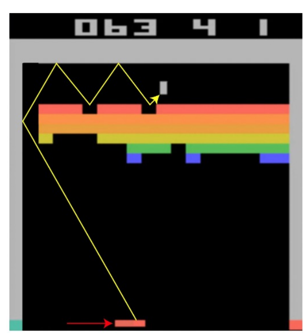
Model Free RL

Ron Parr
CSCI 2951-F
Brown University

With thanks to Kris Hauser for some content, and Dillon Sandhu for edits

Notable Model Free RL Successes

- Most impressive examples are games:
  - Backgammon (1993) and Go (2016)
  - Atari games (2014)
  - StarCraft II (2019)
  - Dogfighting in realistic simulators (2020)



- Sutton & Barto RL Book is one of the most cited references in CS (~68K citations as of Winter 2024)

Image source: Max Tegmark

Comparison w/Other Kinds of Learning

- Machine Learning is split into:
  - Supervised: predicting human-labelled data
  - Unsupervised: Discovering patterns in high dimensional data
  - Reinforcement Learning: maximizing a reward that may be delayed
- What the last thing that happens before an accident?

Why We Need Model Free RL

- We would like to solve an MDP, but…
- Where do we get transition probabilities?
- How do we store them?
  - Big problems have big models
  - Model size is quadratic in state space size
- Where do we get the reward function?

RL Framework

- Learn an optimal policy by “trial and error”
- No assumptions about model
- No assumptions about reward function
- Can assume:
  - True state is known at all times
  - Immediate reward is known
  - Discount is known

RL for Our Game Show

- Problem: We don’t know probability of answering correctly

Source: Wikipedia page
For “Who Wants to be a Millionaire”

- Solution:
  - Buy the home version of the game
  - Practice on the home game to refine our strategy
  - Deploy strategy when we play the real game

First steps: "Batch" Learning

- Observe execution **trials** of an agent that acts according to some unobserved policy π
- Problem: estimate V^π ,the value function for this policy
- Important view of $V^\pi(s)$
  - Recall $V^\pi(s)$ is the ***expected, discounted value*** of following policy π from state s
  - $V^\pi(s) = E\Sigma_{St}[\gamma^t R(S_t)]$ where S_t is the random variable denoting the distribution of states at time t

Batch RL

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

→

3	0.81	0.87	0.92	+1
2	0.76		0.66	-1
1	0.71	0.66	0.61	0.39
	1	2	3	4

1. Observe trials $t^{(i)}=(s_0^{(i)},a_1^{(i)},s_1^{(i)},r_1^{(i)},\ldots,a_{ti}^{(i)},s_{ti}^{(i)},r_{ti}^{(i)})$ for i=1,…,n
2. For each state $s \in S$:
   3. Find all trials $t^{(i)}$ that pass through s at, e.g., time step k
   4. Compute value $V^{t(i)}(s)=\Sigma_{t=k \text{ to } ti}\gamma^{t-k} r_t^{(i)}$
   5. Set $V^\pi(s)$ to the average observed values

Note: This is called Monte Carlo Evaluation

Downsides of Batch RL

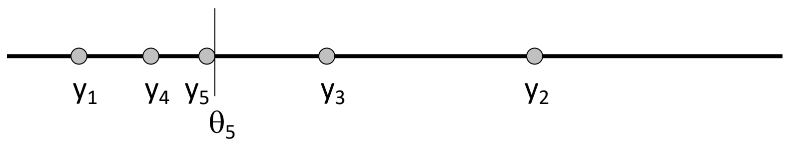
- Batch approach:
  - Store **all data** for trials 1…n
  - Use all data to compute value after each trial
  - Storage and computation grow monotonically
- We would like to use less space and memory
- We would like to learn immediately from experience, instead of waiting until the end of a trajectory

Incremental ("Online") Learning

- Generic description (not just RL)
- Data is streaming into learner:
  $x_1,y_1, \ldots, x_n,y_n \quad y_i = f(x_i)$
- Problem: keep a running estimate of f(x) to predict y_{n+1} from x_{n+1}

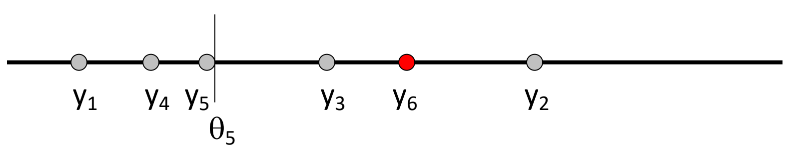
Example: Online Mean Estimation

- $y_i = \theta$ + noise (constant - no x's)
- Estimate the mean θ_n from the noisy observations $y_1, \ldots, y_n$



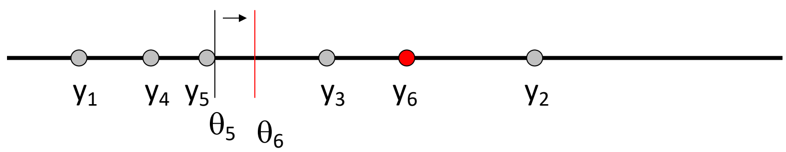
Example: Online Mean Estimation

- Now we observe y_6
- What is our new mean, θ_6?



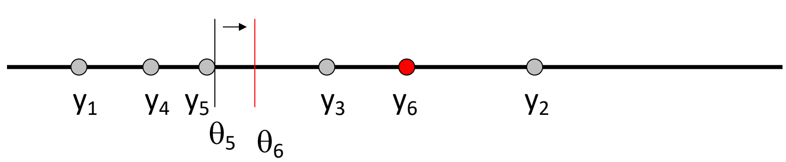
Example: Online Mean Estimation

- Update:
- $\theta_6 = 5/6\ \theta_5 + 1/6\ y_6$



Example: Online Mean Estimation

- $\theta_6 = 5/6\ \theta_5 + 1/6\ y_6$
- In general, $\theta_{n+1} = (n/n+1)\ \theta_n + 1/(n+1) y_{n+1}$
  $= \theta_n + 1/(n+1)\ (y_{n+1} - \theta_n)$



Proof of Online Mean Update

- Current estimate:
- $\theta_n = 1/n \ \Sigma_{i=1\ldots n} \ y_i$

- Next Estimate:
- $\theta_{n+1} = 1/(n+1) \ \Sigma_{i=1\ldots n+1} \ y_i$
  $= 1/(n+1) \ (y_{n+1} + \Sigma_{i=1\ldots n} \ y_i)$
  $= 1/(n+1) \ (y_{n+1} + n \ \theta_n)$
  $= 1/(n+1) \ (y_{n+1} + (n+1) \ \theta_n - \theta_n)$
  $= \theta_n + 1/(n+1) \ (y_{n+1} - \theta_n)$

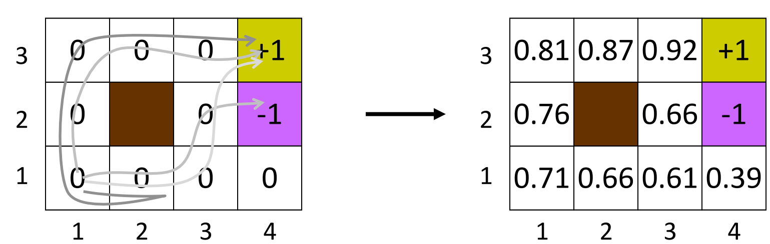
Learning Rates

- In fact, $\theta_{n+1} = \theta_n + \alpha_n \ (y_{n+1} - \theta_n)$ converges to the mean for any α_n such that:
  - $\alpha_n \to 0$ as $n \to \infty$
  - $\Sigma\alpha_n \to \infty$
  - $\Sigma\alpha_n^2 \to C < \infty$
- $\alpha_n = O(1/n)$ does the trick
- If α_n is close to 1, then the estimate shifts strongly to recent data; close to 0, and the old estimate is preserved

Learning Rates in RL in Practice

- Maintain a per-state count N[s]
- Learning rate is function of N[s], α(N[s])
- Sufficient to satisfy theory: α(N[s])=1/N(s)
- 1/N(s) often viewed as too slow
  - α drops quickly
  - Convergence is slow
- In practice, often a floor on, α, e.g., α = 0.01
- Floor leads to faster learning, but less stability

Incremental Value Estimation



1. Store counts N[s] and estimated values $V^{\pi}(s)$ (initialize to 0, typically)
2. After a trial t, for each state s in the trial:
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Calculate $V^{t}(s)$ using discounted sum of rewards
   5. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(V^{t}(s)-V^{\pi}(s))$

- Doesn't require storing all trajectories, but…
  - Simple averaging
  - Slow learning, because Bellman equation is not used to pass knowledge between adjacent states

Temporal Difference (TD) Learning

- Learn directly from a partial episode (i.e. a single step)
- Update towards the value given by the Bellman Equation, *using current estimate for next state*
- Trial-Based Value Estimation:

$$V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(\Sigma_i \gamma^i r_i - V^{\pi}(s))\ ^{*}$$

- TD Learning:

$$V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$$

* Sum is sum of observed rewards until end of episode

Temporal Difference Learning

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

$$V_{t+1}(s) = R(s) + \gamma \sum_{s' \in Succ(s,a)} P(s'|s,a) V_t(s')$$

Online estimation of mean over value next states

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
   3. Set $N[s] \leftarrow N[s]+1$
   4. Adjust value $V^{\pi}(s) \leftarrow \boxed{V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))}$

- Instead of averaging at the level of trajectories…
- Average at the level of states

Temporal Difference Learning

3	0	0	0	+1
2	0		0	-1
1	0	0	0	0
	1	2	3	4

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
   3. Set $N[s] \leftarrow N[s]+1$
   4. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$

Temporal Difference Learning

3	0	0	0	+1
2	0		0	-1
1	-0.02	0	0	0
	1	2	3	4

With learning rate
α=0.5

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
   3. Set $N[s] \leftarrow N[s]+1$
   4. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$

Temporal Difference Learning

3	-0.02	0.02	0	+1
2	-0.02		0	-1
1	-0.02	0	0	0
	1	2	3	4

With learning rate
α=0.5

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$

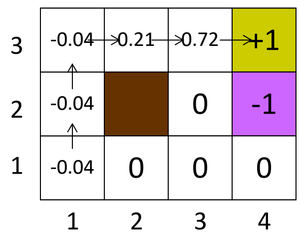
Temporal Difference Learning

3	-0.02	-0.02	0.48	+1
2	-0.02		0	-1
1	-0.02	0	0	0
	1	2	3	4

With learning rate
α=0.5

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$

Temporal Difference Learning

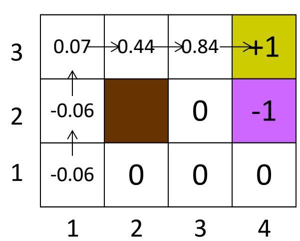


With learning rate
α=0.5

After a second trajectory
from start to +1

1. Store counts N[s] and estimated values $V^\pi(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^\pi(s) \leftarrow V^\pi(s)+\alpha(N[s])(r+\gamma V^\pi(s')-V^\pi(s))$

Temporal Difference Learning

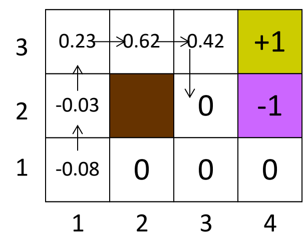


With learning rate
α=0.5

After a third trajectory
from start to +1

1. Store counts N[s] and estimated values $V^\pi(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^\pi(s) \leftarrow V^\pi(s)+\alpha(N[s])(r+\gamma V^\pi(s')-V^\pi(s))$

Temporal Difference Learning

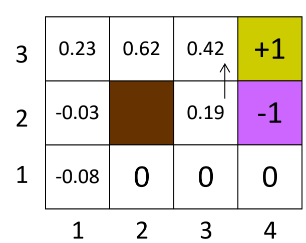


With learning rate
α=0.5

Our luck starts to run
out on the fourth trajectory

1. Store counts N[s] and estimated values $V^\pi(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^\pi(s) \leftarrow V^\pi(s)+\alpha(N[s])(r+\gamma V^\pi(s')-V^\pi(s))$

Temporal Difference Learning



With learning rate
α=0.5

But we recover…

1. Store counts N[s] and estimated values $V^\pi(s)$
2. For each observed transition (s,r,a,s'):
   3. Set N[s] $\leftarrow$ N[s]+1
   4. Adjust value $V^\pi(s) \leftarrow V^\pi(s)+\alpha(N[s])(r+\gamma V^\pi(s')-V^\pi(s))$

Temporal Difference Learning

3	0.23	0.62	0.69	+1
2	-0.03		0.19	-1
1	-0.08	0	0	0
	1	2	3	4

With learning rate
α=0.5

...and reach the goal!

1. Store counts N[s] and estimated values $V^{\pi}(s)$
2. For each observed transition (s,r,a,s'):
    3. Set N[s] $\leftarrow$ N[s]+1
    4. Adjust value $V^{\pi}(s) \leftarrow V^{\pi}(s)+\alpha(N[s])(r+\gamma V^{\pi}(s')-V^{\pi}(s))$

- Learns a little with each step taken (no need to wait for complete trajectories)
- Uses current estimate of next state value, rather than direct estimate of entire trajectory value

Using TD for Control

- Recall value iteration:

$$V^{i+1}(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s' | s,a) V^i(s')$$

- Why not pick the maximizing **a** and then do:

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

  – s' is the observed next state after taking action **a**

What breaks?

- Action selection
  - How do we pick a?
  - Need to P(s'|s,a), but the reason why we're doing RL is that we don't know this!

- Even if we magically knew the best action:
  - Can only learn the value of the policy we are following
  - If initial guess for V suggests a stupid policy, we'll never learn otherwise

Q-Values

- Learning V is not enough for action selection because a transition model is needed
- Solution: learn Q-values: Q(s,a) is the utility of choosing action a in state s
- "Shift" or "split" Bellman equation
  - $V(s) = \max_a Q(s,a)$
  - $Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$

- So far, everything is the same… but what about the learning rule?

Q-Learning

- Recall TD:
  - Update: $V(s) \leftarrow V(s)+\alpha(N[s])(r+\gamma V(s')-V(s))$
  - Use P to pick actions? $a \leftarrow \arg\max_a \Sigma_{s'} P(s'|s,a)V(s')$
- Q-Learning:
  - Update: $Q(s,a) \leftarrow Q(s,a)+\alpha(N[s,a])(r+\gamma \max_{a'} Q(s',a')-Q(s,a))$
  - Select action: $a \leftarrow \arg\max_a Q(s,a)$
- – Key difference: average over $P(s'|s,a)$ is "baked in" to the Q function
- – Q-learning is therefore a **model-free** learner

Q-Learning Demo
(see demo)

- Keyboard controlled Q-learning agent using robot grid world from R&N (and previous MDP slides)
- Differences:
  - Discount of 0.5
  - Noise =0 for manual
  - No step cost (previously -0.04)
  - Learning rate 0.5

Q-learning vs. TD-learning

- TD converges to value of policy you are following
- Q-learning converges to values of optimal policy independent of of whatever policy you follow during learning!
- Caveats:
  - Converges in limit, assuming all states are visited infinitely often
  - In case of Q-learning, all states and actions must be tried infinitely often

Note: If there is only one action possible in each state, then Q-learning and TD-learning are identical

Exploration vs. Exploitation

- Greedy strategy purely **exploits** current knowledge
  - The quality of this knowledge improves only for those states that the agent observes often
- A good learner must perform **exploration** to improve knowledge about states not often observed
  - But pure exploration is useless (and costly) if it is never exploited

Restaurant Problem

Exploration vs. Exploitation Theory and Practice

- Can assign an “exploration bonus” to states (or state-action combinations) you haven’t experienced much
  - Versions of this are provably efficient, e.g., R-Max (will eventually learn the optimal policy requiring polynomial effort in size of problem)
  - Works for small state spaces – will have more to say about this
- In practice ε-greedy action selection is used most often
  - Choose greedy action w.p. $1-\varepsilon$
  - Choose random action w.p. ε

Value Function Representation

- Fundamental problem remains unsolved:
  - TD/Q learning solves model-learning problem, but
  - Large models still have large value functions
  - Too expensive to store these functions
  - Impossible to visit every state in large models
- Function approximation
  - Combine fitted value/Q-iteration with RL
  - Use machine learning methods to generalize
  - Avoid the need to visit every state

Updates with Approximation

- Recall regular TD update:

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

- With function approximation:

$$V(s) \approx V(s;w)$$

Vector operations

- Update the weights:

$$w^{i+1} = w^{i} + \alpha(r + \gamma V(s';w) - V(s;w))\nabla_w V(s;w)$$

Neural networks are a special case of this.

For linear value functions

- Gradient is trivial:

$$V(s;w) = \sum_{j=1}^{k} w_j \varphi_j(s)$$

$$\nabla_{w_j} V(s;w) = \varphi_j(s)$$

- Update is trivial:

Individual components

$$w_j^{i+1} = w_j^{i} + \alpha(r + \gamma V(s';w) - V(s;w))\varphi_j(s)$$

Neural Networks

- s = input into neural network
- w = weights of neural network
- g(s, θ) = output of network
- Try to minimize

$$E = \sum_s \left(f(s) - g(s,\theta)\right)^2$$

- Compute gradient of error wrt weights

$$\frac{\partial E}{\partial \theta}$$

- Adjust to minimize error

Combining NNs with TD

- Recall TD:

$$V^{temp}(s) = R(s) + \gamma V^i(s')$$

$$V^{i+1}(s) = (1-\alpha)V^i(s) + \alpha V^{temp}(s)$$

- Compute error function:

$$E=(V^i(s,\theta)-V^{temp}(s,\theta))^2$$

- Update:

$$\theta^{i+1} = \theta^i - \alpha \frac{\partial E}{\partial \theta}$$

$$= \theta^i + 2\alpha\left[V^{temp}(s,\theta) - V(s,\theta)\right]\frac{\partial V(s,\theta)}{\partial \theta}$$

Is this right???

Gradient-based Updates

$$\theta^{i+1} = \theta^i - \alpha \frac{\partial E}{\partial \theta}$$

$$= \theta^i + 2\alpha\left[V^{temp}(s,\theta) - \hat{V}(s,\theta)\right]\frac{\partial V(s,\theta)}{\partial \theta}$$

- Equivalent to one step of backprop with V^{temp} as target
- Constant factor absorbed into learning rate
- Table-updates are a special case
- Perceptron, linear regression are special cases

Properties of approximate RL

- Exact case (tabular representation) = special case
- Can be combined with Q-learning

- Convergence not guaranteed
  - Policy evaluation with linear function approximation converges if samples are drawn "on policy"
  - In general, convergence is not guaranteed
    - Chasing a moving target
    - Errors can compound
- Success has traditionally required very carefully chosen features
- Deep RL (next slide set) changes the paradigm

Conclusions

- Reinforcement learning solves an unknown MDP

- Converges for exact value function representation

- Can be combined with approximation methods

- Good results require good features (traditionally) or lots of data and deep learning