

EE496 : COMPUTATIONAL INTELLIGENCE

RL03: REINFORCEMENT TD LEARNING

UGUR HALICI

METU: Department of Electrical and Electronics Engineering (EEE)

METU-Hacettepe U: Neuroscience and Neurotechnology (NSNT)

So far

Given an MDP model we know how to find optimal policies

- Value Iteration or Policy Iteration

- But what if we don't have any form of the model of the world (e.g., T , and R)

- Like when we were babies . . .

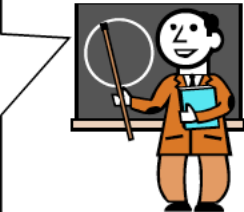
- All we can do is wander around the world observing what happens, getting rewarded and punished

- This is what reinforcement learning about

Why not supervised learning

- In supervised learning, we had a teacher providing us with training examples with class labels

Has Fever	Has Cough	Has Breathing Problems	Ate Chicken Recently	Has Asian Bird Flu
true	true	true	false	false
true	true	true	true	true
false	false	false	true	false



- The agent figures out how to predict the class label
- given the features.

Reinforcement/Reward

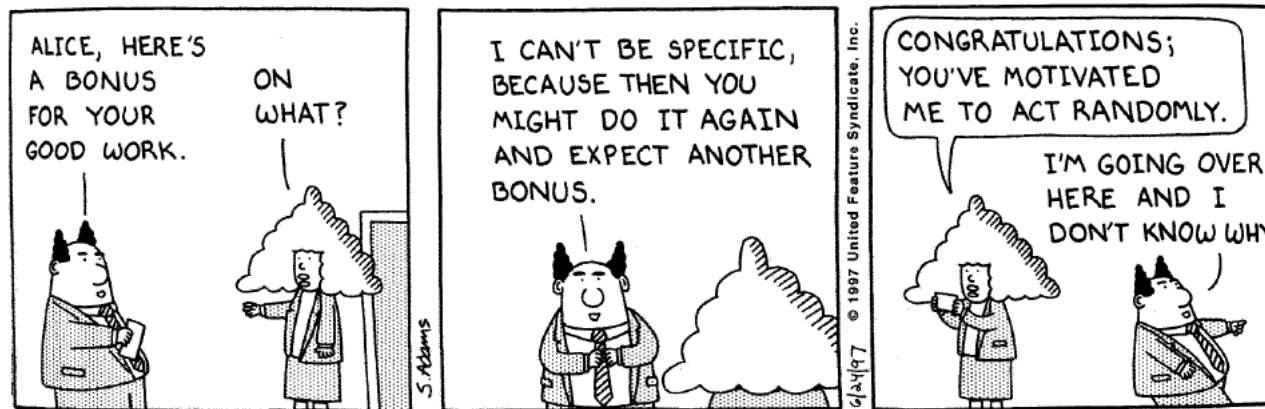
- The key to this trial-and-error approach is having some sort of feedback about what is good and what is bad
- We call this feedback **reward** or **reinforcement**
- In some environment, rewards are frequent
 - Ping-pong: each point scored
 - Learning to crawl: forward motion
- In other environments, reward is delayed
 - Chess: reward only happens at the end of the game

Can We Use Supervised Learning?

- Now imagine a complex task such as learning to play a board game
- Suppose we took a supervised learning approach to learning an evaluation function
- For every possible position of your pieces, you need a teacher to provide an accurate and consistent evaluation of that position
 - This is not feasible

Importance of Credit Assignment

-



Trial and Error

- A better approach: imagine we don't have a teacher
- Instead, the agent gets to experiment in its environment
- The agent tries out actions and discovers by itself which actions lead to a win or loss
- The agent can learn an evaluation function that can estimate the probability of winning from any given position

Reinforcement

- This is very similar to what happens in nature with animals and humans
 - Positive reinforcement:
Happiness, Pleasure, Food
 - Negative reinforcement:
Pain, Hunger, Lonelinesss
-
- What happens if we get agents to learn in this way?
 - This leads us to the world of Reinforcement Learning

Reinforcement Learning

- Agent placed in an environment and must learn to behave optimally in it
- Assume that the world behaves like an MDP, except:
 - Agent can act but does not know the transition model
 - Agent observes its current state its reward but doesn't know the reward function
- Goal: learn an optimal policy

Factors that Make RL Difficult

- Actions have non- deterministic effects
 - which are initially unknown and must be learned
- Rewards / punishments can be infrequent
 - Often at the end of long sequences of actions
 - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment problem)
 - World is large and complex

Passive vs. Active learning

Passive learning

- The agent acts based on a fixed policy π and tries to learn how good the policy is by observing the world go by
- Analogous to policy evaluation in policy iteration
- Active learning
- The agent attempts to find an optimal (or at least good) policy by exploring different actions in the world
- Analogous to solving the underlying MDP

Model-Based vs. Model-Free RL

Model based approach to RL:

- learn the MDP model (T and R), or an approximation of it
- use it to find the optimal policy

Model free approach to RL:

- derive the optimal policy without explicitly learning the model

We will consider both types of approaches

Passive Reinforcement Learning

- Suppose agent's policy π is fixed
- It wants to learn how good that policy is in the world ie. it wants to learn $U^\pi(s)$
- This is just like the policy evaluation part of policy iteration
- The big difference: the agent doesn't know the transition model or the reward function (but it gets to observe the reward in each state it is in)

Passive RL

- Suppose we are given a policy
- Want to determine how good it is

Given π :

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

Need to learn $U_\pi(S)$:

3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Passive RL

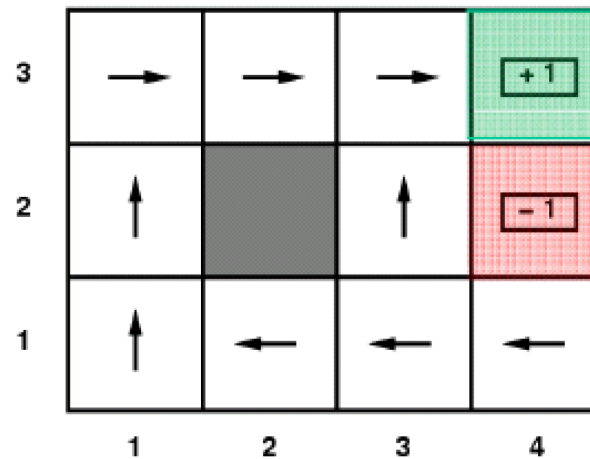
Given policy π

- estimate $U^\pi(s)$

Not given :

- transition matrix, nor
- reward function !
- Simply follow the policy for many epochs
- Epochs: training sequences:

$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,4) \text{ } \underline{+1}$
 $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (3,4) \text{ } \underline{+1}$
 $(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2) \text{ } \underline{-1}$



Appr. 1: Direct Utility Estimation

Direct utility estimation (model free)

- Estimate $U^\pi(s)$ as **average total reward of epochs** containing s
(calculating from s to end of epoch)

Reward to go of a state s

- the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed **reward to go of** the state as the direct evidence of the actual expected utility of that state

Direct Utility Estimation

Suppose we observe the following trial:

$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04}$
 $\rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$

The total reward starting at (1,1) is 0.72. We call this a sample of the observed-reward-to-go for (1,1).

For (1,2) there are two samples for the observed-reward-to-go (assuming $\gamma=1$):

1. $(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04}$
 $\rightarrow (4,3)_{+1}$ [Total: 0.76]
2. $(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$ [Total: 0.84]

Direct Utility Estimation

- Direct Utility Estimation keeps a running average of the observed reward-to-go for each state
- Eg. For state (1,2), it stores $(0.76+0.84)/2 = 0.8$
- As the number of trials goes to infinity, the sample average converges to the true utility

The big problem with Direct Utility Estimation: it converges very slowly!

- Doesn't exploit the fact that utilities of states are not independent
- Utilities follow the Bellman equation

$$U_{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_{\pi}(s')$$

Note the dependence on neighboring states

Direct Utility Estimation

- Using the dependence to your advantage

3				+1
2				-1
1	START			
	1	2	3	4

Remember that
each blank state
has $R(s) = -0.04$

- Suppose you know that state (3,3) has a high utility
- Suppose you are now at (3,2)
- The Bellman equation would be able to tell you that (3,2) is likely to have a high utility because (3,3) is a neighbor.
- DEU can't tell you that until the end of the trial

Adaptive Dynamic Programming

- This method does take advantage of the constraints in the Bellman equation
- Basically learns the transition model T and the reward function R
- Based on the underlying MDP (T and R) we can perform policy evaluation (which is part of policy iteration previously taught)
- Recall that **policy evaluation** in policy iteration involves solving the utility for each state if policy π_i is followed.
- This leads to the equations:

$$U_{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_{\pi}(s')$$

- The equations above are linear, so they can be solved with linear algebra in time $O(n^3)$ where n is the number of states

Adaptive Dynamic Programming

- Make use of policy evaluation to learn the utilities of states
- In order to use the policy evaluation eqn:

$$U_{\pi}(s) = \underline{R(s)} + \gamma \sum_{s'} \underline{T(s, \pi(s), s')} U_{\pi}(s')$$

- the agent needs to learn the transition s' model $T(s, a, s')$ and the reward function $R(s)$
- How do we learn these models?

Adaptive Dynamic Programming

Learning the reward function $R(s)$:

Easy because it's deterministic. Whenever you see a new state, store the observed reward value as $R(s)$

Learning the transition model $T(s,a,s')$:

Keep track of how often you get to state s' given that you're in state s and do action a .

eg. if you are in $s = (1,3)$ and you execute Right three times and you end up in $s'=(2,3)$ twice, then $T(s,\text{Right},s') = 2/3$

ADP Algorithm

function PASSIVE-ADP-AGENT(percept) returns an action

inputs: percept, a percept indicating the current state s' and reward signal r'

static:

π , a fixed policy

mdp, an MDP with model T , rewards R , discount γ

U , a table of utilities, initially empty

N_{sa} , a table of frequencies for state-action pairs, initially zero

$N_{sas'}$, a table of frequencies for state-action-state triples, initially zero

s , a the previous state and action, initially null

if s' is new **then do** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$ /* Update reward function */

if s is not null, **then do** /* Update transition model */

increment $N_{sa}[s,a]$ and $N_{sas'}[s,a,s']$

for each such that $N_{sas'}[s,a,t]$ is nonzero do

$T[s,a,t] \leftarrow N_{sas'}[s,a,t] / N_{sa}[s,a]$

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \text{mdp})$

if **TERMINAL?** $[s']$ **then** $s, a \leftarrow \text{null}$ **else** $s, a \leftarrow \pi[s']$

return a

Temporal Difference Learning

- Instead of calculating the exact utility for a state can we approximate it and possibly make it less computationally expensive?
- Yes we can! Using Temporal Difference (TD) learning

$$U_{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_{\pi}(s')$$

Instead of doing this sum over all successors, only adjust the utility of the state based on the successor observed in the trial.

It does not estimate the transition model – model free

TD Learning

Example:

- Suppose you see that $U^\pi(1,3) = 0.84$ and $U^\pi(2,3) = 0.92$ after the first trial.
- If the transition $(1,3) \rightarrow (2,3)$ happens all the time, you would expect to see:

$$U^\pi(1,3) = R(1,3) + U^\pi(2,3)$$

$$\Rightarrow U^\pi(1,3) = -0.04 + U^\pi(2,3)$$

$$\Rightarrow U^\pi(1,3) = -0.04 + 0.92 = 0.88$$

- Since you observe $U^\pi(1,3) = 0.84$ in the first trial, it is a little lower than 0.88, so you might want to “bump” it towards 0.88.

Aside: online mean estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers
 - E.g. to estimate the mean of a r.v. from a sequence of samples.

$$\begin{aligned}\hat{X}_{n+1} &= \frac{1}{n+1} \sum_{i=1}^{n+1} x_i = \frac{1}{n} \sum_{i=1}^n x_i + \frac{1}{n+1} \left(x_{n+1} - \frac{1}{n} \sum_{i=1}^n x_i \right) \\ &= \hat{X}_n + \frac{1}{n+1} (x_{n+1} - \hat{X}_n)\end{aligned}$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample $x(n+1)$, the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and old estimate

Temporal Difference Learning (TD)

- TD update for transition from s to s' :

$$U_{\pi}(s) = U_{\pi}(s) + \alpha(R(s) + \gamma U_{\pi}(s') - U_{\pi}(s))$$

learning rate

New (noisy) sample of utility
based on next state

- So the update is maintaining a “mean” of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g. $1/n$) then the utility estimates will eventually converge to true values!

$$U_{\pi}(s) = R(s) + \gamma \sum_{s'} T(s, a, s') U_{\pi}(s')$$

Temporal Difference Update

- When we move from state s to s' , we apply the following update rule

$$U_{\pi}(s) = U_{\pi}(s) + \alpha(R(s) + \gamma U_{\pi}(s') - U_{\pi}(s))$$

- This is similar to one step of value iteration
- We call this equation a “**backup**”

Convergence

- Since we're using the observed successor s' instead of all the successors, what happens if the transition $s \rightarrow s'$ is very rare and there is a big jump in utilities from s to s' ?
- How can $U\pi(s)$ converge to the true equilibrium value?
Answer: The average value of $U\pi(s)$ will converge to the correct value
- This means we need to observe enough trials that have transitions from s to its successors
- Essentially, the effects of the TD backups will be averaged over a large number of transitions
- Rare transitions will be rare in the set of transitions observed

Comparison between ADP and TD

- Advantages of ADP:
 - Converges to the true utilities faster
 - Utility estimates don't vary as much from the true utilities
- Advantages of TD:
 - Simpler, less computation per observation
 - Crude but efficient first approximation to ADP
 - Don't need to build a transition model in order to perform its updates (this is important because we can interleave computation with exploration rather than having to wait for the whole model to be built first)

Overall comparison

- - **Direct Estimation (model free)**
 - Simple to implement
 - Each update is fast
 - Does not exploit Bellman constraints and converges slowly
 - **Adaptive Dynamic Programming (model based)**
 - Harder to implement
 - Each update is a full policy evaluation (expensive)
 - Fully exploits Bellman constraints
 - Fast convergence (in terms of epochs)
 - **Temporal Difference Learning (model free)**
 - Update speed and implementation similar to direct estimation
 - Partially exploits Bellman constraints---adjusts state to 'agree' with observed successor
 - Not *all* possible successors
 - Convergence in between direct estimation and ADP