# APS360 Artificial Intelligence Fundamentals

Lisa Zhang

Lecture 14; July 15, 2019

# Agenda

- Reinforcement Learning
- Unfortunately no code today

Goal:

- Understand standard terminology used in RL
- Be able to set up an RL problem
  - What is the state?
  - What are the possible actions?
  - What are the rewards?
  - What discount factor should we use?
  - What should the model architecture look like? (What are the input/output to our model?)
- We won't actually train an RL model

# Reinforcement Learning

# Example

- Playing Go: https://www.youtube.com/watch?v=HT-UZkiOLv8
- Mario: https://www.youtube.com/watch?v=L4KBBAwF_bE
- Breakout: https://www.youtube.com/watch?v=V1eYniJ0Rnk
- Biped: https://www.youtube.com/watch?v=TEFXp2Ro-10

# Game Playing as Classification?

Let's say we want to build a neural network agent to play Breakout.

- **Input**: pixel intensities of the screen over the last few frames
- **Output**: how to move the paddle (left, right, or no movement)

We can build a convolutional network to predict the correct movement.

Q: Why can't we just train the conv net like before? (e.g. with CrossEntropy loss)

# Game Playing: Difficulty

**Problem:** We don't have the "correct" answer (ground truth move) at each step

- We have a signal to tell us whether our agent completed the game.
- . . . but the signal is delayed . . .
- we don't see the signal until the end of the game.

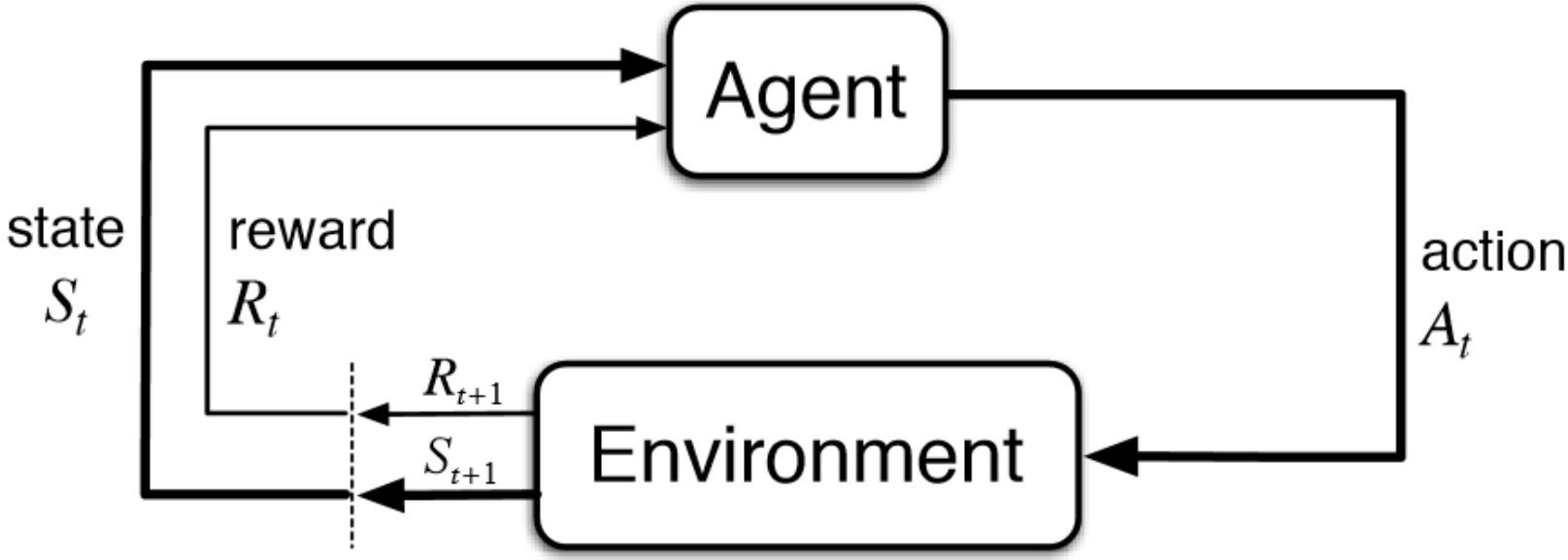
Q: Why can't we just use win/loss information as a label for whether the current move is good/bad?

# Reinforcement Learning

Reinforcement learning is different from supervised learning:

- There is no supervision (no "correct" answer), only a scalar **reward** signal
- There is a notion of "time" (steps/moves)
- Feedback is delayed, not instantaneous
- Agent's action affects the subsequent data it receives

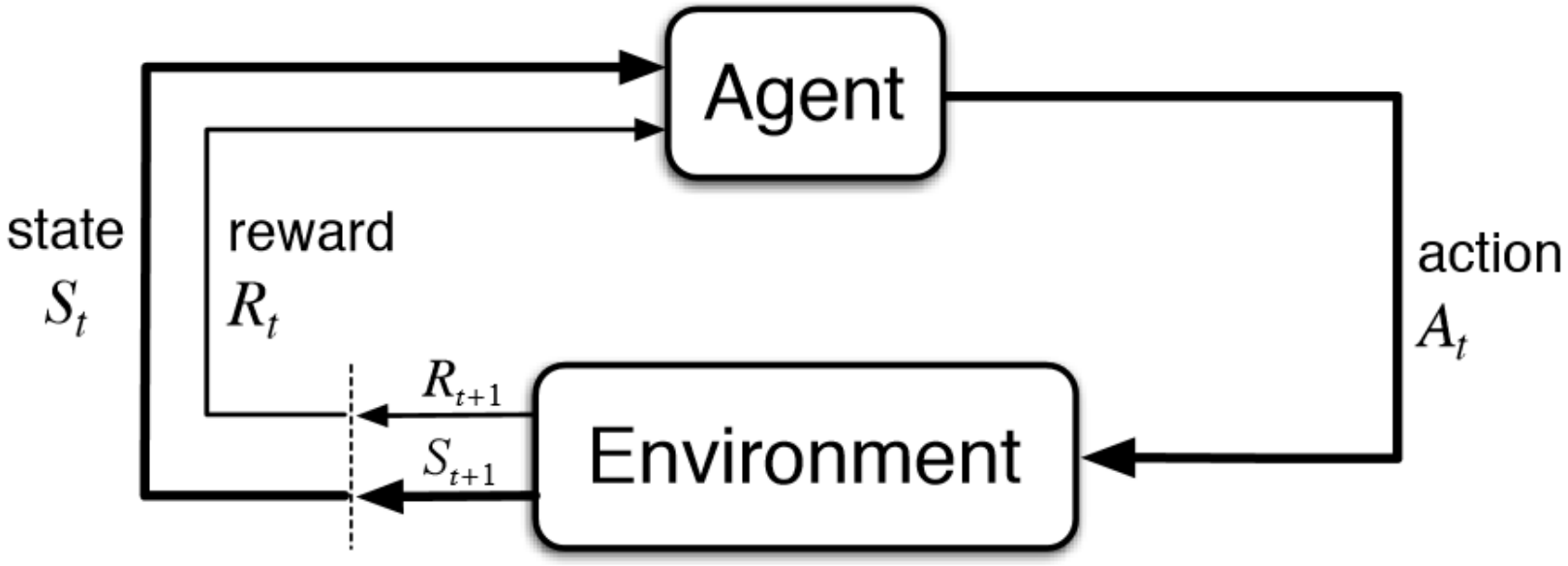
# Reinforcement Learning Setup




- **Environment**
  - Provides the agent with the current **state**
  - Provides a **reward** at each time step
- **Agent**
  - Chooses an **action** at each time step, given the **state**
  - Can modelled using a neural network!

Representing the **state** and the **action** can be tricky – **Worksheet**

# Reward




- A **reward** $R_t$ is a scalar feedback signal
- The goal of the agent is to selection actions to maximize total future reward.

# Bad Rewards


Setting up good rewards to get good agent behaviour is tricky.

# Good Rewards

- **Sparse reward**: there is a nonzero reward in a few time steps
  - e.g. win/loss at the end
- **Dense reward**: there is a nonzero reward in most time steps
  - when possible, it is easier (faster) to train with dense rewards

**Worksheet**

# Reward Hypothesis

*All reinforcement learning task can be trained by maximizing some (correctly defined) accumulative reward.*

Q: Do you believe in the reward hypothesis? Why or why not?

# Episode

An **episode** is a sequence:

$S_0, A_0, R_0, S_1, A_1, R_1, \ldots \; S_T, A_T, R_T$

that concludes with a terminal state.

In our game-playing examples, an **episode** is one "match" or one "game".

# Return

The **discounted return** at a time step $R_t$ is defined to be:

$G_t = \sum_{s=t}^{T} \gamma^{s-t} R_s$

Where $0 < \gamma \leq 1$ is a constant **discount factor**.

With discounting, getting a reward now is better than getting the same reward later on.

# Major Possible Components of an RL Agent

- **Policy**: an agent's behaviour function
- **Value function**: expected future reward at a state
- **Model**: the agent's representation of the environment

Let's look at each of these components in turn.

# Policy

- A **policy** is the agent's behaviour: the action it chooses at a state. It is usually represented with the letter $\pi$
- The policy function is a function that maps state to action
  - **deterministic**: $a = \pi(s)$
  - **stochastic**: $\pi(a|s) = P[A_t = a|S_t = s]$

We can parameterize $\pi$ using a neural network!

# Value Function

- A **value function** is a prediction of future reward, assuming we follow a particular policy
- Used to evaluate the goodness or badness of states
- The value function is a function that maps state to expected discounted return
- The **Q-function** maps (state, action) pairs to expected discount return

We can parameterize the Q-function using a neural network!

# Model

- A **model** predicts what the environment will do next
  - predict the next state (given the current state and an action to take)
  - predict the next reward

# Types of RL Agents

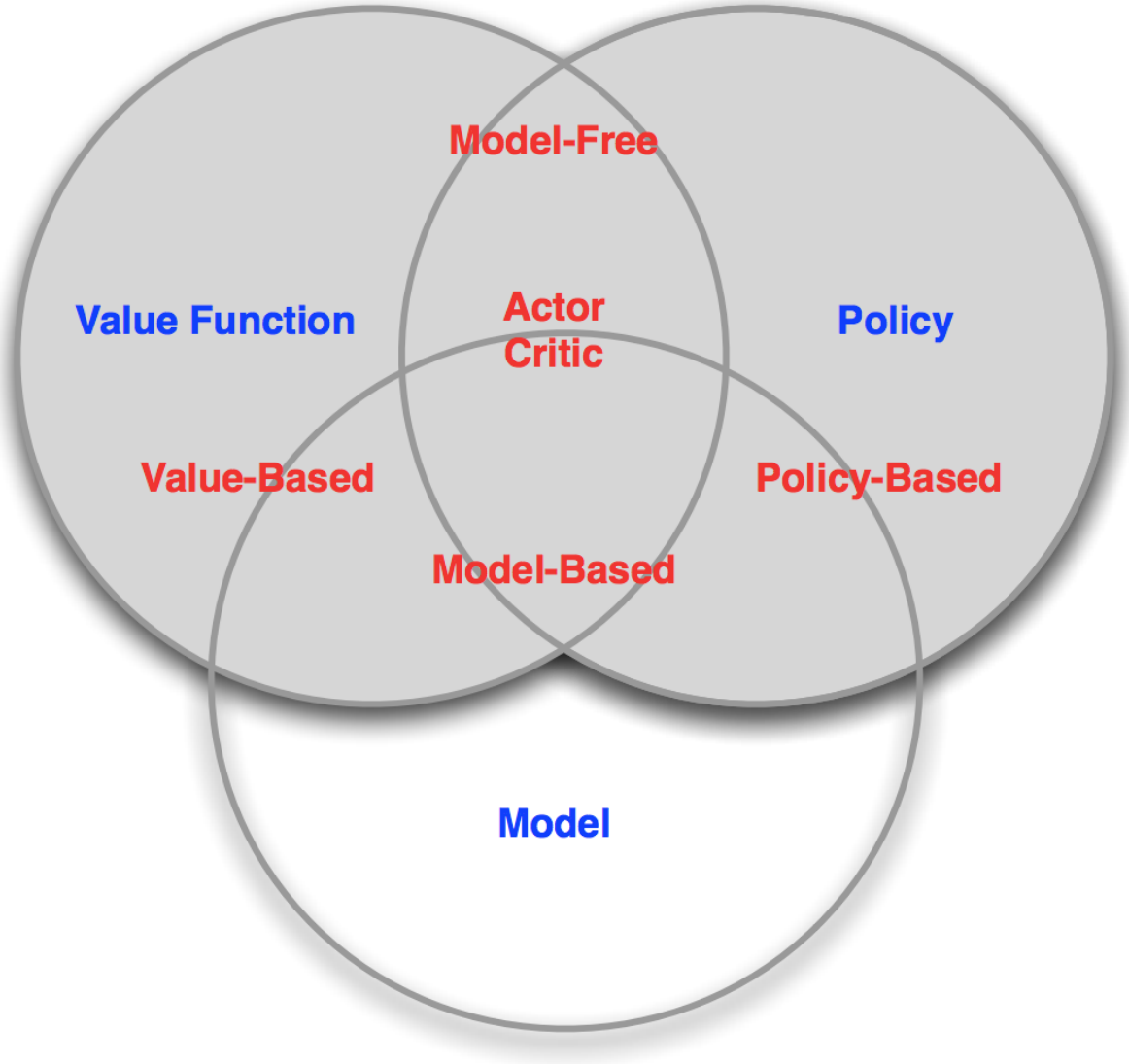
An RL agent can contain machine learning models that learns one or more of the policy, value function, or model.

- **Policy Based** agent:
  - learn policy function only (no estimate of value function, no prediction of the environment)
- **Value Based** agent:
  - learn value function only
  - implicit policy: choose the action that maximize the value function of the next state
- **Actor Critic**:
  - has an explicit policy (actor)
  - also learns a value function (critic)

# Model Free vs Model Based RL

- **Model Free** agent:
  - Only policy and/or value functions
  - **No model**
- **Model Based** agent:
  - Policy and/or value functions
  - Model (to predict what the environment will do next)

# RL Agent Taxonomy



# Example of Policy Based Agent: Pong

https://www.youtube.com/watch?v=YOW8m2YGtRg

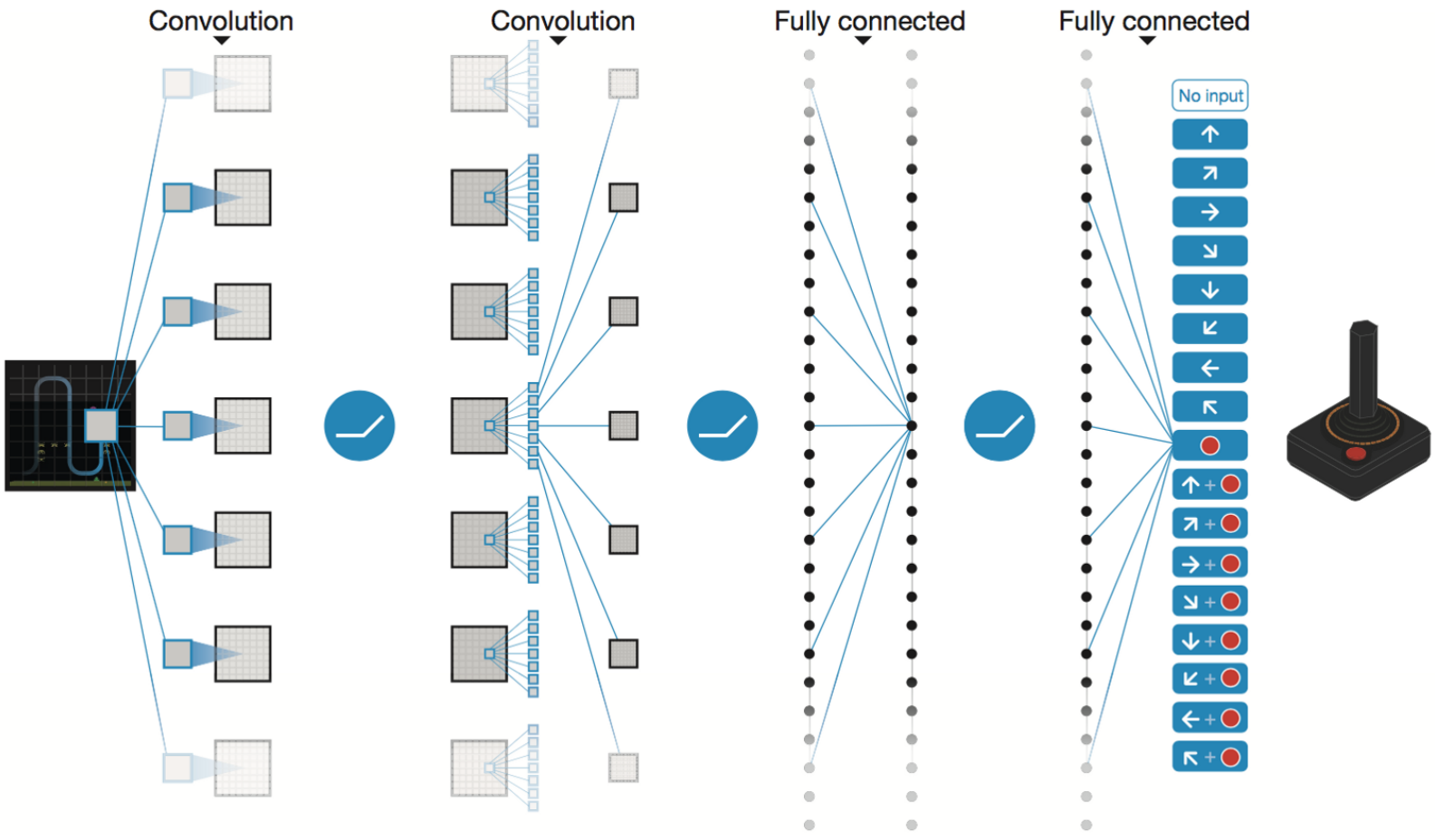
- **Policy Based** agent
  - Input to the policy function: pixel intensities (over last few time steps?)
  - Output of the policy function: velocity of paddle

# Example of Value Based Agent: Breakout

https://www.youtube.com/watch?v=V1eYniJ0Rnk

- **Value Based** agent
  - Input to the model:
    - pixel intensities (over last 4 time steps)
  - Output of the model:
    - expected future reward for each possible *action*

# Model Architecture




From https://towardsdatascience.com/atari-reinforcement-learning-in-depth-part-1-ddqn-ceaa762a546f

# Reinforcement Learning

- The environment is initially unknown
- The agent interacts with the environment, and receives rewards
- Based on the rewards / discounted returns, update the agent weights

The precise way in which the agent is trained is beyond the scope of this course.

# Training the Policy Function: Overview

Update the parameters of the neural network to maximize the **return**.

Rough idea:

- play several games (several episodes) using the current model weights
- for each episode $i$ and time step $k$, compute the return at that step $G_k^{(i)}$
- modify the weights of the neural networks so that **actions** $A_k^{(i)}$ that produce **large returns** $G_k^{(i)}$ becomes **more likely**

# Training the Policy Function: Idea

Q. Why can't we just update the parameters of the neural network to maximize the immediate reward (rather than the discounted return)?

(This way, we won't have to wait until an episode is complete.)

# Using the Policy/Value Function

Q. How should we decide what move to take based on the output of the policy/value function?

As an example, let's say that we are playing breakout. The policy's prediction distribution over the next move looks like this:

- $0.8$ = move paddle left
- $0.2$ = move paddle right

Alternatively, the estimated discounted return for the two actions looks like this:

- $78$ = move paddle left
- $17$ = move paddle right

Which move should we actually choose?

# Exploration vs Exploitation

- **Exploitation**: Make moves the function already thinks will lead to a good outcome vs
- **Exploration**: Try making novel moves and see if you discover a way to adjust the function to get even better outcomes

Need a good balance of **exploration** vs **exploitation** to learn a good RL agent

# Epsilon Greedy Policy

- Use a "greedy" approach with probability $1 - \epsilon$
  - Choose the action with the highest probability or estimated return
- Choose a random action with probability $\epsilon$

You could change $\epsilon$ over time – choosing a smaller $\epsilon$ as training progresses.

# Where to go from here?

- Textbook: Reinforcement Learning by Sutton & Barto
- David Silver's video lectures:
  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Half of this lecture is based on David Silver's first lecture