CS 440: Introduction to Artificial Intelligence

Lecture 21, Nov 23, 2015

Matthew Stone

November 30, 2015

Recap—Decision principle

- Agent prefers outcome that maximizes expected utility

Recap—Decision principle

- Agent prefers outcome that maximizes expected utility

Formalism

- Choose a as

$$\underset{a}{\operatorname{argmax}}\, EU(a|\mathbf{e})$$

Recap—Methodology

- Build prototype agent
- Build schema of possible designs
- Get experience from agent acting randomly
- Build model from schema plus experience
- Solve model for policy
- Use policy

Recap—Efficient Representation

- Have set of *states*
- Have set of *actions*
- Have transition model $P(S_{i+1}|A_i, S_i)$
- Have reward function $R(S_i, A_i, S_{i+1})$
- Utility is sum of rewards, perhaps discounted into the future (1 unit of fun tomorrow is worth γ units of fun now)

Value and Q-value

- Value V gives expected outcome for each state.

$$V(S) = \max_A \sum_{S'} P(S'|A,S)(R(S,A,S') + \gamma V(S'))$$

- Q-value Q gives expected outcome for each action in each state

$$Q(S,A) = \sum_{S'} P(S'|A,S)(R(S,A,S') + \gamma V(S'))$$

or

$$Q(S,A) = \sum_{S'} P(S'|A,S)(R(S,A,S') + \gamma \max_{A'} Q(S',A'))$$

Basic Q Learning

- Have table of Q-values
- Start in state S
- Choose action A
- Get reward r
- Move to state S'
- That gives new estimate of $Q(S, A)$:

$$r + \gamma \max_{A'} Q(S', A')$$

- Adjustment rule:

$$Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma \max_{A'} Q(S', A') - Q(S, A))$$

- Demo http://thierry.masson.free.fr/IA/en/qlearning_applet.htm

Case Study: Simple Blackjack

- Cards have point values
  (face value, 10 for face cards, 1 or 11 for ace)
- Must have 21 or less to win
- Must have more than dealer to win
- Dealt 2 cards to start
- Then as many more as you ask for
- Can't see dealer's card—simplification!
- Win or lose only—simplification of actual betting!

Modeling

- States with choices:
  Total score (4 to 20)
  Extra bit for scores 12 to 20:
  Is there an ace counted for 11 that could be counted 1?
  (yes states numbered 23–31 in demo)
- Plus final stop state with reward from game outcome
- Decision:
  Hit – get another card
  Stand – stop
- Transition:
  State $\times$ Stand $\rightarrow$ Stop
  State $\times$ Hit $\rightarrow$ Total with new card or Stop if over 21
- Reward: 1 if you win, -1 if you lose.

Demo

- Dealer stands at 17 or higher.

Demo: `http://lcn.epfl.ch/tutorial/english/reinf-bj/html/index.html`

Approximating Q-values

- Q learning is based on comparing prediction

$$Q(S, A)$$

to actual outcome

$$r + \gamma \max_{A'} Q(S', A')$$

- Works by incremental adjustments

$$Q(S, A) \leftarrow Q(S, A) + \alpha(r + \gamma \max_{A'} Q(S', A') - Q(S, A))$$

Much Like Perceptron Learning

- Make prediction
$$w \cdot x > 1$$
- Compare to actual outcome
  ϵ is sign of error given true class of x
- Make incremental adjustments
$$w \leftarrow w + \alpha \epsilon x_i$$

Until weights converge

Combining Neural Nets and Reinforcement Learning

- Represent state S in terms of a feature vector x_S
  What are the important aspects of the world that predict successful strategy in S?
- Use function approximation to model $Q(S, A)$
  For example: $Q(S, A) = w_A \cdot x_S$
- Update by temporal difference procedure
  Shared by Q-learning and perceptron learning

Combining Neural Nets and Reinforcement Learning

- Get estimate from experience:

$$r + \gamma \max_{A'} (w_{A'} \cdot x_{S'})$$

- Update weights to reduce error:

$$w_A \leftarrow w_A + \alpha (r + \gamma \max_{A'} (w_{A'} \cdot x_{S'}) - w_A \cdot x_S) x_S$$

Generalization of this technique used in first top computer Backgammon player (Tesauro)

Games

A *game* is an environment with rewards that depend on the action of more than one agent.

- Games with turns
  tic–tac–toe, checkers, chess, backgammon, go
- Games with simultaneous moves
  Rock–paper–scissors, prisoner's dilemma, sealed-bid auctions
- Approaches in the two cases are different

Games with Turns

Intuitively like planning

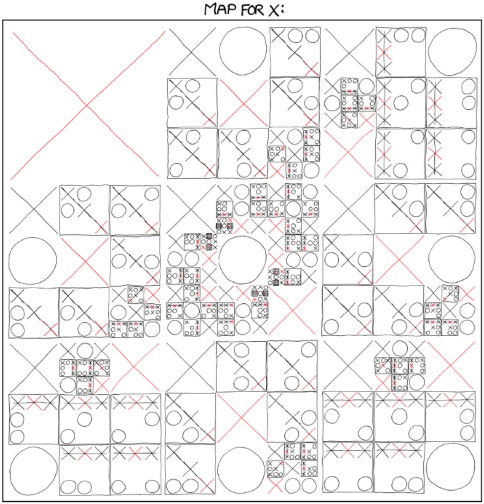
- Map out the future
- Anticipate that you will make good choices
- Expect that your opponent will make good choices
- Work backwards to what you should do now

Games with Turns

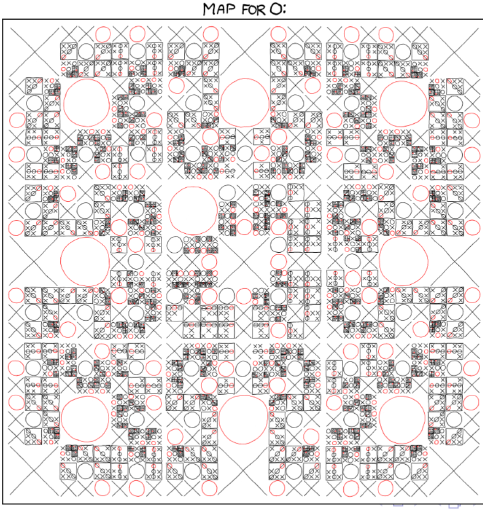
Wind up with a strategy

- Describes optimal play
- Given optimal (or possible) play by opponent

Example



Example



Games with simultaneous play

Intuitively like solving an equation

- Your move depends on your opponent's move
- Their move depends on yours
- You choose them simultaneously
- Good strategies "balance" the decisions

Example: Rock–paper–scissors

R and C choose actions jointly. C gets these payoffs:

		C		
		rock	paper	scissors
	rock	0	1	-1
R	paper	-1	0	1
	scissors	1	-1	0

Example: Rock–paper–scissors

Need to be unpredictable

- If R knows what C is going to do, R can win
- If C knows what R is going to do, C can win

If both guess any move randomly with probability $\frac{1}{3}$, neither can exploit the other

Different needs for AI techniques

Games with turns

- manage large search spaces
- develop good heuristics and approximations
- (only secondarily) learn specifically about opponent

Games with simultaneous play

- learn specifically about opponent
- (less so) develop good heuristics and approximations
- (only secondarily) manage search