

---

## Reinforcement Learning



# Reinforcement Learning

- Reinforcement Learning has been around since 1950s and
  - Produced many interesting applications over the years
  - Like TD-Gammon, a Backgammon playing program



---

# Reinforcement Learning

---

- In 2013, researchers from an English startup **DeepMind**
  - Demonstrated a system that could learn to play
  - Just about any Atari game from scratch and
  - Which outperformed human
  - It used only raw pixels as inputs and
  - Without any prior knowledge of the rules of the game
- In March 2016, their system AlphaGo defeated
  - Lee Sedol, the world champion of the game of Go
  - This was possible because of Reinforcement Learning



---

# Reinforcement Learning

---

- In this chapter, we will learn about
  - What Reinforcement Learning is and
  - What it is good at

---

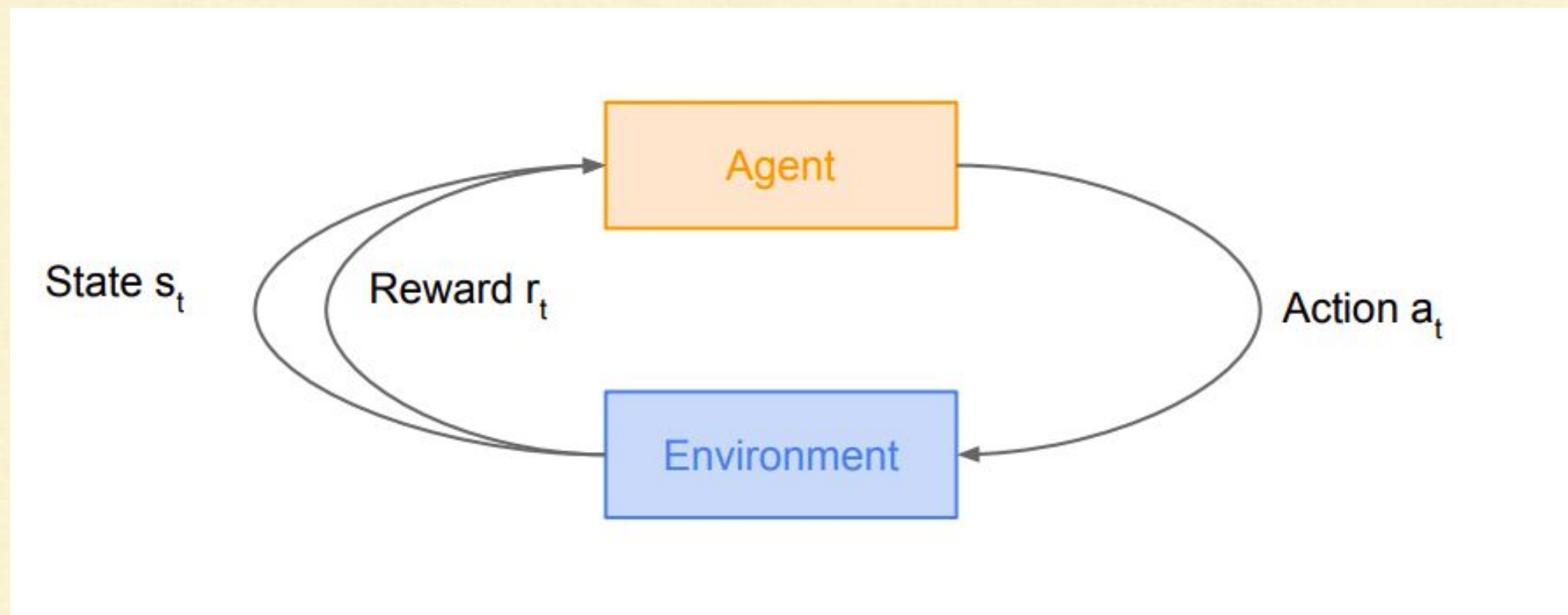
# Reinforcement Learning

---

Learning to Optimize Rewards

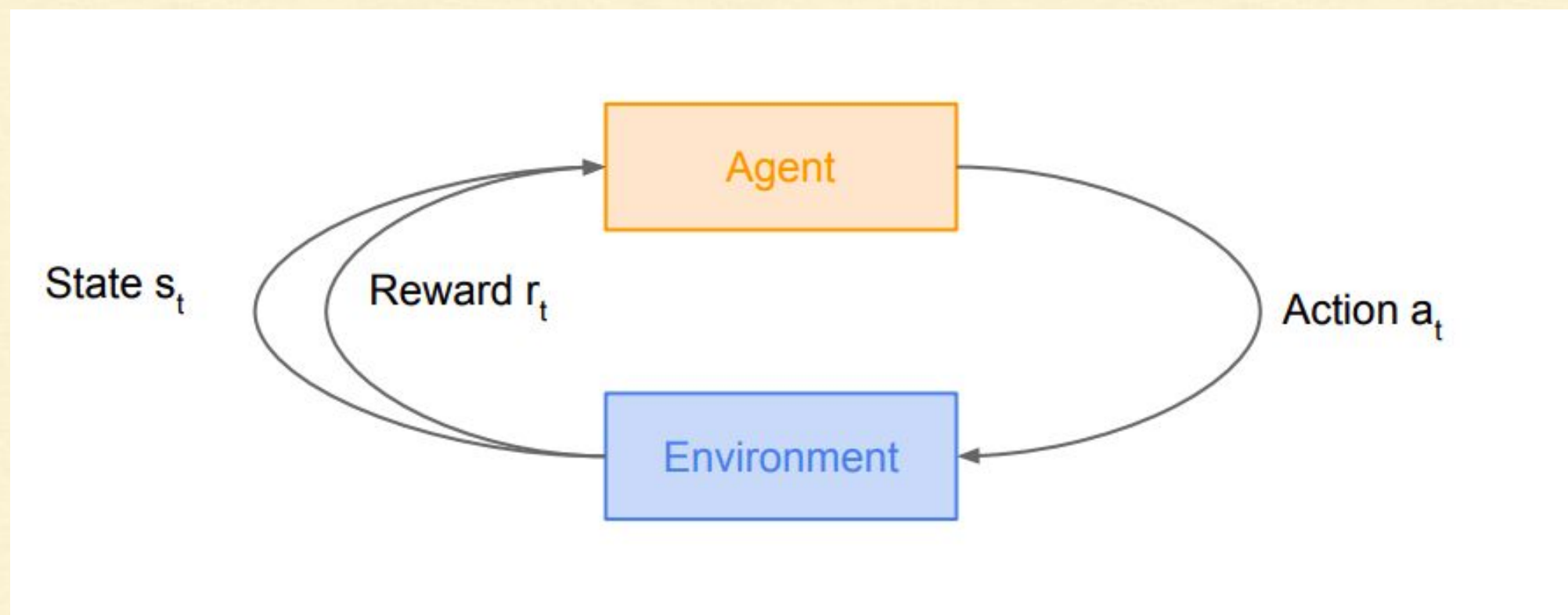
# Learning to Optimize Rewards

- In Reinforcement Learning
  - A software **agent** makes **observations** and
  - Takes **actions** within an **environment** and
  - In return it receives rewards



# Learning to Optimize Rewards

**Goal?**

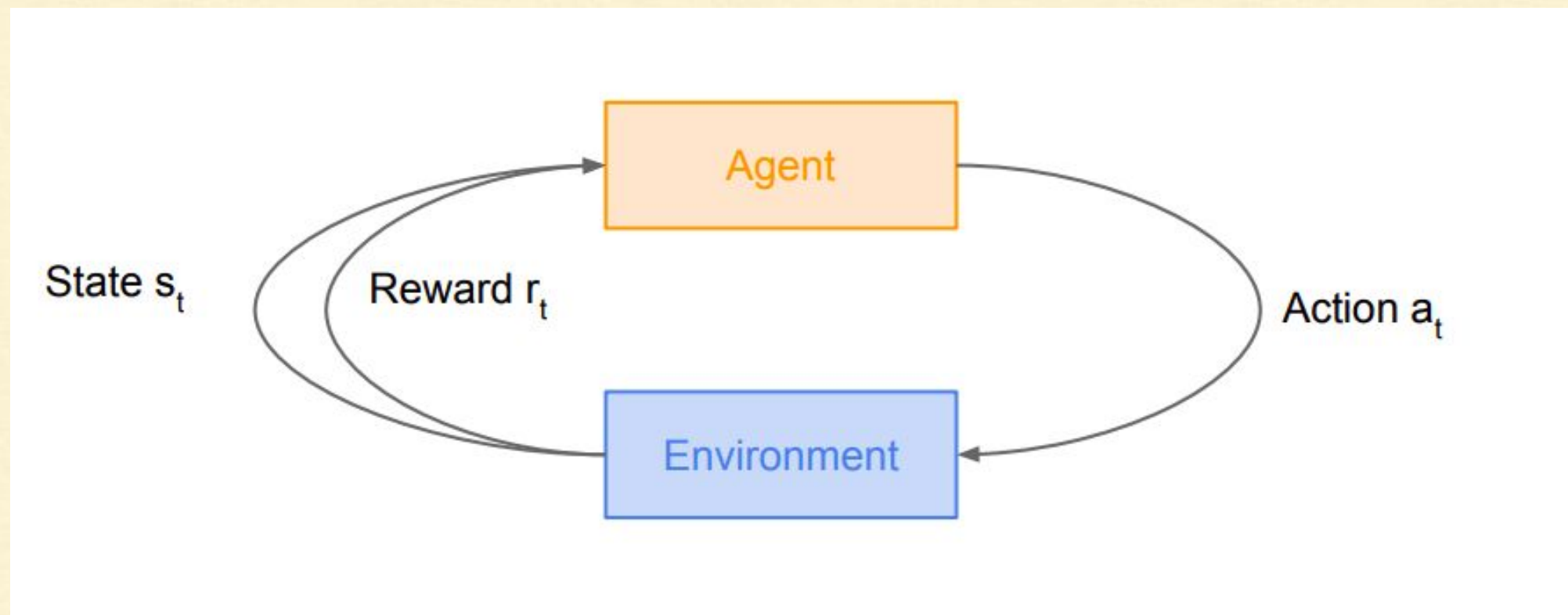




# Learning to Optimize Rewards

## Goal

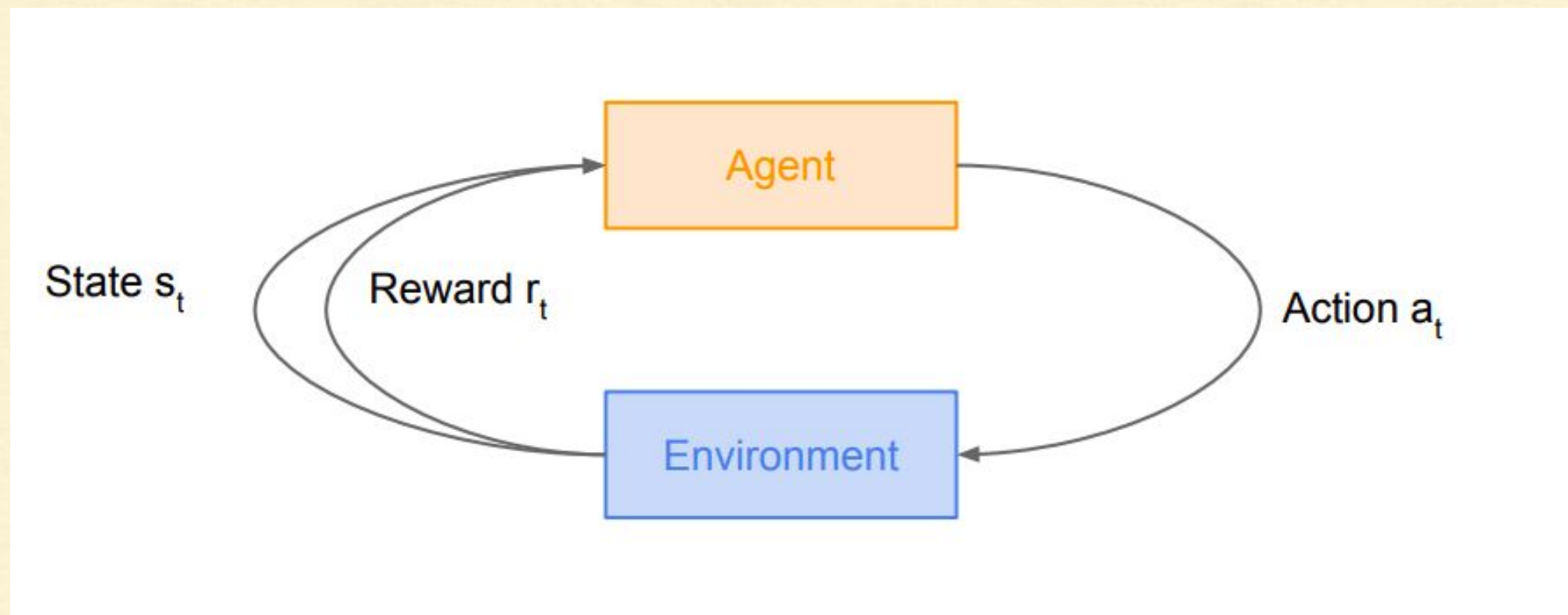
Learn how to take actions in order to maximize reward





# Learning to Optimize Rewards

- In short, the agent acts in the environment and
  - Learns by trial and error to
  - Maximize its reward



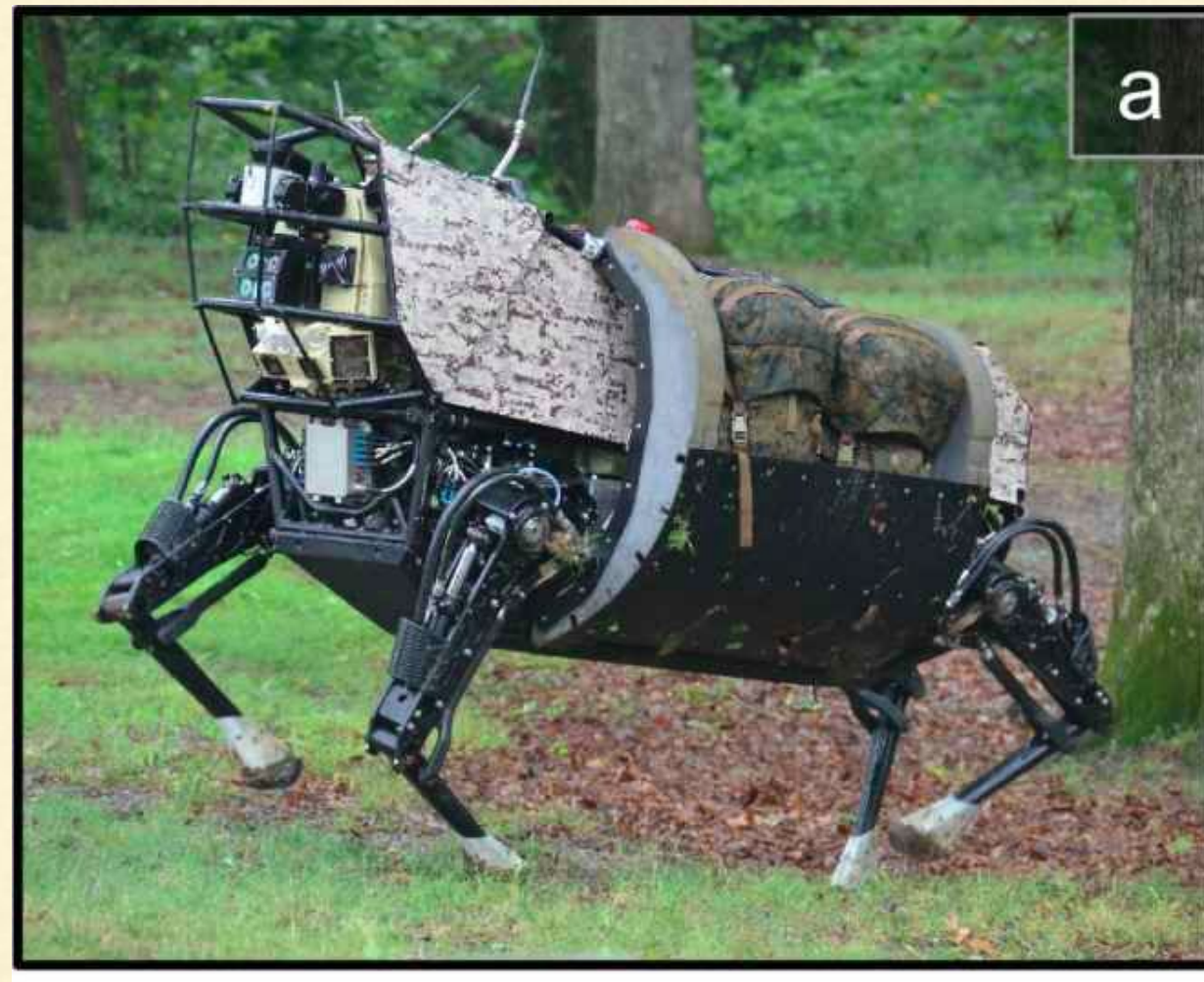
---

# Learning to Optimize Rewards

---

So how can we apply this in real-life applications?

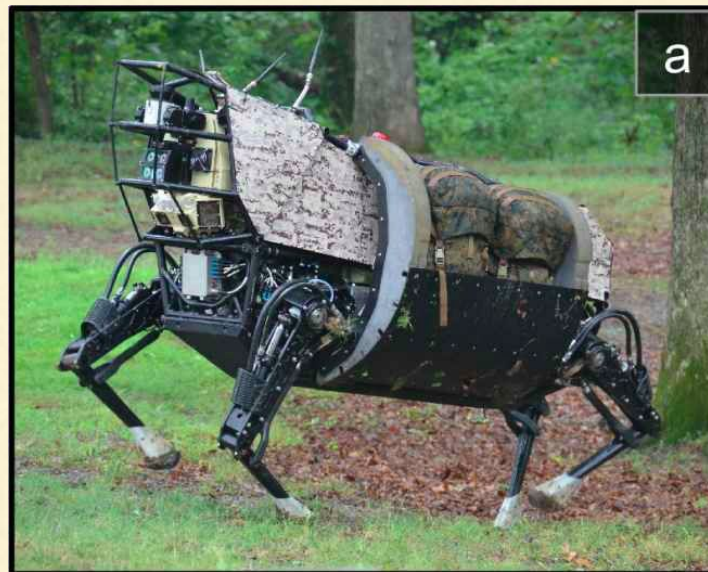
# Learning to Optimize Rewards - Walking Robot





# Learning to Optimize Rewards - Walking Robot

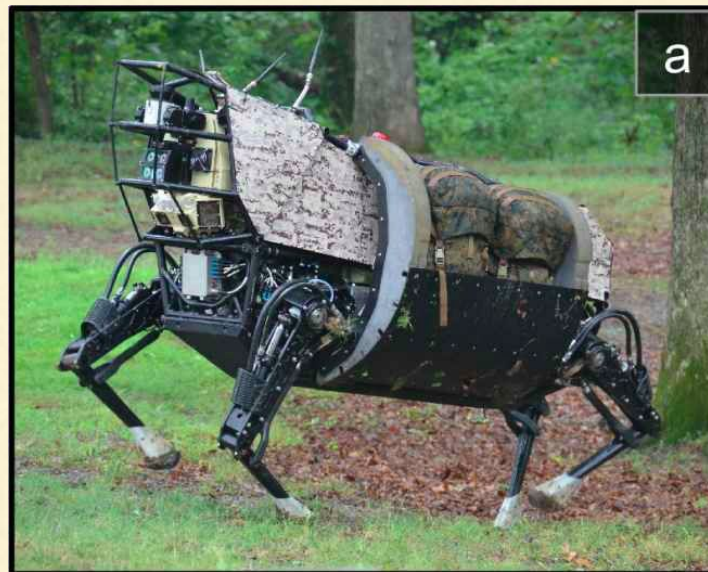
- Agent - Program controlling a walking robot
- Environment - Real world
- The agent observes the environment through a set of sensors such as
  - Cameras and touch sensors
- Actions - Sending signals to activate motors



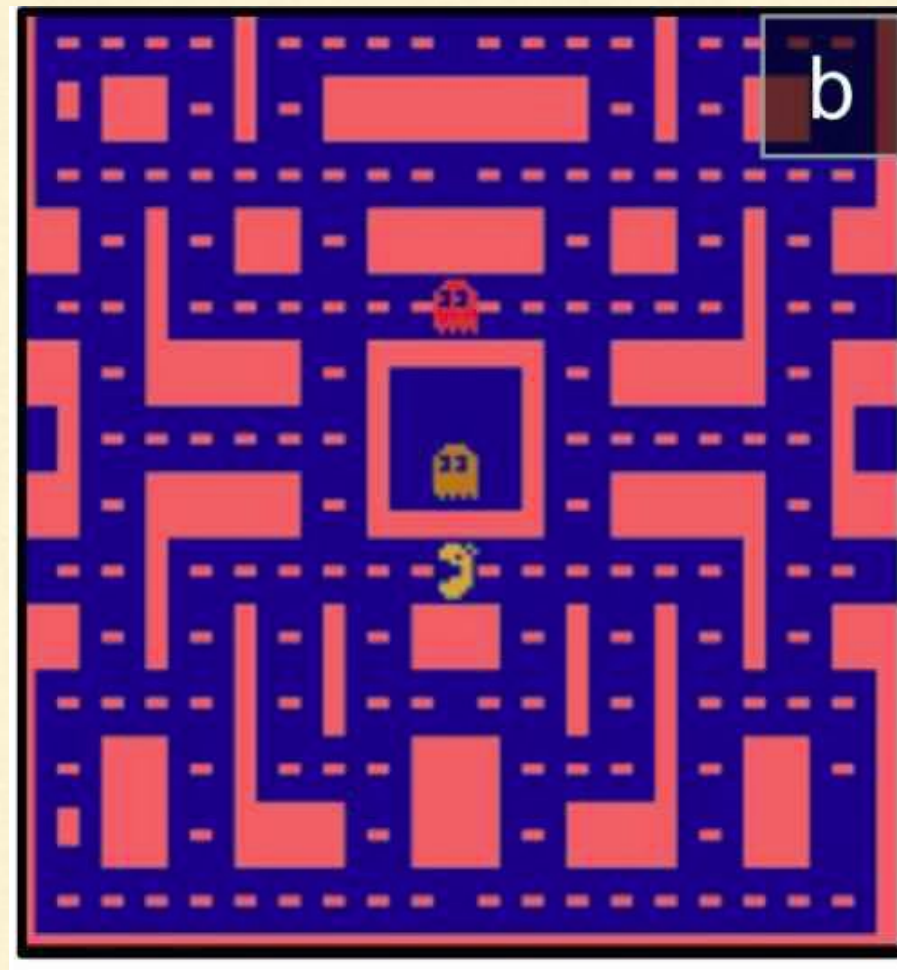


# Learning to Optimize Rewards - Walking Robot

- It may be programmed to get
  - Positive rewards whenever it approaches the target destination and
  - Negative rewards whenever it
    - Wastes time
    - Goes in the wrong direction or
    - Falls down

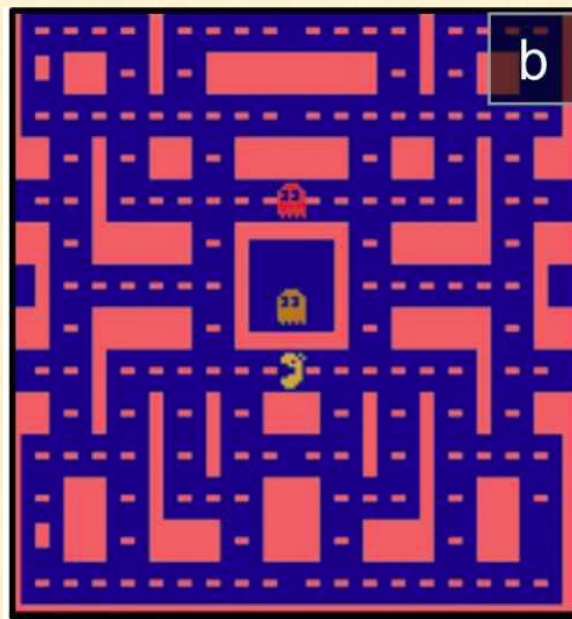


# Learning to Optimize Rewards - Ms. Pac-Man



# Learning to Optimize Rewards - Ms. Pac-Man

- Agent - Program controlling Ms. Pac-Man
- Environment - Simulation of the Atari game
- Actions - Nine possible joystick positions
- Observations - Screenshots
- Rewards - Game points





# Learning to Optimize Rewards - Thermostat





# Learning to Optimize Rewards - Thermostat

- Agent - Thermostat
  - Please note, the agent does not have to control a
  - Physically (or virtually) moving thing
- Rewards -
  - Positive rewards whenever agent is close to the target temperature
  - Negative rewards when humans need to tweak the temperature
- Important - Agent must learn to anticipate human needs



# Learning to Optimize Rewards - Auto Trader



# Learning to Optimize Rewards - Auto Trader

- Agent -
  - Observes stock market prices and
  - Decide how much to buy or sell every second
- Rewards - The monetary gains and losses





---

# Learning to Optimize Rewards

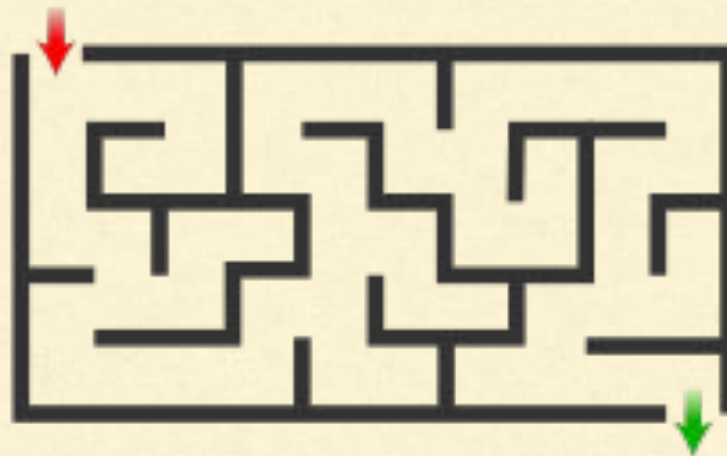
---

- There are many other examples such as
  - Self-driving cars
  - Placing ads on a web page or
  - Controlling where an image classification system
    - Should focus its attention



# Learning to Optimize Rewards

- Note that there may not be any positive rewards at all
- For example
  - The agent may move around in a maze
  - Getting a negative reward at every time step
  - So it better find the exit as quickly as possible



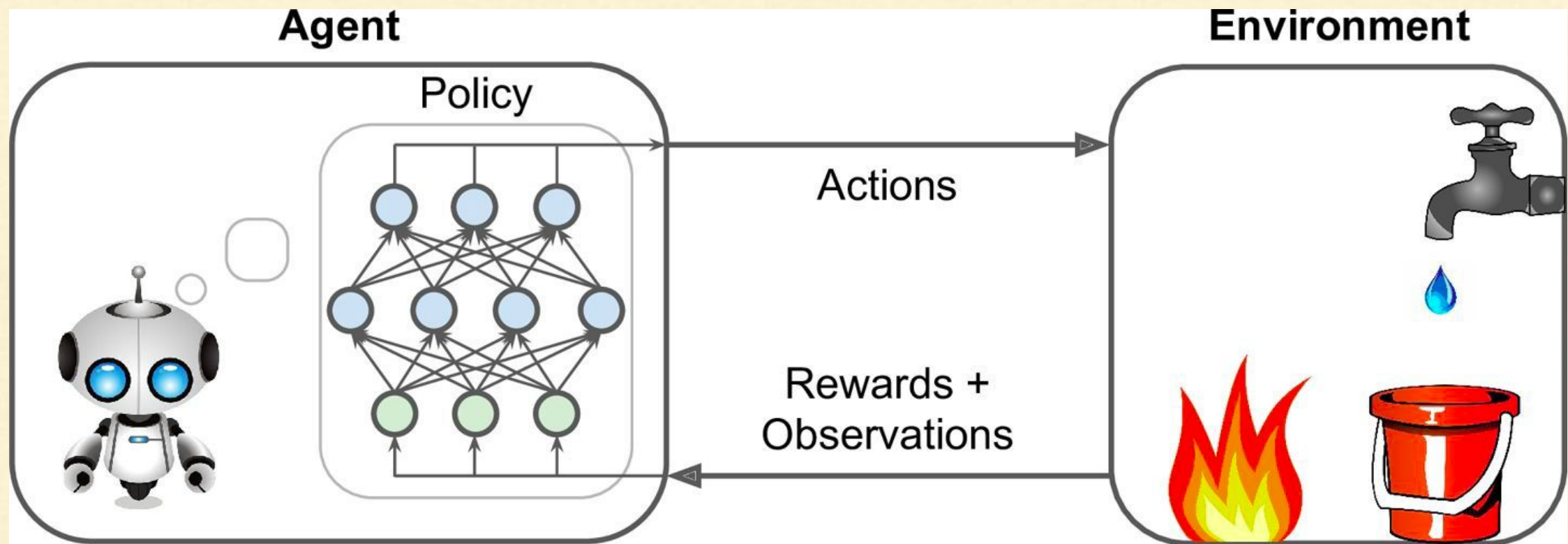
---

---

# Policy Search

# Policy Search

- The algorithm used by the software agent to
  - Determine its actions is called its **policy**
- For example, the policy could be a neural network
  - Taking observations as inputs and
  - Outputting the action to take



---

# Policy Search

---

- The policy can be any algorithm we can think of
  - And it does not even have to be deterministic
- For example, consider a robotic vacuum cleaner
  - Its 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**



---

# Policy Search

---

- The robot will have an erratic trajectory, which guarantees that
  - It can 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?

---

# Policy Search

---

How would we train such a robot?

---

# Policy Search

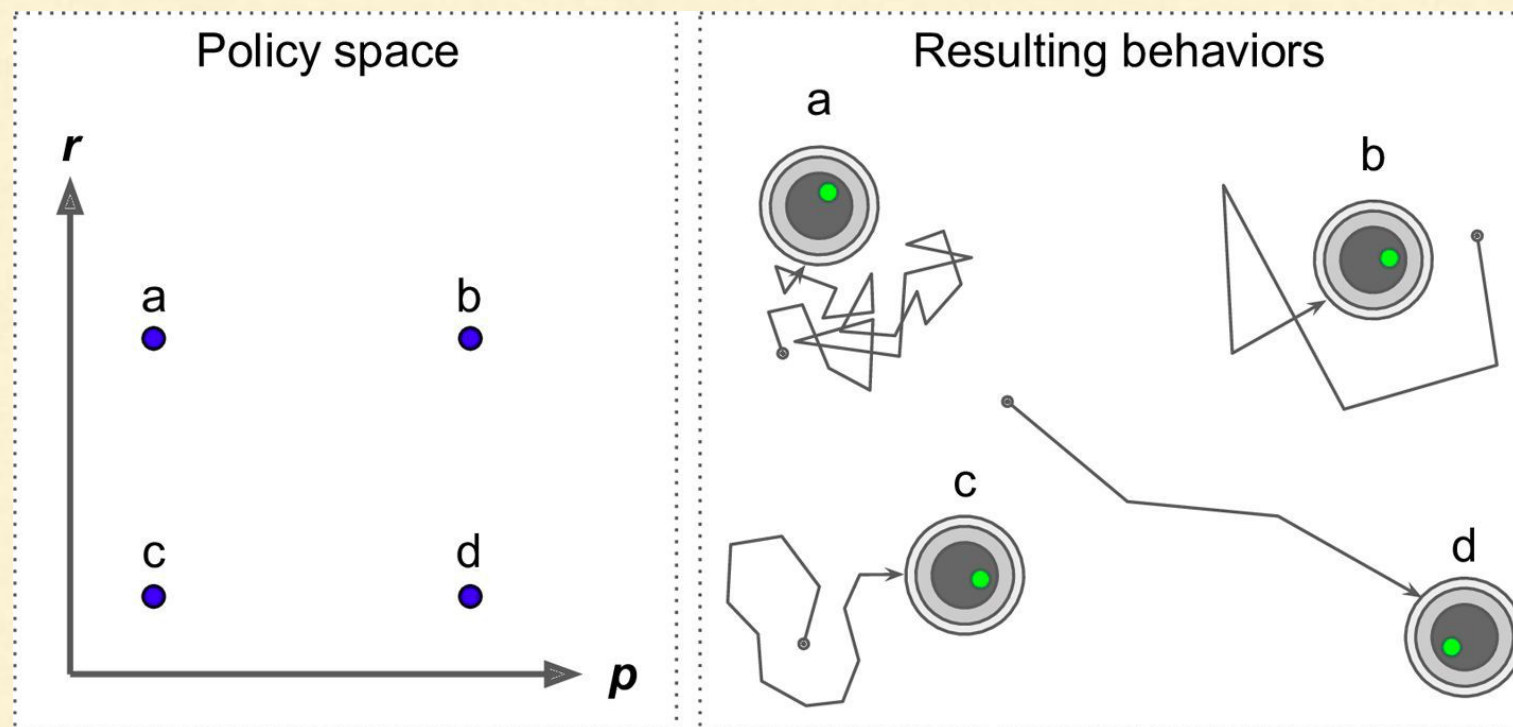
---

- There are just two policy parameters which we can tweak
  - The probability  $p$  and
  - The angle range  $r$



# Policy Search - Brute Force Approach

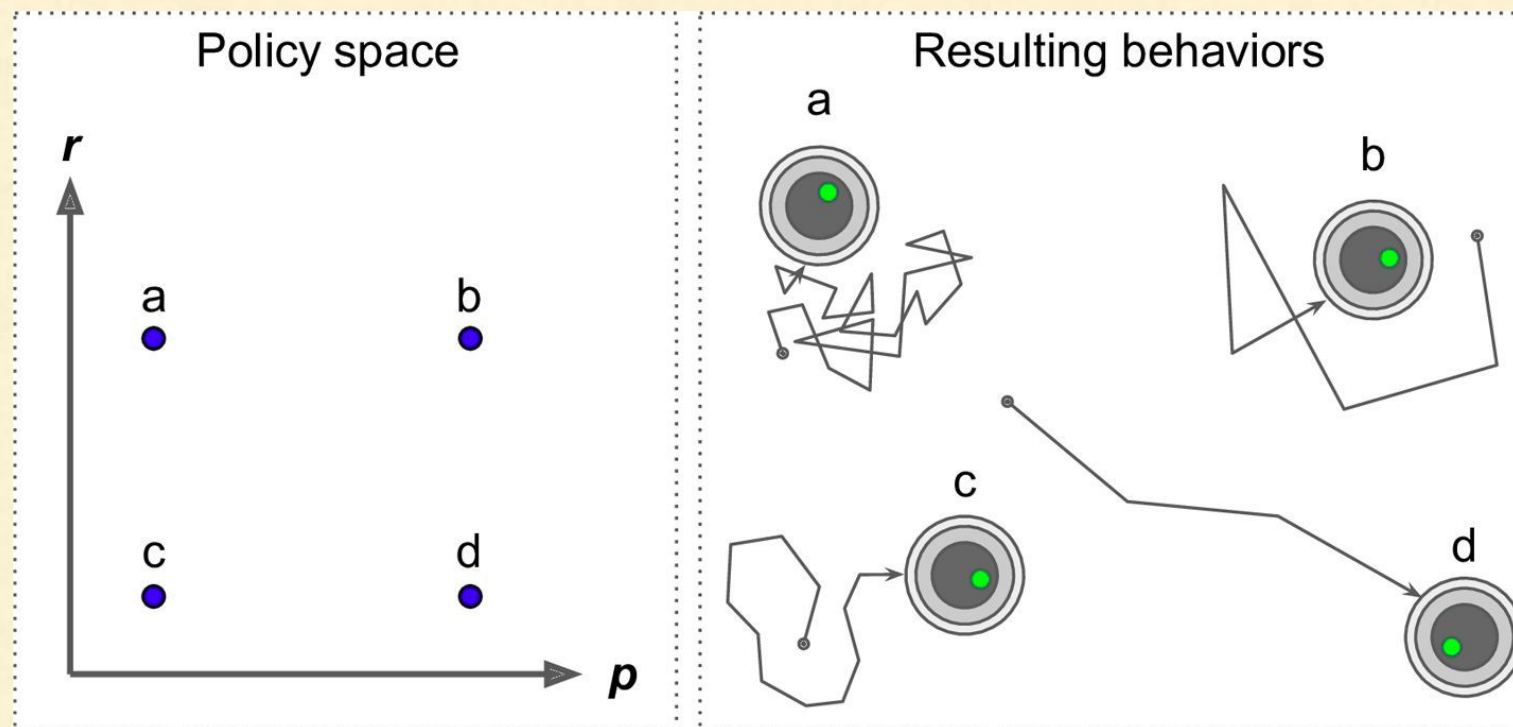
- One possible learning algorithm could be to
  - Try out many different values for these parameters, and
  - Pick the combination that performs best
- This is the example of policy search using a brute force approach



Four points in policy space and the agent's corresponding behavior

# Policy Search - Brute Force Approach

- When the policy space is too large then
  - Finding a good set of parameters this way is like
  - Searching for a needle in a gigantic haystack



Four points in policy space and the agent's corresponding behavior

---

# Policy Search - Genetic Algorithms

---

- Another way to explore the policy space is to
  - Use **genetic algorithms**
- For example
  - Randomly create a first generation of 100 policies and
  - Try them out, then “kill” the 80 worst policies and
  - Make the 20 survivors produce 4 offspring each
- An offspring is just a copy of its parent
  - Plus some random variation



---

# Policy Search - Genetic Algorithms

---

- The surviving policies plus their offspring together
  - Constitute the second generation
- Continue to iterate through generations this way
  - Until a good policy is found

---

# Policy Search - Optimization Techniques

---

- Another approach is to use
  - Optimization Techniques
- Evaluate the gradients of the rewards
  - With regards to the policy parameters
  - Then tweaking these parameters by following the
  - Gradient toward higher rewards (gradient ascent)
- This approach is called **policy gradients**

---

# Policy Search - Optimization Techniques

---

- Let's understand this with an example
- Going back to the vacuum cleaner robot example, we can
  - Slightly increase  $p$  and evaluate
  - Whether this increases the amount of dust
  - Picked up by the robot in 30 minutes
  - If it does, then increase  $p$  some more, or
  - Else reduce  $p$



---

---

# Introduction to OpenAI Gym

---

# Introduction to OpenAI Gym

---

- To train an agent in Reinforcement Learning
  - We need a working environment
- For example, if we want agent to run how to play Atari game,
  - We will need a Atari game simulator
- OpenAI gym is a toolkit that provides wide variety of simulations like
  - Atari games
  - Board games
  - 2D and 3D physical simulations and so on

---

# Policy Search - Optimization Techniques

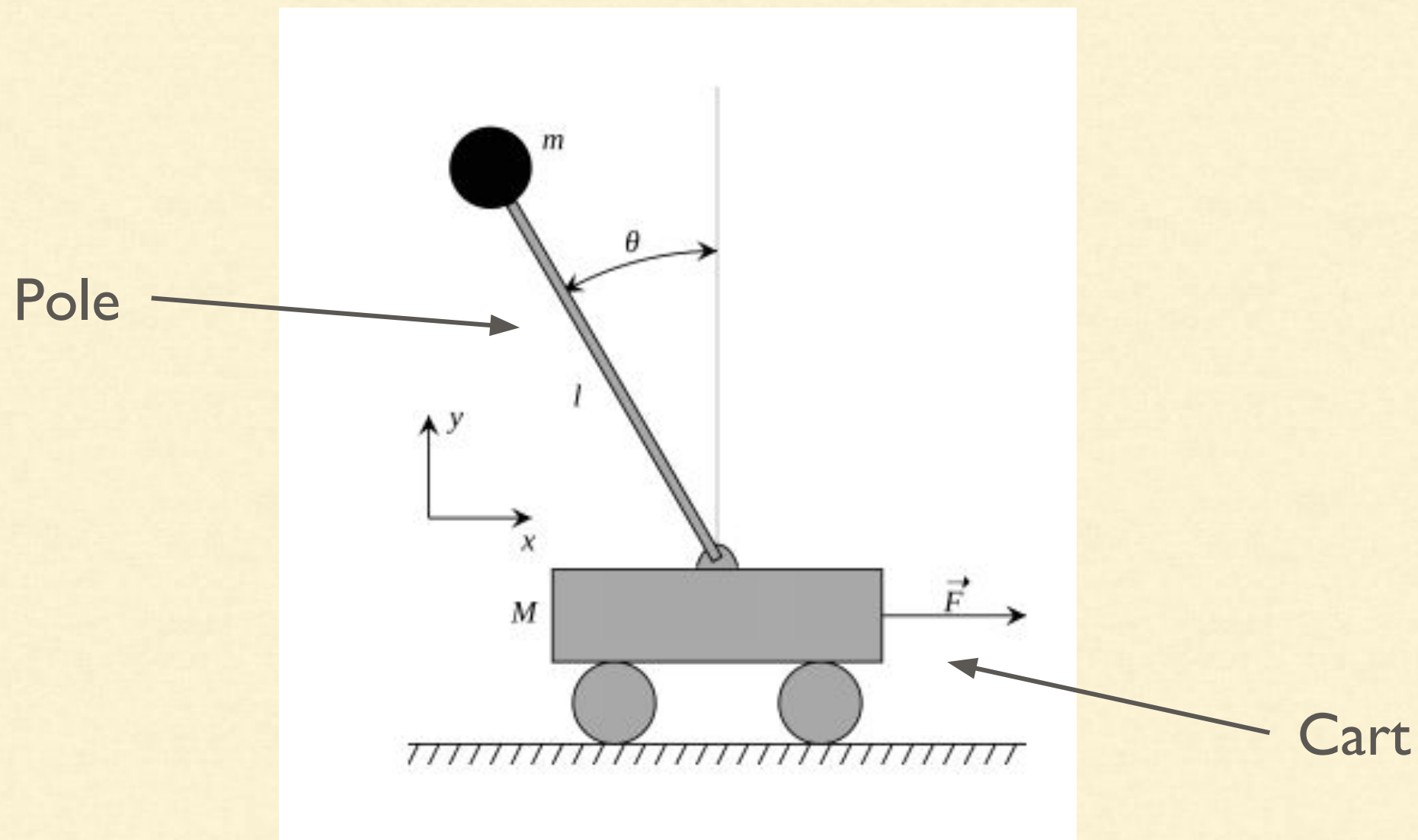
---

Let's do a hands-on



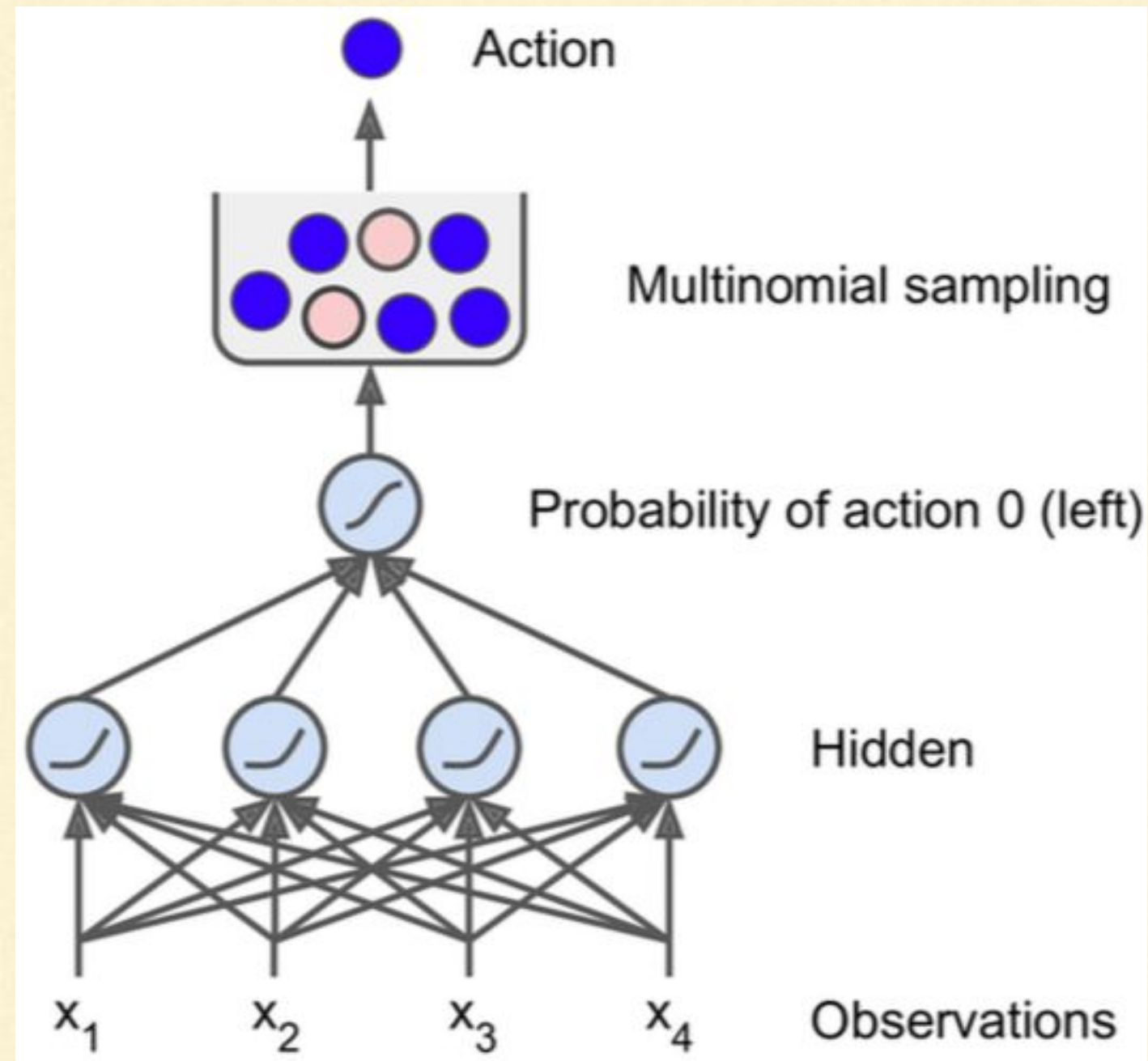
# Policy Search - Optimization Techniques

Goal - Balance a pole on top of a movable cart. Pole should be upright



# Neural Network Policies

Let's create a neural network policy.



- Take an observation as input
- Output the action to be executed
- Estimate probability for each action
- Select an action **randomly according to the estimated probabilities**

**Example, In cart pole:**

- If it outputs 0.7, then we will pick action 0 with 70% probability, and action 1 with 30% probability.

---

# Neural Network Policies

---

Q: Why we are picking a random action based on the probability given by the neural network, rather than just picking the action with the highest score.



---

# Neural Network Policies

---

Q: Why we are picking a random action based on the probability 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.**

---

# Neural Network Policies

---

Q: Why we are picking a random action based on the probability 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.**

We will never discover a new dish at restaurant if we don't try anything new. Give serendipity a chance.

---

# Neural Network Policies

---

Check the complete code in Jupyter notebook



---

## Evaluating Actions: The Credit Assignment Problem

---

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 and the target probability.

It would just be regular supervised learning.

---

## Evaluating Actions: The Credit Assignment Problem

---

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 and the target probability.

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**.

---

## Evaluating Actions: The Credit Assignment Problem

---

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.



---

## Evaluating Actions: The Credit Assignment Problem

---

***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.
- Think of a dog that gets rewarded hours after it behaved well; will it understand what it is rewarded for?

---

## Evaluating Actions: The Credit Assignment Problem

---

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 rate  $r$  at each step.
- Typical discount rates are 0.95 or 0.99.

# Evaluating Actions: The Credit Assignment Problem



discount rate  $r = 0.8$



**Actions:** Right

Right

Right

**Rewards:**

+10

0

-50

**Sum discounted  
rewards:**

-22

-40

-50

+80%

+80%

*Discount  
ratio*

$$10 + r \times 0 + r^2 \times (-50) = -22.$$



---

# Policy Gradients

---

## PG algorithms

- Optimize the parameters of a policy
- by following the gradients toward higher rewards.

One popular class of PG algorithms, called REINFORCE algorithms:

- was introduced back in 1992 by Ronald Williams. Here is one common variant:

---

# Policy Gradients

---

Here is one common variant of REINFORCE:

- I. Let the neural network policy play the game several times
  - a. Compute the gradients that would make the chosen action even more likely,
  - b. but don't apply these gradients yet.

---

# Policy Gradients

---

Here is one common variant of REINFORCE:

1. Let the neural network policy play the game several times
  - a. Compute the gradients that would make the chosen action even more likely,
  - b. but don't apply these gradients yet.
2. Once you have run several episodes, compute each action's score (using the method described earlier).



---

# Policy Gradients

---

Here is one common variant of REINFORCE:

3.

If an **action's score is positive**, it means that the action was good and you want to **apply the gradients** computed earlier to make the action even more likely to be chosen in the future.

However, if the **score is negative**, it means the action was bad and you want to **apply the opposite gradients** to make this action slightly less likely in the future. The solution is simply to multiply each gradient vector by the corresponding action's score.

---

# Policy Gradients

---

Here is one common variant of REINFORCE:

4. Finally, compute the mean of all the resulting gradient vectors, and use it to perform a Gradient Descent step.

---

# Policy Gradients

---

Check the complete code in Jupyter notebook



---

# Reinforcement Learning

---

## Further Learning

1. [Markov Decision Processes](#)
2. [Temporal Difference Learning](#)
3. [Q-Learning](#)
4. Try [Deep Reinforcement Learning in Pac-man](#)

---

---

**Machine Learning Check List:**

**<https://tinyurl.com/ybj69s3n>**

# Questions?

---

<https://discuss.cloudxlab.com>

[reachus@cloudxlab.com](mailto:reachus@cloudxlab.com)

