Introduction to Artificial Intelligence

Lecture 20: Sequential decision making and reinforcement learning

November 13, 2025

Sequential decision-making: Learning from rewards

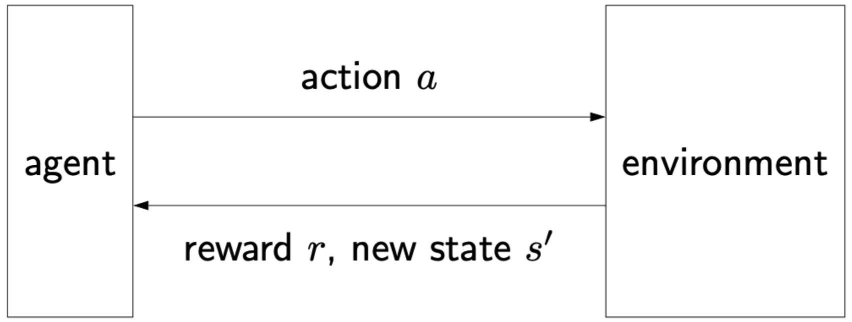
- Markov decision process
  - Have a mental model of how the world works (offline training)
  - Find policy to collect maximum rewards

- Reinforcement learning
  - Don't know how the world works (online learning)
  - Perform actions in the world to find out and collect rewards
  - Differs from MDP because the agent is not given the MDP as a problem to solve; the agent is in the MDP. It may not know the transition model or reward function

- Same goal: maximize the expected sum of rewards

Sequential decision-making: Learning from rewards

- Algorithm: reinforcement learning from past actions and reward signals
  - For $t = 1,2,3,\dots$
    - Choose action $a_t = \pi(s_{t-1})$
    - Receive reward r_t and observe new state s_t
    - Update model parameters



Mystery game

- **Example: mystery buttons**
  - For reach round $r = 1,2, \ldots$
    - You choose A or B
    - You move to a new state and get some rewards

- **Challenges of reinforcement learning**: we should take good actions to get rewards, but in order to know which actions are good, we need to explore and try different actions

Lecture plan

- Sequential decision-making and reinforcement learning
  - **Model-based reinforcement learning**
  - Model-free reinforcement learning
  - SARSA
  - Q-learning
  - Epsilon-greedy
  - Function approximation

Model-based value iteration

- **Observational data**: $s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$

- **Key idea: model-based reinforcement learning**
  - Estimate the MDP: $T(s, a, s')$ and $Reward(s, a, s')$

- Transition model:

$$\hat{T}(s, a, s') = \frac{\#\text{ times } (s, a, s') \text{ occurs}}{\#\text{ times } (s, a) \text{ occurs}}$$

- Rewards:

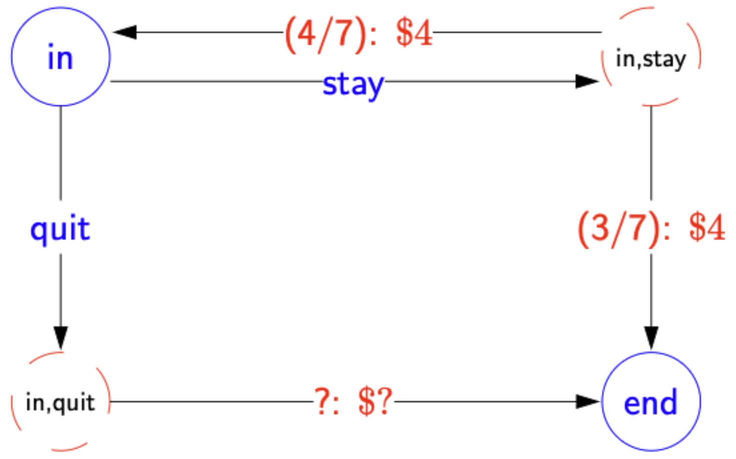
$$\hat{R}(s, a, s') = r \text{ in } (s, a, r, s')$$

Dice game II

- Imagine you are playing the dice game, but now you don’t know the rules of the game. Instead, part of the game is to figure out the rule of the game
- For each round $r = 1,2,\ldots$
  - You choose stay or quit
  - Similar to dice game I:
    - If quit, you get $10 and we end the game
    - If stay, you get $4 and then I roll a 6-sided dice
      - If the dice results in 1 or 2, we end the game
      - Otherwise, continue to the next round

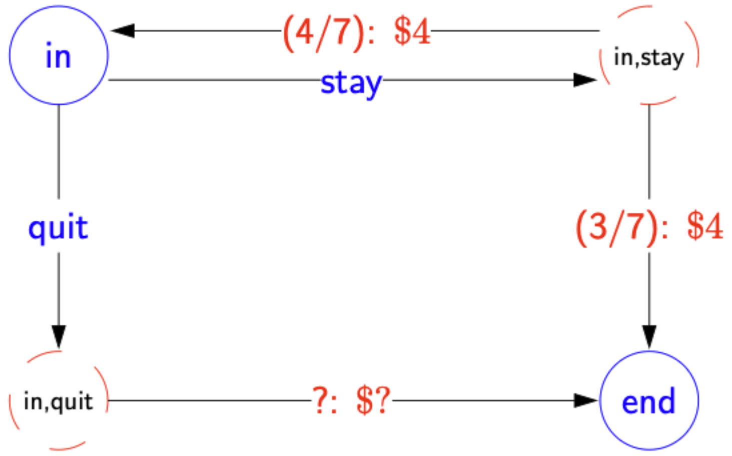
Model-based value iteration

- The first idea is called model-based value iteration
  - We try to estimate the model (transitions and rewards) using Monte Carlo simulation
  - Monte Carlo is a standard way to estimate the expectation of a random variable by taking an average over samples of that random variable



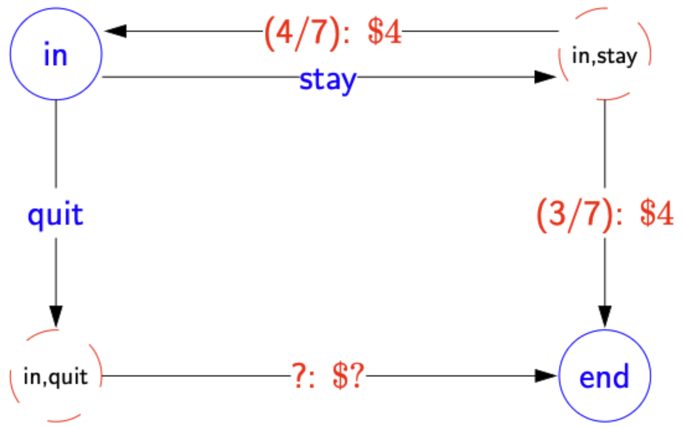
Model-based value iteration

- Data (following policy $\pi(s) = stay$): [in; stay, 4, end]
  - Estimates converge to true values after sufficiently many observations from the game
  - With the estimated MDP $(\hat{T}, \hat{R})$, compute the optimal policy using **value iteration**



Problem with this approach

- Problem: won't see (s, a) if $a \neq \pi(s)$ ($a =$ quit)
  - Never able to estimate their Q-value and know what the effect of those unseen actions are
- **Key idea: exploration**
  - To do reinforcement learning, need to explore the state space
- **Solution**: need to estimate π so that we could **explore** more explicitly (now we have to act to get data---this is a key difference to supervised learning)



Lecture plan

- Sequential decision-making and reinforcement learning
  - Model-based reinforcement learning
  - **Model-free reinforcement learning**
  - SARSA
  - Q-learning
  - Epsilon-greedy
  - Function approximation

From model-based to mode-free methods

- If our goal is to just find good policies, we can get a good estimate of the policy values $\hat{Q}_{opt}$

$$\hat{\mathrm{Q}}_{\mathrm{opt}}(\mathrm{s},\mathrm{a}) = \sum_{s'} \hat{T}(\mathrm{s},\mathrm{a},\mathrm{s}')\left[\hat{\mathrm{R}}(\mathrm{s},\mathrm{a},\mathrm{s}') + \gamma \hat{V}_{\mathrm{opt}}(\mathrm{s}')\right]$$

  - Use the estimate of $Q_{opt}(s,a)$ for prediction

- **Key idea of model-free learning: learn directly from experience, instead of estimating transition model or reward function**
  - Try to estimate $Q_{opt}(s,a)$ directly

Model-free Monte Carlo

- Data (following policy π):

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \dots; a_n, r_n, s_n$$

- Recall: $Q_\pi(s, a)$ is expected utility starting at s, first taking action a, and then following policy π
- Utility function: assume γ is known

$$u_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots$$

- Estimate Q-function

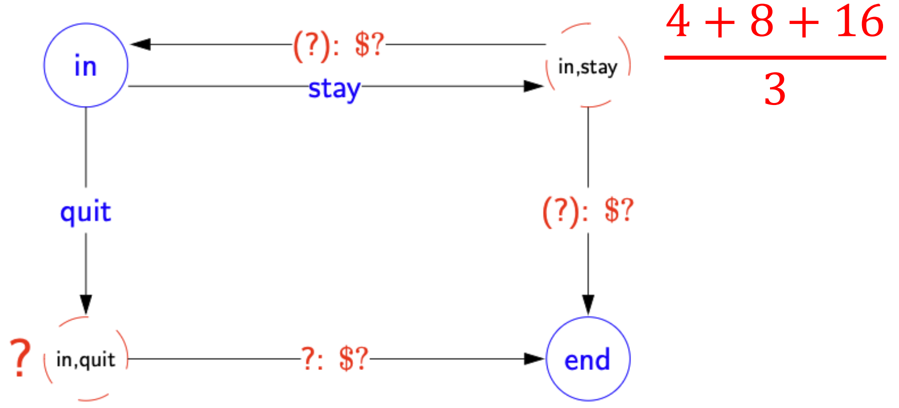
$$\hat{Q}_\pi(s, a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

Model-free Monte Carlo

- Data (following policy $\pi(s) = stay$):

[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]

- Note: we are estimating Q_π (not Q_{opt})



- **Definition: on-policy vs off-policy**
  - **On-policy algorithm**: estimate the value of the policy used to generate the data
  - **Off-policy learning**: model-based Monto Carlo, because the model we estimate did not depend on the exact policy

Model-free Monte Carlo

- We will show this algorithm from different perspectives
  - Observation data (following policy π)

$$s_0; a_1, r_1, s_1; a_2, r_2, s_2; a_3, r_3, s_3; \ldots; a_n, r_n, s_n$$

  - **Original formulation**

$$\hat{Q}_\pi(s,a) = \text{ average of } u_t \text{ where } s_{t-1} = s, a_t = a$$

  - **Equivalent formulation** (convex combination): on each (s, a, u)

$$\eta = \frac{1}{1 + \left(\# \, updates \; to \; (s,a)\right)}$$

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

- Set η to a rate that does not decay as quickly. This rate implies that a new examples contributes more than an old example

Model-free Monte Carlo

- Equivalent formulation (convex combination): On each (s, a, u),

$$\hat{Q}_\pi(s, a) \leftarrow (1 - \eta)\hat{Q}_\pi(s, a) + \eta u$$

- Equivalent formulation (stochastic gradient): on each (s, a, u),

$$\hat{Q}_\pi(s, a) \leftarrow \hat{Q}_\pi(s, a) - \eta[\hat{Q}_\pi(s, a) - u]$$

- **Objective function**: equivalent to running stochastic gradient descent on the following least-squares regression

$$\sum_{(s,a,u)} \left(\hat{Q}_\pi(s, a) - u\right)^2,$$

where $\hat{Q}_\pi(s, a)$: prediction

u: target

Lecture plan

- Sequential decision-making and reinforcement learning
  - Model-based reinforcement learning
  - Model-free reinforcement learning
  - **State-action-reward-state-action (SARSA)**
  - Q-learning
  - Epsilon-greedy
  - Function approximation

Using the utility

- SARSA: a classic on-policy temporal different algorithm in reinforcement learning
- Example: Observational data

[in; stay, 4, end]	$u = 4$
[in; stay, 4, in; stay, 4, end]	$u = 8$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 12$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$u = 16$

- Algorithm from the previous module (model-free Monte Carlo): On each (s, a, u):

$$\hat{Q}_\pi(s,a) \leftarrow (1-\eta)\hat{Q}_\pi(s,a) + \eta u$$

Using the reward plus future Q-values

- Suppose current estimate: $\widehat{Q}_\pi(s, stay) = 11$. With data

[in; stay, 4, end]	$4 + 0$
[in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$
[in; stay, 4, in; stay, 4, in; stay, 4, in; stay, 4, end]	$4 + 11$

- In SARSA, we update $\hat{Q}_\pi$ on each $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ as

$$\hat{Q}_\pi(s_t, a_t) \leftarrow (1 - \eta)\hat{Q}_\pi(s_t, a_t) + \eta\left[r_t + \gamma \hat{Q}_\pi(s_{t+1}, a_{t+1})\right]$$

- Importantly, SARSA's target is a combination of the data (the first step) and the estimate (for the rest of the steps). In contrast, model-free Monte Carlo's u is taken purely from the data

Lecture plan

- Sequential decision-making and reinforcement learning
  - Model-based reinforcement learning
  - Model-free reinforcement learning
  - State-action-reward-state-action (SARSA)
  - **Q-learning**
  - Epsilon-greedy
  - Function approximation

Q-learning

- Problem: model-free Monte Carlo only estimates Q_{π}, but want Q_{opt} to act optimally

- Can we get also an estimation of Q_{opt} in a model-free manner?
  - Q-learning is an algorithm that accomplishes this: value-based, off-policy algorithm
  - **Directly learns the optimal action-value function**

Q-learning

- **Recall: Bellman optimality equation**

$$Q_{opt}(s,a) = R(s,a) + \sum_{s'} \gamma T(s,a,s') V_{opt}(s'),$$

where $V_{opt}(S') = \max_{a \in Action(s)} Q_{opt}(s,a)$

- **Algorithm**: Q-learning changes the above on several aspects [Watkins/Dayan, 1992]
  - On each (s,a,r,s'):

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta\left(r + \gamma \hat{V}_{opt}(s')\right)$$

  - Recall: $\hat{V}_{opt}(s') = \max_{a' \in Actions(s')} \hat{Q}_{opt}(s',a')$ (note: take max based on current policy)

Remarks on Q-learning

- No expectation over s', just one sample s'

- Don't want to just replace $\hat{Q}_{opt}(s, a)$ with target value, but interpolate between old value (prediction) and new value (target)

- Replace actual reward with observed reward

- Replace $V_{opt}(s')$ with estimate $\hat{V}_{opt}(s')$

Off-policy vs. on-policy

- **On-policy**: The agent learns about the same policy it uses to interact with the environment
  - In SARSA: a_{t+1} is taken according to the current policy

- **Off-policy**: The agent learns about a different policy from the one generating the data
  - In Q-learning: updates uses the greedy target, learns $Q^{\star}$—the value of the optimal policy—even while exploring

on-policy	off-policy
SARSA, policy-gradient	Q-learning, deep Q-network

Lecture Plan

- Sequential decision-making and reinforcement learning
    - Model-based reinforcement learning
    - Model-free reinforcement learning
    - State-action-reward-state-action (SARSA)
    - Q-learning
    - **Epsilon-greedy**
    - Function approximation

Exploration

- Algorithm: reinforcement learning template
  - For $t = 1,2,3, \dots$
    - Choose action $a_t = \pi(s_{t-1})$
    - Receive reward r_t and observe new state s_t
    - Update parameters
- Which exploration policy π to use?

Epsilon-greedy

- Algorithm: epsilon-greedy policy

$$\pi(s) = \begin{cases} \arg\max_{a \in Actions} \hat{Q}_{opt}(s,a), & w.p. 1-\epsilon \\ \text{randomly from } Actions(s), & w.p. \ \epsilon \end{cases}$$

- It is natural to let ϵ decrease over time
  - When you're in a new environment, you want to explore a lot ($\epsilon = 1$), e.g., choose a new restaurant every time you eat outside
  - After a certain point, when you feel like you've seen all there is to see in this new environment, you start to exploit ($\epsilon = 0$), e.g., visit the same restaurant every time
- **Theoretical guarantees**: suppose there are K arms (or options to explore), ϵ-greedy finds the near-optimal policy in $O(\sqrt{KT})$ rounds

Lecture plan

- Sequential decision-making and reinforcement learning
  - Model-based reinforcement learning
  - Model-free reinforcement learning
  - State-action-reward-state-action (SARSA)
  - Q-learning
  - Epsilon-greedy
  - **Function approximation**

Generalization

- If we revisit the Q-learning algorithm, think about it through the lens of machine learning, you will find that we've just been memorizing Q-values for each (s, a), treating each pair independently
  - On each (s, a, r, s'):

$$\hat{Q}_{opt}(s,a) \leftarrow (1-\eta)\hat{Q}_{opt}(s,a) + \eta\left(r + \gamma\hat{V}_{opt}(s')\right)$$

- What about more general scenarios? Can you learn to memorize the behaviors?

Surrogate function approximation

- Function approximation fixes this by parameterizing $\hat{Q}_{opt}$ by a weight vector and a feature vector, as we did in linear regression

- **Key idea: linear regression model**
  - Define features $\phi(s, a)$ and weights w

$$\hat{Q}_{opt}(s, a; w) = w \cdot \phi(s, a)$$

Function approximation

- Algorithm: Q-learning with function approximation
  - On each (s, a, r, s'):

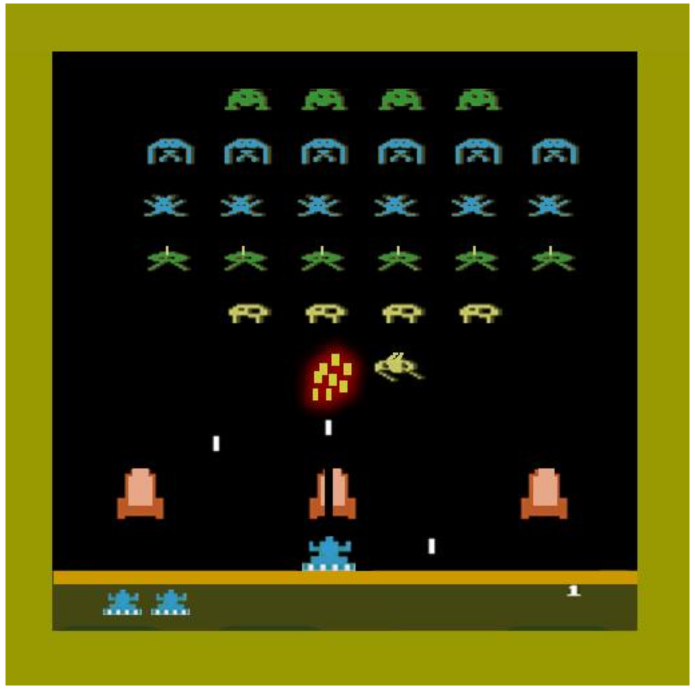
$$w \leftarrow w - \eta[\hat{Q}_{opt}(s, a; w) - \left(r + \gamma \hat{V}_{opt}(s')\right)]\phi(s, a)$$

- Implied objective function:

$$\left(\hat{Q}_{opt}(s, a; w) - \left(r + \gamma \hat{V}_{opt}(s')\right)\right)^2$$

Extension: Deep reinforcement learning

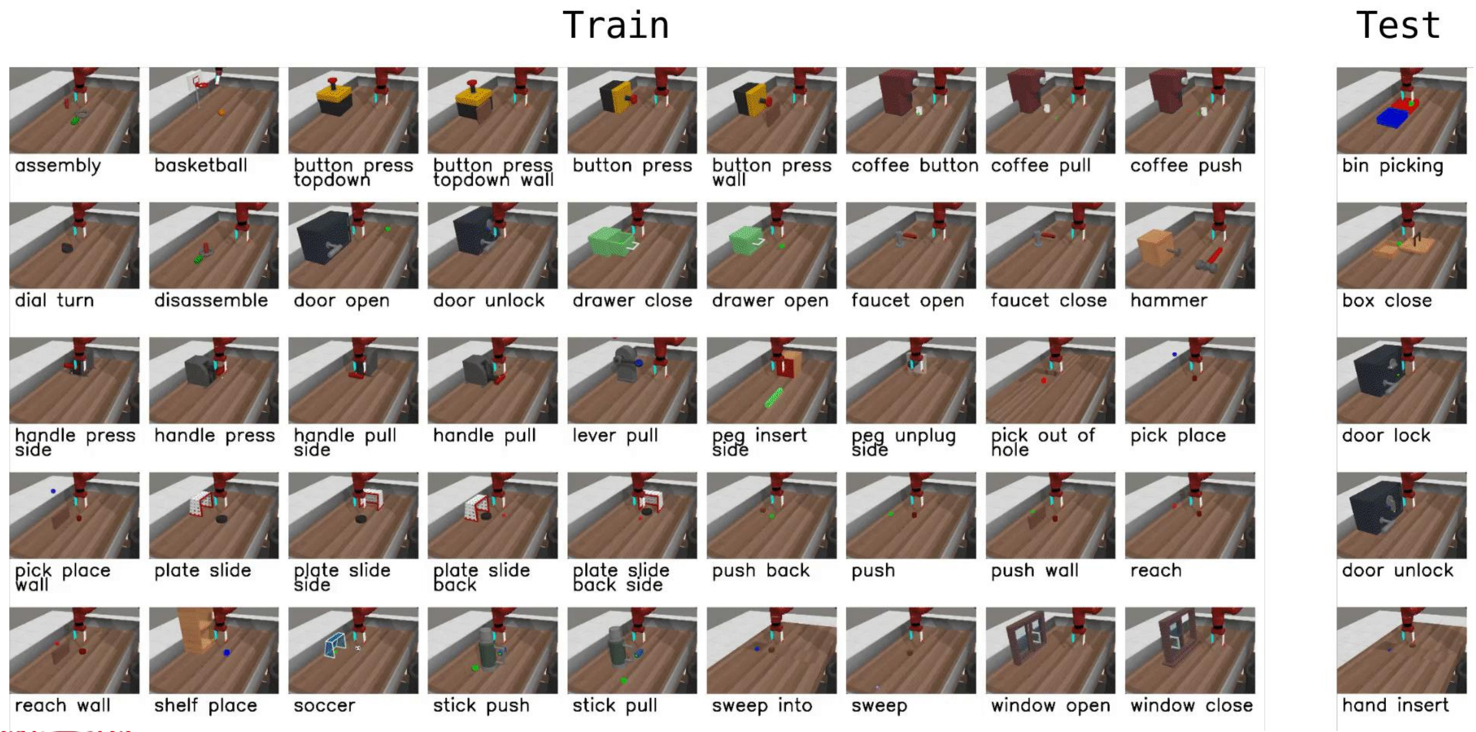
- Use a neural network to represent $\hat{Q}_{opt}(s, a), \pi_{opt}, T$
- Atari game [DeepMind, 2013]

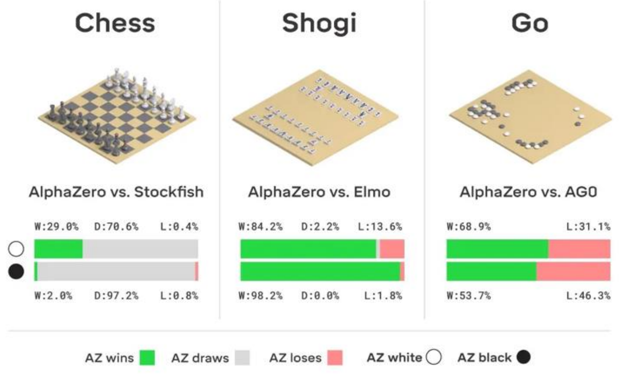


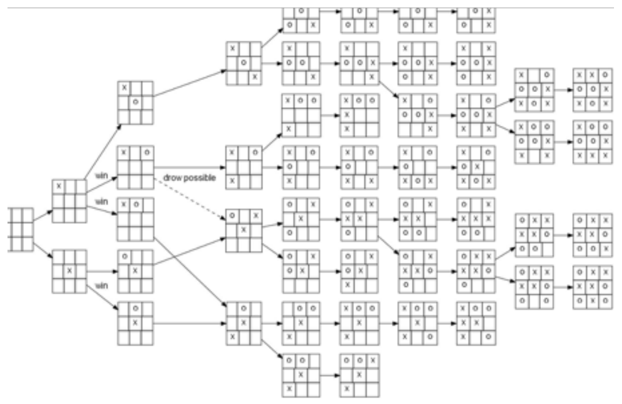
  - Treat last 4 frames as images, and then train a CNN to predict the keystrokes
  - Apply the ϵ-greedy, train over 10 million frames with 1 million replay memory
  - This algorithm achieves human-level performance on some games (breakout), but works not so good on other games (space invaders)

Extension: Deep reinforcement learning

- **Policy gradient**: train a policy $\pi(a \mid s)$ using a neural network-based policy map to directly maximize the expected reward
  - Robotic manipulation: pick object, navigate room
  - DeepMind's AlphaGo (2016), AlphaZero (2017)







Summary

- In Markov decision processes (MDPs), we learn to cope with uncertainty in a stochastic system (traffic control, robotics, online games)
- Solutions are **policies** rather than paths
- **Policy evaluation** computes policy value (expected utility)
- **Value iteration** computes optimal value (maximum expected utility) and optimal policy

Summary (continued)

- Online setting: learn and take actions in the real world
- **Monte Carlo estimation**: estimate transitions, rewards, Q-functions from observational data
  - Several algorithms: model-based value iteration, model-free Monte Carlo, SARSA, and Q-learning
  - Update towards target that depends on estimate rather than just raw data
- **Exploration vs. exploitation**: address the key challenge of partial supervision / feedback