THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Applied Machine Intelligence and Reinforcement Learning

Professor Hamza F. Alsarhan

SEAS 8505

Lecture 9

August 10, 2024

Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (Raise Hand). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, *be sure to mute yourself again*.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class for their own private use. Releasing these recordings is strictly prohibited.

Agenda

- Reinforcement Learning
- Deep Reinforcement Learning
- Deep Reinforcement Learning Applications
- Homework Overview

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Reinforcement Learning

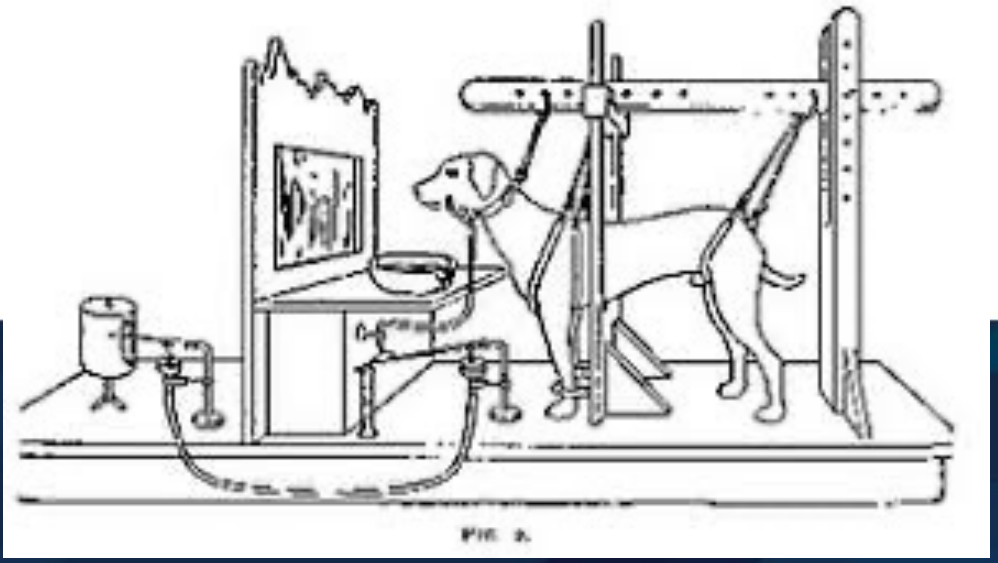
Professor Hamza F. Alsarhan
SEAS 8505

What is reinforcement learning?

In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward (or penalty) for its actions in trying to solve a problem.

After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.

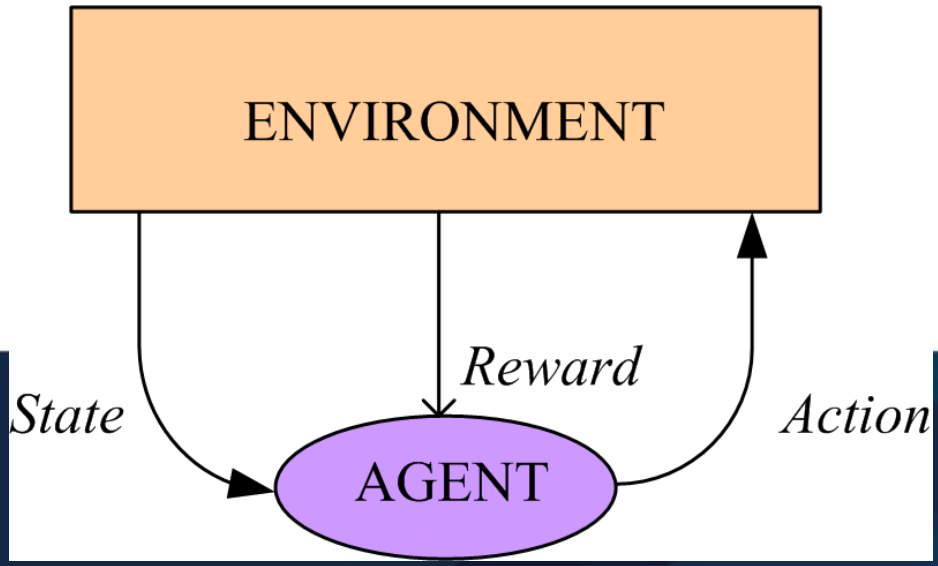
Pavlov's Dogs:



Machine Learning, Tom Mitchell, CMU

Basic Applications and Properties

- Game-playing: Sequence of moves to win a game
- Robot in a maze: Sequence of actions to find a goal
- **Agent** has a **state** in an environment, takes an **action** and sometimes receives **reward** and the state changes
- Credit-assignment
- Learn a policy



Control Learning

Consider learning to choose actions, e.g.,

- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output
- Learning to play Backgammon

Note several problem characteristics:

- Delayed reward
- Opportunity for active exploration
- Possibility that state only partially observable
- Possible need to learn multiple tasks with same sensors/effectors

Machine Learning, Tom Mitchell, CMU

One Example: TD-Gammon

[Tesauro, 1995]

Learn to play Backgammon

Immediate reward

- +100 if win
- -100 if lose
- 0 for all other states

Trained by playing 1.5 million games against itself

Now approximately equal to best human player

RL Overview

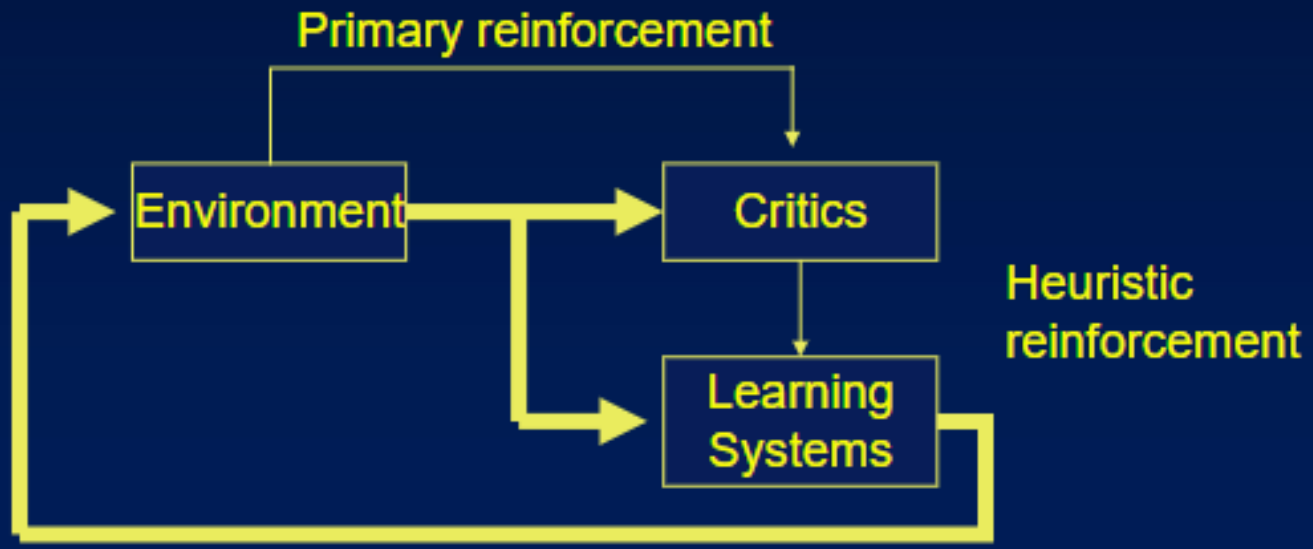
- Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.
- This very generic problem covers tasks such as learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games.
- Each time the agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state.
- For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states.
- The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.
- We are going to talk about an algorithm called ***Q*** learning that can acquire optimal control strategies from delayed rewards, even when the agent has no prior knowledge of the effects of its actions on the environment.
- Reinforcement learning algorithms are related to dynamic programming algorithms frequently used to solve optimization problems

Machine Learning, Tom Mitchell, CMU

Learning without a Teacher

- Reinforcement learning
  - No teacher to provide direct (desired) response at each step
    - example : good/bad, win/lose (But human feedback can help)
  - must solve temporal credit assignment problem
    - since critics may be given at the time of final output

More Recent Finding



Context of Reinforcement Learning

- Reinforcement Learning (RL) is one of the most exciting fields of Machine Learning today, and also one of the oldest
- It has been around since the 1950s, producing many interesting applications over the years, particularly in games and in machine control
- A revolution took place in 2013, when researchers from a British startup called DeepMind demonstrated a system that could learn to play just about any Atari game from scratch, eventually outperforming humans in most of them, using only raw pixels as inputs and without any prior knowledge of the rules of the games.
- This was the first of a series of amazing feats, culminating in March 2016 with the victory of their system AlphaGo against Lee Sedol, a legendary professional player of the game of Go, and in May 2017 against Ke Jie, the world champion
  - So how did DeepMind achieve all this? With hindsight it seems rather simple: they applied the power of Deep Learning to the field of Reinforcement Learning, and it worked beyond their wildest dreams.

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

Markov Decision Processes

- MDP has a fixed number of states, and it randomly evolves from one state to another at each step. The probability for it to evolve from a state s to a state s′ is fixed, and it depends only on the pair (s, s′), not on past states (this is why we say that the system has no memory).

- The algorithms are less direct: the agent learns to estimate the expected return for each state, or for each action in each state, then it uses this knowledge to decide how to act.

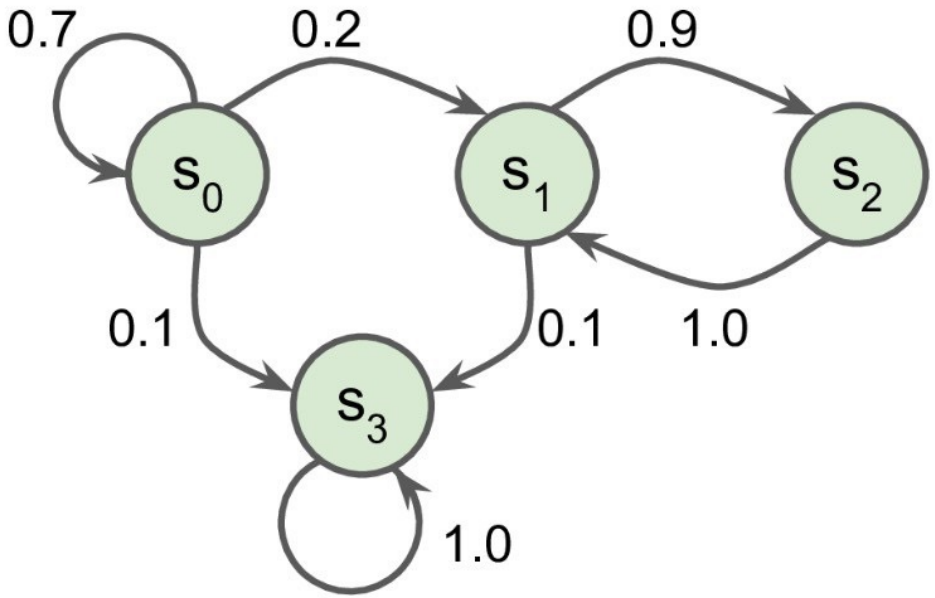


Figure 18-7. Example of a Markov chain

Markov Decision Processes

- At each step, an agent can choose one of several possible actions, and the transition probabilities depend on the chosen action.
- Moreover, some state transitions return some reward (positive or negative).
- The agent's goal is to find a policy that will maximize reward over time.

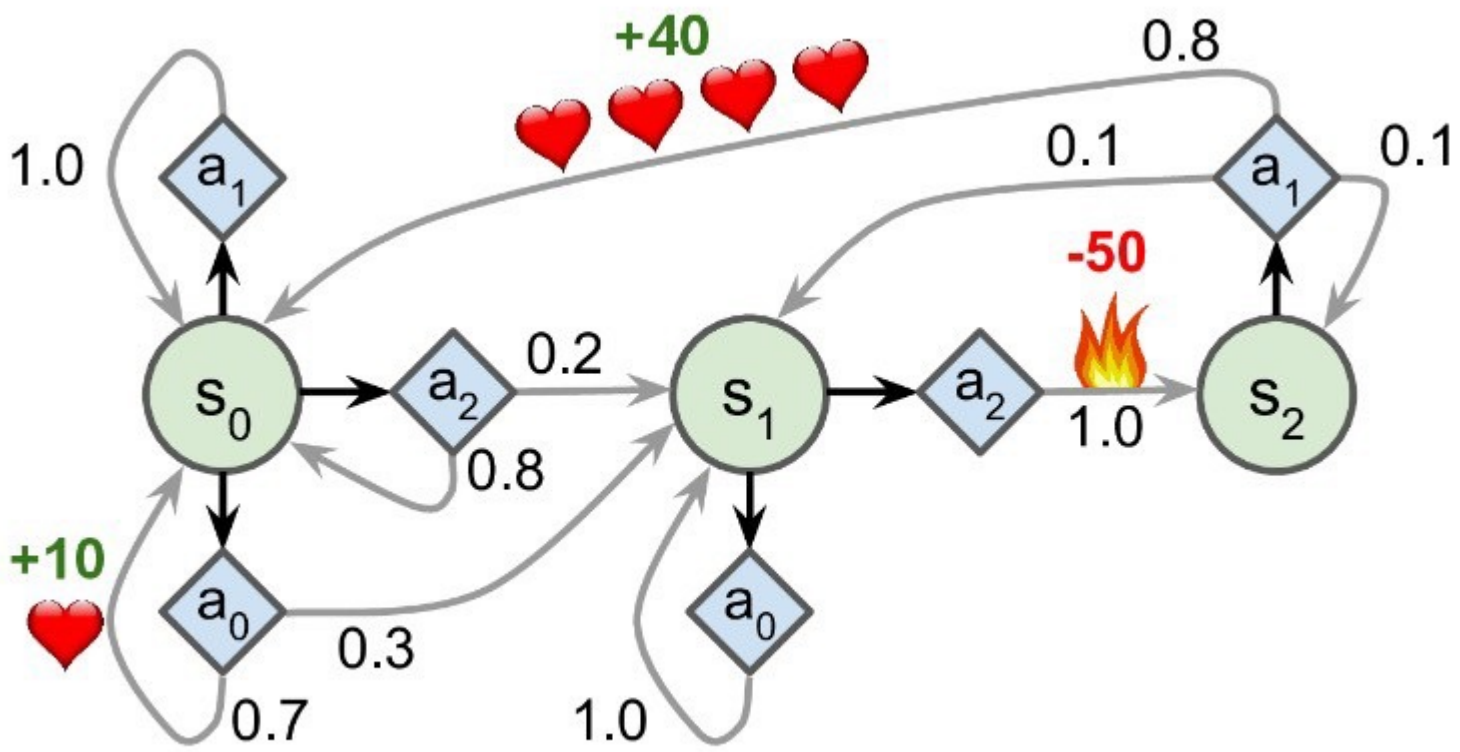


Figure 18-8. Example of a Markov decision process

Bellman's Equations

Equation 18-1. Bellman optimality equation

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot V^*(s')\right] \quad \text{for all } s$$

In this equation:

- $T(s, a, s')$ is the transition probability from state s to state s', given that the agent chose action a. For example, in Figure 18-8, $T(s_2, a_1, s_0) = 0.8$.
- $R(s, a, s')$ is the reward that the agent gets when it goes from state s to state s', given that the agent chose action a. For example, in Figure 18-8, $R(s_2, a_1, s_0) = +40$.
- γ is the discount factor.

Equation 18-2. Value iteration algorithm

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot V_k(s')\right] \quad \text{for all } s$$

In this equation, $V_k(s)$ is the estimated value of state s at the k^{th} iteration of the algorithm.

Equation 18-3. Q-value iteration algorithm

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right] \quad \text{for all } (s,a)$$

Once you have the optimal Q-values, defining the optimal policy, noted $\pi^*(s)$, is trivial; when the agent is in state s, it should choose the action with the highest Q-value for that state: $\pi^*(s) = \underset{a}{\text{argmax}}\, Q^*(s,a)$.

Learning to Optimize Rewards

- In Reinforcement Learning, a software agent makes observations and takes actions within an environment, and in return it receives rewards
- Its objective is to learn to act in a way that will maximize its expected rewards over time.
- In short, the agent acts in the environment and learns by trial and error to maximize its pleasure and minimize its pain.

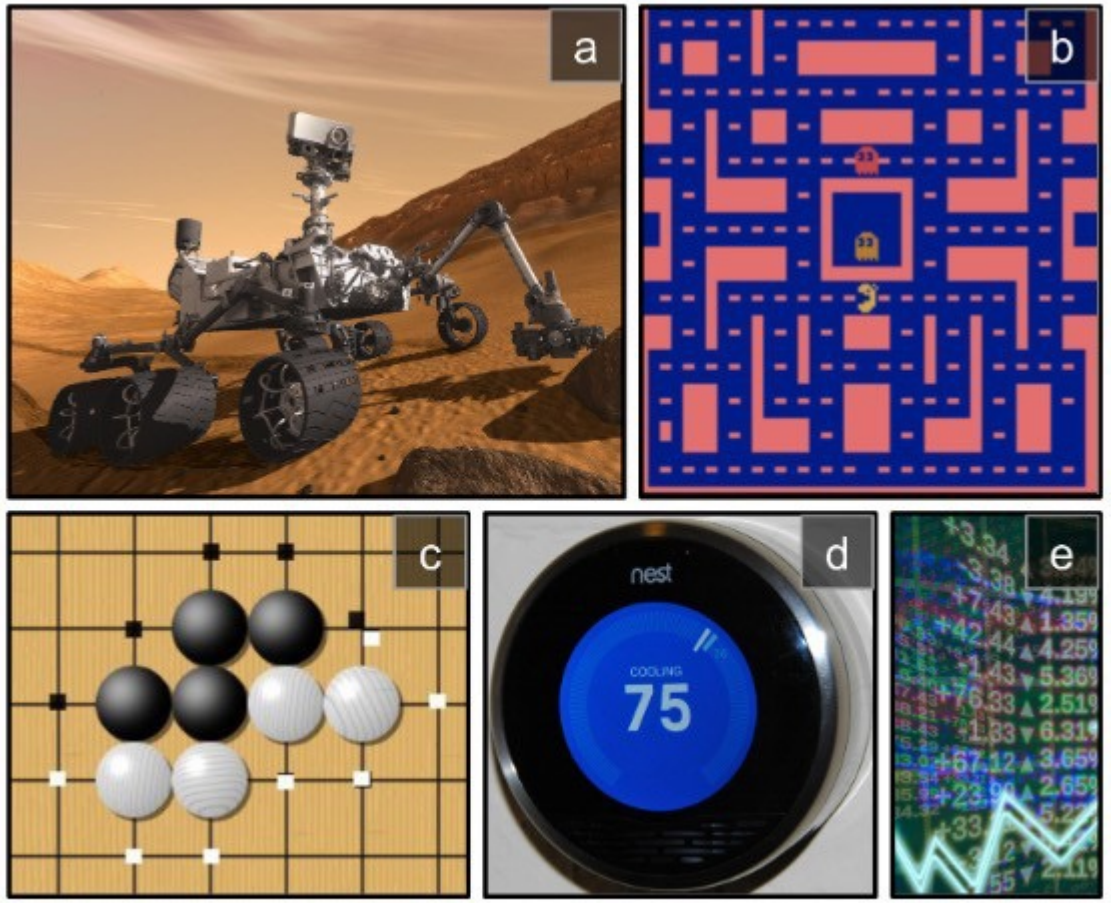
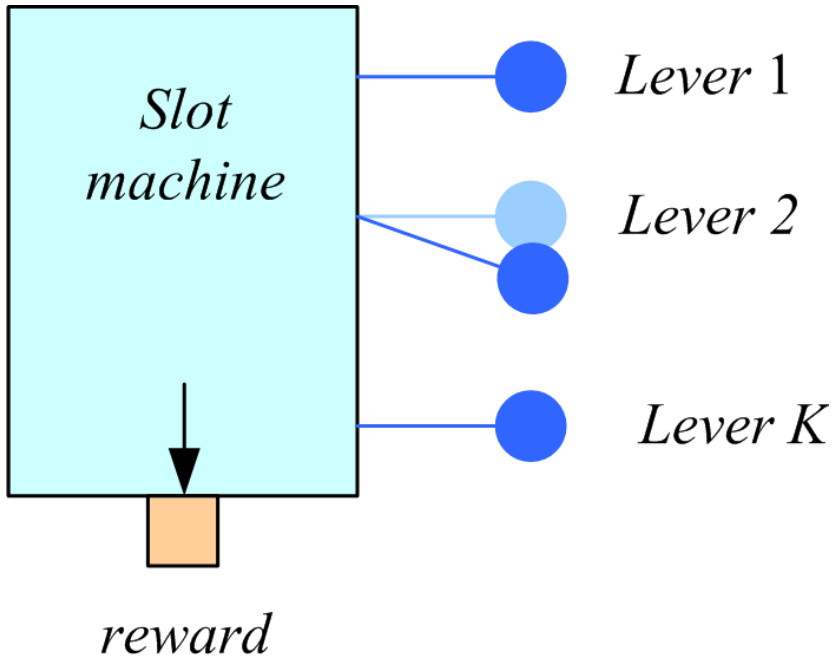


Figure 18-1. Reinforcement Learning examples: (a) robotics, (b) *Ms. Pac-Man*, (c) Go player, (d) thermostat, (e) automatic trader[5]

Single State: K-armed Bandit

- Among K levers, choose the one that pays best
  - $Q(a)$: value of action a
  - Reward is r_a
  - Set $Q(a) = r_a$
  - Choose a^* if:
    - $Q(a^*)=\max_a Q(a)$



- Rewards stochastic (keep an *expected* reward):

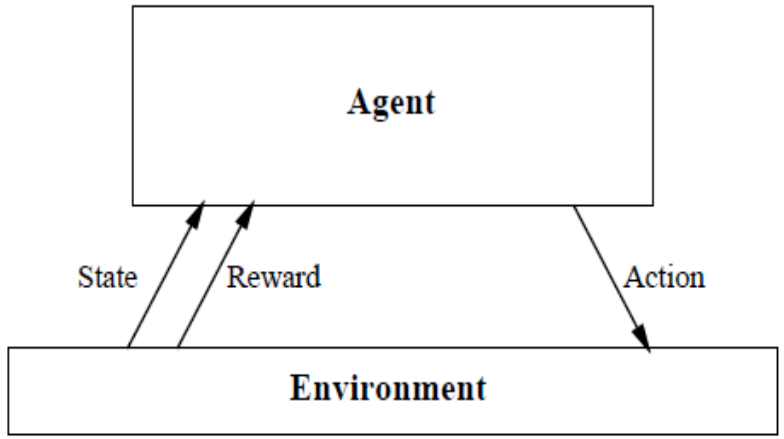
$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta\left[r_{t+1}(a) - Q_t(a)\right]$$

Reinforcement Learning Problem

An agent interacting with its environment. The agent exists in an environment described by some set of possible states *S.*
It can perform any of a set of possible actions A.

Each time it performs an action a_i in some state s_t the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states s_i, actions a_i, and immediate rewards r_i as shown in the figure.

The agent's task is to learn a control policy, π: S→ A, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \text{, where } 0 \leq \gamma < 1$$

Machine Learning, Tom Mitchell, CMU

Q-Values (Quality Values)

- Knowing the optimal state values can be useful, in particular to evaluate a policy, but it does not give us the optimal policy for the agent.
- **Q-Values (Quality Values)** - the optimal Q-Value of the state-action pair (s, a), noted Q*(s, a), is the sum of discounted future rewards the agent can expect on average after it reaches the state s and chooses action a, but before it sees the outcome of this action, assuming it acts optimally after that action.
- Here is how it works:
  - Start by initializing all the Q-Value estimates to zero, then you update them using the Q-Value Iteration algorithm
  - Once you have the optimal Q-Values, defining the optimal policy, noted π*(s), is trivial: when the agent is in state s, it should choose the action with the highest Q-Value for that state

Off-Policy Algorithm

- The Q-Learning algorithm is called an **off-policy algorithm** because the policy being trained is not necessarily the one being executed:

- For example. the policy being executed (the exploration policy) could be completely random, while the policy being trained will always choose the actions with the highest Q- Values.

- Q-Learning is capable of learning the optimal policy by just watching an agent act randomly.

Approximate Q-Learning and Deep Q-Learning

- The main problem with Q-Learning is that it does not scale well to large (or even medium) MDPs with many states and actions.

- The solution is to find a function that approximates the Q-Value of any state-action pair (s, a) using a manageable number of parameters (given by the parameter vector θ). This is called **Approximate Q-Learning.**

- Deep neural networks can work much better, especially for complex problems, and it does not require any feature engineering.
  - A DNN used to estimate Q-Values is called a Deep Q-Network (DQN), and using a DQN for Approximate Q-Learning is called Deep Q-Learning.

Q-Learning

- Q-Learning works by watching an agent play (e.g., randomly) and gradually improving its estimates of the Q-Values
- Once it has accurate Q-Value estimates (or close enough), then the optimal policy is choosing the action that has the highest Q-Value (i.e., the greedy policy).
- This algorithm will converge to the optimal Q-Values, but it will take many iterations, and possibly quite a lot of hyperparameter tuning.

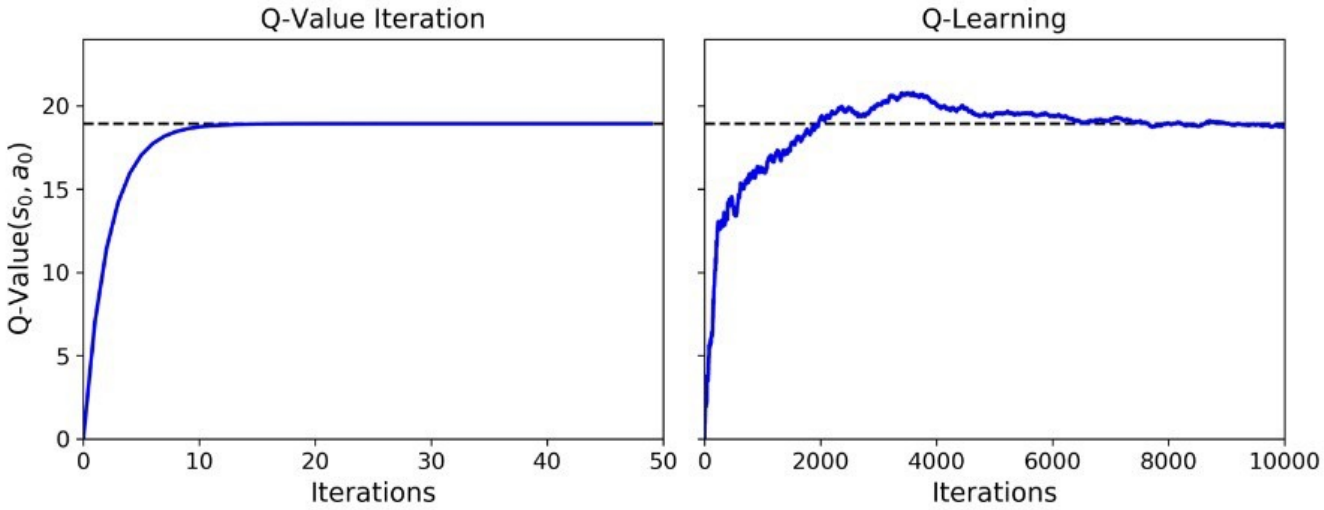


Figure 18-9. The Q-Value Iteration algorithm (left) versus the Q-Learning algorithm (right)

Q Function

Define new function very similar to V^*

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

If agent learns Q, it can choose optimal action even without knowing δ!

$$\pi^*(s) = \operatorname*{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

$$\pi^*(s) = \operatorname*{argmax}_a Q(s,a)$$

Q is the evaluation function the agent will learn

Q-learning

Initialize all $Q(s, a)$ arbitrarily

For all episodes

  Initalize s

  Repeat

    Choose a using policy derived from Q, e.g., ϵ-greedy

    Take action a, observe r and s'

    Update $Q(s, a)$:

      $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$

    $s \leftarrow s'$

  Until s is terminal state

Machine Learning,
Tom Mitchell, CMU

https://www.youtube.com/watch?v=9m_6q_KECTk

Reinforcement Learning (RL): Key Concepts

Agent: takes actions.

MIT 6.S191 Introduction to Deep Reinforcement Learning

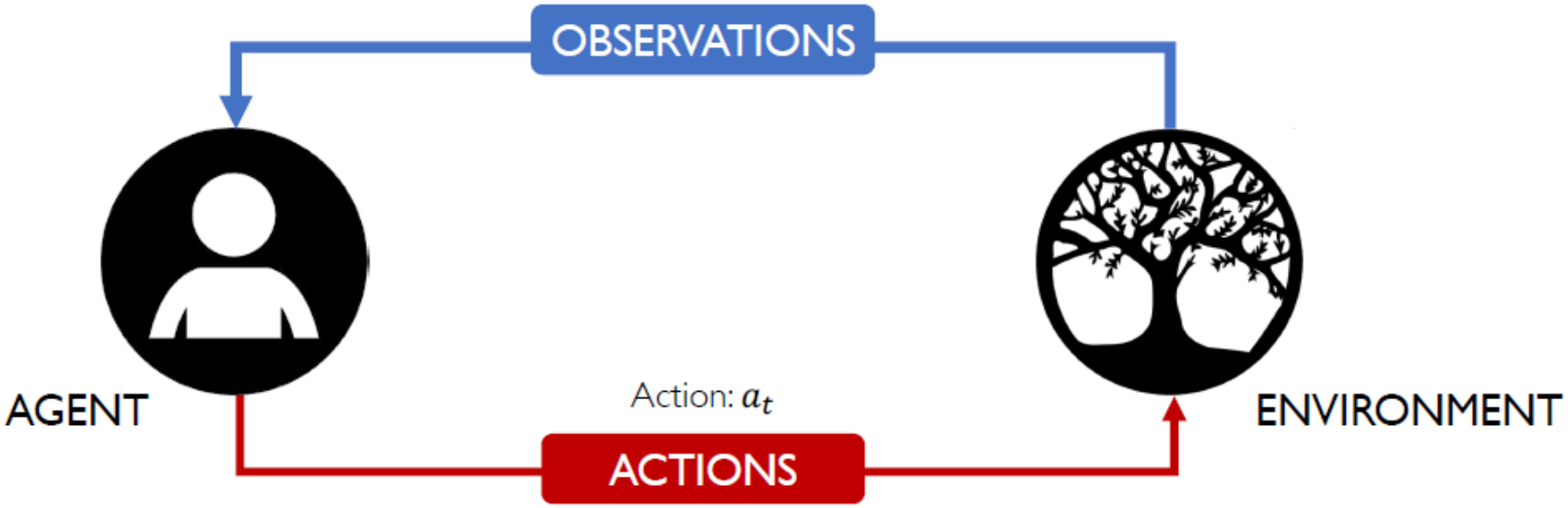
Reinforcement Learning (RL): Key Concepts

Environment: the world in which the agent exists and operates.

Reinforcement Learning (RL): Key Concepts

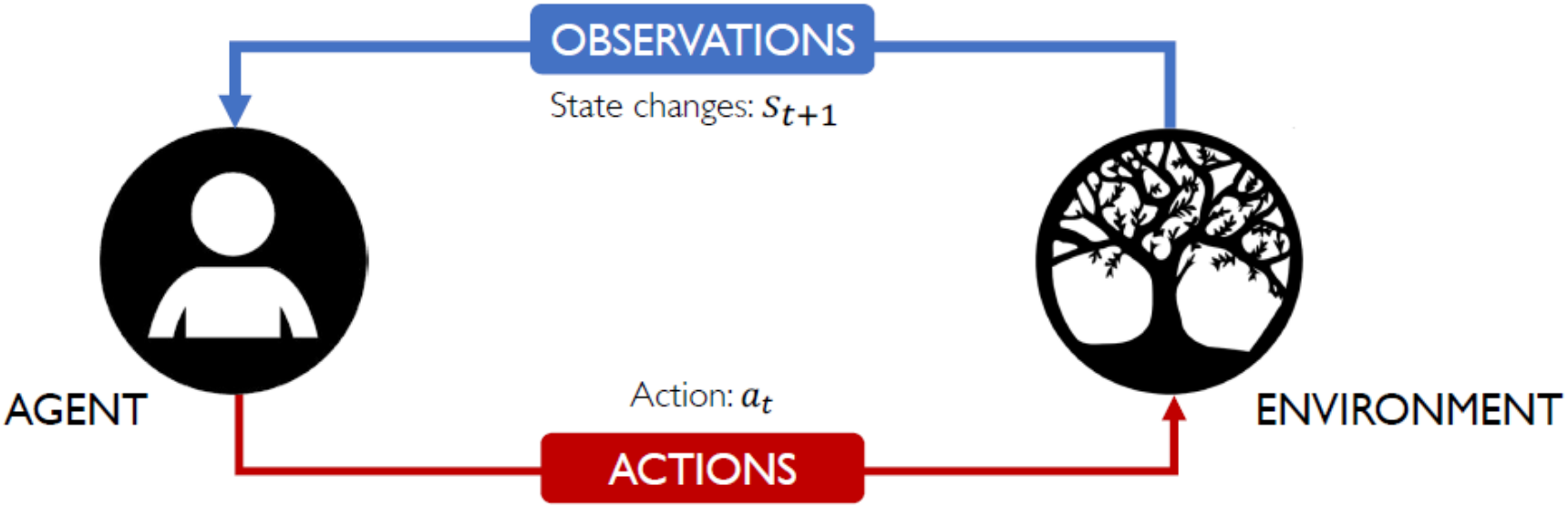
Action: a move the agent can make in the environment.

Reinforcement Learning (RL): Key Concepts



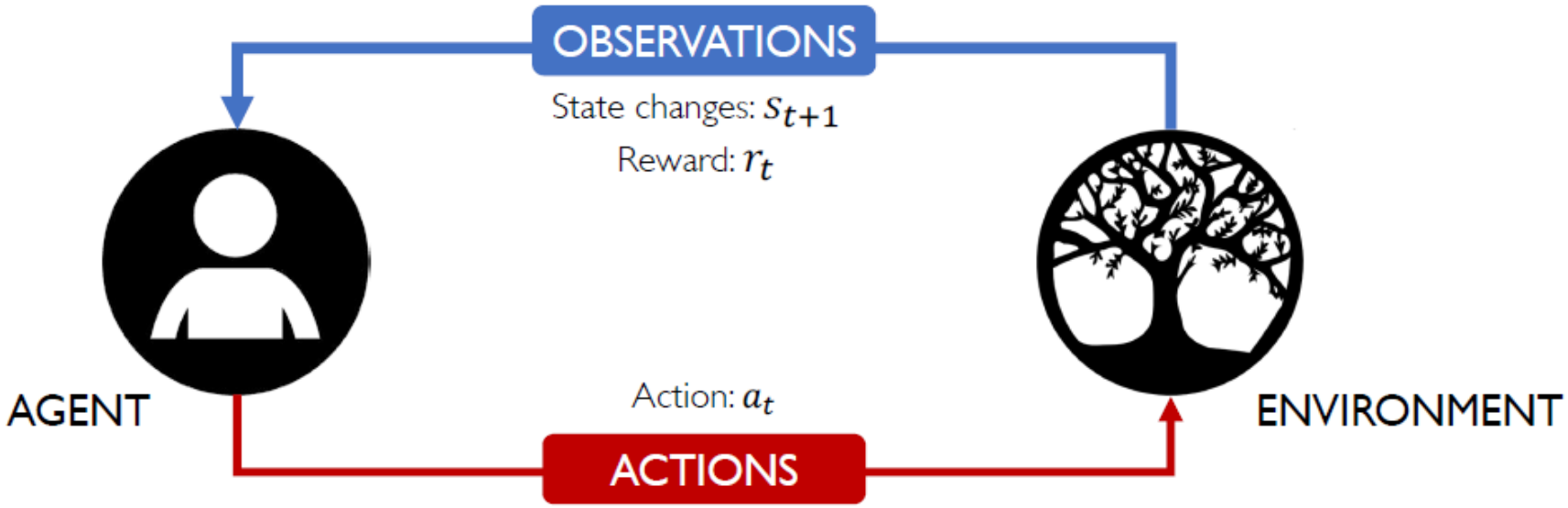
Observations: of the environment after taking actions.

Reinforcement Learning (RL): Key Concepts



State: a situation which the agent perceives.

Reinforcement Learning (RL): Key Concepts



Reward: feedback that measures the success or failure of the agent's action.

Reinforcement Learning (RL): Key Concepts

Total Reward

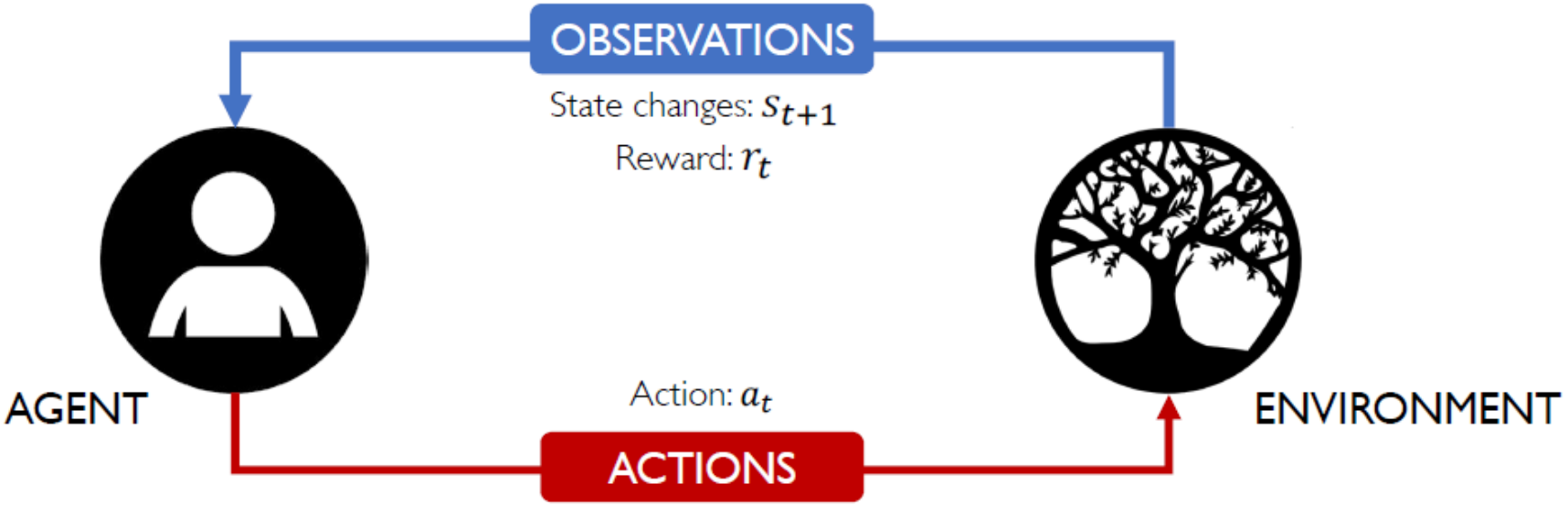
$$R_t = \sum_{i=t}^{\infty} r_i$$

Reinforcement Learning (RL): Key Concepts

Total Reward

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} \ldots + r_{t+n} + \cdots$$

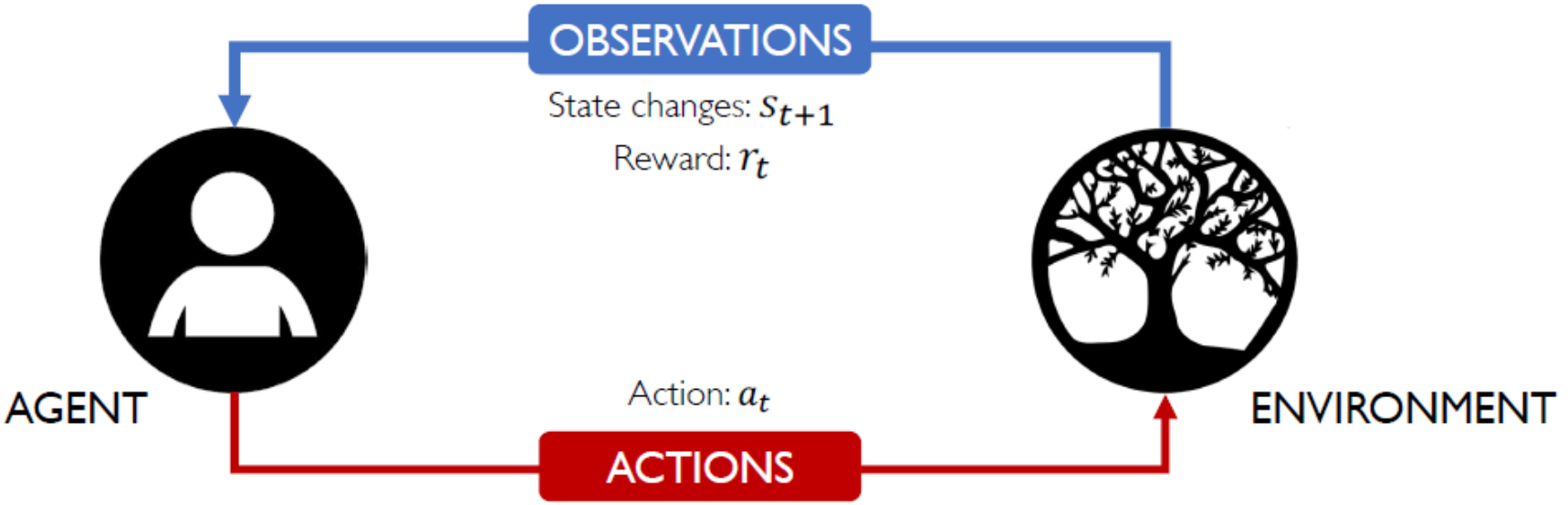
Reinforcement Learning (RL): Key Concepts



Discounted Total Reward

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i$$

Reinforcement Learning (RL): Key Concepts



Discounted Total Reward

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{t+1} r_{t+1} \ldots + \gamma^{t+n} r_{t+n} + \cdots$$

γ: discount factor

Defining the Q-function

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

Total reward, R_t, is the discounted sum of all rewards obtained from time t

Defining the Q-function

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

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s, a) = \mathbb{E}[R_t]$$

The Q-function captures the **expected total future reward** an agent in state, s, can receive by executing a certain action, a

How to take actions given a Q-function?

$$Q(s, a) = \mathbb{E}[R_t]$$

(state, action)

How to take actions given a Q-function?

$$Q(s, a) = \mathbb{E}[R_t]$$

(state, action)

Ultimately, the agent needs a **policy $\pi(s)$**, to infer the **best action to take** at its state, s

How to take actions given a Q-function?

$$Q(s, a) = \mathbb{E}[R_t]$$

(state, action)

Ultimately, the agent needs a **policy $\pi(s)$**, to infer the **best action to take** at its state, s

Strategy: the policy should choose an action that maximizes future reward

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Introduction to OpenAI Gym

- One of the challenges of Reinforcement Learning is that in order to train an agent, you first need to have a working environment.
  - If you want to program an agent that will learn to play an Atari game, you will need an Atari game simulator.
  - If you want to program a walking robot, then the environment is the real world, and you can directly train your robot in that environment, but this has its limits: if the robot falls off a cliff, you can't just click Undo.
    - You can't speed up time either; adding more computing power won't make the robot move any faster. And it's generally too expensive to train 1,000 robots in parallel.
- Training is hard and slow in the real world, so **you generally need a simulated environment at least for bootstrap training**
- OpenAI Gym is a toolkit that provides a wide variety of simulated environments (Atari games, board games, 2D and 3D physical simulations, and so on), so you can train agents, compare them, or develop new RL algorithms.

Simple Environment - CartPole

- CartPole environment. This is a 2D simulation in which a cart can be accelerated left or right in order to balance a pole placed on top of it (see Figure 18-4)

- The CartPole problem is thus as simple as can be; the observations are noise-free, and they contain the environment's full state.

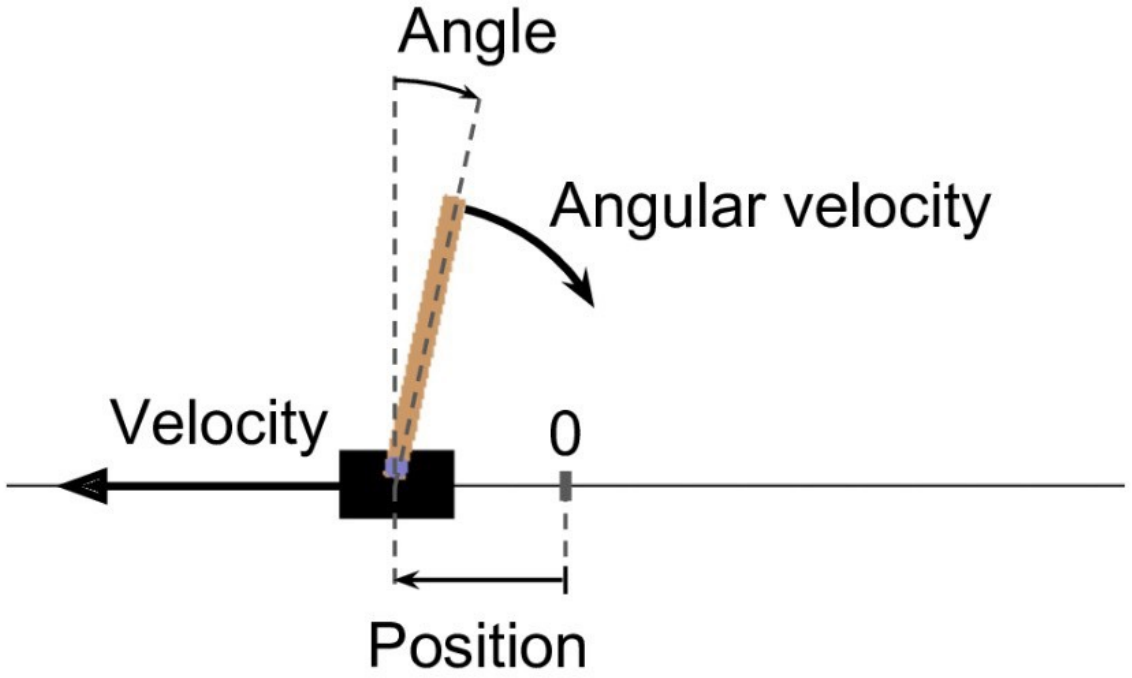


Figure 18-4. The CartPole environment

Policy

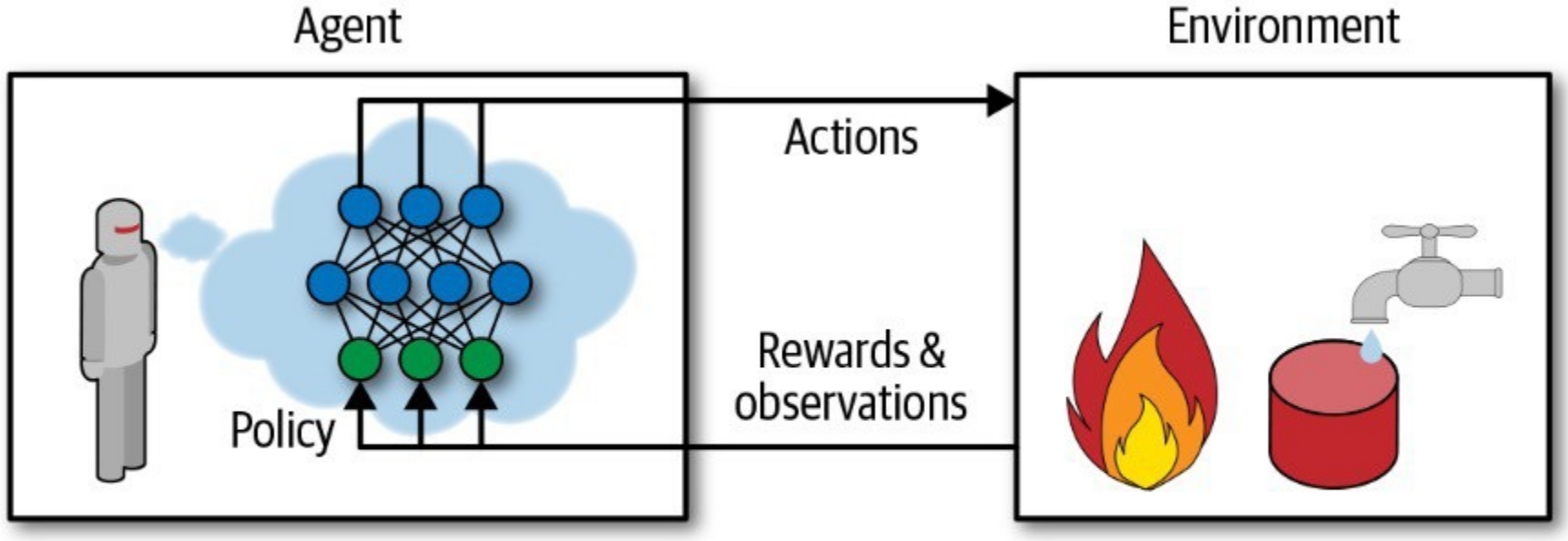


Figure 18-2. Reinforcement Learning using a neural network policy

- The algorithm a software agent uses to determine its actions is called its **policy**
- The policy could be a neural network taking observations as inputs and outputting the action to take

Policy

- The **policy** can be any algorithm you can think of, and it does not have to be deterministic (will always produce the same output if given the same input).
- For example, consider a robotic vacuum cleaner whose reward is the amount of dust it picks up in 30 minutes.
  - Its policy could be to move forward with some probability p every second, or randomly rotate left or right with probability 1 – p. The rotation angle would be a random angle between –r and +r. Since this policy involves some randomness, it is called a stochastic policy.
  - The robot will have an erratic trajectory, which guarantees that it will eventually get to any place it can reach and pick up all the dust.
  - ► The question is, how much dust will it pick up in 30 minutes?

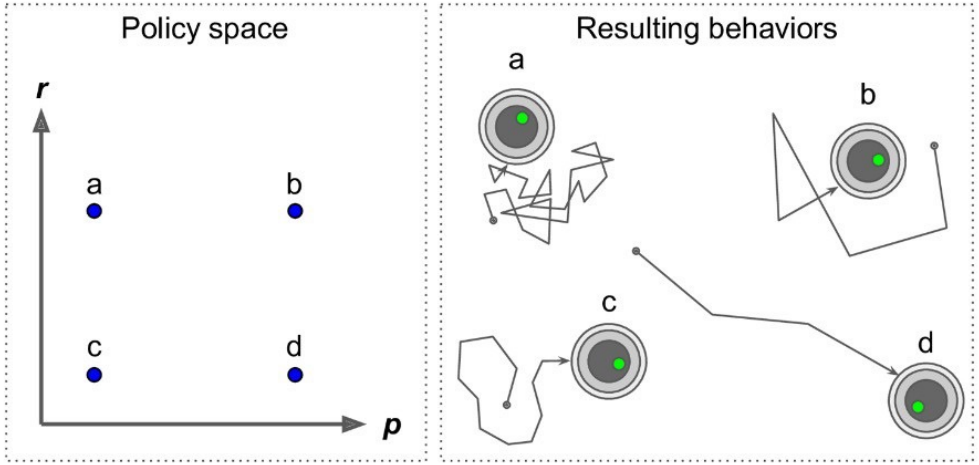


Figure 18-3. Four points in policy space (left) and the agent's corresponding behavior (right)

Finding a Policy

- Another way to explore the policy space is to use genetic algorithms.
  - For example, you could randomly create a first generation of 100 policies and try them out, then “terminate” the 80 worst policies and make the 20 survivors produce 4 offspring each. An offspring is a copy of its parent plus some random variation.
  - The surviving policies plus their offspring together constitute the second generation. You can continue to iterate.
- Yet another approach is to use optimization techniques, by evaluating the gradients of the rewards with regard to the policy parameters, then tweaking these parameters by following the gradients toward higher rewards.
- Going back to the vacuum cleaner robot, you could slightly increase p and evaluate whether doing so increases the amount of dust picked up by the robot in 30 minutes; if it does, then increase p some more, or else reduce p through generations this way until you find a good policy.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Deep Reinforcement Learning

Professor Hamza F. Alsarhan
SEAS 8505

Deep Reinforcement Learning

- Deep Reinforcement Learning (DRL) is a subfield of machine learning and reinforcement learning (RL) that combines the principles of RL with deep learning techniques, particularly deep neural networks.

- In DRL, agents learn to make sequential decisions by interacting with an environment in order to maximize cumulative rewards, and they utilize deep neural networks to represent complex functions that map observations to actions or value estimates.
- The key components of DRL include:

1. **Environment**: The environment in which the agent operates, providing observations, rewards, and feedback in response to the agent's actions.
2. **Agent**: The learning agent that interacts with the environment. The agent's objective is typically to learn a policy (mapping from states or observations to actions) or a value function (estimating the expected cumulative rewards).
3. **Deep Neural Networks**: Deep learning architectures, particularly deep neural networks, are used to approximate complex functions such as policies or value functions. These networks are capable of learning hierarchical representations from raw sensory inputs (e.g., images or sensor data) and can effectively capture intricate patterns and relationships in high-dimensional data.
4. **Training Algorithms**: DRL algorithms train the agent's neural network model by optimizing certain objectives, such as maximizing expected rewards or minimizing the difference between estimated and actual values. Common DRL algorithms include Deep Q-Networks (DQN), Policy Gradient methods (e.g., REINFORCE), Actor-Critic methods, and Proximal Policy Optimization (PPO), among others.

Applications of Deep Reinforcement Learning

DRL has achieved remarkable success in a variety of domains, including but not limited to:

- Playing complex board games such as Go, chess, and shogi, where DRL agents have surpassed human performance.
- Controlling robotic systems for tasks like manipulation, locomotion, and navigation.
- Managing resource allocation and decision-making in finance, logistics, and supply chain management.
- Personalizing recommendations and content delivery in online platforms and digital media.

Overall, DRL represents a *powerful and versatile* approach to *learning sequential decision-making tasks in complex and high-dimensional environments*, and it continues to drive advancements in artificial intelligence research and applications.

Neural Network Policies

- A neural network can take an observation as input, and output the action to be executed
- More precisely, it can estimate a probability for each action, and then we select an action randomly, according to the estimated probabilities

```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(5, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid"),
])
```

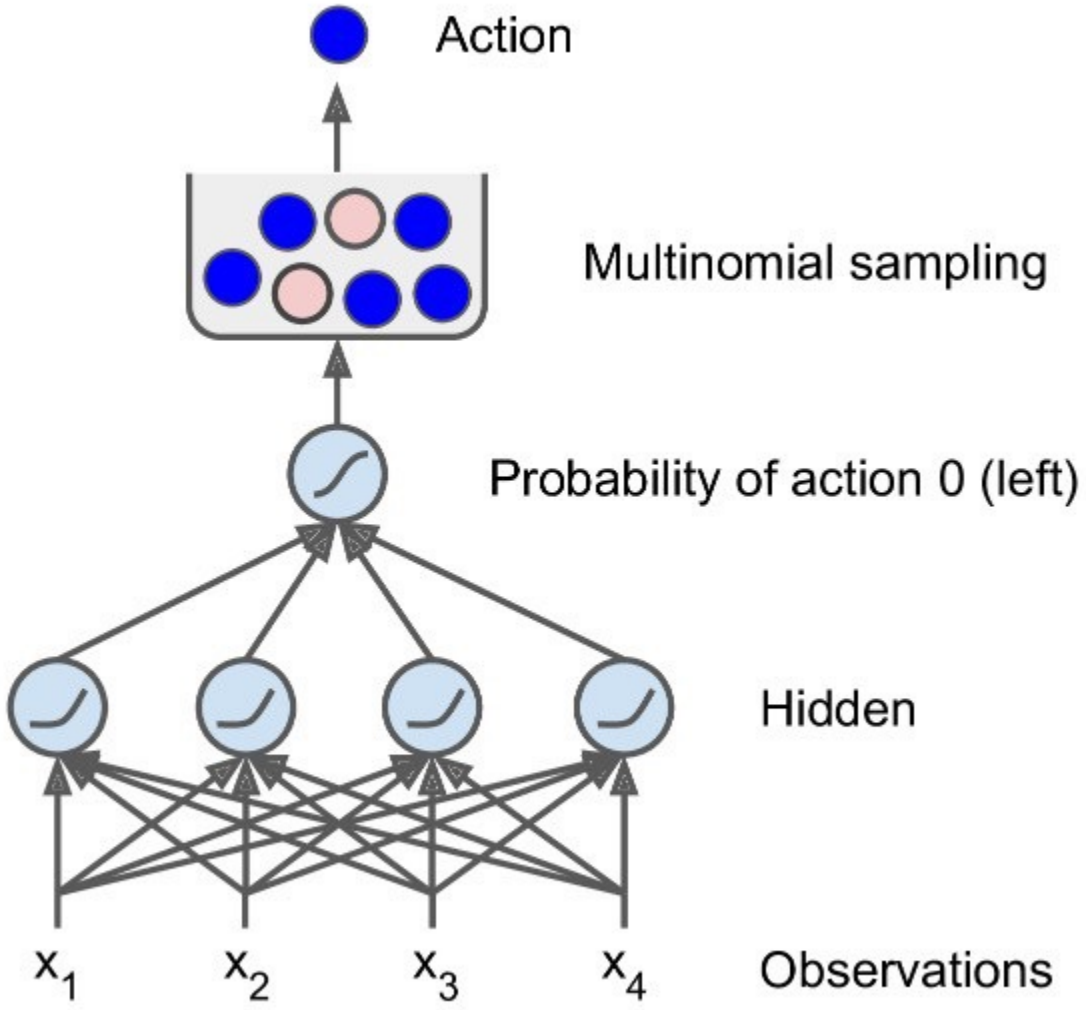


Figure 18-5. Neural network policy

For Consideration

- Why pick a random action based on the probabilities given by the neural network, rather than just picking the action with the highest score?
  - This approach lets the agent find the right balance between exploring new actions and exploiting the actions that are known to work well.
  - Suppose you go to a restaurant for the first time, and all the dishes look equally appealing, so you randomly pick one.
    - If it turns out to be good, you can increase the probability that you will order it next time.
    - However, you should not increase that probability up to 100%, or else you will never try out the other dishes, some of which may be even better than the one you tried.

Evaluating Actions: The Credit Assignment Problem

- How do we train a neural network policy?
- If we knew what the best action was at each step, we could train the neural network as usual, by minimizing the cross entropy between the estimated probability distribution and the target probability distribution.
  - It would just be regular supervised learning
- However, in Reinforcement Learning, the only guidance the agent gets is through rewards, and ***rewards are typically sparse and delayed***
- For example, if the agent manages to balance the pole for 100 steps, how can it know which of the 100 actions it took were good, and which of them were bad?
  - All it knows is that the pole fell after the last action, but surely this last action is not entirely responsible
  - This is called the **credit assignment problem**: when the agent gets a reward, it is hard for it to know which actions should get credited (or blamed) for it

Reward Discount Factor

- To tackle this problem, a common strategy is to evaluate an action based on the sum of all the rewards that come after it, usually applying a discount factor at each step
- This sum of discounted rewards is called the action's return
- In the example on the right, the first action's return is $10 + \gamma \times 0 + \gamma^2 \times (-50) = -22$, where γ is 80%
- How to determine the discount rate:
  - If the task involves actions with long-term consequences (e.g., chess, where sacrificing a piece now can lead to victory later), a lower discount rate may be appropriate.
  - If the task involves actions where only immediate outcomes matter (e.g., some video games), a higher discount rate may be suitable.

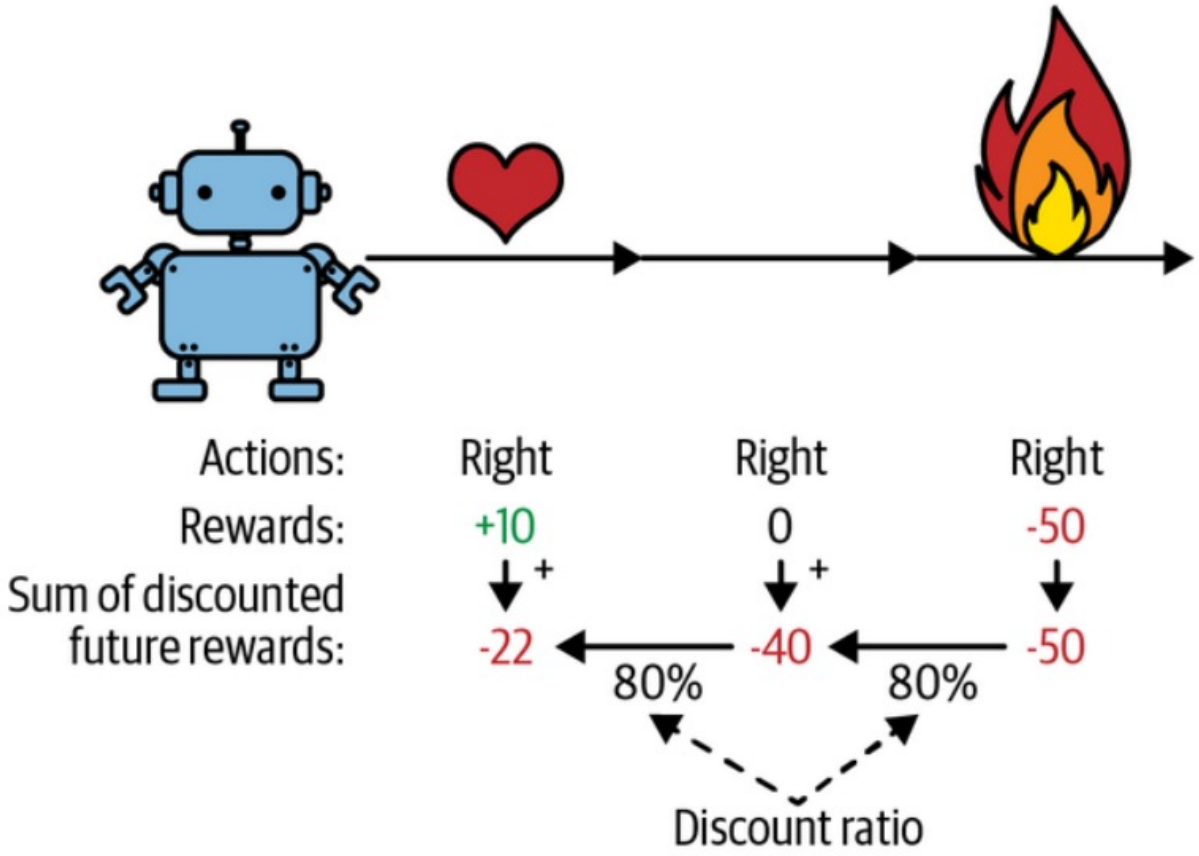


Figure 18-6. Computing an action's return: the sum of discounted future rewards

Action Advantage

- We want to estimate how much better or worse an action is, compared to the other possible actions, on average. This is called the action advantage.
  - A good action may be followed by several bad actions that cause the pole to fall quickly, resulting in the good action getting a low return.
  - If we play the game enough times, on average good actions will get a higher return than bad ones.
- For this, we must run many episodes and normalize all the action returns (by subtracting the mean and dividing by the standard deviation).
  - Can reasonably assume that actions with a negative advantage were bad while actions with a positive advantage were good.

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\text{argmax}}\, Q(s, a)$$

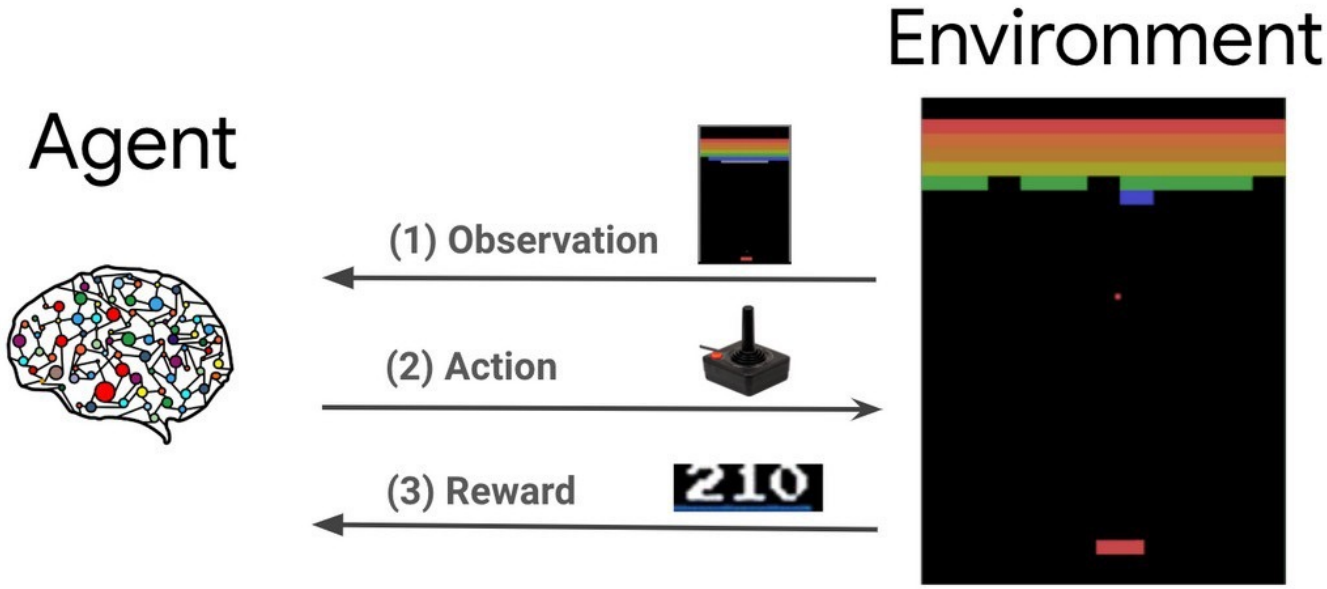
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

TensorFlow Intro to RL and Deep Q Networks

- The DQN (Deep Q-Network) algorithm was developed by DeepMind in 2015.
- It was able to solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale.
- The algorithm was developed by enhancing a classic RL algorithm called Q- Learning with deep neural networks and a technique called *experience replay*.



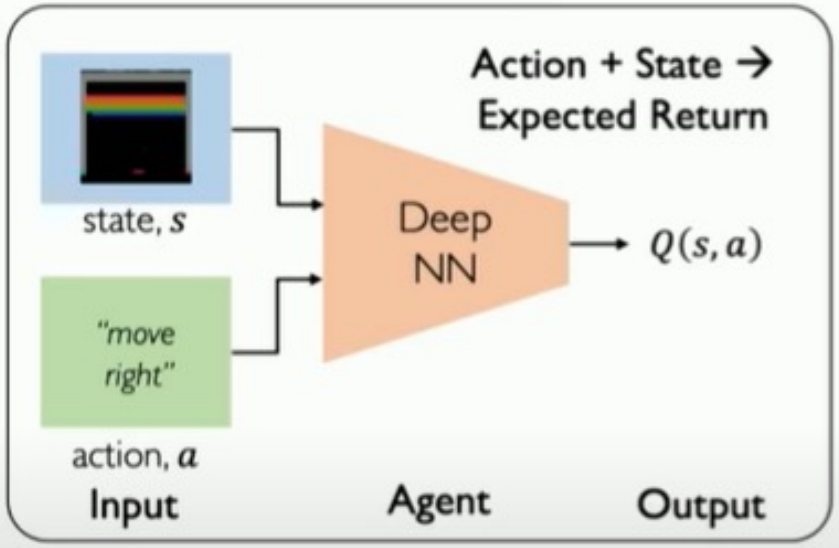
- DQN on Cartpole: TF-Agents provides all the components necessary to train a DQN agent, such as the agent itself, the environment, policies, networks, replay buffers, data collection loops, and metrics.

How Can We Train a DQN?

- Consider the approximate Q-Value computed by the DQN for a given state-action pair (s, a)
- We want this approximate Q-Value to be as close as possible to the reward r that we actually observe after playing action a in state s, plus the discounted value of playing optimally from then on.
- To estimate this sum of future discounted rewards, we can simply execute the DQN on the next state s′ and for all possible actions a′
  - We get an approximate future Q-Value for each possible action.
- We then pick the highest (since we assume we will be playing optimally) and discount it, and this gives us an estimate of the sum of future discounted rewards.
- By summing the reward r and the future discounted value estimate, we get a target Q-Value y(s, a) for the state-action pair (s, a)

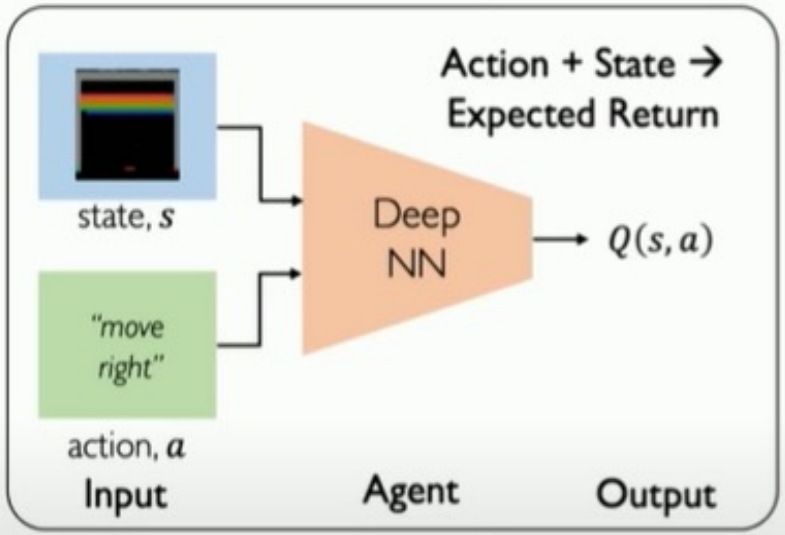
Deep Q Networks (DQN)

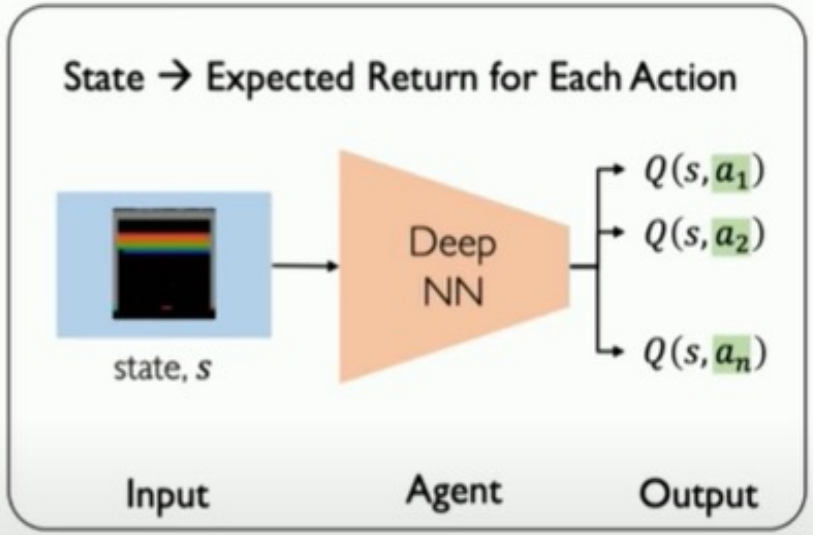
How can we use deep neural networks to model Q-functions?



Deep Q Networks (DQN)

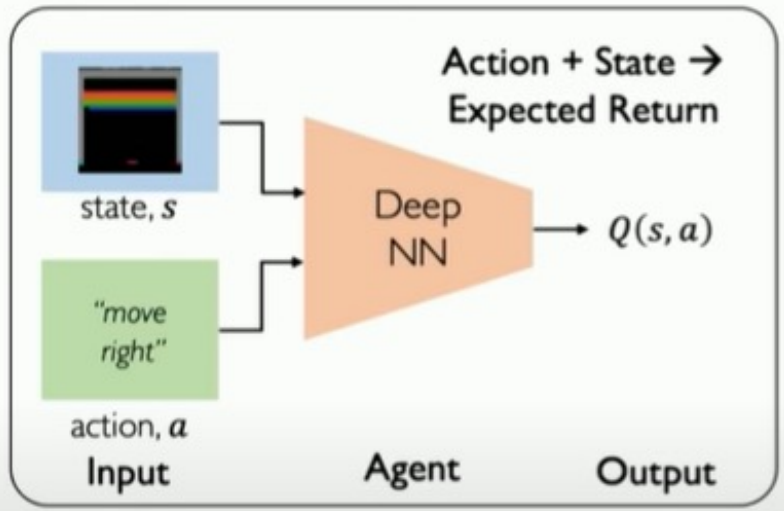
How can we use deep neural networks to model Q-functions?

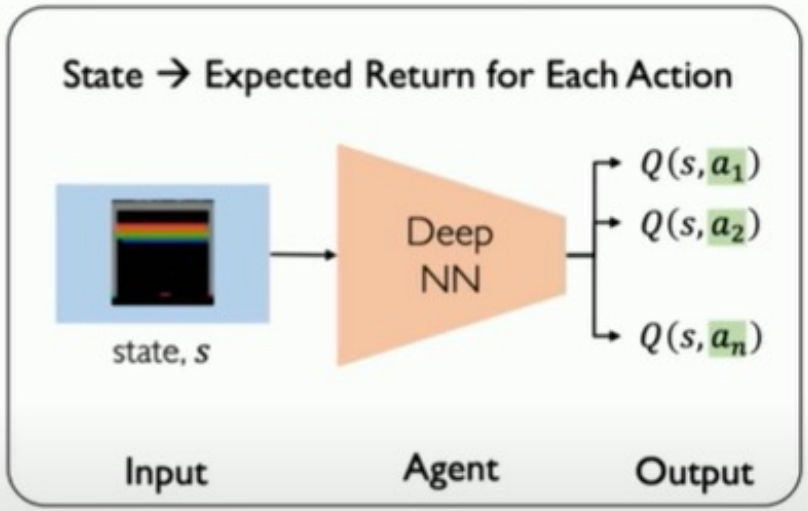




Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?

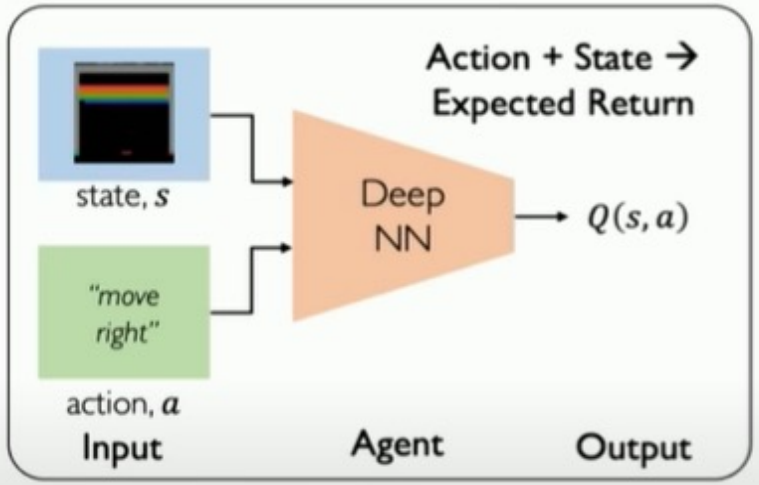


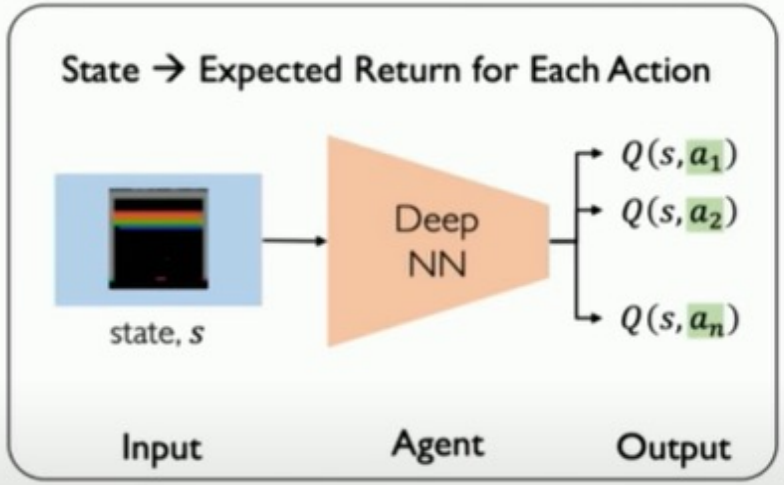


What happens if we take all the best actions?

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?



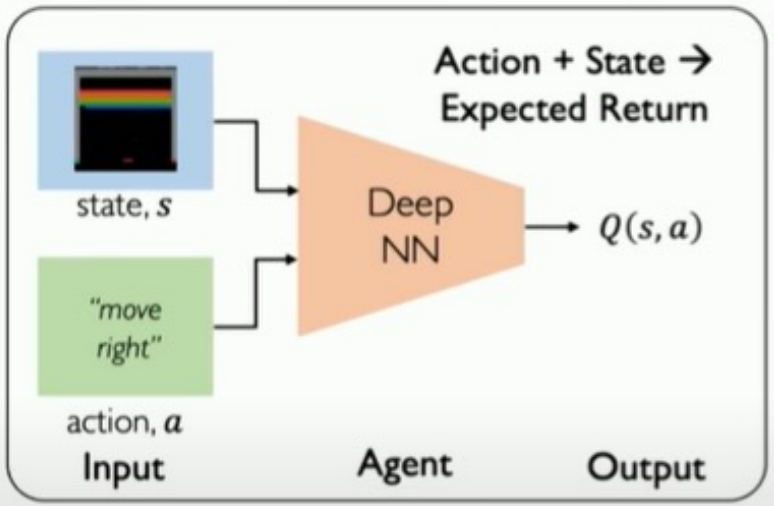


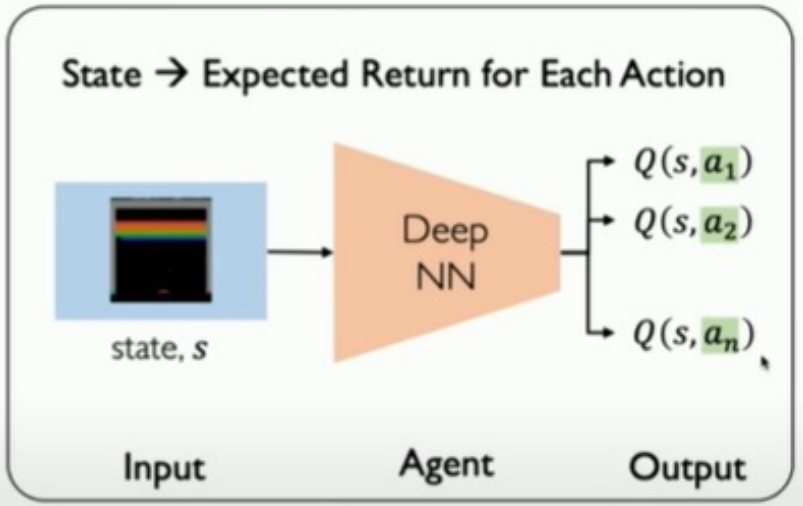
$$\text{target} \quad \left(r + \gamma \max_{a'} Q(s', a')\right)$$

Take all the best actions → target return

Deep Q Networks (DQN): Training

How can we use deep neural networks to model Q-functions?





target

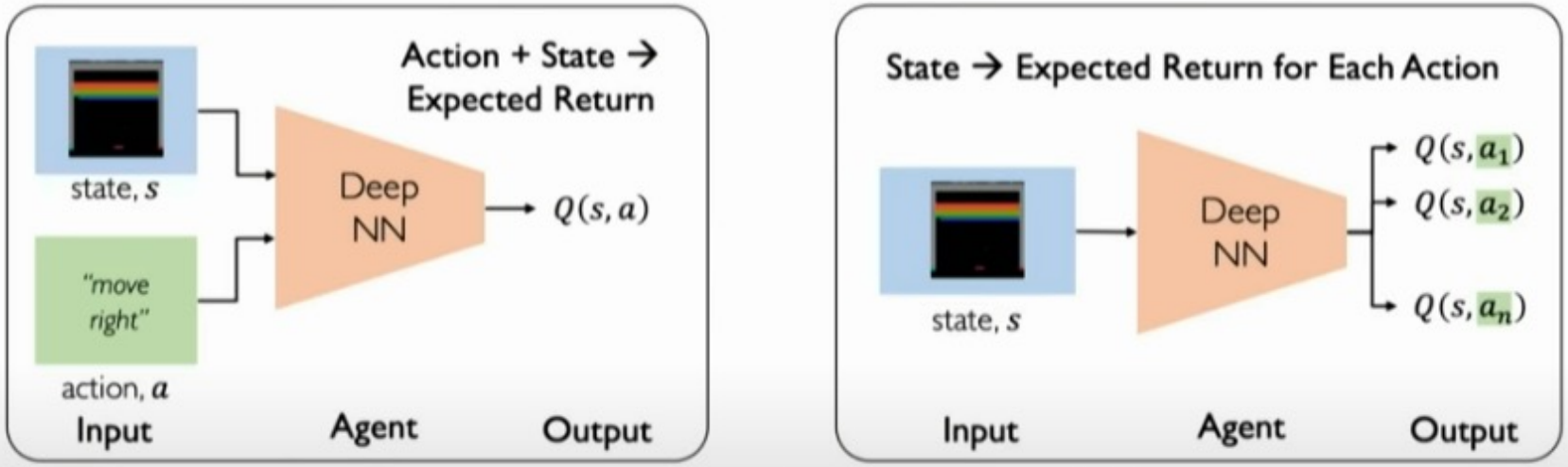
predicted

$$\left(r + \gamma \max_{a'} Q(s', a')\right) \qquad Q(s,a)$$

Network prediction

Deep Q Networks (DQN): Training

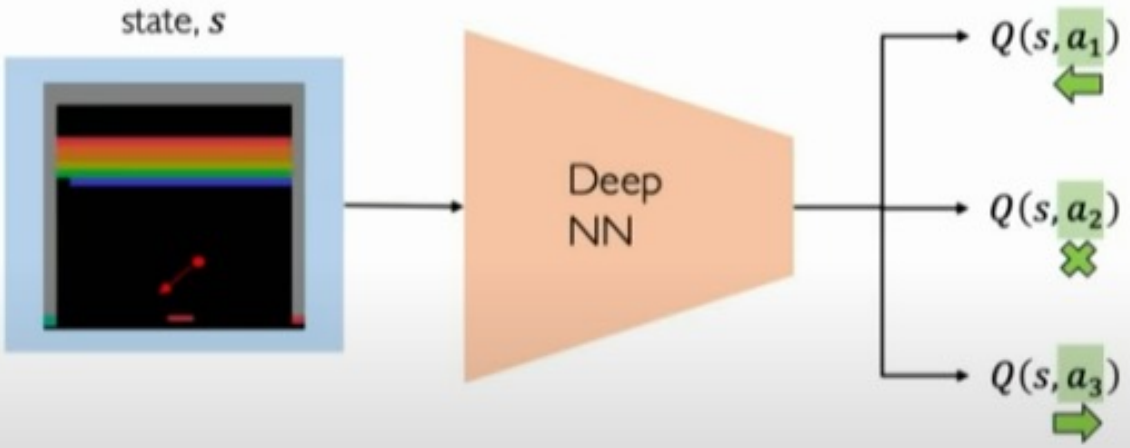
How can we use deep neural networks to model Q-functions?



$$\mathcal{L} = \mathbb{E}\left[\left\|\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}} - \overbrace{Q(s,a)}^{\text{predicted}}\right\|^2\right] \quad \text{Q-Loss}$$

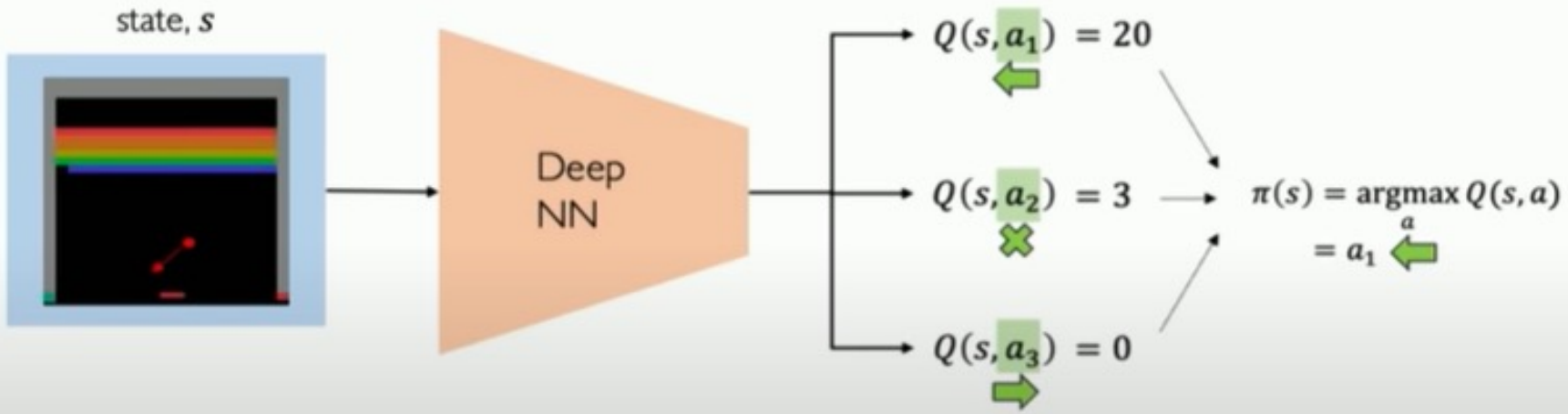
Deep Q Network Summary

Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



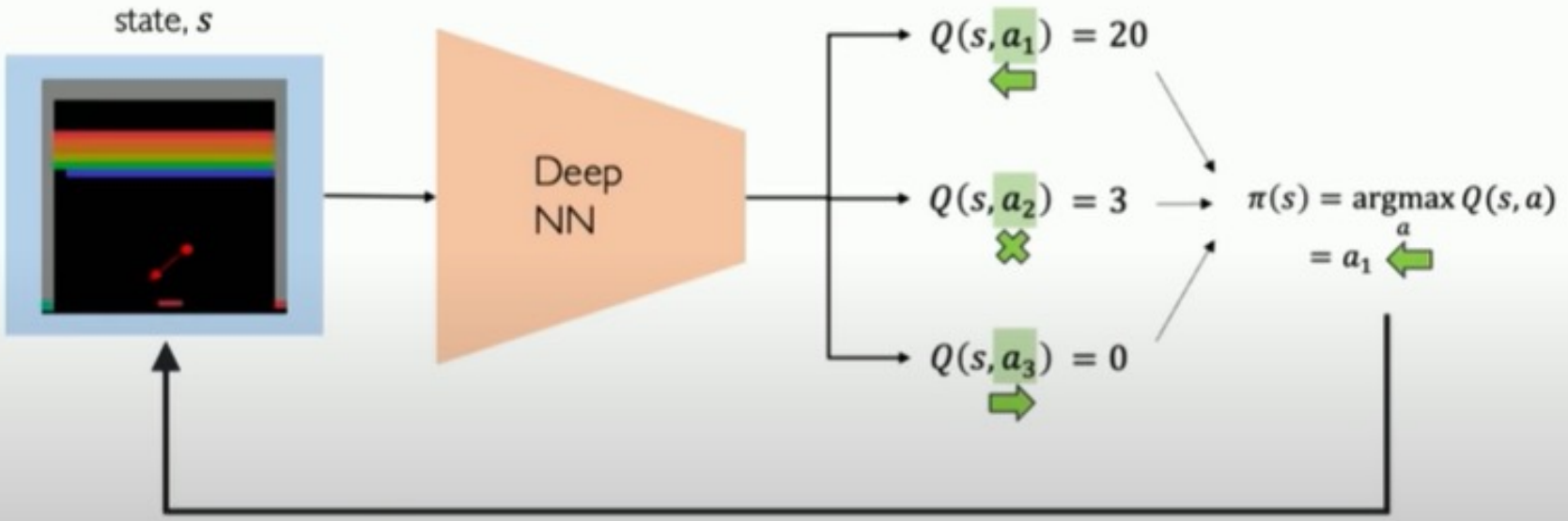
Deep Q Network Summary

Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



Deep Q Network Summary

Use NN to learn Q-function and then use to infer the optimal policy, $\pi(s)$



Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

Flexibility:

- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

Flexibility:

- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

To address these, consider a new class of RL training algorithms: Policy gradient methods

Note - Metrics

- Discounting the rewards makes sense for training or to implement a policy, as it makes it possible to balance the importance of immediate rewards with future rewards.

- However, once an episode is over, we can evaluate how good it was overall by summing the undiscounted rewards.

- For this reason, the AverageReturnMetric computes the sum of undiscounted rewards for each episode, and it keeps track of the streaming mean of these sums over all the episodes it encounters.

Why Loss is a Poor Indicator of the Model's Performance

- The loss might go down, yet the agent might perform worse (e.g., this can happen when the agent gets stuck in one small region of the environment, and the DQN starts overfitting this region).

- Conversely, the loss could go up, yet the agent might perform better (e.g., if the DQN was underestimating the Q-Values, and it starts correctly increasing its predictions, the agent will likely perform better, getting more rewards, but the loss might increase because the DQN also sets the targets, which will be larger too).

Catastrophic Forgetting

- As the agent explores the environment, it updates its policy, but what it learns in one part of the environment may break what it learned earlier in other parts of the environment. The experiences are quite correlated, and the learning environment keeps changing.
  - If you increase the size of the replay buffer (storing previous experiences), the algorithm will be less subject to this problem.
  - Reducing the learning rate may also help.
- Reinforcement Learning is hard: training is often unstable, and you may need to try many hyperparameter values and random seeds before you find a combination that works well.

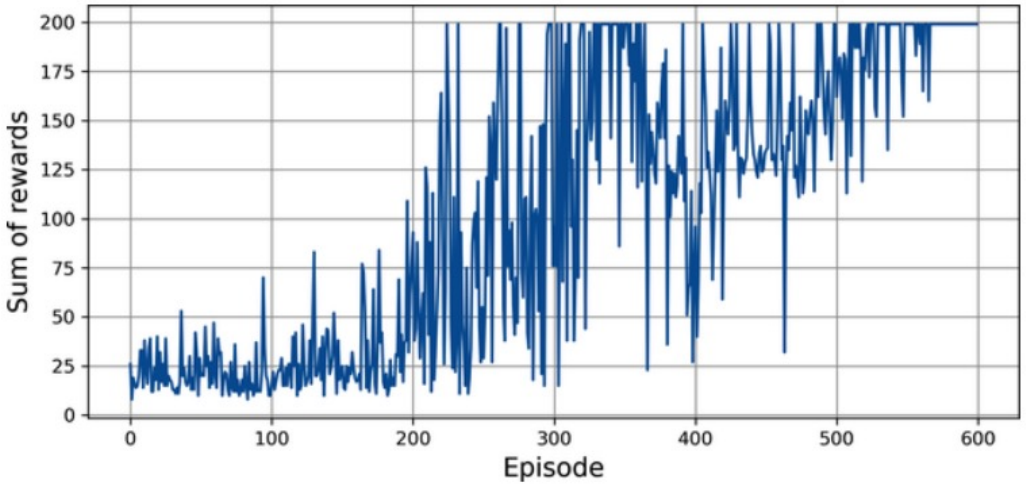


Figure 18-10. Learning curve of the deep Q-learning algorithm

Deep Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy Learning

Find $\pi(s)$

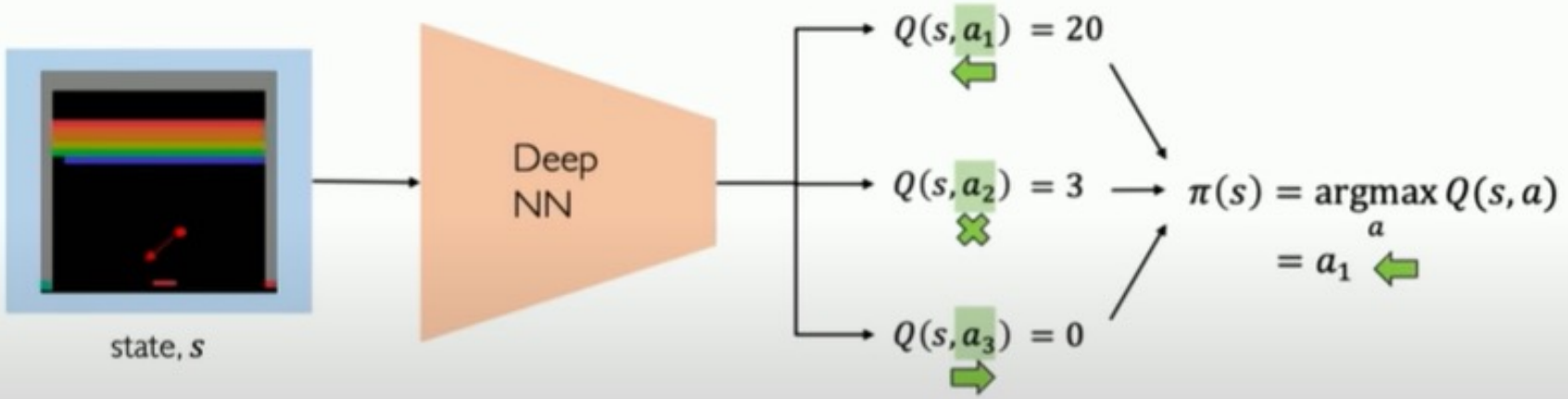
Sample $a \sim \pi(s)$

MIT 6.S191 Introduction to Deep Reinforcement Learning

http://introtodeeplearning.com/

Deep Q Networks (DQN)

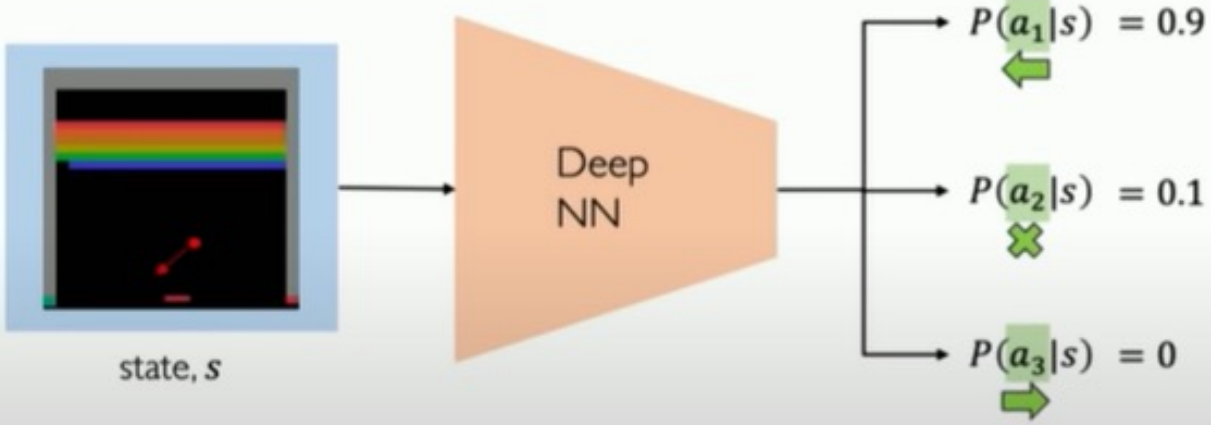
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

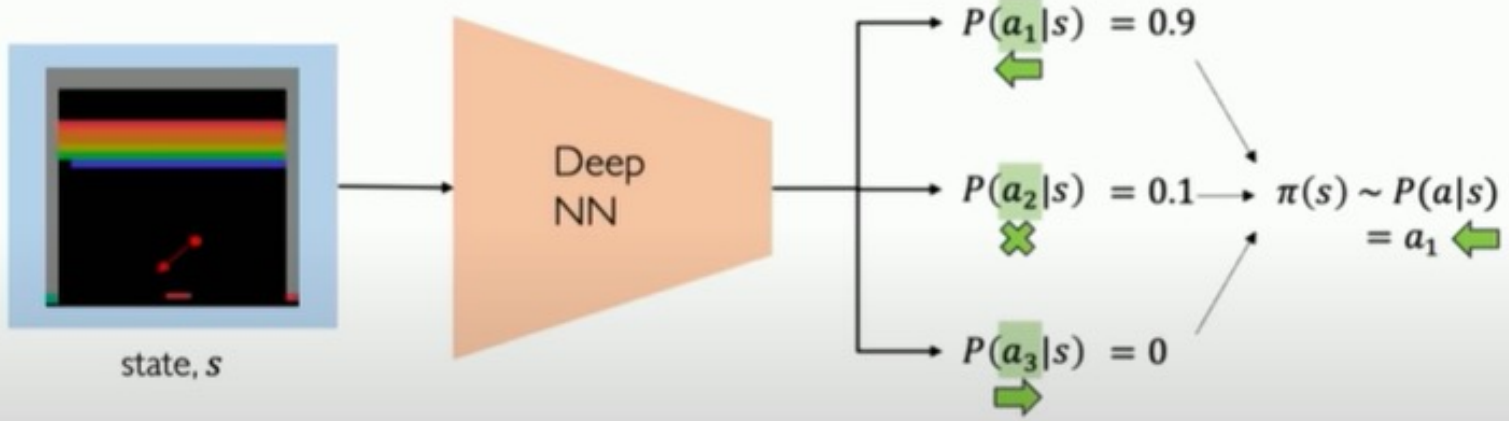
Policy Gradient: Directly optimize the policy $\pi(s)$



Policy Gradient (PG): Key Idea

DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

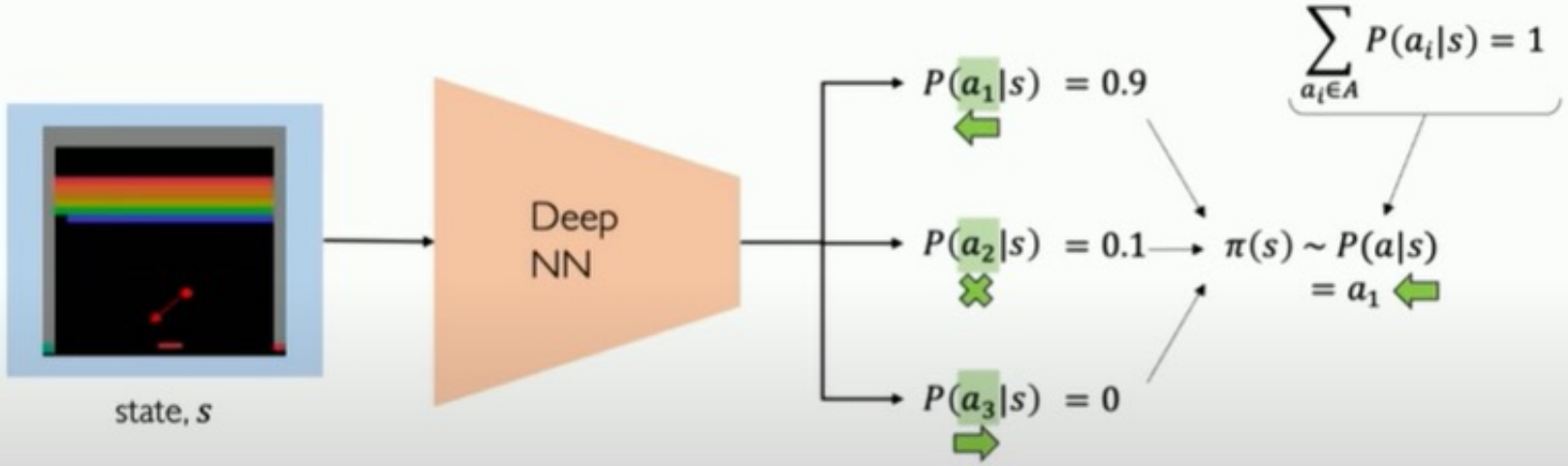
Policy Gradient: Directly optimize the policy $\pi(s)$



Policy Gradient (PG): Key Idea

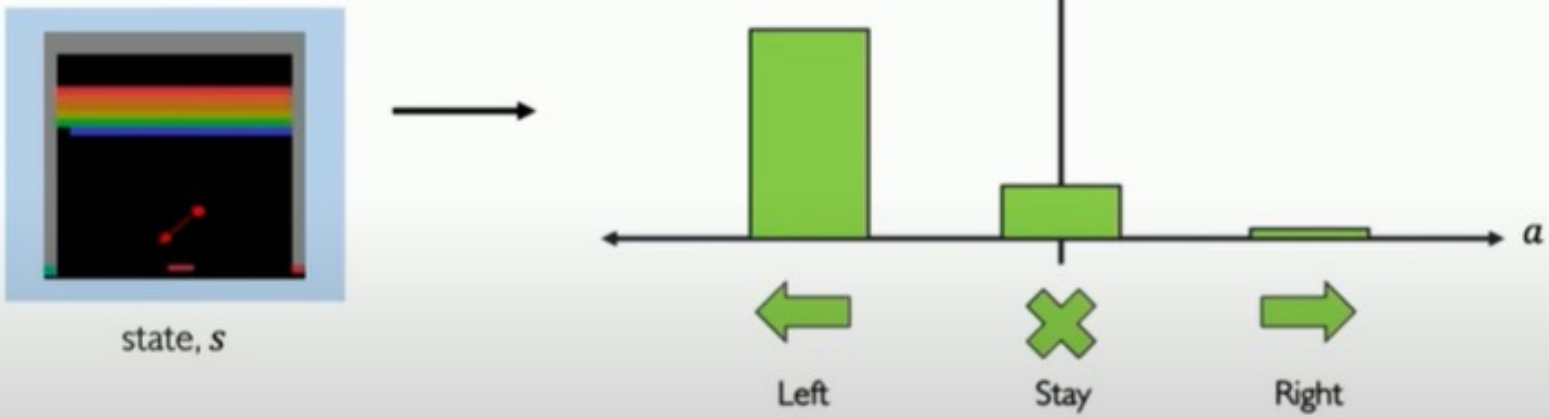
DQN: Approximate Q-function and use to infer the optimal policy, $\pi(s)$

Policy Gradient: Directly optimize the policy $\pi(s)$



Discrete vs Continuous Action Spaces

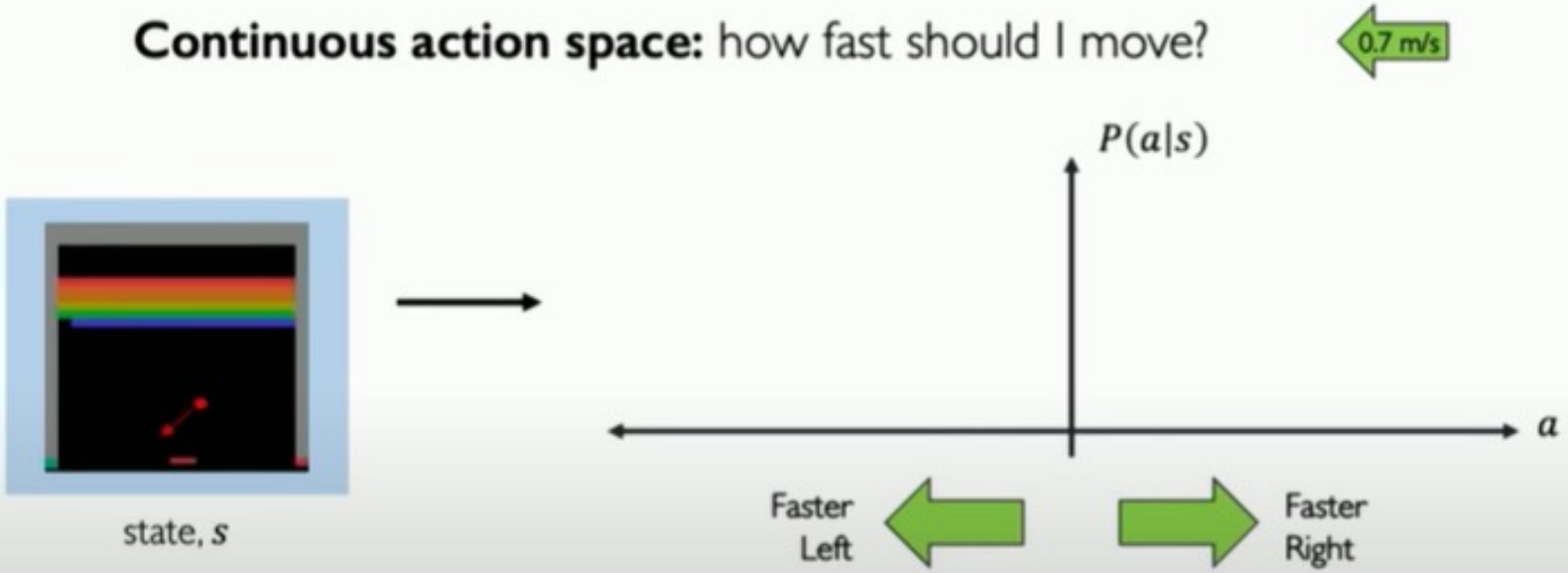
Discrete action space: which direction should I move?



Discrete vs Continuous Action Spaces

Discrete action space: which direction should I move?

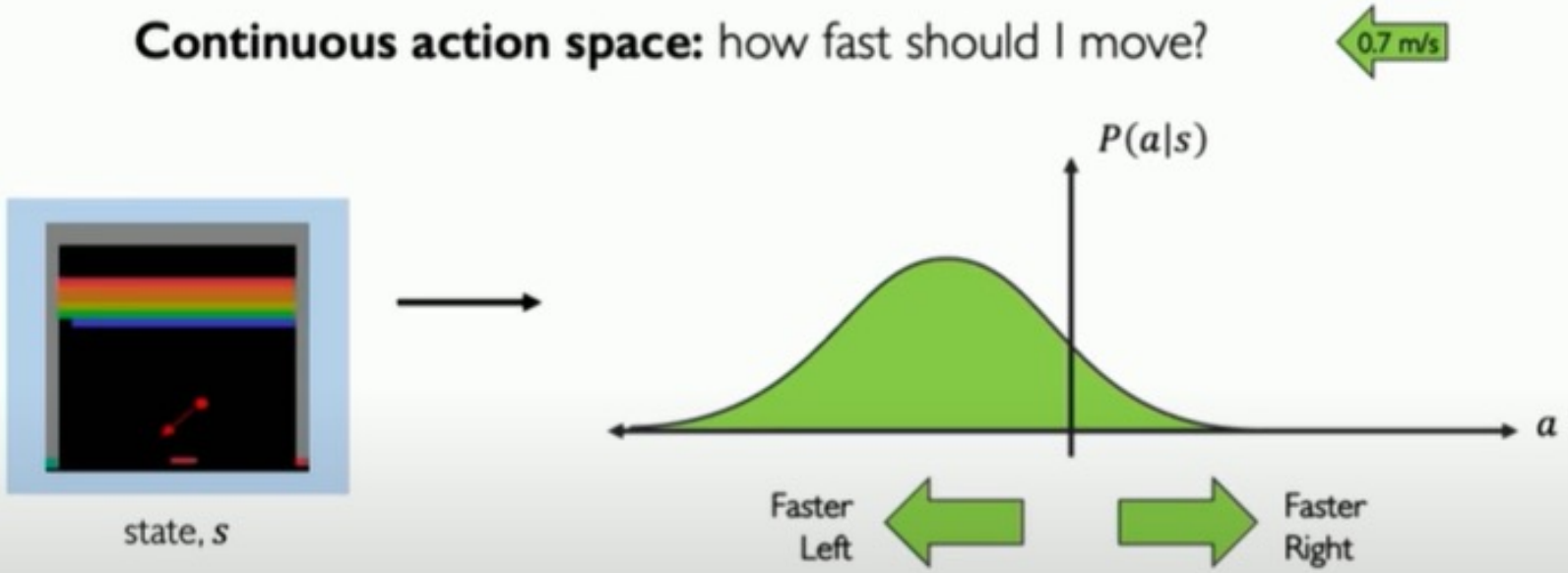
Continuous action space: how fast should I move?



Discrete vs Continuous Action Spaces

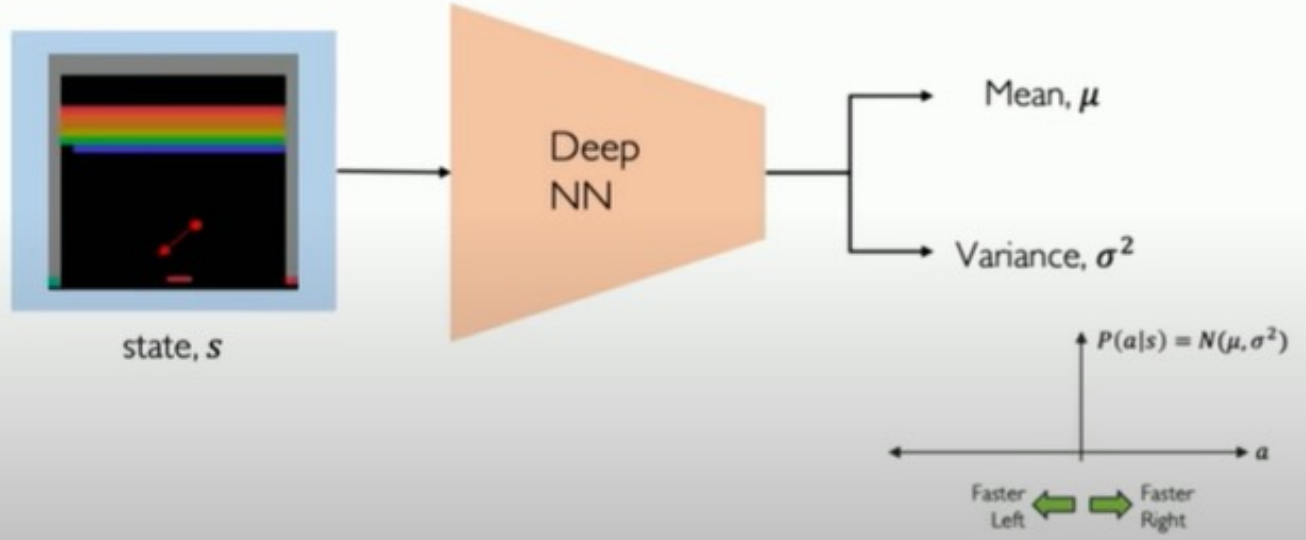
Discrete action space: which direction should I move?

Continuous action space: how fast should I move?



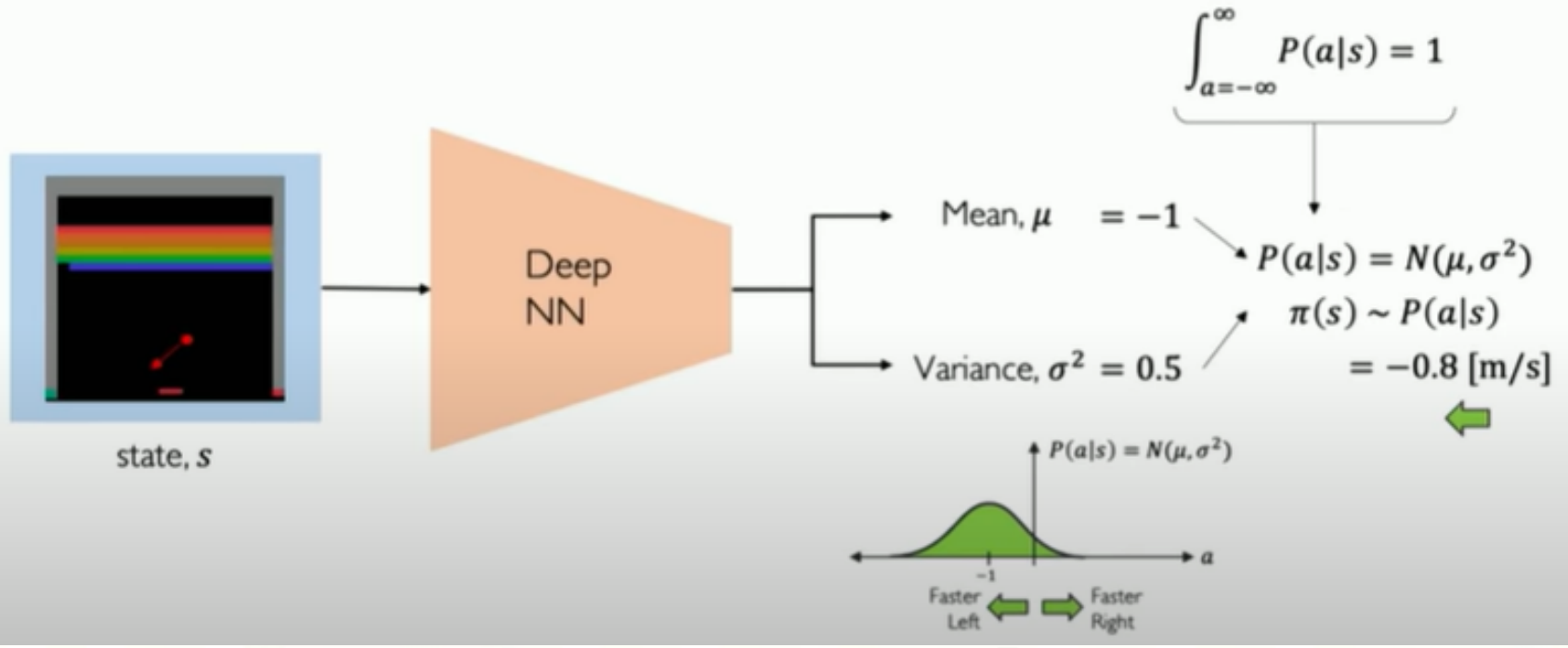
Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



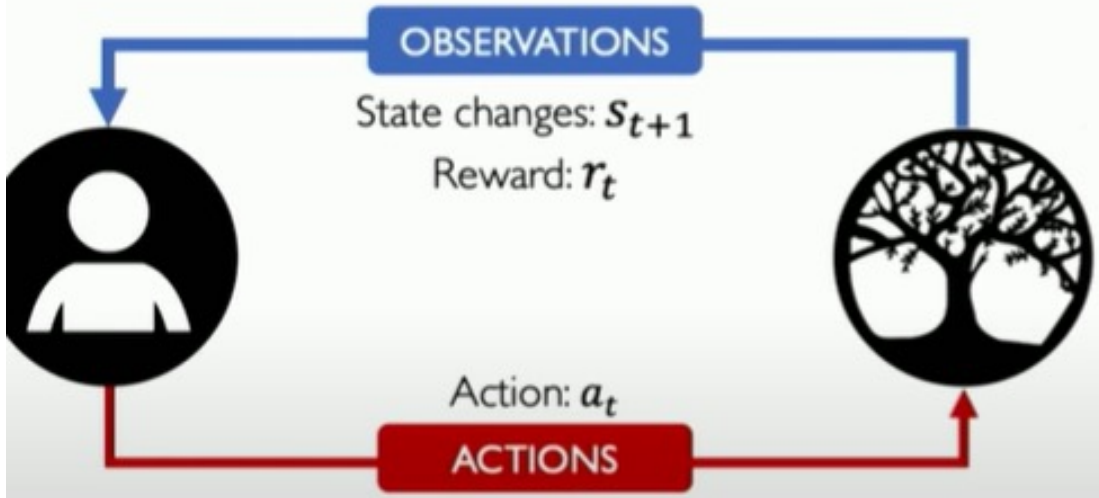
Policy Gradient (PG): Key Idea

Policy Gradient: Enables modeling of continuous action space



Training Policy Gradients: Case Study

Reinforcement Learning Loop:



Training Policy Gradients: Case Study

Reinforcement Learning Loop:

Case Study – Self-Driving Cars

Agent: vehicle

State: camera, lidar, etc

Action: steering wheel angle

Reward: distance traveled

Training Policy Gradients

Training Algorithm

Training Policy Gradients

Training Algorithm

1. Initialize the agent

Training Policy Gradients

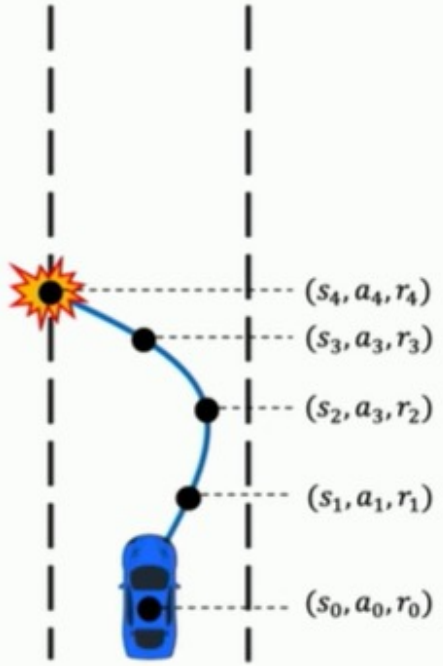
Training Algorithm

1. Initialize the agent
2. Run a policy until termination

Training Policy Gradients

Training Algorithm

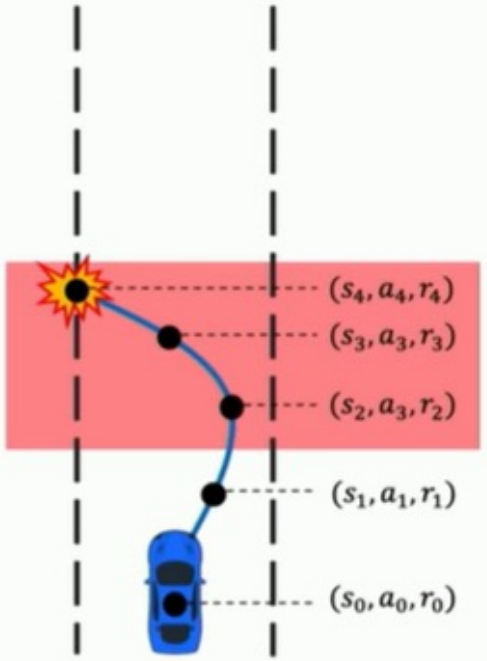
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards



Training Policy Gradients

Training Algorithm

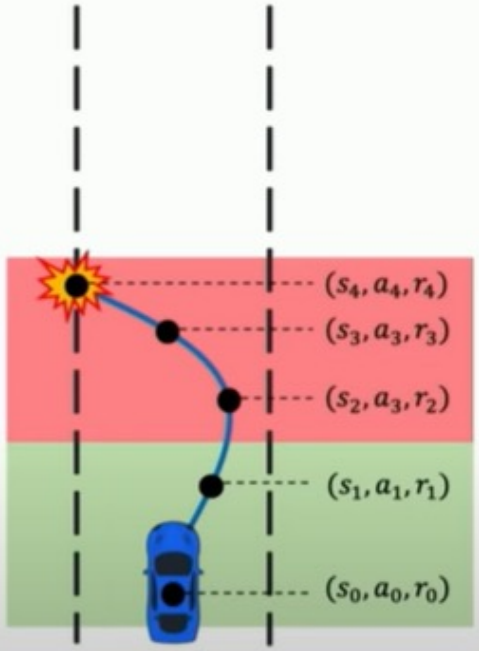
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

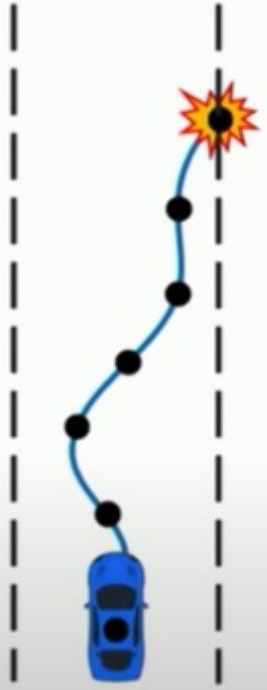
Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Training Policy Gradients

Training Algorithm

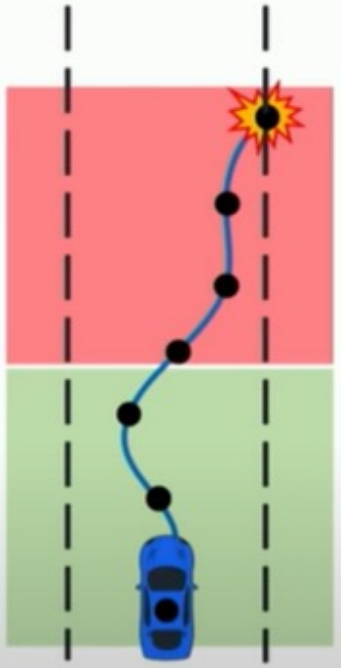
1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Training Policy Gradients

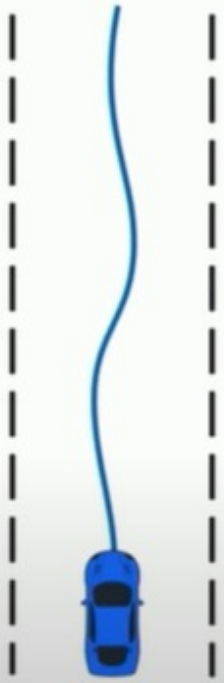
Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward



Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

Gradient descent update:

$$w' = w - \nabla \textbf{loss}$$
$$w' = w + \nabla \log \mathrm{P}(a_t|s_t)\, R_t$$

Training Policy Gradients

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

log-likelihood of action

$$\textbf{loss} = -\log \mathrm{P}(a_t|s_t)\, R_t$$

reward

Gradient descent update:

$$w' = w - \nabla \textbf{loss}$$

$$w' = w + \nabla \log \mathrm{P}(a_t|s_t)\, R_t$$

Policy gradient!

Policy Gradients

A human takes actions based on observations. You have to rely on the fact that you put the work in to create the muscle memory and then trust that it will kick in.

The reason you practice and work on it so much is so that during the game your instincts take over to a point where it feels weird if you don't do it the right way.

https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146

Cart Pole Problem

- The simple policy gradients algorithm we just trained solved the CartPole task, but it would not scale well to larger and more complex tasks.
- Indeed, it is highly *sample inefficient*, meaning it needs to explore the game for a very long time before it can make significant progress.
- This is due to the fact that it must run multiple episodes to estimate the advantage of each action, as we have seen.
- However, it is the foundation of more powerful algorithms, such as *actor-critic* algorithms.

Demo

https://www.youtube.com/watch?v=Eq0X_50fEx8

Tip

- Researchers try to find algorithms that work well even when the agent initially knows nothing about the environment.
- You should not hesitate to inject prior knowledge into the agent, as it will speed up training dramatically.
- For example, since you know that the pole should be as vertical as possible, you could add negative rewards proportional to the pole's angle.
- This will make the rewards much less sparse and speed up training.
- Also, if you already have a reasonably good policy (e.g., hardcoded), you may want to train the neural network to imitate it before using policy gradients to improve it.

Reinforcement Learning in Real Life

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Reinforcement Learning in Real Life

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Reinforcement Learning in Real Life

Training Algorithm

1. Initialize the agent
2. Run a policy until termination
3. Record all states, actions, rewards
4. Decrease probability of actions that resulted in low reward
5. Increase probability of actions that resulted in high reward

Data-driven Simulation for Autonomous Vehicles

VISTA: Photorealistic and high-fidelity simulator for training and testing self-driving cars

Deploying End-to-End RL for Autonomous Vehicles

Policy Gradient RL agent trained entirely within VISTA simulator

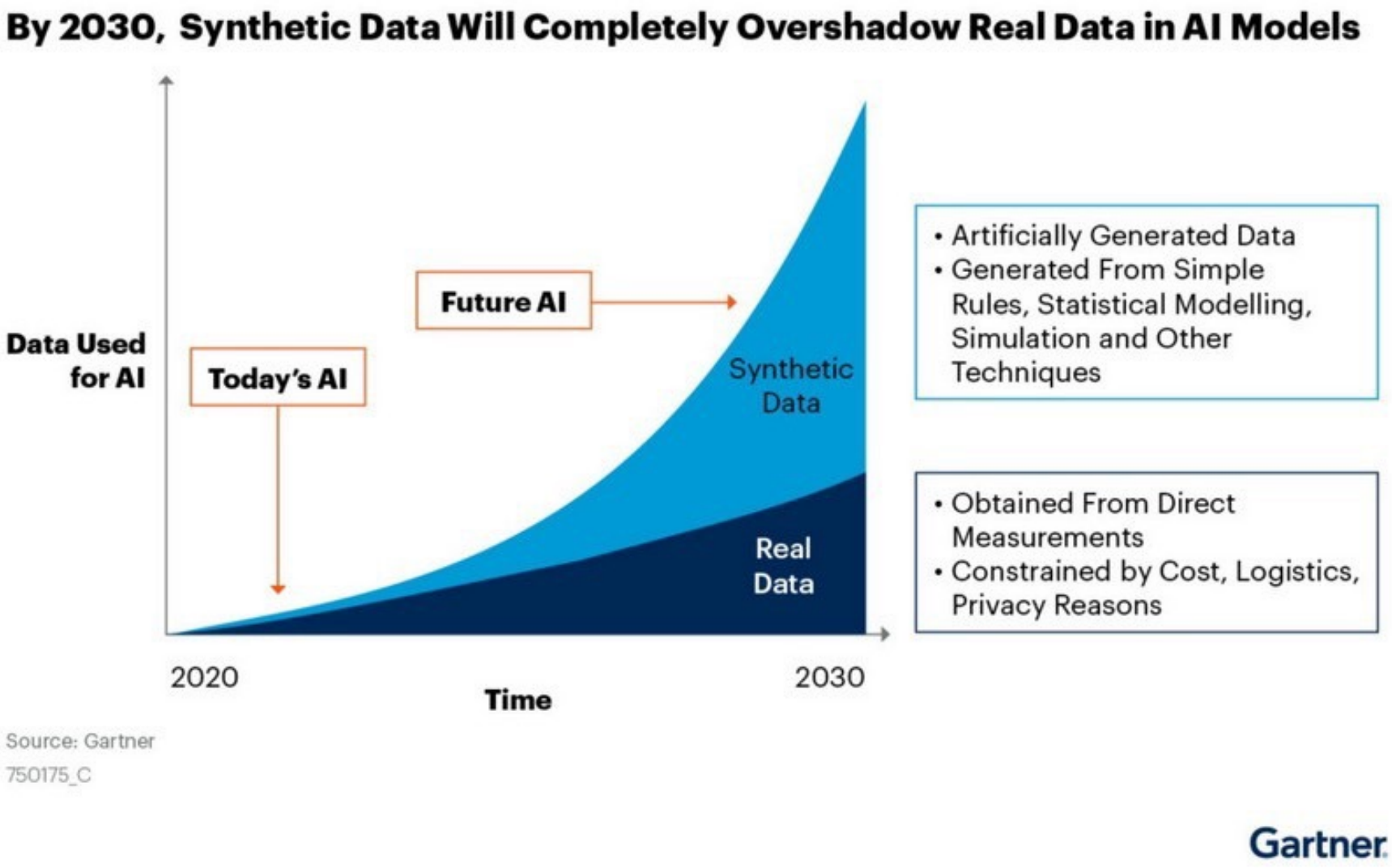
End-to-end agent directly deployed into the real-world

First full-scale autonomous vehicle trained using RL entirely in simulation and deployed in real life!

Synthetic Data

- Synthetic data is annotated information that computer simulations or algorithms generate as an alternative to real-world data.
- Developers of deep neural networks increasingly use synthetic data to train their models. Indeed, a 2019 survey of the field calls use of synthetic data "one of the most promising general techniques on the rise in modern deep learning, especially computer vision" that relies on unstructured data like images and video.
- Developers need large, carefully labeled datasets to train neural networks. More diverse training data generally makes for more accurate AI models (can be time consuming and expensive). A single image that could cost $6 from a labeling service can be artificially generated for six cents, estimates Paul Walborsky, who co-founded one of the first dedicated synthetic data services, AI.Reverie.
- Helpful for reducing bias by ensuring you have the data diversity to represent the real world
- Because synthetic datasets are automatically labeled and can deliberately include rare but crucial edge cases, it's sometimes better than real-world data.

https://blogs.nvidia.com/blog/2021/06/08/what-is-synthetic-data/



Synthetic data will become the main form of data used in AI. Source: Gartner, "Maverick Research: Forget About Your Real Data – Synthetic Data Is the Future of AI," Leinar Ramos, Jitendra Subramanyam, 24 June 2021.

https://www.gartner.com/en/documents/4002912-maverick-research-forget-about-your-real-data-synthetic-

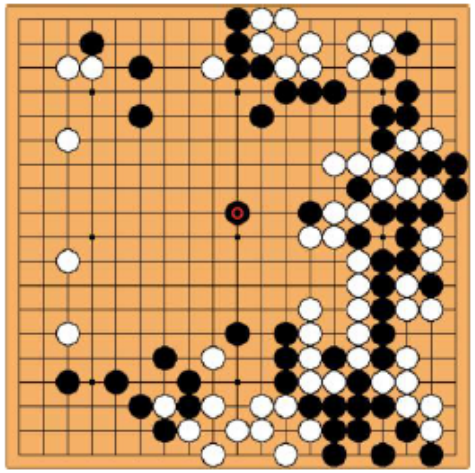
THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Deep Reinforcement Learning Applications

Professor Hamza F. Alsarhan
SEAS 8505

The Game of Go

Aim: Get more board territory than your opponent.



Board Size n x n	Positions 3^{n^2}	% Legal	Legal Positions
1×1	3	33.33%	1
2×2	81	70.37%	57
3×3	19,683	64.40%	12,675
4×4	43,046,721	56.49%	24,318,165
5×5	847,288,609,443	48.90%	414,295,148,741
9×9	4.434264882×10^{38}	23.44%	$1.039191487 91\times10^{38}$
13×13	4.300233593×10^{80}	8.66%	$3.72497923077\times10^{79}$
19×19	$1.740896506\times10^{172}$	1.20%	$2.08168199382\times10^{170}$

Greater number of legal board positions than atoms in the universe.

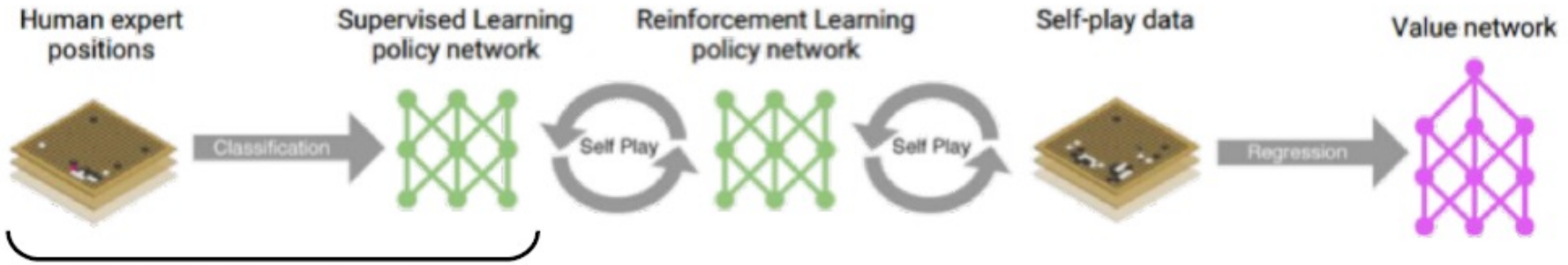
Source: Wikipedia.

AlphaGo Beats Top Human Player at Go

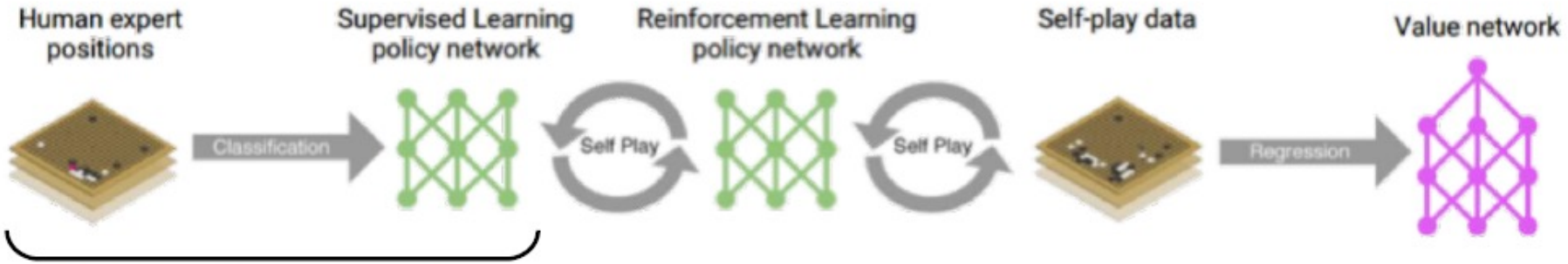
Silver et al., *Nature* 2016.

AlphaGo Beats Top Human Player at Go

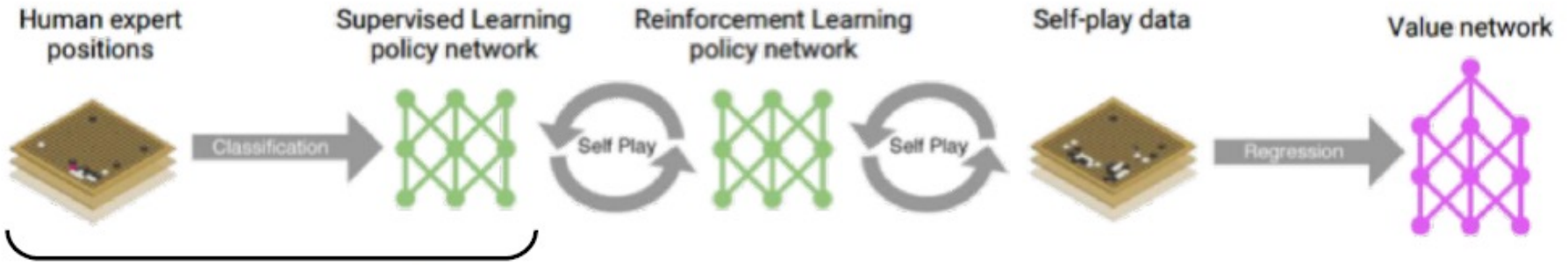
AlphaGo Beats Top Human Player at Go



AlphaGo Beats Top Human Player at Go



AlphaGo Beats Top Human Player at Go

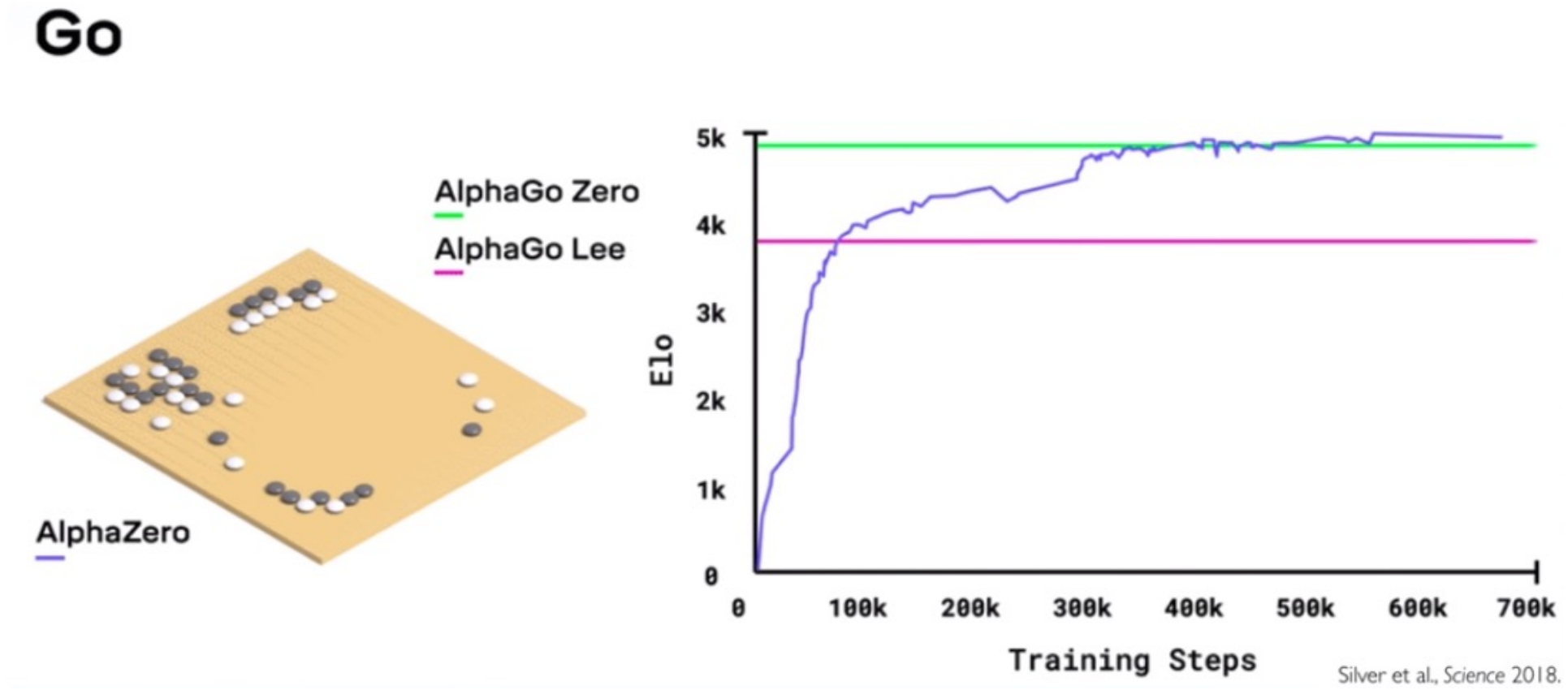


1) Initial training: human data

2) Self-play and reinforcement learning
→ super-human performance

3) "Intuition" about board state

AlphaZero: RL from Self-Play (2018)



Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces

- Recent advances in deep reinforcement learning
  - Have shown success in complex tasks
  - But requires substantial training data
- Leveraging human trainers' input
  - Real-time feedback (good vs bad)
  - Especially useful when expert demonstrations are challenging
- Previous approaches (e.g., TAMER framework)
  - Useful for real-time feedback
  - Limited to low-dimensional state spaces
  - No employment of deep learning
- Proposed solution: Deep TAMER
  - Extension of TAMER framework
  - Utilizes deep neural networks
  - Learns complex tasks efficiently with human trainer input

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Research Question

What is the impact of using deep neural networks on the effectiveness of learning from real-time, scalar-valued human feedback in high-dimensional state spaces?

Goal:
Agents that quickly learn effective policies using real-time, interactive, human critique

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

The Problem Space

- Rapid training time is crucial in certain scenarios
- **Proposed approach:** Incorporate human trainers into reinforcement learning
- Utilize human trainers' task understanding as valuable prior information
- Previous studies introduced TAMER framework to leverage human feedback
- TAMER framework demonstrated improved agent performance and sample efficiency
- Follow-up research examined factors affecting human feedback quality
- TAMER framework primarily applied in **low-dimensional state spaces until now**

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Focus, Approach and Contributions

Focus:

- Extending the TAMER framework to make it applicable in higher-dimensional state spaces.
- The approach involves integrating recent function approximation methods from deep learning, which have proven effective in handling complex environments in reinforcement learning.

Contributions:

1. Introduced enhancements to the TAMER framework tailored for high-dimensional state spaces, forming what we term Deep TAMER.

2. Conducted a comparative analysis to evaluate the performance gap between TAMER and the proposed technique.

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Training an Agent Manually via Evaluative Reinforcement (TAMER)

- Focus on TAMER technique for training autonomous agents
- TAMER allows agents to learn from non-expert human feedback in real-time
- Human trainers provide scalar feedback on the quality of the agent's behavior
- Agents adjust their behavior based on feedback to improve performance
- TAMER has shown success in limited tasks but hasn't been tested extensively in high-dimensional state spaces

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Methods and Findings

- Evaluation focused on Atari game of BOWLING
- Traditional deep RL struggles with game's complexity
- Recent frameworks yield agents scoring 35 to 70 out of 270
- Deep TAMER shows promise in rapid agent improvement
- Human Trainers achieve better scores more quickly in BOWLING through Deep TAMER

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Learning from Human Preferences

Comparison with Christiano et al. (2017): Christiano, Paul F., Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep reinforcement learning from human preferences." Advances in neural information processing systems 30 (2017).

- **Similarities:**
  - Both utilize deep learning in learning from human interaction.
- **Differences:**
  - Christiano et al.'s method requires access to and utilization of a simulator during human interaction.
  - Deep TAMER does not rely on a simulator during interaction.
  - Christiano et al.'s technique demands approximately 10 million learning time steps in the simulator during interaction.
  - Deep TAMER requires only a few thousand time steps of interaction without a simulator.
  - Christiano et al.'s approach necessitates access to powerful hardware.
  - Our method is effective with standard computing hardware.

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Mathematical Formulation

$$H(\cdot,\cdot) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$$

Maintains Estimate of human reward function:

$$\hat{H}(\mathbf{s},\mathbf{a}) \approx H(\mathbf{s},\mathbf{a})$$

… then **acts** by trying to maximize predicted human reward:

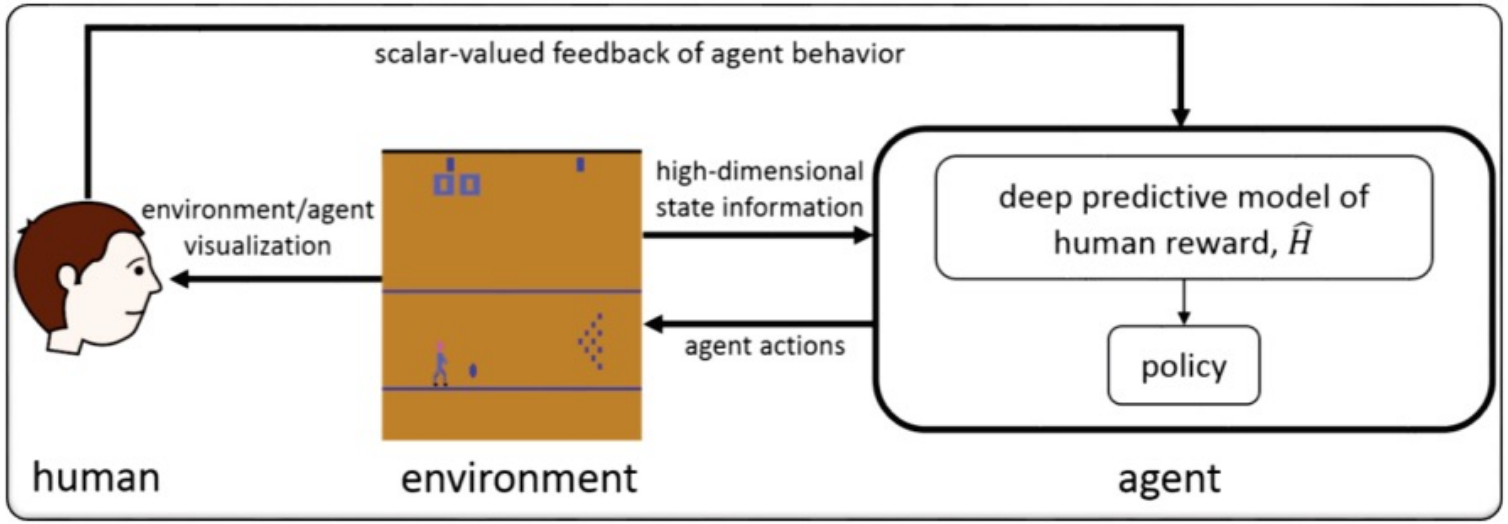
$$\pi(\mathbf{s}) = \max_{\mathbf{a}} \hat{H}(\mathbf{s},\mathbf{a})$$

But NOT the GAME SCORE!

… resulting in an extreme reduction in learning time!

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Advancements in RL: Deep TAMER (Human in the Loop)

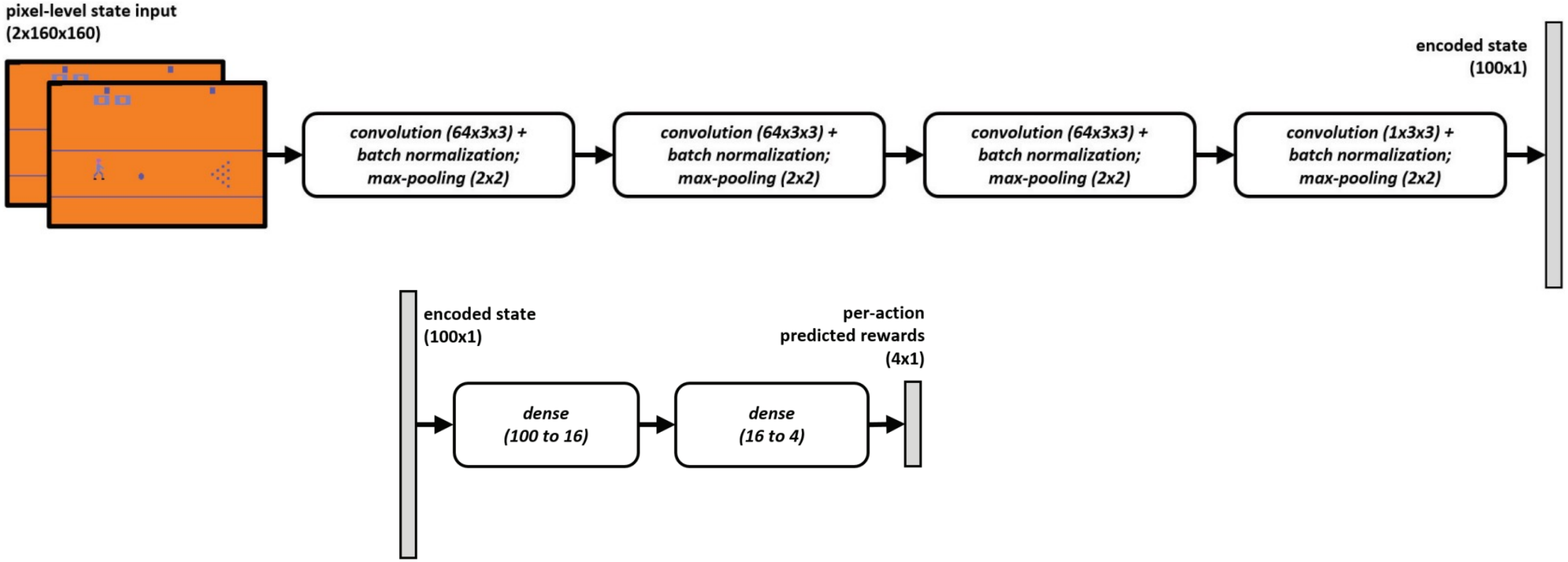


Within this framework, a human observes an autonomous agent attempting to execute a task in a high-dimensional environment and offers scalar-valued feedback to influence the agent's behavior.

Through this interaction, the agent learns the parameters of a deep neural network, denoted as $\hat{H}$, which predicts the human's feedback. Subsequently, this prediction guides the agent's behavioral policy. The focus of this study is particularly on the Atari game of BOWLING, which operates within a pixel-level state space.

Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Network Structure

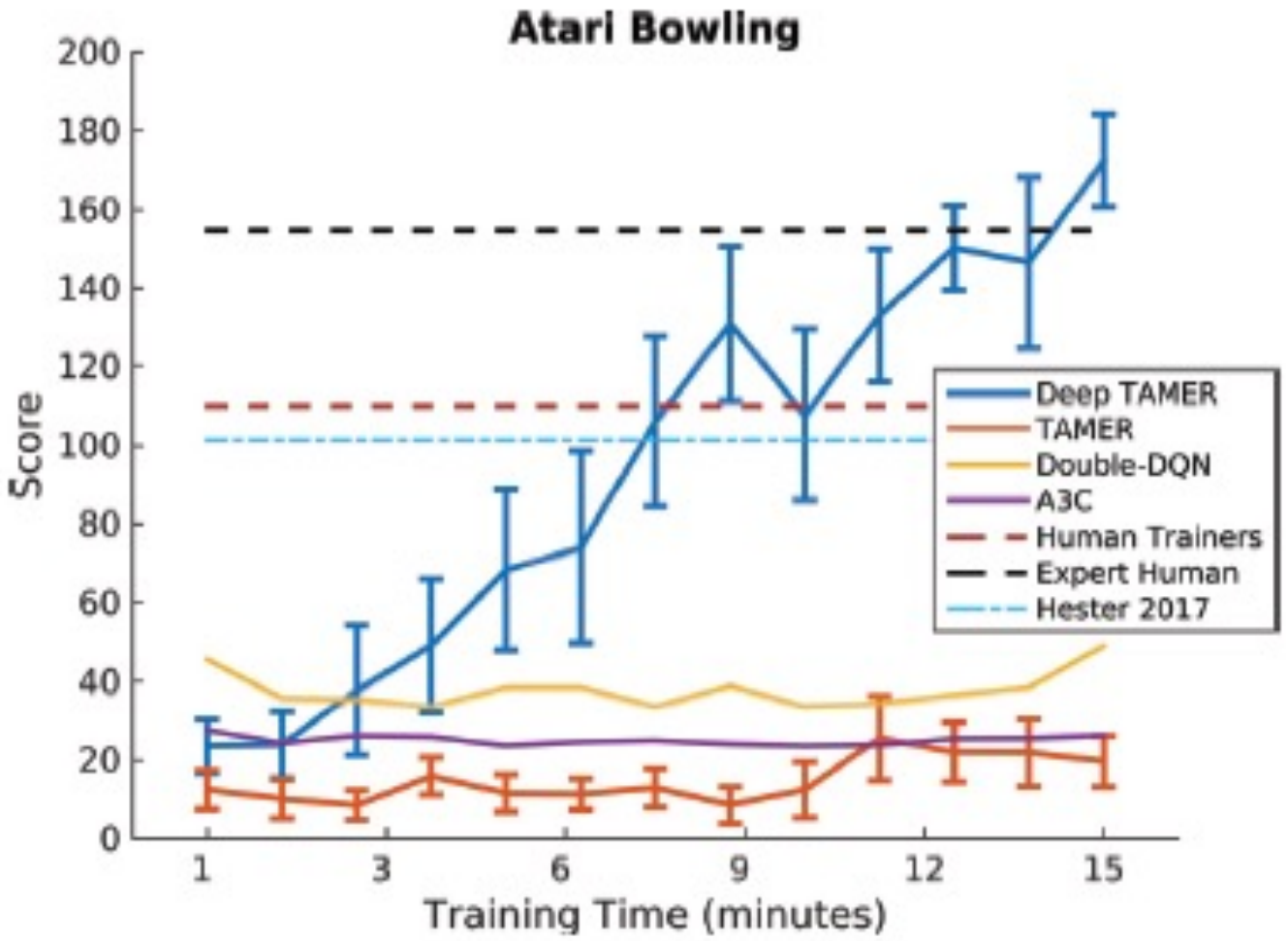


Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

Success of Deep TAMER

- Demonstrated superiority in unsolved Atari game of BOWLING

- Super-human Performance achieved with just 15 minutes of human feedback



Warnell, Garrett, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. "Deep tamer: Interactive agent shaping in high-dimensional state spaces." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1. 2018.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Deep Q-Learning Variants

Professor Hamza F. Alsarhan
SEAS 8505

Fixed Q-Value Targets

- **Target Model Creation**:
  - Clone the online model's architecture: **target = tf.keras.models.clone_model(model)**
  - Copy the weights from the online model: **target.set_weights(model.get_weights())**
- **Updating Q-value Targets**:
  - In the **training_step()** function, use the target model to compute Q-values of next states: **next_Q_values = target.predict(next_states, verbose=0)**
- **Regular Weight Copying**:
  - In the training loop, copy the weights of the online model to the target model at intervals, such as every 50 episodes
- **Stabilizing Training**:
  - By updating the target model less frequently than the online model, the stability of Q-value targets improves, mitigating the detrimental effects of the feedback loop.
- **Additional Stabilization Strategies**:
  - DeepMind researchers used a small learning rate of 0.00025, infrequent updates to the target model (every 10,000 steps), and a large replay buffer of 1 million experiences.
  - They gradually reduced epsilon from 1 to 0.1 over 1 million steps and allowed the algorithm to run for 50 million steps.
  - A deep convolutional net was employed for their DQN.

Double DQN

- In 2015, DeepMind researchers introduced enhancements to their DQN algorithm
- Variant known as double DQN aimed to improve performance and stabilize training
- Addressed issue of overestimation in Q-values observed in the target network
- Target model prone to overestimating Q-values due to approximations
- Proposed solution involved using online model to select best actions for next states
- Target model used solely to estimate Q-values for these optimal actions

Prioritized Experience Replay

- Importance sampling (IS) or prioritized experience replay (PER) - Introduced by DeepMind researchers in a 2015 paper
- Aims to sample important experiences more frequently from the replay buffer
- Importance determined by the magnitude of the TD error, reflecting the disparity between predicted and actual rewards
- Experiences with a large TD error are considered more informative for learning
- Initially, experiences are assigned high priorities to ensure sampling
- Upon sampling, priorities are adjusted based on the computed TD error
- Sampling probability proportional to priority raised to a power determined by hyperparameter ζ
- ζ controls the extent of importance sampling, with higher values indicating more aggressive prioritization
- **Challenge**: Bias introduced towards important experiences can lead to overfitting
  - To mitigate bias, experiences are weighted based on their importance
- Weight determined by a function involving the number of experiences and hyperparameter β
  - β controls the degree of compensation for the importance sampling bias
- Optimal adjustment of hyperparameters crucial for balance and stability in learning process

Dueling DQN

- Introduction:
  - Dueling DQN algorithm (DDQN) introduced in a 2015 paper by DeepMind researchers.
  - Distinct from double DQN, it decomposes Q-value into state value (V(s)) and action advantage (A(s, a)).
- Implementation Steps:
  - Define input layer for state observations.
  - Apply dense layers for feature extraction.
  - Separate into two branches:
    - One for estimating state values.
    - Another for predicting raw action advantages.
  - Adjust raw advantages by subtracting the maximum predicted advantage across actions.
  - Combine state values and adjusted advantages to compute Q-values.
- Benefits of Dueling DQN:
  - Efficient calculation of Q-values by decoupling state values and action advantages.
  - Allows for more stable learning by explicitly modeling state value and action advantage separately.
- Extensions:
  - Integrating additional techniques like prioritized experience replay.
  - Various reinforcement learning techniques can be combined synergistically, as shown by DeepMind's Rainbow agent in a 2017 paper.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Popular RL Algorithms Overview

Professor Hamza F. Alsarhan
SEAS 8505

AlphaGo

- AlphaGo employs Monte Carlo tree search (MCTS) with deep neural networks to defeat human champions in Go.
- MCTS, originating in 1949, explores the search tree from the current position, spending more time on promising branches.
- Initially, AlphaGo used a policy network to select moves, trained with policy gradients, alongside three additional neural networks.
- The AlphaGo Zero paper simplified the approach by utilizing a single neural network for both move selection and game state evaluation.
- AlphaZero extended this algorithm to tackle chess and shogi, not just Go.
- The MuZero paper further improved upon the algorithm, achieving remarkable performance even without prior knowledge of game rules.

Actor-Critic

- Actor-critic algorithms combine policy gradients with deep Q-networks in RL.
  - An actor-critic agent comprises two neural networks: a policy net and a DQN.
  - The DQN is trained conventionally from the agent's experiences.
  - The policy net learns differently and faster than regular PG.
  - Instead of evaluating action values across multiple episodes, it relies on the DQN's estimates.
  - This approach is likened to an athlete learning with the guidance of a coach.

- Asynchronous Advantage Actor-Critic (A3C)
  - Multiple agents learn concurrently in parallel environments
  - Agents explore distinct copies of the environment
  - Periodically update a master network with weight updates asynchronously
  - Collaborative approach allows agents to refine the master network collectively
  - DQN estimates the advantage of each action for enhanced training stability

Proximal Policy Optimization

- Developed by John Schulman and other OpenAI researchers
- Based on A2C (Advantage Actor-Critic) algorithm
- Incorporates a clipped loss function to prevent excessively large weight updates
- Derived from the Trust Region Policy Optimization (TRPO) algorithm
- OpenAI Five, an AI based on PPO, defeated world champions in Dota 2

Curiosity Based Exploration

- Instead of relying on external rewards, they advocate for fostering curiosity in the agent to promote exploration.
- This approach draws parallels to stimulating curiosity in children, often yielding better outcomes than simple reward systems.
- The agent continuously predicts outcomes of actions and seeks situations where outcomes deviate from predictions, aiming for novelty.
- Predictable outcomes lead to loss of interest, prompting the agent to seek new experiences.
- If the agent perceives no control over unpredictable outcomes, it eventually loses interest.

Open Ended Learning

- Objective of OEL: Train agents for continuous learning of new tasks
- Example: POET algorithm (2019)
  - Generates multiple simulated 2D environments
  - Agents trained per environment to walk quickly while avoiding obstacles
  - Curriculum learning: Environments gradually become more challenging
- Competition among agents:
  - Each agent competes against others across all environments
  - Winners replace existing agents, fostering knowledge transfer
- Results:
  - Agents trained with POET outperform those on single tasks
  - Adaptability demonstrated in handling harder environments

Summary – Reinforcement Learning

- We covered many topics in this chapter: Policy Gradients, Markov chains, Markov decision processes, Q-Learning, Approximate Q-Learning, and Deep Q-Learning and its main variants (fixed Q-Value targets, Double DQN, Dueling DQN, and prioritized experience replay).

- We discussed how to use TF-Agents to train agents at scale, and finally we took a quick look at a few other popular algorithms.

- Reinforcement Learning is a huge and exciting field, with new ideas and algorithms popping out every day

Types of Reinforcement Learning: Comparisons

Objective:

1. Value Learning: Estimate the value function (either state-value or action-value) to quantify the expected cumulative rewards.
2. Q-Learning: Estimate the action-value function, which represents the expected cumulative rewards of taking a specific action in a given state and following an optimal policy thereafter.
3. Policy Gradient Learning: Directly optimize the policy function to maximize expected cumulative rewards.

Representation:

1. Value Learning: Represents the value function, which can be either state-value (V-function) or action-value (Q-function).
2. Q-Learning: Represents the action-value function, Q(s, a), which estimates the expected cumulative rewards of taking action 'a' in state 's'.
3. Policy Gradient Learning: Represents the policy function, which maps states to actions.

Update Rule:

1. Value Learning: Update the value function based on the Bellman equation, relating the value of a state or state-action pair to the value of its successor states or state-action pairs.
2. Q-Learning: Update the Q-values based on the Bellman equation, aiming to minimize the difference between the estimated Q-values and the actual rewards observed during learning.
3. Policy Gradient Learning: Update the policy parameters by following the gradient of a performance measure (e.g., expected cumulative rewards) with respect to the policy parameters.

Types of Reinforcement Learning: Comparisons

Exploration vs. Exploitation:

1. Value Learning: Typically employs an exploration strategy such as ε-greedy, where the agent explores randomly with a small probability ε and exploits the learned value function otherwise.
2. Q-Learning: Utilizes an ε-greedy strategy for exploration, similar to value learning methods.
3. Policy Gradient Learning: Uses stochastic policies, allowing for exploration of the action space by sampling actions according to their probabilities.

Sample Efficiency:

1. Value Learning: Can be more sample-efficient compared to policy gradient methods, especially in environments with discrete action spaces.
2. Q-Learning: Can also be sample-efficient, particularly in environments with discrete action spaces, as it updates Q-values based on individual state-action transitions.
3. Policy Gradient Learning: May require more samples to converge, as it updates the policy based on the entire episode trajectory.

In summary, while all three approaches aim to solve reinforcement learning problems, they differ in their objectives, representations, update rules, exploration strategies, and sample efficiency. The choice between them depends on the specific characteristics of the problem domain and the requirements of the task at hand.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Homework Overview

Professor Hamza F. Alsarhan
SEAS 8505

This Week

- Reading:
  - Chapter 18 in the textbook
- HW #8
- Final discussion paper

- Reminder: No extensions provided. Start assignments early!

Next Steps

- Come to office hours with any questions you may have.
- Work on your HW and Discussion and submit them by 9:00 am ET on Saturday.
- See you next class!

Thank you!