Reinforcement Learning

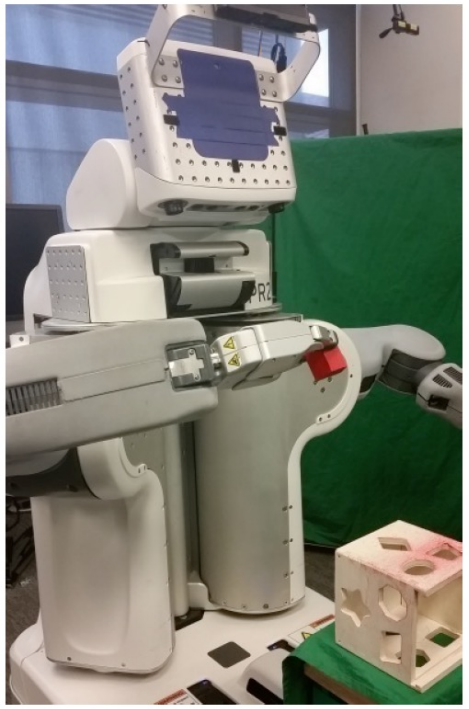
Forest Agostinelli
University of South Carolina

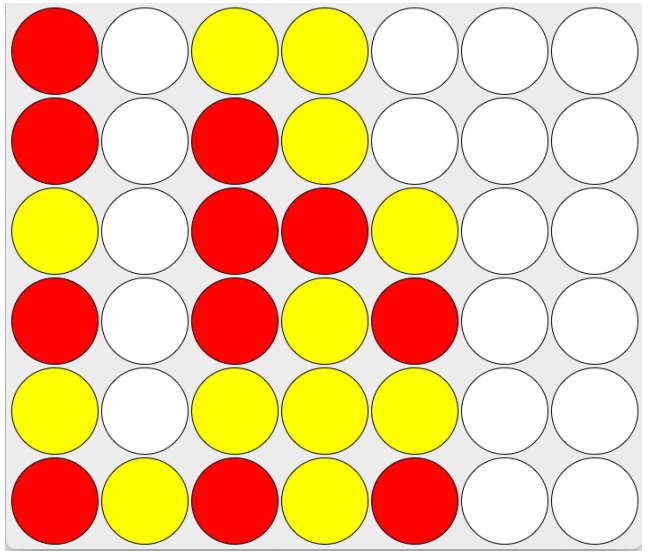
Outline

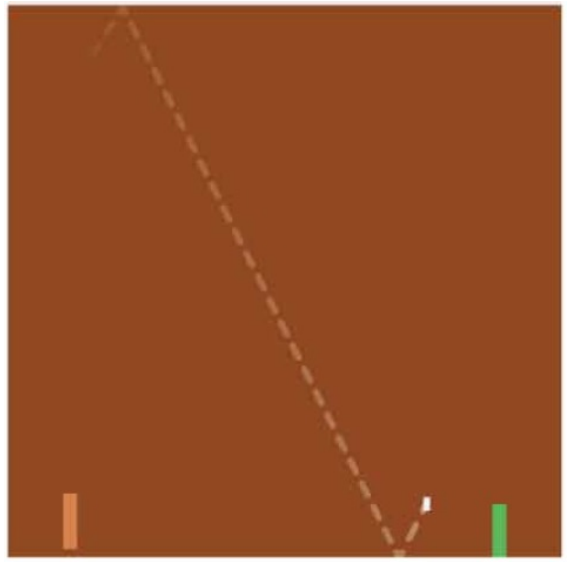
- Background
- Value functions
- Dynamic programming
- Model-free reinforcement learning

Reinforcement Learning

- Learning to maximize reward in sequential decision-making problems
- Learning is done through experience
- Decisions affect experience and experience affects decisions
- There may be stochasticity
  - Inherent randomness in the outcome of actions
  - The probabilities associated with this randomness may be unknown

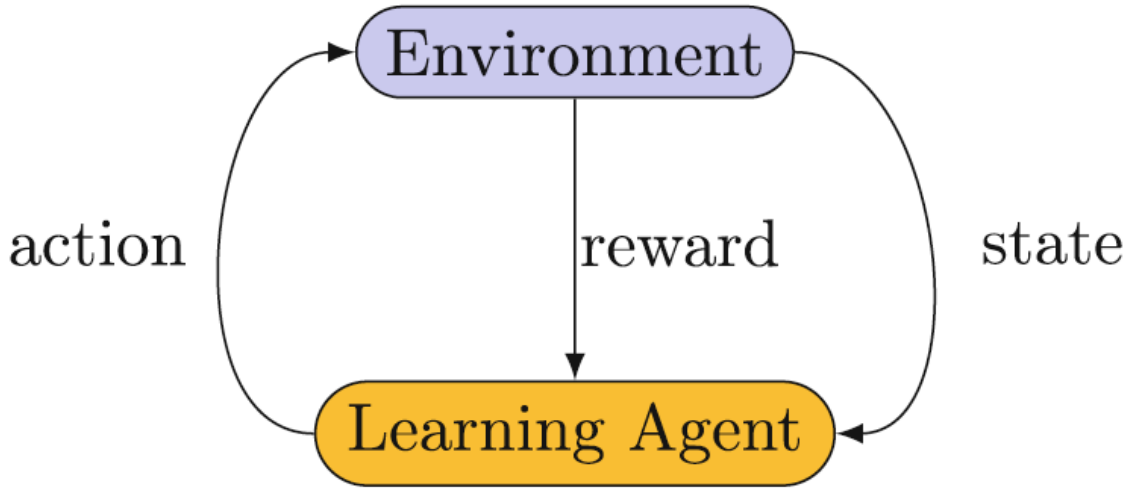






Reinforcement Learning

- **Reinforcement learning**: learning to map states to actions so that we maximize the expected future reward we receive from the environment.
- This mapping of states to actions is called a policy function.
  - Deterministic: $a = \pi(s)$
  - Stochastic: $\pi(a|s) = P(A = a|S = s)$
- At each time step t
  - In state S_t, agent takes action A_t
  - Based on state s_t and action a_t, the environment transitions to state S_{t+1} and outputs reward R_{t+1}



Agostinelli, Forest, et al. "From reinforcement learning to deep reinforcement learning: An overview." Braverman Readings in Machine Learning. Key Ideas From Inception to Current State. Springer, Cham, 2018. 298-328.

Markov Decision Processes (MDPs)

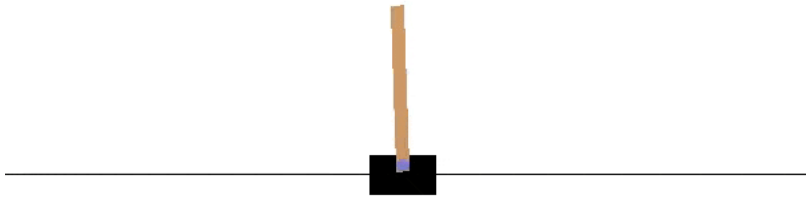
- To formally pose a problem as a reinforcement learning problem, we must define it as a Markov decision process
- **States**
  - The possible configurations of problem
- **Actions**
  - The actions that the agent can take
- **Transition Probabilities**
  - Given state, s, and action, a, this defines
    - The probability of transitioning to state, s’
    - The probability of obtaining reward, r

The Markov Property

- In reinforcement learning, it is often assumed that the Markov property holds
- The Markov property states
  - Given the current state and action taken, we do not need any of the history to define the transition probabilities
  - The history consists of previous states, actions, and rewards
- In other words
  - The next state and reward is independent of the history given the current state and action

Markov Property Example

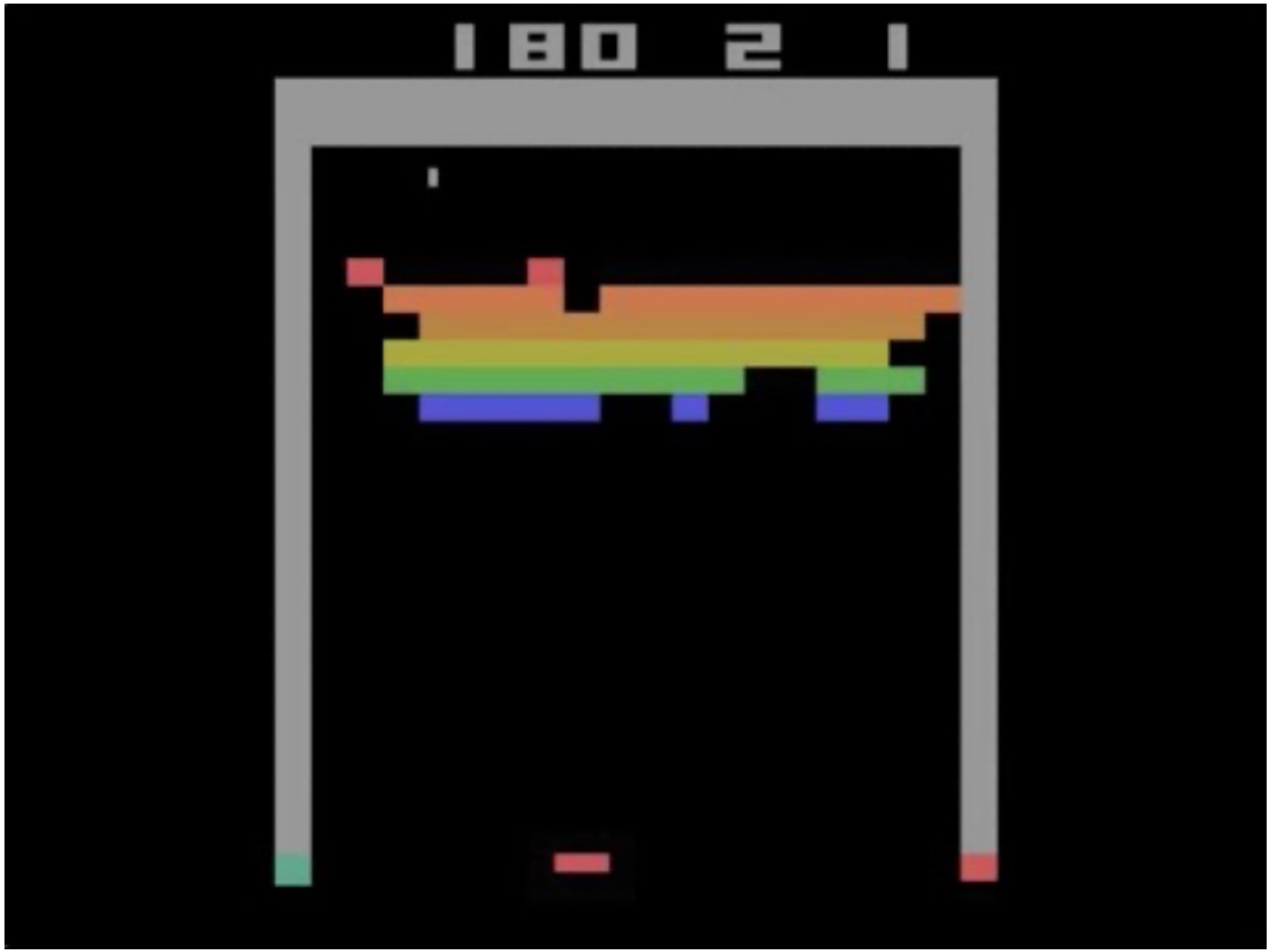
- **Cartpole**
- Actions: apply force left, apply force right
- Rewards: +1 for every step
- Episode ends when the pole falls over



- The Markov property does **not** hold if we represent the state as the position of the cart and pole
  - Why?
- To ensure the Markov property holds, we must include the velocity of the cart and pole

RL Successes: Atari

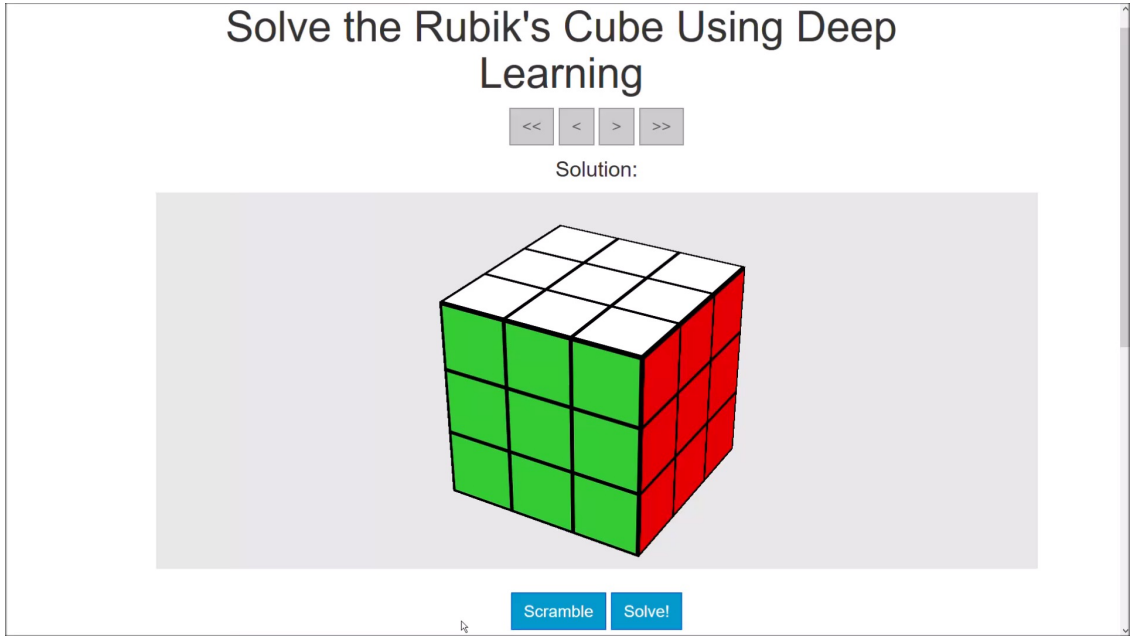
- Using RL algorithms, a deep neural network is trained to play Atari games using raw pixels.
- Current work shows we can train deep neural networks to play better than humans on 57 different Atari games.
- RL was combined with **deep neural networks**

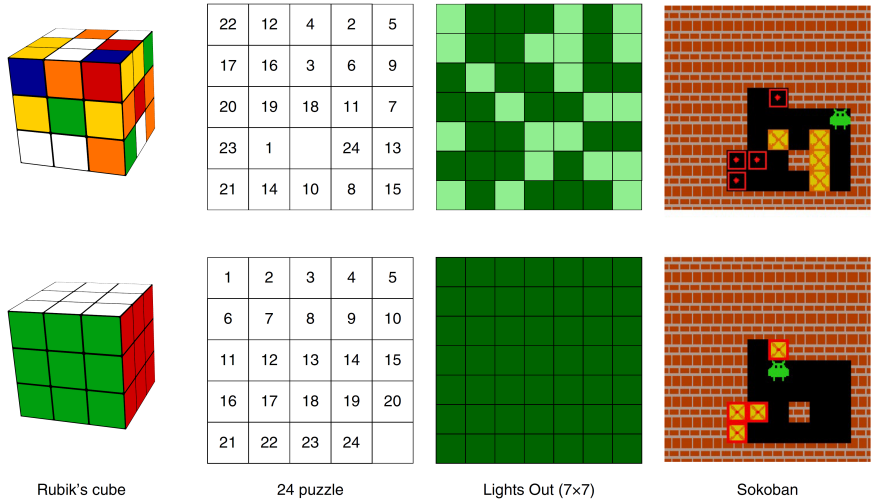


Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

RL Successes : Solving the Rubik's Cube

- 4.3 x 10^{19} possible combinations
- No domain-specific knowledge
- DeepCubeA solves the Rubik's cube and other puzzles
  - Puzzles have up to 3.0 x 10^{62} possible combinations.
- Finds a shortest path in the majority of verifiable cases
- http://deepcube.igb.uci.edu/
- RL was combined with **deep neural networks** and **A* search**

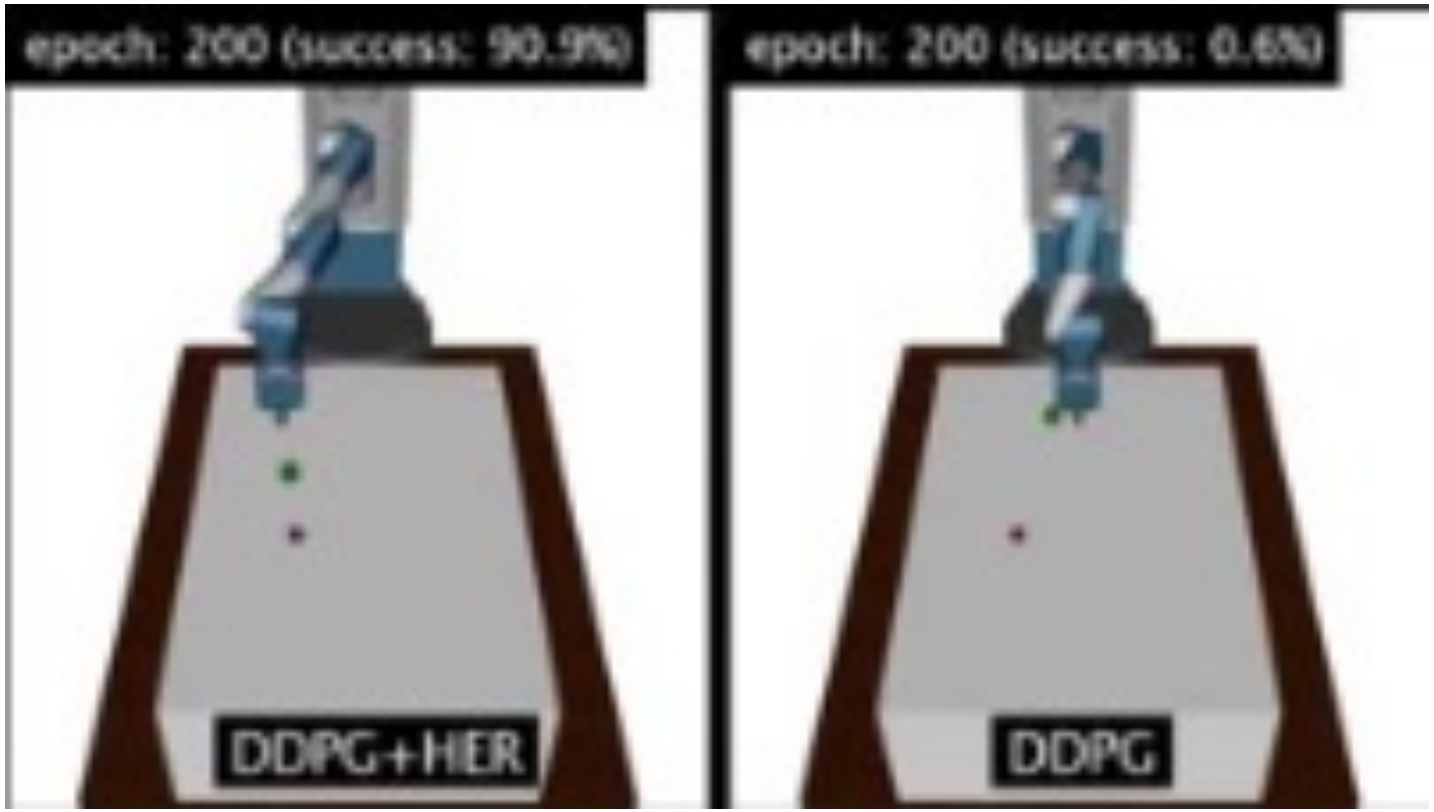




Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.

RL Successes: Robotics

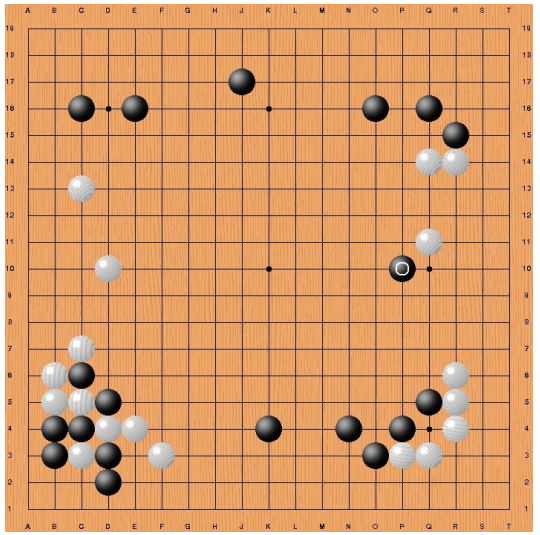
- Can solve problems in continuous environments
- Can train in simulation and transfer to the real world (Sim2Real)
- RL was combined with **deep neural networks**



Andrychowicz, Marcin, et al. "Hindsight experience replay." *Advances in neural information processing systems* 30 (2017).

RL Successes: Go

- AlphaGo learned how to play Go from expert demonstrations data and from self-play
  - Defeated one of the best Go players, Lee Sedol, 4 to 1
  - Move 37
- AlphaGoZero builds on AlphaGo and learns only from self-play
- RL was combined with **deep neural networks** and **Monte Carlo tree search**



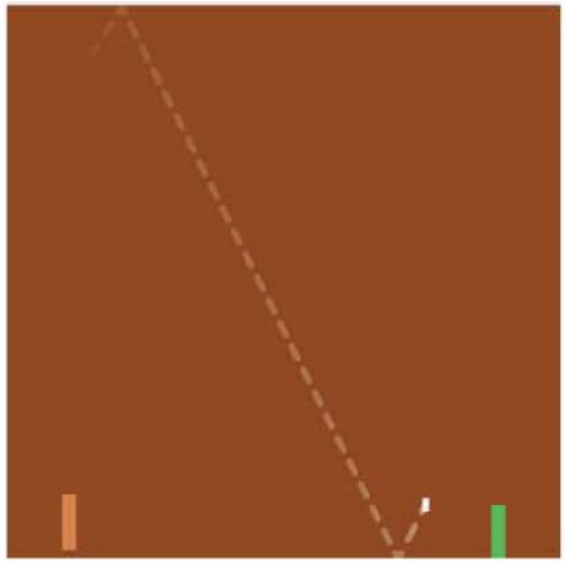
Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

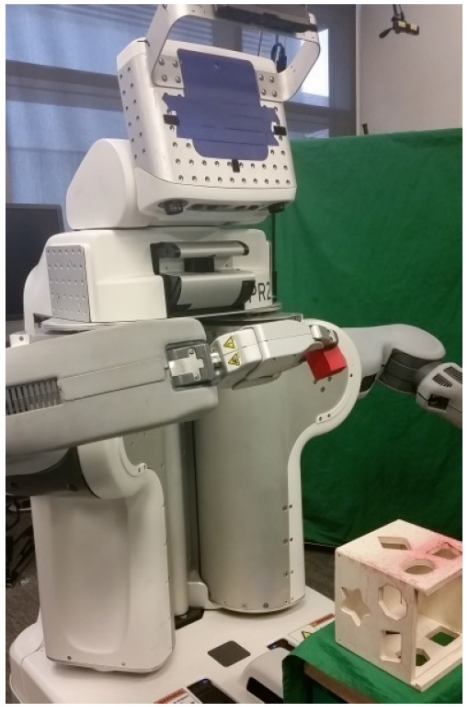
Imitation Learning

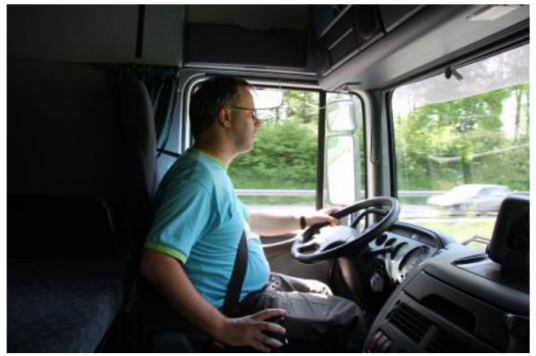
- If we know how to solve the problem efficiently (or at all!), we can use imitation learning to train an agent to imitate the actions we want it to take
- Collect a dataset of states and the actions humans took to solve the problem
  - Input: states
  - Output: actions
  - Objective function: accurately predict actions given states
- We can now pose the problem as a supervised learning problem
  - We already have a plethora of tools to solve these problems

Reinforcement Learning: Motivation

- Collecting data from humans may be very time-consuming
- Distribution shift may be an issue



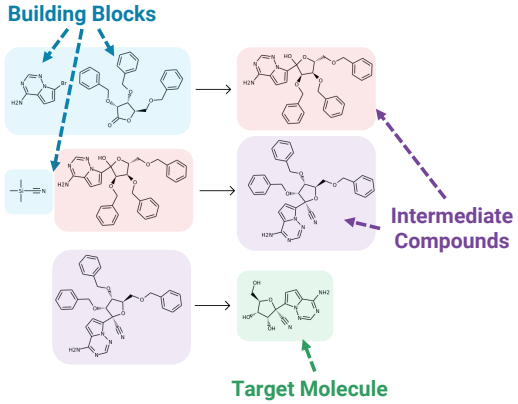


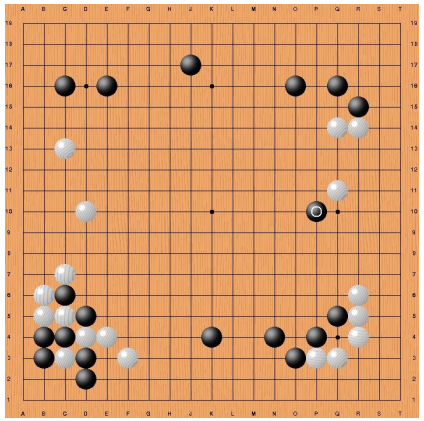


Fawzi, Alhussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning." Nature 610.7930 (2022): 47-53.

Reinforcement Learning: Motivation

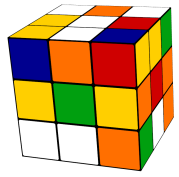
- Humans may not know how to solve the problem efficiently, or at all!



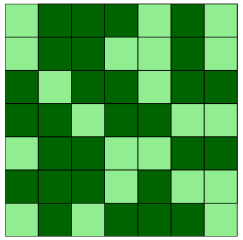


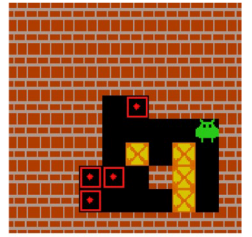
Input: $n \times n$ skew-symmetric matrix **A**, vector **b**.
Output: The resulting vector **c** = **Ab** computed in $\frac{(n-1)(n+2)}{2}$ multiplications.

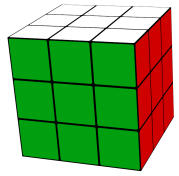
(1) **for** $i = 1, \ldots, n-2$ **do**
(2) **for** $j = i+1, \ldots, n$ **do**
(3) $w_{ij} = a_{ij}(b_j - b_i)$ ▷ Computing the first $(n-2)(n+1)/2$ intermediate products
(4) **for** $i = 1, \ldots, n$ **do**
(5) $q_i = b_i \sum_{j=1}^{n} a_{ji}$ ▷ Computing the final n intermediate products
(6) **for** $i = 1, \ldots, n-2$ **do**
(7) $c_i = \sum_{j=1}^{i-1} w_{ji} + \sum_{j=i+1}^{n} w_{ij} - q_i$
(8) $c_{n-1} = -\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-2} w_{ij} - \sum_{j=1}^{n-2} w_{jn} + \sum_{i=1, i \neq n-1}^{n} q_i$
(9) $c_n = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} + \sum_{i=1}^{n-1} q_i$



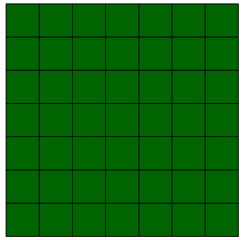
22	12	4	2	5
17	16	3	6	9
20	19	18	11	7
23	1		24	13
21	14	10	8	15

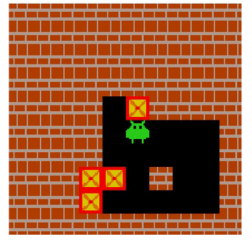






1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	





Rubik's cube | 24 puzzle | Lights Out (7×7) | Sokoban

Fawzi, Alhussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning." Nature 610.7930 (2022): 47-53.

Outline

- Background
- Value functions
- Dynamic programming
- Model-free reinforcement learning

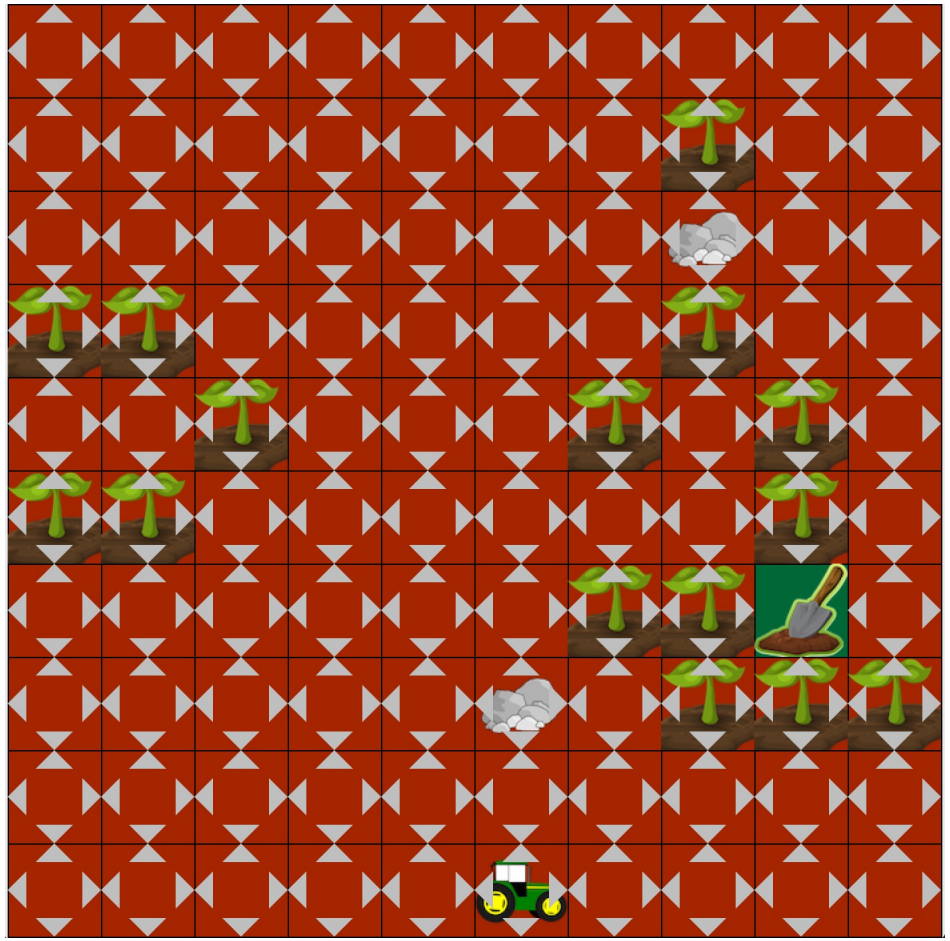
Value Functions

- **Episode:** Starts at some start state at timepoint 0 and ends at a special state, called the terminal state, at timepoint T
- **Return:** the sum of rewards after timestep t
  - $G_t = R_{t+1} + R_{t+2} + R_{t+3} \ldots + R_T$
  - We seek to maximize the expected return
- **State-value function**
  - $v_\pi(s)$ is the expected return when in state s and following policy π
- **Action-value function**
  - $q_\pi(s, a)$ is the expected return when taking action a in state s and then following policy π
- Value functions are specific to a given policy π

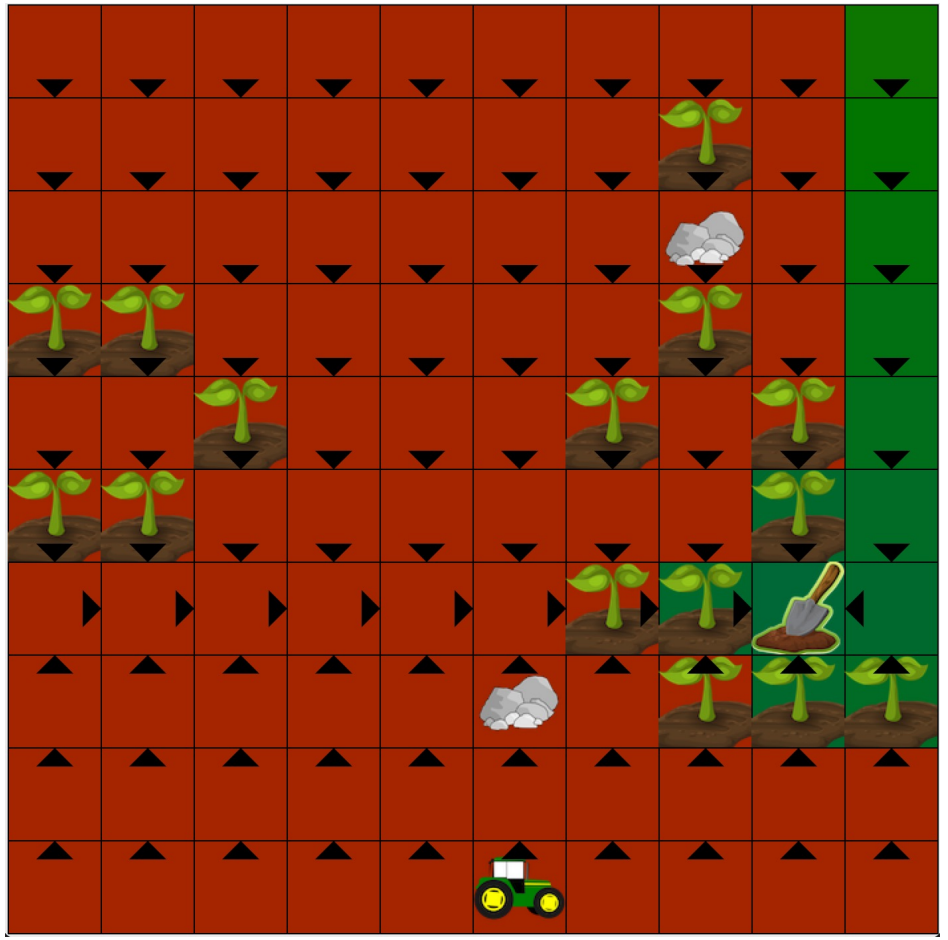
Optimal Policy and Value Function

- Which policy is better?
  - $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$
- A policy that achieves the greatest possible return from any state is an optimal policy
  - $\pi_* \geq \pi'$ for all π'
- The optimal value function is the value function obtained when following the optimal policy
  - $v_*(s) = \max_\pi v_\pi(s)$
  - $q_*(s,a) = \max_\pi q_\pi(s,a)$
- Optimal policies can be obtained by behaving greedily with respect to the optimal value function
- Many RL methods first learn a value function and then induce a policy by behaving greedily with respect to the value function
- Is the optimal policy unique?
- Is the optimal value function unique?

Value Functions

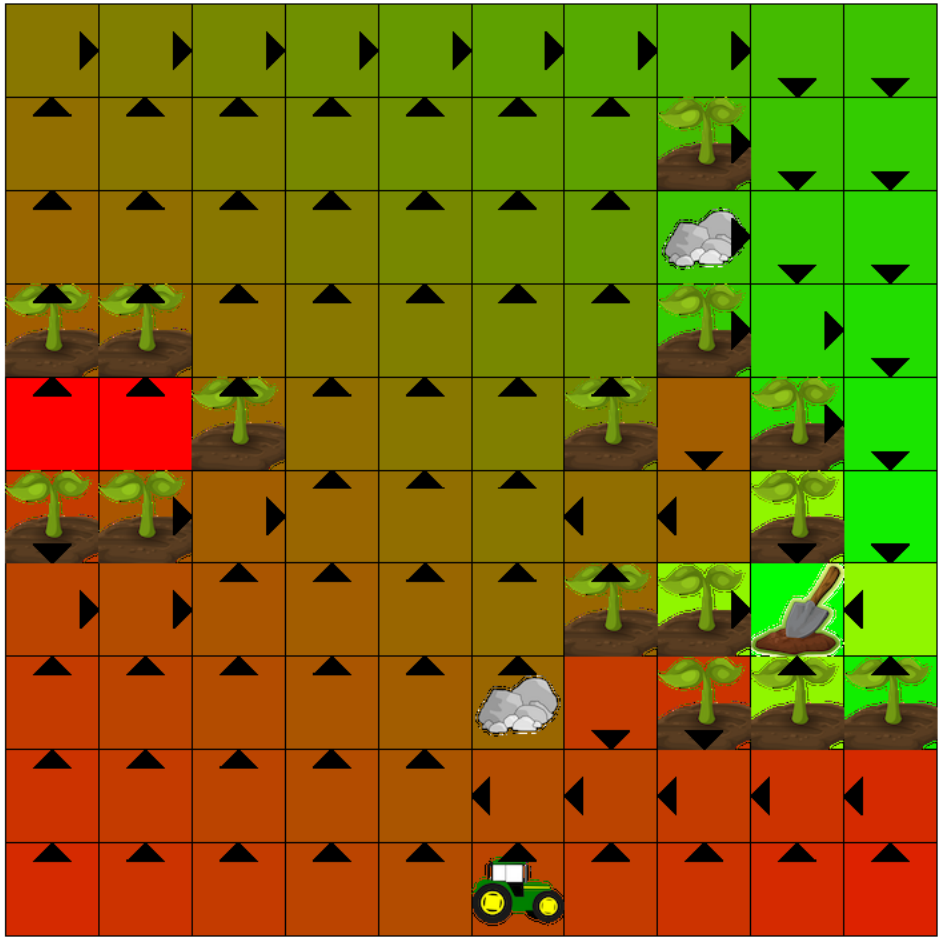


$v_\pi(s)$
π is uniform random for all states



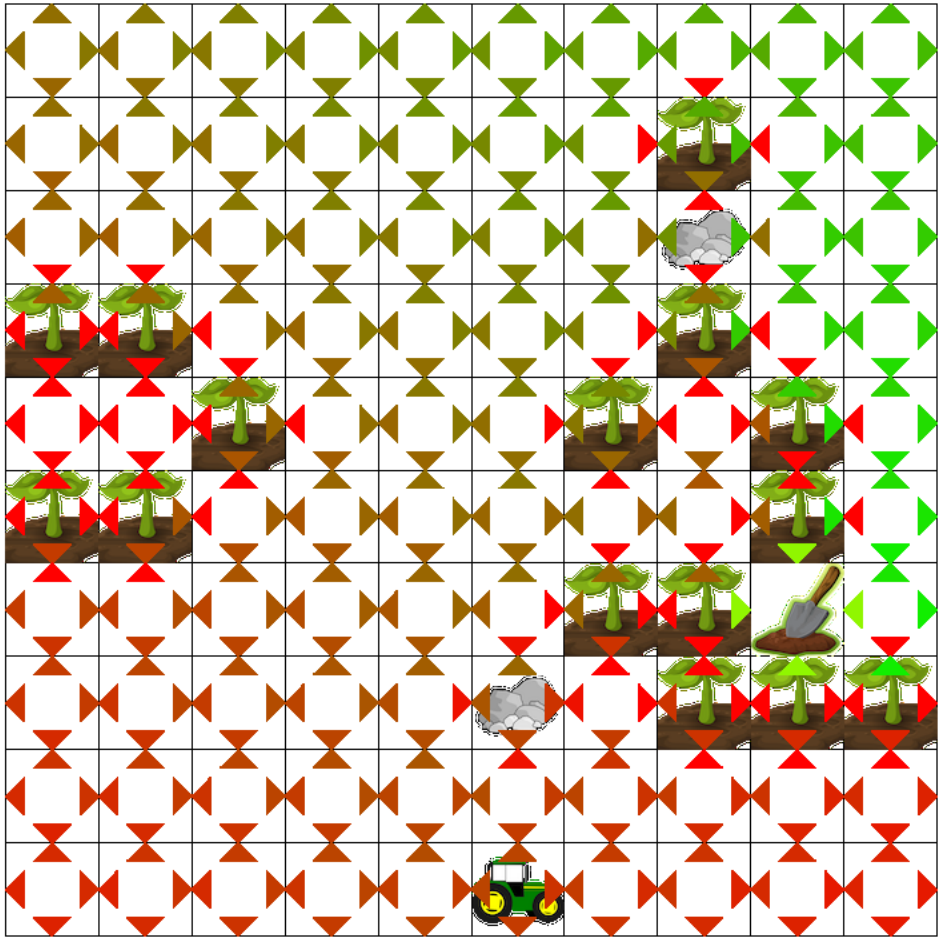
$v_\pi(s)$
π says to go up if below goal, go down if above goal, otherwise, go in the direction of the goal

Optimal Value Function



$$v_*(s)$$

$$\pi_* = \underset{a}{\operatorname{argmax}}\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s')\right)$$



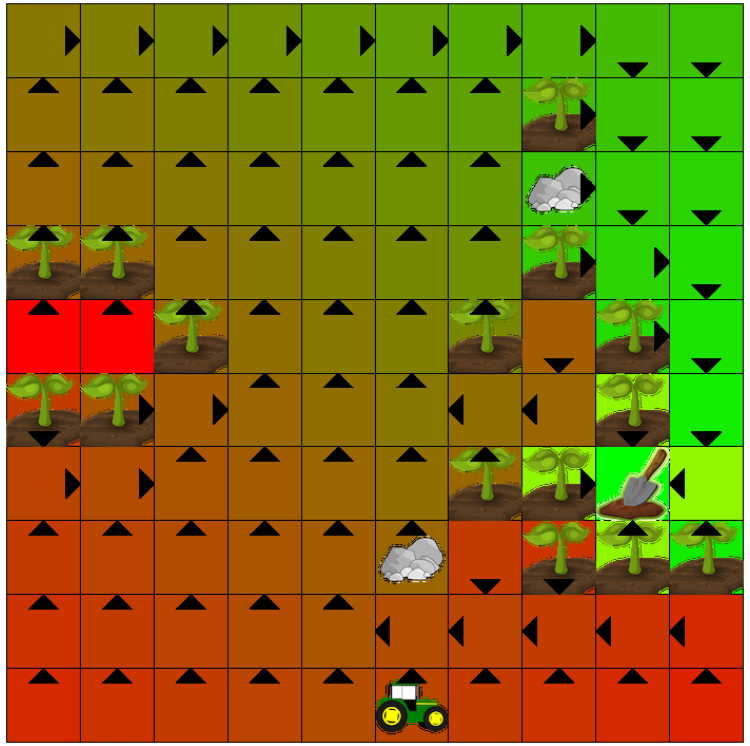
$$q_*(s,a)$$

$$\pi_* = \underset{a}{\operatorname{argmax}}\, q_*(s,a)$$

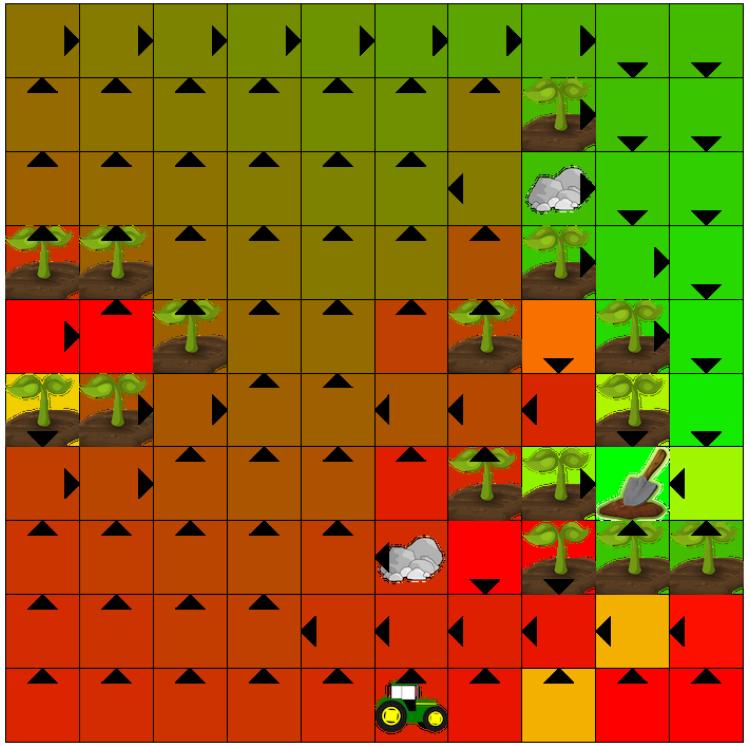
Optimal Value Function: Strong Winds

- Wind blows you right with some probability p

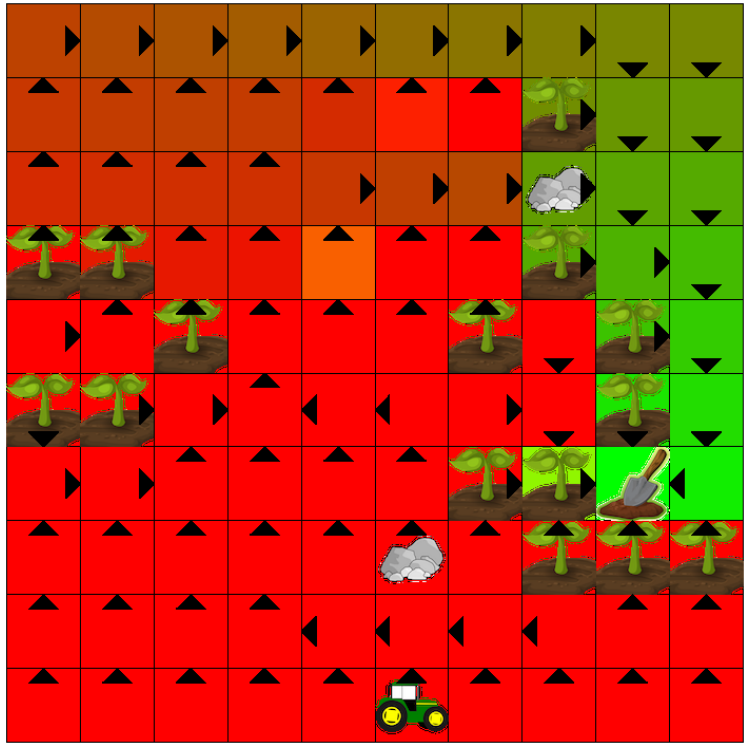
$p = 0$

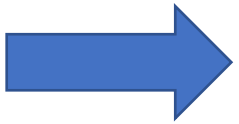


$p = 0.1$



$p = 0.5$

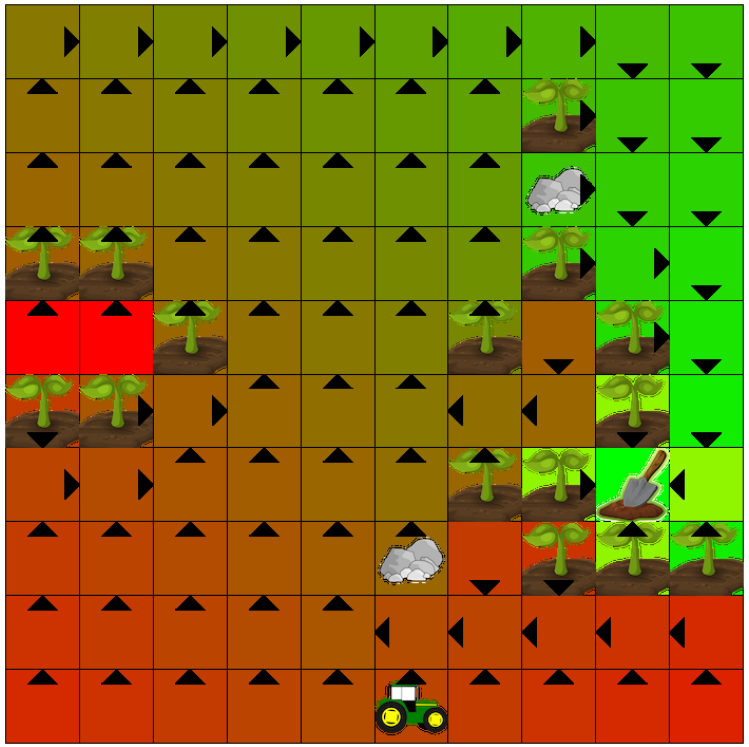




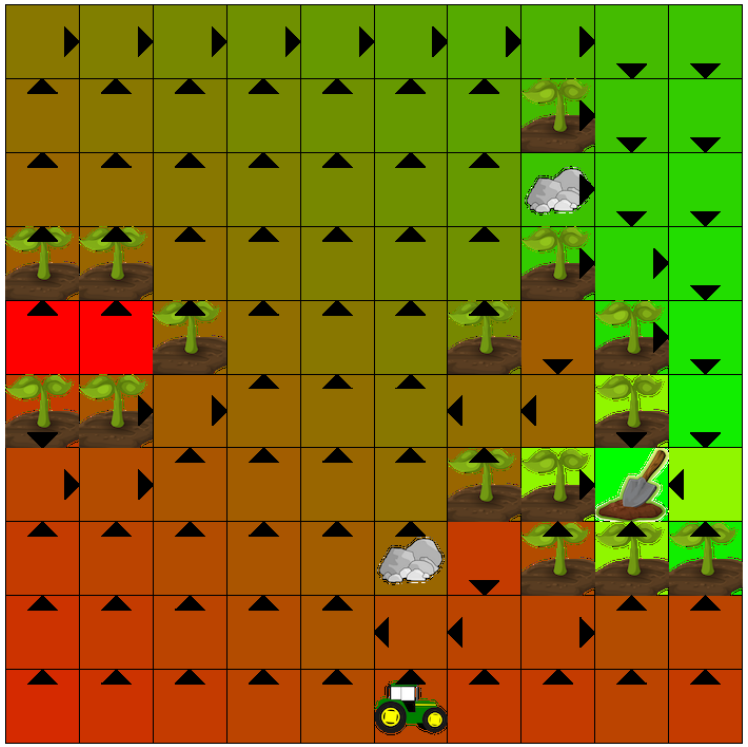
Optimal Value Function: Effect of Rewards

- Reward r of driving on a plant

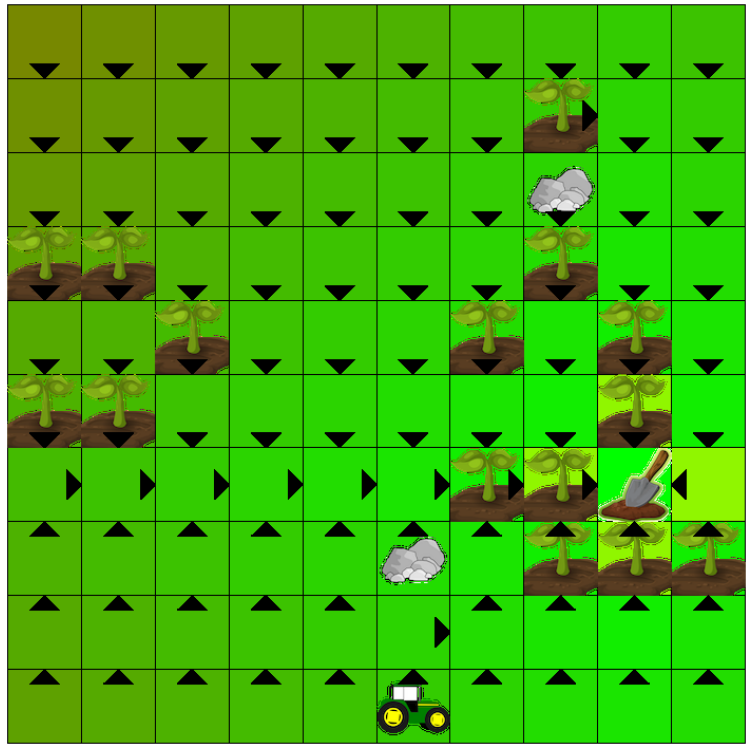
$r = -50$



$r = -20$



$r = -2$

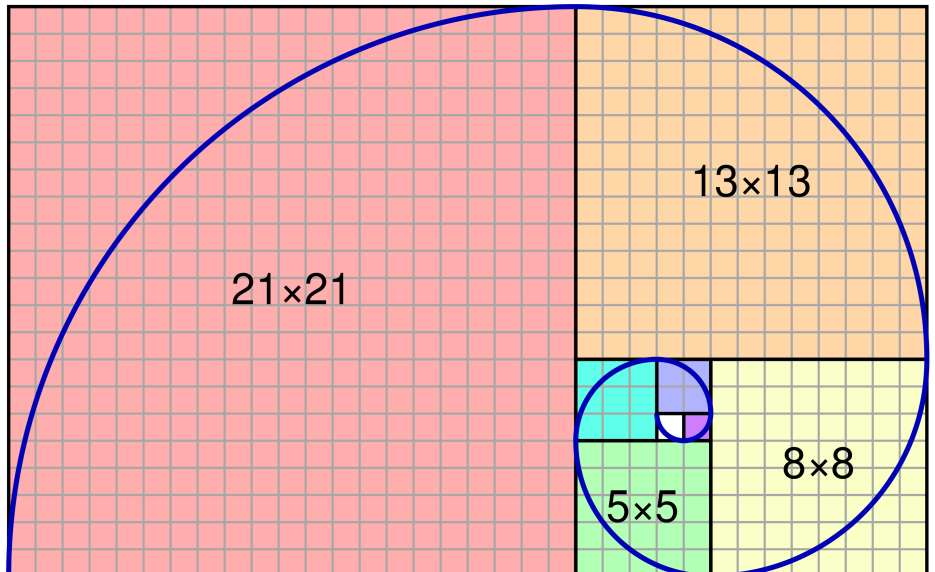


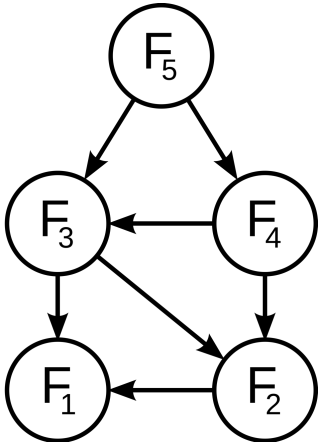
Outline

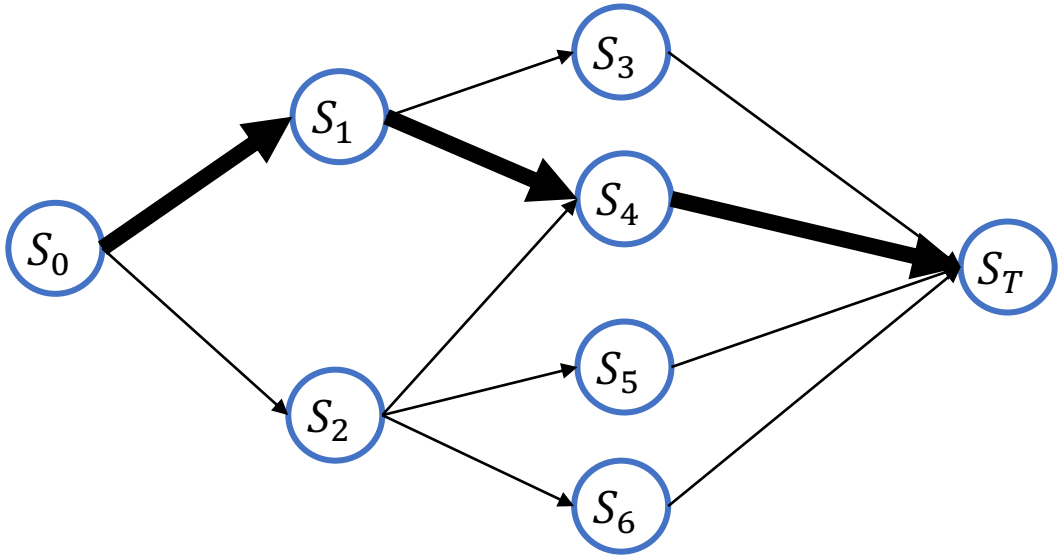
- Background
- Value functions
- Dynamic programming
- Model-free reinforcement learning

Dynamic Programming

- Solves problems by recursively breaking them down into simpler subproblems
- Requires
  - Optimal substructure: Can construct an optimal solution from optimal solutions of subproblems
    - Principle of optimality
  - Overlapping subproblems: Solutions to subproblems are re-used
    - Value functions
- Other applications: Fibonacci sequence, Scheduling, Sequence alignment (DNA)





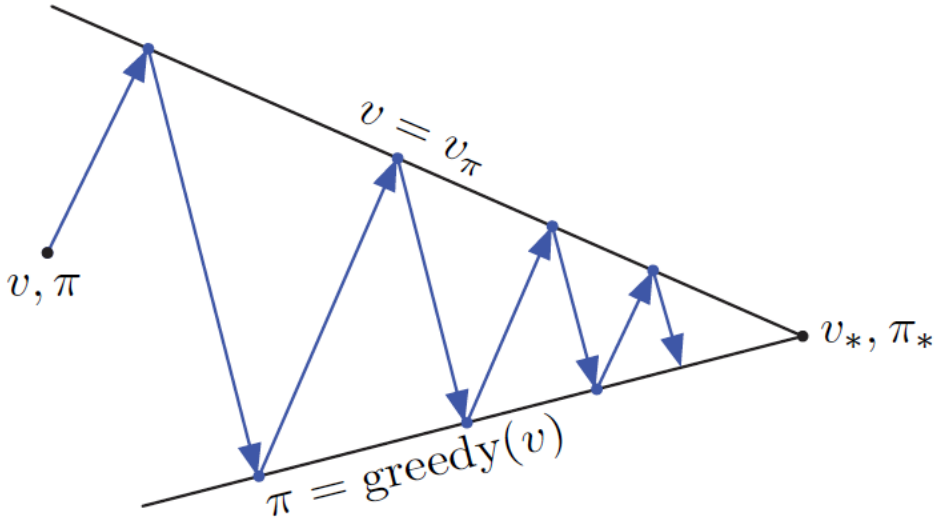


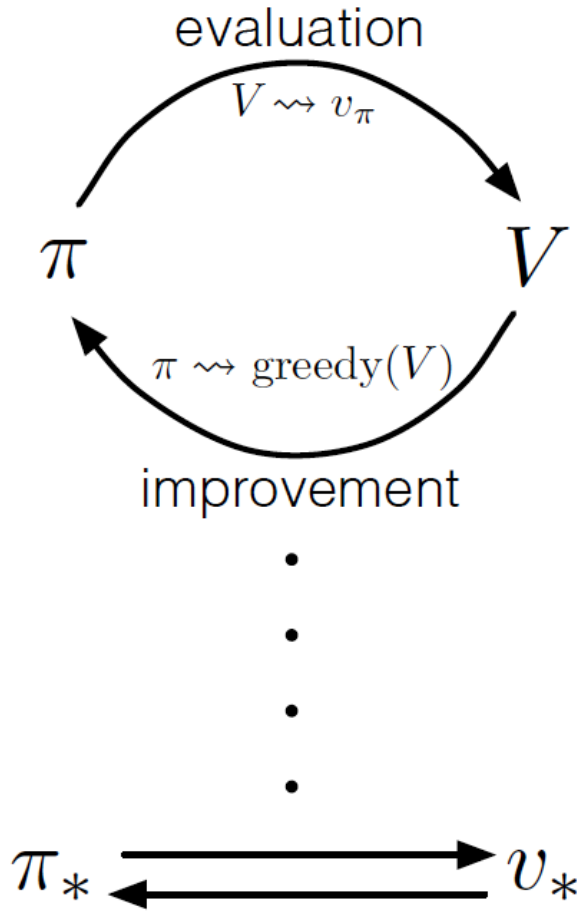
Dynamic Programming

- We will use it to evaluate a policy and compute an optimal policy given a perfect model an MDP
- Foundational for reinforcement learning
- Using dynamic programming, we will do policy iteration by iterating between policy evaluation and policy improvement

Generalized Policy Iteration

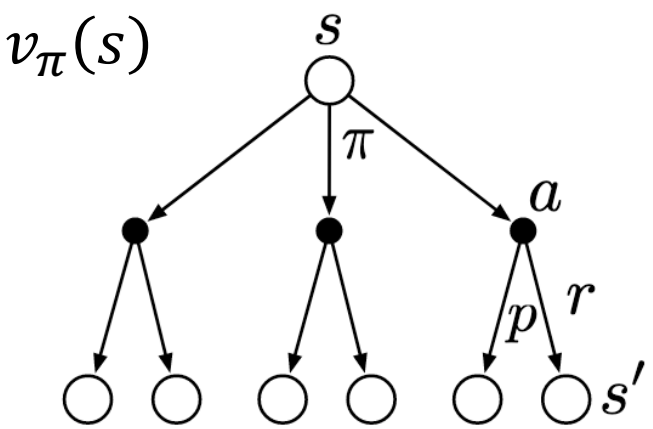
- **Policy Evaluation:** Estimate the expected future reward when following policy π
- **Policy Improvement:** Improve policy π so that it obtains a greater expected future reward
- We can obtain an optimal policy by iterating between policy evaluation and policy improvement

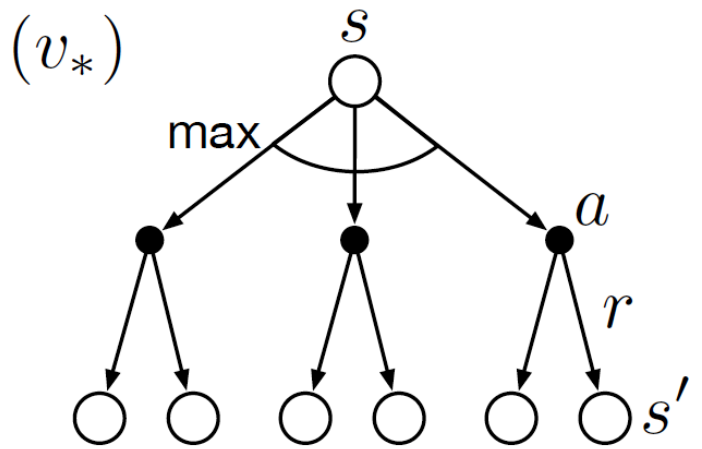




Bellman Equation and Bellman Optimality Equation

- To determine the value of a state, s, we need only perform a one-step lookahead
  - Look at every possible action we can take
  - Look at the expected reward for taking those actions
  - Look at the probability of transitioning to a next state, s'
  - Look at value of next state, s'
- Bellman equation
  - Summarize based on probability of taking those actions
  - $v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$
- Bellman optimality equation
  - Summarize based on the max of all possible actions
  - $v_*(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s'))$





Policy Evaluation

- Construct v_π, the expected future reward when following policy π
- Use the Bellman equation as an update rule
- Theory shows that, in the tabular case, this is guaranteed to converge to v_π

Policy Evaluation AI Farm

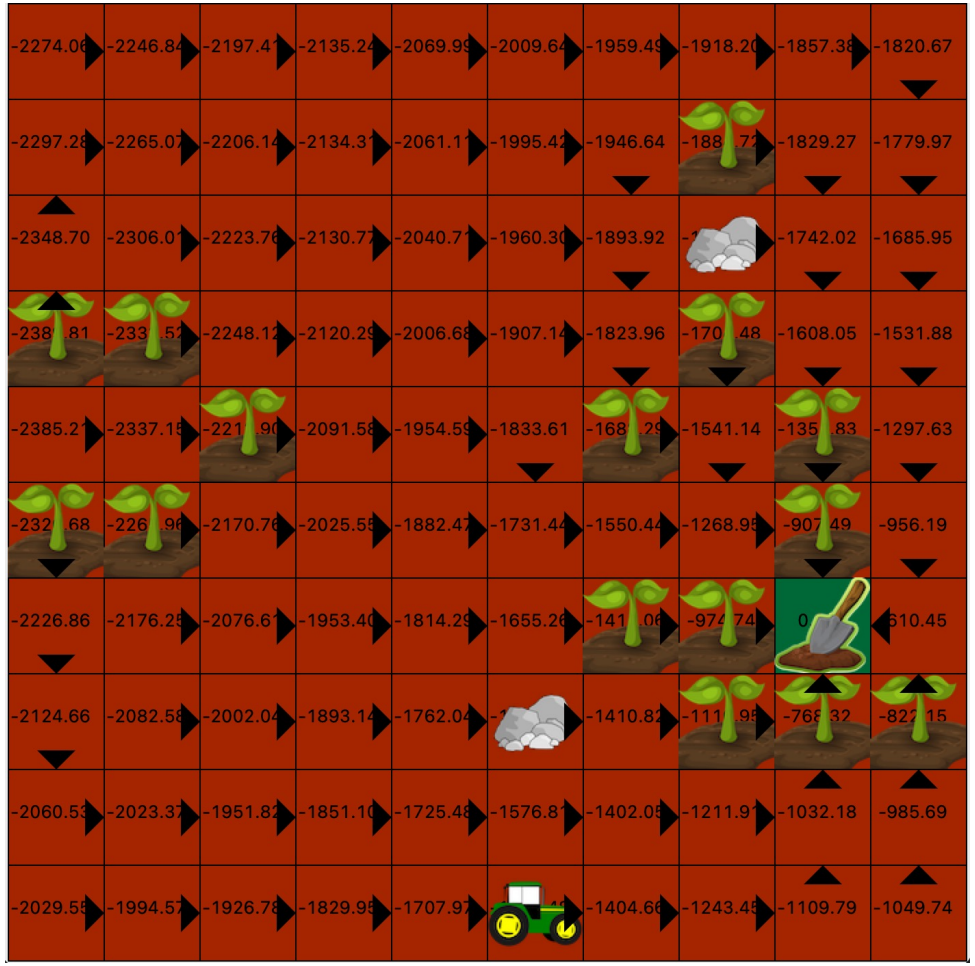
- Policy: Uniform random

Policy Improvement

- We have evaluated policy π, how can we find a better policy?
- $\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_{\pi}(s)$ for all $s \in \mathcal{S}$
- We set the policy to be greedy with respect to v_{π}
  - Perform a one-step lookahead and choose the action with the highest expected future reward
  - $\pi'(s) = \underset{a}{\text{argmax}}(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_{\pi}(s'))$
- This policy will always be the same or better than the previous one
  - Policy improvement theorem.

Policy Improvement

- Better than original uniform random policy
- Still not optimal



When acting greedily with respect to v_π

Policy Iteration

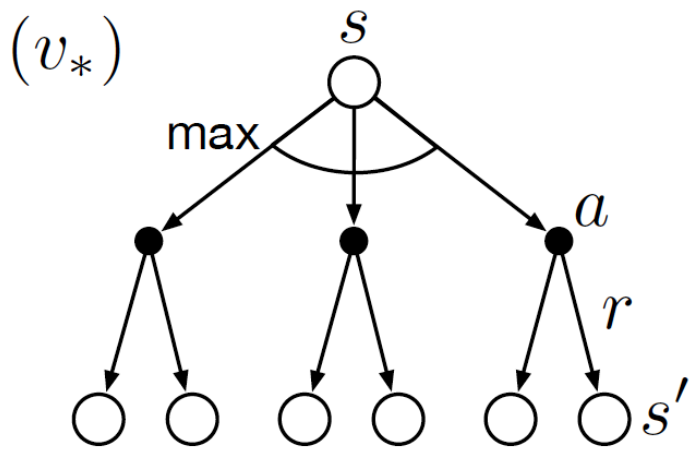
- Policy improvement improves a policy π and obtains a new policy π' such that, $\pi' \geq \pi$
- If $\pi' = \pi$, then $v_{\pi'} = v_{\pi}$
- Therefore, $v_{\pi'}(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_{\pi'}(s'))$
- This is the same as the Bellman optimality equation!
- Does this mean $\pi' = \pi_*$ and $v_{\pi'} = v_*$?
- There is a proof showing that the Bellman optimality equation is a unique fixed point
  - Similar to that of the Bellman equation

Policy Iteration: AI Farm

```
(drl) forestagostinelli@Forests-MacBook-Pro Farm_Grid_World % pytho
n run_policy_iteration.py --map maps/map1.txt --wait 1.0 --wait_eva
l 0.0 --rand_right 0.0
```

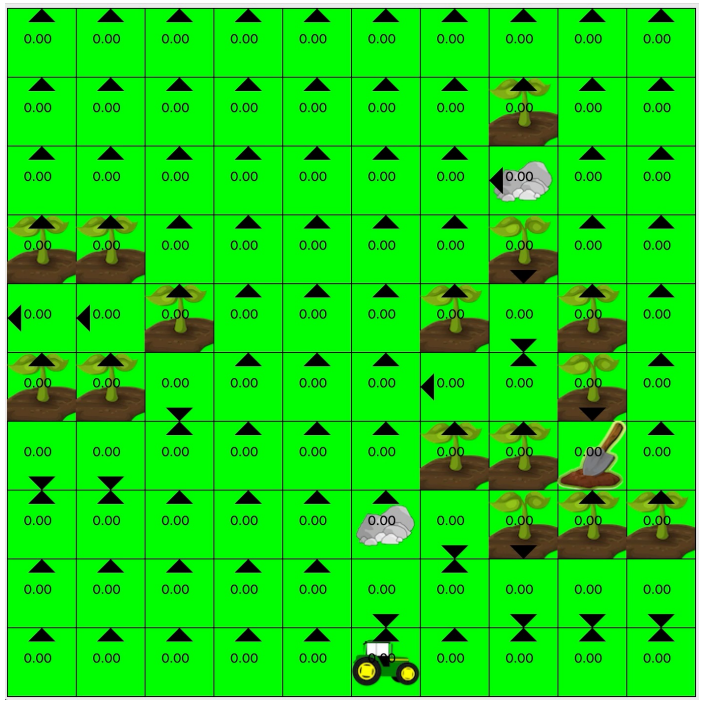
Value Iteration

- Find the optimal value function
- Recall the Bellman optimality equation
  - $v_*(s) = \max_a(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s'))$
- Use this as an update rule
  - $V(s) = \max_a(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$
  - Guaranteed to converge to v_*
- Combines policy evaluation and policy improvement into one step

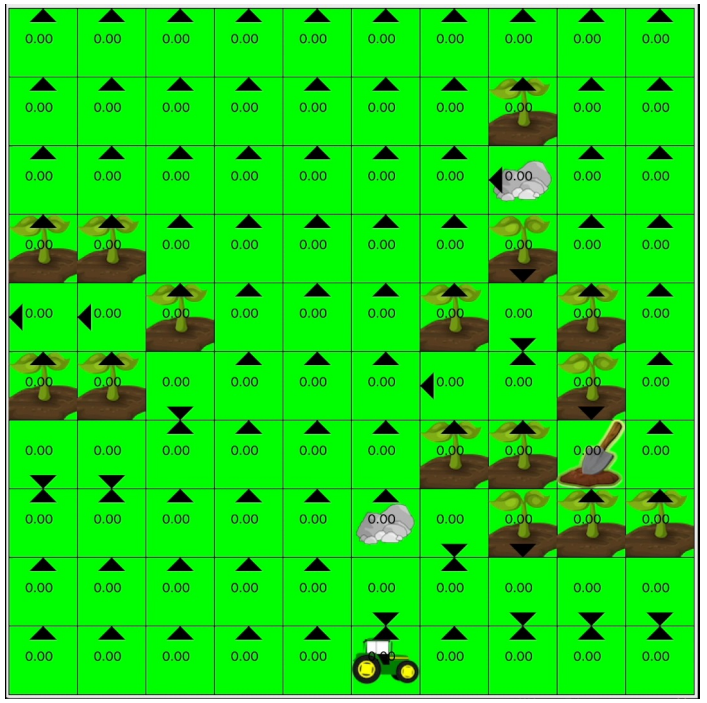


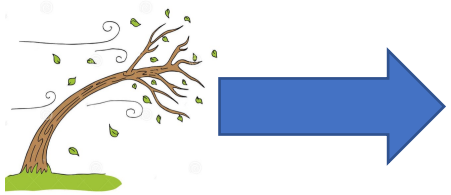
Value Iteration: AI Farm

$p = 0$

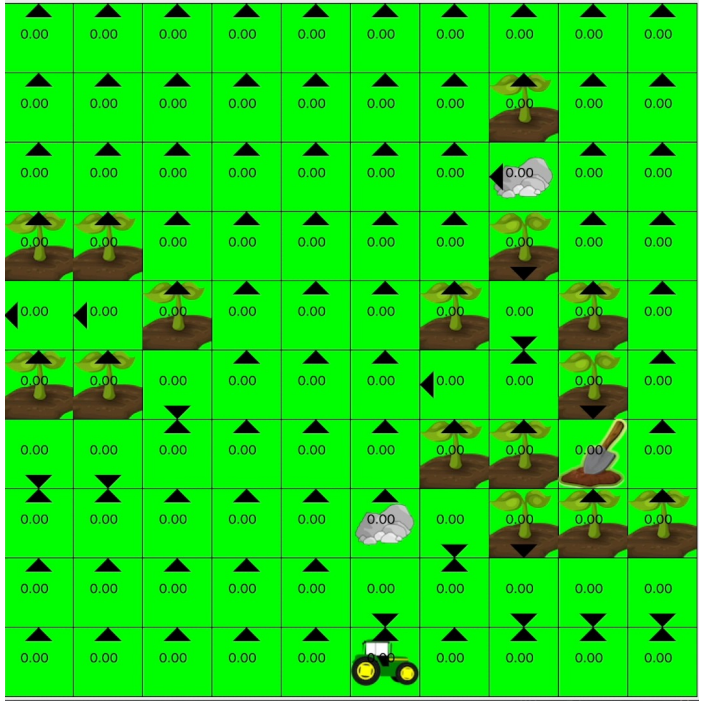


$p = 0.1$



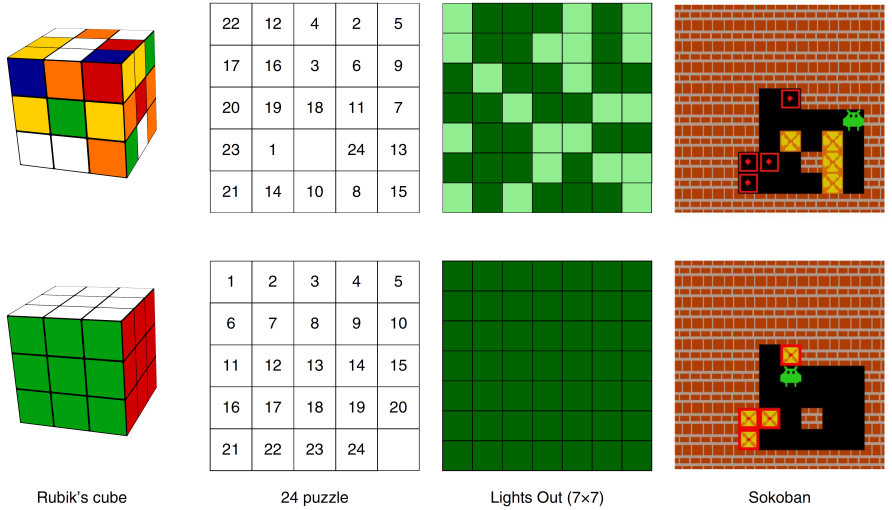


$p = 0.5$

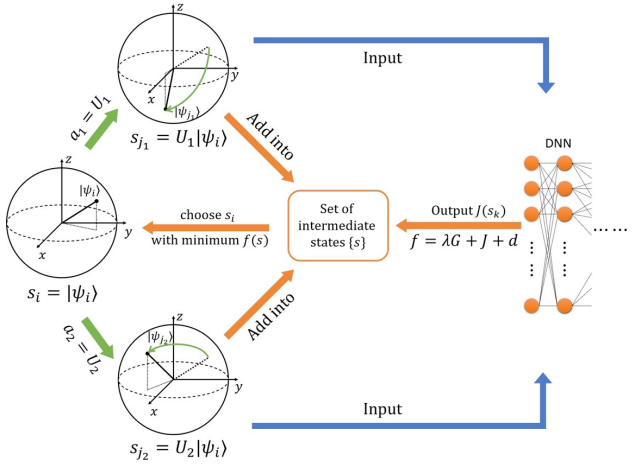


Approximate Value Iteration

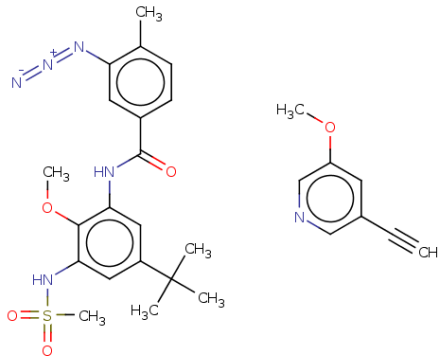
- Sometimes, we can accurately define the MDP, however, the state space is too large to store in a table
- Therefore, we will need to use an architecture to approximate the value
- The goal is to have an architecture whose number of parameters is far less than the size of the state space



Puzzles



Quantum Compiling



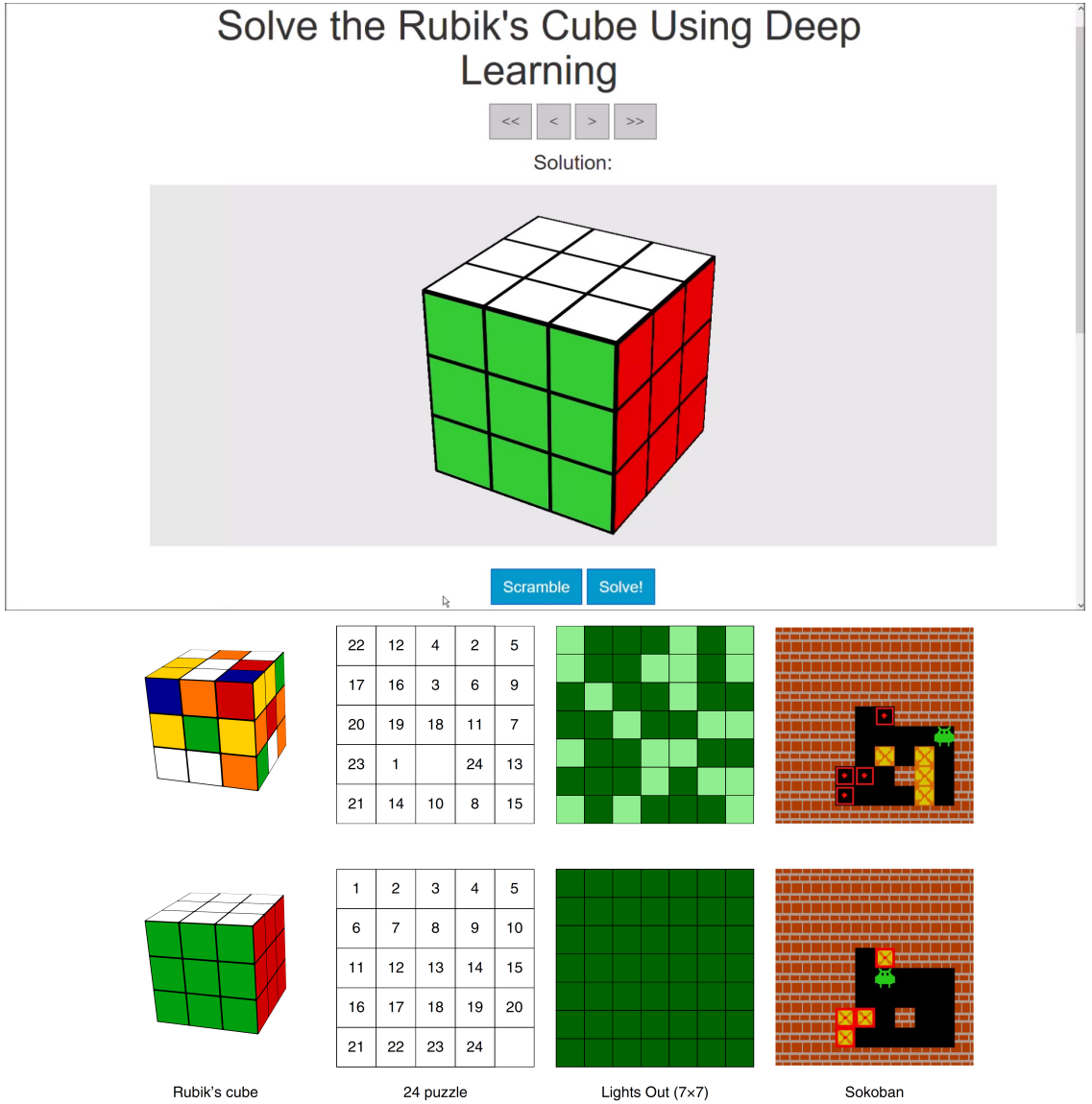
Chemical Synthesis

Approximate Value Iteration (AVI)

- Value Iteration
  - $V(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$
- Approximate Value Iteration
  - $y = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, \hat{v}(s', \boldsymbol{w}))$
  - $E(\boldsymbol{w}) = \frac{1}{2}\big(y - \hat{v}(s, \boldsymbol{w})\big)^2$
  - $\nabla_{\mathbf{w}} E(\mathbf{w}) = \big(y - \hat{v}(s, \boldsymbol{w})\big) \nabla_{\mathbf{w}} \hat{v}(s, \boldsymbol{w})$
  - Even though y depends on $\mathbf{w}$, we do not differentiate y with respect to $\mathbf{w}$.
  - Only supervision is that $y = 0$ for terminal states
- Unlike in the tabular case, we cannot guarantee convergence in the case of neural network function approximators
- However, there are many different methods that we can use to make deep reinforcement learning work in practice

AVI: Solving the Rubik's Cube

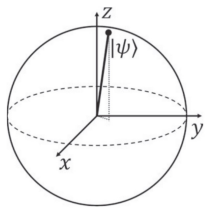
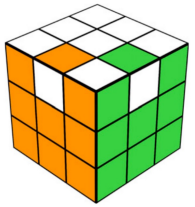
- 4.3 x 10^{19} possible combinations
- No domain-specific knowledge
- DeepCubeA solves the Rubik's cube and other puzzles
  - Puzzles have up to 3.0 x 10^{62} possible combinations.
- Finds a shortest path in the majority of verifiable cases
- http://deepcube.igb.uci.edu/
- RL was combined with **deep neural networks** and **A* search**

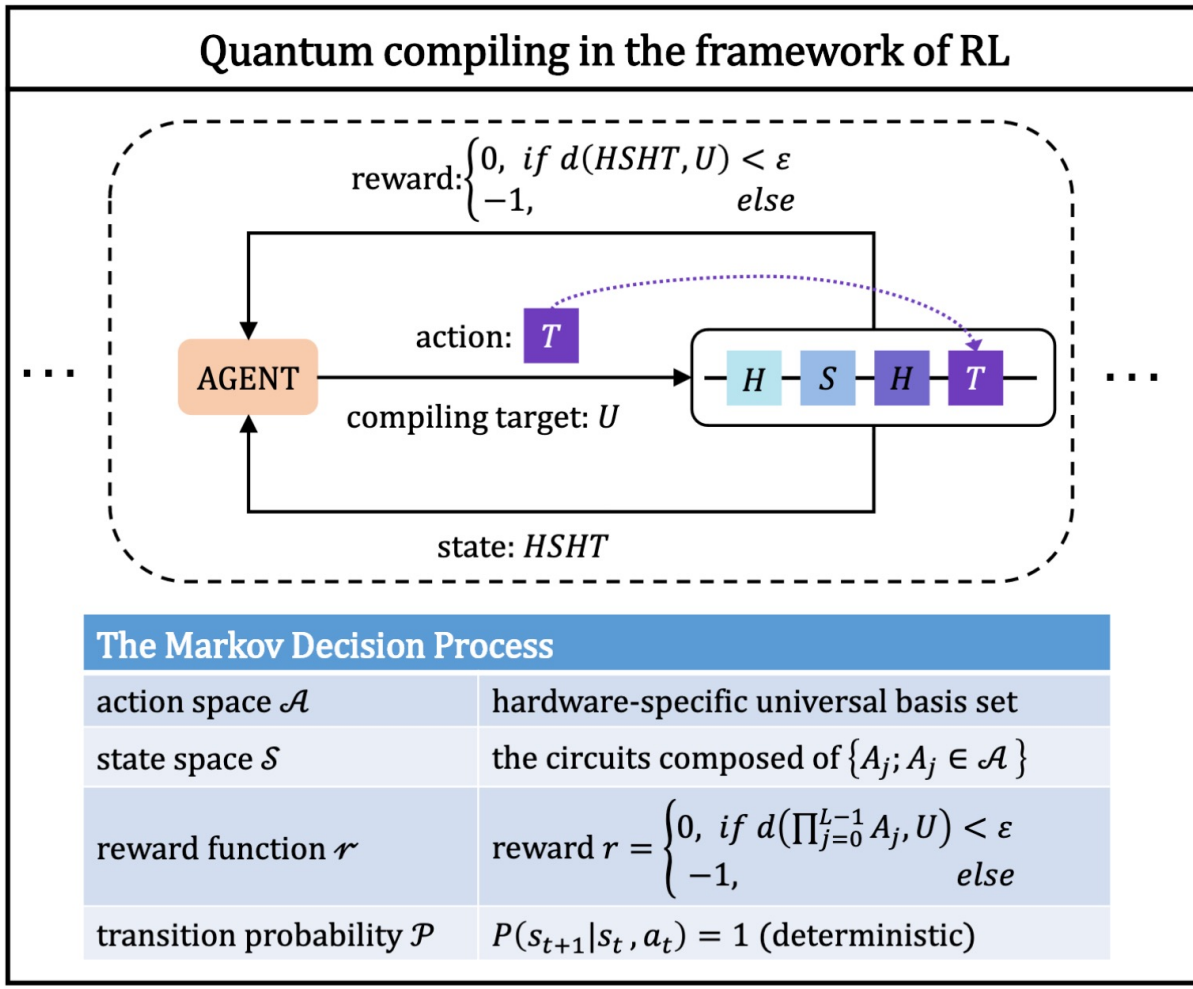


Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.

AVI: Quantum Algorithm Compilation

- Given a quantum algorithm, a compiler must synthesize a quantum circuit for this algorithm from a given set of quantum gates
- If a given circuit is below an error threshold, then the problem is considered solved

System		
		
Initial state	The unitary to be approximated	The scrambled cube
Target state	The identity matrix	The solved cube
Basic move	A gate from the universal set	Rotation of one face

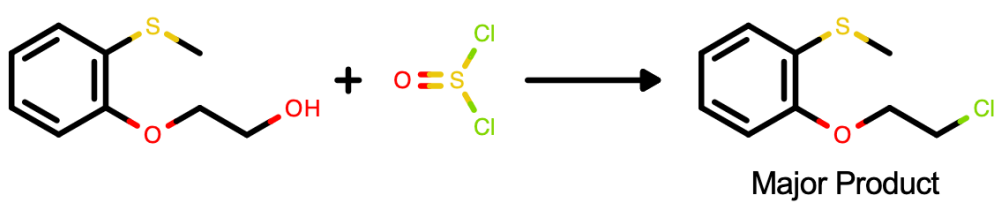


The Markov Decision Process	
action space $\mathcal{A}$	hardware-specific universal basis set
state space $\mathcal{S}$	the circuits composed of $\{A_j; A_j \in \mathcal{A}\}$
reward function r	reward $r = \begin{cases} 0, & if\ d(\prod_{j=0}^{L-1} A_j, U) < \varepsilon \\ -1, & else \end{cases}$
transition probability $\mathcal{P}$	$P(s_{t+1}\|s_t, a_t) = 1$ (deterministic)

Zhang, Yuan-Hang, et al. "Topological Quantum Compiling with Reinforcement Learning." Physical Review Letters 125.17 (2020): 170501.
Bao, Ning, and Gavin S. Hartnett. "Twisty-puzzle-inspired approach to Clifford synthesis." *Physical Review A* 109.3 (2024): 032409.

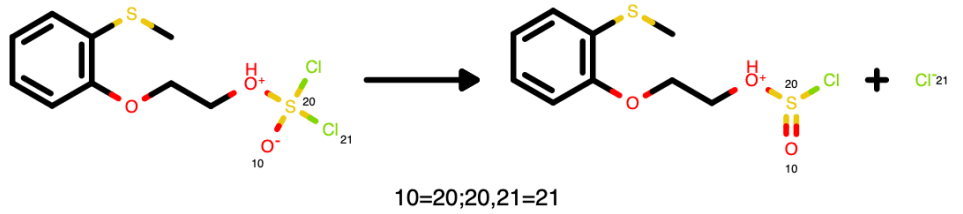
AVI: Reaction Mechanisms

- Chemical reactions are composed of smaller steps called reaction mechanisms
- Knowledge of the reaction mechanisms that compose a chemical reaction allows practitioners to
  - Validate reaction feasibility
  - Improve reaction efficiency
  - Predict reaction outcome under different conditions
- Most chemical reaction prediction methods skip reaction mechanisms and predict products directly from reactants

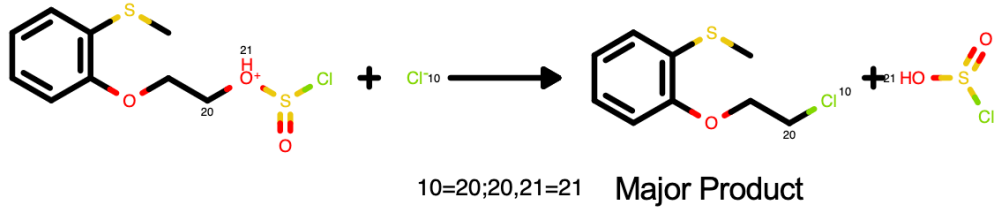


1

2



3



Panta, Rojina, et al. “Finding Reaction Mechanism Pathways with Deep Reinforcement Learning and Heuristic Search.” *ICAPS PRL Workshop 2024*

Outline

- Background
- Value functions
- Dynamic programming
- Model-free reinforcement learning

Crucial Assumption

- Assuming environment dynamics are known
  - $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$
- Environment dynamics are unknown in many real-world scenarios
- Even if known, may be too costly to compute (i.e. physics)

$$V(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$$

Unknown Unknown

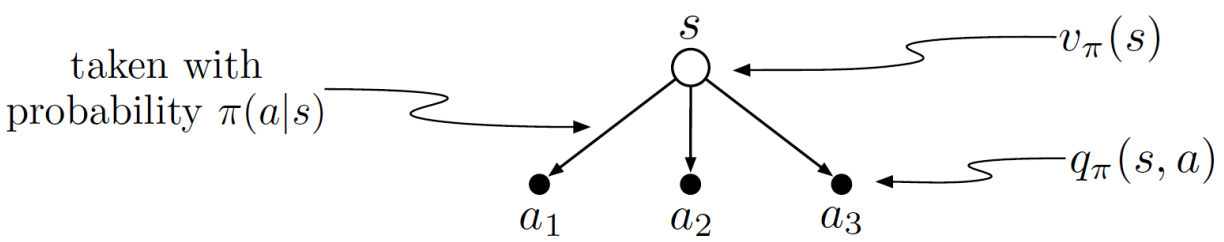
- Instead of using a model, we must learn from experience

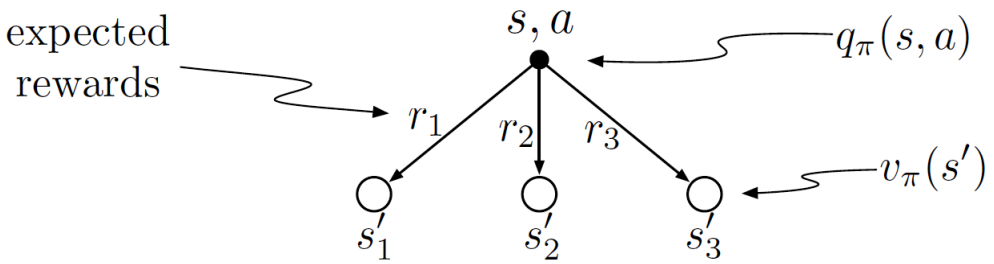
Model-Free RL Concepts

- Exploration vs Exploitation
  - **Exploration**: Learn more about the environment
  - **Exploitation**: Use what you have learned to obtain more reward
- On-policy vs Off-policy
  - Your policy determines your experience
  - We need to explore using randomness
    - May not be the best policy
  - Experience may be delicate and hard to obtain (i.e. a hospital)
  - **Behavior policy**: policy that we use to interact with the environment
  - **Target policy**: policy that we wish to evaluate and/or improve

Model-Free Control

- In this dynamic programming, we induced a policy by doing a one step lookahead using the value function
  - $\pi(s) = \underset{a}{\text{argmax}}(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$
- However, we cannot do this in the model-free case because we do not have access to a model
- Therefore, we use an action-value function to induce a policy
  - $q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s')$
  - $\pi(s) = \underset{a}{\text{argmax}}(Q(s,a))$



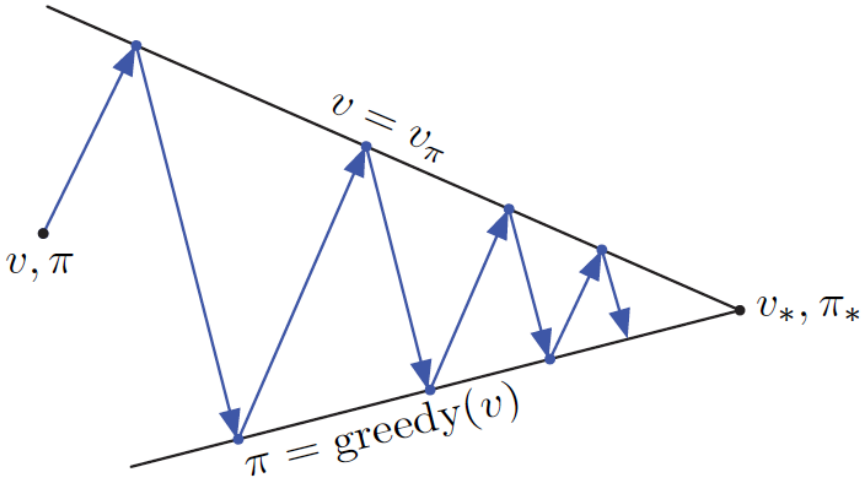


Model-Free Control: Exploration

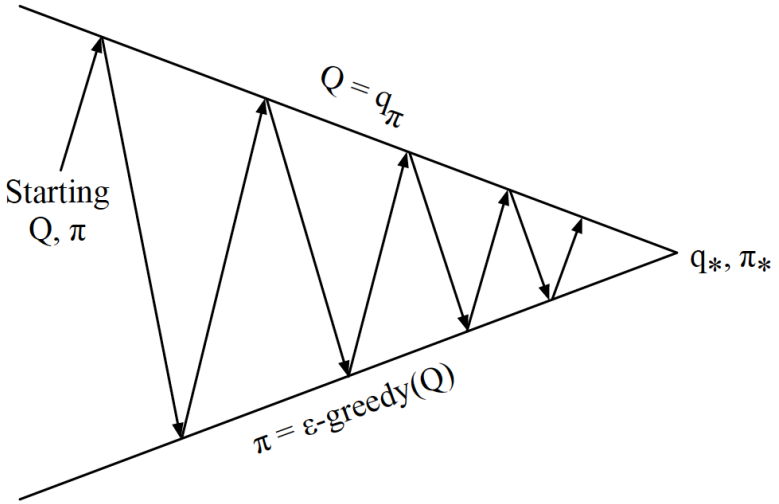
- How do we ensure that we explore our state space?
  - In dynamic programming, we assumed that we could just loop over every possible state
  - Cannot do this in the model-free case
- ϵ-greedy policy
  - Take a random action with probability ϵ
  - Take the greedy action, $\underset{a}{\text{argmax}}\big(Q(s,a)\big)$, with probability 1- ϵ
- While there are many more sophisticated exploration methods, ϵ-greedy exploration can work well on some problems

Model-Free Control

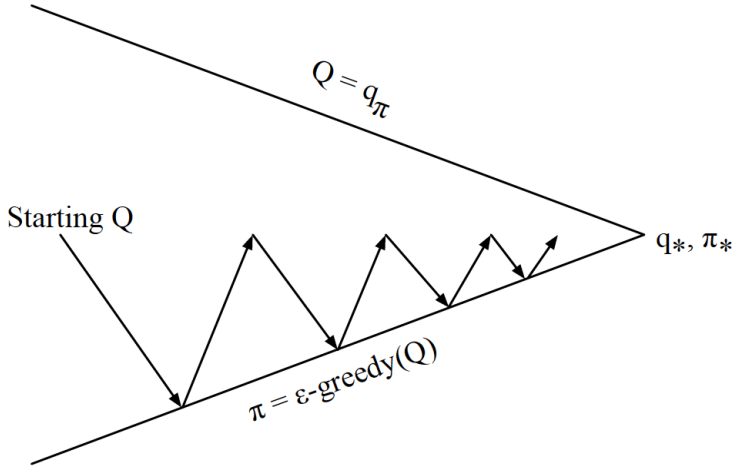
- **Policy Evaluation**: Learn an action-value function.
- **Policy Improvement**: Act epsilon greedily with respect to it.



Dynamic programming



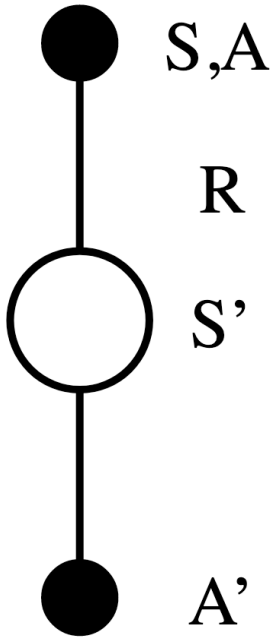
Model-free with infinite time to estimate q_π



Model-free with finite time to estimate q_π

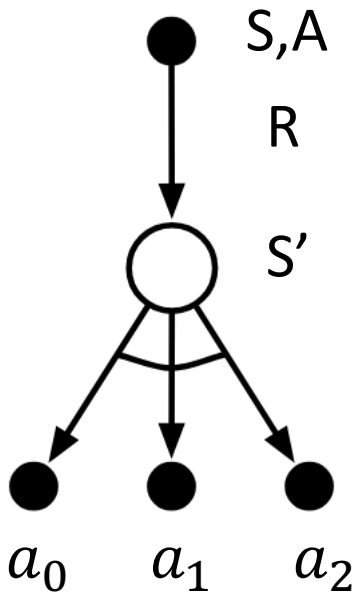
Sarsa

- Model-free on-policy prediction (policy evaluation)
  - $V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$
- Sarsa: model-free on-policy temporal-difference control
  - Sarsa: State, action, reward, state (next), action (next)
  - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  - Behavior policy: epsilon greedy
  - Target policy: epsilon greedy
  - Shown to converge to q_* if greedy in the limit with infinite exploration and if Robbins-Monro conditions hold for α



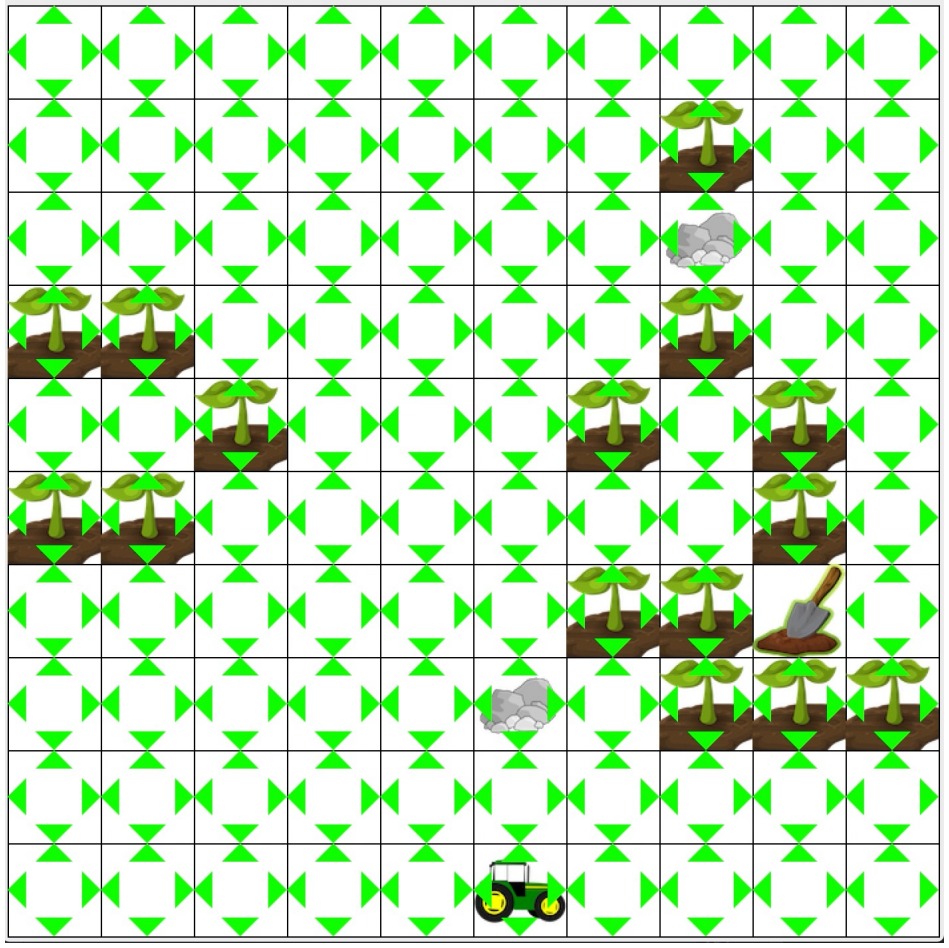
Q-learning

- Q-learning: model-free off-policy temporal-difference control
  - $\mathrm{Q}(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
  - Behavior policy: epsilon greedy
  - Target policy: greedy
  - Converges to q_* if Robbins-Monro conditions hold for α

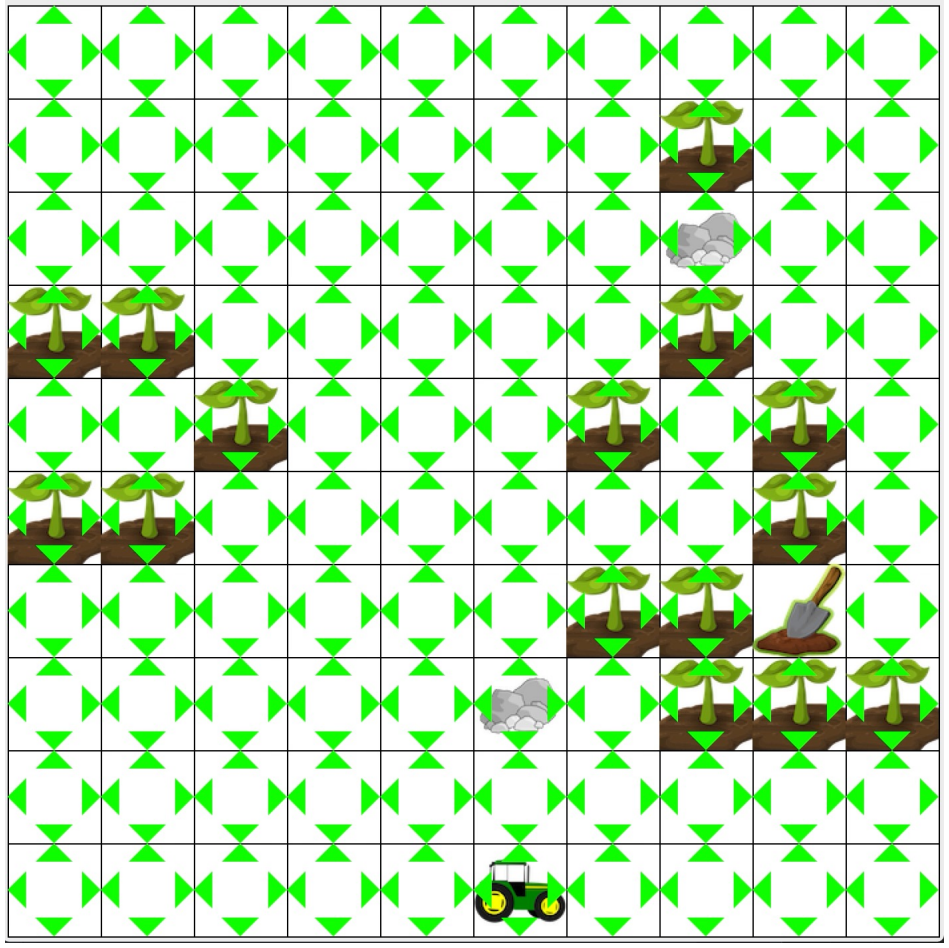


Q-learning

- Why are the action-values for the area surrounded by plants not all red?



Q-learning step-by-step

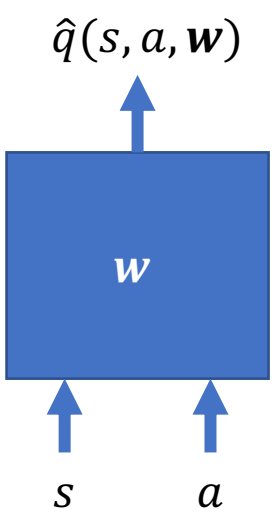


Q-learning. Showing greedy policy after every 100 episodes.

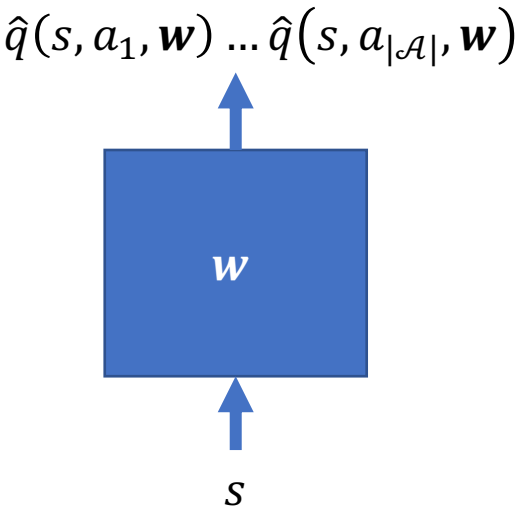
Approximate Q-learning

- Q-learning
  - $\mathrm{Q}(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- Approximate Q-learning
  - $y = r + \gamma \max_{a'} \hat{q}(s',a',\boldsymbol{w})$
  - $E(\boldsymbol{w}) = \frac{1}{2}\big(y - \hat{q}(s,a,\boldsymbol{w})\big)^2$
  - $\nabla_{\mathbf{w}} E(\mathbf{w}) = \big(y - \hat{q}(s,a,\boldsymbol{w})\big)\nabla_{\mathbf{w}} \hat{q}(s,a,\boldsymbol{w})$
  - Even though y depends on $\mathbf{w}$, we do not differentiate y with respect to $\mathbf{w}$.
  - Only supervision is that $y = 0$ for terminal states

Deep Q-Networks (DQNs)

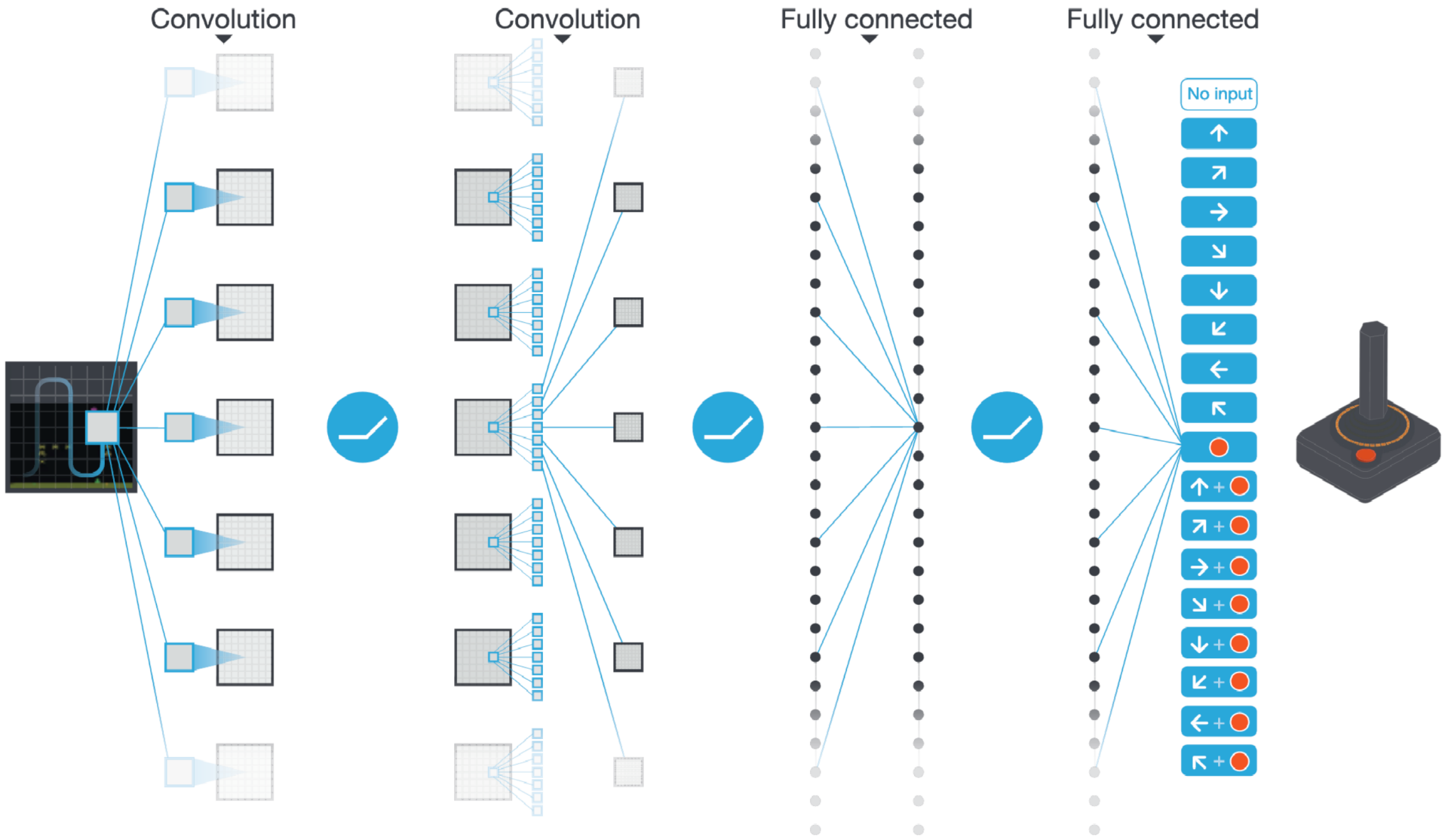


$|\mathcal{A}|$ forward passes needed



1 forward pass needed

DQNs: Atari



Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

DQN: Practical Challenges

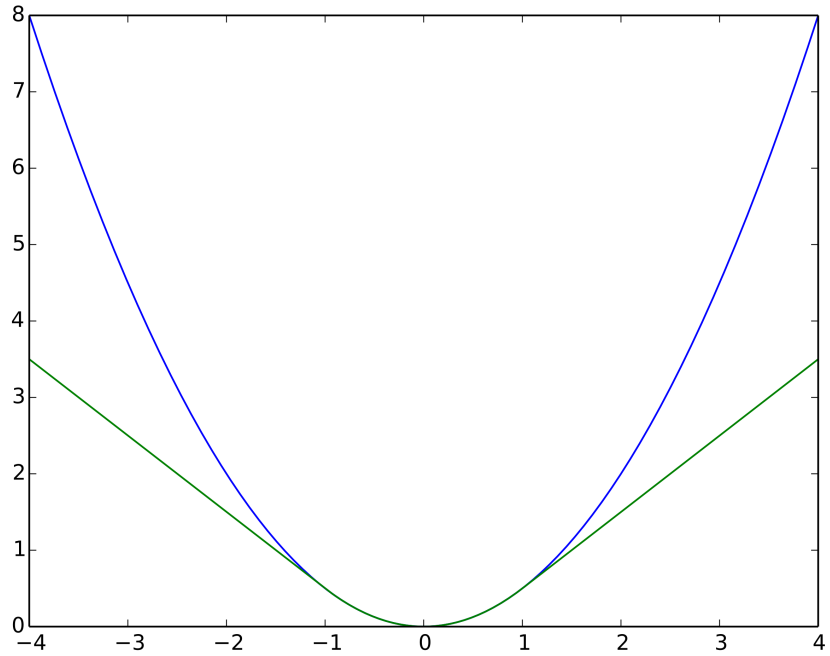
- Training instability due to non-stationary target y
  - $y = r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w})$
  - $E(\boldsymbol{w}) = \frac{1}{2}\left(y - \hat{q}(s, a, \boldsymbol{w})\right)^2$
- Training examples are highly correlated
  - Neural networks are prone to “catastrophic forgetting”
- Training instability due to very large positive or negative rewards

DQNs: Practical Challenges

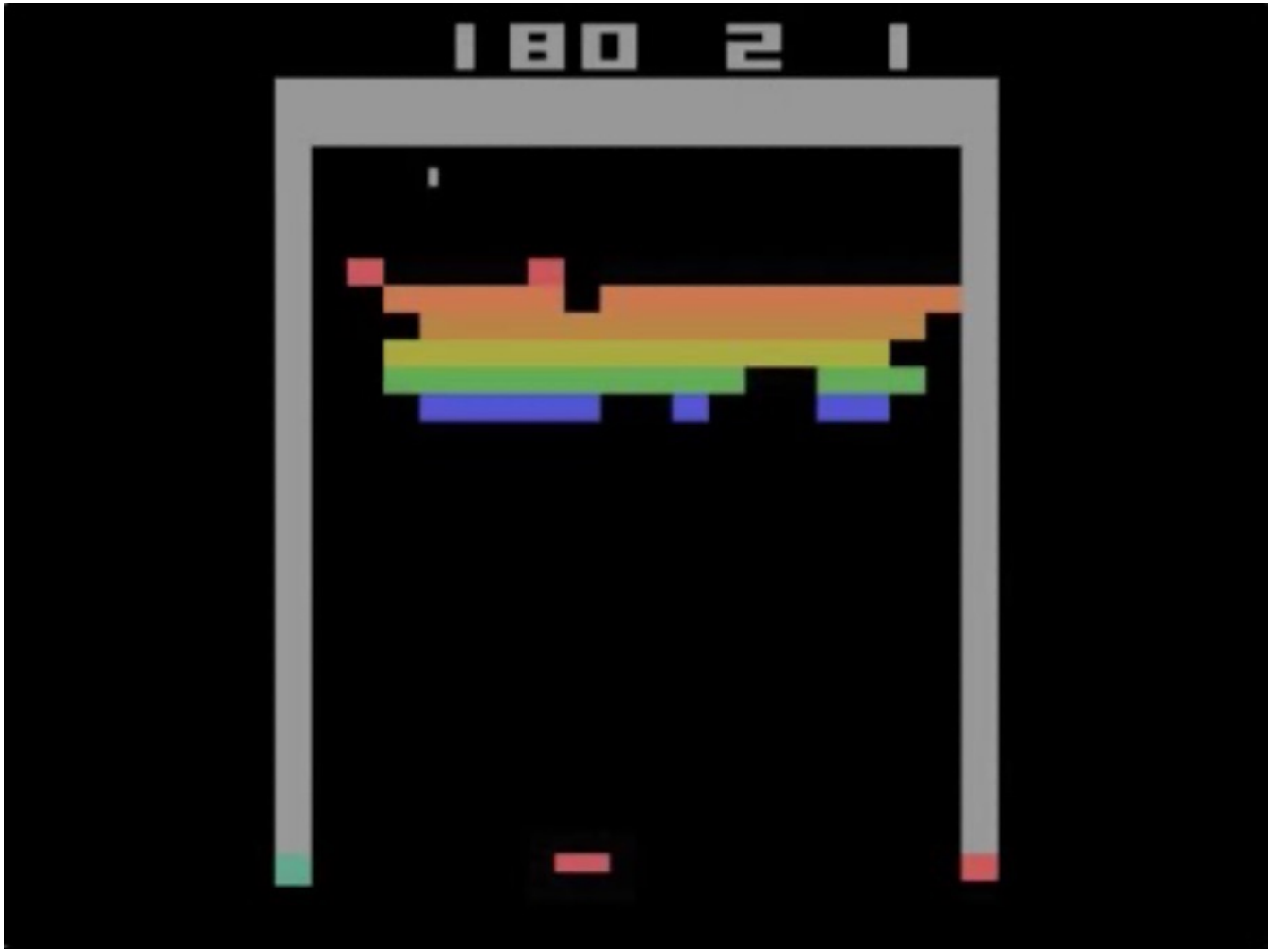
- Instability in target y
  - Solution: Maintain a "target network" with weights $\boldsymbol{w}^-$
  - $y = r + \gamma \max_{a'} \hat{q}(s', a', \boldsymbol{w}^-)$
  - $E(\boldsymbol{w}) = \frac{1}{2}\left(y - \hat{q}(s, a, \boldsymbol{w})\right)^2$
  - Update $\boldsymbol{w}^-$ to equal $\boldsymbol{w}$ every N training iterations
  - Can do this for approximate value iteration, as well
- Training examples are highly correlated
  - Solution: do experience replay
  - Keep all of your experience in a replay buffer
  - Each element is a tuple (s, a, r, s')
  - During training, randomly sample a batch from your replay buffer
  - Decorrelates training examples

DQNs: Huber Loss

- Training instability due to very large positive or negative rewards
- Clips loss after reaching a certain magnitude

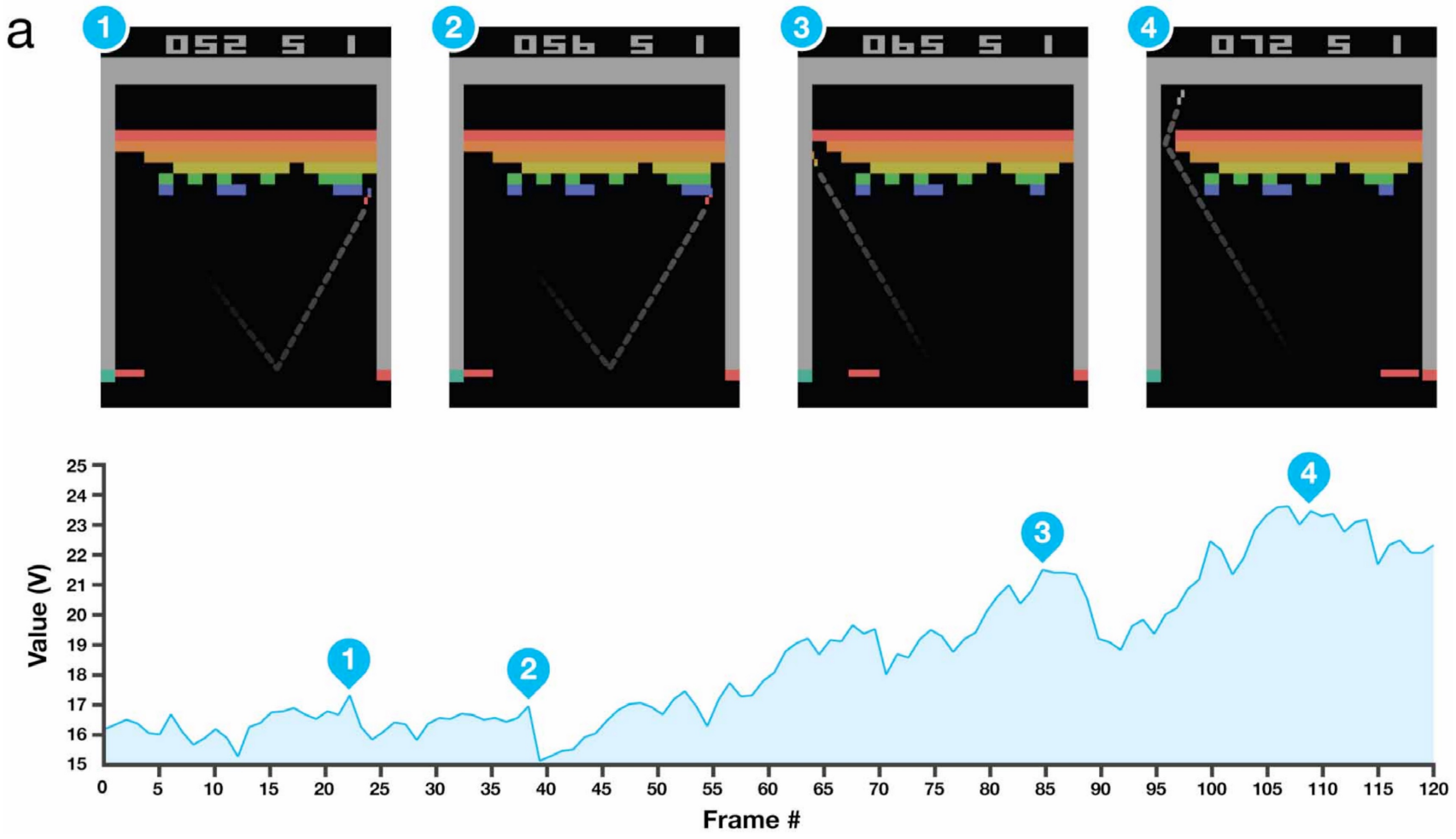


DQNs: Example on Atari

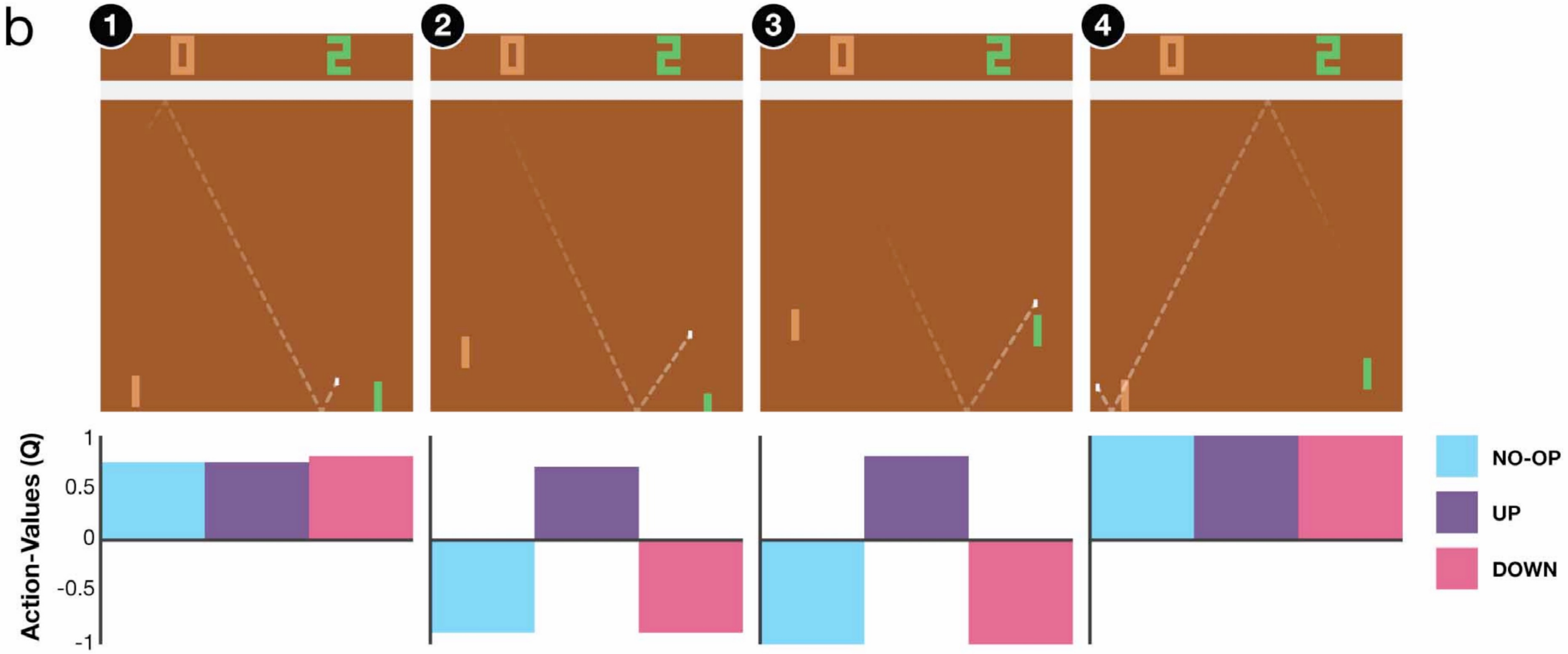


DQN: State Value Behavior

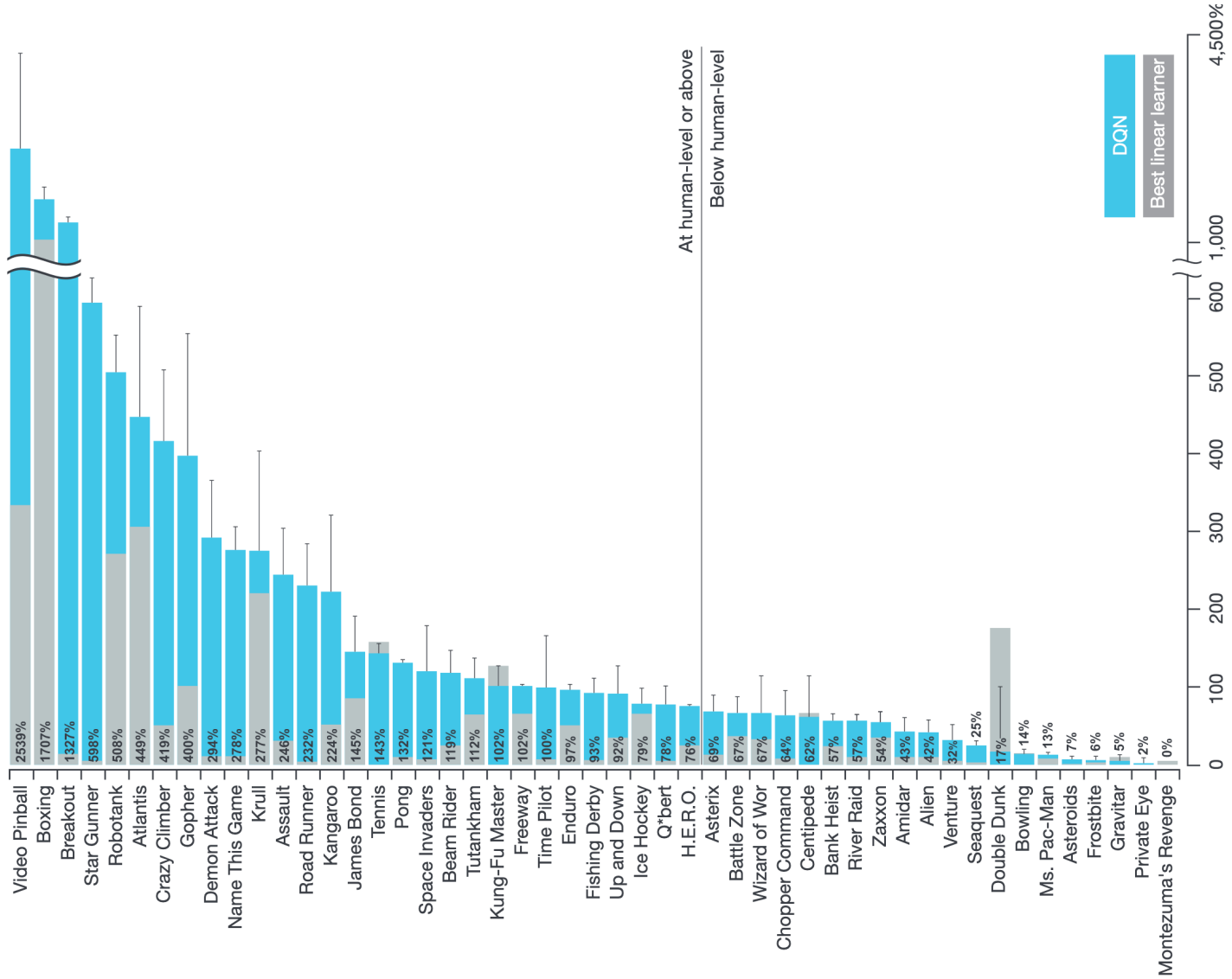
- Value fluctuation due to discounting as well as nature of imperfect function approximation



DQN: Action-Value Behavior



DQN: Performance on Atari



DQN: Ablation Study

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN: Ablation Study

- How do deep neural networks compare with a linear function approximator?

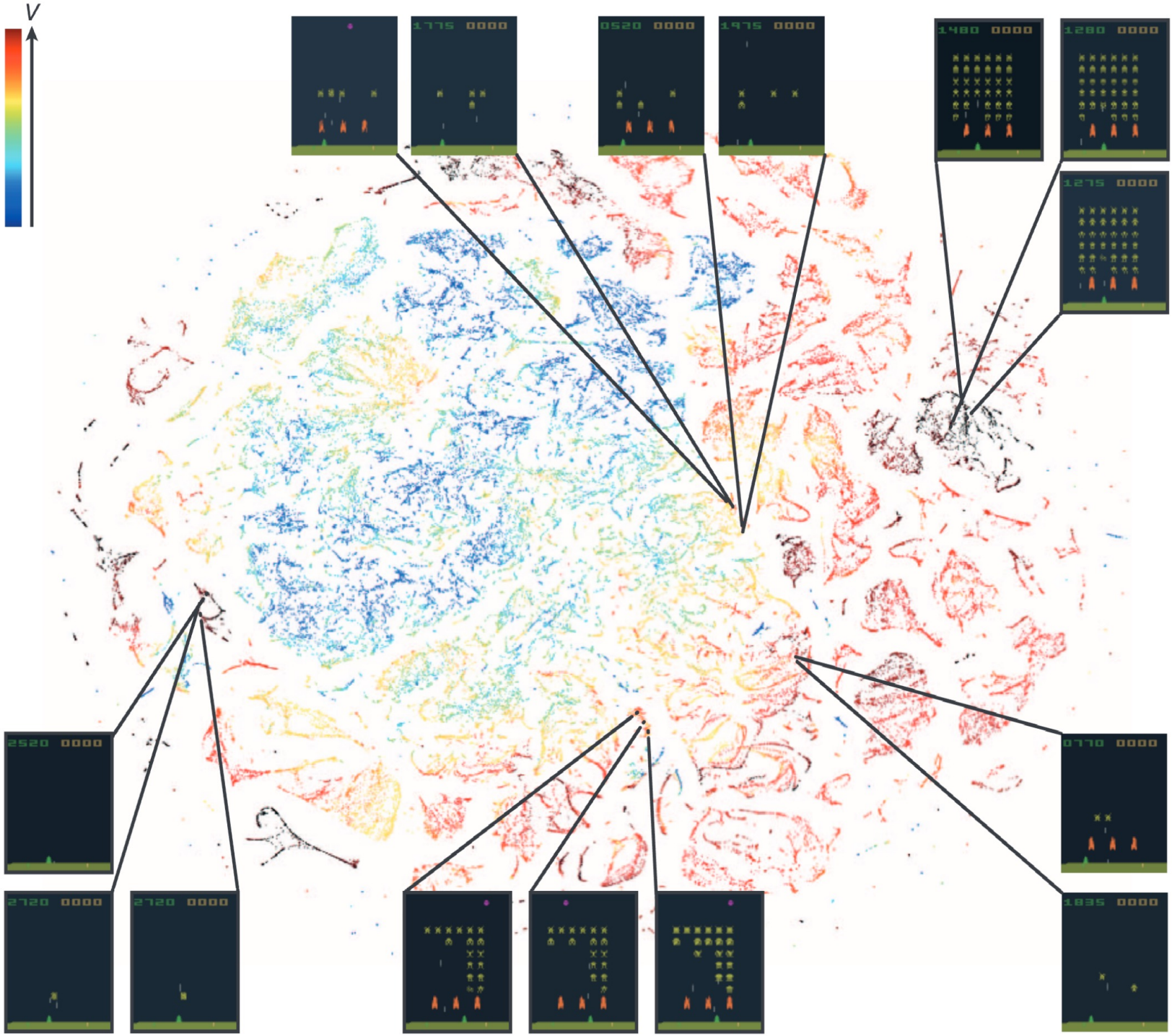
Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3

DQN: Hyperparameters

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.

DQN: Last Hidden Layer Activations

- Done for space invaders
- Two-dimensional projection using t-SNE
- Semantically similar states are clustered next to each other



DQN: Last Hidden Layer Activations

- How does network generalize to states obtained under other policies?
  - Human: Orange
  - AI Agent: Blue