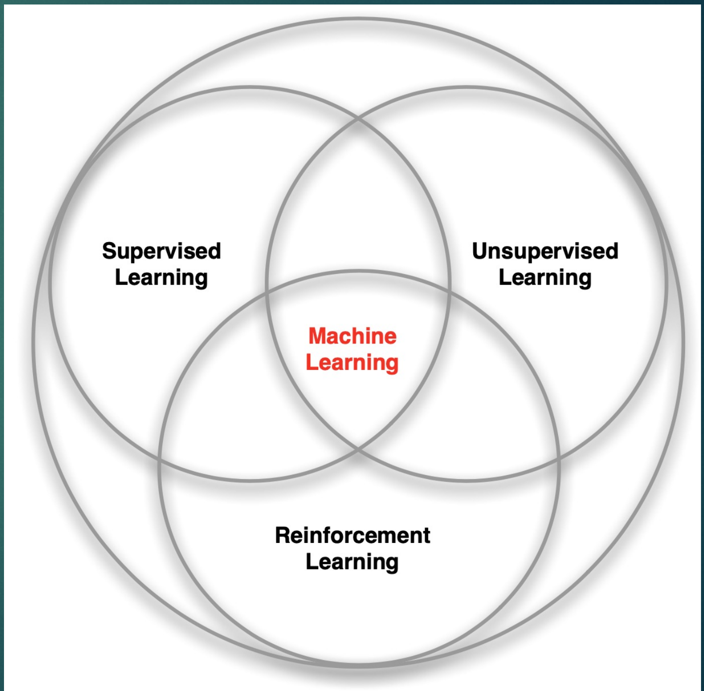
Introduction to Reinforcement Learning

CSCI 8360 DATA SCIENCE PRACTICUM

SPRING 2021

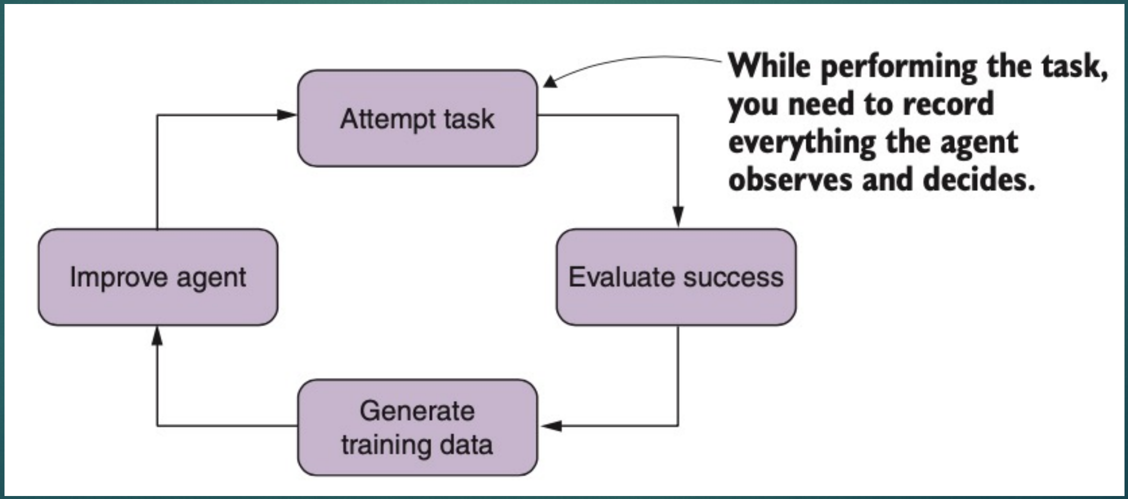
Machine Learning

- Supervised learning
  - Learn a function *f* such that it maps input X to labels Y
  - *f(X) -> Y*
- Unsupervised learning
  - Finding patterns in data without labels
  - Clustering, compression, dimensionality reduction
- Reinforcement learning
  - Sequential decision-making
  - Combines aspects of supervised and unsupervised learning



Reinforcement Learning (RL)

- Key point: learns through **trial-and-error** interaction with the surrounding environment



Jargon

- Agent
  - The “thing” that’s learning the optimal behavior through trial-and-error interaction with the surrounding environment
- Environment
  - The explicit limiting circumstances (spatial, temporal, interactive) in which your agent can freely probe in order to learn
- States
  - Information the agent uses to determine what to do next
  - A function of history, where history is the sequence of observations, actions, and rewards up to the current time *t*
- Actions
  - Possible decisions an agent can make at step *t*
  - Actions will influence reward
- Reward
  - A scalar feedback signal to the agent indicating how well it’s doing at step *t*
  - Or “return” on a policy
- Policy
  - Defines how an agent selects actions to perform
  - Deterministic (direct mapping from action to state) or stochastic (probabilistic mapping)

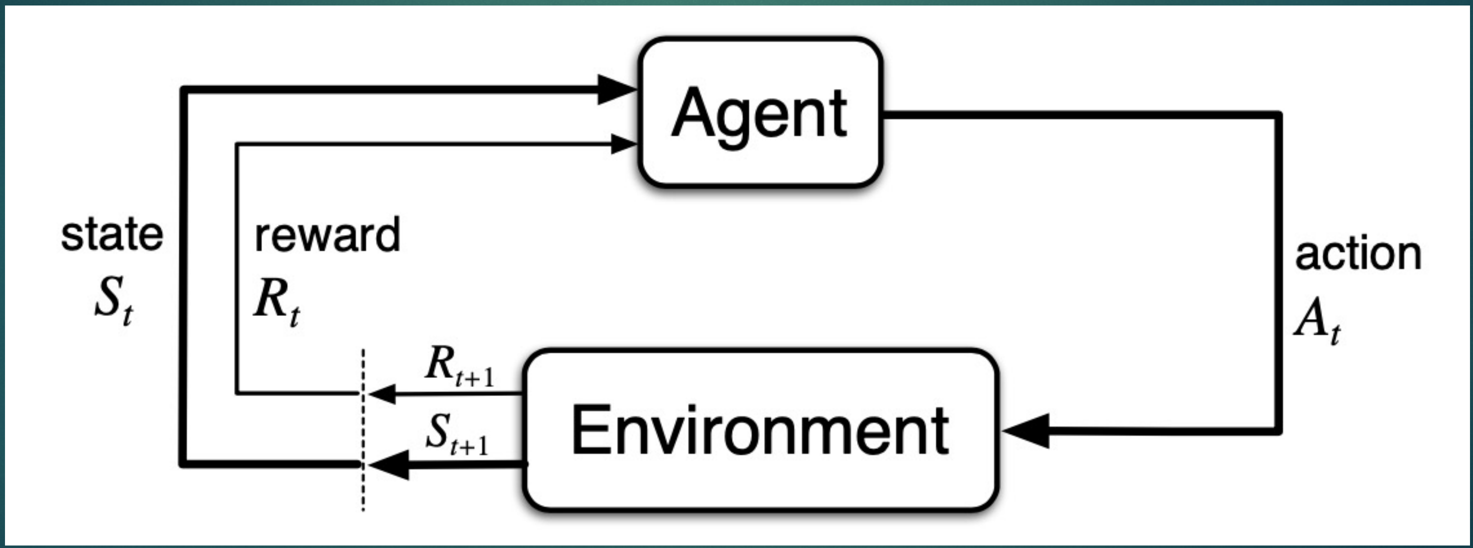
Goal of RL

- Select actions to maximize total future reward
  - (can we know the future reward?)

- Actions may have long-term consequences
- Reward may be delayed
- May be better to sacrifice immediate reward to gain more long-term reward
  - Financial investments
  - Blocking opponent moves

Goal of RL

- The agent-environment interaction in reinforcement learning



Examples of Reward

- Flying a drone
  - + reward for following desired trajectory
  - - reward for crashing
- Defeat world champion at chess
  - + reward for winning a game
  - - reward for losing a game
- Control a power station
  - + reward for producing power
  - - reward for exceeding safety thresholds
- Play different Atari video games
  - + reward for increasing score
  - - reward for lower scores

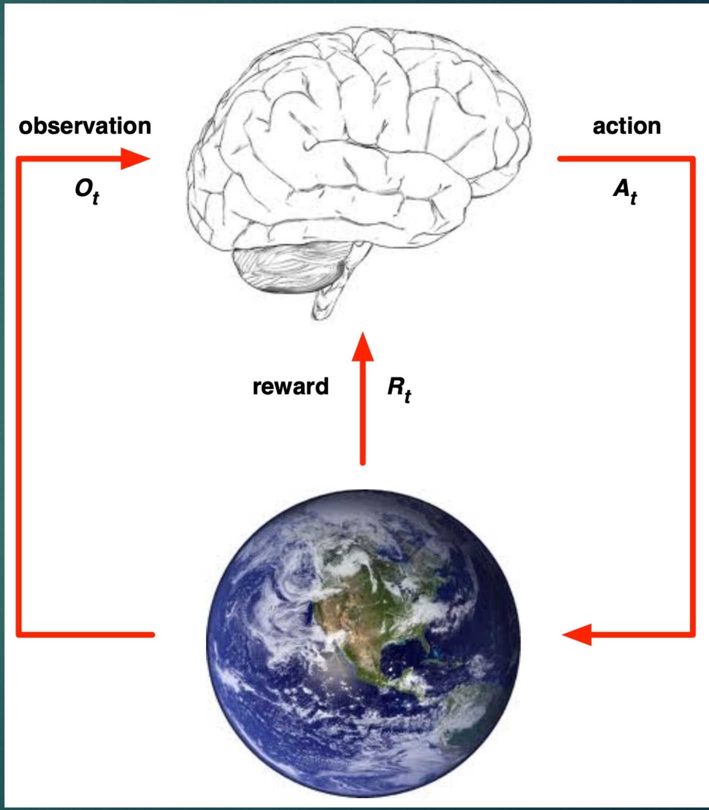
These can be VERY long-term rewards!

Dependency between game outcome and any individual move is ambiguous.

Agent and Environment

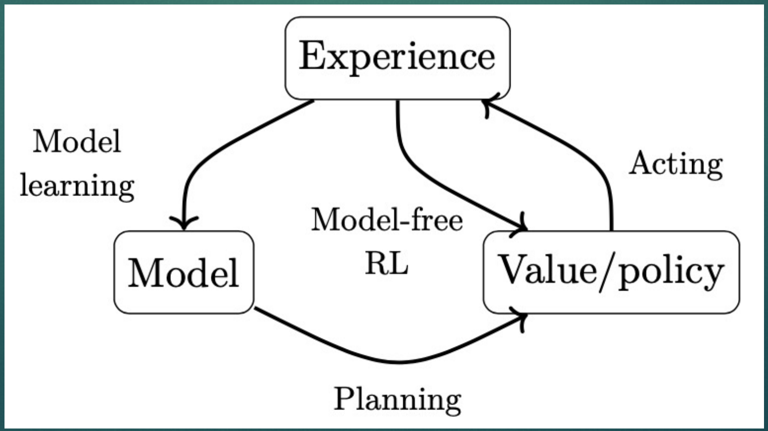
- Select actions to maximize total future reward

- Actions may have long-term consequences
- Reward may be delayed
- May be better to sacrifice immediate reward to gain more long-term reward
  - Financial investments
  - Blocking opponent moves



Model-based or model-free

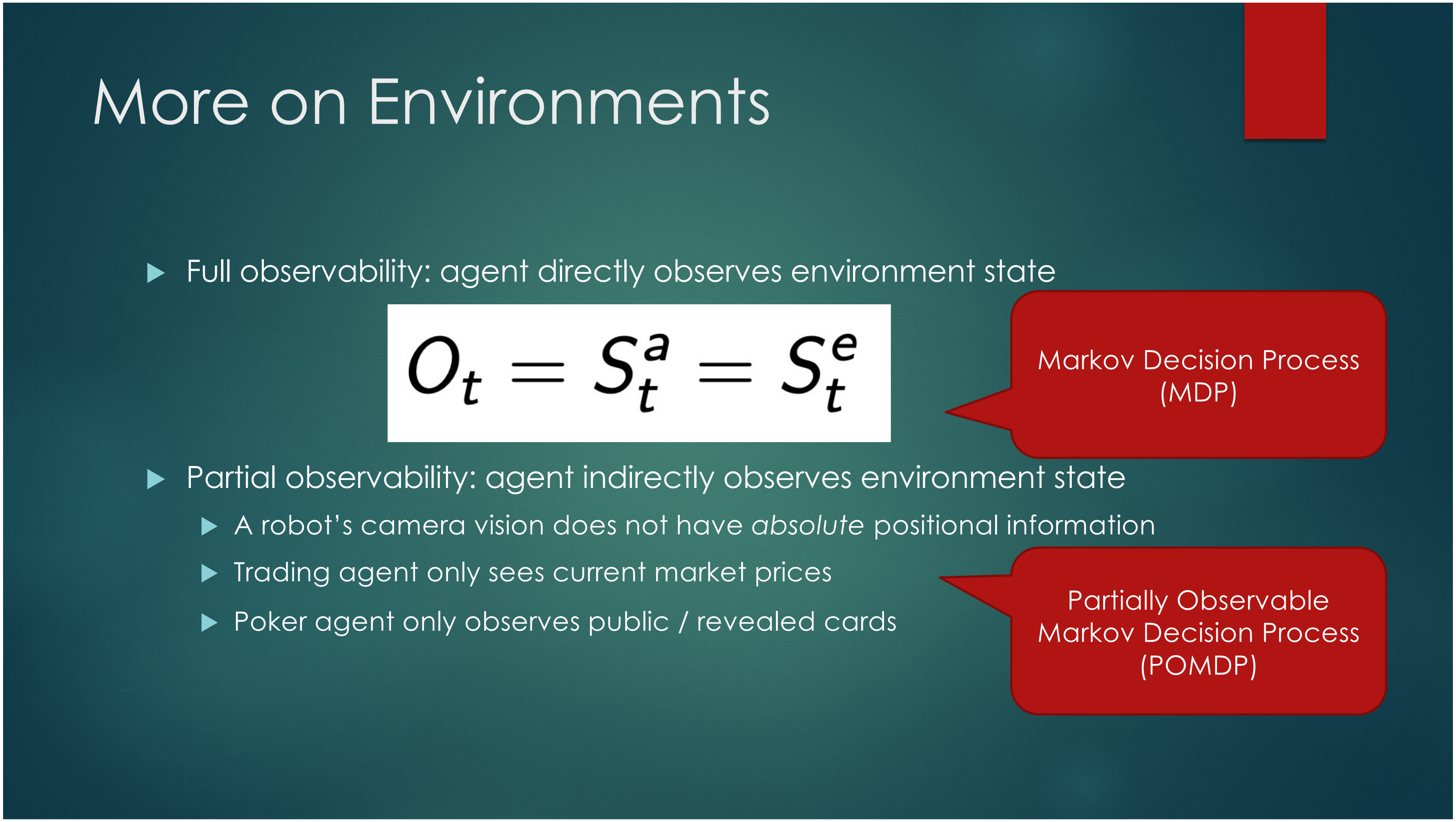
- The agent may have an internal model of the environment
  - Contains estimated transition function and estimated reward function
  - Combines these with a planning algorithm
- Model-free agents optimize the reward function directly



Types of Environments

- Deterministic, non-deterministic
- Fully-known, hidden (or partially hidden)

	Deterministic	Nondeterministic
Perfect information	Go, chess	Backgammon
Hidden information	Battleship, Stratego	Poker, Scrabble


More on Environments
Full observability: agent directly observes environment state
$O_t = S_t^a = S_t^e$
Markov Decision Process (MDP)
Partial observability: agent indirectly observes environment state
A robot's camera vision does not have *absolute* positional information
Trading agent only sees current market prices
Poker agent only observes public / revealed cards
Partially Observable Markov Decision Process (POMDP)

More on Environments

- **In POMDPs, agent must construct its own state representation S_t^a**
- Complete history $S_t^a = H_t$
- *Beliefs* of environment state $S_t^a = (\mathbb{P}[S_t^e = s^1], ..., \mathbb{P}[S_t^e = s^n])$
- Recurrent neural network $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_o)$

Value Function

- *Prediction* of future reward by the agent
- Used to evaluate the goodness/badness of states, and therefore to select between actions

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$

- π: current policy
- s: agent's state
- γ: discount factor (why?)

Value vs Reward

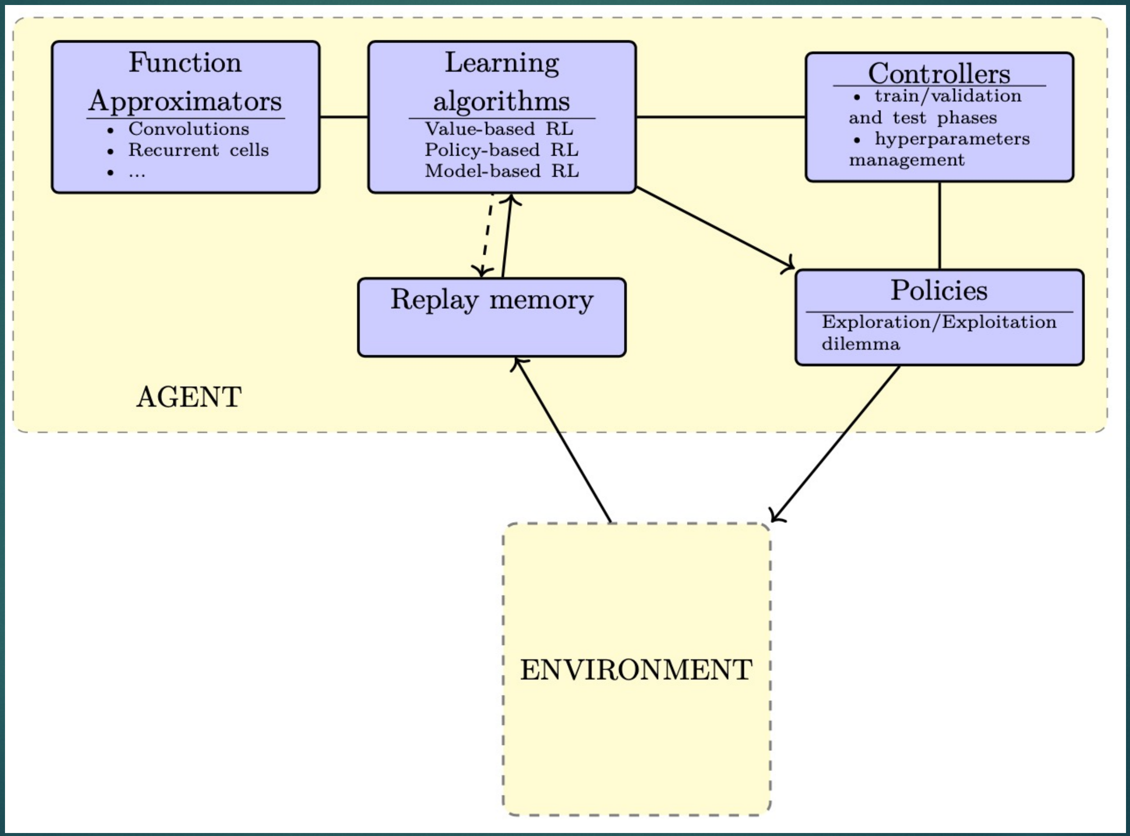
Value

- Prediction
- Evaluated in advance
- Derived by the agent using the information it has on hand

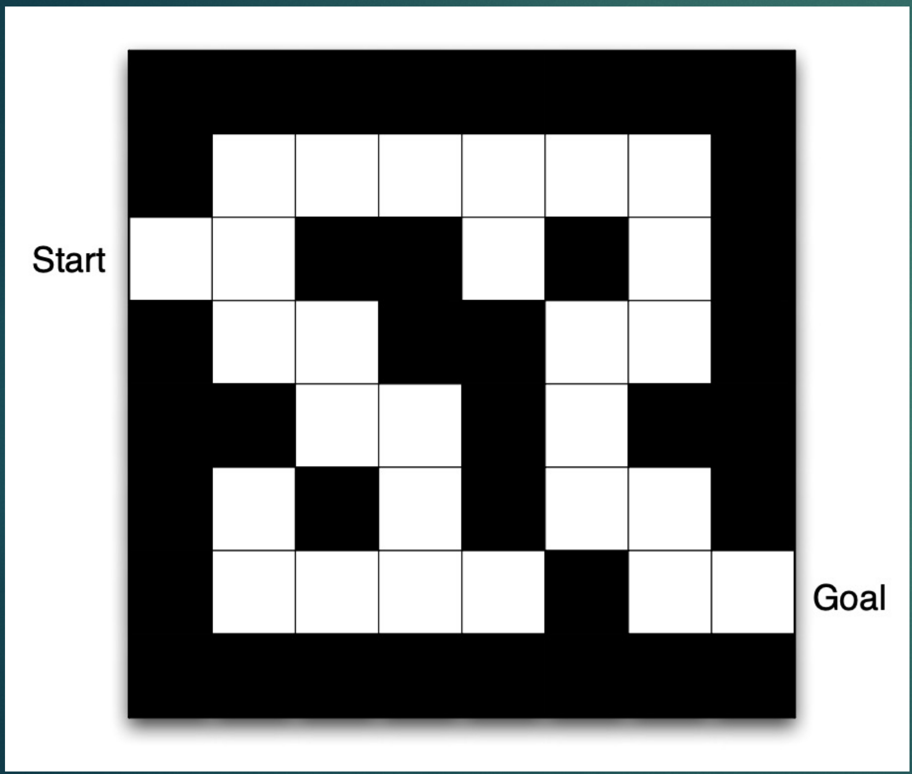
Reward

- Ground truth
- Provided after the fact
- Given by the environment once the ultimate consequences of the agent's actions are known

General RL schema

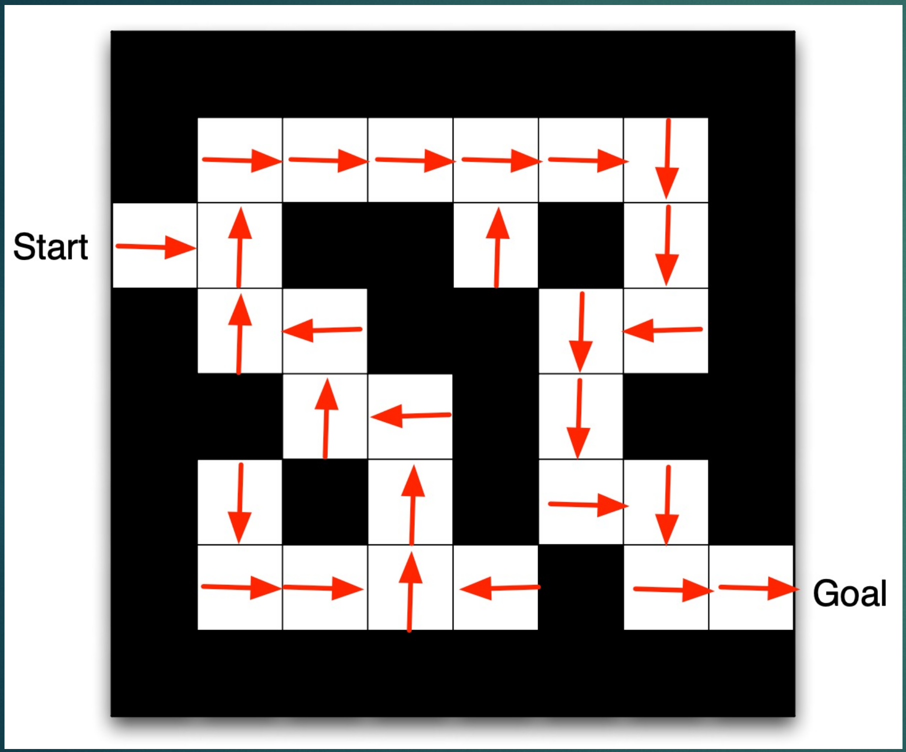


Maze Example



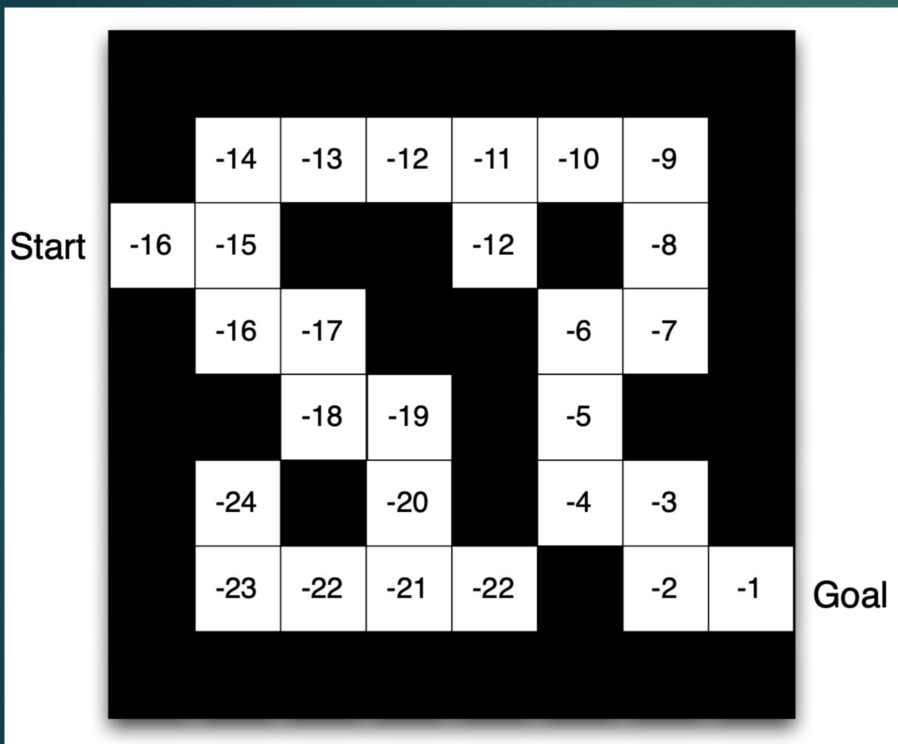
- Rewards: -1 per step
- Actions: Up, Down, Left, Right
- States: agent's location

Maze Example



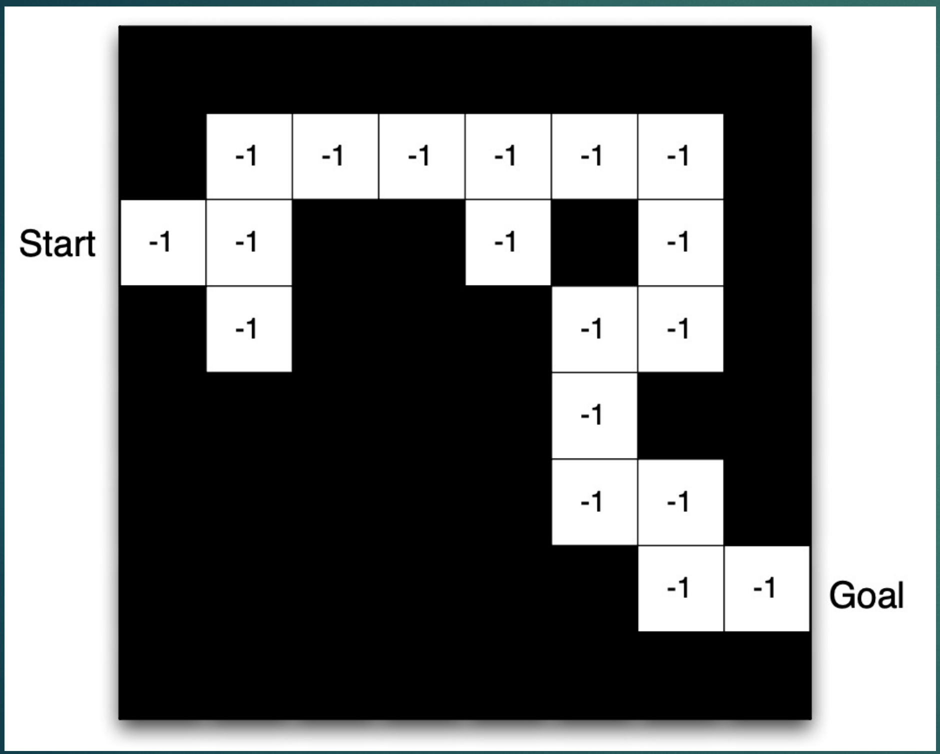
- Arrows represent policy $\pi(s)$ for each state s

Maze Example



- Numbers represent value function $v_\pi(s)$ of each state s

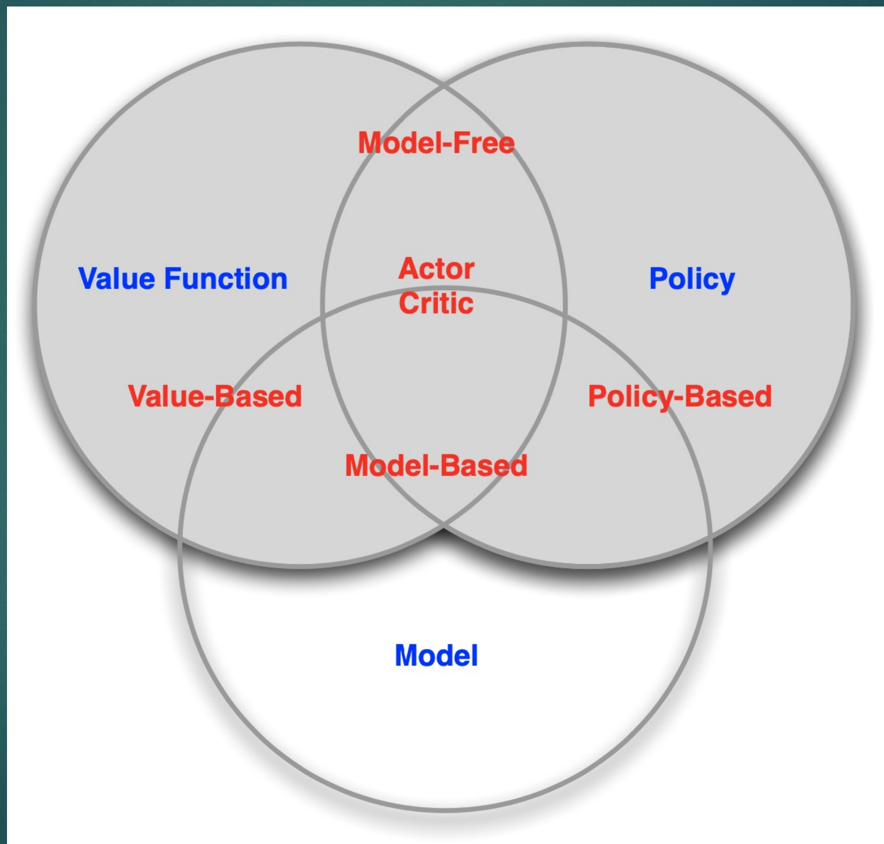
Maze Example



- Agent may have an internal model of the environment
- Dynamics: how actions change state
- Rewards: how much reward from each state
- Model may be imperfect!

- Grid layout represents transition model
- Numbers represent immediate reward from each state

RL agent taxonomy



Sequential decision making

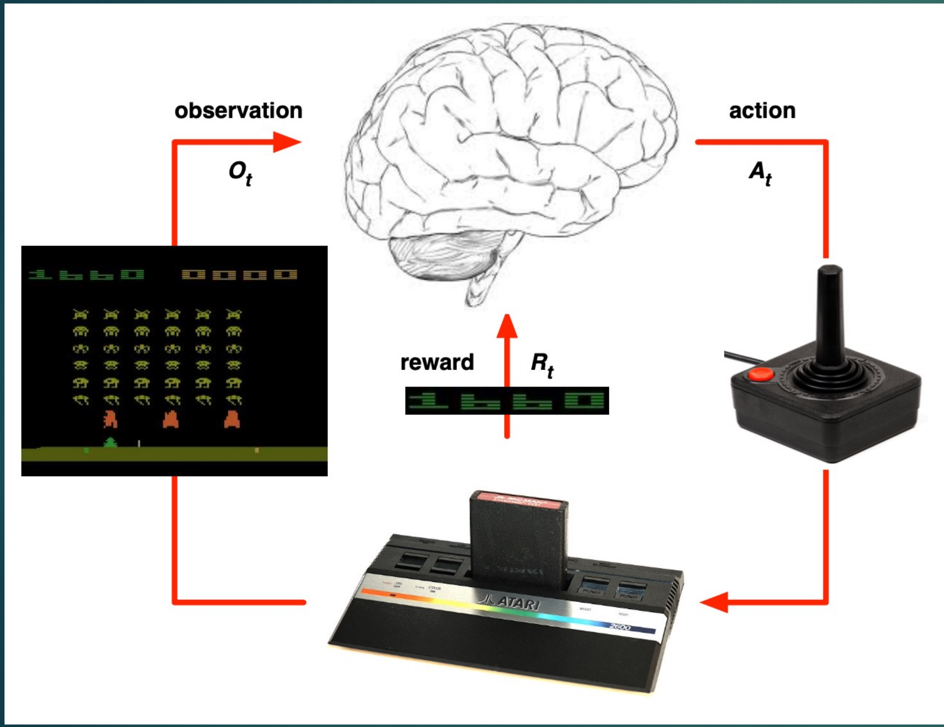
Reinforcement Learning

- Environment initially unknown
- Agent interacts with environment
- Agent improves its policy

Planning

- Model of environment is known (albeit possibly imperfect)
- Agent performs computations with its model (no external interactions)
- Agent improves its policy
- Deliberation, reasoning, introspection, pondering, thought, search, etc

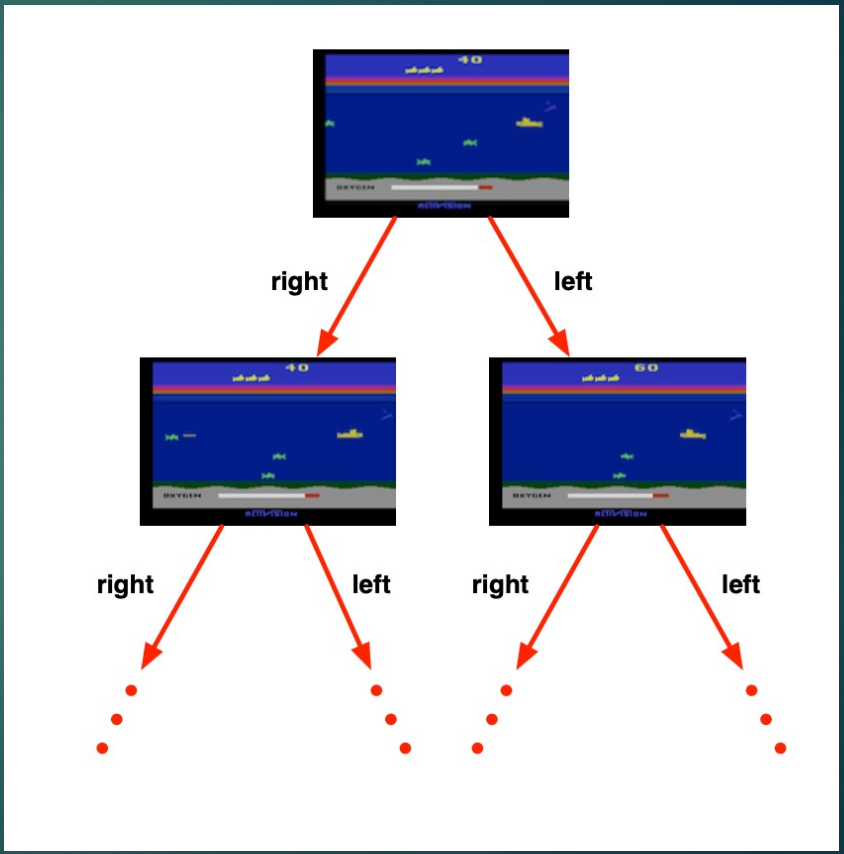
Atari Example: RL



- Rules of the game could be initially unknown
- Learn rules directly from interactive game-play
- Pick actions on joystick, see pixels and scores

Atari Example: Planning

- Rules of the game are known
- Can query the emulator
  - In this case, a perfect model inside agent's brain
- If I take action *a* from state *s*:
  - What would the next state be?
  - What would the score be?
- Plan ahead to find optimal policy
  - E.g. tree search



Two key strategies

Exploration

- Finds more information about the environment
- Try a new restaurant
- Show a new ad
- Play an experimental move

Exploitation

- Exploits known information to maximize reward
- Go to your favorite restaurant
- Show most successful ad
- Play the move you know works best

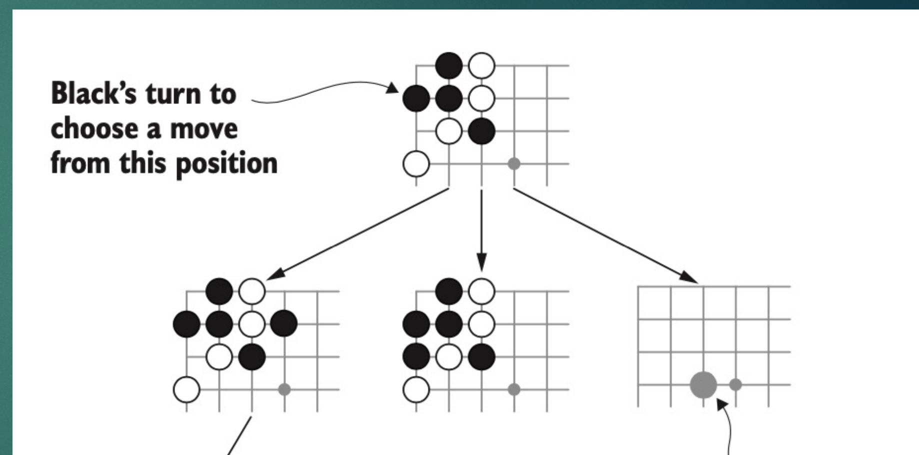
It is usually very important to do both, but one often comes at the expense of the other

Exploration and Exploitation

- RL is like trial-and-error learning
- Agent should discover a good policy via its experiences interacting with the environment
- …without losing too much reward along the way

Games for RL

- Why games?
- [often] simple rules, but deep concepts
- Very large observation and/or action spaces
- Long planning horizons / sparse rewards
- **Fun!**



Core approach: minimax

- You have two players with two distinct policies
- Assume each player enacts their optimal policy (i.e., they're good players)
- Players adapt to each other
- Therefore, players have opposite rewards (what is good for one player must, therefore, be bad for the other): **zero-sum rewards**

- Sound at all familiar?

Minimax

- Nash equilibria $R^1 + R^2 = 0$
- **We see this in GANs!**
- Value function now defines expected total reward given *joint* policies $\pi = \langle \pi^1, \pi^2 \rangle$

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- Redefine a player's value function as a minimax value function

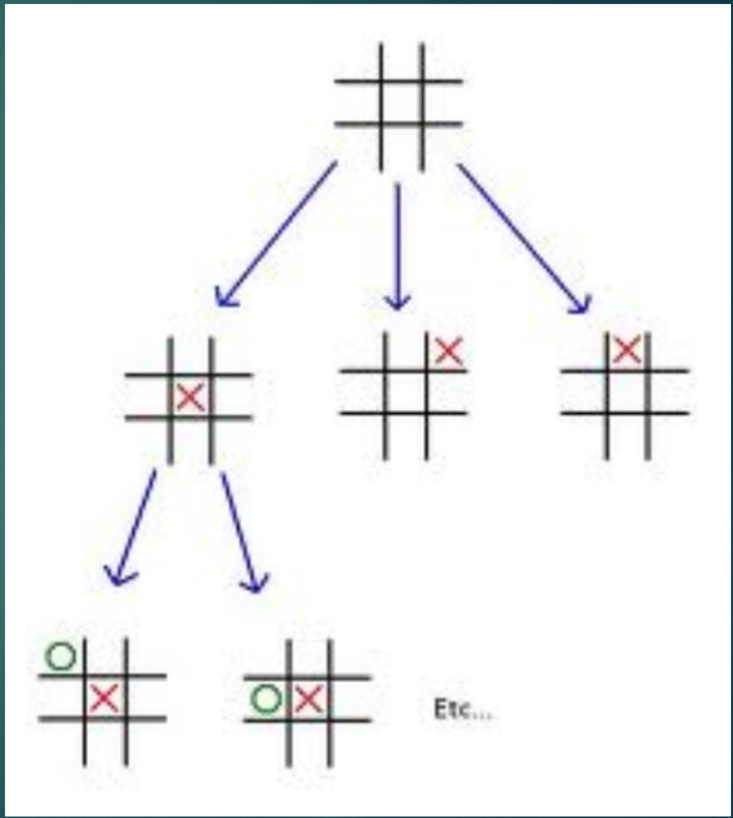
$$v_*(s) = \max_{\pi^1} \min_{\pi^2} v_\pi(s)$$

Subscripts will be swapped depending on which player's value function we're referring to

Minimax

- How to find the minimax?
- Depth-first game-tree search

- **Done! …right?**

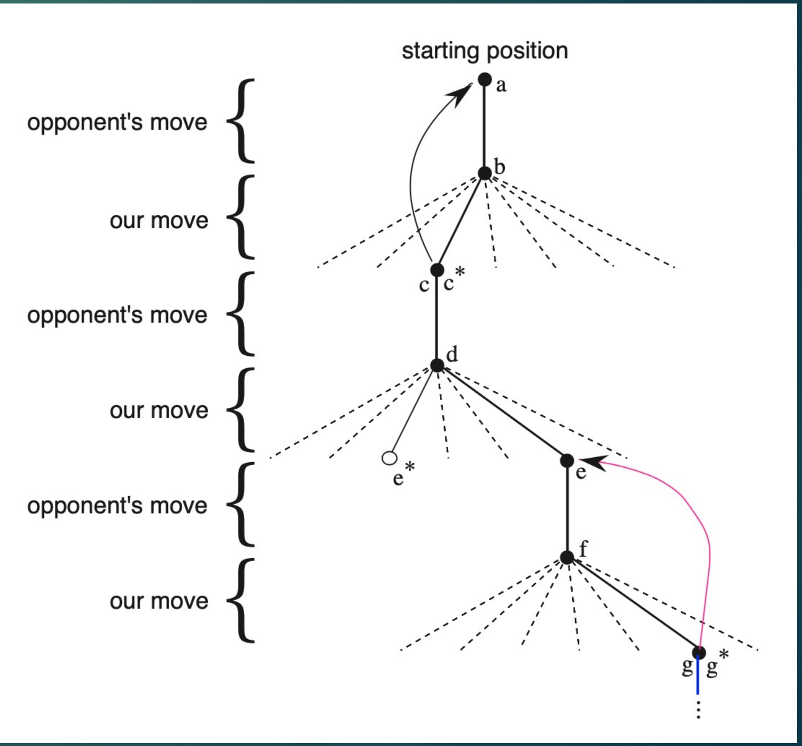


Minimax

- Search tree grows exponentially
- Impractical to search to the end of the game
- Instead, use value approximator

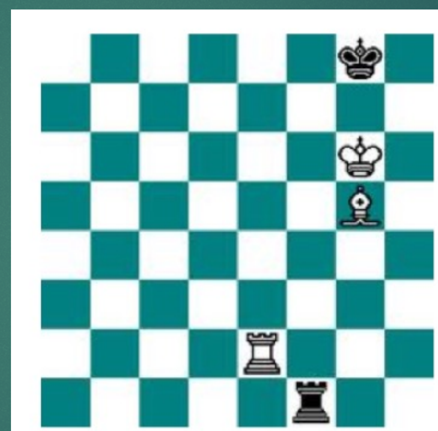
$$v(s, \mathbf{w}) \approx v_*(s)$$

- Minimax search to fixed depth
- *Estimate* minimax value at leaf nodes



Minimax Example: Chess

- Binary-linear value function $v(s, w)$
  - Binary feature vector $x(s)$: one feature per chess piece
  - Weight vector w: value of each chess piece
  - Position is evaluated by summing weights of current features



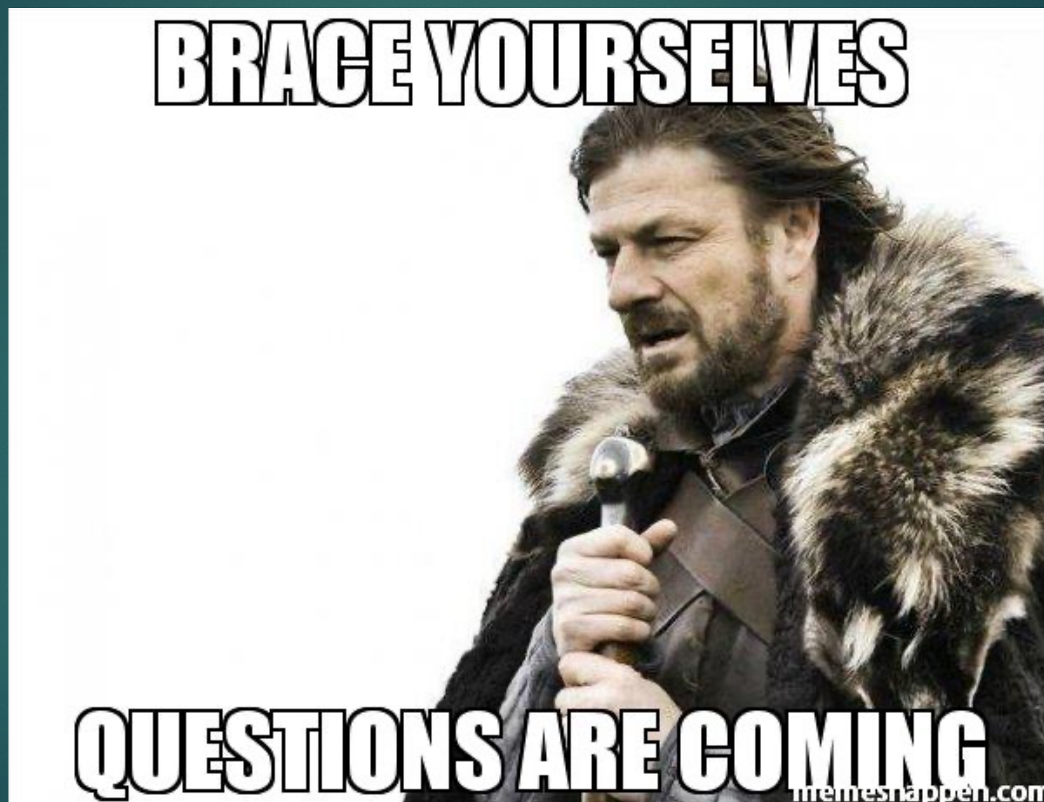
$$v(s, \mathbf{w}) = \mathbf{x}(s) \cdot \mathbf{w} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix} \cdot \begin{array}{cc} \begin{bmatrix} +5 \\ +3 \\ +1 \\ -5 \\ -3 \\ -1 \\ \vdots \end{bmatrix} & \begin{array}{c} ♖ \\ ♗ \\ ♙ \\ ♜ \\ ♝ \\ ♟ \\ \end{array} \end{array}$$

$$v(s, \mathbf{w}) = 5 + 3 - 5 = 3$$

Next week

- More RL game examples
- Rule-based agents and MDPs
- Introduction to Deep RL
- DeepMind's RL framework

Questions?



References

- DeepMind’s intro to RL videos and slides
  - https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver
- *Deep Learning and the Game of Go*
  - https://www.manning.com/books/deep-learning-and-the-game-of-go
  - Code examples: https://github.com/maxpumperla/deep_learning_and_the_game_of_go
- *Reinforcement Learning: An Introduction*
  - Richard Sutton and Andrew Barto (2nd ed), https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- “An Introduction to Deep Reinforcement Learning”
  - https://arxiv.org/abs/1811.12560