Review

Lecture 7: Beyond Classical search
Neural Networks

Chapter 4, Chapter 18

LI Xiaoan (Dustin) 李孝安, Dr. & Associate Prof.

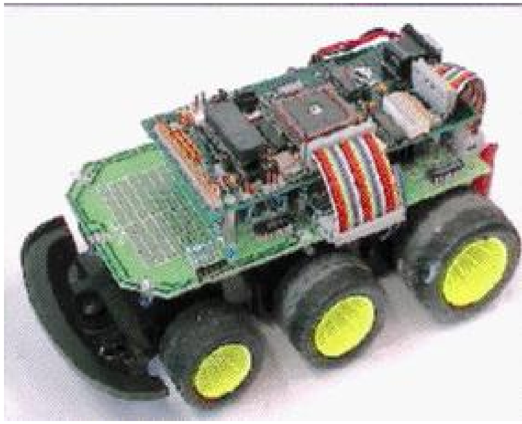
QQ group: AI-2021-Li， 704296172
WeChat group: U10M12018.01 人工智能（英）群

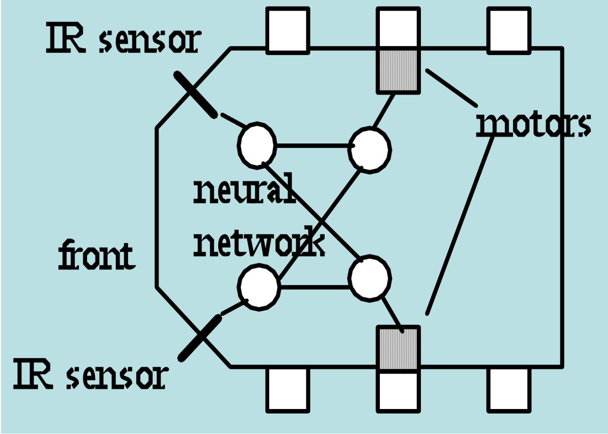
Review

Outline

- Best-first search
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms
- Neural Networks
- Summary

Case1: Evolving a neural controller for an agent

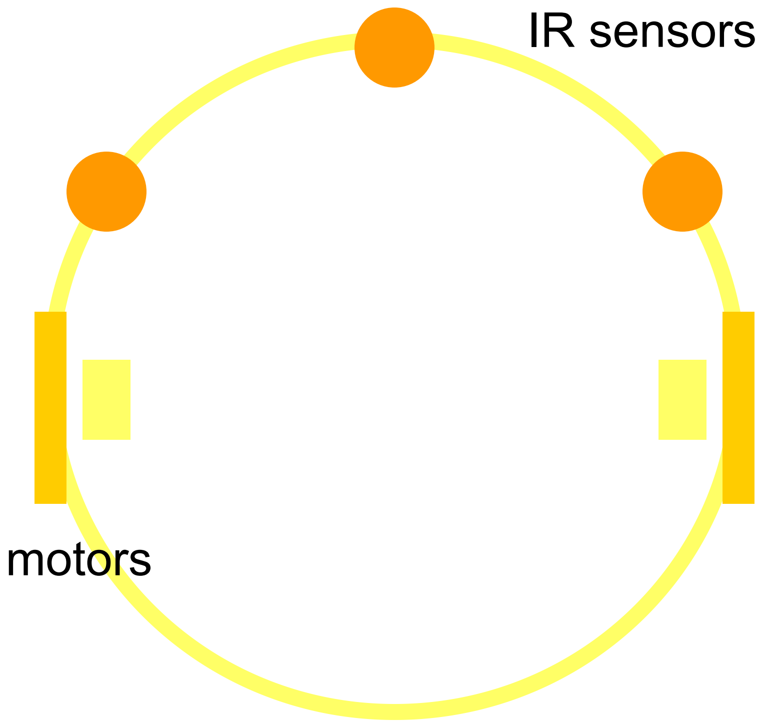




Didabot:
simple robot
for didactical purposes

arena with Styrofoam cubes

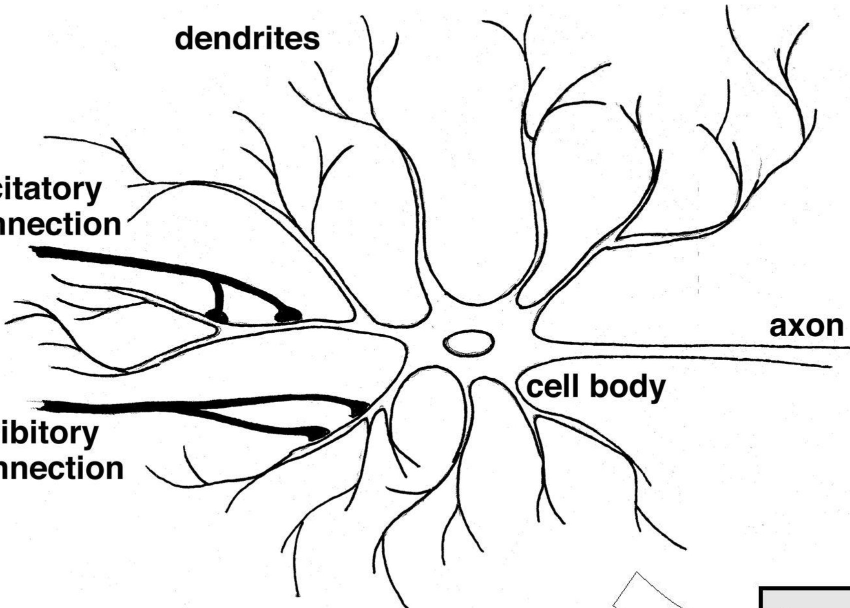
Evolving a neural controller for an agent

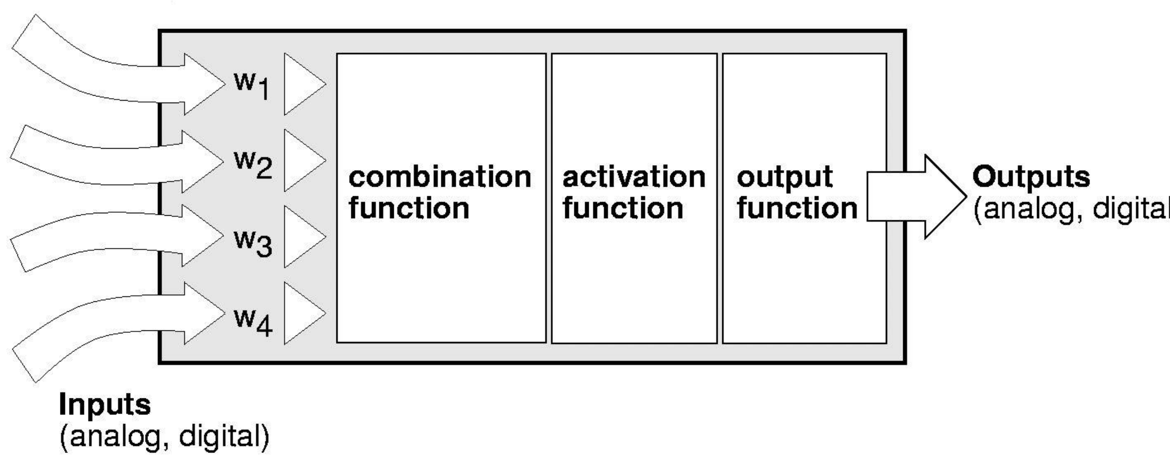


how to proceed?

Modeling a neuron - neural networks

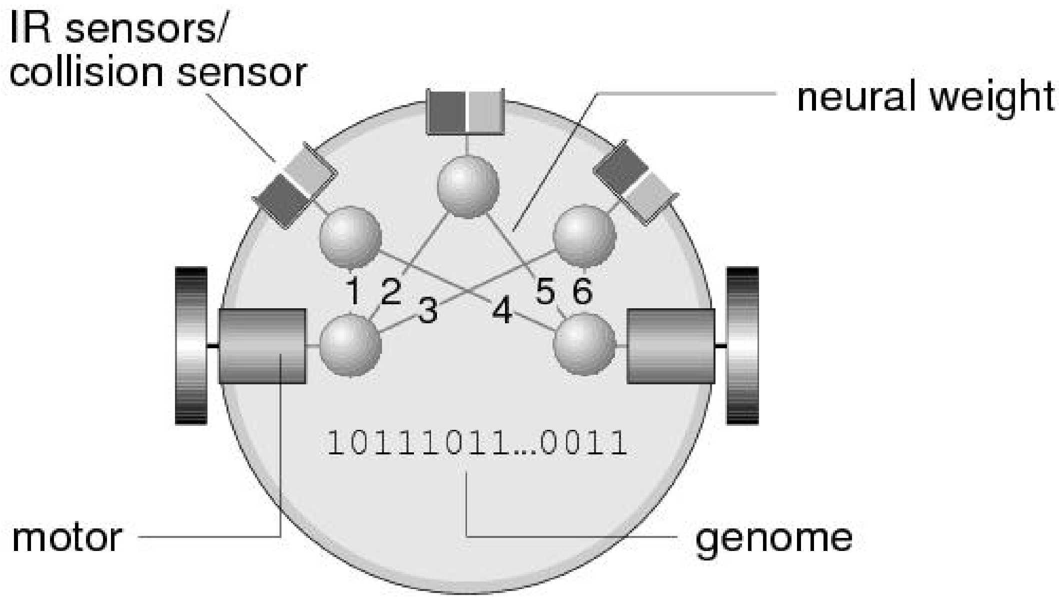
motivation: abstraction from biological brains to artificial neural networks





Encoding in genome

a.



b.	initial genome	1101	0110	0001	0011	1010	1100
	encodes weights (numbers)	1	2	3	4	5	6
c.	initial weights after “development”	.37	−.1	−.43	−.3	.16	.3

Encoding in genome "development"

1. take the one single individual with the highest fitness
2. choose another individual from the population at random, irrespective of fitness, for sexual reproduction
3. add the fittest individual to the new population

fittest individual (highest rank)

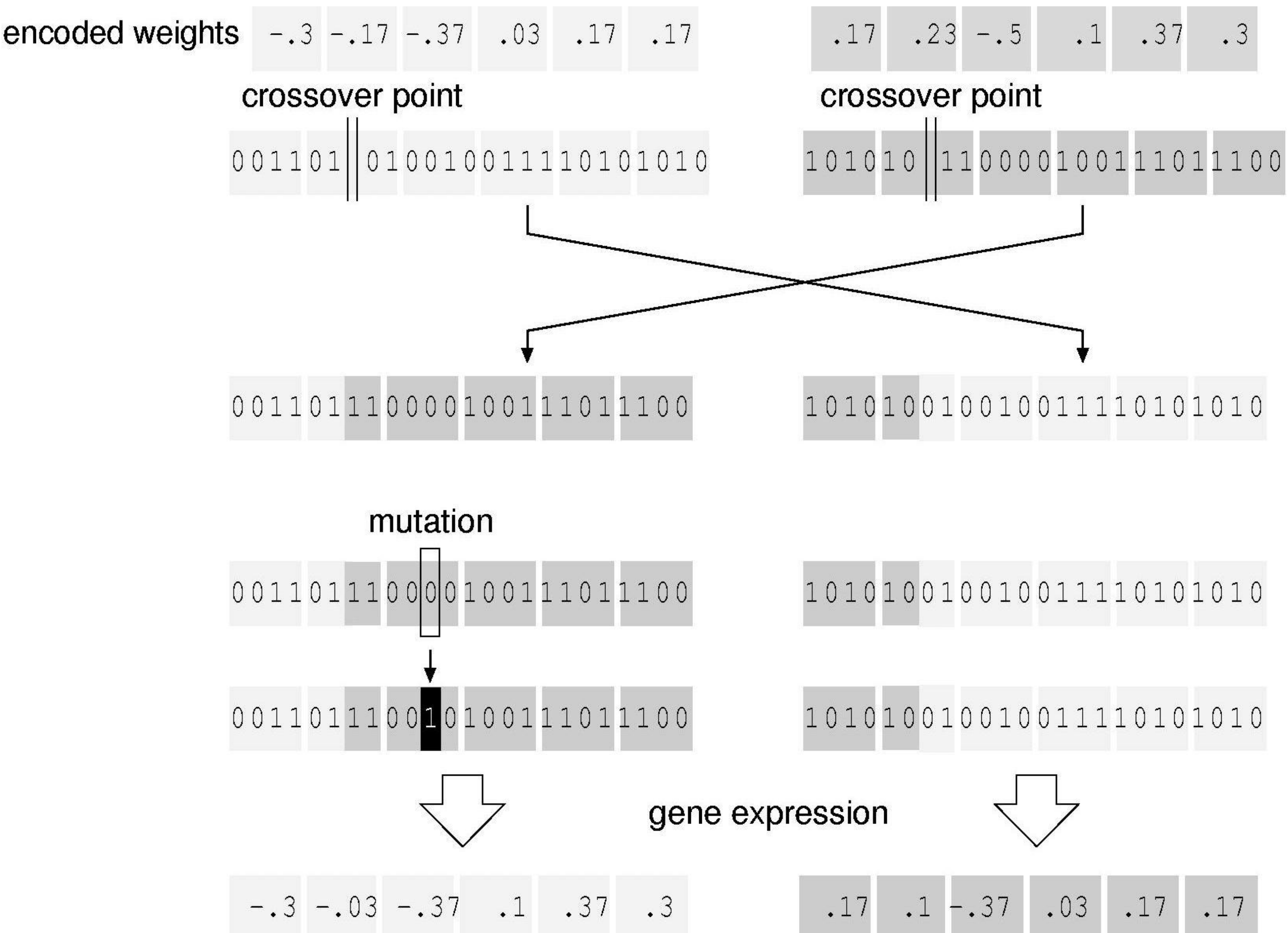
	1	2	3	4	5	6
initial genome	0011	0101	0010	0111	1010	1010
encoded weights	-.3	-.17	-.37	.03	.17	.17

other individual

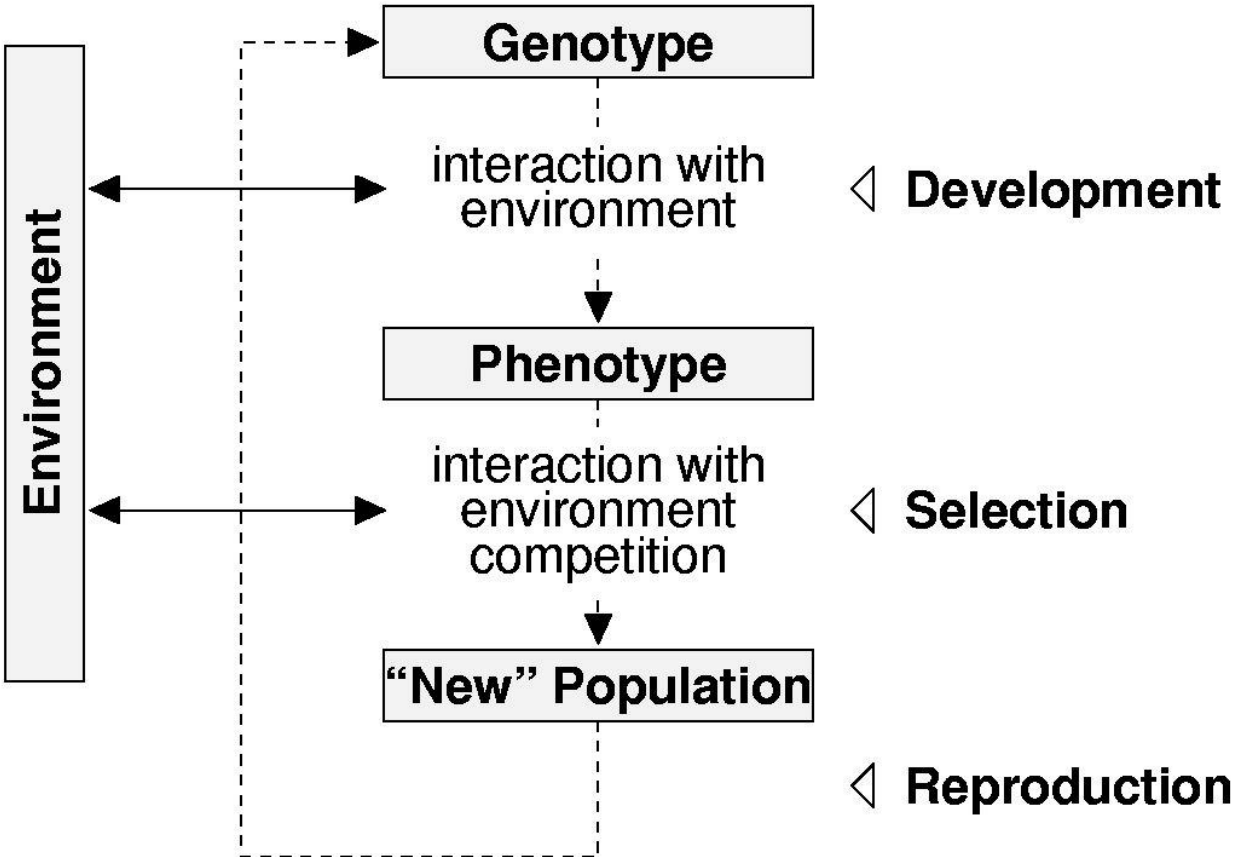
	1	2	3	4	5	6
initial genome	1010	1011	0000	1001	1101	1100
encoded weights	.17	.23	-.5	.1	.37	.3

encoding	development	selection	reproduction
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• "roulette wheel" • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

Reproduction: Crossover and mutation “development”



“Grand” evolutionary scheme



encoding	development	selection	reproduction
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• “roulette wheel” • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

Review

Summary

- Informed Search Algorithms
- Beyond classical search algorithms
  - Local search algorithms
  - Simulated annealing (SA)
  - **Genetic algorithms (GA)**
- *Still challenging*
  - **Neural Networks (NN, ANN)**
  - EA (GA, EP, ES, GP)
  - Deep Learning

  ……

Assignment

- Readings:
  - Chap 4
  - additional slides about "Braitenberg vehicle" , "**SWISS ROBOTS**"
- Exercises
  - Chap 4: exercise 4.1
  - 2 Additional exercises

*Handed in next Tuesday

Assignment (Additional)

1. The Braitenberg vehicle in figure 1 implement its controllers with a simple neural network which has two layers of neurons; starting from the top sides, the first layer receives inputs from the sensors and sends its outputs to the second layer, while the neurons in the second layer drive the motors of the robot. The picture shows a schematic representation of a mobile robot. Assume that it is moving at a default (slow) speed.

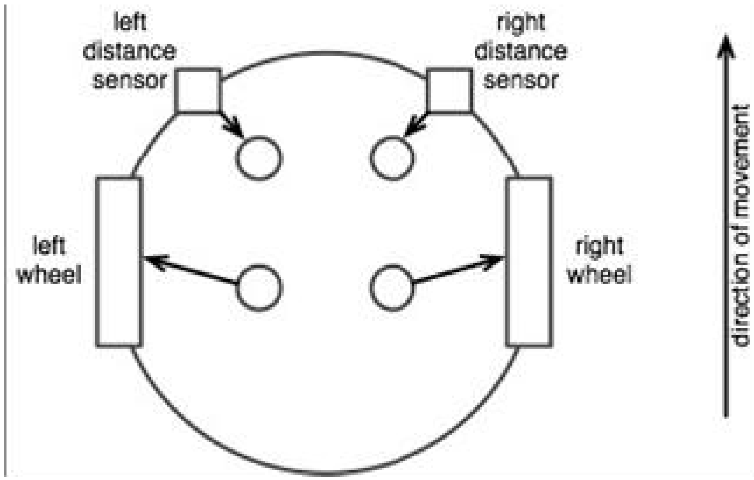


Figure 1

a) Implement a simple neural network by connecting the neurons (small circles) in order to implement the obstacle avoidance behavior (i.e., while moving in the environment, the robot avoids the obstacle that it senses by means of the IR sensors). You do not need to specify the weights of the connections; just say whether each connection is excitatory (+) or inhibitory (–). There are several ways to achieve this, just come up with ONE solution.

b) Are there any inspirations for you to design an intelligent robot?

Assignment (Additional)

2. **Braitenberg Vehicles:** In his book "Vehicles: Experiments in Synthetic Psychology", Braitenberg describes, among other things the following vehicles:

a) The "love" vehicle likes to stay as close to a light source as possible.

b) The "aggression" vehicle tries to destroy the light sources by colliding with them.

c) The "fear" vehicle flees away from any light source.

d) The "explorer" vehicle slows down at each light source and then goes to the next one.

As shown in the following figure 2, each robot possesses two light sensors as well as either positive or negative connections to the motors. All connections have the same absolute weight values. The signs of the weights are given in the head of the corresponding arrows. The signal amplitude of the light sensors is here proportional to the intensity of detected light.

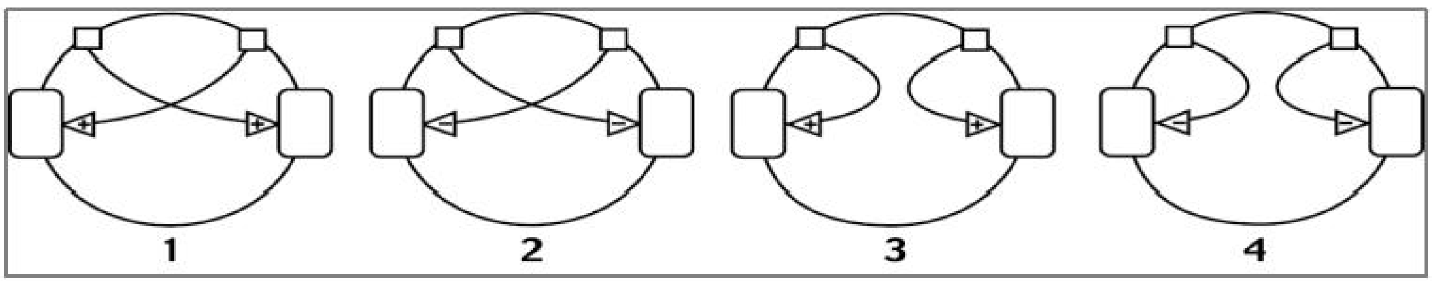


Figure 2

(1) The task is to assign to each robot schema as in figure 2 the correct behavior (a, b, c or d) – write the respective character next to the number.

(2) What happens if you keep the same connections but you move the position of the right sensor to the left side of the robot (just next to the left sensor) in figure 2?

(3) What conclusions can you derive from this experiment?

Assignment (Additional)

3. "SWISS ROBOTS" are a set of simple robots, each robot as shown in Figure 1 was equipped with two motors, one on the left and one on the right, and two infrared sensors, one front left and one front right.

(a) initial state (b) final state (c) *Didabot robot*

Figure 4 Swiss Robots

a) What intelligent behaviors do you think emerged from the "Swiss Robots" after you watched the video illustration as in figure 1?
b) What are the robots really doing when we think in agent's perspective –situated perspective?
c) Are there any inspirations for you to design an intelligent robot?

Lecture 8: Beyond Classical search

Neural networks

LI Xiaoan (Dustin) 李孝安, Dr. & Associate Prof.

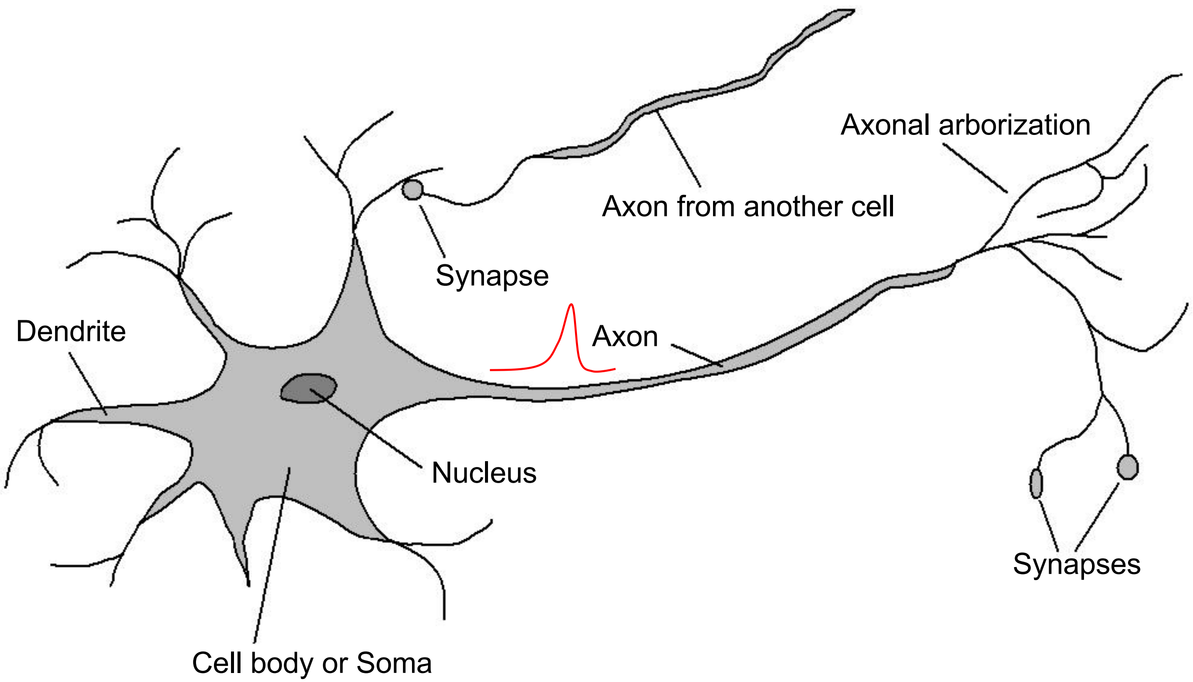
QQ group: AI-2021-Li， 704296172
WeChat group: U10M12018.01 人工智能（英）群

Outline

- Brains
- Neural networks
- Perceptrons
- Multi-layer Perceptrons
- Applications of neural networks
- Evolving a neuro controller for robots
- Summary

Brains

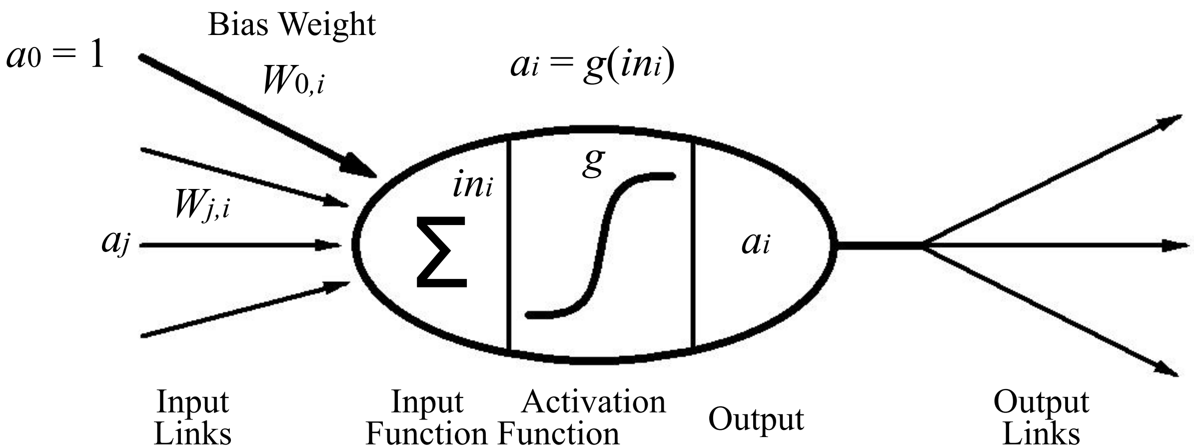
10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



McCulloch–Pitts "unit": M-P model of neuron

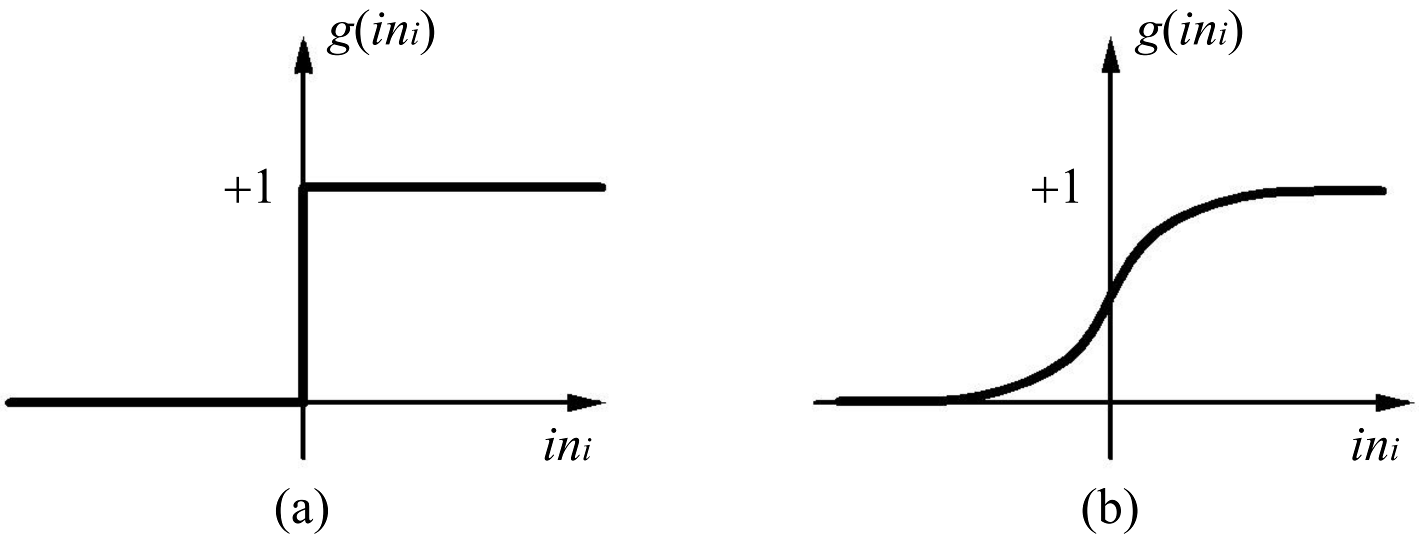
Output is a "squashed" linear function of the inputs:

$a_i \leftarrow g(in_i) = g\ \Sigma_j W_{j,i} a_j$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

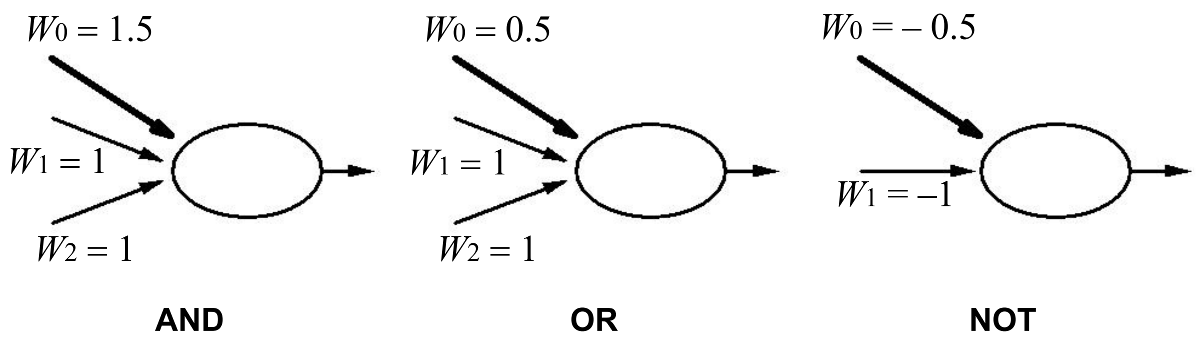


(a) is a step function or threshold function

(b) is a sigmoid function such as $1/(1 + e^{-x})$, $(1-e^{-x})/(1 + e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented

Network structures

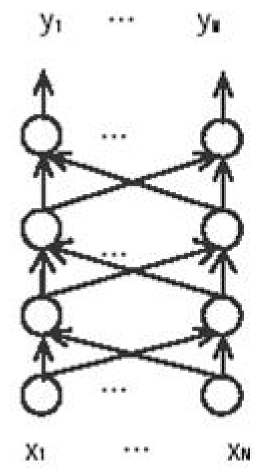


Figure 1 one MLP

Feed-forward networks:
- – single-layer perceptrons
- – multi-layer perceptrons

Feed-forward networks implement functions, have no internal state

Recurrent networks:
- – Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$)
  $g(x) = \mathrm{sign}(x)$, $a_i = \pm 1$; holographic associative memory
- – Boltzmann machines use stochastic activation functions,
  $\approx$ MCMC in Bayes nets
- – recurrent neural nets have directed cycles with delays
  $\Rightarrow$ have internal state (like flip-flops), can oscillate etc.

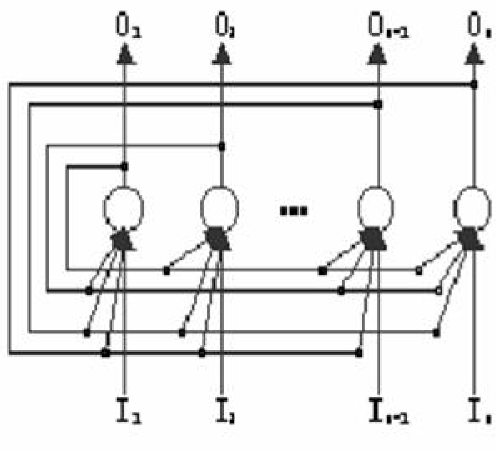
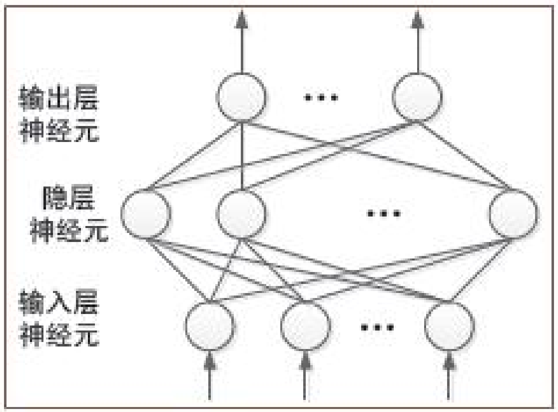


Figure 2 one RNN

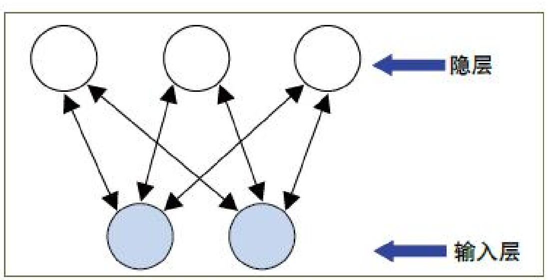
Deep learning networks:

- – Boltzmann machines
- – Deep Believe Networks
- – Convolutional NN

- DBNN: Deep learning
  - RBM
  - BP



(a) BPN



(b) RBM

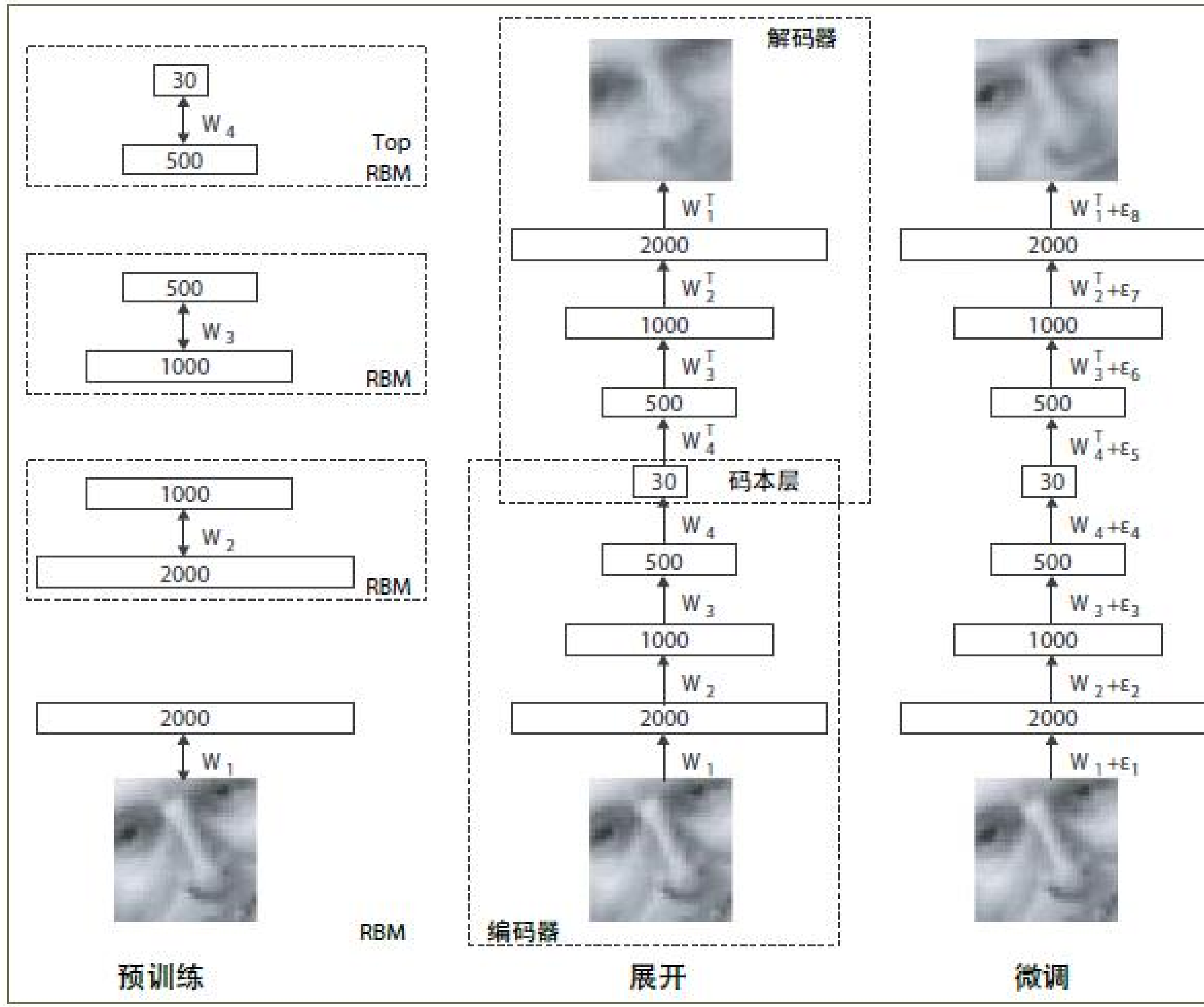
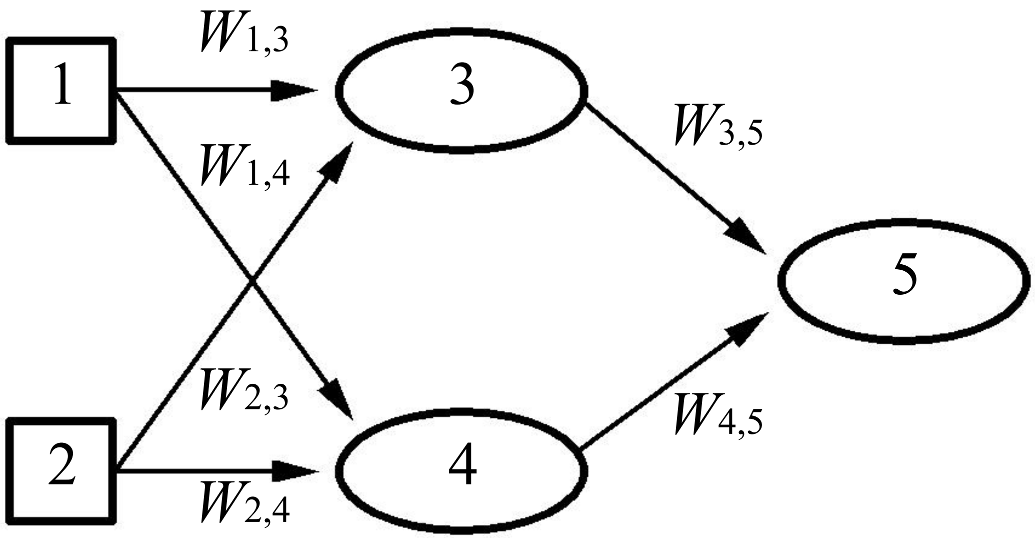


Figure 3 one DLNN

Feed-forward example

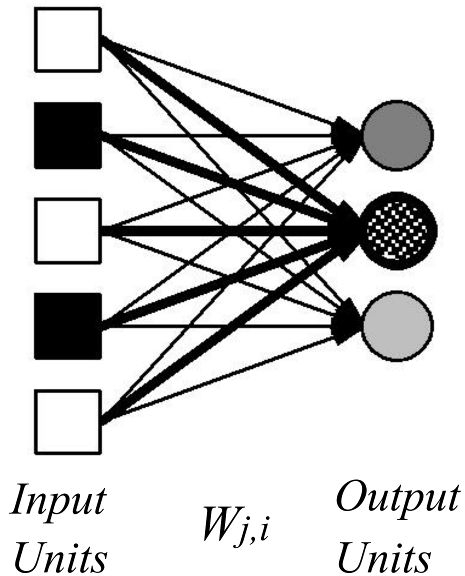


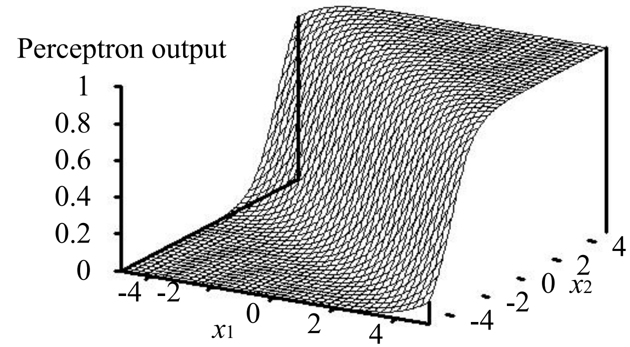
Feed-forward network = a parameterized family of nonlinear functions:

$$
\begin{aligned}
a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\
&= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))
\end{aligned}
$$

Adjusting weights changes the function: do learning this way!

Single-layer perceptrons





Output units all operate separately—no shared weights

Adjusting weights moves the location, orientation, and steepness of cliff

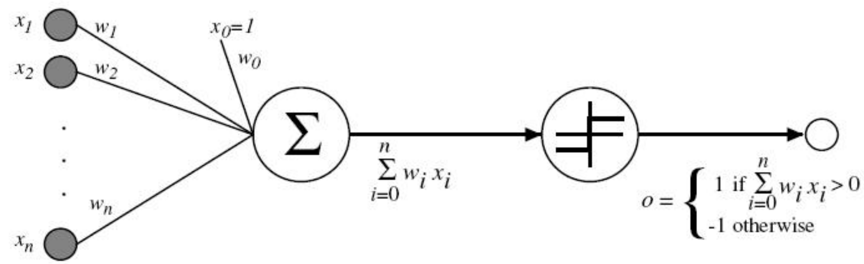
Expressiveness of perceptrons

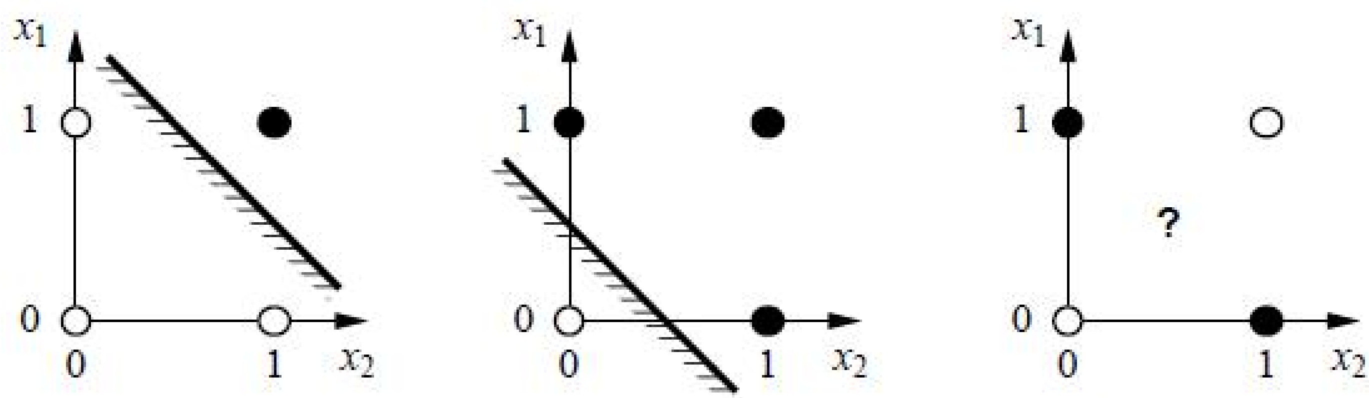
Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$\Sigma_j W_j x_j > 0$ or $W \cdot x > 0$



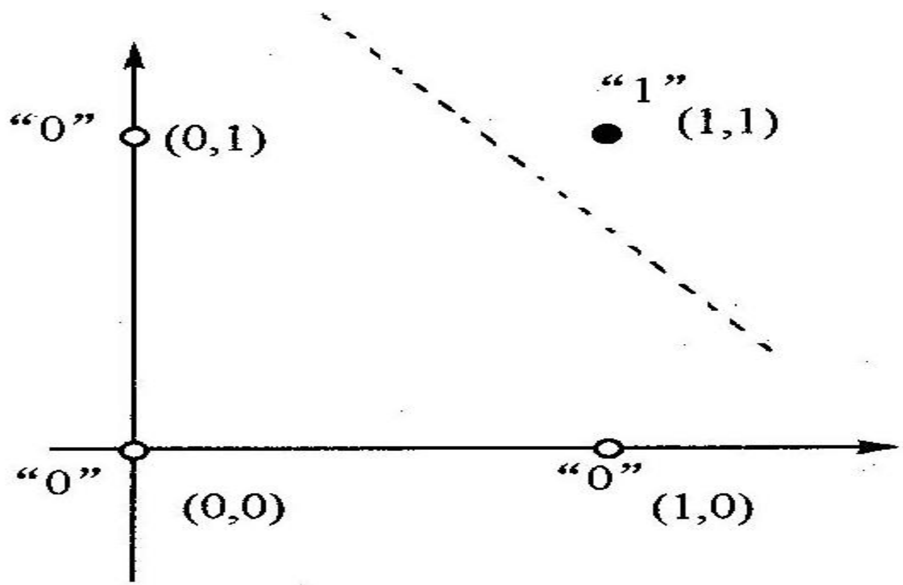


(a) x_1 and x_2 (b) x_1 or x_2 (c) x_1 xor x_2

Minsky & Papert (1969) pricked the neural network balloon because of ***Perceptrons***

Table 2-1 “AND”

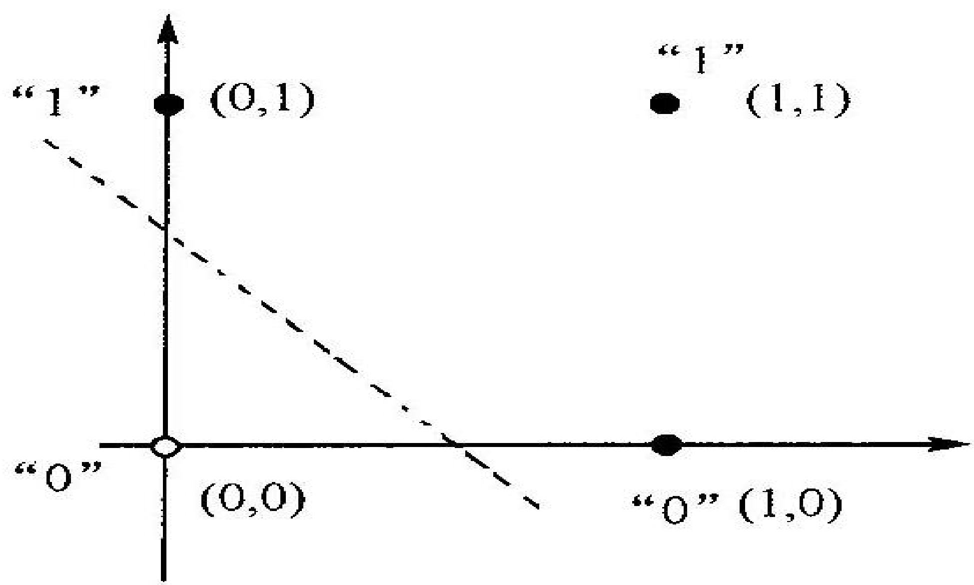
x_1	x_2	$x_1 x_2$	$Y = w_1 \cdot x_1 + w_2 \cdot x_2 - b = 0$	conditions
0	0	0	$Y = w_1 \cdot 0 + w_2 \cdot 0 - b < 0$	$b > 0$
0	1	0	$Y = w_1 \cdot 0 + w_2 \cdot 1 - b < 0$	$b > w_2$
1	0	0	$Y = w_1 \cdot 1 + w_2 \cdot 0 - b < 0$	$b > w_1$
1	1	1	$Y = w_1 \cdot 1 + w_2 \cdot 1 - b \geqslant 0$	$b \leqslant w_1 + w_2$



“AND” $w_1 = 1, w_2 = 1, b = 1.5$

Table 2-2 "OR"

x_1	x_2	$x_1 x_2$	$Y = w_1 \cdot x_1 + w_2 \cdot x_2 - b = 0$	conditions
0	0	0	$Y = w_1 \cdot 0 + w_2 \cdot 0 - b < 0$	$b > 0$
0	1	1	$Y = w_1 \cdot 0 + w_2 \cdot 1 - b \geq 0$	$b \leq w_2$
1	0	1	$Y = w_1 \cdot 1 + w_2 \cdot 0 - b \geq 0$	$b \leq w_1$
1	1	1	$Y = w_1 \cdot 1 + w_2 \cdot 1 - b \geq 0$	$b \leq w_1 + w_2$

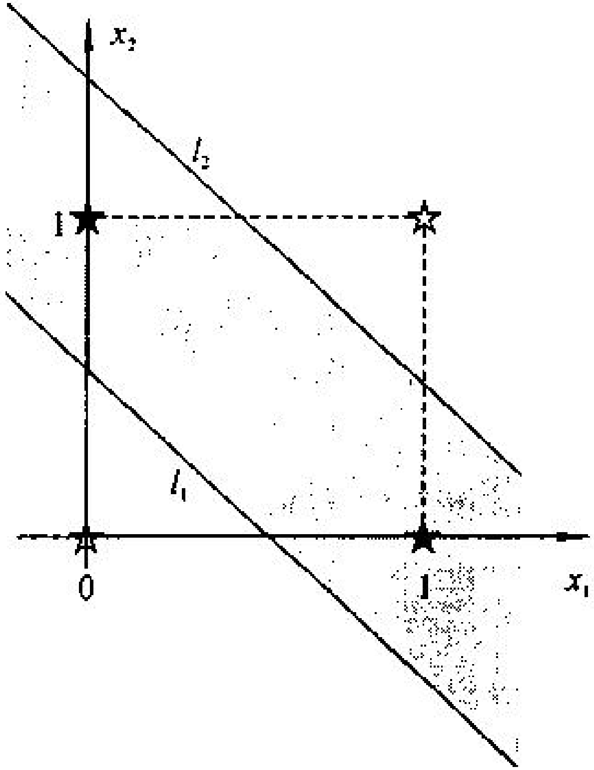


"OR"

$$w_1 = 1, w_2 = 1, b = 0.5$$

XOR based on Perceptrons

Ideas!



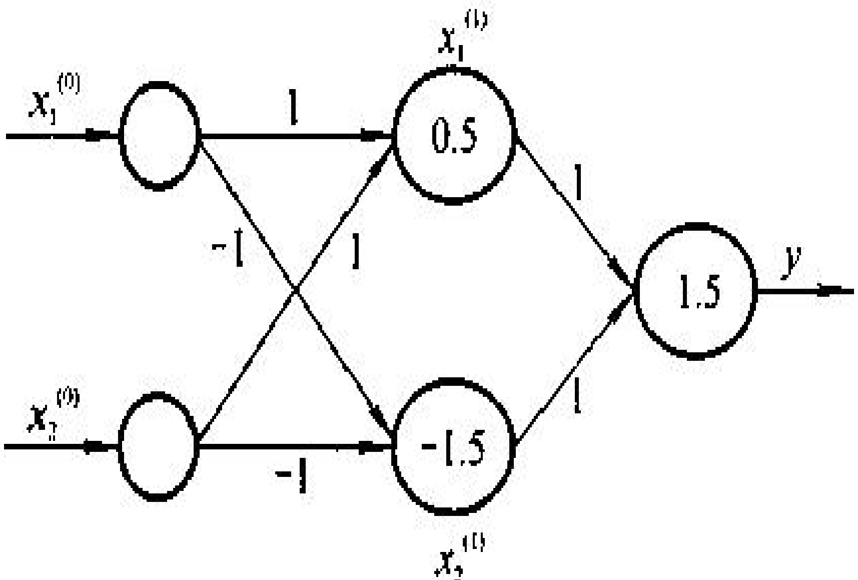
Four rules expressed for XOR:

IF x_1=0 AND x_2=0 THEN y=0

IF x_1=0 AND x_2=1 THEN y=1

IF x_1=1 AND x_2=0 THEN y=1

IF x_1=1 AND x_2=1 THEN y=0



Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input x and true output y is

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2}(y - h_{W}(x))^2 ,$$

Perform optimization search by ***gradient descent***:

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right)$$

$$= -Err \times g'(in) \times x_j$$

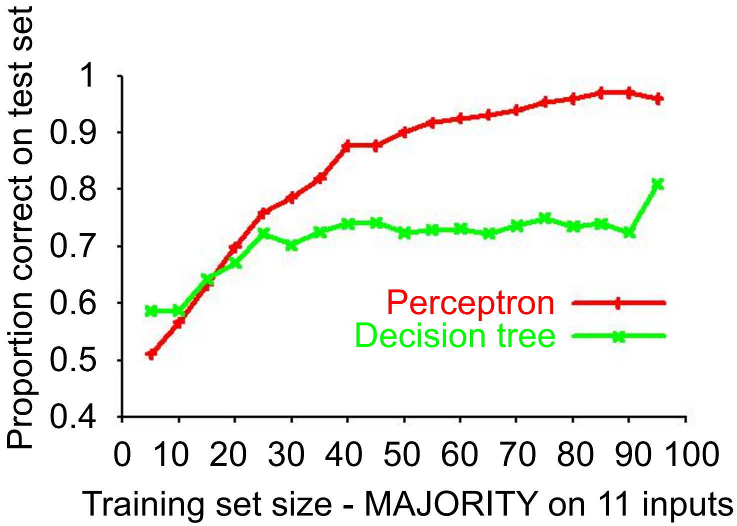
Simple weight update rule:

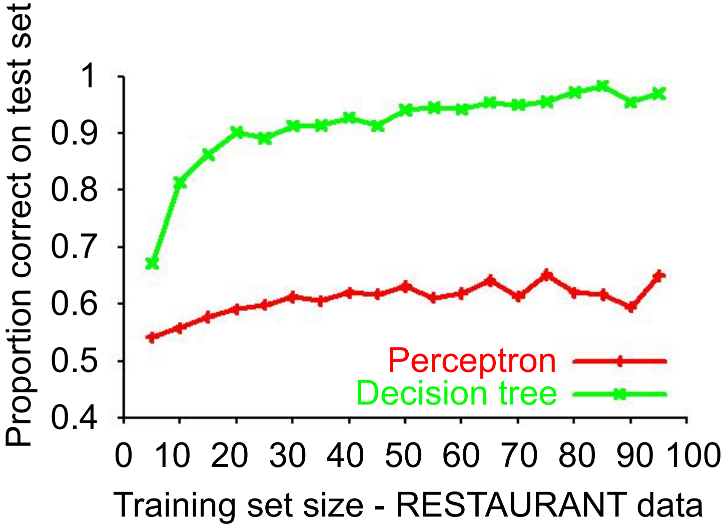
$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

E.g., +ve error ⇒ increase network output
⇒ increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning contd.

Perceptron learning rule converges to a consistent function
for any linearly separable data set



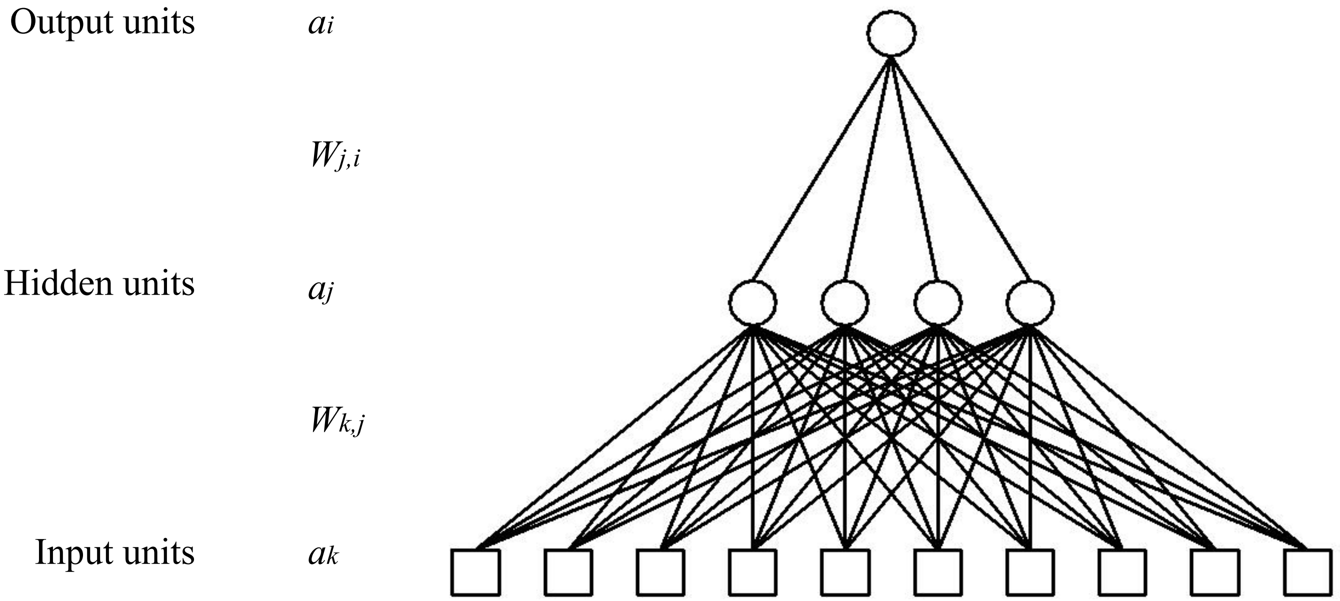


Perceptron learns majority function easily, DTL is hopeless

DTL learns restaurant function easily, perceptron cannot represent it   Why?

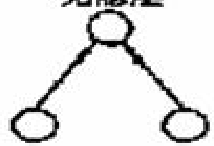
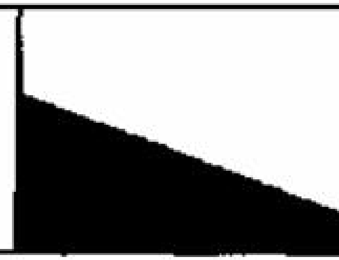
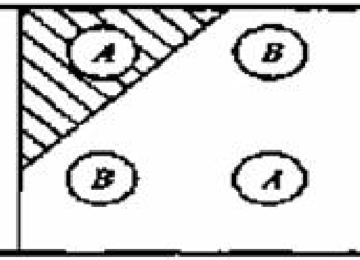
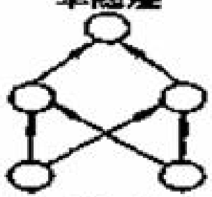
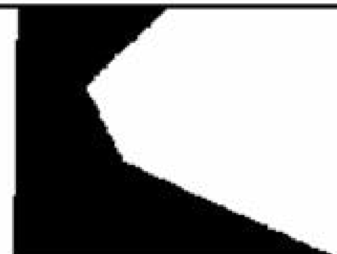
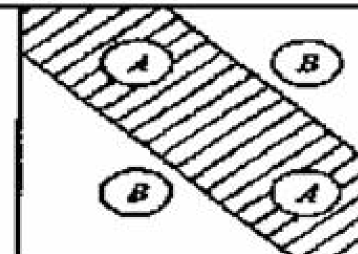
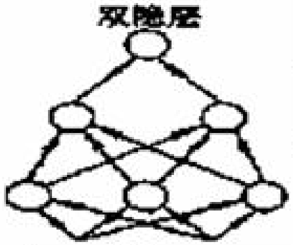
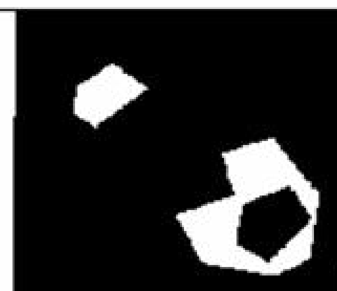
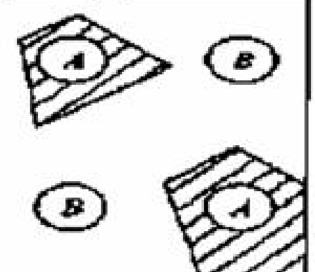
Multilayer perceptrons

Layers are usually fully connected;
numbers of hidden units typically chosen by hand



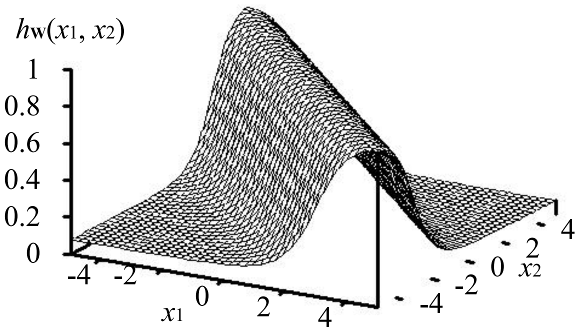
Expressiveness of MLPs

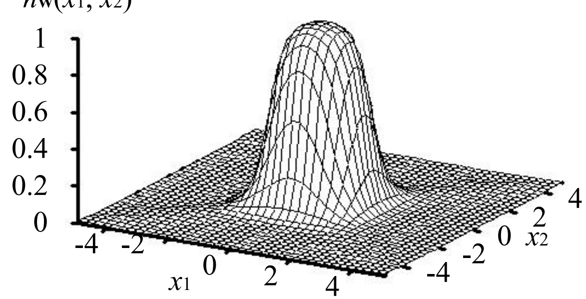
Structure and decision region for Perceptrons

结构	决策区域类型	区域形状		异或问题
无隐层 	由一超平面分成两个			
单隐层 	开凸区域或闭凸区域			
双隐层 	任意形状(其复杂度由单元数目确定)			

Expressiveness of MLPs

All continuous functions w/ 2 layers, all functions w/ 3 layers





Combine two opposite-facing threshold functions to make a ridge

Combine two perpendicular ridges to make a bump

Add bumps of various sizes and locations to fit any surface

Proof requires exponentially many hidden units (cf DTL proof)

Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: back-propagate the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{i\,i} \Delta_i$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \,.$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2,$$

where the sum is over the nodes in the output layer.

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}}$$

$$= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$

$$= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i$$

Back-propagation derivation contd.

$$\frac{\partial E}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{k,j}}$$

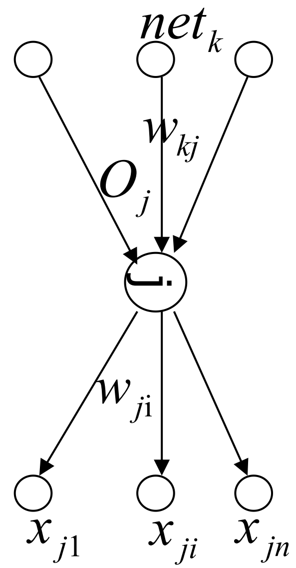
$$= -\sum_i (y_i - a_i)\, g\,(in_i)\frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}}\left(\sum\nolimits_j W_{j,i} a_j\right)$$

$$= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j)\frac{\partial in_j}{\partial W_{k,j}}$$

$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j)\frac{\partial}{\partial W_{k,j}}\left(\sum\nolimits_k W_{k,j} a_k\right)$$

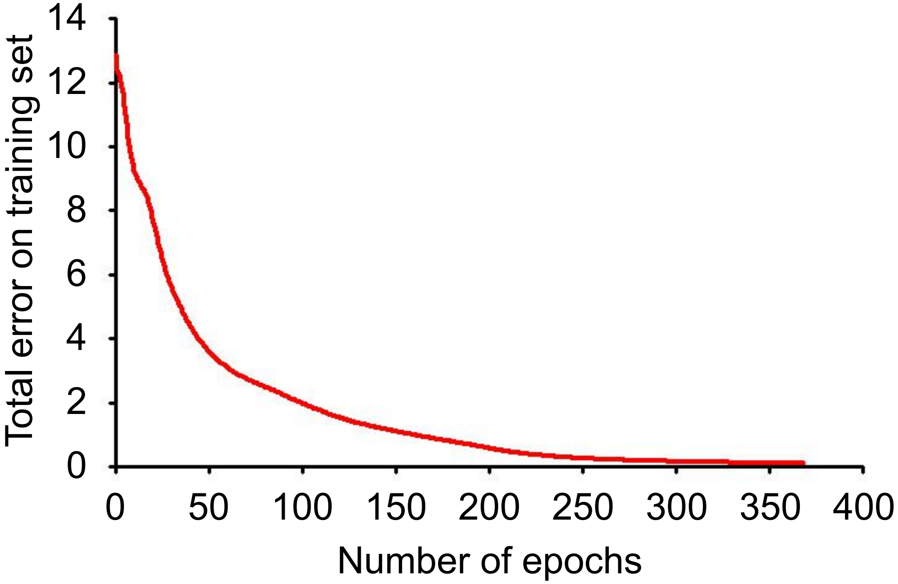
$$= -\sum_i \Delta_i W_{j,i}\, g'(in_j)\, a_k = -a_k \Delta_j$$



Back-propagation learning contd.

At each epoch, sum gradient updates for all examples and apply

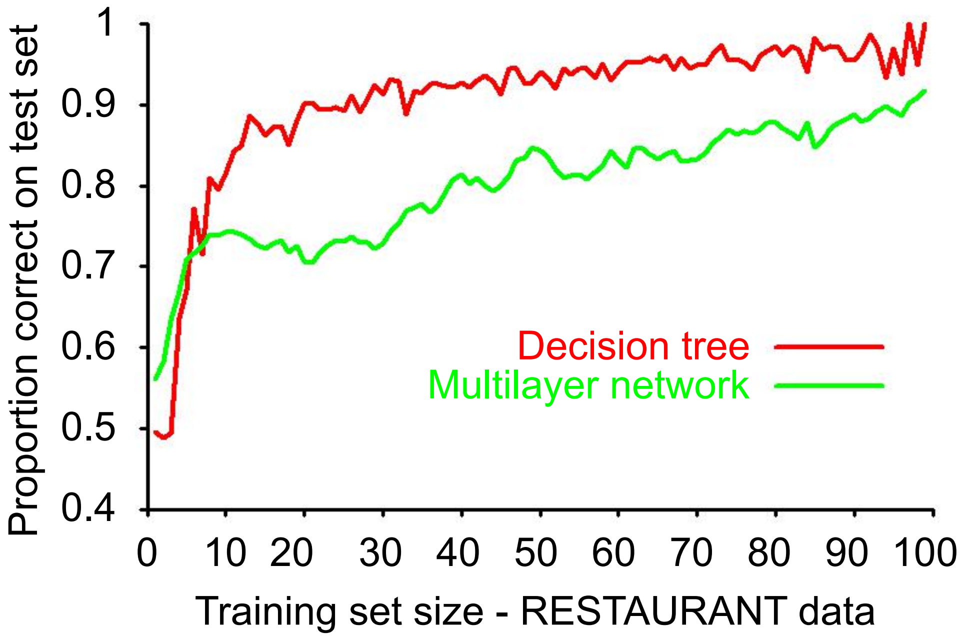
Training curve for 100 restaurant examples: finds exact fit



Typical problems: slow convergence, local minima

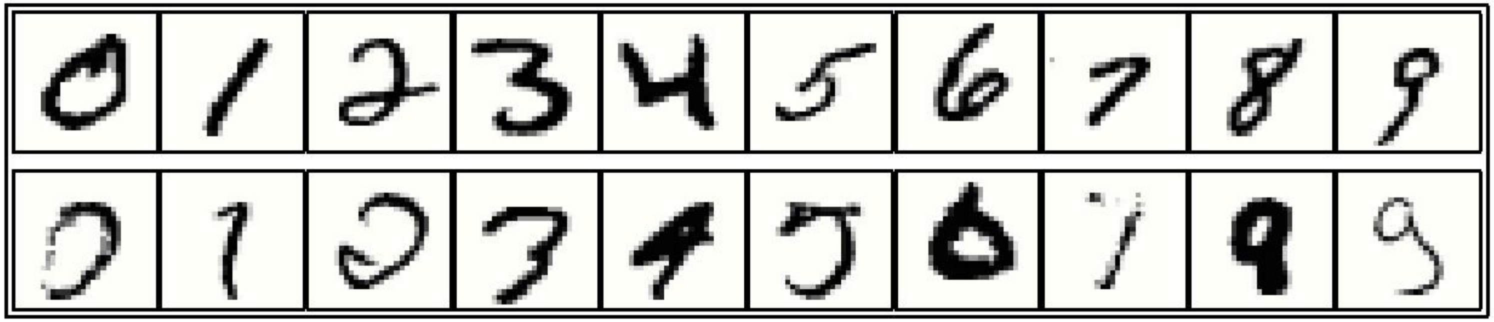
Back-propagation learning contd.

Learning curve for MLP with 4 hidden units:



MLPs are quite good for complex pattern recognition tasks,
but resulting hypotheses cannot be understood easily

Handwritten digit recognition



3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error

Current best (kernel machines, vision algorithms) ≈ 0.6% error

Summary

Most brains have lots of neurons; each neuron ≈ linear–threshold unit (?)

Perceptrons (one-layer networks) insufficiently expressive

Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error ***back-propagation***

Many applications: speech, driving, handwriting, fraud detection, etc.

Engineering, cognitive modelling, and neural system modelling subfields have largely diverged

Back-propagation Algorithm: Pros and Cons
Pros: good learning algorithm for MLP
Cons: local optima, time consuming

Assignment

– Readings: Chap 18.6, 18.7
– Exercises: 18.19
– Additional exercises:
  - Exercise 1: Compute a NN to realize function XOR based on Perceptrons.
  - Exercise 2: Prove the formula of BP Algorithm based.

*Handed in next Tuesday