Reinforcement Learning

Dr J. Charles Sullivan

What is RL?

- So far we have looked at
  - Supervised Learning
  - Unsupervised Learning
- Reinforcement Learning could be considered to be somewhere in-between these two extremes
- It is more like the way we (and animals in general) learn
- Think about how you train a dog
  - Reward good behaviour
  - Punish bad behaviour (maybe)

Learning by Trial and Error

- Reinforcement learning is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal.

- The learner (or agent) is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward **by trying them**.

- In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards.

- These two characteristics (trial-and-error search and delayed reward) are the two most important distinguishing features of reinforcement learning.

Defined by Problems

- Reinforcement learning is concerned with *goal-directed learning from interaction*
- RL is defined not by characterizing learning methods, but by characterizing a learning problem
- Any method that is well suited to solving that problem, we consider to be a reinforcement learning method

Learning by trial and error

- In the conventional framework, the agent does not initially know what effect its actions have on the state of the world, nor what immediate payoffs its actions will produce

- In particular, it does not know what action is ***best*** to do

- Rather, it tries out various actions at various states, and gradually learns which one is best at each state so as to *maximize its long run payoff*

- The agent thus acquires what is known as a closed-loop control policy, or a rule for choosing an action according to the observed current state of the world

Not the standard Engineering Approach

- From an engineering perspective, the most natural way to learn the best actions would be for the agent to try out various actions in various states, and so learn a **predictive model** of the effects its actions have on the state of the world and of what rewards its actions produce.
  - However, this approach to learning does not seem at all plausible for animals

- **Planning ahead** involves envisioning alternative sequences of actions and their consequences
  - the animal would have to imagine what states of the world would result and what rewards it would receive for different possible sequences of actions
  - is particularly difficult if actions may have stochastic effects, so that performing an action may lead to one of several different possible states.

Learning without a predictive model

- "One of the most surprising discoveries in RL is that there are simple learning algorithms by means of which an agent can learn an optimal policy without ever being able to predict the effects of its actions or the immediate payoffs they will produce, and *without ever planning ahead"*

- "It is also surprising that, in principle, learning requires only a minimal amount of episodic memory:
  - an agent can learn if it can consider together the last action it took, the state when it chose that action, the payoff received, and the current state"

  Dayan and Watkins, 2001

The RL agent

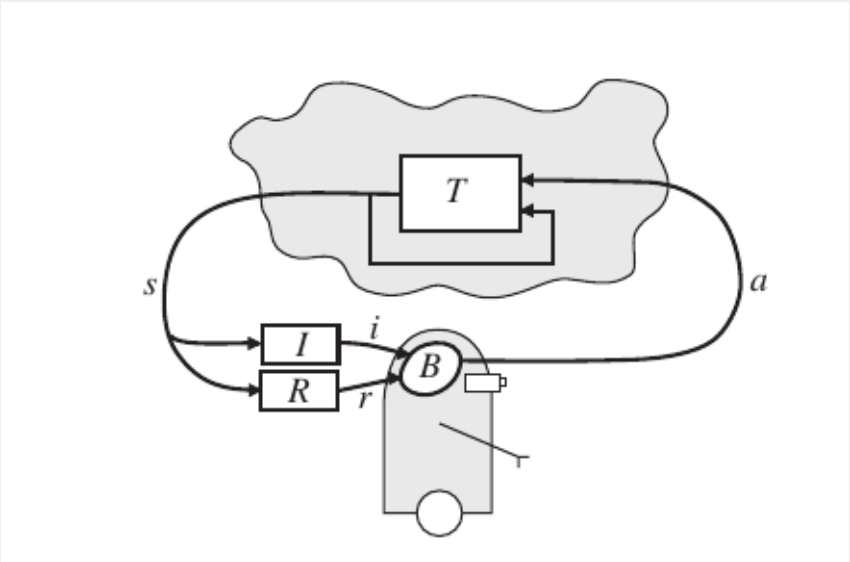
- the basic idea is simply to capture the most important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal

- Clearly, such an agent must be able to ***sense*** the state of the environment to some extent

- and must be able to take ***actions*** that affect the state

- the agent also must have a ***goal*** or goals relating to the state of the environment

Diagram of a RL agent (from Kaelbling)



On each step of interaction:

the agent receives as input, *i,* some indication of the current state, *s*, of the environment

the agent then chooses an action, *a*, to generate as output.

the action changes the state of the environment,

and the value of this state transition is communicated to the agent through a scalar reinforcement signal, *r*.

the agent's behavior, *B*, should choose actions that tend ***to increase the long-run sum of values of the reinforcement signal.***

It can learn to do this over time by systematic ***trial and error***, guided by an action selection policy

Markov Decision Processes

Problems with delayed reinforcement are well modeled as Markov decision processes (MDPs).

An MDP consists of

- a set of states S,
- a set of actions A,
- a reward function $R : S \times A \rightarrow \mathbb{R}$, and
- a state transition function $T : S \times A \rightarrow \Pi(S)$, where a member of $\Pi(S)$ is a probability distribution over the set S (maps states to probabilities).

We write $T(s, a, s')$ for the probability of making a transition from state s to state s' using action a.

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the agent's action.

The reward function specifies expected instantaneous reward as a function of the current state and action.

The model is Markov if the state transitions are independent of any previous environment states or agent actions.

Although general MDPs may have infinite (even uncountable) state and action spaces, we will only discuss methods for solving finite-state and finite-action problems.

Differences compared with Supervised Learning

- Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in machine learning, statistical pattern recognition, and artificial neural networks.
  - Supervised learning is learning from examples provided by a knowledgeable external supervisor
  - Gives gradient information so that learner knows how to improve
- This is an important kind of learning, but alone it is not adequate for learning from interaction.
  - In interactive problems it is often impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act
- In uncharted territory (where one would expect learning to be most beneficial) an agent must be able to **learn from its own experience**

Similarity to EC

- In RL we have a single agent that is trying to learn to 'do better', unlike the population-based approaches of Evolutionary Computation (EC)

- As in EC, there are a number of variants.
  - However, in all cases, the learning algorithm acquires feedback from the environment that is a *scalar* value of 'goodness' or 'fitness'

- The point here is that, like EC, although an agent is 'rated' by interaction with the environment in some way, the number fed back does not, normally, give a clue *what to do better* to get a higher score, just what that score is for the current situation

.

Adaptive Optimal Control

- Adaptive control is also concerned with algorithms for improving a sequence of decisions from experience
- Optimal control aims to extremize a functional (e.g. a cost or performance measure) of the controlled system's behaviour
- nearly always *indirect* methods in which control updates are computed from an estimated system model
- although the dynamic model of the system is not known in advance, and must be estimated from data,
  - the structure of the dynamic model is fixed,
  - leaving model estimation as a parameter estimation problem
- An established method for solving these problems is Dynamic Programming (Bellman,1957)
- Reinforcement Learning can be considered to be a *direct* approximation to DP

Exploration vs Exploitation

- all Reinforcement Learning (RL) algorithms need (at least) two component parts,
  - an **'exploration'** part
    - responsible for managing the exploration of the environment to find new knowledge
  - and an **'exploitation'** part
    - responsible for directing those actions that are based on the knowledge gained to date

- In most cases, these two parts are active at the same time, and there is a gradual 'handover' as more knowledge is gained

Rewards, credit assigment and value maps

- In the more complex versions of these architectures, reinforcement can be delayed, i.e., the agent must take a sequence of actions in order to attain a reward

- In these cases, ‘credit assignment’ is a problem to be solved
  - In other words: how best should the credit gained from the environment be distributed amongst the actions that led to its acquisition?

- In all of the major examples of these algorithms, some sort of ‘map’ of the environment is gradually built up in the exploitation part of the algorithm
  - it represents the agent’s knowledge of the ‘**value**’ of each environmental state that the agent could find itself in.

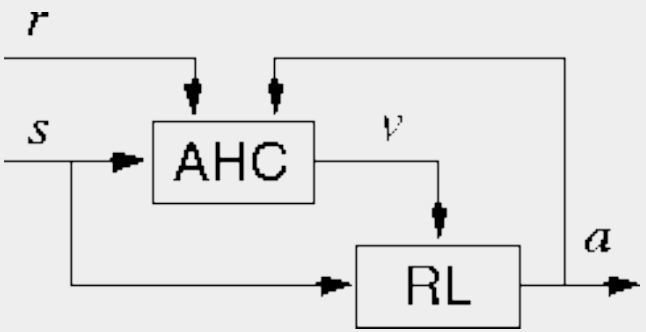
Types of RL Architecture

- We will look at two principal types of RL implementation

  - Adaptive Heuristic Critic (AHC):
    - in which a ‘value’ map V(s) of environmental states is built up

  - Q-learning:
    - in which the map is of the value of each of the possible ***transitions*** from one state to another, or of state, action pairs Q(s,a)

Adaptive Heuristic Critic



- two components: a critic (AHC), and a reinforcement-learning component (labeled RL), modified to deal with multiple states and non-stationary rewards

- instead of acting to maximize instantaneous reward,
  it will be acting to maximize the ***heuristic value, v***, that is computed by the critic

- The critic uses the real external reinforcement signal to learn to map states to their expected discounted values given that the policy being executed is the one currently instantiated in the RL component.

AHC learning algorithm

s is the agent's state before transition,

a is its choice of action,

r the instantaneous reward it receives,

s' its resulting state.

The ***value of a policy*** is learned using Sutton's TD(0) algorithm which uses the update rule

$$V(s) := V(s) + \alpha(r + \gamma V(s') - V(s))$$

Whenever a state ***s*** is visited, its estimated value is updated to be closer to $r + \gamma V(s')$

since r is the instantaneous reward received and V(s') is the estimated value of the actually occurring next state.

The key idea is that $r + \gamma V(s')$

is a sample of the value of V(s), and it is more likely to be correct because it incorporates the real ***r***.

If the learning rate α is adjusted properly (it must be slowly decreased) and the policy is held fixed, TD(0) is guaranteed to converge to the optimal value function.

Temporal Difference (TD) learning

- TD(0) is the simplest example of Temporal Difference learning
  - The (0) implies that the update rule only depends on the current state and reward

- The more general case is known as TD(λ) where λ is an integer indicating a number of past states to which credit is assigned

- Although TD(λ) is a more powerful method credit assignment method in dynamic learning problems, it is considerably more computationally expensive

Q-learning

- Q-learning combines the critic and actor in one component which learns separate predictions for each action.
- Proposed by Watkins (1989) who proved convergence under certain conditions
- Better-developed theory than AHC
- Has been found empirically to converge faster in many cases
- Requires memory and computing proportional to the number of state-action pairs
- One way to deal with this problem as the state/action space increases is to use a parameterised structure such as a neural network to store the estimate of Q

K-armed Bandit Problem

The simplest possible reinforcement-learning problem is known as the ***k*-armed bandit problem**

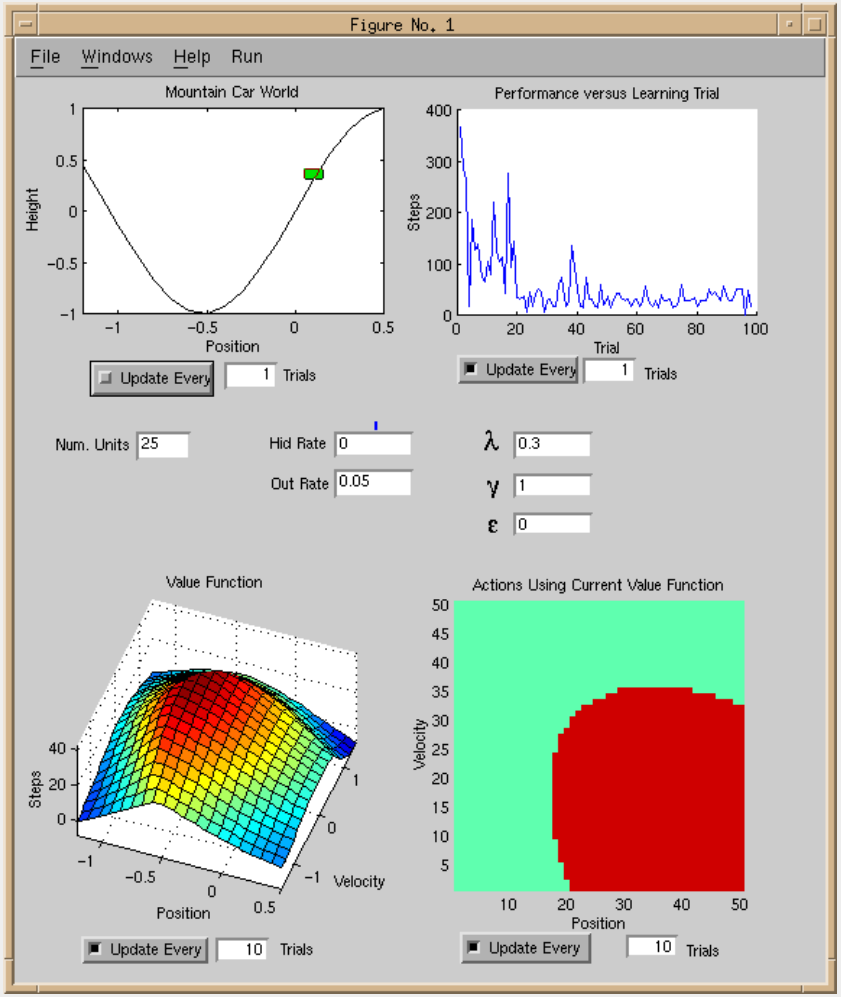
- a room with a collection of k gambling machines ("one-armed bandits")
- The agent is permitted a fixed number of pulls, h.
  - Any arm may be pulled on each turn
- The machines do not require a deposit to play;
  - the only cost is in wasting a pull playing a suboptimal machine
- When arm i is pulled,
  - machine i pays off 1 or 0, according to probability parameter p_i ,
  - payoffs are independent events and the p_i s are unknown.
  - what should the agent's strategy be?
- illustrates the fundamental tradeoff between **exploitation and exploration**.
  - the agent might believe that a particular arm has a fairly high payoff probability;
  - should it choose that arm all the time,
  - or should it choose another one that it has less information about, but seems to be worse?

.

MATLAB Examples

Charles Anderson is one of the leading researchers in the field of RL and its application in optimal control problems.

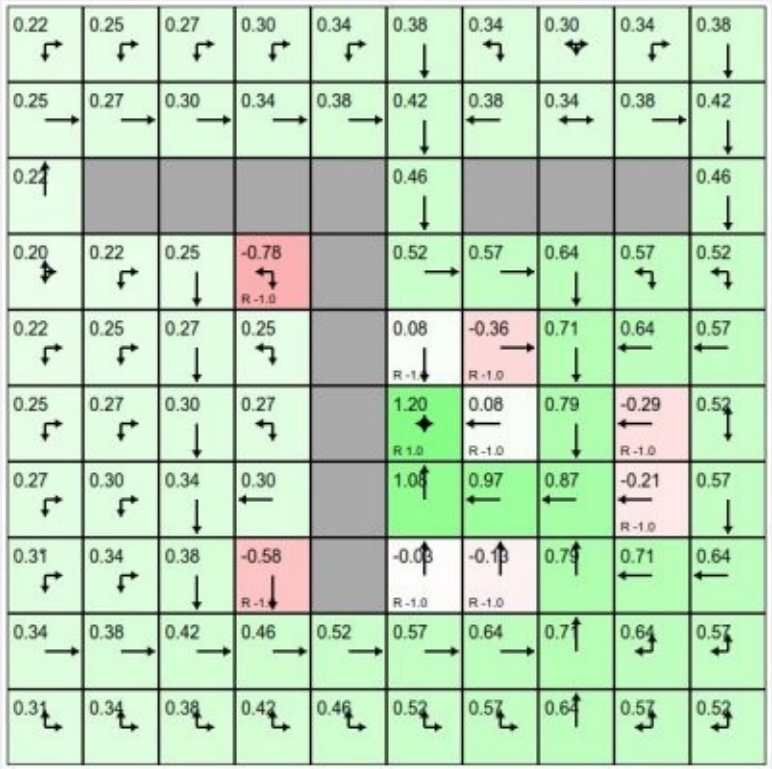
He has written a number of programs which illustrate the paradigm, including a MATLAB GUI which demonstrates the principles using a 2D simulation of the "Mountain Car" problem:



http://www.cs.colostate.edu/~anderson/res/rl/matlabpaper/rl.html

Grid-world examples

http://cs.stanford.edu/people/karpathy/reinforcejs/



REINFORCEjs is a Reinforcement Learning library that implements several common RL algorithms supported with fun web demos

(includes Deep Q Learning)

Applications

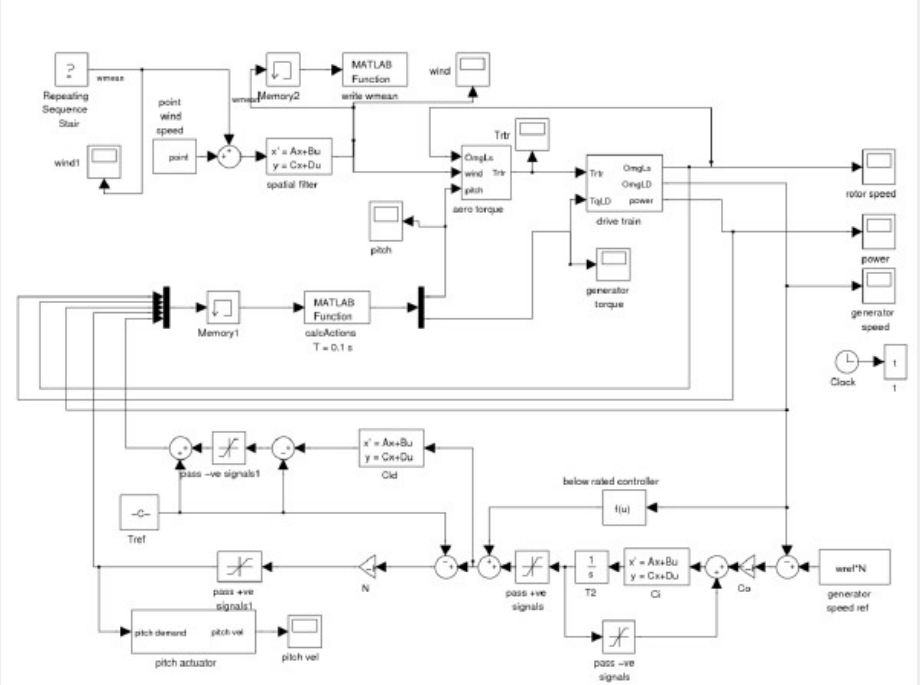
- Because RL agents can learn without expert supervision, the type of problems that are best suited to RL are complex problems where there appears to be no obvious or easily programmable solution. Two of the main ones are:

- **Game playing** - determining the best move to make in a game often depends on a number of different factors, hence the number of possible states that can exist in a particular game is usually very large.
  - To cover this many states using a standard rule based approach would mean specifying an also large number of hard coded rules.
  - RL cuts out the need to manually specify rules, agents learn simply by playing the game.
  - For two player games such as backgammon, agents can be trained by playing against other human players or even other RL agents.
  - Of course, this is not a serious application domain but often used to demonstrate an ability to solve 'hard' problems

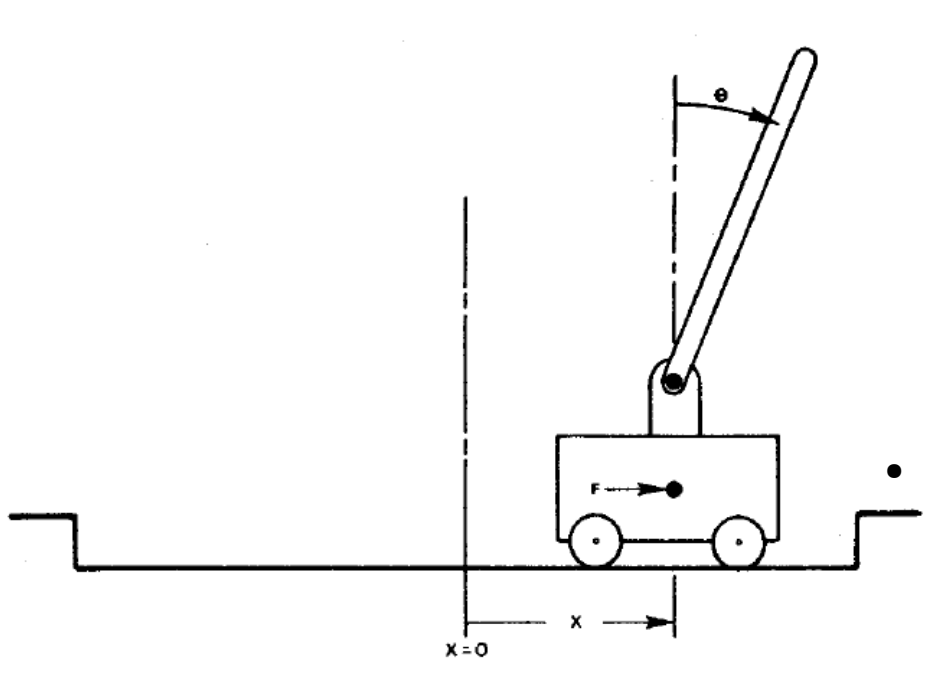
- **Control problems** - such as elevator scheduling. Again, it is not obvious what strategies would provide the best, most timely elevator service. For control problems such as this, RL agents can be left to learn in a simulated environment and eventually they will come up with good controlling policies.
  - One advantage of using RL for control problems is that an agent can be retrained easily to adapt to environment changes, or trained continuously while the system is online, improving performance all the time.
  - A good example is Anderson's use of RL for on-line optimisation of Wind Turbine conrol (Anderson,2010)

Wind Turbine Control

- Control of wind turbines complicated by
  - Variations of wind velocity
  - Complex dynamics
- Inaccuracies in models of interaction of blades with air and mechanical and electrical dynamics lead to loss of efficiency
- Anderson (2010) combined RL agent with existing controller
- Increased power by 6%

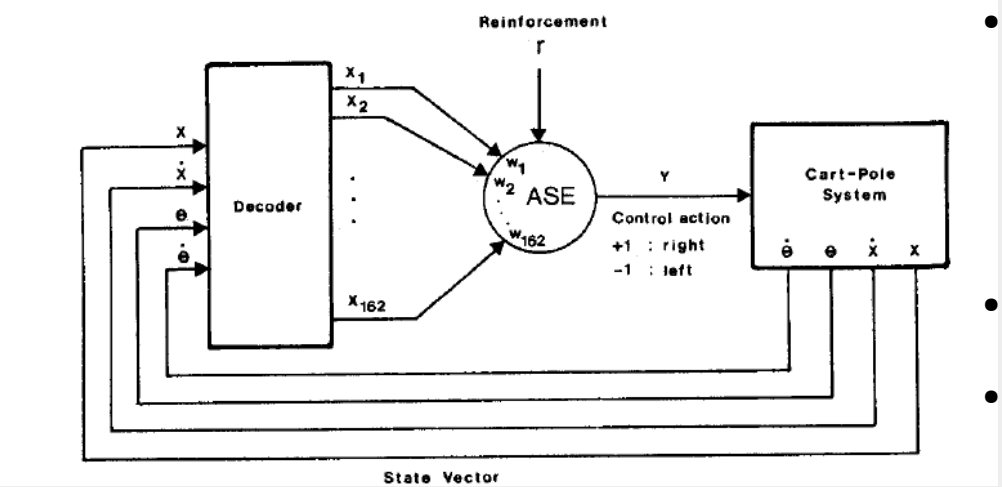


Cart-pole balancing



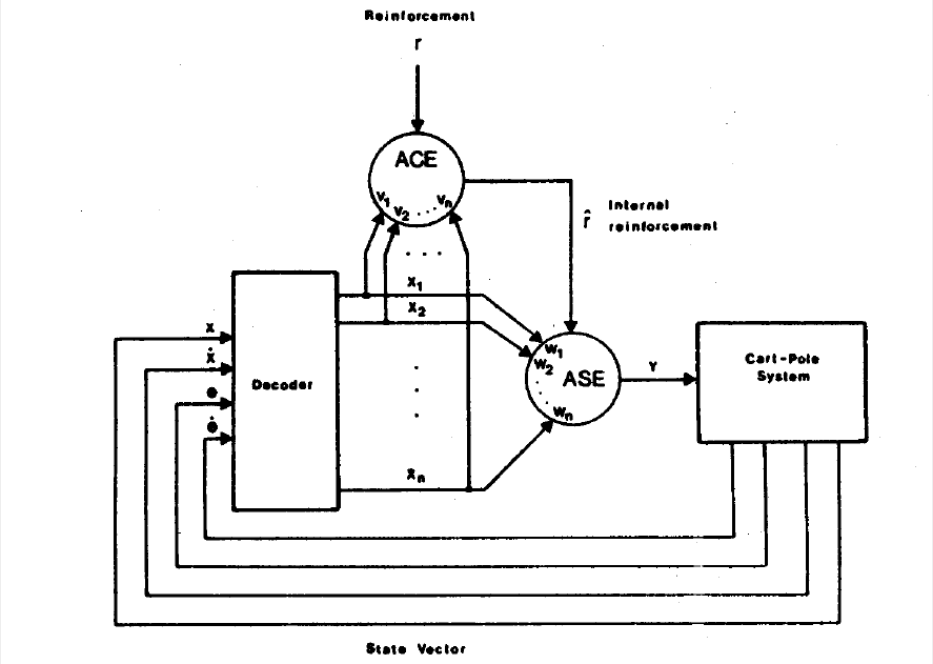
Barto, Sutton and Anderson (1983), "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," IEEE Trans. Syst., Man, Cybern., Vol. SMC-13, pp. 834--846, Sept.--Oct. 1983

Associative Search Element



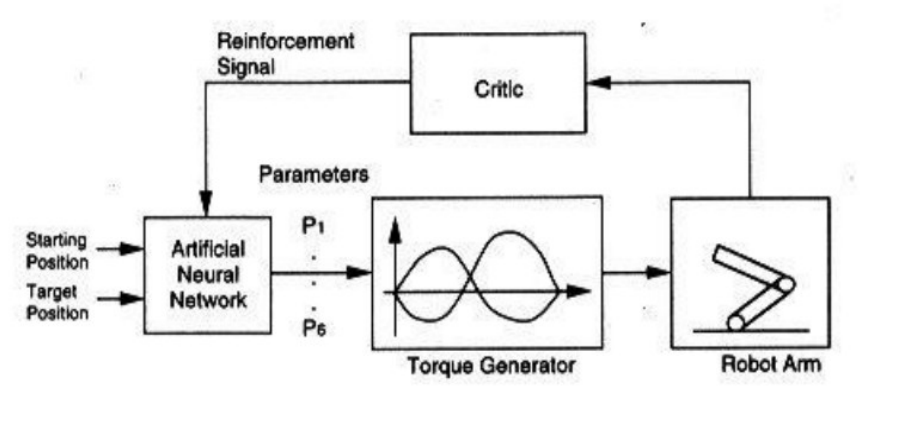
- ASE input from current state vector
  - by decoder that produces vector of zeros with single one indicating which 'box' is current
- ASE output determines force
- Reinforcement is constant throughout trial and becomes -1 to signal failure

Adaptive Critic Element



- The ACE receives external reinforcement signal and
  - computes an improved reinforcement signal for the ASE
- Its job is to store in each "box" a ***prediction of the reinforcement*** that can eventually be obtained from the environment
- Permits learning to occur throughout the pole-balancing rather than just on failure

The Distal Error Problem



Lipitkas (1993) used reinforcement learning to solve the problem of learning to move a robot arm end effector from given starting positions to desired target positions

Given inputs coding starting and target positions of the end-effector, the neural network controller learns to provide correct parameters to a torque generator which generates time-varying torque signals to the arm in open-loop mode.

The reinforcement signal evaluates the success of each movement after its completion

This is an example of the *distal error problem* (Jordan & Rumelhart 1992) : the end-effector position error is distal to the output of the controller that is to be learned and it does not tell the controller how to change its behaviour to reduce it.

A supervised learning approach would require a forward model to translate these errors into error vectors but this is not necessary for reinforcement learning.

It required a few thousand movements with different starting and target points to be able to move with reasonable accuracy to new positions.

TD-gammon

- Tesauro (1992) combined a RL algorithm with an artificial neural network to learn to play backgammon, which has approximately 10^{20} states. With this many states it is impossible ever to experience more than a small fraction of them

- the program learned to play far better than any previous program, and even played at the level of the world's best human players

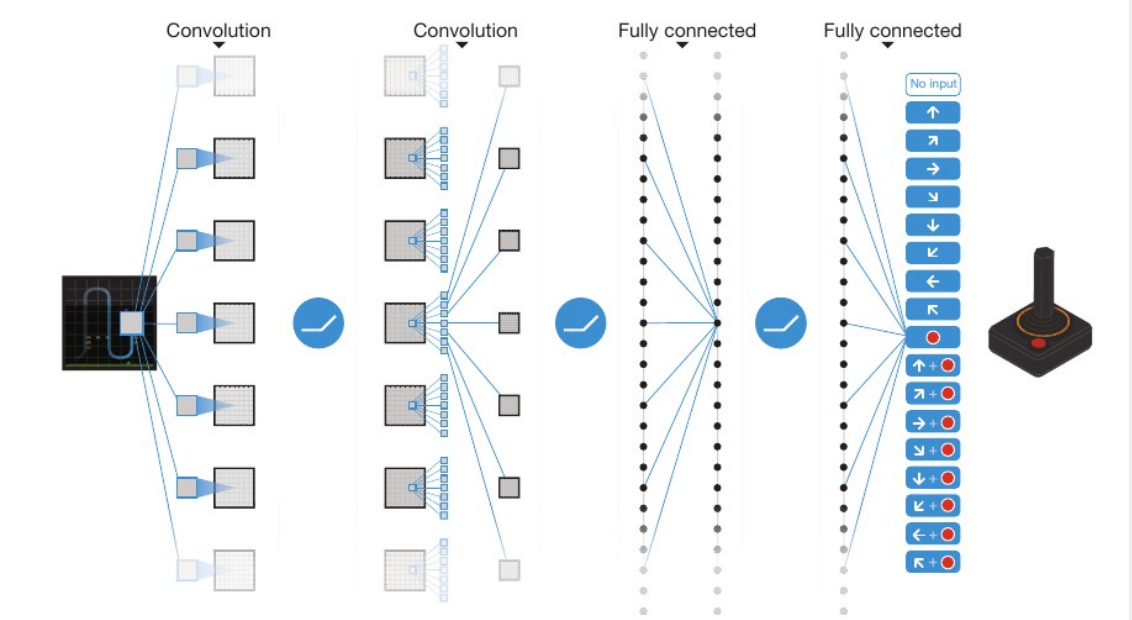
- the neural network provided the program with the ability to generalize from its experience, so that in new states it selected moves based on information saved from similar states faced in the past, as determined by its network

- how well a reinforcement learning system can work in problems with such large state sets is intimately tied to how well it can **generalize** from past experience

- it is in this role that we have the greatest need to combine supervised learning methods with reinforcement learning. Neural networks are one way to do this.

Deep Reinforcement Learning

- In 2013 a small company in London called DeepMind uploaded their pioneering paper “Playing Atari with Deep Reinforcement Learning” to Arxivhttp://arxiv.org/abs/1312.5602
- It has been hailed since then as the first step towards general artificial intelligence – an AI that can survive in a variety of environments, instead of being confined to strict realms such as playing chess
- first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning
- The model is a convolutional neural network
  - trained with a variant of Q-learning
  - input is raw pixels
  - output is a value function estimating future rewards.
- applied method to seven Atari 2600 games
- outperforms all previous approaches on six of the games and surpasses a human expert on three of them



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

- In contrast to TD-Gammon utilizes a technique known as **experience replay**
  - store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$
  - in a data-set $D = e_1, ..., e_N$, pooled over many episodes into a replay memory
  - During the inner loop of the algorithm apply Q-learning updates, or minibatch updates, to samples of experience $e \sim D$, drawn at random from the pool of stored samples
- After performing experience replay the agent selects and executes an action according to an ε-greedy policy.
- using histories of arbitrary length as inputs to a neural network can be difficult:
  - instead works on fixed length representation of histories produced by a function φ

Deep Recurrent Q-Learning

Deep Recurrent Q-Learning for Partially Observable MDPs https://arxiv.org/abs/1507.06527v4

Matthew Hausknecht, Peter Stone (Submitted on 23 Jul 2015 (v1), last revised 11 Jan 2017

- Deep Reinforcement Learning has yielded proficient controllers for complex tasks
  - However, these controllers have limited memory and rely on being able to perceive the complete game screen at each decision point.

- *Deep Recurrent Q-Network* (DRQN) adds recurrency to a Deep Q-Network (DQN) by replacing the first post-convolutional fully-connected layer with a recurrent LSTM.
  - although capable of seeing only a single frame at each timestep, successfully integrates information through time and replicates DQN's performance on standard Atari games and partially observed equivalents featuring flickering game screens.
  - when trained with partial observations and evaluated with incrementally more complete observations, DRQN's performance scales as a function of observability.
  - when trained with full observations and evaluated with partial observations, DRQN's performance degrades less than DQN's.
- given the same length of history, recurrency is a viable alternative to stacking a history of frames in the DQN's input layer
- while recurrency confers no systematic advantage when learning to play the game, the recurrent net can better adapt at evaluation time if the quality of observations changes.

AlphaGo beats Pro Go Player

- A master player of the game Go has won his first match against a Google computer program, after losing three in a row in a best-of-five competition.
- DeepMind chief executive Demis Hassabis said AlphaGo "played itself, different versions of itself, millions and millions of times and each time got incrementally slightly better".
- "It learns from its mistakes," he told the BBC
- BBC Radio 4 “Today” 14/03/2016

Brain's reward system earns researchers €1 million prize

https://www.newscientist.com/article/2123578-brains-reward-system-earns-researchers-e1-million-prize/

- Wolfram Schultz, Peter Dayan, and Ray Dolan have been awarded the €1 million Brain Prize by Denmark's Lundbeck Foundation.
- The prize recognises researchers who have made vital contributions to understanding how our brains work
- "Nature has endowed us with a fantastic system for optimising our behaviour," said Dayan at a press briefing in London. Dayan is now working on applying the logic of decision making seen in the dopamine system to artificial intelligence algorithms. "That's how you get computers to make predictions," he said

Further Reading

See Richard Sutton's book on reinforcement learning at:

http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

one of the most important authorities on this topic, his book is excellent

A more concise introduction is given by Dayan and Watkins:

http://www.gatsby.ucl.ac.uk/~dayan/papers/dw01.pdf

Alternatively, try:http://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html

for a review of these algorithms from another well-known researcher in this area, Leslie Kaelbling.

You could also visit http://reinforcementlearning.ai-depot.com/Main.html for a plethora of other stuff on RL, including code examples and papers.

Lastly, http://www.cse.unsw.edu.au/~cs9417ml/RL1/introduction.html

gives a quite nice concise description and includes a Java applet which illustrates the idea of Q-learning