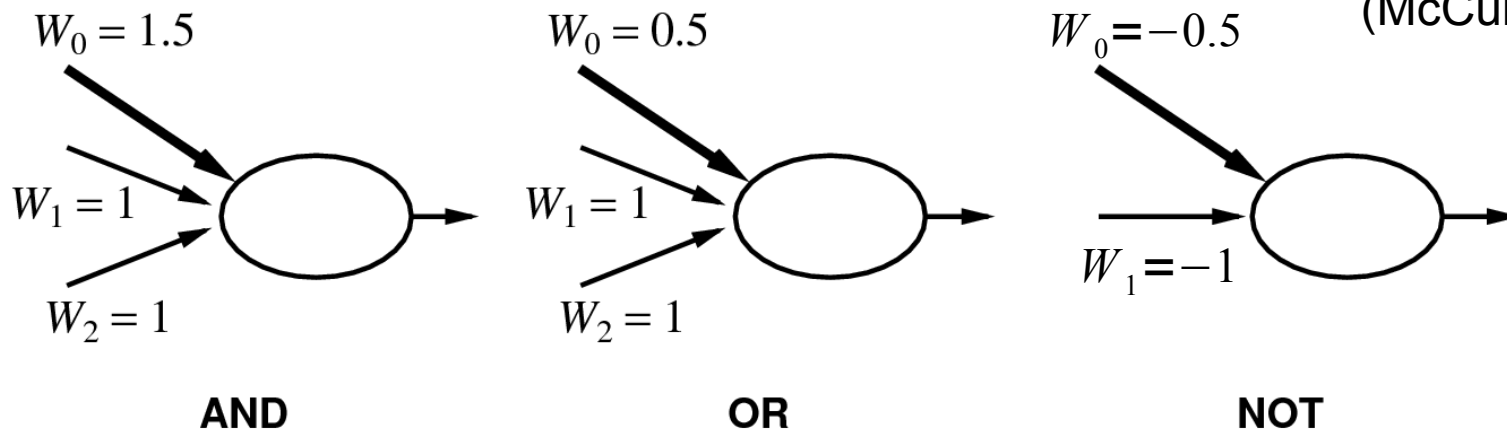
- Thus it implements a **linear separator**
 - i.e., a hyperplane that divides n -dimensional space into a region with output 0 and a region with output 1



Perceptrons and Boolean Functions

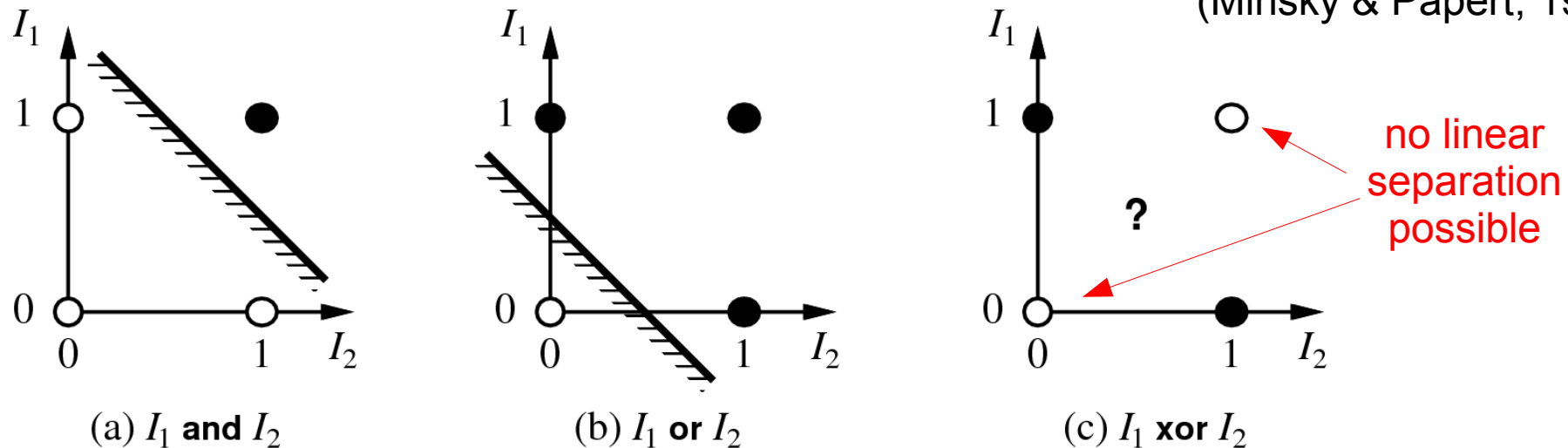
- a Perceptron can implement all elementary logical functions

(McCulloch & Pitts, 1943)



- more complex functions like XOR cannot be modeled

(Minsky & Papert, 1969)



no linear
separation
possible

Perceptron Learning

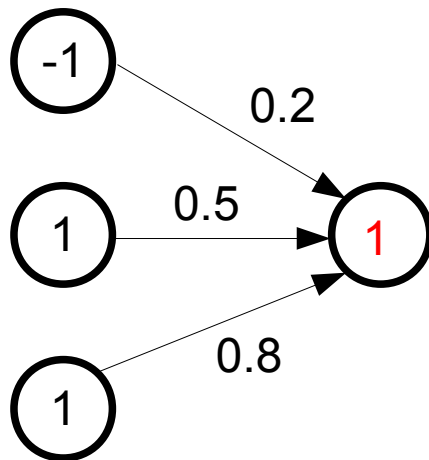
- Perceptron Learning Rule for Supervised Learning

$$W_i \leftarrow W_i + \alpha \cdot (f(\mathbf{x}) - h(\mathbf{x})) \cdot x_i$$

learning rate

error

- Example:



Computation of output signal $h(x)$

$$\text{in}(x) = -1 \cdot 0.2 + 1 \cdot 0.5 + 1 \cdot 0.8 = 1.1$$

$$h(x) = 1 \text{ because } \text{in}(x) > 0$$

Assume target value $f(x) = -1$ (and $\alpha = 0.5$)

$$W_0 \leftarrow 0.2 + 0.5 \cdot (-1 - 1) \cdot -1 = 0.2 + 1 = 1.2$$

$$W_1 \leftarrow 0.5 + 0.5 \cdot (-1 - 1) \cdot 1 = 0.5 - 1 = -0.5$$

$$W_2 \leftarrow 0.8 + 0.5 \cdot (-1 - 1) \cdot 1 = 0.8 - 1 = -0.2$$

Measuring the Error of a Network

- The error for one training example \mathbf{x} can be measured by the squared error
 - the squared difference of the output value $h(\mathbf{x})$ and the desired target value $f(\mathbf{x})$

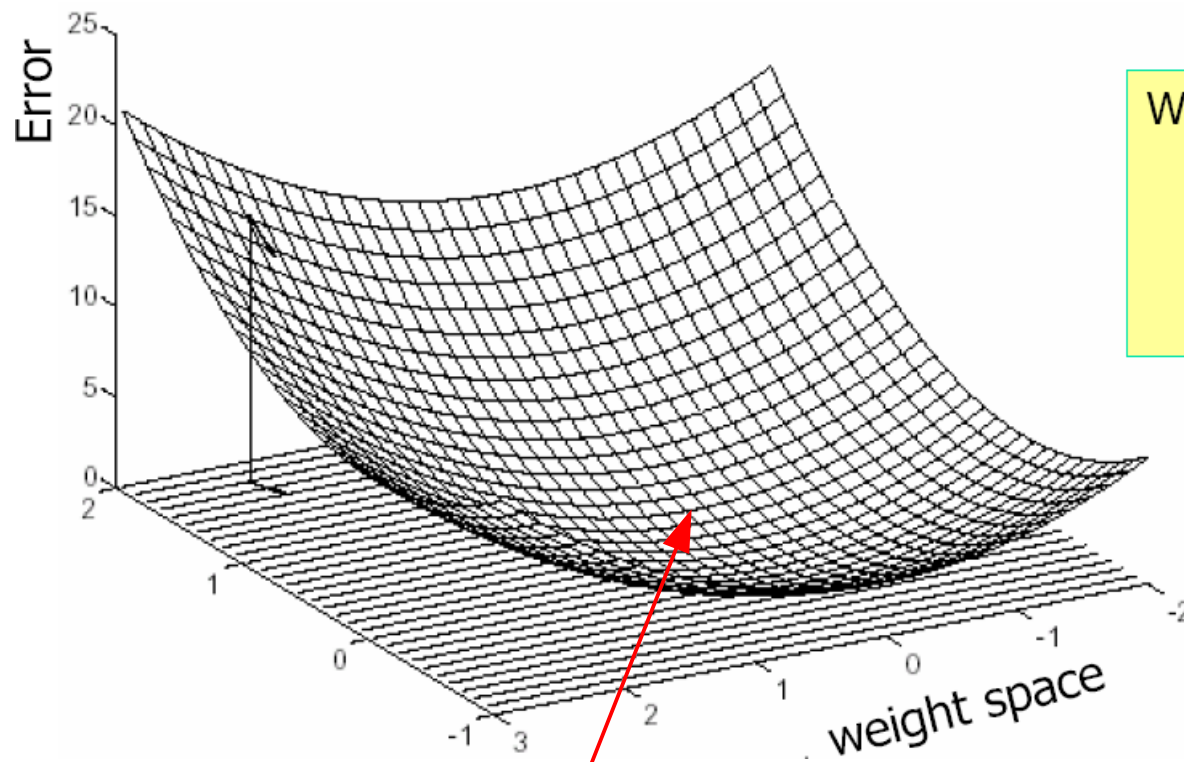
$$E(\mathbf{x}) = \frac{1}{2} \text{Err}^2 = \frac{1}{2} (f(\mathbf{x}) - h(\mathbf{x}))^2 = \frac{1}{2} \left(f(\mathbf{x}) - g\left(\sum_{i=0}^n W_j \cdot x_j\right) \right)^2$$

- For evaluating the performance of a network, we can try the network on a set of datapoints and average the value
(= sum of squared errors)

$$E(\text{Network}) = \sum_{i=1}^N E(\mathbf{x}_i)$$

Error Landscape

- The error function for one training example may be considered as a function in a multi-dimensional weight space



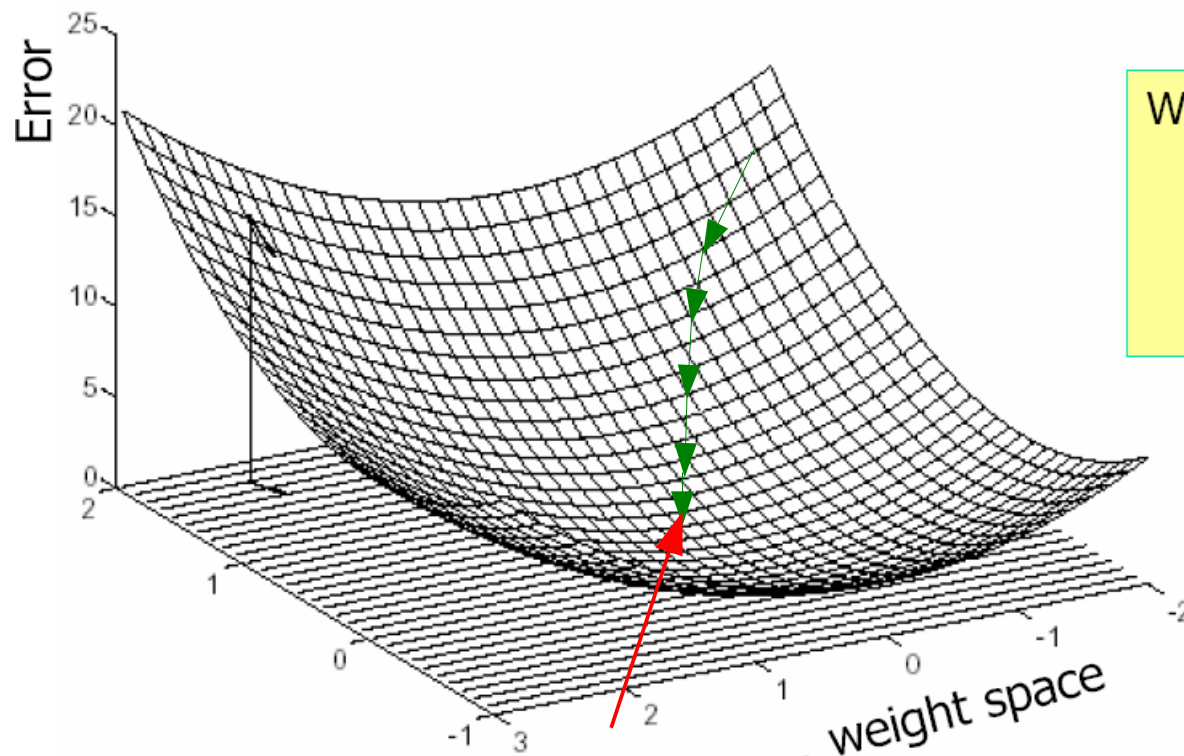
Weight space is N-dimensional, where N is the total number of weights in the network

$$E(W) = \frac{1}{2} \left(f(\mathbf{x}) - g\left(\sum_{i=0}^n W_j \cdot x_j\right) \right)^2$$

- The best weight setting for an example is where the error measure for this example is minimal

Error Minimization via Gradient Descent

- In order to find the point with the minimal error:
 - go downhill in the direction where it is steepest



Weight space is N-dimensional, where N is the total number of weights in the network

$$E(W) = \frac{1}{2} \left(f(\mathbf{x}) - g\left(\sum_{i=0}^n W_j \cdot x_j\right) \right)^2$$

- ... but make small steps, or you might step over the target

Error Minimization

- It is easy to derive a perceptron training algorithm that minimizes the squared error

$$E = \frac{1}{2} Err^2 = \frac{1}{2} (f(\mathbf{x}) - h(\mathbf{x}))^2 = \frac{1}{2} \left(f(\mathbf{x}) - g\left(\sum_{i=0}^n W_i \cdot x_i\right) \right)^2$$

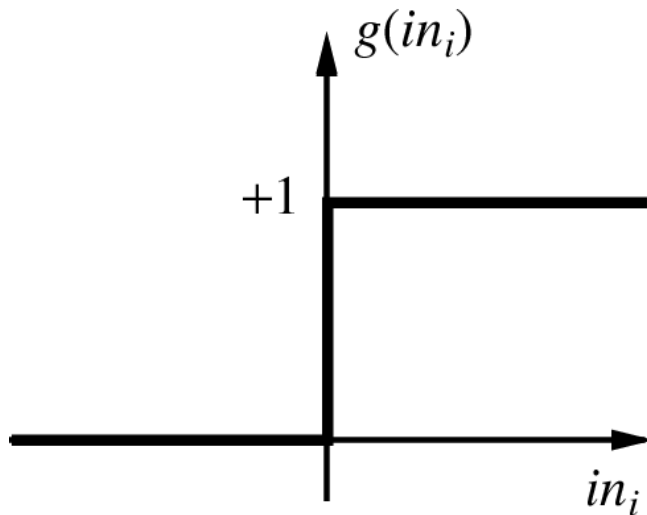
- Change weights into the direction of the steepest descent of the error function

$$\frac{\partial E}{\partial W_j} = Err \cdot \frac{\partial Err}{\partial W_j} = Err \cdot \frac{\partial}{\partial W_j} \left(f(\mathbf{x}) - g\left(\sum_{i=0}^n W_i \cdot x_i\right) \right) = -Err \cdot g'(\text{in}) \cdot x_j$$

- To compute this, we need a continuous and differentiable activation function g !
- Weight update with learning rate α : $W_j = W_j + \alpha \cdot Err \cdot g'(\text{in}) \cdot x_j$
 - positive error \rightarrow increase network output
 - increase weights of nodes with positive input
 - decrease weights of nodes with negative input

Threshold Activation Function

- The regular threshold activation function is problematic
 - $g'(x) = 0$, therefore $\frac{\partial E}{\partial W_j} = -Err \cdot g'(in) \cdot x_j = 0$
→ no weight changes!

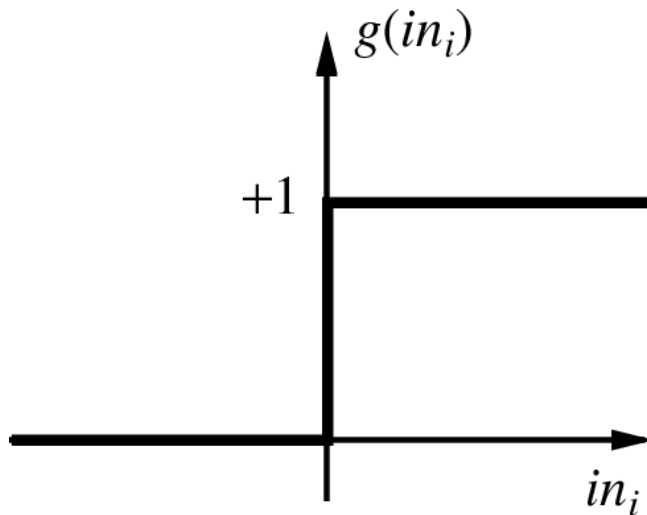


$$g(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

$$g'(x) = 0$$

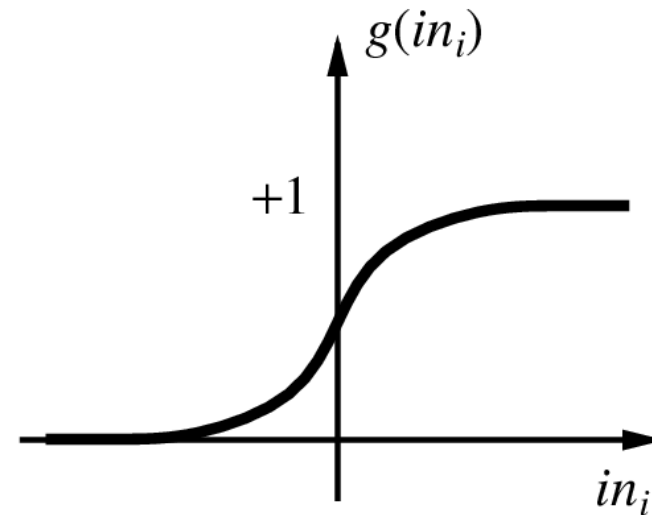
Sigmoid Activation Function

- A commonly used activation function is the sigmoid function
 - similar to the threshold function
 - easy to differentiate
 - non-linear



$$g(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

$$g'(x) = 0$$

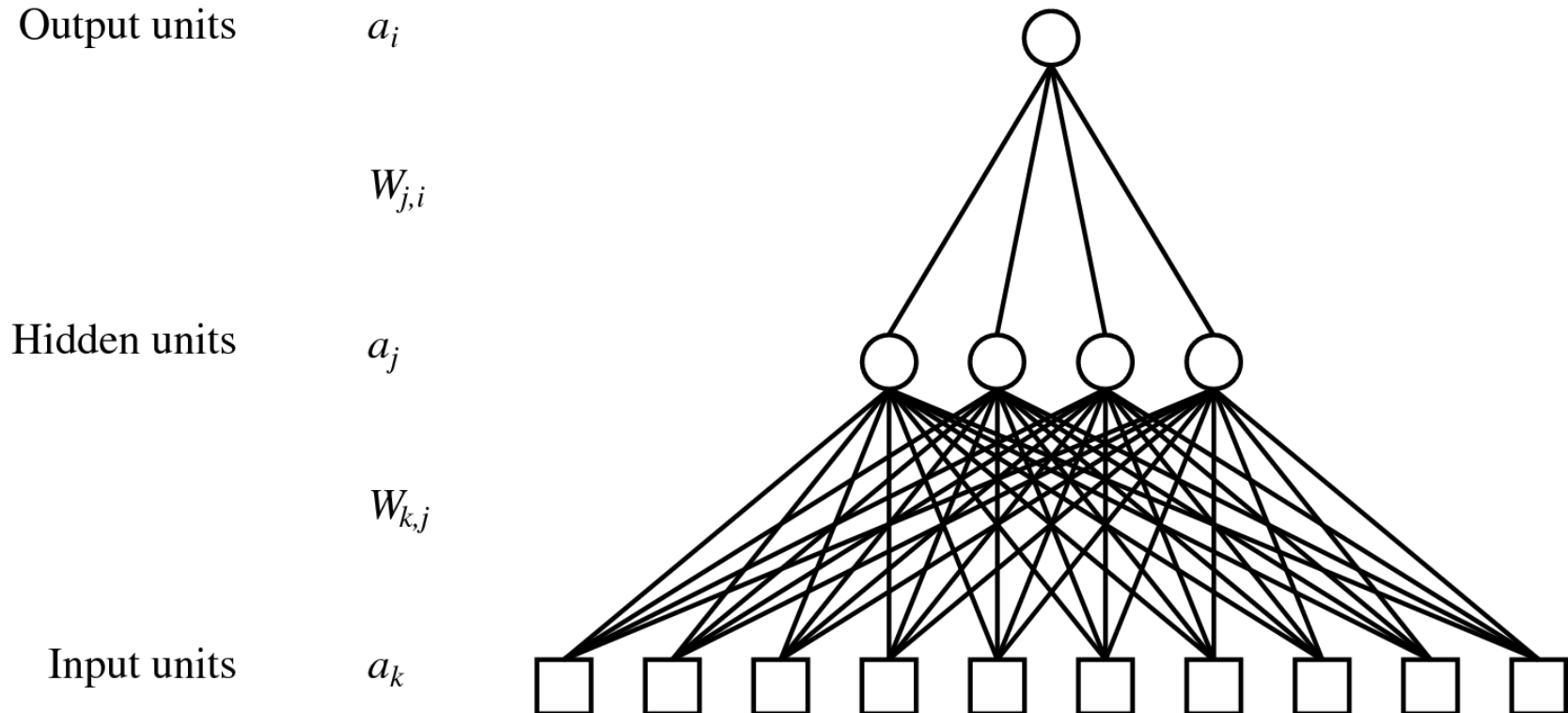


$$g(x) = \frac{1}{1 + e^{-x}}$$

$$g'(x) = g(x)(1 - g(x))$$

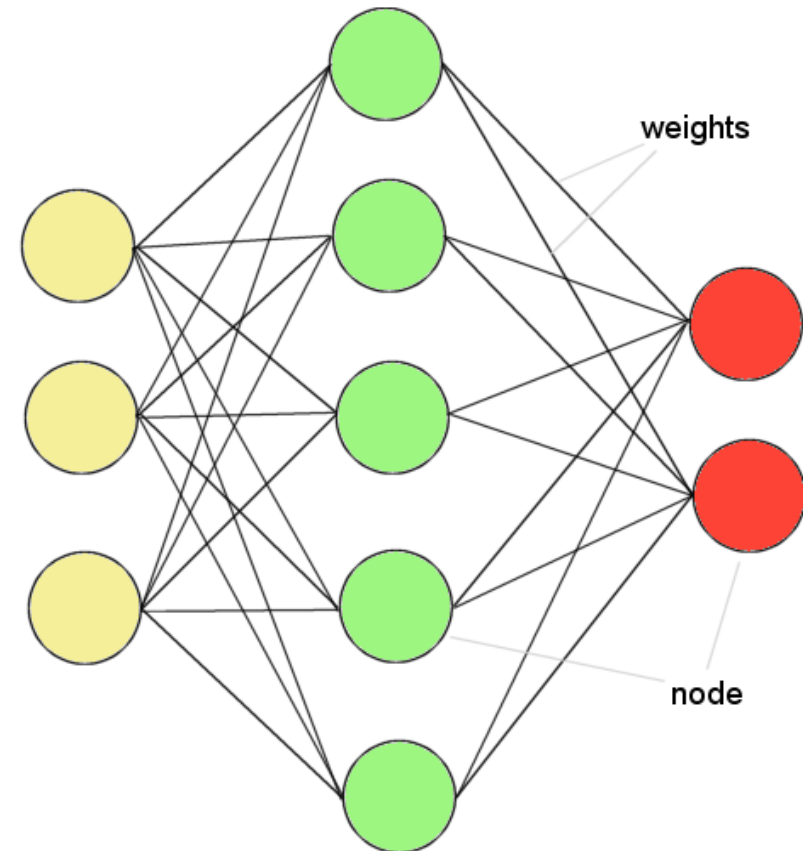
Multilayer Perceptrons

- Perceptrons may have multiple output nodes
 - may be viewed as multiple parallel perceptrons
- The output nodes may be combined with another perceptron
 - which may also have multiple output nodes
- The size of this **hidden layer** is determined manually



Multilayer Perceptrons

Input Hidden Output



- Information flow is unidirectional
 - Data is presented to *Input layer*
 - Passed on to *Hidden Layer*
 - Passed on to *Output layer*
- Information is distributed
- Information processing is parallel

Information



Expressiveness of MLPs

- Every continuous function can be modeled with three layers
 - i.e., with one hidden layer
- Every function with four layers
 - i.e., with two hidden layers

Backpropagation Learning

- The **output nodes** are trained like a normal perceptron

$$W_{ji} = W_{ji} + \alpha \cdot Err_i \cdot g'(in_i) \cdot x_j = W_{ji} + \alpha \cdot \Delta_i \cdot x_j$$

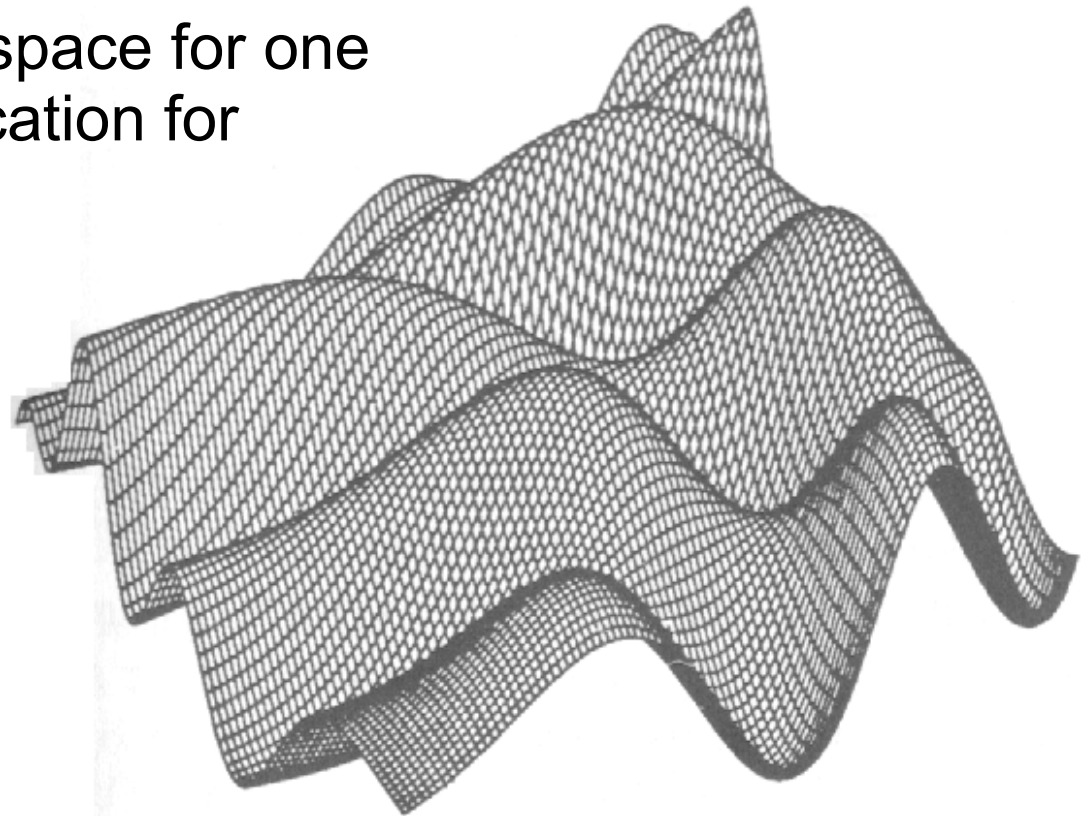
- Δ_i is the error of output node i times the derivation of its inputs
- the error of the output layers is propagated back to the **hidden layer**

$$\Delta_j = \left(\sum_i W_{ji} \cdot \Delta_i \right) \cdot g'(in_j) \qquad W_{kj} = W_{kj} + \alpha \cdot \Delta_j \cdot x_k$$

- the training signal of hidden layer node j is the weighted sum of the errors of the output nodes

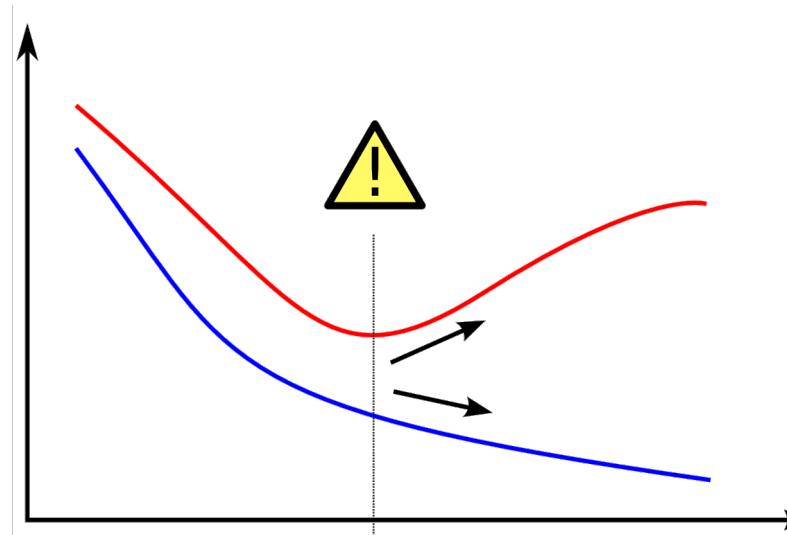
Minimizing the Network Error

- The error landscape for the entire network may be thought of as the sum of the error functions of all examples
 - will yield many local minima → hard to find global minimum
- Minimizing the error for one training example may destroy what has been learned for other examples
 - a good location in weight space for one example may be a bad location for another examples
- **Training procedure:**
 - try all examples in turn
 - make small adjustments for each example
 - repeat until convergence
- One Epoch = One iteration through all examples



Overfitting

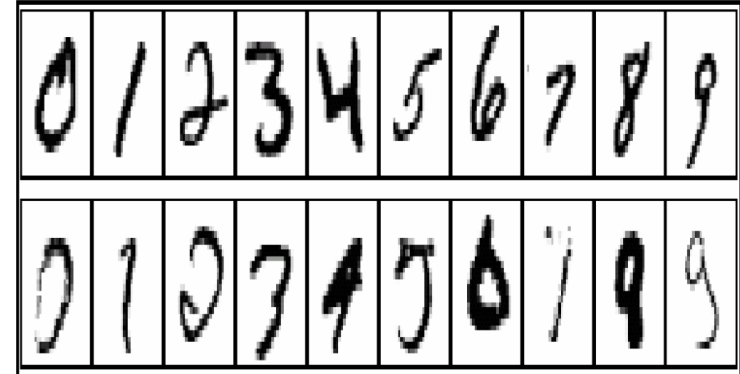
- **Training Set Error** continues to decrease with increasing number of training examples / number of epochs
 - an epoch is a complete pass through all training examples
- **Test Set Error** will start to increase because of overfitting



- Simple training protocol:
 - keep a separate **validation set** to watch the performance
 - validation set is different from training and test sets!
 - stop training if error on validation set gets down

Wide Variety of Applications

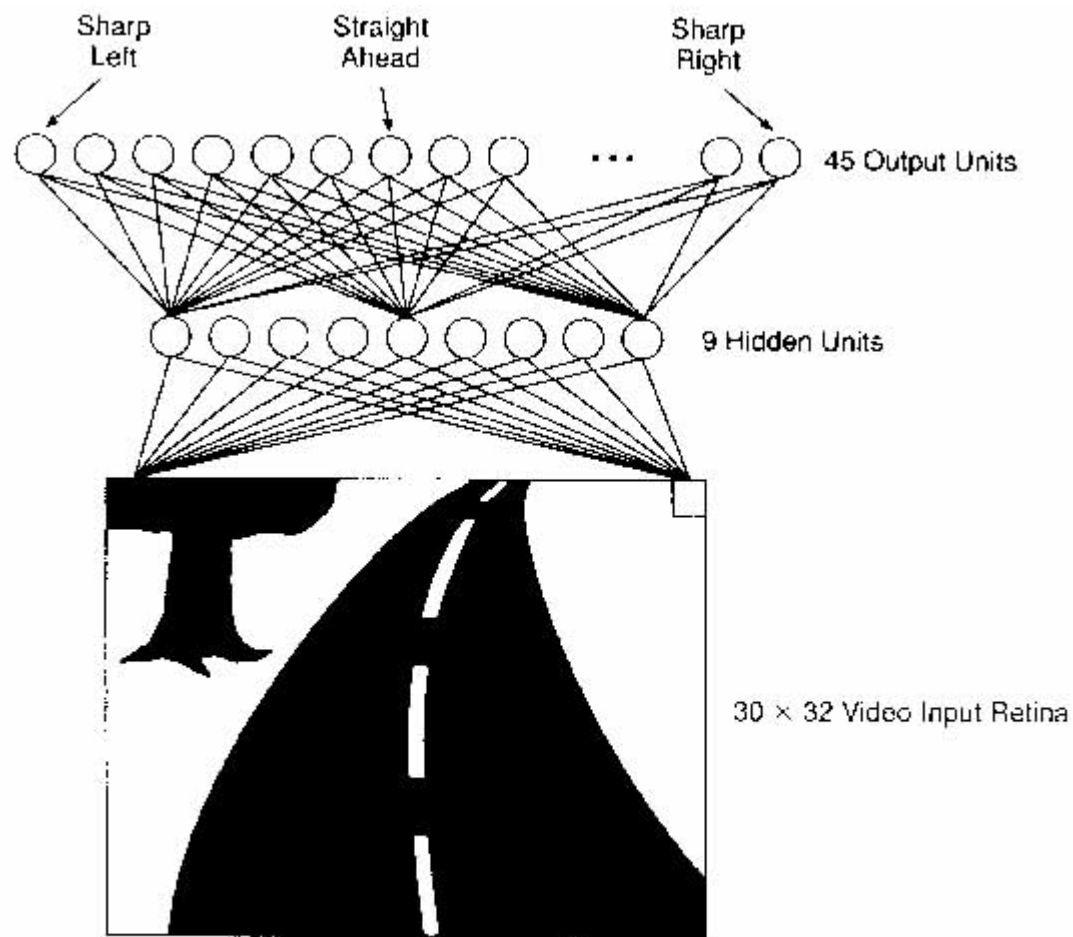
- Speech Recognition
- Autonomous Driving
- Handwritten Digit Recognition
- Credit Approval
- Backgammon
- etc.



- **Good** for problems where the final output depends on combinations of many input features
 - rule learning is better when only a few features are relevant
- **Bad** if explicit representations of the learned concept are needed
 - takes some effort to interpret the concepts that form in the hidden layers

Autonomous Land Vehicle In a Neural Network (ALVINN)

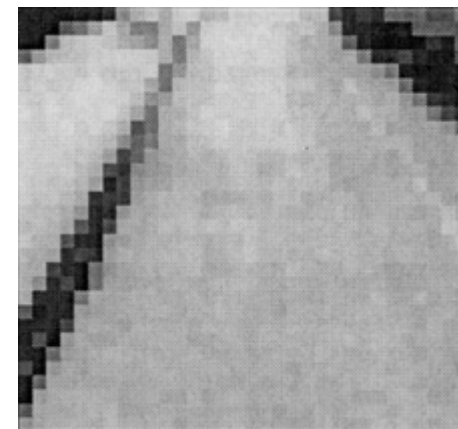
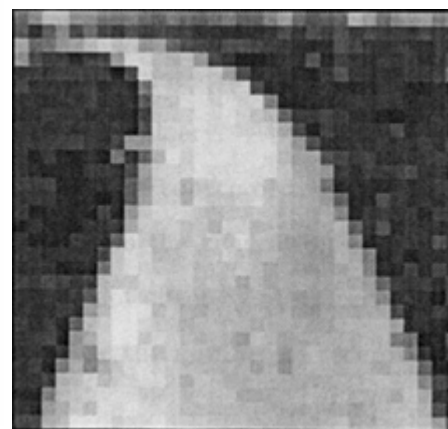
(Dean Pomerleau)



Training Data: Human Driver on real roads and simulator



Sensory input is directly fed into the network



Multilayer Network Control Unit for Navl.ab
(From Kanade et al., 1994. © 1994 ACM. Courtesy of the authors.)