Reinforcement Learning 15 March 2007
Lecture 20

COMBINING KOHONEN NETS AND REINFORCEMENT LEARNING

- Another method of generalisation
- Generalising to continuous states and actions
- Andrew Smith's method

WHAT SHOULD A COMBINED METHOD BE ABLE TO DO?

•Generalise over large state spaces with a SOFM

•Continuously valued states

•Extend to generalising over action spaces

•Continuously valued actions

•Adaptable, not fixed, representation of action space

•Reuse information on learned actions – actions must be tried out to find their effect, so this information is expensive. Make the best use of it.

•Multiple actions for each state – so can learn Q-values

•Andrew Smith: Dynamic generalisation of continuous action spaces in reinforcement learning: a neurally inspired approach. PhD Thesis, University of Edinburgh 2001.
http://www.era.lib.ed.ac.uk/items-by-author?author=Smith%2C+Andrew+James See also A.J.Smith: Applications of the self-organising map to reinforcement learning. Neural Networks 15, 1107–1124, 2002.
http://www.psychology.mcmaster.ca/∼smitha/PAGE_PAPERS/neural-networks-2002.pdf

TEST TASK

•Simple obstacle avoidance using a Khepera simulator (2D earlier version of Webots).

•See figures of robot and environment

•6 sensors: 0 – 1 (full off, i.e. no obstacle visible – full on, i.e. obstacle immediately in front)

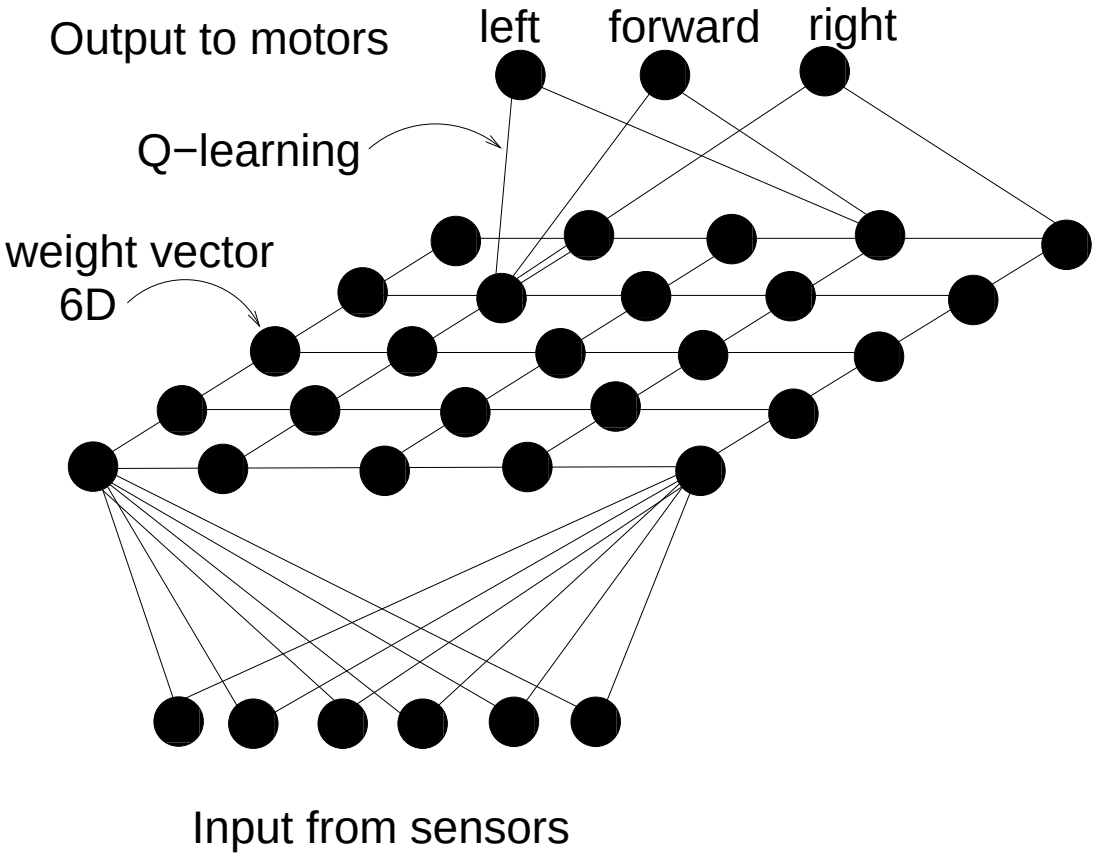
•State vector = 6D

•2 motors: left, right: 0 – 1 (full off to full on).

•Actions: either 2D $< M_L, M_R >$ continuous actions, or 3 discrete actions: left, forward, right

•Default behaviour is move forward if each sensor $<$0.2. If obstacle, do obstacle avoidance.

SIMPLE ARCHITECTURE



•Input vector = 6D (one from each sensor), fully connected to each node on grid

•Map/grid space = 5x5 Kohonen map. Each node has a 6D weight vector. Each node connected to each action node

•Actions: 3 fixed actions – turn right on spot, forward at full speed, turn left on spot

•Learn Q-values between each map node and each action

SMITH'S ALGORITHM

1. Present input $\mathbf{x}_t$ to Kohonen map
2. Find winning unit $i_t^* = \arg\min_i \|\mathbf{x}_t - \mathbf{w}_i\|$ – smallest Euclidean distance
3. Take action a_t – with best Q if exploiting, random if exploring – ϵ-greedy, ϵ decreases with time
4. Run robot on a_t, get reward r_t, new input vector $\mathbf{x}_{t+1}$
5. Get return of state-action pair h timesteps ago:

$$R = r_{t-h} + \gamma r_{t-h-1} + \gamma^2 r_{t-h+1} + \ldots + \gamma^h r_t$$

6. Update $Q(i_{t-h}^*, a_{t-h})$ towards R, learning rate α
7. Update SOM using input vector randomly chosen from a short term buffer. Update winner plus neighbours, learning rate and neighbourhood size decay with time
8. Go to 1.

Reward $r_t = 1$ if no sensor > 0.2 – no obstacle
Reward $r_t =$ -S1 - 0.5 x S3 - 0.25 x S5 -S2 - 0.5 x S4 - 0.25 x S6 – otherwise

DETAILS OF ALGORITHM

Why the particular **return**?: it's a fixed horizon discounted sum – effectively a truncated SARSA update.

We assume the effect of an action is limited to a fixed time window

$h = 4$, $\gamma = 0.95$, α decays from 1 to 0

Why the **short-term buffer** for updating the input map? Kohonen works better if successive inputs are not correlated. Best if they're evenly distributed about the space. So use a 100-long buffer and update with randomly chosen states from this buffer.

The algorithm learns only when the obstacle avoidance algorithm is active – except can still learn with buffered stimuli and may still be updating Q values whose r's are still coming in

So we learn the Kohonen map and the Q values at the same time.

RESULTS

See results diagrams/graphs

Sample paths show better obstacle avoidance behaviour after learning

Average reward against time increases

Compare with a handcrafted strategy:

turn right if S1 + 0.5 x S3 + 0.25 x S5 >
S2 + 0.5 x S4 + 0.25 x S6

turn left otherwise

Exploration parameter has most effect on performance:

- the faster we exploit, the faster the performance increases at the beginning
- but slower exploitation leads to better long-term performance

WHAT THE GRID* MAP REPRESENTS

The bars in the diagram represent the 6D weight vectors

The circles represent the action with the highest Q-value: white = left, black = right

•Left turns taken when right sensors activated

•Right turns taken when left sensors activated

•Forward action not activated

•Neighbouring units respond to similar inputs

•Two actions meet at "ambiguous" categories, e.g. row three has situations that only just need a left run

In weight space (6D so can only plot two dimensions at one time):

•Both front sensors activated together – S1 and S2

•Side units activated singly – obstacles encountered one one side or the other

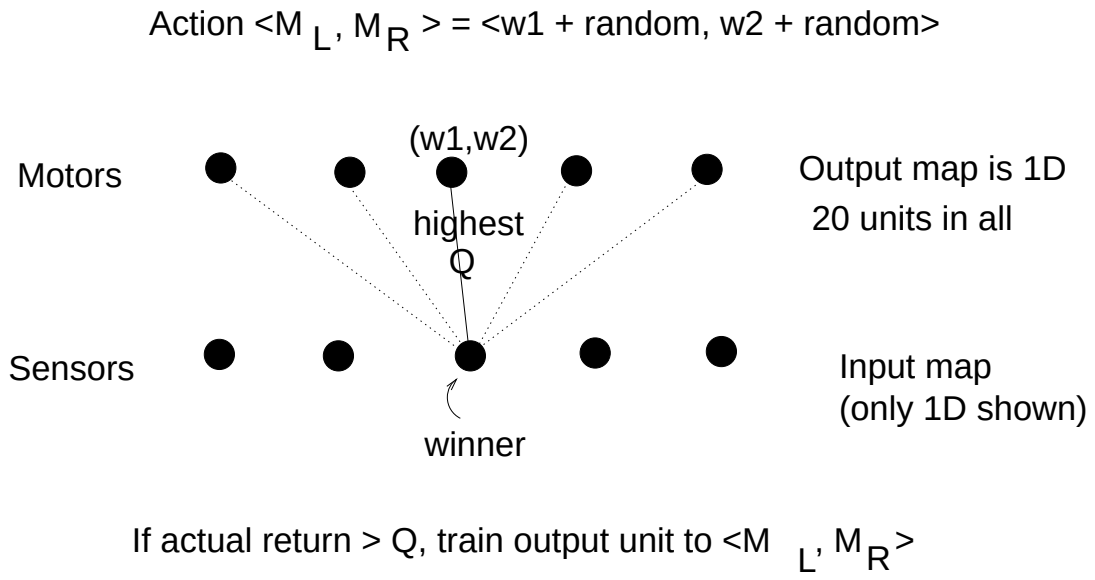
•Lower figures coloured according to output with highest Q-value: above diagonal have more activity on left sensor than right (so 'turn right' has highest Q-value)

* Smith refers to the 'Input' map rather than 'Grid' map

EXTENSIONS

•Can also **use** the topology: in step 6, update **all** Q values towards R, weighted by neighbourhood function

•Can also use a Kohonen map in the output, i.e. action, space. Get the winning input unit, find the action unit with the highest Q connected to it, carry out that action (modified with addition of small random amount)



Also update the Q

•Can also update **all** Qs towards the winner's R, weighted by neighbourhood functions in input **and** output space:

$$Q = Q + \alpha N(input) * N(output)[\text{winner's } R - Q]$$

SEE ALSO

Philip Sterne: Reinforcement Sailing, MSc thesis 2005.
http://www.inf.ed.ac.uk/publications/thesis/online/IM040206.pdf

Applying reinforcement learning to a sailing controller, learning to sail downwind and to tack. Used A. Smith's double SOFMs as one of the control architectures investigated.