Introduction to advanced learning algorithms in artificial neural networks

AIMS

- Discuss the use of multi-layered networks to overcome the limitations of the perceptron, and possible applications of such networks.
- Explain the problems posed to learning by the credit assignment problems caused by correct responses not being provided for each neuron, or for each time-step or input stimulus.
- Discuss how reinforcement learning and genetic algorithms overcome the problems of temporal credit assignment.
- Discuss the relative biological plausibility of these learning algorithms

READING

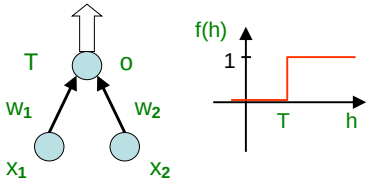
Books 1,2,4,6,14.

· Rumelhart D E, Hinton G E & Williams R J, (1986) `Learning internal representations by error propagation', In Rumelhart, D. E. and McClelland, J. L. (Eds.) Parallel Distributed Processing, 1 151-193 MIT Press.

· Patarnello S & Carnevali P (1989) `A neural network model to simulate a conditioning experiment' Int. J. Neural Systems' 1 47-53.

· Barto A G & Sutton R S (1981) `Landmark learning: an illustration of associative search', Biological Cybernetics 42 1-8.

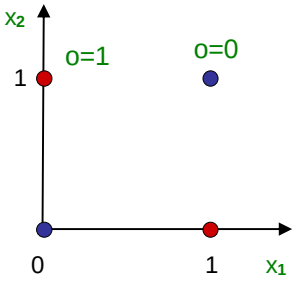
Non-linearly separable problems



$w_i \rightarrow w_i + \varepsilon x_i^n (t^n - o^n)$

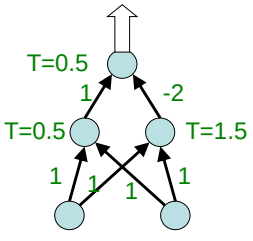
Perceptrons perform linear discrimination and can't solve 'non-linearly separable' problems (Minsky & Papert, 1969)

e.g. XOR 'exclusive OR'

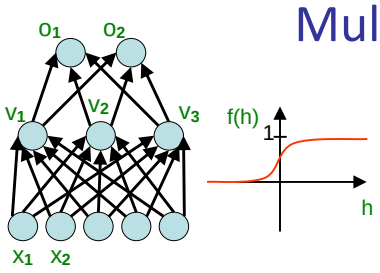


o	x1: 0	x1: +1
x2: 0	0	1
x2: +1	1	0

'Hidden' processing units can give more power, e.g:



Multi-layer perceptrons



Delta rule:

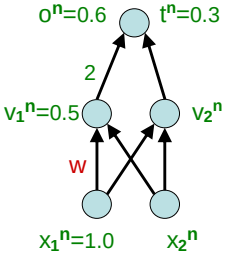
$w_{ij} \rightarrow w_{ij} + \varepsilon\, x_j^n\, (t_i^n - o_i^n)$

Q: How train connection weights if the 'internal representation' is not known (i.e. t_i^n not known for 'hidden layer', x_j^n not known for output?

A: if the neurons use a smooth (continuous) transfer function, changing a connection weight from any (active) neuron anywhere in the network will change the output firing rate slightly. If the output o_i^n is now closer to its target value t_i^n then the change was good..

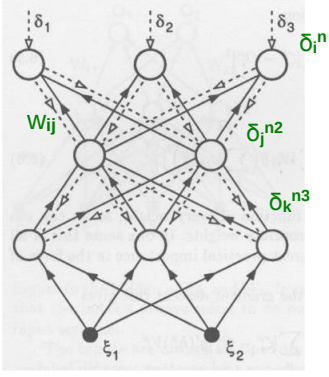
But note: The effect of changing a connection weight depends on the values of the errors $\delta_i^n = (t_i^n - o_i^n)$, connections w_{ij} and activations v_i further up in the network..

Example:

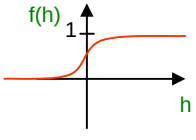


If w decreases the error for input pattern n reduces

The 'generalised delta rule' or 'error back-propagation'



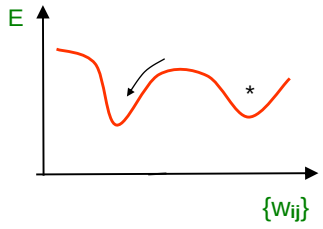
- Present input pattern n work out all other activations (a 'forward' pass)
- Work out the errors made by the outputs; $\delta_i^n = (t_i^n - o_i^n)$
- Work out the contributions of units in layer lower down to output errors (a 'backward' pass). This depends on connections w_{ij} to output, giving a 'delta' for that layer: $\delta_j^{n2} = \sum_i w_{ij}\, \delta_i^n$. Similarly, for the next layer: $\delta_k^{n3} = \sum_j w_{jk}\, \delta_j^{n2}$.
- Change connection weights in each layer using the delta rule.
- Repeat for next pattern.
- Repeat for whole training set many times.



This rule is equivalent to working out the total squared error on the whole set of training patterns: $E = \sum_n \sum_i (t_i^n - o_i^n)^2$ and changing each weight to reduce this.

Rumelhart, Hinton & Williams (1986).

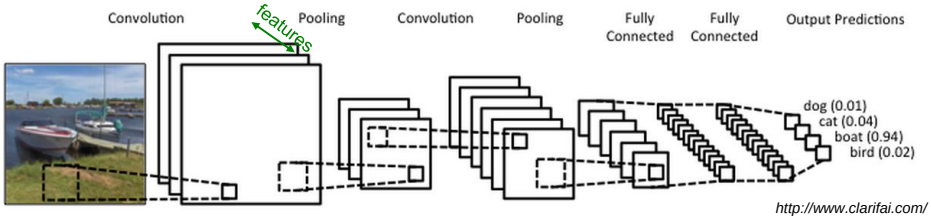
Problems with the 'generalised delta rule' or 'error back-propagation'



- *Local minima* in error*: you can't be sure it will find the best set of weights – maybe just finds those that can't be improved by any small change.
- Learning and generalisation depends on the choice of *architecture*
- Not *biologically plausible* since the connection weights are changed using *non-local* information.

However, with enough neurons and enough training data (and recent increases in processing power), these networks outperform all other methods in some tasks..

Image classification with deep convolutional neural networks



Krizhevsky et al (2012) trained *large* deep network (9 layers, 750K neurons, 60M wts) to recognise 1000 categories in *large* training set of 1.2 M labled images (ImageNet, 224x224xRGB). Error B-P, 90 cycles through training set x 2048 translations and reflections (6 days on 2 GPUs).

The network significantly outperformed all alternatives in categorising new images

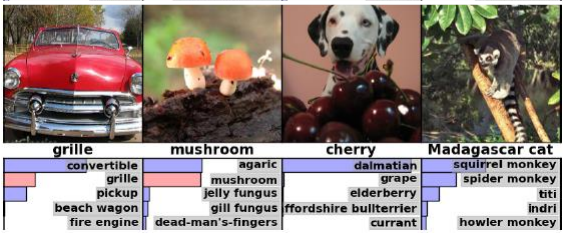
Krizhevsky, Sutskever & Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*.

Image classification with deep convolutional neural networks

Example outputs

correct:

'incorrect':

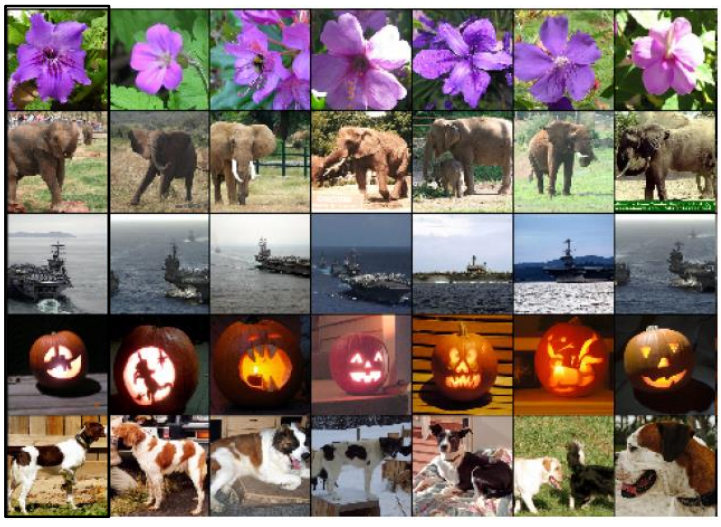


Krizhevsky, Sutskever & Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*.

Image classification with deep convolutional neural networks

Test images

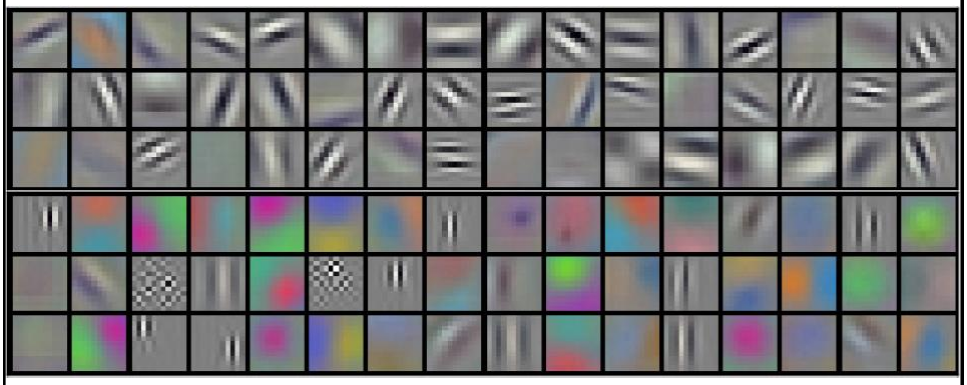
Training images with most similar activity in last hidden layer



Krizhevsky, Sutskever & Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*.

Image classification with deep convolutional neural networks

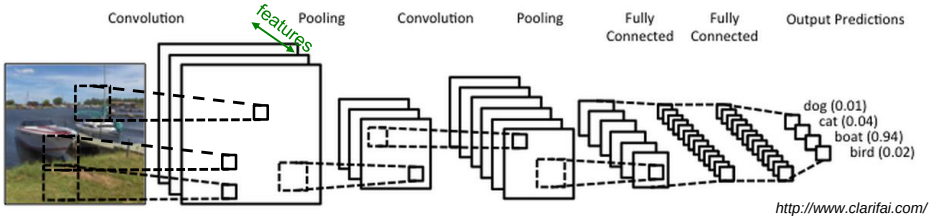
Example of convolutional connection weights learned in first hidden layer



..bears some similarity to neural responses in the early stages of the visual system.

Krizhevsky, Sutskever & Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*.

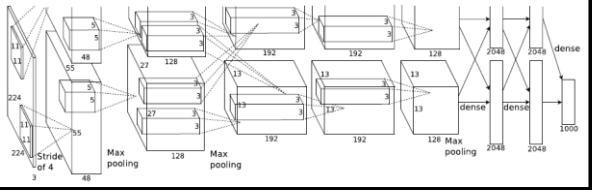
Deep convolutional neural networks – details that make them work



http://www.clarifai.com/

'Convolution': use BP, but make all patches of weights identical (average weight changes across corresponding connections in each patch), each sheet of neurons codes for the same "feature" independent of location in the image.
'Pooling': take the max or average output of local units (smoothing/ shrinking representation)
'Dropout': set randomly set neurons = 0 in 50% of trials during training, use all x0.5 during testing – learns more robust sets of connections, reduces 'overfitting'
Threshold linear transfer function: f(h) = h if h>0, f(h) = 0 if h<0. Speeds up learning.
'Augmented dataset': 2048 translations and reflections; random noise in pixel values on each cycle - improves 'generalisation'.

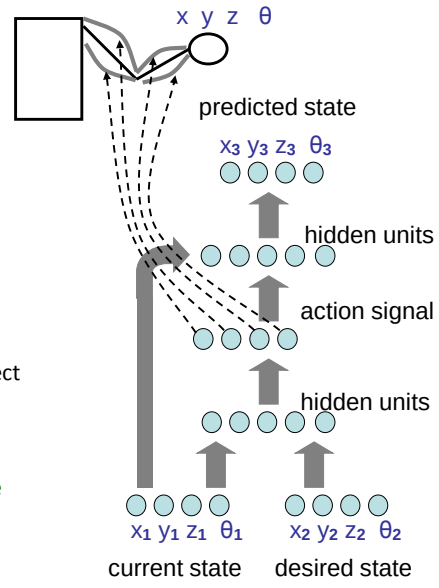
Enough layers, enough training data, enough processing power..



Example of a sensorimotor control model: using error back-propagation.

Problem: generate action signal to achieve a desired state from given current state (e.g. play a stroke in tennis).

1. Train upper network to predict the (sensory) effect of an action signal (i.e. a *forward model*). Use B-P of the difference between predicted and observed effect.
2. Train the lower network to produce the correct action signal to achieve a desired (sensory) effect (an *inverse model*). Fix the connection weights in the upper network to use it as *feedback controller*. Use B-P of the difference between desired and observed effect to change weights in the lower network.



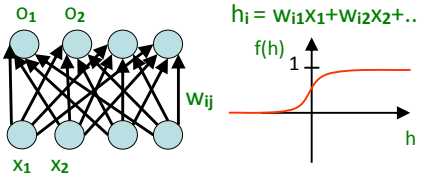
Jordan & Rumelhart (1992) Forward Models. *Cognitive Science 16, 307-354*

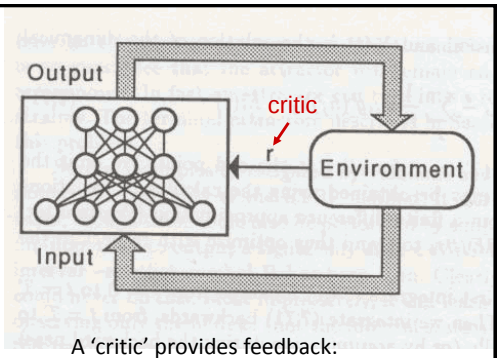
Introduction to reinforcement learning

Critic provides single reward/ penalty signal r^n for each output o^n
- but not for each output unit: a 'credit assignment problem'.

Associative reward-penalty 'A_{RP}'
Barto & Sutton (1981).

Output units are *probabilistic*: o_i=0 or 1, where $f(h_i)$ is the *probability* of o_i = 1.
Allows exploration of possible output values.





A 'critic' provides feedback:

$$r^n = \begin{cases} 1 \text{ 'reward'} \\ -1 \text{ 'penalty'} \end{cases} \quad \text{according to } o^n$$

Define 'target' output:

$$t_i^n = \begin{cases} o_i^n & \text{if } r^n=1 \\ 1-o_i^n & \text{if } r^n=-1 \end{cases}$$

and use 'delta rule': $w_{ij} \rightarrow w_{ij}+\varepsilon x_j^n \delta_i^n$
where $\delta_i^n = (t_i^n - f(h_i^n))$

i.e. if doing well: use delta rule to 'reinforce' current h_i to make current output more likely, otherwise 'penalise'.

Using A_{RP} to navigate

4 inputs: proximity of 4 landmarks in N, S, E, W (via smell).
4 outputs: actions N, S, E, W
Critic: 'did I get closer to tree?' (via smell).

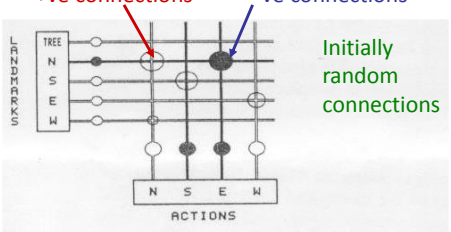


Fig. 3. The ASN controlling locomotion in the spatial environment. The five input pathways are labelled vertically on the left according to the landmarks to which they respond. The shaded input pathway *N* indicates that the ASN is near the north neutral landmark. The four output pathways controlling actions are labelled horizontally at the bottom according to the direction of movement they cause. The shaded output elements indicate that a southeast movement is being made. The associative matrix weights are displayed as circles centered on the intersections of the horizontal input pathways and vertical output pathways. Positive weights are shown as hollow circles, and negative weights are shown as solid circles

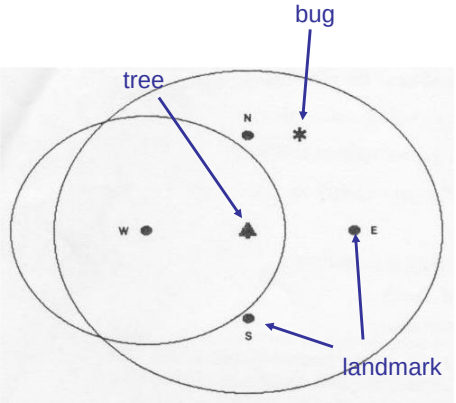


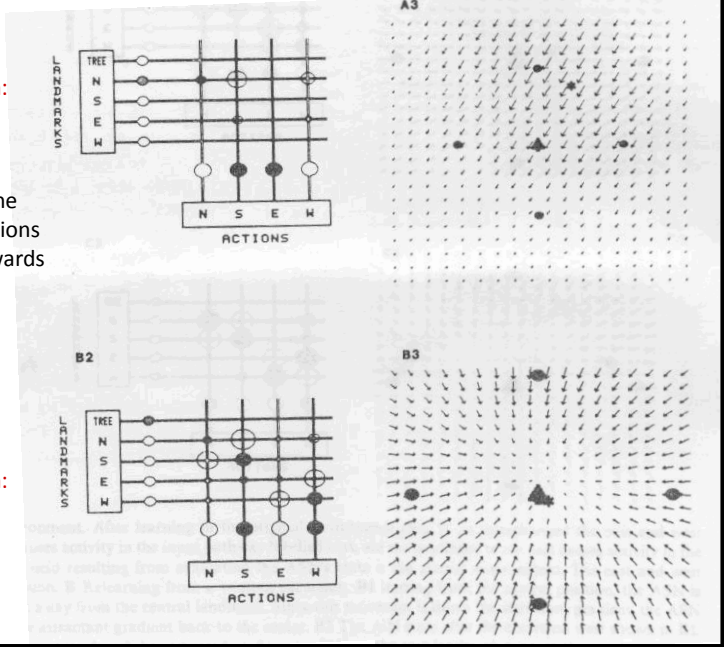
Fig. 2. A spatial environment consisting of a central landmark (shown as a tree) surrounded by four other landmarks (shown as disks). Each landmark possesses a distinctive "odor" which can be sensed at a distance. Odor distributions decrease linearly from their associated landmarks and become undetectable at ellipses. The asterisk shows the location of the ASN

Using A_{RP} to navigate, cont.

Before exploration:

The network learns the input-output associations required to move towards the goal from all start positions.

After exploration:

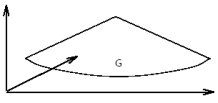


Introduction to reinforcement learning, cont.

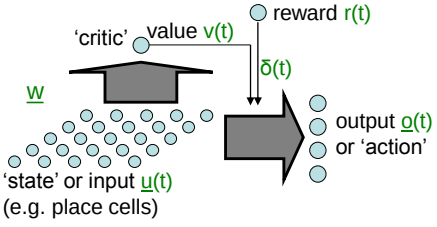
- The critic in the example of A_{RP} has an easy task due to the smell of the goal (tree): it is easy to know if a move has been good or bad (smell gets stronger or weaker).
- The general problem of reinforcement learning is to develop a 'critic' network that learns to provide feedback at each time-step even though the environment provides feedback rarely (e.g. when the tree is reached), i.e. how do you know which outputs were good or bad: the problem of 'temporal credit assignment'.
- This involves developing an idea of the total expected future reward, and how this changes after making an action (i.e. if it changes for the better or for the worse)..
- A role for the Dopamine system? – see Lecture on Reinforcement Learning.

Intro. to 'temporal difference' or 'sequential reinforcement' solution to the temporal credit assignment problem

In A_{rp} the strength of the smell of the tree provides reinforcement r(t) to evaluate for any action (i.e. output $\underline{o}(t)$): does it leads closer to the goal or not?



If reinforcement r(t) is intermittent (e.g. only when goal is reached), a 'critic' learns an '*evaluation function*': the value v of each state (or input) $\underline{u}$ is *the expected future reward from that state* (given how actions are usually made).



The change in value v(t+1)-v(t) + any reward r(t) is used to evaluate any action $\underline{o}(t)$ so output weights can be modified as in A_{rp} (using $\delta(t) = v(t+1)-v(t)+r(t)$: if $\delta(t)>0$, action $\underline{o}(t)$ was good).

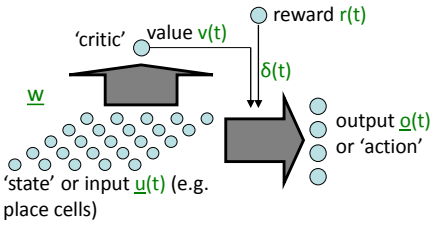
But how to learn v? A simple learning rule creates a set of connection weights $\underline{w}$ so that $v(t) = \underline{w}.\underline{u}(t)$. This is: $\underline{w} \rightarrow \underline{w} + \varepsilon\delta(t)\underline{u}(t)$, i.e. you can also use δ to learn weights for the critic!

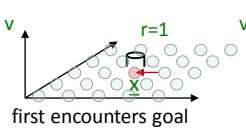
Introduction to temporal difference learning (spatial example).

Use $\delta(t) = v(t+1)-v(t)+r(t)$ to evaluate actions

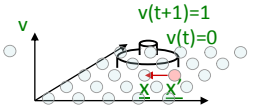
Use $v(t) = \underline{w}.\underline{u}(t)$ where $\underline{w} \rightarrow \underline{w} + \varepsilon\delta(t)\underline{u}(t)$ to learn value function.

Dopamine signals $\delta(t)$?

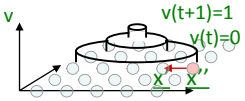




first encounters goal at location x: r=1 so increase weight to critic from place cell at x. Now v(x)=1

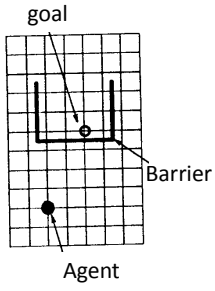


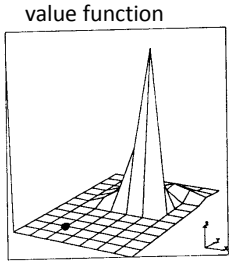
next time go to x from x', v(t+1)-v(t)=1, so increase weight to critic from place cell at x'.
Now v(x')=1



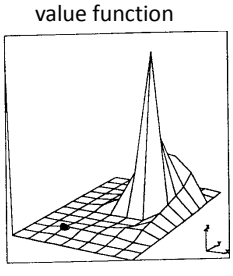
next time go to x' from x'', v(t+1)-v(t)=1, so increase weight to critic from place cell at x'', and so on..

Temporal difference learning is a slow 'trial and error' based method. Development of own value function overcomes problem of temporal credit assignment





After initial exploration (e.g. 100 runs to goal)

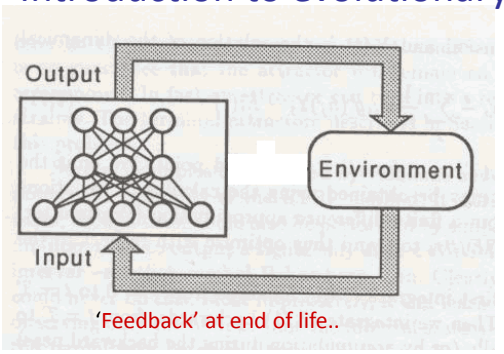


After 2000 runs to goal

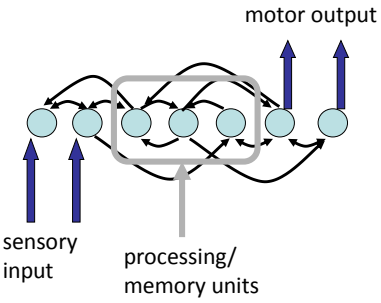
Initial actions will be random (takes a long time to get to goal), actions improve as a result of learning in the Arp action network (but a random element is required to enable ongoing improvement: i.e. finding new shorter routes).

Dayan (1991) Neural Information Processing Systems 3. p464-70. Morgan Kaufmann

Introduction to evolutionary algorithms



'Feedback' at end of life..



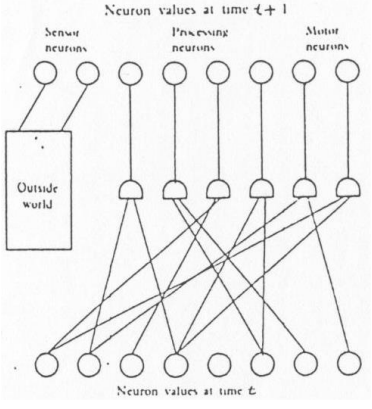
Simplest model:

- 'Genetic' code determines connection weights
- Simulate behaviour of several networks with random weights over long period,
- 'Reproduce' most successful networks with small random changes to weights
- Repeat for many 'generations'.

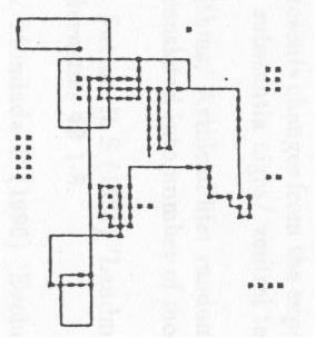
More complex models:

- Include numbers & locations neurons, rules for development in genetic code.
- 'Sexual reproduction of codes.
- Allow for non-transferable weight modification (learning) during life..

Paternello & Carnevali's 'ameoba' (1989)

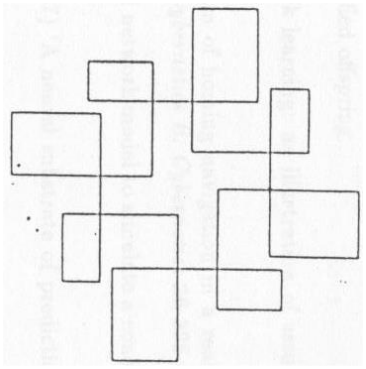


Threshold logic units (0/1 output)
Connections found by genetic algorithm.
Sensory input1: 'see' food ahead
Sensory input2: 'smell' food near
Motor output1: left-leg, (0,0)=rest, (1,0)=L
Motor output2: right-leg, (0,1)=R, (1,1)=forward

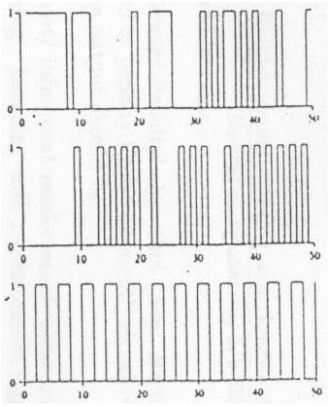


Learns to find food in a grid world after 100s of generations.
No. of offspring depends on amount of food eaten during life.
Pattern of connectivity varied.

Paternello & Carnevali's 'ameoba' cont.



After many generations, it learns to search in loops of the same size as the typical patches of food..



It develops various types of processing units including a 'clock'.