University of Colorado **Boulder**

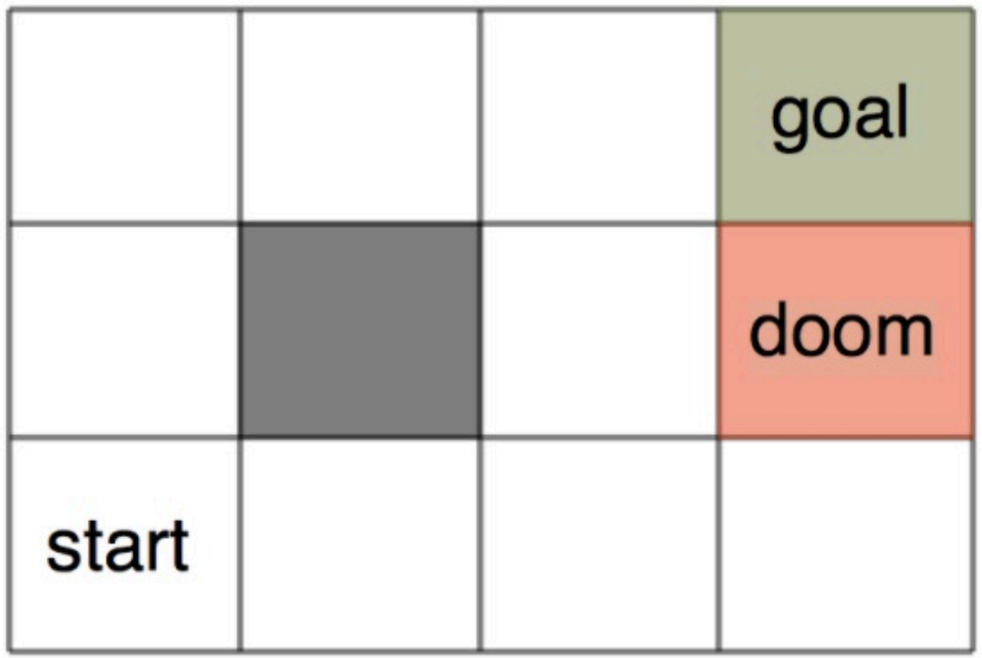
Department of Computer Science

CSCI 5622: Machine Learning

Chris Ketelsen

Lecture 21:
Reinforcement Learning Part 2

The Grid World with Randomness



Available Actions: Up, Down, Left, Right

Random Rules: $P(\text{action}) = 0.8$ and $P(\text{right angle}) = 0.1$

Markov Decision Processes

States: S

Actions: A, or $A(s)$

Transition Model: $T(s, a, s') = P(s' \mid s, a)$

Rewards: $R(s)$, $R(s, a)$, $R(s, a, s')$

Policy: $\pi(s) \rightarrow a$

Goal: Find **optimal** policy $\pi^\star$ that maximizes longterm reward

Sequences of Rewards

Cumulative value based on **discounted** reward

$$V(s_0, s_1, s_2, \ldots) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \quad \text{for some } 0 \leq \gamma < 1$$

The parameter γ is the called the **discount factor**

The discount factor controls how long we're willing to wait for delayed rewards

Optimal Policies

The optimal policy $\pi^\star$ is the policy that maximizes the expected long-term discounted reward

$$\pi^\star = \text{argmax}_\pi \, \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, \pi \right]$$

Define the **cumulative value** of being in state s and following policy π in terms of the expected reward from starting in that state

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \,\middle|\, \pi, s_0 = s \right]$$

Optimal Policies

Can define *implicit* definition of $\pi^\star$ in terms of cumulative value

$$\pi^\star(s) = \text{argmax}_a \sum_{s'} P(s' \mid s, a)\, V^{\pi^\star}(s')$$

Bellman's Equation:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, V(s')$$

This allows us to solve the entire MDP

If we can find the optimal policy that maximizes the cumulative value at all states, then we're done

Actually Finding Optimal Policies

$$V(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, V(s')$$

Value Iteration:

1. Start with arbitrary values for $V(s)$
2. Update V's based on neighbors
3. Repeat until values don't change anymore

Define the *approximation* of the value at iteration k as $\hat{V}_k(s)$

Update Rule: $\hat{V}_{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a)\, \hat{V}_k(s')$

Value Iteration

Once we've converged to $V^{\pi^\star}$, get optimal policy via

$$\pi^\star(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V^{\pi^\star}(s')$$

- Guaranteed to converge to the optimal $V^{\pi^\star}$
- which gives us the optimal $\pi^\star$
- $\hat{V}_k$ might converge very slowly to $V^{\pi^\star}$
- but might converge quickly to $\pi^\star$

So far we haven't had to **learn** anything yet

Value Iteration

Once we've converged to $V^{\pi^\star}$, get optimal policy via

$$\pi^\star(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) V^{\pi^\star}(s')$$

- Guaranteed to converge to the optimal $V^{\pi^\star}$
- which gives us the optimal $\pi^\star$
- $\hat{V}_k$ might converge very slowly to $V^{\pi^\star}$
- but might converge quickly to $\pi^\star$

Next Time: What happens if we don't know model parameters

Next Time: Tweak Bellman Equations to get to Q-Learning

Q-Learning

To see how this works, we're going to take one step back so we can take two steps forward

Assume for a second that we have a deterministic MDP

That is, assume for a particular (s, a) pair, the resulting s' is defined

Now, define the evaluation function $Q(s, a)$ as follows

$$Q(s, a) = r(s) + \gamma V^{\pi^\star}(s')$$

Note then that our old cumulative value function is exactly

$$V^{\pi^\star}(s) = \arg\max_a Q(s, a)$$

Q-Learning

So now the function $Q(s, a)$ is defined as the cumulative value obtained by being in state s and taking action a

Which means we can define the optimal policy by

$$\pi^{\star}(s) = \arg\max_{a} Q(s, a)$$

The nice thing about this is that if we can learn the $Q(s, a)$ function then we don't even need to know the reward or transition models

And, since the definition of $Q(s, a)$ is relatedly so closely to V, much of the derivation of the learning process is analogous to what we've done before.

Q-Learning

The two are so related, in fact, that the Q function satisfies a form of Bellman's Equation

In the deterministic setting, we have

$$V(s) = R(s) + \gamma\, V(s')$$

The analogous statement with the Q-function is

$$Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')$$

This says that the cumulative value for being in state s and taking action a is the instantaneous reward plus the discounted cumulative value of being in the resulting state and taking the best next action

Q-Learning

OK, we once again have a recursive definition for the long term cumulative value, but as a function of state and action taken

We can find the optimal value function using an iterative method

The form of the iteration will look like

$$\hat{Q}_{k+1}(s,a) = R(s) + \gamma \max_{a'} \hat{Q}_k(s',a')$$

The difference this time is that we'll iterate by letting an agent explore the world, and update a single (s,a) table entry at a time

The Q-Learning Algorithm

For each s, a initialize table entry $\hat{Q}(s, a)$ to zero

Do forever:

- Select an action a and execute it
- Receive immediate reward R
- Observe new state s'
- Update table entry as

$$\hat{Q}(s, a) \leftarrow R(s) + \gamma \max_{a'} \hat{Q}(s', a')$$

- Set $s \leftarrow s'$

The Q-Learning Algorithm

Question: How do we navigate the board?

We could do is use the current value of $\hat{Q}$ to determine actions

When do we stop?

Essentially you run the game until the terminal state is reached

Then start new *episode* from a randomish-ly chosen state s

What are the Rewards?

Common thing to do is to choose $R(s) = 0$ for nonterminal states

This Seems Hokey. Will it Converge?

The Q-Learning Algorithm

It turns out that it will converge under some mildly ridiculous assumptions

Convergence Theorem: Assume bounded rewards $(\forall s, |R(s)| \leq R_{max})$. If $\hat{Q}(s,a)$ is initialized to random finite values and uses $0 \leq \gamma < 1$. If each state-action pair is visited infinitely often, then $Q_k(s,a)$ converges to $Q(s,a)$ as $k \to \infty$ for all (s,a).

Proof Sketch: Let Δ_k be the maximum error in $\hat{Q}_k$, i.e.

$$\Delta_k = \max_{s,a} \left| \hat{Q}_k(s,a) - Q(s,a) \right|$$

Now, take one step in the algorithm

The Q-Learning Algorithm

What is the error in $\hat{Q}_{k+1}(s,a)$?

$$\left|\hat{Q}_{k+1}(s,a) - Q(s,a)\right| =$$

$$\left|(R + \gamma \max_{a'} \hat{Q}_k(s',a') - (R + \gamma \max_{a'} Q(s',a'))\right| =$$

$$\left|\gamma \max_{a'} \hat{Q}_k(s',a') - \gamma \max_{a'} Q(s',a')\right| \leq$$

$$\gamma \max_{a'} \left|\hat{Q}_k(s',a') - Q(s',a')\right| \leq$$

$$\gamma \max_{s'',a'} \left|\hat{Q}_k(s'',a') - Q(s'',a')\right|$$

Thus $\left|\hat{Q}_{k+1}(s,a) - Q(s,a)\right| \leq \gamma \Delta_k$

The Q-Learning Algorithm

Thus $\left|\hat{Q}_{k+1}(s,a) - Q(s,a)\right| \leq \gamma \Delta_k$

This says that the error in $\hat{Q}_{k+1}(s,a)$ is no worse than γ times the worst error in the table

Now, we know that Δ_0 is bounded, because we have finite rewards and our initial table values were finite

So after (s,a) is visited k times, the error will be at most $\gamma^k \Delta_0$, which converges to 0 as $k \to \infty$ because $\gamma < 1$

What's the catch? The part about visiting each (s,a) pair infinitely many times

The Q-Learning Algorithm

In practice, we have to ensure that our agent can sufficiently explore the world

Exploration vs. Exploitation: How do we best explore the world?

As we update Q we're also updating π

Following π might make us miss interesting states

Better to follow π but with some randomness thrown in

Choose actions according to

$$P(a_i \mid s) = \frac{k^{\hat{Q}(s,a_i)}}{\sum_j k^{\hat{Q}(s,a_j)}}$$

The Q-Learning Algorithm

The amazing thing about this convergence theorem is that we don't need to follow optimal action sequences in order to converge

We can learn the Q function while training on completely randomly chosen actions if we want (though maybe not a good idea)

Idea: Prioritized Sweeping: After each time $\hat{Q}(s, a)$ is updated keep track of how much it changed

Add s to priority queue based on size of change

Start new episode in s at top of priority-queue

Intuition: If $\hat{Q}$ changed a lot at s previously, maybe it still has a lot more change to go. So let's start there.

Non-Deterministic Actions

In the nondeterministic case we need to model the stochastic nature between attempted actions from the current state and the actual next state achieved

In many applications we model the instantaneous reward as stochastic as well

The development is very similar to what we did in the previous lecture. Assume the cumulative value is an expectation of the longterm discounted reward

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

Non-Deterministic Actions

We then define the Q-function by analogy

$$\begin{aligned} Q(s,a) &= \mathbb{E}\left[r(s) + \gamma V^{\pi^\star}(\delta(s,a))\right] \\ &= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}\left[V^{\pi^\star}(\delta(s,a))\right] \\ &= \mathbb{E}\left[r(s)\right] + \gamma \sum_{s'} P(s' \mid s,a) V^{\pi^\star}(s') \end{aligned}$$

And now we use the fact that $V(s) = \max_{a'} Q(s,a)$ to get

$$Q(s,a) = \mathbb{E}\left[r(s)\right] + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q(s',a')$$

So it turns out that simple Value-Iteration will not converge here

Question: What if we iterate with the true Q as the starting guess?

Non-Deterministic Actions

It's best to think of this process with a real world example.

Suppose that you have a robot that is trying to learn to do some task. You send the actual robot out into the world to learn.

In a mechanical setting, if the robot chooses a particular action from a state, then occasionally it will not end up in the deterministic state (either because of sensor malfunctions or weirdness in the environment)

In other words, there is some transition probability, but we do not know what it is

Should we attempt to estimate the parameters in the transition model?

Temporal Differencing

It turns out that we don't need to.

Instead we iterate using the observed states, actions, and rewards

The stochasticity is handled by taking a weighted avg. of the current value of the Q-function and the value predicted by the new update

Update: $\hat{Q}_{k+1}(s,a) = r + \gamma \max_{a'} \hat{Q}_k(s',a')$

Old Value: $\hat{Q}_k(s,a)$

Weighted Avg: $(1-\alpha_k)\hat{Q}_k(s,a) + \alpha_k(r + \gamma \max_{a'} \hat{Q}_k(s',a'))$

$$\hat{Q}_{k+1}(s,a) = \hat{Q}_k(s,a) + \alpha_k \left[r + \gamma \max_{a'} \hat{Q}_k(s',a') - \hat{Q}_k(s,a) \right]$$

Temporal Differencing

$$\hat{Q}_{k+1}(s,a) = \hat{Q}_k(s,a) + \alpha_k \left[r + \gamma \max_{a'} \hat{Q}_k(s',a') - \hat{Q}_k(s,a) \right]$$

The trick is to decrease the learning rate as you go

Intuition:

- At the beginning, stochastic update is useful for exploring
- At the end you've probably converged to good Q, so dampen the effect of noisy samples

Temporal Differencing

The Model-Free Method:

- Slower than Model-Based method, but much simpler
- How do we choose learning rate $0 < \alpha < 1$?
- Too small: TD conv. slowly, biased towards current $\hat{Q}(s, a)$
- Too large: $\hat{Q}(s, a)$ biased towards current episode

Convergence Guaranteed So Long As:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Reinforcement Learning

OK, so what's the problem with this?

Suppose you're trying to train a computer to play Backgammon.

How big is your table representing Q?

Reinforcement Learning

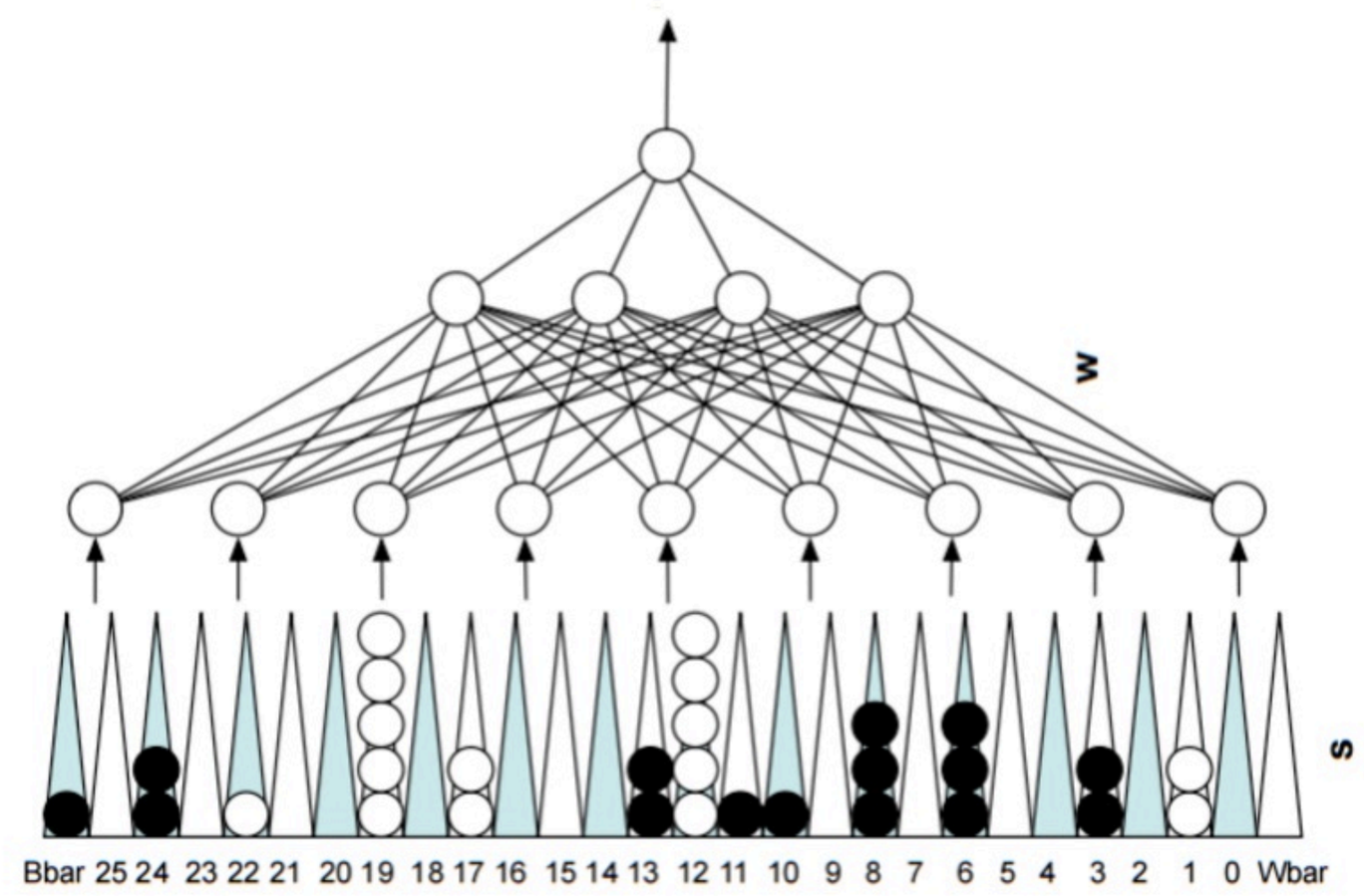
26 board positions, 30 checkers, dice combos...

$$34 \times 10^{21} \quad (s, a) \text{ pairs}$$

That's a lot...

Reinforcement Learning

What if instead of learning $Q(s, a)$ for every pair, we instead represent Q that takes in states and actions.



Reinforcement Learning

Still doing TD Learning

$$Loss = \frac{1}{2}\left[R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]^2$$

Given a transition (s, a, R, s'), do the following

1. Feedforward for current state s to predict $Q(s, a)$
2. Feedforward for next state s' for to predict each $Q(s', a')$
3. Update weights via backpropagation

Towards Model-Free Learning

How well does it work?

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	other programs	tied for best
TD-Gam 1.0	80	300,000	Robertie, Magriel, ...	-13 pts / 51 games
TD-Gam 2.0	40	800,000	various Grandmasters	-7 pts / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	-1 pt / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

Tesauro, Gerald. *Temporal Difference Learning and TD-Gammon*. *Comm. of ACM.* Vol 38, Issue 3, (1995)