Reinforcement Learning in Neural Networks

Keith L. Downing

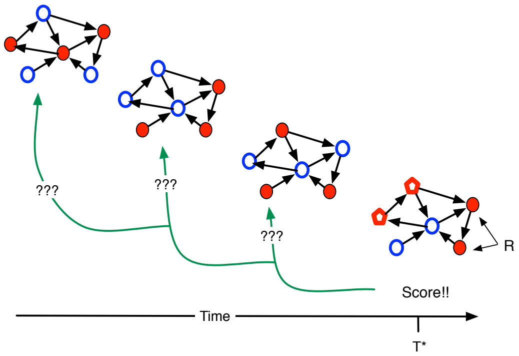
The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@idi.ntnu.no

April 20, 2017

Reinforcement Learning (RL)

- **Occasional** feedback from an instructor, indicating only right/wrong, but **not** the **correct** answer/response for each context.
- Problem solutions consist of many steps/stages, but the reinforcement (= reward or penalty) comes only at the end of the sequence and/or after only some of the steps.
- Credit assignment problem - how to give credit/blame to intermediate steps?
- RL Theory is well documented (*Reinforcement Learning*, Sutton and Barto (1998)) and is totally independent of neural network research.
- Typical usage: problems with a) limited possibilities for intermediate feedback, but b) concrete feedback at sequence end.
- For example: games (Backgammon, Othello, Checkers...) and robotics.
- ANNs involve RL in many different ways, although typically, the ANN realizes RL's the value function.

Credit Assignment

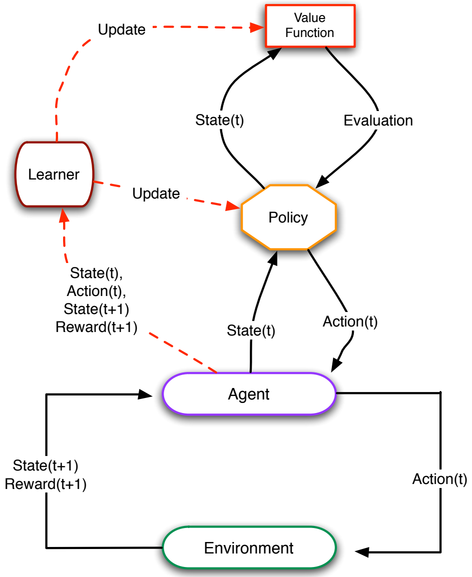


Assign credit or blame to network components that a) were not active at the same **time** as the reward/punishment, T*, or b) have different **spatial** locations than the components that were active at T*

RL = ”On the Job Training”

- The RL system uses a control strategy for choosing actions in contexts,
- while simultaneously building a policy = mapping from contexts to **most appropriate** actions.
- The policy uses a value function = mapping from either a state (or a state-action pair) to the total, expected **future** reinforcement for an agent currently in that state.

On-Policy RL Architecture



On-Policy -vs- Off-Policy RL

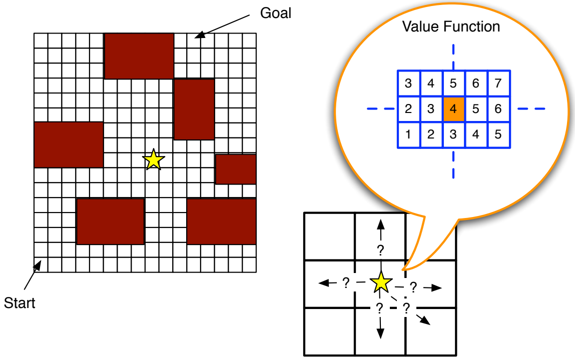
On-Policy

- Current policy determines control choices and value-function updates.
- Value function (and hence policy) updated after each move.
- Exploitation dominates exploration
- Hence, early actions based on an unrefined policy

Off-Policy

- Control decisions and value-function updates are independent of the current policy.
- Value function (and hence policy) are still updated after each move.
- More explorative, trying to investigate all possible moves instead of just doing those that, so far, seem best for a given context.
- Early and late actions based on general quest to gain more knowledge about the environment and thus improve the value function and policy.

Classic RL Problem: Maze Search



- A complete value function covers all (of the many) states.
- A good policy bases action choices on a good value function: choose the action that leads to the highest-valued, successor state.

Large RL Search Spaces:

State space = cross product of all variable states.

Robotics

1. Sensor readings
2. Location, orientation, velocity, acceleration
3. Battery level

Texas Hold ’Em Poker

1. A player’s hole cards, of which there are $\binom{52}{2}$ possibilities.
2. Number of raises in the current betting round.
3. Number of players still active in the current betting round.
4. Money in the pot (preferably discretized into k different bins)
5. The (discretized and binned) strength rating of the player’s hand - based on her hole cards plus the mutual cards (flop, turn, river).
6. Number of active players known to bluff frequently

The Reinforcement in RL

- r_t = reinforcement (a.k.a. reward) at time t.
- R_t = cumulative reward from time t until episode termination time (T).

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_T$$

Normally in RL, future rewards are **discounted** by a factor, γ, yielding:

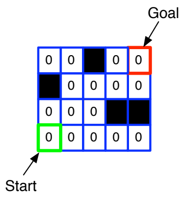
$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T = R_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$$

Value function = mapping from states (or state-action pairs in Q-Learning) to expected R_t, where rewards are a function of the policy, Π:

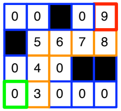
$$V^{\Pi}(s) = E_{\Pi}\{R_t | s_t = s\} = E_{\Pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\}$$

$$Q^{\Pi}(s,a) = E_{\Pi}\{R_t | s_t = s, a_t = a\} = E_{\Pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\}$$

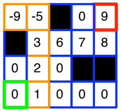
Updating the Value Function via Experience



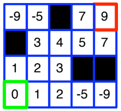
Reach Goal + Backup Reward



Reach Deadend + Backup Penalty



After Many Exploratory Rounds



The Bellman Equation

A set of constraints, one per state, s, that should hold, given policy Π:

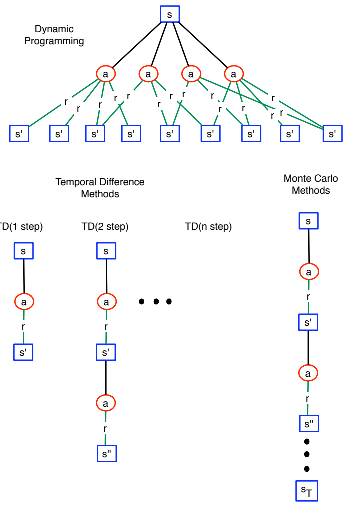
$$V^{\Pi}(s) = \sum_{a} \Pi(s,a) \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\Pi}(s')]$$

where:

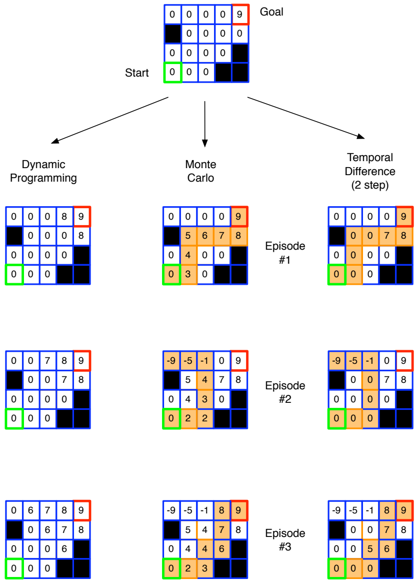
1. $\Pi(s,a)$ = prob choosing action a in state s (under policy Π),
2. $P^{a}_{ss'}$ = prob transitioning from state s to s' on action a,
3. $R^{a}_{ss'}$ = immediate reward received on that transition,
4. γ = the discount factor.

Goal of RL = satisfy all Bellman equations (after many rounds of problem solving), but in practice, search spaces are too large. Still useful as framework for backing up state-value information during problem solving.

Backup Schemes



3 Successive Backup Rounds for 3 RL Approaches



Generalized Policy Iteration (GPI): 2 alternating calcs

Policy Evaluation (PE)

Updating V(s) based on the current policy, Π, using, e.g., the Bellman optimality equation.

$$V(s) \leftarrow \sum_{s'} P_{ss'}^{\Pi(s)}[R_{ss'}^{\Pi(s)} + \gamma V(s')]$$

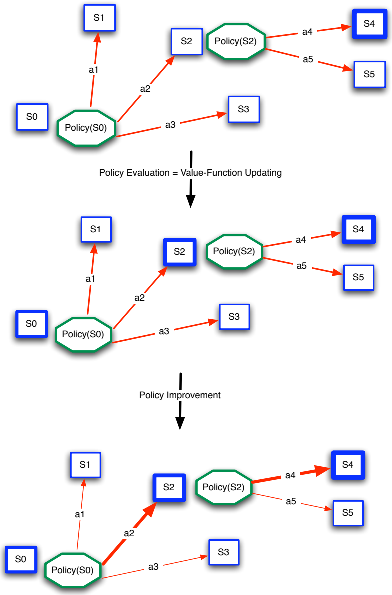
Policy Improvement (PI)

Updating of Π based on V(s); e.g., greedy selection of the action leading to the highest-valued successor.

$$\Pi(s) \leftarrow \operatorname*{argmax}_a P_{ss'}^{a}[R_{ss'}^{a} + \gamma V(s')]$$

In contrast to the Actor-Critic model (shown later), the PI step directly involves the evaluation function, V(s) $\rightarrow$ Actor (handler of Π) and critic (handler of V(s)) are not so well-separated in GPI.

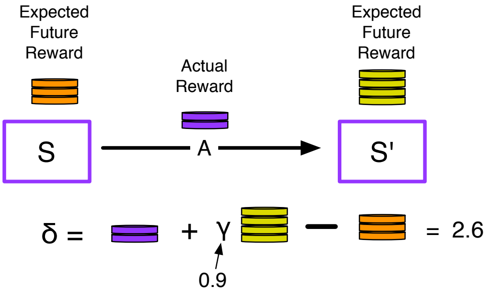
Generalized Policy Iteration



Temporal Difference (TD) Learning

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

- α = learning rate,
- γ = discount factor,
- r = reward after moving from state s to s'
- $\delta = r + \gamma V(s') - V(s)$ = TD Error = Level of **Surprise**



Q-Learning (Watkins, 1992)

Off-policy TD using state-action pairs

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- $Q(s_t, a_t)$ = Expected total remaining reward if performing action a_t in state s_t.
- γ = discount rate
- α = learning rate
- $\delta = r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ = TD Error = **Surprise**

This is **off-policy** due to the use of $max_a Q(s_{t+1}, a)$, which says to base the update on the **best possible** move from s_{t+1}, not necessarily on the move sanctioned by the current policy.

Eligibility Traces

Instead of a discrete number of backup steps, all states have an eligibility for update which gradually decays if the state is not visited during problem-solving search.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

- s_t = state of system at the current time, t.
- γ = discount rate
- λ = trace-decay factor

Now, instead of just the last k (visited) states, ALL states (in the current episode) are updated, although many by a very small amount.

TD(λ) = TD Learning with Eligibility Traces

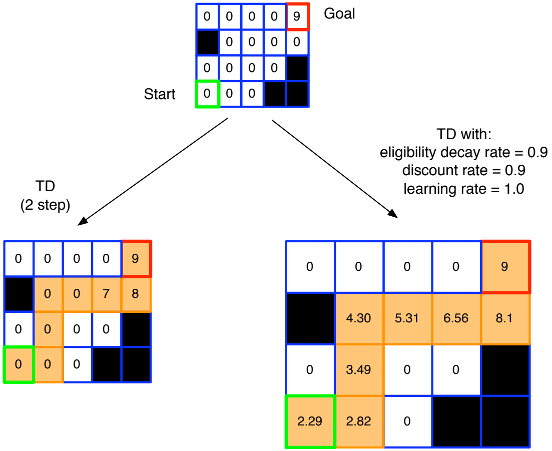
For each step of a problem-solving episode, do:

1. $a \leftarrow$ the action with highest probability in the policy component $\Pi(s)$
2. Performing action a from state s moves the system to state s' and achieves the immediate reinforcement r.
3. $\delta \leftarrow r + \gamma V(s') - V(s)$ (TD Error)
4. e(s) $\leftarrow$ e(s) + 1
5. $\forall s \in S$
    1. V(s) $\leftarrow$ V(s) + $\alpha\delta e(s)$ (Value Update)
    2. e(s) $\leftarrow \gamma\lambda e(s)$

Similar for Q-Learning, but with:

- $\delta = r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ (TD Error)
- $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$ (Value Update)

TD(λ) -vs- TD(2)

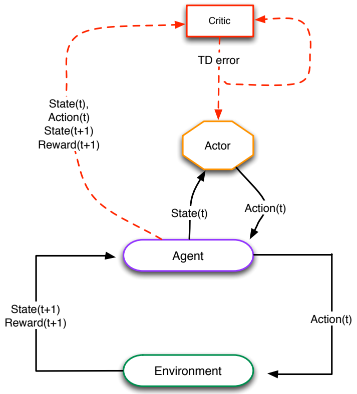


0	0	0	0	9
■	0	0	0	0
0	0	0	0	■
0	0	0	■	■

0	0	0	0	9
■	0	0	7	8
0	0	0	0	■
0	0	0	■	■

0	0	0	0	9
■	4.30	5.31	6.56	8.1
0	3.49	0	0	■
2.29	2.82	0	■	■

Actor-Critic Paradigm



- Critic handles value function and its updates.
- Actor handles the policy and updates it based on TD error sent by critic.

Actor-Critic Policy Updates

Transition probabilities modified by TD error (δ) and eligibility trace, e(s,a):

$$\Pi(s,a) \leftarrow \Pi(s,a) + \alpha \delta e(s,a)$$

Transition probabilities normalized:

$$\Pi(s,a) \leftarrow \frac{\Pi(s,a)}{\sum_{a' \in \Pi(s)} \Pi(s,a')}$$

In contrast to the Generalized Policy Iterator (GPI), the actor-critic policy updates are only indirectly based on V(s), since the actor only receives δ, while the critic computes δ from V(s) and V(s').

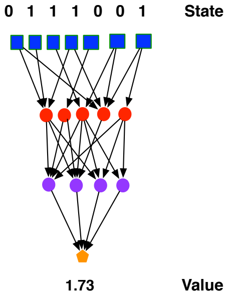
The Actor-Critic Model and Neural Networks

1. Temporal differencing is the most feasible approach to state and policy updating, due to:
    - The enormous search-space size of typical ANN problem domains, which make comprehensive updates very costly.
    - The difficulty of wiring up neural circuitry to handle DP or MC approaches.
2. Eligibility traces are a simple bookkeeping mechanism requiring no extra caches (of, for example, all states in the current episode) nor (temporary) rewiring of the network.
3. The actor-critic separation of V(s) from $\Pi(s)$ maps nicely to modular neural networks, whereas a tightly integrated combination of the two does not.

Replacing Tables with Neural Nets

State	Value
000...000	2.13
001...011	1.45
011...010	0.05
101...110	1.14

●
●
●



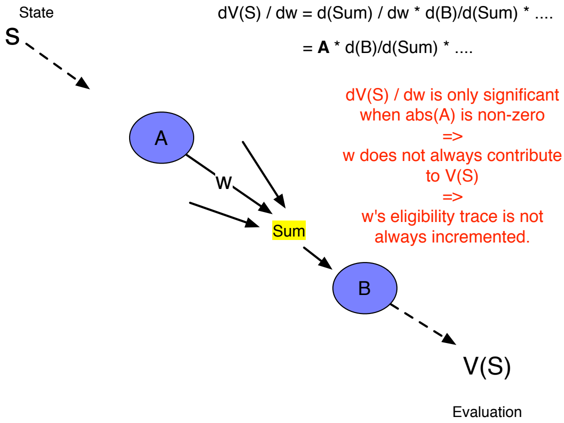
- Explosion of possible states $\rightarrow$ tables are impossible to build.
- Neural net embodies a function from states to evaluations (critic) or states to actions (actor).
- These functions handle every possible state as input.
- Key generality ssumption: For the vast majority of cases, similar states map to similar values.

Eligibility Traces in Backprop Nets

Eligibilities are associated with each weight. These decay on each step and are incremented when $\frac{\partial V(s)}{\partial w}$ is non-zero.

$$w_j \leftarrow w_j + \alpha \delta e_j$$

$$e_j \leftarrow e_j + \frac{\partial V(s_t)}{\partial w_j}$$

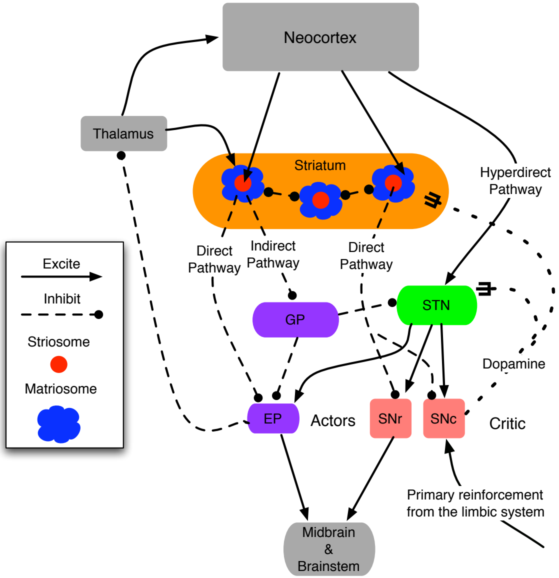


dV(S) / dw = d(Sum) / dw * d(B)/d(Sum) * ....

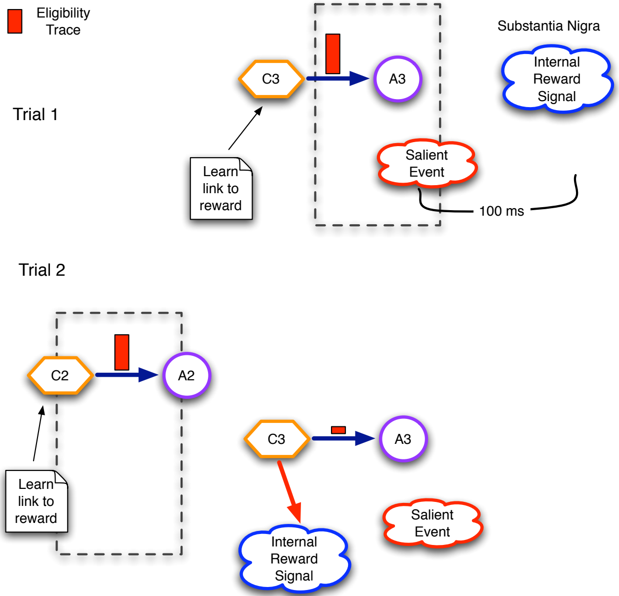
= **A** * d(B)/d(Sum) * ....

dV(S) / dw is only significant when abs(A) is non-zero => w does not always contribute to V(S) => w's eligibility trace is not always incremented.

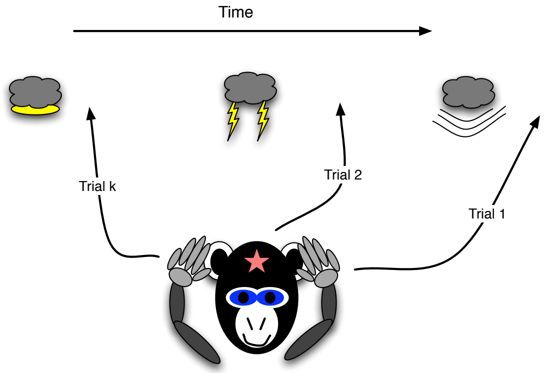
Actor-Critic Model of the Basal Ganglia



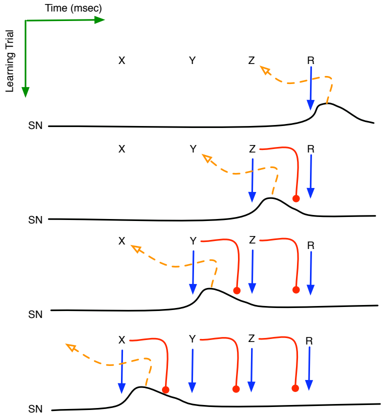
Eligibility Traces in the Basal Ganglia



Temporal Regression of Predictions

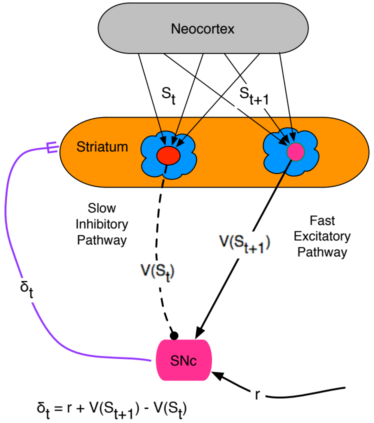


Temporal Regression via the Basal Ganglia



Only surprise (i.e. TD error) evokes dopamine signalling.

Computing TD error in the Basal Ganglia



- V(s) $\approx$ firing strength, F_s, of the context detector for s, C_s.
- F_s determined by strength of synapses entering C_s, which dopamine-induced learning has modified.

Interpretations of the Basal Ganglia

Neuroscientist

The basal ganglia does reinforcement learning due to:

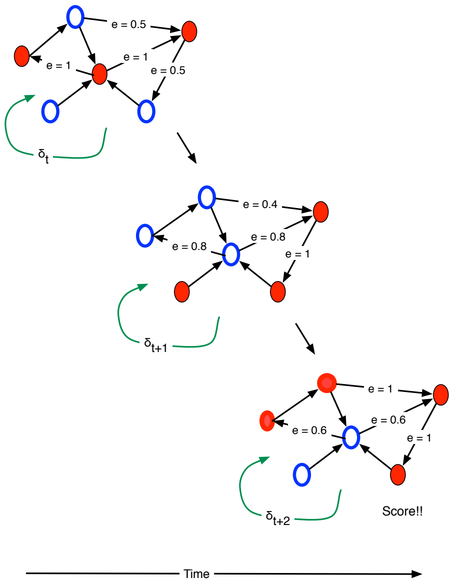
- a dopamine signal driven by surprise
- synaptic modification (e.g. learning) based on that signal

Artificial Intelligence Researcher

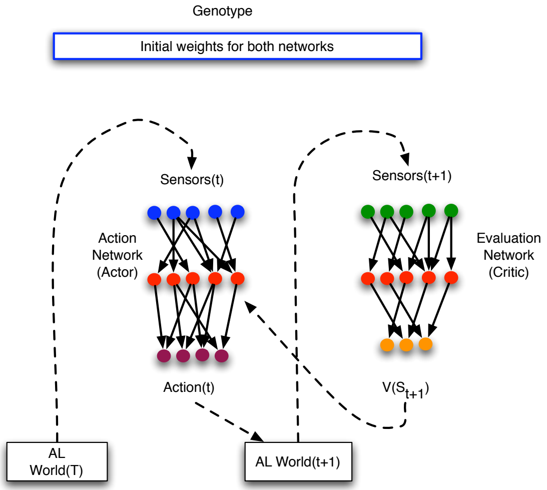
The basal ganglia does reinforcement learning due to:

- calculation of TD error in the Substantia Nigra
- use of TD error to modify the incoming connection weights to context detectors in the striatum, which affects the values associated with those contexts.

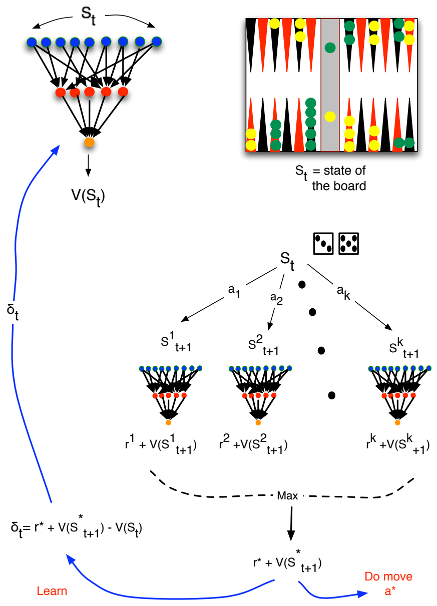
ANN Credit Assignment via Eligibility and TD Error



Ackley and Littman's (1992) Actor-Critic ANN Pair



TD-Gammon (Tesauro, 1995)



General Observations

ANN as Value function

Many search spaces are too large to yield accurate V(s) entries via standard RL exploration. ANNs are an adaptive approximator of V(s) that allow systems to boostrap to effective policies.

RL to Supervised Learning via TD

1. In TD, each move leads to a comparison of expectations to slightly more informed expectations, and any difference between the two is a **surprise** that leads to learning.
2. If an ANN (or any system) can predict future states (based on current states and actions), then
3. when the future arrives, reality serves as the *correct answer*, which, when compared to the prediction, provides an error term, which can guide learning.
4. This can be done at every timestep.
5. So prediction + comparison to reality turns RL into supervised learning.

Not as bad as it looks

- On the surface, RL looks hopelessly resource consuming, since many trials are required to gain and backup information about each state.
- Monte Carlo methods and TD learning greatly simplify things.
- TD methods allow a system to bootstrap its way to intelligence.
- This was a key breakthrough in RL applicability to complex problems.
- It also allows plausible interpretations of the brain (e.g. basal ganglia) as an actor-critic reinforcement learner.