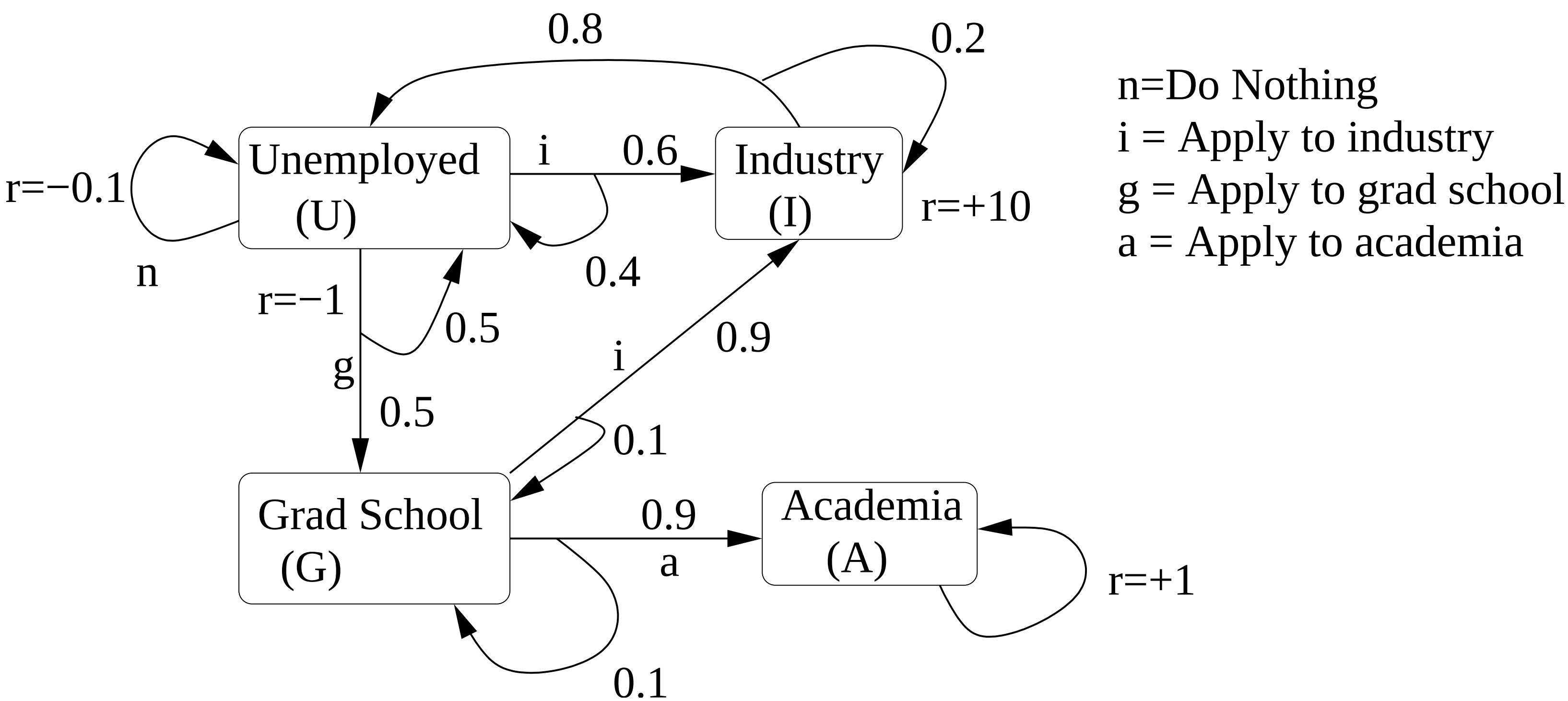
Lecture 23: Reinforcement Learning

- MDPs revisited
- Model-based learning
- Monte Carlo value function estimation
- Temporal-difference (TD) learning
- Exploration

Recall: Markov Decision Processes



- A set of states S
- A set of actions A
- Expected rewards $R(s, a)$
- Transition probabilities $T(s, a, s')$
- Discount factor γ

Recall: Policy Evaluation Problem

- Suppose someone told us a policy for selecting actions, $\pi : S \times A \longrightarrow [0,1]$
- How much return do we expect to get if we use it to behave?

$$V^{\pi}(s) = E_{\pi}[R_t | s_t = s] = E_{\pi}[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} | s_t = s]$$

- If we knew this, we could then *improve* the policy (e.g., using policy iteration)

Iterative Policy Evaluation

1. Start with some initial guess V_0
2. During every iteration k, update the values of all states:

$$V_{k+1}(s) \leftarrow \sum_a \pi(s,a)\left(R(s,a) + \gamma \sum_{s'} T(s,a,s')V_k(s')\right), \forall s$$

3. Stop when the maximum change between two iterations is smaller than a desired threshold (the values stop changing)

The value of one state is updated based on the values of the states that can be reached from it

How Is Learning Tied with Dynamic Programming?

- Observe transitions in the environment, learn an *approximate model* $\hat{R}(s,a), \hat{T}(s,a,s')$
  - Use maximum likelihood to compute probabilities
  - Use supervised learning for the rewards
- Pretend the approximate model is correct and use it for any dynamic programming method
- This approach is called *model-based reinforcement learning*
- Many believers, especially in the robotics community

Monte Carlo Methods

- Suppose we have an episodic task: the agent interacts with the environment in trials or episodes, which terminate at some point
  - E.g. game playing
- The agent behaves according to some policy π for a while, generating several trajectories.
- How can we compute V^π?

Monte Carlo Methods

- Suppose we have an episodic task: the agent interacts with the environment in trials or episodes, which terminate at some point
- The agent behaves according to some policy π for a while, generating several trajectories.
- How can we compute V^π?
- Compute $V^\pi(s)$ by *averaging the observed returns* after s on the trajectories in which s was visited.

Implementation of Monte Carlo Policy Evaluation

Let V_{n+1} be the estimate of the value from some state s after observing $n+1$ trajectories starting at s.

$$\begin{aligned}
V_{n+1} &= \frac{1}{n+1}\sum_{i=1}^{n+1} R_i = \frac{1}{n+1}\left(\sum_{i=1}^{n} R_i + R_{n+1}\right) \\
&= \frac{n}{n+1}\frac{1}{n}\sum_{i=1}^{n} R_i + \frac{1}{n+1}R_{n+1} \\
&= \frac{n}{n+1}V_n + \frac{1}{n+1}R_{n+1} = V_n + \frac{1}{n+1}\left(R_{n+1} - V_n\right)
\end{aligned}$$

If we do not want to keep counts of how many times states have been visited, we can use a *learning rate* version:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t))$$

What If State Space Is Too Large ?

- Represent the state as a vector of input variables
- Represent V as a linear/polynomial function, or as a neural network
- Use $R_t - V(s_t)$ as the error signal!

Temporal-Difference (TD) Prediction

- Monte Carlo uses as a target estimate for the value function the actual return, R_t:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[R_t - V(s_t) \right]$$

- The simplest TD method, TD(0), uses instead an *estimate* of the return:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right]$$

If $V(s_{t+1})$ were correct, this would be like a dynamic programming target!

TD Is Hybrid between Dynamic Programming and Monte Carlo!

- Like DP, it *bootstraps* (computes the value of a state based on estimates of the successors)
- Like MC, it estimates expected values by *sampling*

TD Learning Algorithm

1. Initialize the value function, $V(s) = 0, \forall s$
2. Repeat as many times as wanted:
   (a) Pick a start state s for the current trial
   (b) Repeat for every time step t:
      i. Choose action a based on policy π and the current state s
      ii. Take action a, observed reward r and new state s'
      iii. Compute the TD error: $\delta \leftarrow r + \gamma V(s') - V(s)$
      iv. Update the value function:
$$V(s) \leftarrow V(s) + \alpha_s \delta$$
      v. $s \leftarrow s'$
      vi. If s' is not a terminal state, go to 2b

Example

Suppose you start will all 0 guesses and observe the following episodes:

- B,1
- B,1
- B,1
- B,1
- B,0
- A,0; B (reward not seen yet)

What would you predict for $V(B)$? What would you predict for $V(A)$?

Example: TD vs Monte Carlo

- For B, it is clear that $V(B) = 4/5$.
- If you use Monte Carlo, at this point you can only predict your initial guess for A (which is 0)
- If you use TD, at this point you would predict $0 + 4/5$! And you would adjust the value of A towards this target.

Example (continued)

Suppose you start will all 0 guesses and observe the following episodes:

- B,1
- B,1
- B,1
- B,1
- B,0
- A,0; B 0

What would you predict for $V(B)$? What would you predict for $V(A)$?

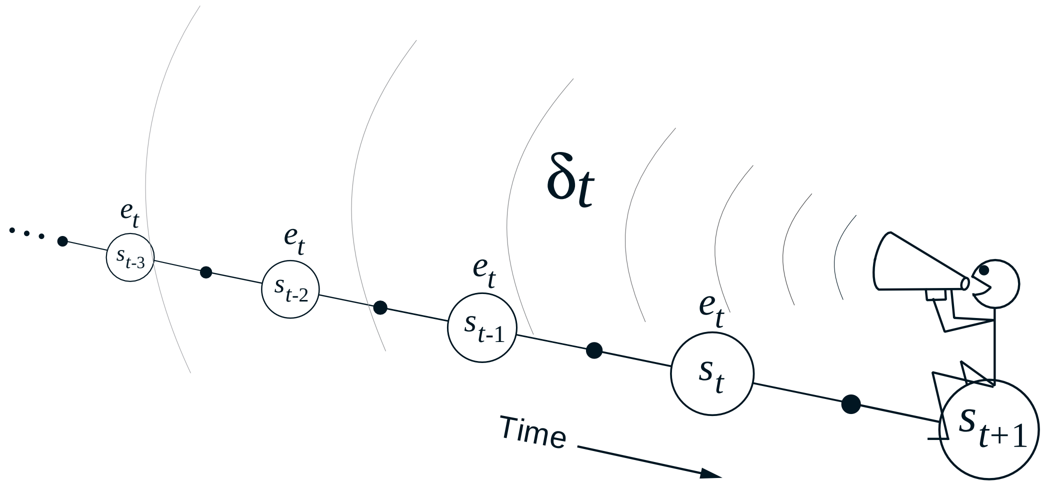
Example: Value Prediction

- The estimate for B would be $4/6$
- The estimate for A, if we use Monte Carlo is 0; this minimizes the sum-squared error on the training data
- If you were to learn a model out of this data and do dynamic programming, you would estimate the A goes to B, so the value of A would be $0 + 4/6$
- TD is an *incremental* algorithm: it would adjust the value of A towards $4/5$, which is the current estimate for B (before the continuation from B is seen)
- This is closer to dynamic programming than Monte Carlo
- TD estimates take into account *time sequence*

Example: Eligibility Traces

- Suppose you estimated $V(B) = 4/5$, then saw $A, 0, B, 0$.
- Value of A is adjusted right away towards $4/5$
- But then the value of B is decreased from $4/5$ to something like $4/6$
- It would be nice to propagate this information to A as well!

Eligibility Traces (TD(λ))



- On every time step t, we compute the TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

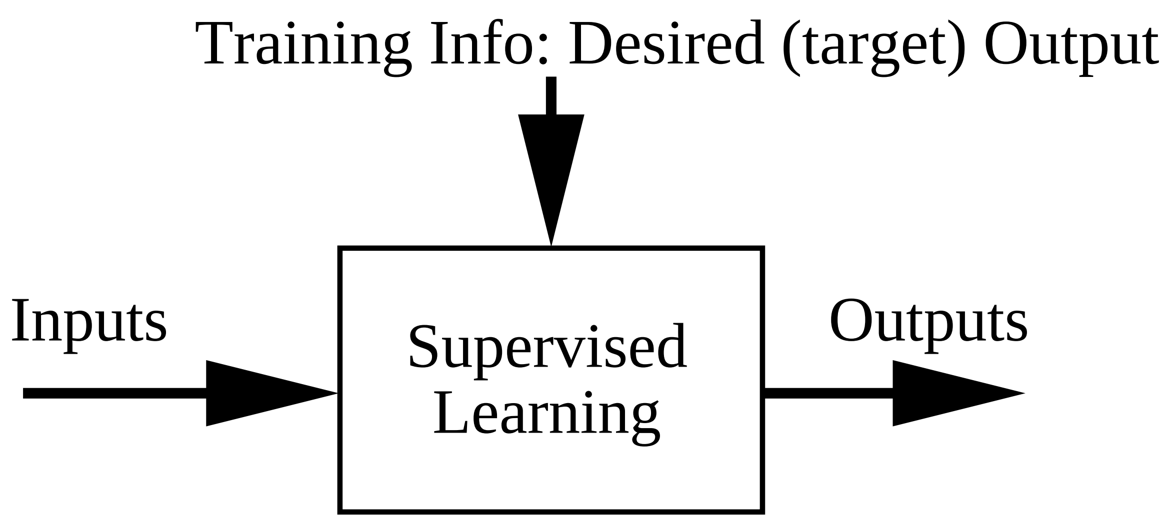
- Shout δ_t backwards to past states
- The strength of your voice decreases with temporal distance by $\gamma\lambda$, where $\lambda \in [0, 1]$ is a parameter

Advantages

- No model of the environment is required! TD only needs experience with the environment.
- On-line, incremental learning:
  - Can learn before knowing the final outcome
  - Less memory and peak computation are required
- Both TD and MC converge (under mild assumptions), but TD usually learns faster.

Large State Spaces: Adapt Supervised Learning Algorithms

- A training example has an input and a target output
- The error is measured based on the difference between the actual output and the desired (target) output



Value-Based Methods

We will use a function approximator to represent the value function

- The input is a description of the state
- The output is the predicted value of the state
- The target output comes from the TD update rule: the target is $r_{t+1} + \gamma V(s_{t+1})$

On-line Gradient Descent TD

1. Initialize the weight vector of the function approximator $\mathbf{w}$
2. Pick a start state s
3. Repeat for every time step t:
   (a) Choose action a based on policy π and the current state s
   (b) Take action a, observe immediate reward r and new state s'
   (c) Compute the TD error: $\delta \leftarrow r + \gamma V(s') - V(s)$
   (d) Update the weight vector: $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla V(s)$
   (e) $s \leftarrow s'$

Observations

- For linear function approximators, the gradient is just the input feature vector
- For neural networks, this algorithm reduces to running backpropagation with TD error at the output

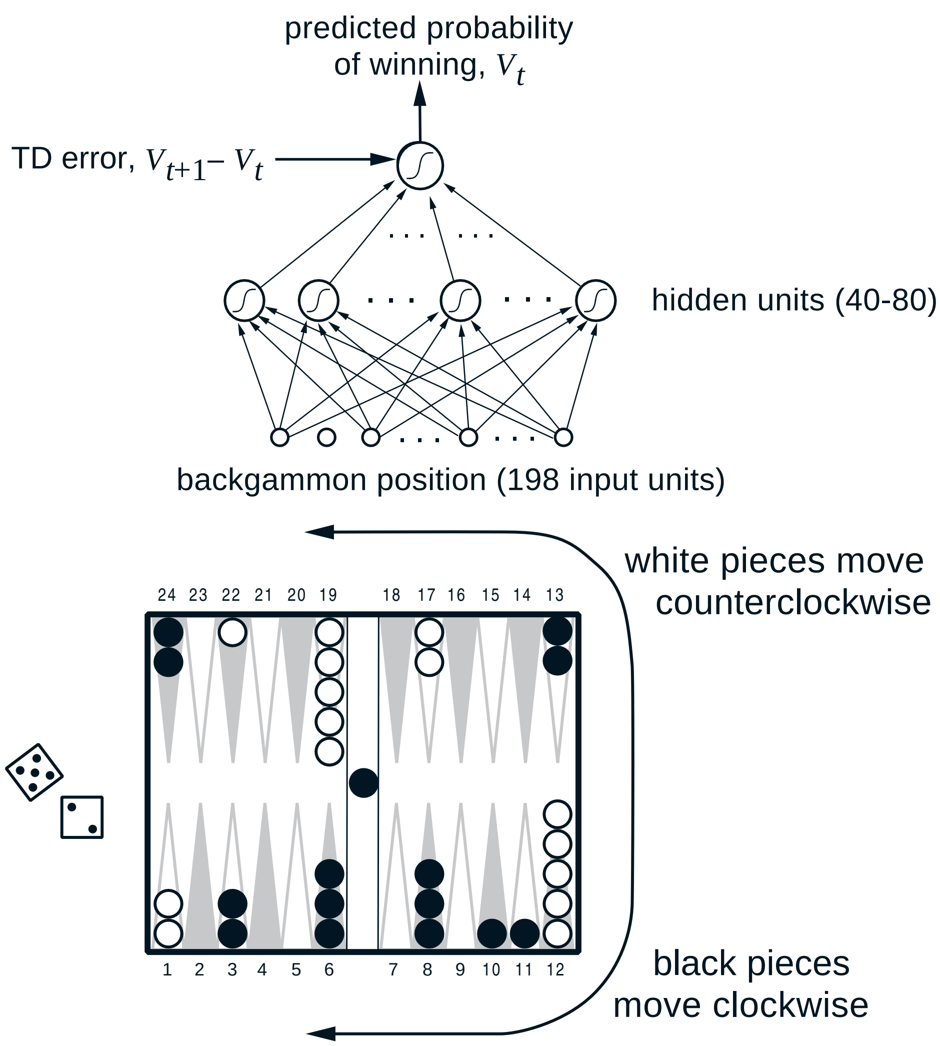
RL Algorithms for Control

- TD-learning (as above) is used to compute values for a *given* policy π
- *Control methods* aim to find the *optimal policy*
- In this case, the behavior policy will have to balance two important tasks:
  - *Explore* the environment in order to get information
  - *Exploit* the existing knowledge, by taking the action that currently seems best

Exploration

- In order to obtain the optimal solution, the agent must try all actions
- Simplest exploration scheme: *ϵ-greedy*
  - With probability $1 - \epsilon$ choose the action which currently appears best
  - With probability ϵ choose an action uniformly randomly
- Much research is done in this area!

TD-Gammon (Tesauro, 1992-1995)



TD-Gammon: Training Procedure

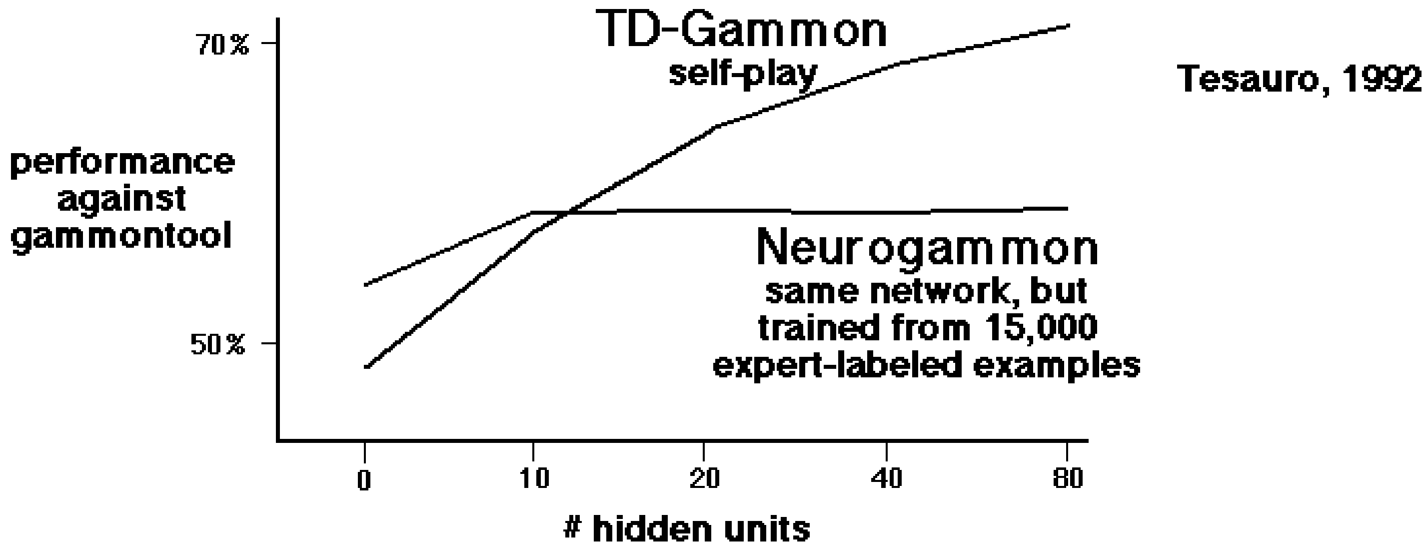
Immediate reward:

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games *against itself*

Now approximately equal to best human player

The Power of Learning from Experience



- Expert examples are expensive and scarce
- *Experience is cheap and plentiful!*

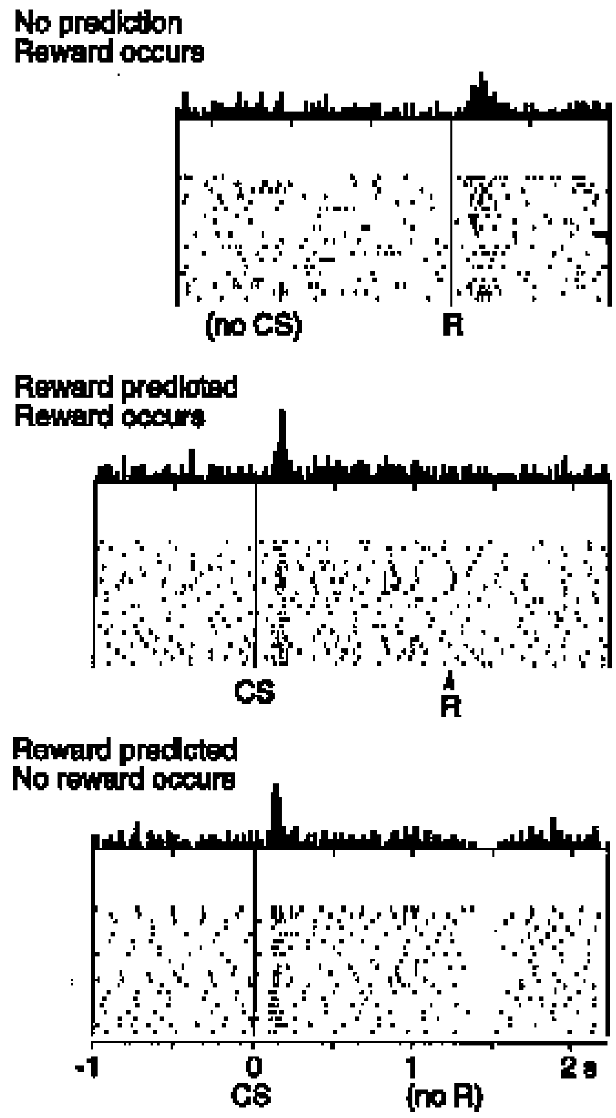
Applying Reinforcement Learning to Chess

- TD does not replace search, it gives a way for computing value functions
- Useful trick: instead of evaluating state before your move, evaluate the state *after* your move (called afterstate)
- This makes it easier to choose moves
- Exploration is very important, because the game is deterministic and self-play can get into behavior loops.
- Example: KnightCap (Baxter Tridgell & Weaver, 2000)

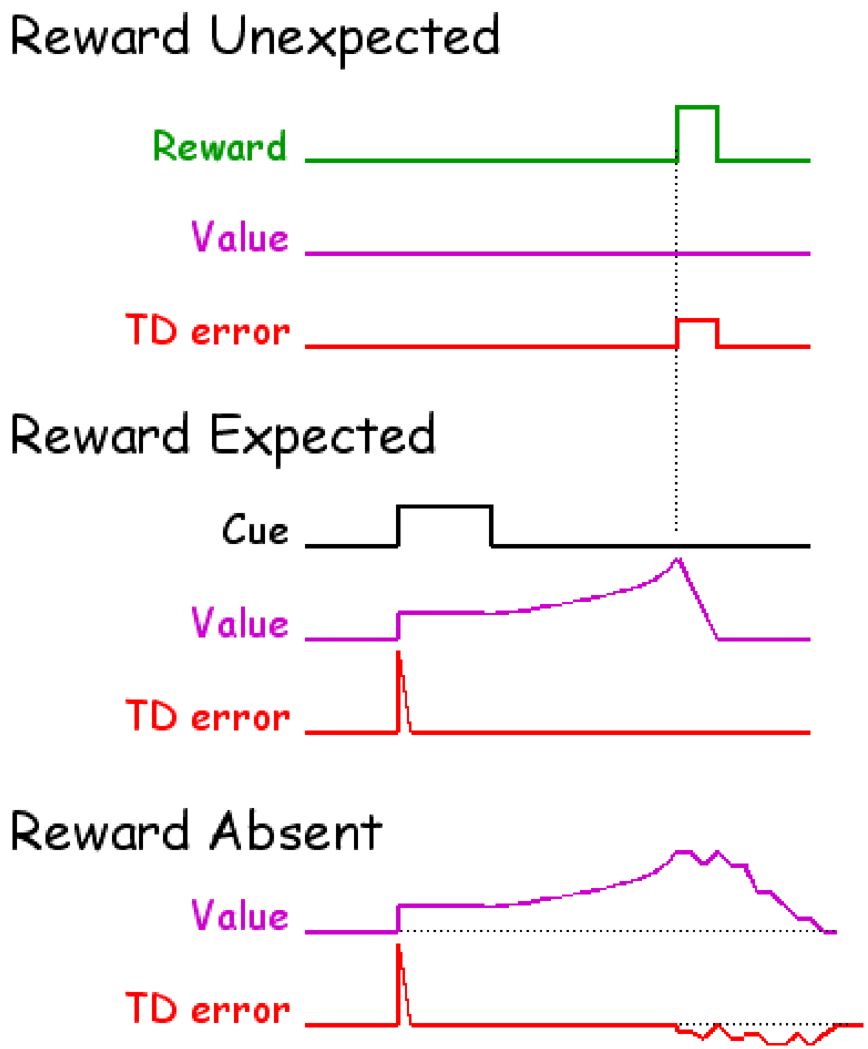
Success Stories

- TD-Gammon (Tesauro, 1992)
- Elevator dispatching (Crites and Barto, 1995): better than industry standard
- Inventory management (Van Roy et. al): 10-15% improvement over industry standards
- Job-shop scheduling for NASA space missions (Zhang and Dietterich, 1997)
- Dynamic channel assignment in cellular phones (Singh and Bertsekas, 1994)
- Robotic soccer (Stone et al, Riedmiller et al...)
- Helicopter control (Ng, 2003)
- Modelling neural reward systems (Schultz, Dayan and Montague, 1997)

Dopamine Neurons Signal "Error/Change" in Prediction of Reward



Computation Theoretical TD Errors



Summary

- Reinforcement learning can be used to learn value functions directly from interaction with an environment
- Monte Carlo methods use samples of the actual return
- TD methods use just samples of the next transition!
- Both converge in the limit, but TD is usually faster
- It is easy (algorithmically) to combined RL with function approximation
- In this case, it is much harder to establish the theoretical properties of the algorithms, but they often work well in practice.