Machine Learning, Reinforcement Learning

AI Class 25 (Ch. 21.1, 20.2–20.2.5, 20.3)

Slides drawn from Drs. Tim Finin, Paula Matuszek, Rich Sutton, Andy Barto, and Marie desJardins, with thanks

1

Bookkeeping

- **12/5: Panel by the Drs. Matuszek.** 40-50 minutes discussion, followed by 25-35 minutes of Q&A.
- **12/8: Your final, functioning agent.*** This is what will play in the tournament.
- **12/10: Agent cards tournament!**
- **12/11: Final writeup**
- **TBD: Final review session.**
- **12/17: Final exam**

2

More

- Everyone will get credit for 2 missing homeworks
- Please turn in your agent **on Blackboard**
  - Those who emailed it on time will receive extra credit
- We will release the game engine today or tomorrow
- We still have office hours!

3

Today's Class

- Machine Learning: A quick retrospective
- Reinforcement Learning
- Next time:
  - The EM algorithm
  - Monte Carlo and Temporal Difference
- Upcoming classes:
  - 100+ Years of AI – **Bring questions**
  - Tournament

4

Review: What is ML?

- ML is a way to get a computer (in our parlance, a **system**) to do things without having to explicitly describe what steps to take.
- By giving it **examples** (training data)
- Or by giving it **feedback**
- It can then look for patterns which explain or predict what happens.
- The learned system of beliefs is called a **model**.

5

5

Revew: Architecture of a ML System

- Every machine learning system has four parts:
  1. A **representation or model** of what is being learned.
  2. An **actor**: Uses the representation and actually does something.
  3. A **critic**: Provides feedback.
  4. A **learner**: Modifies the representation / model, using the feedback.

6

6

Review: Representation

- A learning system must have a representation or model of what is being learned.
- This is what changes based on experience.
- In a machine learning system this may be:
  - A mathematical model or formula
  - A set of rules
  - A decision tree
  - A policy
  - Or some other form of information

8

8

Review: Formalizing Agents

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A **mapping from states to actions**
  - Which is a **policy**, π

9

9

Learning Without a Model

- We saw how to learn a value function and/or a policy from a transition model
- What if we don't have a transition model?
- Idea #1: Build one
  - Explore the environment for a long time
  - Record all transitions
  - Learn the transition model
  - Apply value iteration/policy iteration
  - Slow, requires a lot of exploration, no intermediate learning
- Idea #2: Learn a value function (or policy) directly from interactions with the environment, **while exploring**

10

Reinforcement Learning

- We often have an agent which has a **task** to perform
  - It takes some actions in the world
  - At some later point, gets feedback on how well it did
  - The agent performs the same task repeatedly
- This problem is called **reinforcement learning**:
  - The agent gets positive reinforcement for tasks done well
  - And gets negative reinforcement for tasks done poorly
  - Must somehow figure out which actions to take next time

11

11

Animals Game

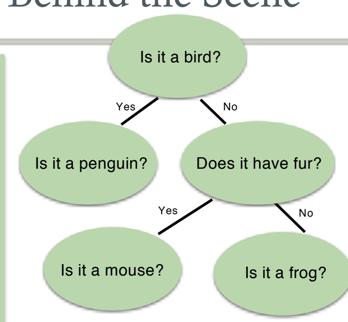
- Human: I'm thinking of an animal.
- Computer: Is it a bird?
- Human: No.
- Computer: Is it a frog?
- Human: No.
- Computer: What is it?
- Human: A mouse.
- Computer: **Tell me a question that distinguishes a frog from a mouse.**
- Human: Does it have fur?
- Computer: What is the answer for a frog?
- Human: No.

12

12

Animals Behind the Scene

Computer: Is it a bird?
Human: No
Computer: Is it a frog?
Human: No
Computer: What is it?
Human: A mouse
Computer: Tell me a question that distinguishes a frog from a mouse.
Human: Does it have fur?
Computer: What is the answer for a frog?
Human: no



After several rounds...

13

13

Animals Guessing Game Architecture

- All of the parts of ML Architecture:
  - The Representation is a sequence of questions and pairs of yes/no answers (called a binary decision tree).
  - The Actor "walks" the tree, interacting with a human; at each question it chooses whether to follow the "yes" branch or the "no" branch.
  - The Critic is the human player telling the game whether it has guessed correctly.
  - The Learner elicits new questions and adds questions, guesses and branches to the tree.

14

14

Reinforcement Learning

- This is a simple form of **Reinforcement Learning**
- Feedback is at the end, on a **series** of actions.
- Very early concept in Artificial Intelligence!
- Arthur Samuels' checker program was a simple reinforcement based learner, initially developed in 1956.
- In 1962 it beat a human checkers master.

www-03.ibm.com/ibm/history/ibm100/us/en/icons/ibm700series/impacts/

15

Reinforcement Learning (cont.)

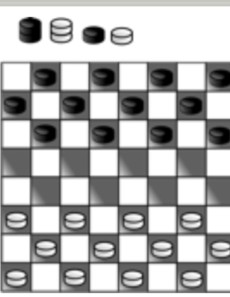
- Goal: agent acts in the world to maximize its rewards
- Agent has to figure out what it did that made it get that reward/punishment
  - This is known as the credit assignment problem
- RL can be used to train computers to do many tasks
  - Backgammon and chess playing
  - Job shop scheduling
  - Controlling robot limbs

16

16

Simple Example

- Learn to play checkers
  - Two-person game
  - 8x8 boards, 12 checkers/side
  - relatively simple set of rules: http://www.darkfish.com/checkers/rules.html
  - Goal is to eliminate all your opponent's pieces



https://pixabay.com/en/checker-board-black-game-pattern-29911

17

Representing Checkers

- First we need to represent the game
- To completely describe one step in the game you need
  - A representation of the game board.
  - A representation of the current pieces
  - A variable which indicates whose turn it is
  - A variable which tells you which side is "black"
- There is no history needed
- A look at the current board setup gives you a complete picture of the state of the game

which makes it a ____ problem?

18

18

Representing Rules

- Second, we need to represent the rules
- Represented as a **set of allowable moves** given board state
  - If a checker is at row x, column y, and row x+1 column y±1 is empty, it can move there.
  - If a checker is at (x,y), a checker of the opposite color is at (x+1, y+1), and (x+2,y+2) is empty, the checker must move there, and remove the "jumped" checker from play.
- There are additional rules, but all can be expressed in terms of the state of the board and the checkers.
- Each rule includes the outcome of the relevant action in terms of the state.

19

19

What Do We Want to Learn

- Given
  - A description of some state of the game
  - A list of the moves allowed by the rules
  - **What move should we make?**
- Typically more than one move is possible
  - Need strategies, heuristics, or hints about what move to make
  - **This is what we are learning**
- We learn **from** whether the game was won or lost
  - Information to learn from is sometimes called "training signal"

22

22

Simple Checkers Learning

- Can represent some heuristics in the same formalism as the board and rules
  - If there is a legal move that will create a king, take it.
    - If checkers at (7,y) and (8,y-1) or (8,y+1) is free, move there.
  - If there are two legal moves, choose the one that moves a checker farther toward the top row
    - If checker(x,y) and checker(p,q) can both move, and x>p, move checker(x,y).
  - But then each of these heuristics needs some kind of priority or **weight.**

23

23

Formalization for RL Agent

- Given:
  - A state space S
  - A set of actions $a_1, \ldots, a_k$ including their results
  - A set of heuristics for resolving conflict among actions
  - Reward value at the end of each trial (series of action) (may be positive or negative)
- Output:
  - A policy (a mapping from states to preferred actions)

24

24

Learning Agent

- The general algorithm for this learning agent is:
  - Observe some state
  - If it is a terminal state
    - Stop
    - If won, **increase** the weight on **all** heuristics used
    - If lost, **decrease** the weight on **all** heuristics used
  - Otherwise choose an action from those possible in that state, using heuristics to select the preferred action
  - Perform the action

25

25

Policy

- A complete mapping from states to actions
  - There must be an action for each state
  - There may be more than one action
  - Not necessarily optimal
- The goal of a learning agent is to **tune** the policy so that the preferred action is optimal, or at least good.
  - analogous to training a classifier
- Checkers
  - Trained policy includes all legal actions, with **weights**
  - "Preferred" actions are **weighted up**

26

26

Approaches

- Learn policy directly: Discover function mapping from states to actions
  - Could be directly learned values
    - Ex: Value of state which removes last opponent checker is +1.
  - Or a heuristic function which has itself been trained
- Learn utility values for states (value function)
  - Estimate the value for each state
  - Checkers:
    - How happy am I with this state that turns a man into a king?

27

27

Value Function

- The agent knows what state it is in
- It has actions it can perform in each state
- Initially, don't know the value of any of the states
- If the outcome of performing an action at a state is deterministic, then the agent can update the utility value U() of states:
  - U(oldstate) = reward + U(newstate)
- The agent learns the utility values of states as it works its way through the state space

28

28

Learning States and Actions

- A typical approach is:
- At state S choose, some action A ← How?
- Taking us to new State S_1
  - If S_1 has a positive value: increase value of A at S.
  - If S_1has a negative value: decrease value of A at S.
  - If S_1 is new, initial value is unknown: value of A unchanged.
- One complete learning pass or **trial** eventually gets to a terminal, deterministic state. (E.g., "win" or "lose")
- Repeat until? Convergence? Some performance level?

29

29

Selecting an Action

- Simply choose action with highest (current) expected utility?
- Problem: each action has two effects
  - Yields a **reward** on current sequence
  - Gives **information** for learning future sequences
- Trade-off: immediate good for long-term well-being
  - Like trying a shortcut: might get lost, might find quicker path
- Exploration vs. exploitation again.

30

30

Exploration vs. Exploitation

- Problem with naïve reinforcement learning:
  - What action to take?
  - **Best apparent action, based on learning to date** } Exploitation
    - Greedy strategy
    - Often prematurely converges to a suboptimal policy!
  - **Random (or unknown) action** } Exploration
    - Will cover entire state space
    - Very expensive and slow to learn!
    - When to stop being random?
  - Balance exploration (try random actions) with exploitation (use best action so far)

31

More on Exploration

- Agent may sometimes choose to explore suboptimal moves in hopes of finding better outcomes
  - Only by visiting all states frequently enough can we guarantee learning the true values of all the states
- When the agent is **learning**, ideal would be to get accurate values for all states
  - Even though that may mean getting a negative outcome
- When agent is **performing**, ideal would be to get optimal outcome
- A learning agent should have an **exploration policy**

32

32

Exploration Policy

- Wacky approach (exploration): act randomly in hopes of eventually exploring entire environment
  - Choose any legal checkers move
- Greedy approach (exploitation): act to maximize utility using current estimate
  - Choose moves that have in the past led to wins
- Reasonable balance: act more wacky (exploratory) when agent has little idea of environment; more greedy when the model is close to correct
  - Suppose you know no checkers strategy?
  - What's the best way to get better?

33

33

Example: N-Armed Bandits

- A row of slot machines
- Which to play and how often?
- State Space is a set of machines
  - Each has cost, payout, and percentage values
- Action is pull a lever.
- Each action has a positive or negative result
  - …which then adjusts the perceived utility of that action (pulling that lever)

34

N-Armed Bandits Example

- Each action initialized to a standard payout
- Result is either some cash (a win) or none (a lose)
- **Exploration:** Try things until we have estimates for payouts
- **Exploitation:** When we have some idea of the value of each action, choose the best.
- Clearly this is a heuristic.

After some # of successful trials, or with some statistical **confidence**, or when our value function isn't changing (much), or…

- No proof we ever find the best lever to pull!
  - The more exploration we can do the better our model
  - But the higher the cost over multiple trials

35

RL Summary 1:

- **Reinforcement learning systems**
  - Learn **series** of actions or decisions, rather than a single decision
  - Based on feedback given at the end of the series
- A reinforcement learner has
  - A goal
  - Carries out trial-and-error search
  - Finds the best paths toward that goal

36

RL Summary 2:

- A typical reinforcement learning system is an active agent, interacting with its environment.
- It must balance:
  - Exploration: trying different actions and sequences of actions to discover which ones work best
  - Exploitation (achievement): using sequences which have worked well so far
- Must learn **successful sequences of actions** in an uncertain environment

37

RL Summary 3

- Very hot area of research at the moment
- There are **many** more sophisticated RL algorithms
  - Most notably: probabilistic approaches
- Applicable to game-playing, search, finance, robot control, driving, scheduling, diagnosis, …

38