CS 461: Machine Learning
Lecture 9

Dr. Kiri Wagstaff
wkiri@wkiri.com

Plan for Today

- Reinforcement Learning
- Ensemble Learning
  - How to combine forces?
  - Voting
  - Bagging
  - Boosting
- Evaluations

Reinforcement Learning

Chapter 16

Summary: Reinforcement Learning

- Reinforcement Learning
  - How different from supervised, unsupervised?
- Key components
  - Actions, states, transition probs, rewards
  - Markov Decision Process
  - Episodic vs. continuing tasks
  - Value functions, optimal value functions
- How to learn a good policy?

Define a reinforcement learning problem: Drive a car

- States?
- Actions?
- Reward?

Value Functions

- The value of a state = expected return starting from that state; depends on the agent's policy:

State-value function for policy π:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The value of taking an action in a state under policy π = expected return starting from that state, taking that action, and then following π:

Action-value function for policy π:

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma\left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{ R_t \,\middle|\, s_t = s \right\} \\ &= E_\pi\left\{ r_{t+1} + \gamma V^\pi\left(s_{t+1}\right) \,\middle|\, s_t = s \right\} \end{aligned}$$

Or, without the expectation operator:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

Optimal Value Functions

- For finite MDPs, policies can be partially ordered:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi}(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- Optimal policy = π^*
- Optimal state-value function:

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all } s \in S$$

- Optimal action-value function:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

This is the expected return for taking action a in state s and thereafter following an optimal policy.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^*, one-step-ahead search produces the long-term optimal actions.

Given Q^*, the agent does not even have to do a one-step-ahead search:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

Model-Based Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is known
- There is no need for exploration
- Can be solved using dynamic programming
- Solve for

$$V^*(s_t) = \max_{a_t}\left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1})\right)$$

- Optimal policy

$$\pi^*(s_t) = \arg\max_{a_t}\left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1})\right)$$

Temporal Difference Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is not known; model-free learning
- There is need for exploration to sample from $P(s_{t+1} \mid s_t, a_t)$ and $p(r_{t+1} \mid s_t, a_t)$
  - Exploration vs. Exploitation
- Use the reward received in the next time step to update the value of current state (action)
- The temporal difference between the value of the current action and the value discounted from the next state

Deterministic Rewards and Actions

- Deterministic: single possible reward and next state

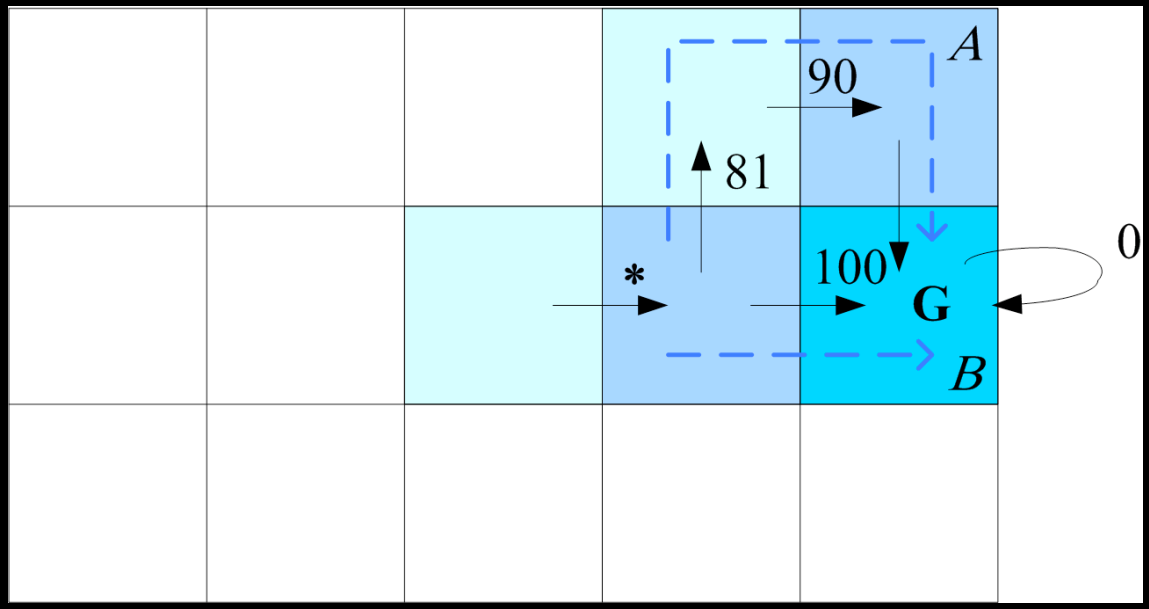
$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

- Used as an update rule (backup)

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

- Updates happen only after reaching the reward (then are "backed up")

Starting at zero, *Q* values increase, never decrease



γ=0.9

Consider the value of action marked by '*':

If path A is seen first, Q(*)=0.9*max(0,81)=73

Then B is seen, Q(*)=0.9*max(100,81)=90

Or,

If path B is seen first, Q(*)=0.9*max(100,0)=90

Then A is seen, Q(*)=0.9*max(100,81)=90

Q values increase but never decrease

Nondeterministic Rewards and Actions

- When next states and rewards are nondeterministic (there is an opponent or randomness in the environment), we keep averages (expected values) instead as assignments
- Q-learning (Watkins and Dayan, 1992):

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \eta\left(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1},a_{t+1}) - \hat{Q}(s_t,a_t)\right)$$

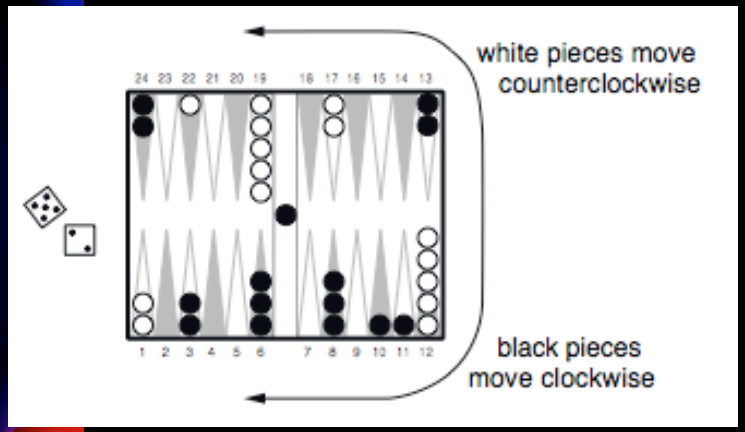
backup

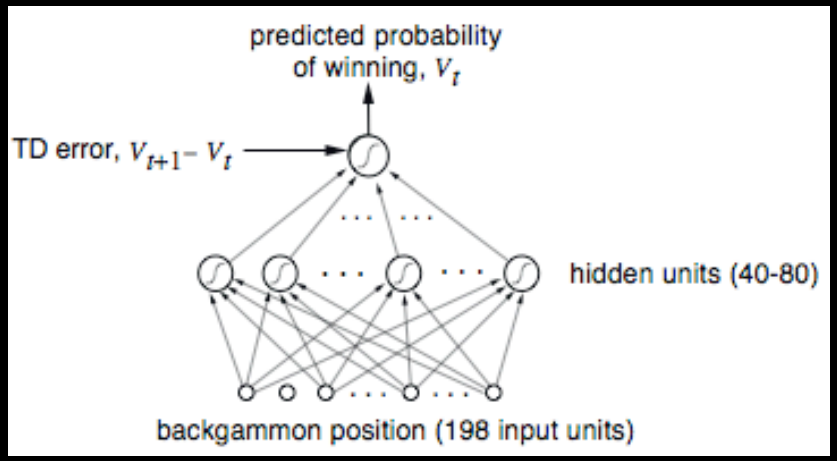
- Learning V (TD-learning: Sutton, 1988)

$$V(s_t) \leftarrow V(s_t) + \eta\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$$

TD-Gammon

Tesauro, 1992–1995





Start with a random network

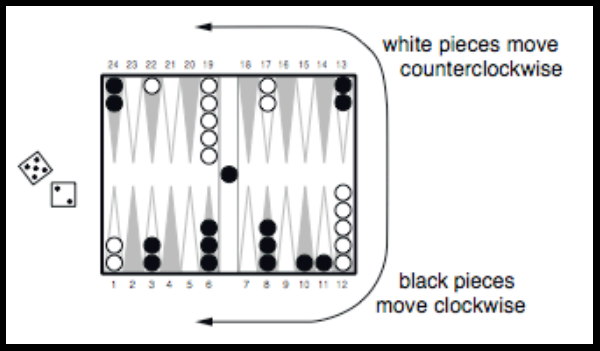
Play very many games against self

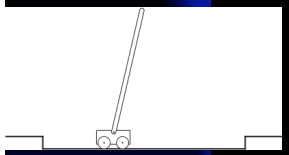
Learn a value function from this simulated experience

Action selection by 2–3 ply search

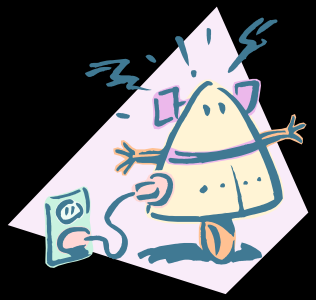
Program	Training games	Opponents	Results
TDG 1.0	300,000	3 experts	-13 pts/51 games
TDG 2.0	800,000	5 experts	-7 pts/38 games
TDG 2.1	1,500,000	1 expert	-1 pt/40 games

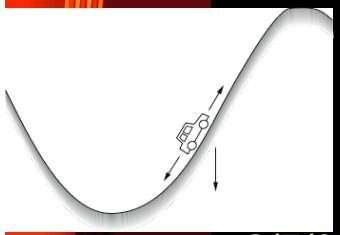
Review





- Reinforcement Learning
  - How different from supervised, unsupervised?
- Key components
  - Actions, states, transition probs, rewards
  - Markov Decision Process
  - Episodic vs. continuing tasks
  - Value functions, optimal value functions
- Learn: policy (based on V, Q)
  - Model-based: value iteration, policy iteration
  - TD learning
    - Deterministic: backup rules (max)
    - Nondeterministic: TD learning, Q-learning (running avg)





Ensemble Learning

Chapter 15

What is Ensemble Learning?

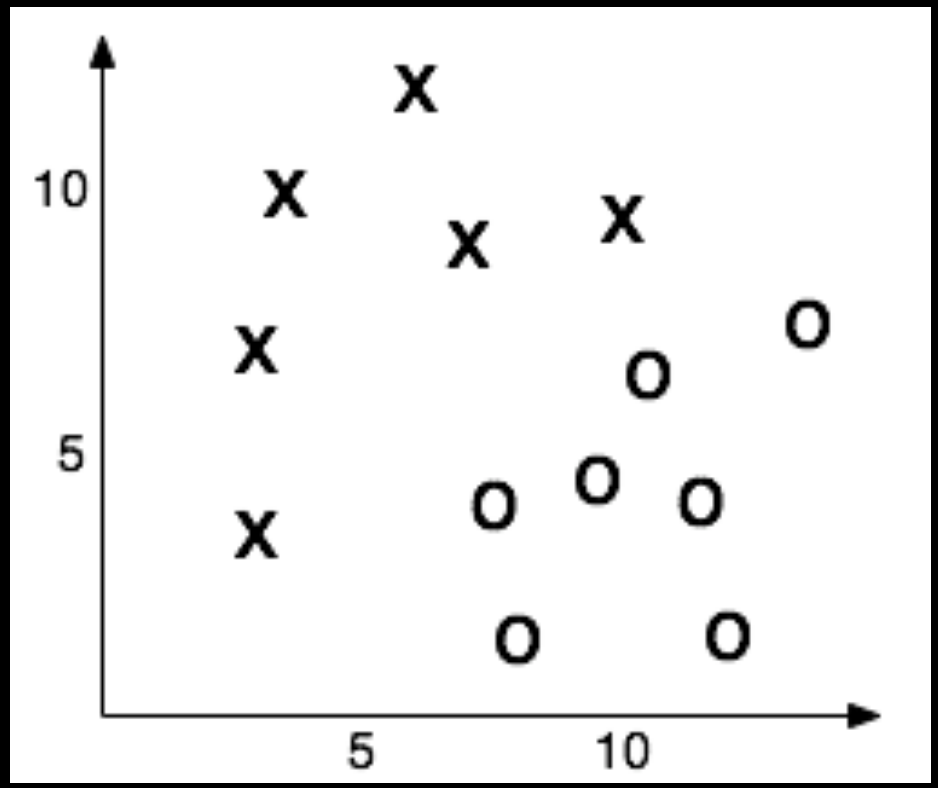
- "No Free Lunch" Theorem
  - No single algorithm wins all the time!
- Ensemble: collection of base learners
  - Combine the strengths of each to make a super-learner
  - Also considered "meta-learning"
- How can you get different learners?

- How can you combine learners?

Where do Learners come from?

- Different learning algorithms
- Algorithms with different choice for parameters
- Data set with different features
- Data set = different subsets
- Different sub-tasks

Exercise: x's and o's

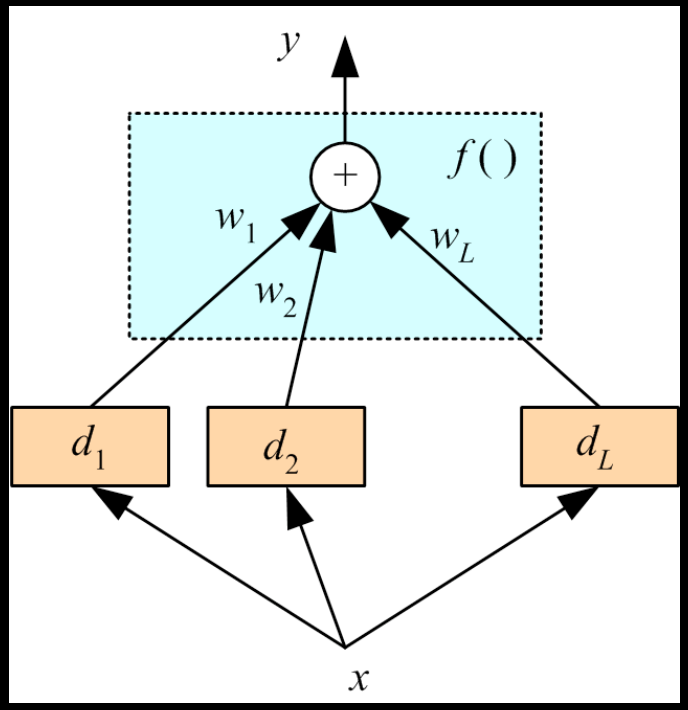


Combine Learners: Voting

- Linear combination (weighted vote)

$$y = \sum_{j=1}^{L} w_j d_j$$

$$w_j \geq 0 \quad \text{and} \quad \sum_{j=1}^{L} w_j = 1$$



- Classification

$$y_i = \sum_{j=1}^{L} w_j d_{ji}$$

Bayesian

$$P(C_i \mid x) = \sum_{\text{all models } \mathcal{M}_j} P(C_i \mid x, \mathcal{M}_j) P(\mathcal{M}_j)$$

Different Learners: Bagging

- Bagging = "bootstrap aggregation"
  - Bootstrap: draw N items from X with replacement
- Want "unstable" learners
  - Unstable: high variance
  - Decision trees and ANNs are unstable
  - K-NN is stable
- Bagging
  - Train L learners on L bootstrap samples
  - Combine outputs by voting

Different Learners: Boosting

- Boosting: train next learner on mistakes made by previous learner(s)
- Want "weak" learners
  - Weak: P(correct) > 50%, but not necessarily by a lot
  - Idea: solve easy problems with simple model
  - Save complex model for hard problems

Original Boosting

1. Split data X into {X1, X2, X3}
2. Train L1 on X1
   - Test L1 on X2
3. Train L2 on L1's mistakes on X2 (plus some right)
   - Test L1 and L2 on X3
4. Train L3 on disagreements between L1 and L2

- Testing: apply L1 and L2; if disagree, use L3

- Drawback: need large X

AdaBoost = Adaptive Boosting

- Arbitrary number of base learners
- Re-use data set (like bagging)
- Use errors to adjust probability of drawing samples for next learner
  - Reduce probability if it's correct
- Testing: vote, weighted by training accuracy

- Key difference from bagging:
  - Data sets not chosen by chance; instead use performance of previous learners to select data

AdaBoost

Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

Testing:

Given x, calculate $d_j(x), j = 1, \ldots, L$

Calculate class outputs, $i = 1, \ldots, K$:

$$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

AdaBoost Applet

- http://www.cs.ucsd.edu/~yfreund/adaboost/index.html

Summary: Key Points for Today

- Reinforcement Learning
  - Value and Return functions
  - Model-based learning
  - TD-learning
    - Deterministic: dynamic programming for best policy
    - Non-deterministic: Q-learning, TD-learning
- No Free Lunch theorem
- Ensemble: combine learners
  - Voting
  - Bagging
  - Boosting

Next Time

- Reading question volunteers:
- Final Project Presentations
  - Use order on website
- Submit slides on CSNS by midnight March 13
  - No, really
  - You may not be able to present if you don't
- Reports are due to CSNS midnight March 14
  - Early submission: March 9