Reinforcement Learning

Chapter 21

Outline

- ♢ The Machine Learning version
- ♢ The Brain's Version: Basal Ganglia

General Learning Types

1. Unsupervised Learning
   (a) **No** environmental feedback concerning correctness
   (b) Learning system detects invariant patterns in the data without attaching right/wrong status to them.
2. Reinforcement Learning (RL)
   (a) **Occasional** environmental feedback of form right/wrong or good/bad.
   (b) Feedback often comes at the end of a long sequence of actions.
   (c) Credit Assignment Problem: use sparse feedback to rate earlier actions.
3. Supervised Learning
   (a) **Frequent** environmental (e.g. teacher) feedback that includes the **correct action/response**.
   (b) Many classic ML algorithms rely on this constant feedback.

RL + Unsupervised Learning are most common in the real world. Good evidence for both in the brain.

RL from the Master, Richard Sutton

- Learning to act in unknown environments with only occasional feedback.
- Agent learns from its own experience in the environment.
- RL involves learning the whole problem at once, not via combined subproblems.
- Much ML (e.g. concept learning) focuses on subproblems without indicating how they could be used to solve interesting whole problems.
- Balance of exploration -vs- exploitation is key to getting complete information about the environment.
- RL helps tie AI to control theory, statistics, operations research, etc. - A move from symbolic to more quantitative methods.

Basic Reinforcement Learning Scenario

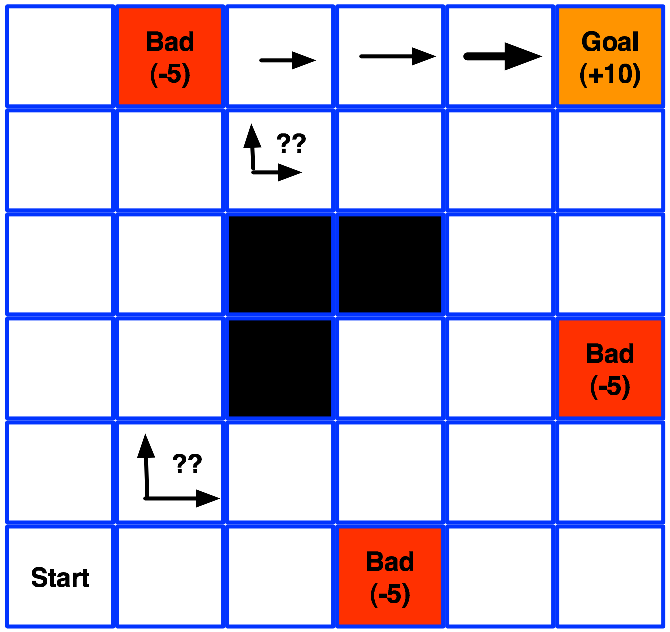
1. An agent performs actions in a relatively unfamiliar environment.
2. By acting, the agent experiences different states of the environment.
3. By receiving occasional feedback, the agent gives numeric evaluations to:
   - states (Utility learning)
   - state-action pairs (Q-learning)

This info can be used to define a **policy**: $\forall s \in States$ best-action(s)
States can involve:

1. Locations in a spatial grid
2. Values of sensors
3. Values of actuators
4. All of the above, and more

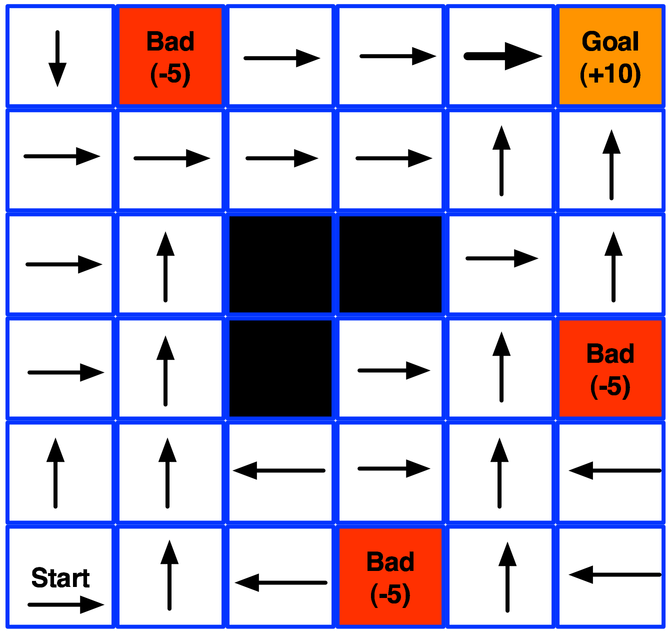
Problem: Exponential state spaces $\longrightarrow$ long time to learn anything.

Starting Out: Blank Slate



1. Explore environment
2. Find reinforcements (rewards or penalties)
3. Relay (modified) reinforcement info upstream along the state-action-state sequence.
4. Use this info to modify evaluations for states and/or state-act pairs.
5. Use these evals to bias further exploration (in *on-policy* versions of RL).

An Informed Policy



For each state(cell), we now have the best direction in which to travel, i.e., best movement action.

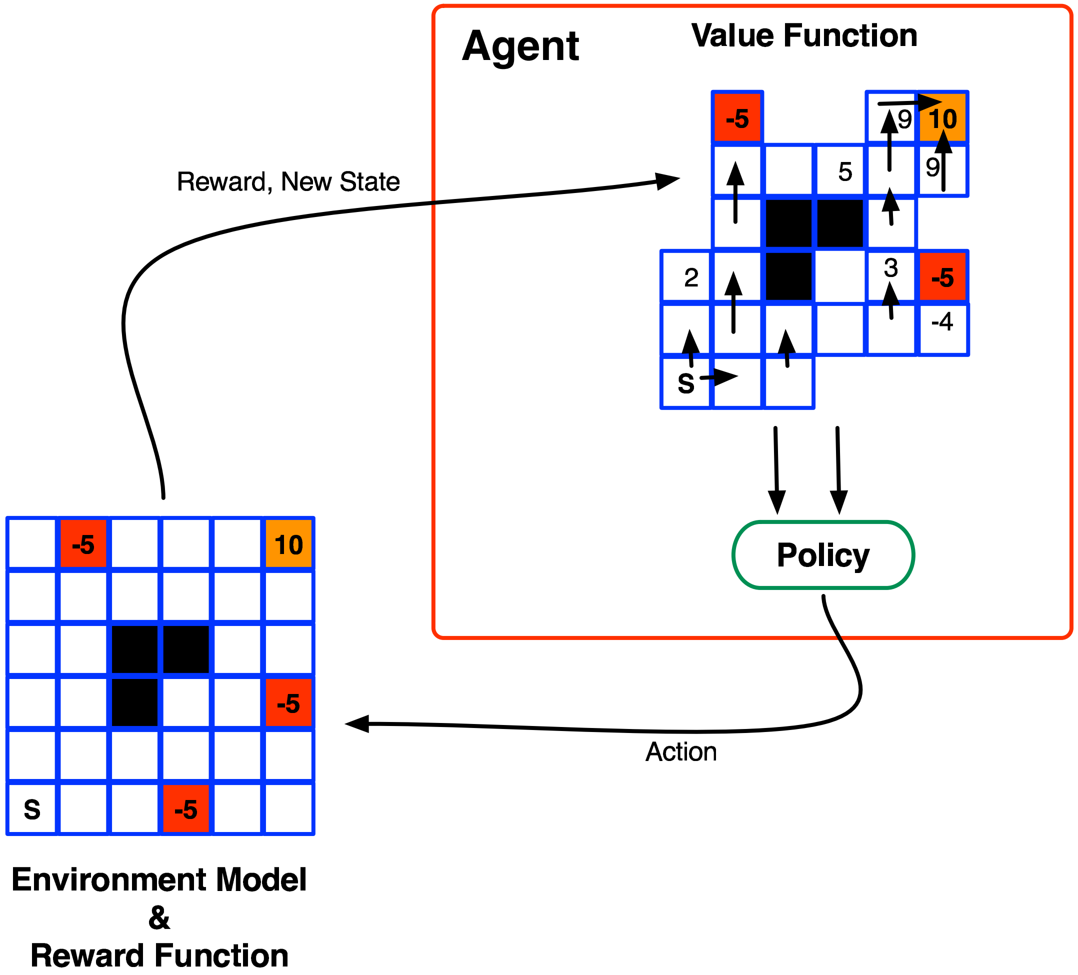
Underlying this is a complete **value function** giving the utility of performing each of the 4 actions (move left, right, up or down) in each of the cells.

Key Components of an RL System

Sutton & Barto (1996), *Reinforcement Learning: An Introduction*.

1. Policy - mapping from states to actions that governs the agents behavior during problem-solving and learning. Note: In RL, learning and prob-solving occur simultaneously. *On the job training*.
2. Reward Function - mapping from states to **immediate** rewards. States with no direct payoff have 0 in this mapping/table. Goals are states with the highest possible reward. In bio systems, the rewarded states might be those involving high immediate pleasure or pain.
3. Value Function - mapping from states (or state-action pairs) to **potential** rewards, assuming one follows a path through this state to a goal or penalty state. A more far-sighted evaluation than the Reward Function. A value = a **prediction** of reward/penalty.
4. Environmental Model - a mapping from (state,action,state) triples to transition probabilities; e.g. When doing action A in state S1, what is the probability of transitioning to state S2? This reflects the dynamics of the *environment* in which problem-solving occurs.

Agent-Environment Interaction in RL



Direct Estimation of the Value Function

Value(s) = expected total reward found by moving onward from s to the goal, with movement directed by a stochastic transition table given by the policy, π.

Bellman equation:

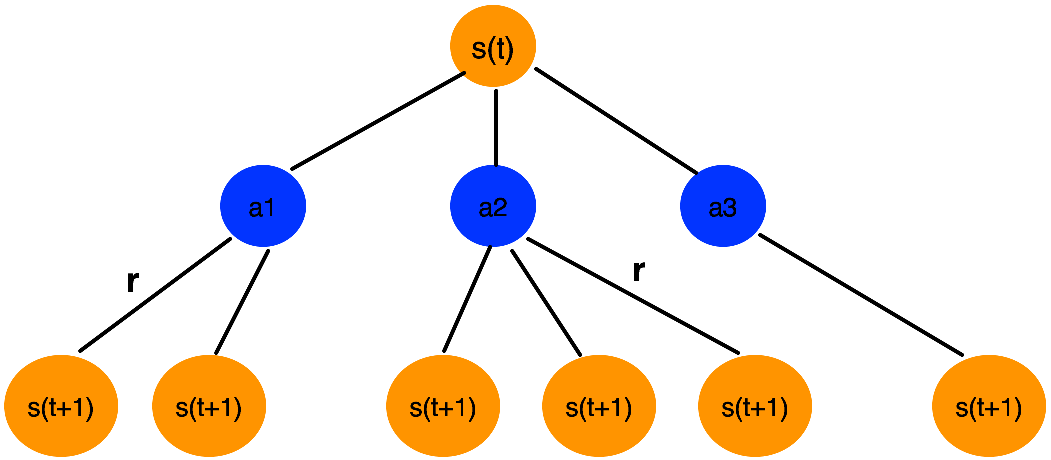
$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s') \tag{1}$$

- Value(s) = Reward achieved at s + expected future reward.
- Future reward = weighted (by transition probabilty from $s \rightarrow s'$) reward in each immediate downstream state, s'.
- Everything is relative to π, the current policy, which determines the action-choice probabilities and thus the relationship between exploration and exploitation.

Backups

- Backing up $\longrightarrow$ evaluations of states (or state-action pairs) **later** in a search path are used to update evaluations of states (or pairs) **earlier** in the path.
- Different RL methods have different backup schemes.

Backup diagram for V^{π} when using the Bellman equation:



Temporal-Difference (TD) Learning

A very popular form of Reinforcement Learning

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \tag{2}$$

- α = learning rate
- R_t = total **actual** reward from time t until search ends at reinforced state.

Update **prediction** $V(s_t)$ *by its deviation from* **actual** *accumulated reward.*
However, R_t is not known until the search ends.
TD methods **estimate** R_t via a bootstrapping assumption:

$$R_t \approx r_{t+1} + V(s_{t+1}) \tag{3}$$

I do not know R_t, so I will estimate it recursively as the actual reward at time t+1 plus the estimated value of the next state. Hence:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + V(s_{t+1}) - V(s_t)] \tag{4}$$

The Temporal Difference Error

- The temporal difference error (TD error) is the difference between the value estimate at time t and the slightly more informed estimate at t+1.
- Why more informed at t+1? Because it has more actual information: r_{t+1}
- By updating $V(s_t)$ in this manner, we move it closer to the correct value.
- Barto(1998): *The blind being led by the slightly less blind*

The TD(0) Algorithm for estimating V^{π}

From Sutton & Barto (1996):

- Init V(s) arbitrarily and choose a policy, π, to govern search.
- Repeat for each search episode
  - Begin at a start state
  - Repeat until a terminal state is entered.
    - $a \leftarrow \pi(s)$
    - Do action a
    - Observe reward, r, and next state s'
    - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
    - $s \leftarrow s'$

Note: γ is a discounting factor, which, when < 1, says that future expected values are worth less in the present.
The 0 in TD(0) $\rightarrow$ eligibility traces not used: Effects of future rewards are not directly passed back to earlier acts in the action sequence.

The Policy

- Determines what actions to take from each state.
- May be deterministic or stochastic.
- The policy, π, should be an intelligent combination of exploration and exploitation. For example, you do not always want to take the best-action-so-far (over-exploitation), since you need to give other actions a fair evaluation.
- **Greedy** policies strongly favor the best-so-far, while **softer** policies are more explorative by giving unproven actions frequent chances.
- With further exploration, the RL system learns more about the environment and hence learns the states and state-action pairs that are most promising.
- This information can make π more knowledgeable in its attempt to balance exploration and exploitation
- So RL **can** use this info to modify π **during** training.

Off-Policy -vs- On-Policy RL

- **On-Policy RL** uses this experience to update π, and π's action-selection probabilities are taken into account when updating state and/or state-action values. For example, the value of state S is based on the expected values of its neighbor states, with the neighbor values weighted by the actual probability of moving from S to those states. Those probs are given by $\pi +$ the environmental model.
- **Off-Policy RL** does not take the policy into account when updating the value function. However, the dynamically-updated value function may still influence the behavior-generating policy. However, off-policy methods do maintain 2 policies:
  - The behavior policy, π', governs **all** actions during training.
  - The estimation policy, π, is updated by experience and then used for post-training tests.

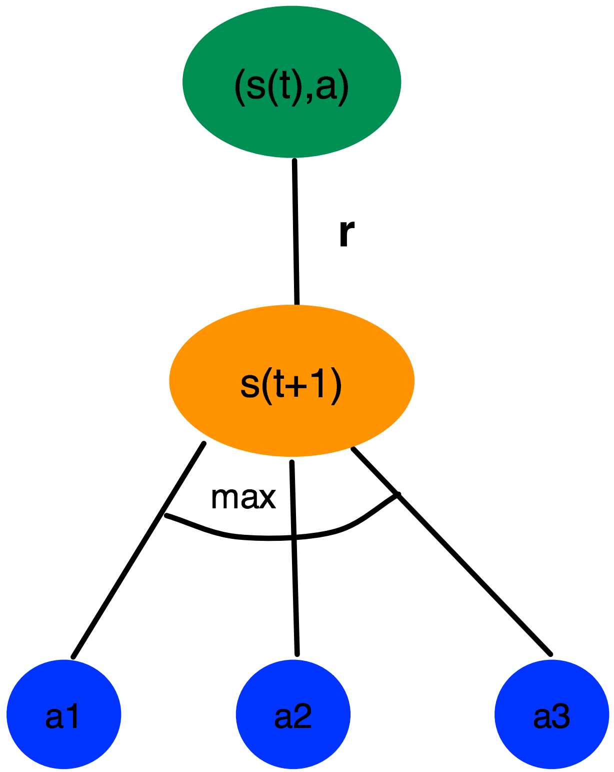
Q-Learning

Off-Policy Temporal Difference RL (Watkins, 1989)
Goal: Learn Q(s,a) table = Evaluations of using all actions in all states.

- Init Q(s,a) arbitrarily and choose a policy, π, to govern search.
- Repeat for each search episode
  - Begin at a start state
  - Repeat until a terminal state is entered.
    - Choose a from s using policy derived from Q
    - Do action a
    - Observe reward, r, and next state s'
    - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
    - $s \leftarrow s'$

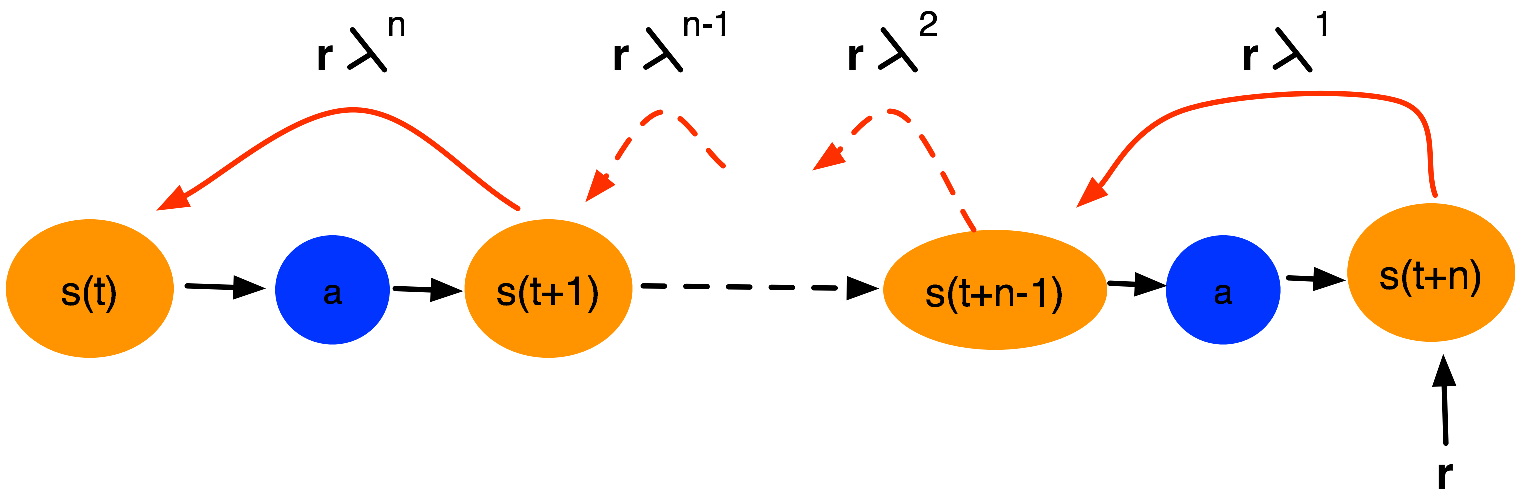
Considered **off-policy**, since the Q(s,a) updates take only the future value based on the best action from state s' into account, not the expected value based on the action-selection probabilities housed in the behavioral policy.

Q-Learning Backup Diagram



Eligibility Traces

- Primary means of **Temporal Credit Assignment**: giving credit/blame to states and acts that occur much earlier in a sequence that eventually achieves a primary reinforcement.
- The reinforcement is *passed back* along the sequence, discounted by the factor λ at each step.
- For $\lambda < 1$, this means that much earlier states only get a small portion of the original reward.
- Most forms of RL can be easily augmented with eligibility traces.



Implementing Traces

The trace for a state or s-a pair decays with each timestep unless that state/pair is the one getting primary reinforcement.

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } (s,a) \neq \text{newly reinforced pair} \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

These are called *replacing traces*.

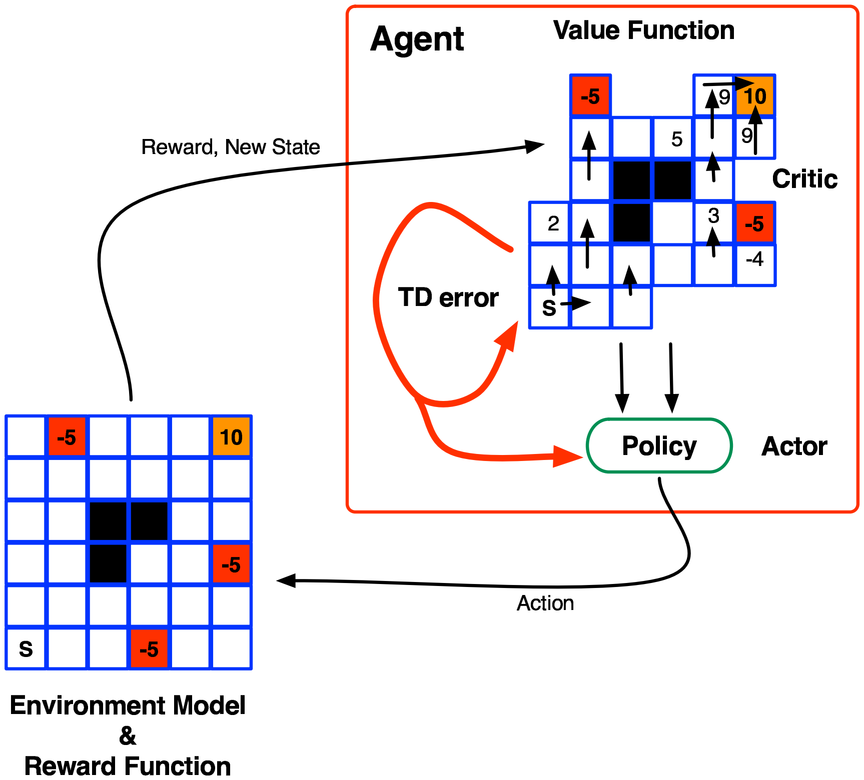
Consider Q-Learning with replacing traces.
After TD-error, δ is computed for the current s-a pair:

1. Every s-a pair updates its eligibility (see above)
2. Every s-a pair updates its value using the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a) \tag{6}$$

So every s-a pair gets a share of the new TD error.

Actor-Critic Methods



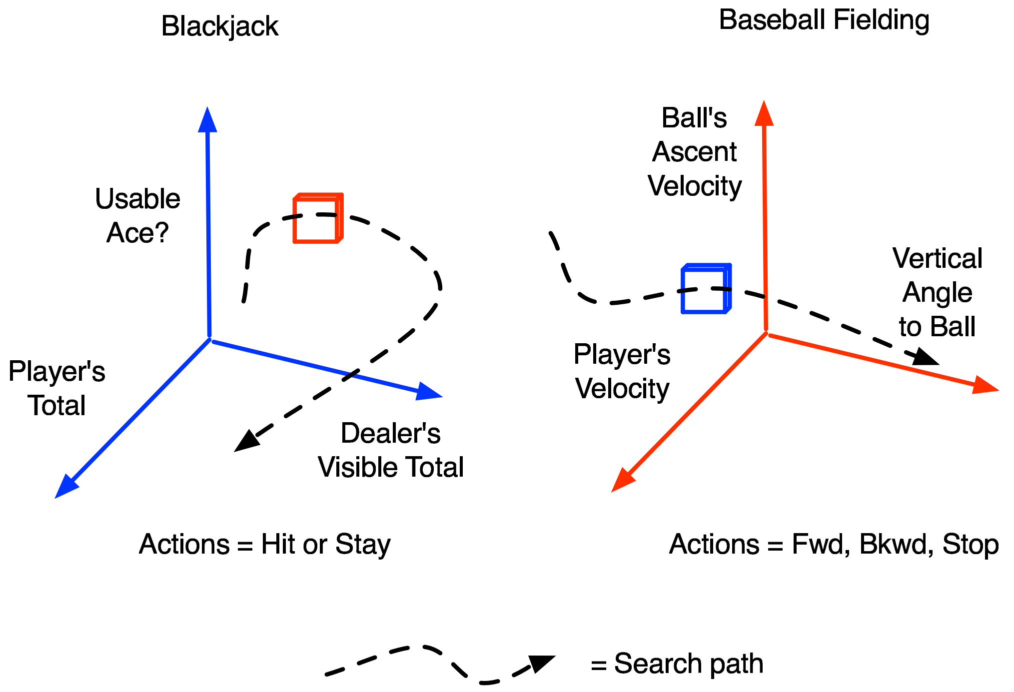
- TD methods with two separate (but interacting) data structures for the policy (actor) and the value function (critic).
- Always on-policy learning, where critic uses TD-error to critique the actions chosen by the actor, since the standard value-function updates affect the policy.

Actor-Critic Interaction

- Use the TD error from time t, δ_t, to modify the value function (critic) using the standard TD update for V(s) or Q(s,a).
- Also use δ_t to modify the policy (actor).
- This is done by changing the **preference** for taking action *a* while in state s_t: $p(s_t, a)$.
  - If $\delta_t > 0$, then action *a* brought us to a **better** state, so $p(s_t, a) \uparrow$.
  - If $\delta_t < 0$, then action *a* brought us to a **worse** state, so $p(s_t, a) \downarrow$.
- Normally, the preferences will be tightly tied to V(s) or Q(s,a), so critic updates will automatically lead to policy changes, and thus changes in the actor's behavior, i.e. changes to its action-choice decision making.

RL State-Spaces

Grid world's are classic RL examples. But state spaces need not have a nice correlation to real-world 2- or 3-d space.



Major Problem: Most real-world search spaces have too many important features, with too many relevant values, to be exhaustively searched by an RL system. Value functions will only reflect a limited set of exploratory experiences.

Combatting Exponential Search Spaces

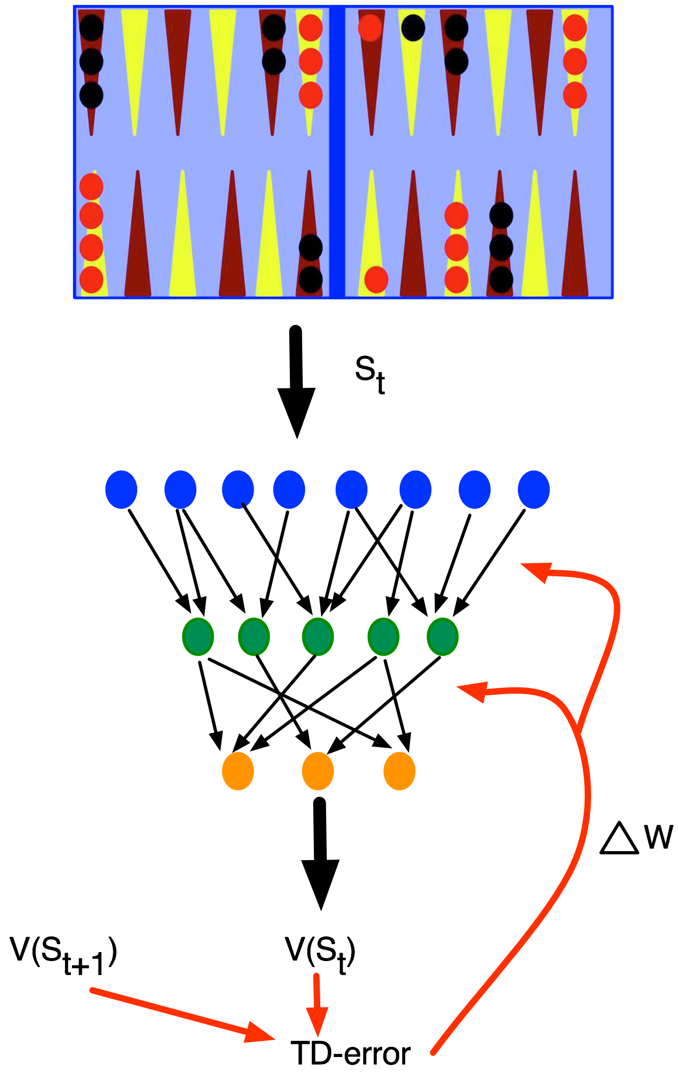
- In complex domains, each state (or state-action pair) cannot be evaluated. Even a very exploratory system will only experience a small fraction of the total state space.
- But from a sample of states and their evaluations, the system should be able to **generalize** and create a value function that can give an accurate evaluation of any possible state.
- This is a classic case of **supervised** learning, and any of the tools of supervised function learning are applicable: neural networks, decision trees, bayesian classifiers, etc.
- So a component of a reinforcement learning system is a supervised learning (SL) system!
- The instances/cases used in RL's SL are the (state,eval) or (state,act,eval) groups that are generated during **backup operations**, i.e. the value updates to the states that have been encountered.

TD-Gammon: RL Backgammon Player

Gerald Tesauro (1994). *TD-Gammon, a self-teaching backgammon program, achieves master-level play*. **Neural Computation**, 6(2):215-19.

- Uses artificial neural network (ANN) to learn the Value function, V(s).
- ANN input = board state, s_t
- ANN output $= V(s_t)$
- Move choice based on $V(s_{t+1})$ for all possible next states, s_{t+1}.
- Reinforcement comes only at end of game, i.e. after 50-100 (or more) moves.
- Incredible results!
  - Beats, or nearly beats best humans in the world.
  - Some TD-gammon strategies adopted by the top humans.

Gradient Descent Learning of Value Functions



Weight Updates

$$\triangle w_t = \alpha(V(s_{t+1}) - V(s_t)) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V(s_k) \tag{7}$$

- $\triangle w_t$ = changes to weights in the neural network.
- α = learning rate
- $V(s_{t+1}) - V(s_t)$ = standard TD-error (assuming no reward - which only comes at end of the game, where you use $R - V(s_t)$ instead)
- λ = decay of the eligibility trace
- $\nabla_w V(s_k) = \frac{\partial V(s_k)}{\partial w}$ = Change in output (on input state s_k) with respect to changes in the neural network weights.

Update the weights so that:

1. $V(s_t)$ moves closer to $V(s_{t+1})$
2. The values for each of the states $s_1 \ldots s_{t-1}$ in the sequence also move closer to the values indicated by their temporal differences, but with less influence by the earlier states (when $\lambda < 1$).

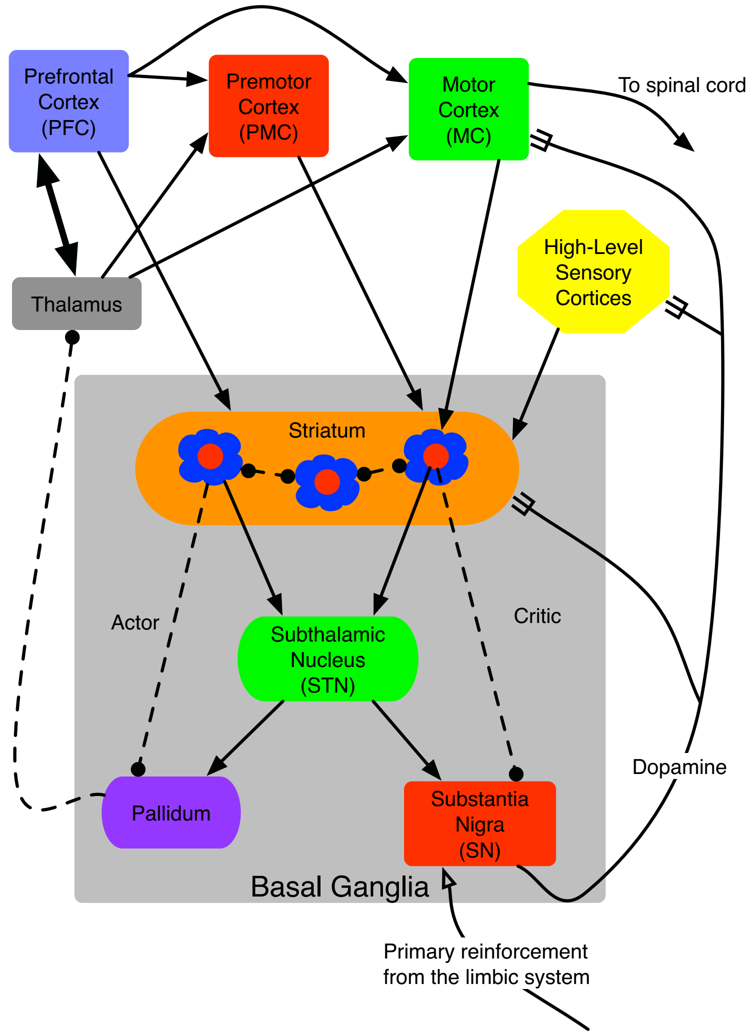
Weight Updates (2)

- So the values for each of states $s_1 \ldots s_t$ are updated by making changes to the SAME neural network.
- When $\lambda < 1$, later states (which are closer to the reinforcement) have a greater effect upon network adjustment than do earlier states.
- Hence, in the beginning of training, the network will produce reasonably accurate value estimates for later (but not earlier) states.
- Over time, the later states will be nearly correct and thus demand little change to the network.
- At this point, the earlier states will begin to influence the network, and their values will be accurately produced by the network as well.
- This is the same late-to-early state-value learning found in standard RL, but now, instead of storing all the values in a table, they are produced by a single function.

Actor-Critic RL in the Brain

- Barto (1998). *Adaptive Critics and the Basal Ganglia*.
- Houk, Adams and Barto (1998). *A Model of How the Basal Ganglia Generate and Use Neural signals that Predict Reinforcement*.
- Much evidence implicates the basal ganglia (BG) in both high-level action selection (actor) and in modification of action-selection circuitry based on reinforcement signals (critic).
- BG circuitry and biochemistry supports a dynamic very similar to the TD update equation, where predictions are updated based on both later predictions and real states of reward/punishment.

The Basal Ganglia



The Actor Circuit

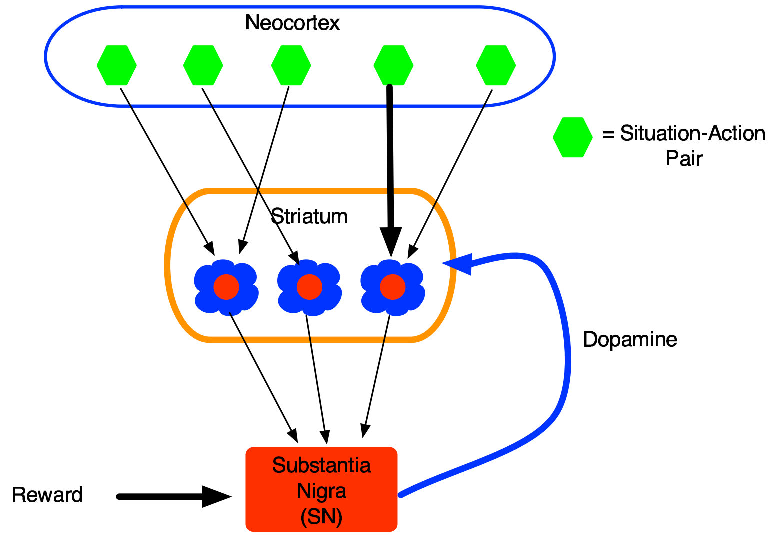
- Striatal neurons behave like competing conjunction detectors:
  - Have thousands of converging inputs from all over the brain.
  - Require many incoming signals in order to reach threshold and fire.
  - Sensitive to neuromodulation via dopamine $\longrightarrow$ Hebbian learning of salient cortical firing patterns.
  - Inhibit one another.
- Pallidal neurons behave like competing disjunction detectors:
  - Tonic activity - often firing.
  - Easily blocked by one or a few inhibitory signals.
  - Inhibit one another.
- Striatal cells may detect situation-action pairs and then send signals to pallidal cells that control activation of the corresponding action.
- Striatal firing $\rightarrow$ gating of intended actions into the prefrontal cortex via double inhibition: striatum $\xrightarrow{-}$ pallidum $\xrightarrow{-}$ thalamus $\xrightarrow{+}$ prefrontal cortex.

The Critic Circuit

RL of the Value Function in the Basal Ganglia

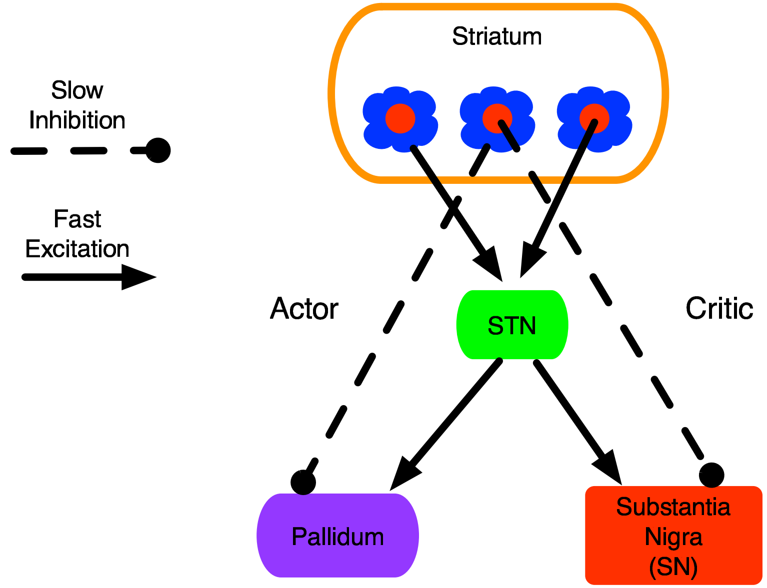
- The BG resembles a Q-Learner, since the contexts being evaluated are combinations of situations and potential actions, i.e. an s-a pair.
- Value(s-a pair) $\approx$ firing strength of the striatal neurons that detect it.
- These strengths can increase if the neuron fires about 100 ms prior to a dopamine signal.
- That is, 100 ms prior to either a) an actual reward or b) the occurrence of another s-a pair that has a strong enough striatal activation to excite the Substantia Nigra (i.e. an s-a pair with a high value).
- So this is just like TD learning: states that precede good states increase in value.
- During skill acquisition, the mind explores many possibilities. Thus, s-a pairs compete for control of striatal neurons, and those that slightly precede dopamine signals gain a foothold and become part of the skill-performing sequence.

Learning Salient Situation-Action Pairs



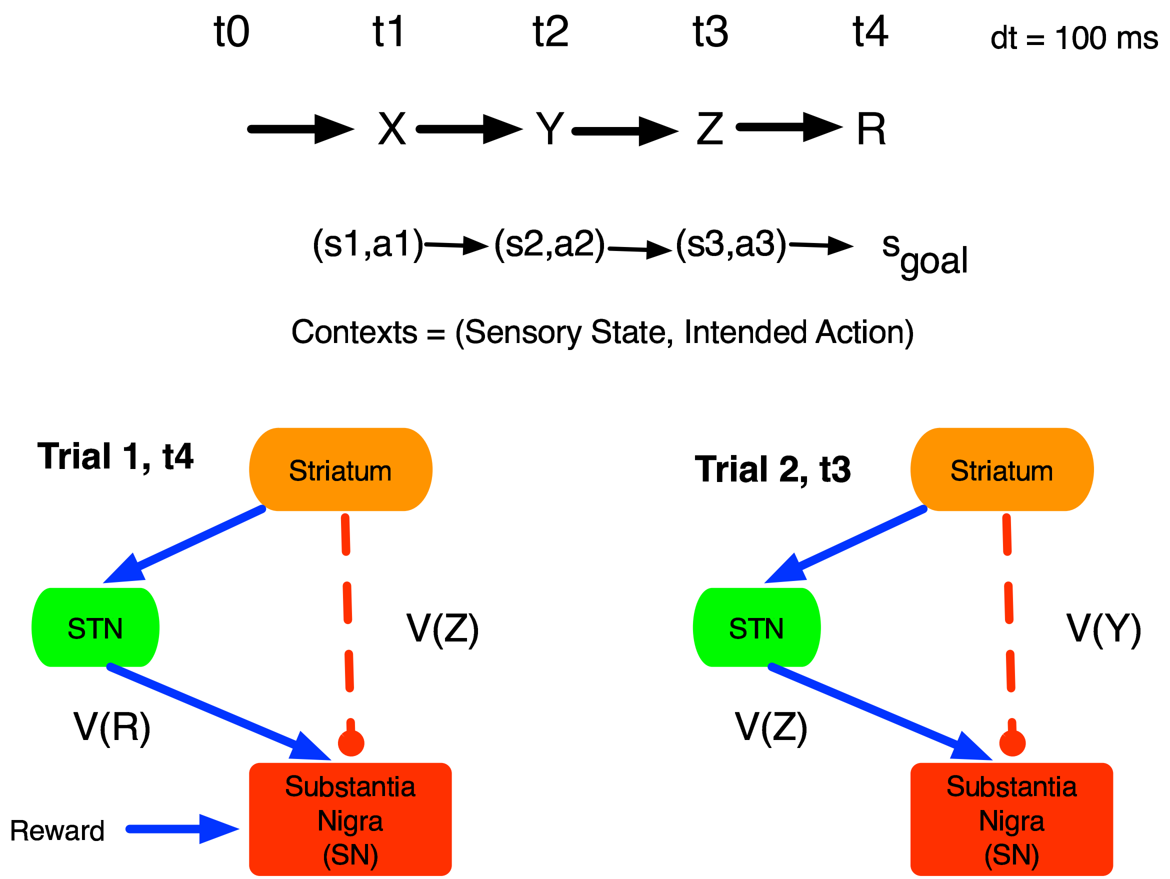
- Situation-action pairs that are active just before ($\approx 100ms$) the reward will be remembered by the Striatum via Hebbian synaptic strengthening.
- Once learned, these pairs will trigger the Substantia Nigra, which then broadcasts dopamine. But now it is **predicting** the reward.
- The 100 ms delay is due to some fascinating chemistry. It's the key to learning *causal* relationships.

Fast and Slow Pathways



- A striatal neuron, whether striosomal or matriosomal, will initially stimulate an SNc (or Pallidal) neuron via the faster STN pathway, but then it will inhibit it by the slower direct pathway.
- Inhibition has longer duration than stimulation.
- Note: The striatal $\rightarrow$ STN pathway is actually a chain of two inhibitory links: striatum $\stackrel{-}{\rightarrow}$ external pallidum (EP) $\stackrel{-}{\rightarrow}$ STN. All striatal outputs are inhibitory. STN is the only promotor in the BG.

TD Learning in the Basal Ganglia



$$\triangle Q(s_t, a_t) = r_{t+1} + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{8}$$

Learning a Predictor Sequence