Reinforcement Learning

15-486/782: Artificial Neural Networks
David S. Touretzky

Fall 2006

Based on slides by
Sebastian Thrun and Nathaniel Daw

Learning Styles

- Supervised learning
  - Teacher provides input and desired output
  - Network learns to imitate the teacher.
  - Should generalize to new cases.
- Unsupervised learning
  - "Environment" provides data points.
  - Network discovers structure in the data.
  - No right or wrong answers.
- Reinforcement learning
  - Reward signal provides feedback, but doesn't give a "right answer."
  - Network learns to optimize reward.

How Do We Maximize Reward?

Find a policy Π that tells us the best thing to do.

$$\Pi: S \rightarrow A$$

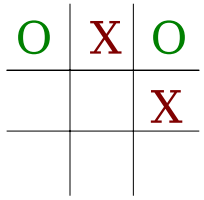
$s \in S$ is current state; $a \in A$ is action to take.

How we represent S is crucial; there could be exponentially many states.

Maximal reward achieved when Π is optimal.

Example: Tic-Tac-Toe

- Given: board position s
- Wanted: best move a (a number in 1..9)
- Find $\Pi: S \rightarrow A$
- How? Trial and error?



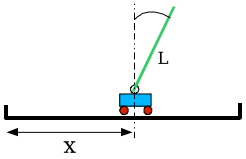
Example: Inverted Pendulum (Pole Balancing)

State: $s = \langle x(t),\ \dot{x}(t),\ \theta(t),\ \dot{\theta}(t) \rangle$

Dynamics:

$$\ddot{\theta} = = \frac{-g\sin\theta + \cos\theta\left[\frac{-F - m_p\dot{\theta}^2\sin\theta - \mu_c sgn(\dot{x})}{m_c + m_p}\right] - \frac{\mu + p\dot{\theta}}{m_p L}}{L\left[\frac{4}{3} - \frac{m_p\cos^2\theta}{m_c + m_p}\right]}$$

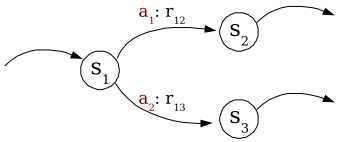
$$\ddot{x} = \frac{F + m_p L\left[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta\right] - \mu_c sgn(\dot{x})}{m_c + m_p}$$



with m_c=1kg, m_p=0.1kg, L=0.5m,
μ_c=0.0005, μ_p=0.000002, g=−9.81 m/s^2

Wanted: force F that balances the pole. Find $\Pi : S \rightarrow F$

Markov Decision Problem



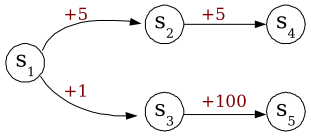
S = states, A = actions, r = reward

δ: $S \times A \rightarrow S$ transition function
r: $S \times A \rightarrow \mathbb{R}$ reward function

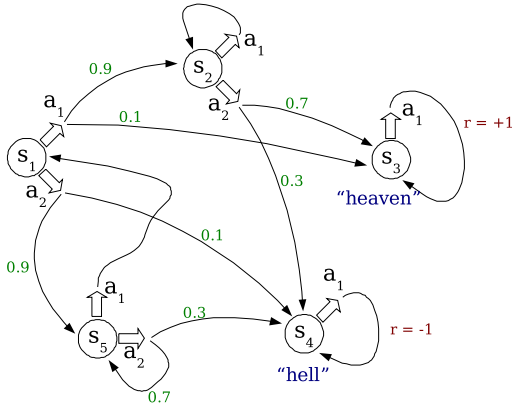
Find policy $\Pi : S \rightarrow A$ that maximizes total reward.

Why Is This Hard?

- Functions δ(s,a) and/or r(s,a) may be unknown.
- System may be stochastic, so δ(s,a) and/or r(s,a) may be probability distributions.
- Delayed reward:



Markov Decision Process (MDP) With Stochastic Actions



How Do We Measure Total Reward?

- Simple summation: diverges!

$$V = \sum_{t=0}^{\infty} r(t)$$

- Exponential discounting: most popular method.

$$V = \sum_{t=0}^{\infty} \gamma^t r(t) \quad \text{with } 0<\gamma<1$$

- Finite horizon h

$$V = \sum_{i=0}^{h} r(t+i)$$

- Average reward ("rate learning")

$$V = \lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n} r(t)$$

Exponential Discounting

Let $V^{\Pi}(s)$ be the cumulative rewward achievable from state s under policy Π.

$$\begin{aligned} V^{\Pi}(s) &= r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots \\ &= \sum_{t=0}^{\infty} \gamma^t r_t \\ &= r_0 + \gamma V^{\Pi}(\delta(s, \Pi(s))) \end{aligned}$$

If we know V, then the best policy is:

$$\Pi^*(s) = argmax_a \left[r(s,a) + \gamma V(\delta(s,a)) \right]$$

Q-Learning (Watkins, 1989)

Iterative algorithm for approximating V^*.

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

$$\Pi^*(s) = argmax_a Q(s,a)$$

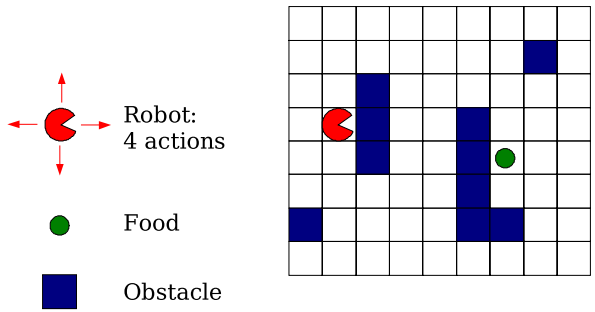
Since V^* is a function of Π^*, we have:

$$Q(s,a) = r(s,a) + \gamma \max_b Q(\delta(s,a), b)$$

Let $\hat{Q}$ be our approximation of Q.
Update rule for $\hat{Q}$ (deterministic actions and rewards):

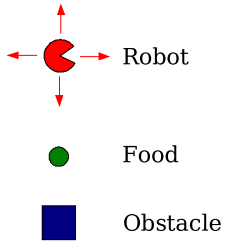
$$\hat{Q}(s,a) \leftarrow r(s,a) + \gamma \max_b \hat{Q}(\delta(s,a), b)$$

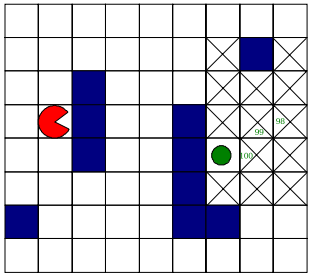
8×9 Grid World



Q table is 72×4

8 × 9 Grid World





Q table is 72×4

Explore/Exploit Tradeoff

Policy $\Pi = argmax_a \hat{Q}(s,a)$ is optimal ***after*** we've learned Q.

If $\hat{Q}$ is not yet a good approximation, we may miss the best action.

Solution 1: act randomly to learn $\hat{Q}$.

Solution 2: choose actions probabilistically.

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

Small k: explore. Large k: exploit.

Convergence of Q-Learning

Theorem: Q-Learning converges to V^*, provided that...

1) The system is a deterministic MDP

2) Immediate rewards are bounded:

$|r(s,a)| < c$

3) Every (s,a) pair is visited infinitely often.

Non-Deterministic Case

$\delta(s,a)$ and $r(s,a)$ are probability distributions.

Q-learning still works!

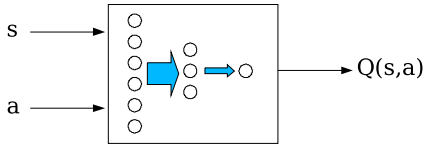
$$\hat{Q}_n(s,a) = (1-\alpha)\hat{Q}_{n-1}(s,a) + \alpha\left[r_n + max_b \hat{Q}_{n-1}(\delta(s,a),b)\right]$$

where $\alpha = 1/n$

Watkins and Dayan (1992) proved that Q-learning converges in the stochastic case.

What Does This Have To Do With Neural Networks?

- Q-learning with an explicit Q table is only feasible for small state/action spaces.
- For larger problems, we must approximate Q(s,a).
- Use a neural net!



Temporal Difference (TD) Learning: Sutton & Barto

- Suppose there are no action choices, because:
  - There are no actions, or ...
  - The policy Π is fixed.
- Now we have a simpler problem: just learn to predict rewards from states.
  - Under a fixed policy Π, an MDP becomes just a Markov process (no decisions).
  - Can learn V(s) rather than Q(s,a).
  - Once you learn the value function V(s), you can use it to improve Π ("policy iteration").
  - You can also interleave Π-learning with V-learning ("actor/critic").

TD Learning

Write V_t as shorthand for V(s(t)). Exponential discounting:

$$V_t = E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau\right]$$

This can be rewritten recursively:

$$V_t = r_t + \gamma V_{t+1}$$

Suppose we're trying to approximate V with a neural network. Consistency condition on V, suggesting an error signal:

$$\delta_t = r_t + \gamma V_{t+1} - V_t$$

TD Learning

- We're in state s(t). Estimate value V_t.
- Receive reward r_t, and transition to some new state s(t+1). Estimate value V_{t+1}.
- How did we do on predicting the value of state s(t)?
  - Early prediction: V_t
  - Later prediction: $r_t + \gamma V_{t+1}$
  - Use error signal δ to adjust our estimate of V_t. Since we're measuring actual rewards r_t, our estimates should improve over time.

Speeding up TD

We can speed up convergence in TD by looking further ahead, i.e., backing up reward information to earlier states.

$$V_t^{(1)} = r_t + \gamma V_{t+1} \qquad \delta_t^{(1)} = r_t + \gamma V_{t+1} - V_t$$

$$V_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V_{t+2} \qquad \delta_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V_{t+2} - V_t$$

$$V_t^{(3)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V_{t+3} \qquad \delta_t^{(3)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V_{t+3} - V_t$$

The TD(λ) Algorithm

$TD(\lambda)$ is a way to average different lookaheads, weighting nearer ones more:

$$\begin{aligned} V_t^{\lambda} &= (1-\lambda)\left[V_t^{(1)} + \lambda V_t^{(2)} + \lambda^2 V_t^{(3)} + \ldots \right] \\ &= V_t^{(1)} - \lambda V_t^{(1)} + \lambda V_t^{(2)} - \lambda^2 V_t^{(2)} + \lambda^2 V_t^{3} - \lambda^3 V_t^{3} + \ldots \\ &= r_t + \gamma \left[(1-\lambda) V_{t+1} + \lambda V_{t+1}^{\lambda} \right] \end{aligned}$$

If you think about this as backing up rewards, you can track recently visited states using exponentially decaying 'eligibility traces' $e_t(s)$, and update all states by $e_t(s)\delta_t$.

Tesauro's Version of TD

$$\Delta w_t = \alpha (V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k$$

- $\alpha(V_{t+1} - V_t)$: TD error, but r and γ are missing: for absorbing Markov chains. Only one reward, at end.
- $\sum_{k=1}^{t} \lambda^{t-k}$: λ sum: updates t previous states at once
- $\nabla_w V_k$: Gradient of V: allows use of arbitrary gradient descent learner.

TD-gammon (Tesauro 1992)

- One of the great successes of machine learning.
- Network learned world-class backgammon entirely from self-play.
- Won computer backgammon championship.
- Lost by 1 point in 40 games against human champion.
- Developed strategies now used by experts.
- Why did it work so well?

TD Is Well-Suited for Backgammon

- Backgammon moves can be seen as an absorbing Markov process:
  - Game is stochastic: involves dice rolling
  - Delayed reward signal: win or lose.
  - Board configuration contains all state.
- Can be played well without deep search.
- Deep search impractical: high branching factor.

  checkers: ~10, chess: ~40, backgammon: ~400

An Earlier Program: Neurogammon (Tesauro & Sejnowski)

- Neurogammon was a backprop net that learned to evaluate backgammon moves.
- The input was a description of the board state before the move, and the state after the move.
- Carefully designed board representation included "hints": pre-computed features that are known to be important for evaluating board positions.
- (1) pip count, (2) degree of contact, (3) # points occupied in inner board, (4) # point occupied in opponent's inner board, (5) total # men in opponent's inner board, (6) presence of a "prime" formation, (7) blot exposure, (8) strength of blockade.

Neurogammon (cont.)

- Best performance was with two hidden layers in a 459-24-24-1 network. Training set had 3202 positions.
- Performance roughly equivalent to an intermediate-level human backgammon player.
- A later version used six networks to make decisions in different phases of the game, plus a seventh network for doubling decisions.
- Won the Computer Olympiad in London in 1989, defeating 5 other backgammon programs.
  - First learning program to win a computer game tournament.

Comparison of Two Later Systems

- TD-gammon
  - Plays games against itself
  - Reward = 1 at end if white wins
  - Should learn probability of white winning in each state
- Supervised backprop (EP: Expert Preferences)
  - Training set is game states with moves (successor states) chosen by an expert player.
  - Net trained to increase its rating of chosen state, decrease rating of other possibilities.
  - Should learn high value for expert's "favorite" states.
  - A simplified version of earlier Neurogammon.

More About the Two Systems

- Value of a state is computed by a sigmoidal neural net with one hidden layer.
  - Number of hidden units varied.
- Input codes the board state, whose move.
  - No derived features.
- To play:
  - Roll dice, find possible moves.
  - Evaluate them
  - Choose thebest

Results

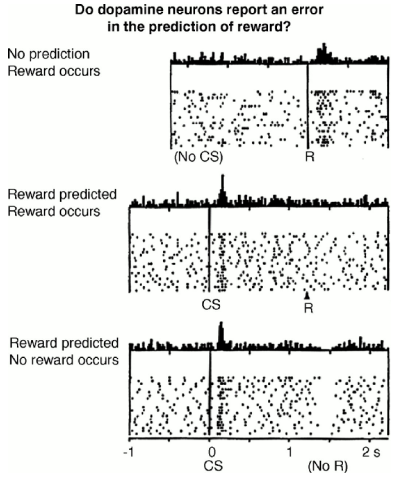
- Trained TD-gammon on 200,000 games.
  - Training games needed scaled linearly with # weights.
- Percentage “chance of win” estimates often badly off (10%), but the bias is consistent between different moves under consideration, so it cancels.
- TD-gammon also beats Neurogammon, which is a highly tweaked EP with hand-tuned features, larger network, and custom training regimen.

TD-gammon 2.0

- Improvement over original TD-gammon by adding:
  - Hand-tuned board features from Neurogammon
  - Two-ply search
  - Much more training (1.5 million games)
- Result: world-class backgammon player.
- Comparable to human experts.

TD Learning in the Brain

- Dopamine is thought to be the brain's reward signal.
- Schultz recorded from dopamine cells in monkeys conditioned to expect reward.
- Unexpected rewards caused bursting. Missed rewards caused a pause in firing.



Dopamine and TD

- The behavior of dopamine neurons resembles the error signal δ in TD learning.

$$\delta(t) = r(t) + \gamma V(t+1) - V(t)$$

  - Dopamine cells have a small tonic firing rate.
  - Positive δ causes bursting.
  - Negative δ causes a pause.
- The basal ganglia project to the dopaminergic neurons.
- BG could be computing the value function V(t). The dopamine error signal is used to train BG & cortex.