Adaptive Computation

Neural Networks II: Reinforcement Learning

Jonathan Shapiro

External Faculty, Santa Fe Institute

Department of Computer Science, University of Manchester

June 18, 2002

Reinforcement Learning

References:

- “Reinforcement Learning: An Introduction”, R. Sutton and A. Barto, MIT Press, 1998 (available on-line at `http://www-anw.cs.umass.edu/~rich/book/the-book.html`.
- “Reinforcement Learning: a Survey”, L. P. Kaelbling and M. L. Littman, on-line at `http://www-2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html`
- ‘Machine Learning”, Tom M. Mitchell, McGraw-Hill, 1997, chapter 13.

What is Reinforcement Learning?

We have seen **Supervised learning** (learning from a teacher):

- Learning from examples labeled with the correct responses, actions, etc.
- I.e. feedback from environment is immediate, and indicates correct action.

Now, consider **Reinforcement learning** (learning from critic):

- Learning where only the *quality* of the responses or actions are known (e.g. good/bad), not what the correct responses are.
- I.e. feedback from environment is evaluative, not instructive.
- In some situations, the reinforcement information may be available only sporadically or periodically.

Types of reinforcement problems

Ways in which the feedback from the environment can be less informative.

The immediate reward, deterministic case: It is unknown which component of the response led to the reward or penalty.

The immediate reward, stochastic case: The best action may not always lead to a positive outcome.

The delayed reward case: The learner may receive reinforcement signal only after a sequence of actions.

Learner influences inputs: The actions of the learner may influence the inputs seen later. Thus, the learner may chose to *explore*, making actions which may lead to new inputs, or may *exploit*, trying to optimize reinforcement signal received based on current knowledge.

Credit Assignment

Reinforcement learning is harder than supervised learning, because there is missing information about *which* component of the behavior produced the reinforcement signal.

Structural credit assignment problem: When there is immediate reinforcement, but there are many components to the response or action, it is not known which component action caused the result.

Temporal credit assignment problem: When there is delay of reward/penalty, a sequence of actions may be required before there is a result. Which of those actions lead to the result?

Why is reinforcement learning important?

Fundamental learning paradigm in animal learning

Thorndike's Law of Effect (1911)

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur. . . .

- Language acquisition
- Learning to walk, control movements, etc.
- Learning social skills

Perhaps related to *implicit learning*

Many problems in Agent-based modeling are reinforcement learning situations.

How do reinforcement learning algorithms work

Two ingredients seem important in most reinforcement algorithms

Search: An extra level of search is required to find the correct action (in addition to the search required to learn the association between the input and the action).

A heuristic to guess the correct action: and thereby turn the problem into a *supervised learning problem*. This removes the credit assignment problem.

Method I

- Search on responses (or actions),
- reinforce those which lead to positive outcomes,
- disassociate from those which lead to negative outcomes.

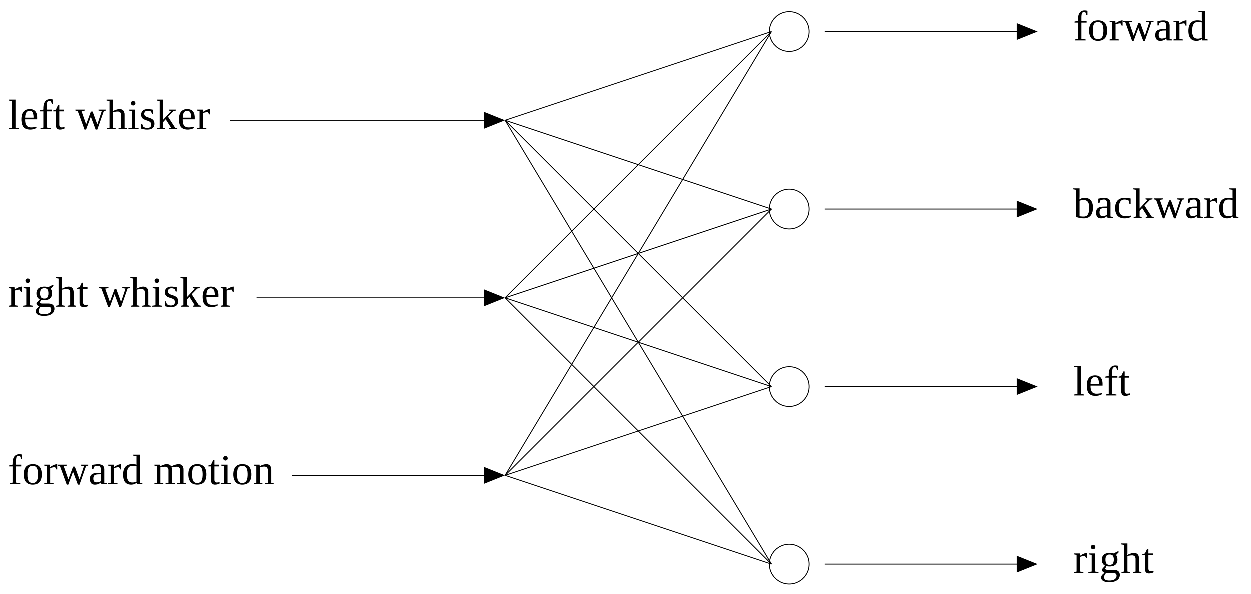
Example — a simple robot controller

Reference: Nehmzow, UMCS TR 94-11-1, 1994.

Controller input: Forward motion detector, two touch sensors (whiskers).

Controller output: Motor control, forward, backwards, left turn, right turn.

Example — a simple robot controller (Continued)



Reinforcement signal: generated internally by “instinct rules” — conditions which the robot wants to satisfy (Edelman).

Learning:

- If instinct rule satisfied, do nothing.
- If instinct rule violated,
  - do action determined by controller neural network for a fixed time (4 s)
  - if instinct rule satisfied, reinforce input-action association
  - if instinct rule not satisfied, try next most active action for slightly longer time (6 s).
  - etc.

Examples of learned actions

- Instinct rule: Keep forward motion detector on; keep touch sensors quiet
- Results: obstacle avoidance.
- Instinct rule: (as above +) if touch sensors quiet for more than 4 s, touch something
- Results: wall-following behavior

Note that this learns the appropriate sensory-motor associations from performance results.

Associative Reward Penalty A_{rp} Networks

Reference: Barto and Anandan, 1985

A simple formalization of the previous approach.

- Probabilistic 0/1 Neurons
- Neuron output: $y_i = \begin{cases} 1 & \text{with probability } p_i = f(\sum_j w_{ij} x_j) \\ 0 & \text{with probability } 1 - p_i \end{cases}$
- Reinforcement signal $r = \begin{cases} 1 & \text{output correct} \\ -1 & \text{output wrong} \end{cases}$

  (variations: sometime 0 is used for wrong output; reinforcement signal can be discrete or continuous.)

A_{rp} Learning Rule

Use gradient descent to minimize

$$E = \left(t_i - f(\sum_j w_{ij} x_j) \right)^2 \tag{1}$$

The assumed target output t_i is determined by the following assumptions

1. If $r = 1$ then reinforce the output the network produced (obviously).
2. If $r = -1$, then do one of the following,
   (a) unlearn the output the network produced, or
   (b) reinforce the *opposite* of what the network produced.

The equations are respectively

$$t_i = r y_i \tag{2}$$

$$t_i = \frac{1+r}{2} y_i + \frac{1-r}{2}(1 - y_i) \tag{3}$$

Points —

- Probabilistic nodes allow exploration of different input-output relations
- Assumed target output (equations 1 and 2 or 1 and 3) turn it into a supervised learning problem.

Method II — Evolutionary methods

- Genetic algorithms and other evolutionary algorithms use reinforcement-type signals to compare one population member with another.
- Evolutionary methods are widely used for reinforcement learning problems. (E.g. "evolutionary robotics".)

Summary

Notice: in both methods heuristic used to guess action:

1. If action led to reward, reinforce that action
2. if action did not, or led to negative reward, guess another action and reinforce that.

What if the best action is unlikely to lead to a reward, but is *more likely* than any other action?

Problem with previous approaches

There are really two tasks:

1. Learn to predict the reward expected after taking an action in a given situation.
2. Find the best policy — what actions should be taken in any situation.

It is useful to separate the two tasks, especially when rewards are probabilistic and may be rare.

Method III — Learning to estimate the value of actions

What are we trying to predict:

Value of the state: *given a policy for choosing actions*

$V(\text{state}|\text{Policy})$,

or

Value of state-action pair: $Q(\text{state}, \text{action})$. Use notation a for action; s for state.

What makes a good policy

During learning, there is a trade-off between

Exploration: find new states which may lead to high rewards;

Exploitation: visit those states which have led to high rewards in the past.

Useful policies:

Greedy policy: pick the state which is predicted to yield the highest value of (discounted) future rewards e.g. best move from state s is $\arg\max_{a'} Q(s, a')$. This maximizes the exploitation.

ϵ**-greedy policy:** Use a greedy policy with probability $1 - \epsilon$; pick a random move with probability ϵ. This allows for some exploration.

Immediate-reward example: Video poker

States: A representation of the 5 cards dealt

Actions: Those cards which are to be discarded and redrawn.

Value: The expected pay-off.

Learning model: A big table of $Q(s,a)$. So no generalization at this stage.

How could we get $Q(s,a)$ to learn the value of the action for each particular hand?

- Play the game repeatedly;
- record the actual payoff at time t, r_t for the state-action pair at time t, (s_t, a_t).
- After each play, update the table,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)\left(1 - \frac{1}{t}\right) + r_t/t. \tag{4}$$

- More generally,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)\,(1 - \alpha_t) + \alpha_t r_t, \tag{5}$$

where α is a learning rate (or step-size) parameter.

- If the problem is stationary (odds don't change over time), it is desired that the Q's convert. Thus, α_t must decrease with t. Sufficient conditions are,

$$\sum_t^{\infty} \alpha_t = \infty \qquad \sum_t^{\infty} \alpha_t^2 < \infty \tag{6}$$

- If the problem is non-stationary (odds change over time), convergence is not desirable. Could use $\alpha_t = \alpha$ constant, for example.

Problems 1

problem 1.1 *Derive equation 4. Solve the recursion relation for* α *constant.*

Generalization

Representation: A representation of poker hands which makes equivalent hands represented the same way;

A multi-layer perceptron: either

- Inputs are representations of the state the action; single output is the expected reward.
- Inputs are a representation of the state; 32 outputs representing the 32 possible actions; each output is expected reward for that action.

Use gradient descent learning to train network on actual pay-outs.

Problems 2

problem 2.2 *Think of a representation of hands for the video poker game. Can the representation give you generalization without using a neural network?*

Value estimation in delayed reward problems

References: Adaptive Heuristic Critic (Barto 1983, Sutton 1984), Temporal-Difference TD(λ) learning (Sutton 1988), Q-learning (Watkins 1989).

Idea: Train the system to predict the reinforcement signal anytime into the future, but discounted by how long into the future you have to wait. (Discount factor γ)

What is the measure of the value:

Discounted future rewards: At any time t optimize

$$J(t) = \sum_{t'=t}^{\infty} r_{t'} \gamma^{t'-t} \tag{7}$$

$$= r_t + \gamma J(t+1) \tag{8}$$

$\gamma = 0$**:** Try only to get positive reinforcement on the next step

$\gamma = 1$**:** Try to get positive reinforcement anytime in the future

$\gamma \in (0, 1)$**:** discount reward k steps in the future by factor γ^k.

TD Learning

Temporal Difference learning (AKA TD(0) learning); so-called because learning couples to prediction at two different times.

What we want is

- To train $V(\text{state}|\text{Policy})$ to be $J(t)$,
- as before, we could use the following update rule

$$V(\text{state}|\text{Policy}) \leftarrow V(\text{state}|\text{Policy})(1-\alpha) + \alpha J(t) \tag{9}$$

$$\tag{10}$$

- **Problem** we don't know $J(t)$, because it involves the future.

- Replace it by our current estimate.

Use,

$$J(t) = r(t) + \gamma J(t+1) \approx r(t) + \gamma V(\text{state}'|\text{Policy})$$

where state$'$ is chosen from state using

Policy: called on-policy learning, or

Some other policy: called off-policy learning.

Q-learning

An off-policy method. Uses a greedy policy to estimate $J(t+1)$

- Initialize Q(s,a)
- Repeat
  - Choose a from s using policy derived from Q (e.g. ϵ-greedy)
  - Take action a, observe r, s'
  - $Q(s,a) = Q(s,a) + \alpha[r + \gamma \arg\max_{a'} Q(s',a') - Q(s,a)]$
  - $s = s'$
- until end of sequence

Problems 3

problem 3.3 *Consider a line segment with integer states $i = 0, 1, 2, \ldots, 10$. The allowed moves are one step to the left $i \leftarrow i - 1$ and one step to the right, $i \leftarrow i + 1$. The 0 state has a reinforcement of 1, the state 10 gives a reinforcement of -1 and all other states give no reinforcement. Work out the first few steps of Q-learning. What will it converges to?*

problem 3.4 *Show that the correct predicted future reward using a greedy policy is a fixed point of Q-learning.*

Sarsa — An on-policy approach

As above but use the policy to estimate $J(t+1)$.

- Initialize Q(s,a)
- Repeat
  - Choose a from s using policy derived from Q (e.g. ϵ-greedy)
  - Take action a, observe r, s'
  - Choose a' from s' using policy derived from Q (e.g. ϵ-greedy)
  - $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
  - $s = s'$, $a = a'$
- until end of sequence

Results

Q-learning: The Q function converges to the correct prediction for discounted future reward (Watkins and Dayan, 1992).

Sarsa: Can work better in practice, because it takes into account the fact that policy is occasionally explorative.

TD(λ) learning

In above, learning takes place when a reward is reached or when a state is reached which is leads through the policy to a reward. Initially only the state which led immediately to a reward has its value updated. Only through many sequences does learning work its way backwards towards the initial states.

Why not update the value of every state in the sequence which led to the reward?

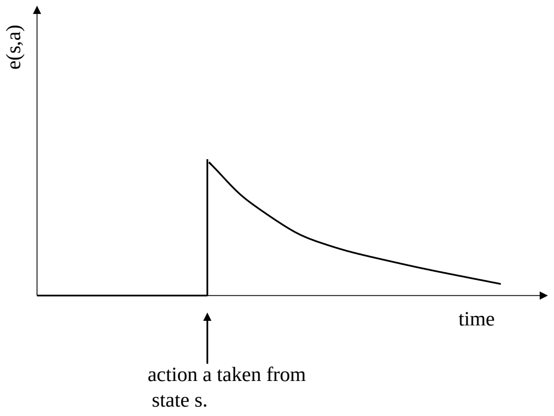
Idea: When reward received, assign credit (or blame) to the action just previous with a weight of 1, the one before that with a weight of λ, the one before that with a weight of λ^2, etc.

Eligibility Traces

An efficient way of accounting for states in the learning sequence.

Let $e(a, s)$ denote the "eligibility" of state-action pair. This is related to how recently in the sequence state s resulted in in action a.

$$e(s, a) = \begin{cases} e(s, a) + 1; & \text{if state } s \text{ results in action } a \\ \lambda\gamma e(s, a); & \text{otherwise} \end{cases}$$



TD(λ) Rule

- Initialize $V(s)$ arbitrarily and $e(s) = 0$ for all s.
- Repeat (for each sequence)
  - Initialize s Repeat
    - a chosen from policy given s
    - Take action a, observe reward r and next state s'
    - $\delta = r + \gamma V(s') - V(s)$
    - e(s) = e(s) + 1;
    - For all states s''
      - $V(s) = V(s) + \alpha\delta e(s)$
      - $e(s) = \gamma\lambda e(s)$
    - $s = s'$
  - Until end of sequence

Problems 4

problem 4.5 *Work out the first few steps of TD(λ) learning for the integer line from problem 3.3.*

Function Approximation and Generalization

Often, it is difficult to record a value for each possible state, because there are too many. Some generalization is required.

Train a neural network to produce the prediction $V(s)$ or $Q(s,a)$. Training examples consist of sequences of states of the system.

- Want to do gradient descent on $[J(t) - V(s_t)]^2$, i.e. V is the network output and J is the target. To update the weights,

$$w_i(t+1) = w_i(t) + \alpha \left[r_t + \gamma V(s_{t+1}) - V(s_t)\right] e_i(t)$$

where

$$e_i(t) = \gamma \lambda e_i(t-1) + \frac{\partial V(s_t)}{\partial w_i(t)}.$$

Conclusions

- Reinforcement learning is learning in which performance can be measured, but correct response is unknown.
- Important in modeling animal learning, control problems, game playing, and other applications.
- One approach is to guess correct response, and search over possible guesses.
- Another class of approaches is to learn the likely reward associated with state-action pairs. The choice of action for any state is treated separately.

Applications: TD-Gammon

Tesauro, 1994

TD(λ) learning has been used to produce a very good backgammon playing program.

Predictor Network: A multi-layer perceptron is used to predict the outcome of the game from the current board position (originally coded as raw board position; later hand-crafted features used).

Controller: A program generates all legal moves from the current position. Predictor network scores them; that with the highest predicted outcome (J) is the moved used.

Training: Data consists of sequences of moves from the standard start position to a result. Program plays itself over and over.

1. Network starts from random state (no knowledge of game)
2. Initially, games incredibly long (100's to 1000's of moves versus 50 – 60 moves for typical human game).
3. Later, basic strategies emerge: hitting opponents, building points, playing safe, ...

Results: TD-Gammon 1.0 Contained 40 hidden units, trained for 200,000 games. Plays at strong intermediate level; good enough to win regional tournaments.

Results: TD-Gammon 2.1 Close to the world's best player.

Has changed how humans play certain board positions (discovered new strategies which appear to be better).

TD-Gammon (cont)

"TD-Gammon has definitely come into its own. There is no question in my mind that its positional judgment is far better than mine. Only on small technical areas can I claim a definite advantage over it . . . . I find a comparison of TD-Gammon and the high-level chess computers fascinating. The chess computers are tremendous in tactical positions where variations can be calculated out. Their weakness is in vague positional games, where it is not obvious what is going on . . . . TD-Gammon is just the opposite. Its strength is in the vague positional battles where judgment, not calculation , is the key. There, it has a definite edge over humans . . . . In particular, its judgment on bold vs. safe play decisions, which is what backgammon really is all about, is nothing short of phenomenal . . . . Instead of a dumb machine which can calculate things much faster than humans such as the chess playing computers, you have built a smart machine which learns from experience pretty much the same way that humans do"

Kit Woolsey, perennial world top 10 backgammon player (ranked #3 when this was written), quoted in Tesauro, 1995.

Classical Conditioning

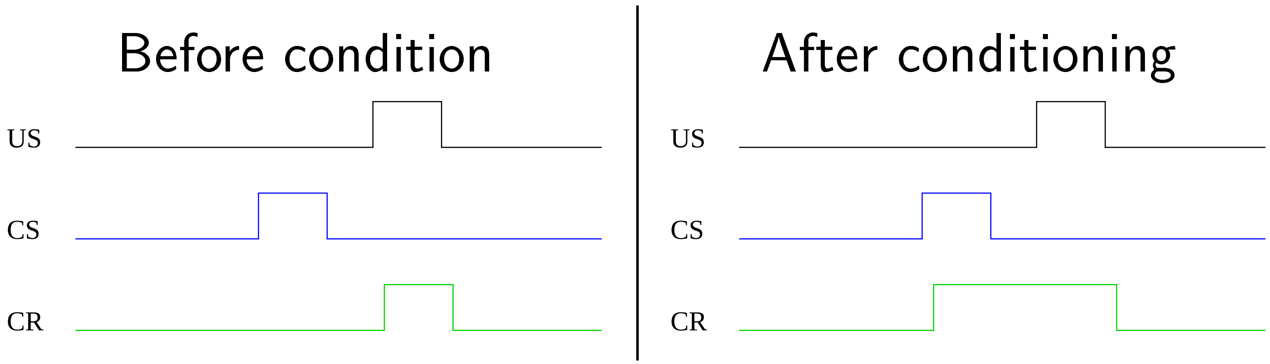
Moore et. al. (1989, 1990, 1996), Sutton and Barto (1990), Desmond (1990).

Unconditioned Stimulus(US): Stimulus which would cause response in the untrained animal.

Conditioned Stimulus (CS): Stimulus caused response only due to training.

Conditioned Response CR: The response.

Famous example discovered by Pavlov. Ringing of bell (CS) followed by food (US) resulted in salivation (CR). Other experimental procedures: CS followed by puff of air directed at eye cause animal to close eye in response to CS.



Question: how does the CS and the CR become associated across time?

Blocking

- After conditioning on stimulus A, US is preceded by *simultaneous* presence of stimuli A and B.

  Result: *Not* conditioned on B.

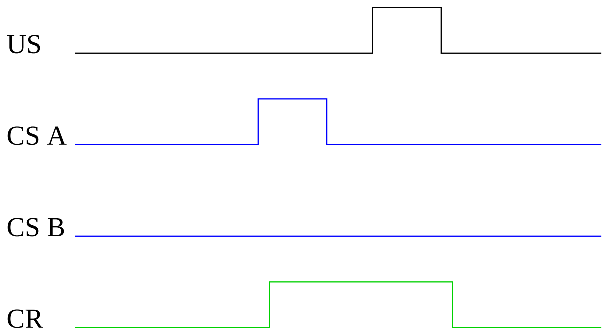
- However, if B precedes A, then conditioning on A is lost and US becomes conditioned on B. (More complicated scenarios also studied).

Question: what prevents the association with stimulus B?

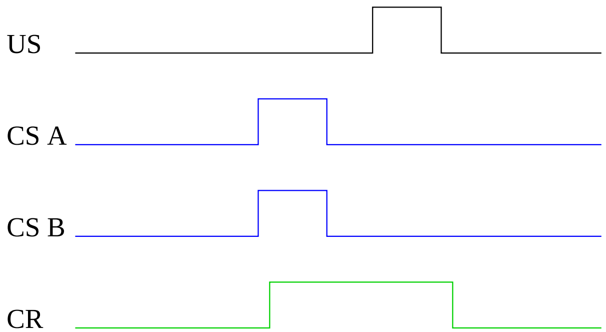
Blocking Experiments

First 10 trials associate A with the US.

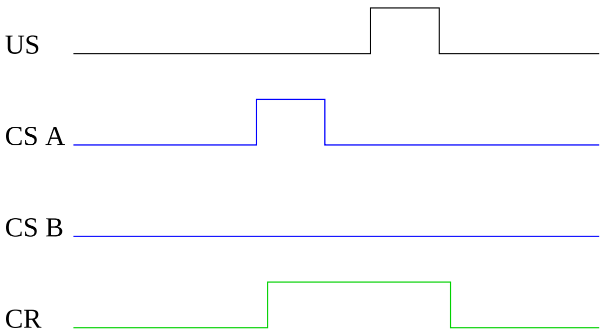
After trial 10



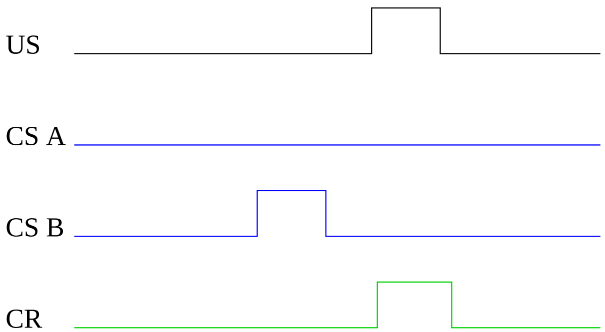
Trials 11–20



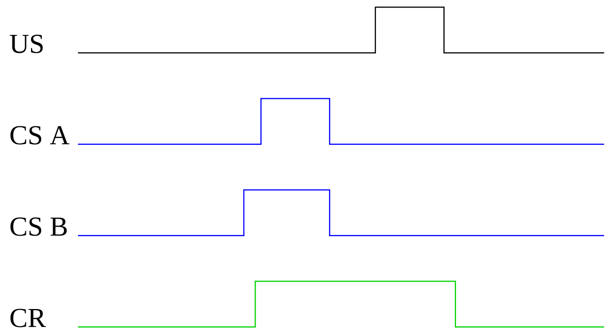
Association with A is unaffected



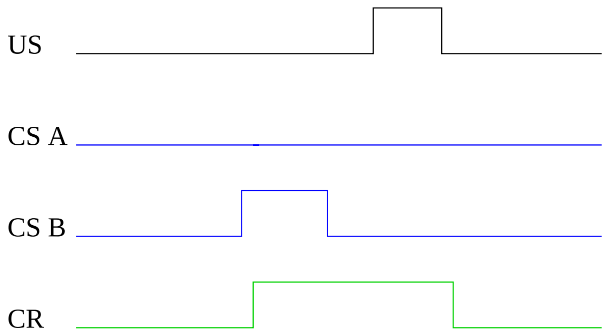
Association with B is blocked



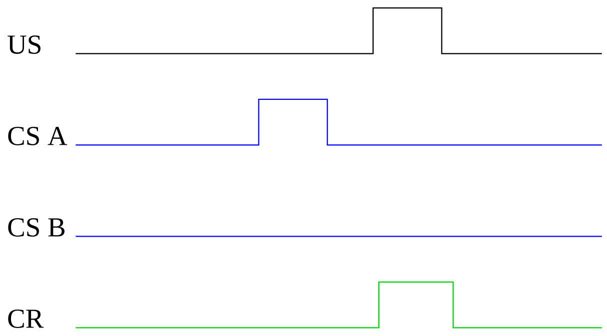
Trials 21–35



Association with B is made



Association with A is extinguished



TD model of classical conditioning

The principles of TD learning can explain these puzzles:

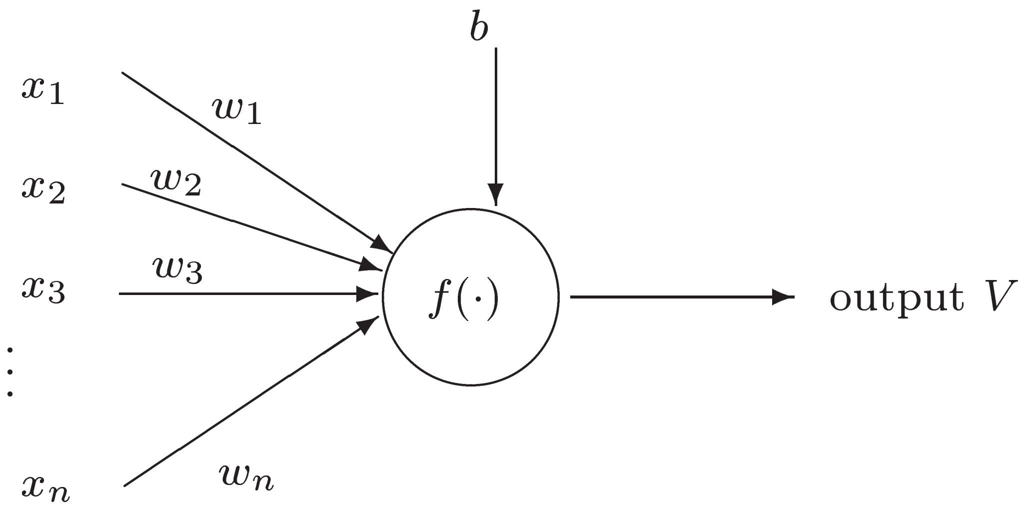
- the animal associates the stimulus with the (discounted) expected future reward,
- the eligibility for the stimulus to participate in learning persists, decaying amount λ per unit time.

A simple neural network model

Inputs Possible stimuli $x_i(t)$.

Output: V represents the predicted value of the response.

$V = \sum_i w_i x_i$ or possible related through a sigmoid function.



Learning Rule

$$\Delta w_i = \alpha \left[r_t + \gamma f \left\{ \sum_j w_j(t) x_j(t+1) \right\} - f(\sum_i w_j(t) x_j(t)) \right] \overline{X_i(t)}$$

Here r is the reward, $f(\cdot)$ is the output function for the neural network, and the eligibility trace obeys,

$$\overline{X_i(t)} = \gamma \lambda \overline{X_i(t-1)} + x_i(t) f' \left[\sum_i w_j(t) x_j(t) \right].$$

This is precisely the TD learning rule for neural networks used in TD-gammon (but with a simpler neural network).

Hebbian like rule, where input activation replaced by slowly decaying eligibility trace, δ is a difference between outputs at different times.

How this gives classical results

1. Eligibility trace cause information about CS to persist.
2. Temporal difference rule causes information about the CR (r) to act during earlier times.

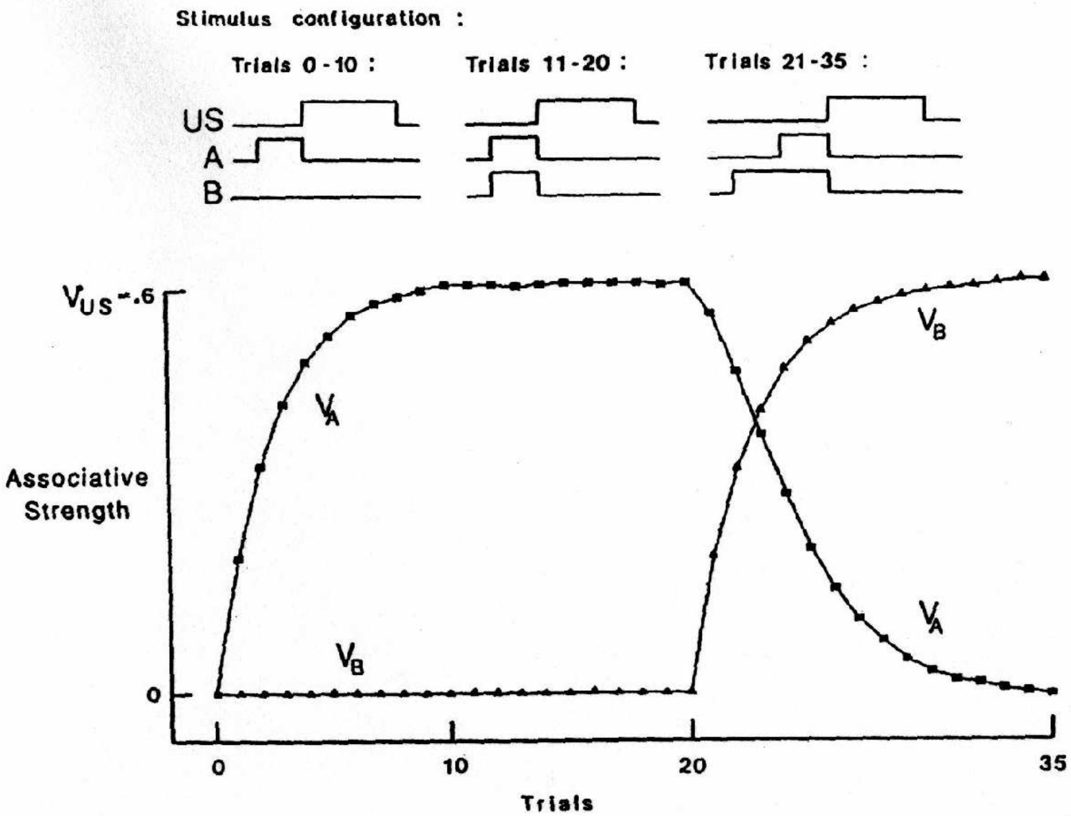


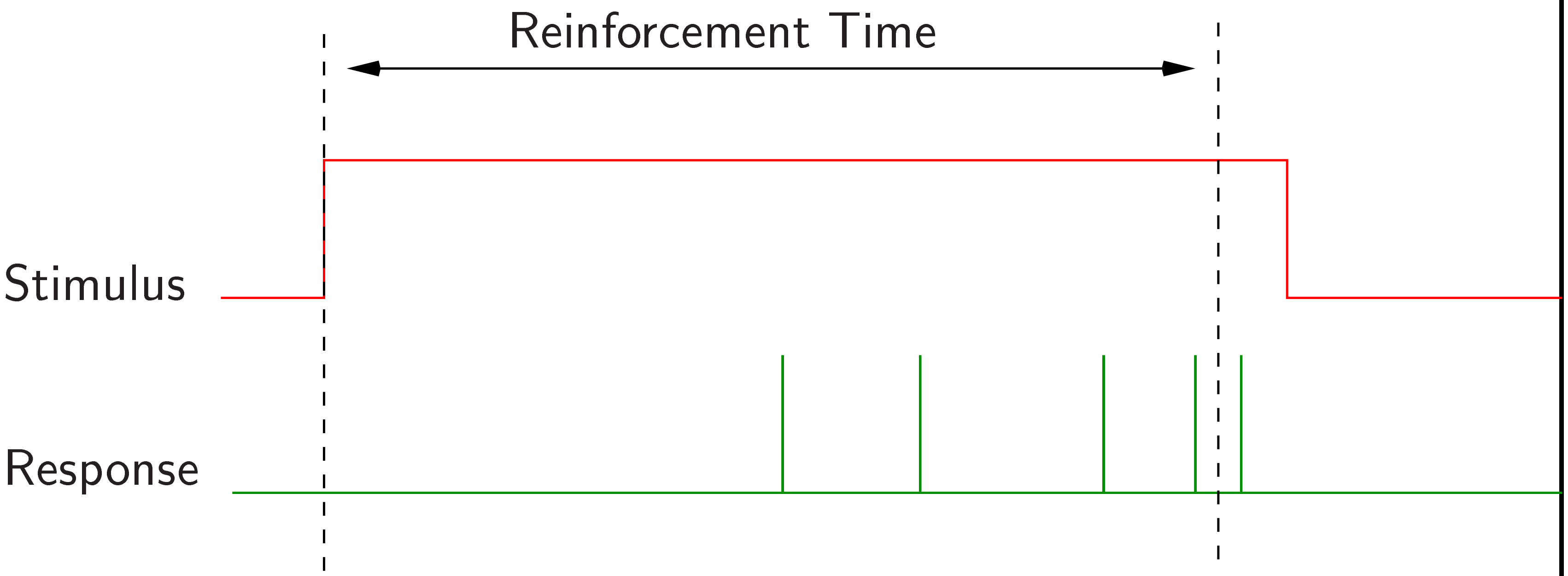
Figure 4
Illustration of the SB model. Shown are the associative strengths after each trial of a simulated experiment involving simple acquisition (trials 1–10), blocking (trials 11–20), and primacy (trials 21–35). Reprinted, with notational changes, from Sutton and Barto 1981a.

Learning expected time to a reward

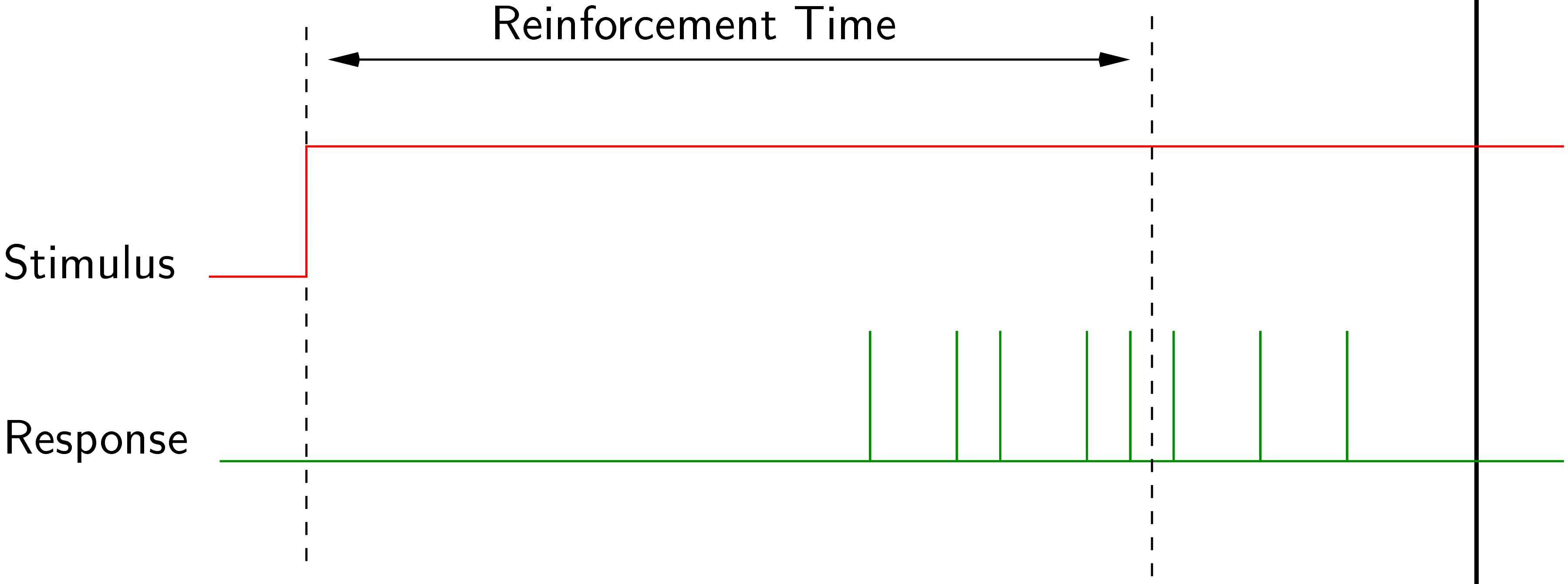
- One aspect of delayed-reward reinforcement learning involves learning the expected time to wait for the reward.
- This may be useful for determining when the expected reward is not forthcoming (e.g. the foraging patch is deleted and it is time to move on), comparing reinforcement rates, etc.
- Numerous experiments that animals can do this, and do it in a characteristic and striking manner — they exhibit *scalar timing*.

Peak procedure experiments

Reward Trials (training) Stimulus (bell, light) comes on to indicate start of trial. Animal is rewarded for the first response after a reinforcement time has passed.



Non-reward Trials Stimulus remains on even after animal responds after reinforcement time has passed. This is to see for how long the animal continues responding.



Results on non-reward trials — the scalar property

Responses averaged over many non-reward trials give a smooth curve. Studies of the curves for different reinforcement times show the following features:

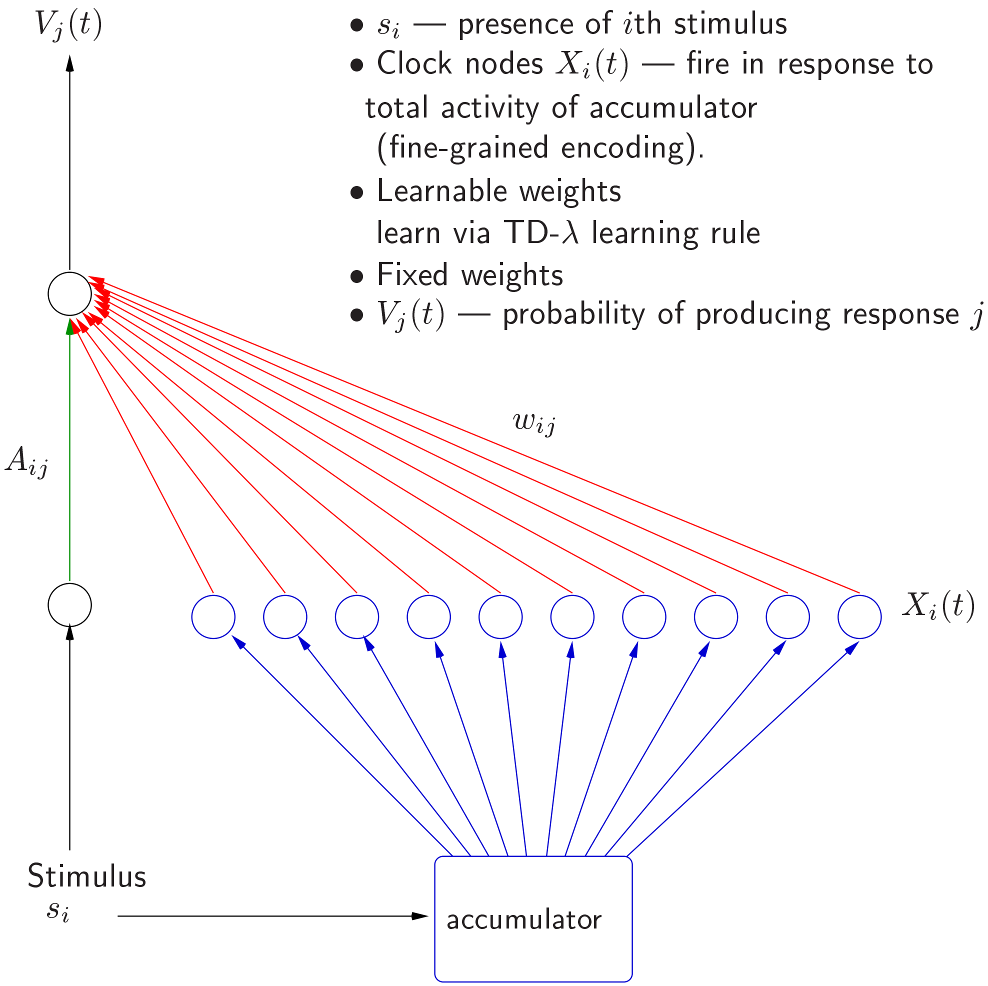
- The highest rate of response is at the expected reinforcement time.
- Scaled curves from different reinforcement times collapse when scaled relative to the reinforcement time

Let T be the actual time since the start of the trial and $\tilde{T}$ be subjective time.

$$\boxed{P(\tilde{T}|T) \approx \frac{1}{T} P_{\mathsf{inv}}\left(\frac{\tilde{T}}{T}\right)}. \tag{11}$$

Here P_{inv} is the function which describes the shape of the scaled curves; it is peaked near one. *This is the scalar property of interval timing.*

Reinforcement Learning of Time Intervals



- s_i — presence of ith stimulus
- Clock nodes $X_i(t)$ — fire in response to total activity of accumulator (fine-grained encoding).
- Learnable weights learn via TD-λ learning rule
- Fixed weights
- $V_j(t)$ — probability of producing response j

Learning model

Time perception system: consists of

- Accumulator network — measures time as accumulation of neural spikes (presented in Shapiro and Wearden, 2001).
- Clock nodes $X_i(t)$ — fire in response to total activity of the accumulator.

Stimulus-response: Both the stimulus and the appropriate clock node must be present for there to be reasonable chance of response.

- Response nodes $V_j(t)$ — determine the response. Response j is triggered with probability $\min[V_j(t), 1]$,

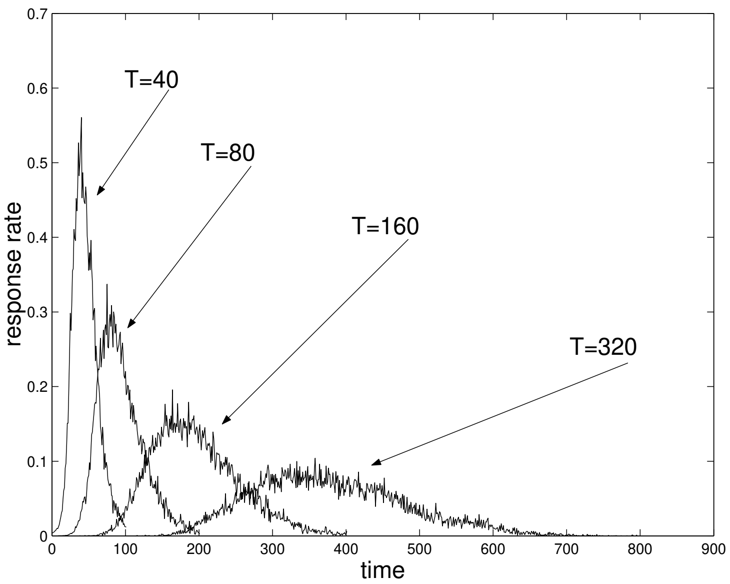
$$V_j(t) = \left\lfloor \sum_i w_{ij} X_i(t) + A_{ij} s_i - \theta_j \right\rfloor , \tag{12}$$

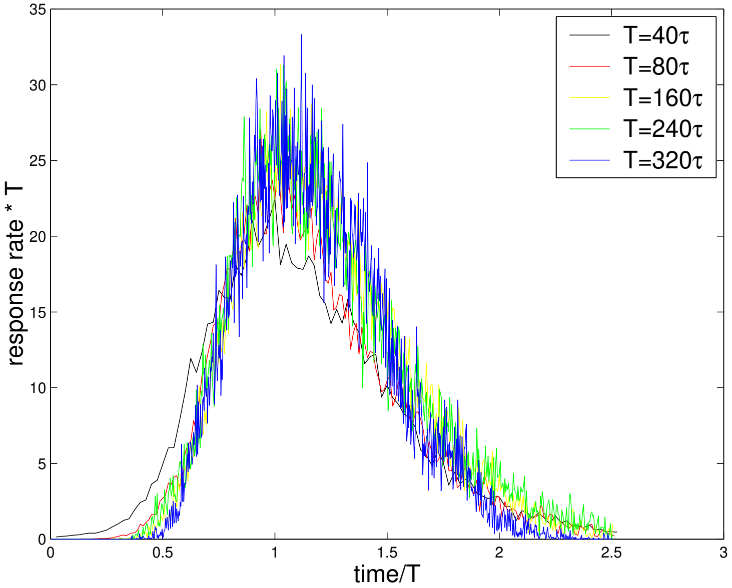
  where $\lfloor \cdot \rfloor$ denotes a function which is the identity for positive argument, zero otherwise. The threshold θ_j is constant.

Reinforcement learning: Learnable weights learn via TD-λ learning rule.

Simulations of Peak Procedure

Method: 50 trials with forced response followed by 1150 trials. Every 10th one is a non-reinforced trial. Learning takes place throughout. Results averaged over the last 100 non-reward trials (i.e. the last 1000 trials).





Response per unit time for different reinforcement times T. Parameters: $m_I = 10$, $C\sigma_\nu^2 = 1$ (Poisson limit), $\gamma = 0.75$, $\lambda = 1$, $\alpha = 0.5$.

Response per relative time versus relative time for different reinforcement times T. The data collapses, clearly showing the scalar property (equation (11)).

Conclusions

- Reinforcement learning is an important paradigm of adaptive computation.
- It is often useful to separate learning the expected reward, and the policy for choosing actions.
- Heuristics which are important for delayed reward problems are:
  - Learn the discounted reward,
  - associate action with reward by discounting a factor λ per unit time.
- These heuristics result in useful examples of AI (TD-gammon) and effective models of animal experiments.