Reinforcement Learning

2018-04-12

Andrey Markov

Announcements

- Extra Credit 1: MOBA competition: Due Apr 22, 23:55
- Extra Credit 2
  - The plan is to run the study the weeks of April 16th and 23rd.
  - https://www.cc.gatech.edu/~surban6/2018sp-gameAI/extra-credit.html
  - https://tinyurl.com/level-design-study-times
- HW7 done; HW 8 is posted (also due Apr 22, 23:55)

N-3: Player models

1. What is a player model? What does it allow?
2. What are two high-level categories of modeling?
3. What are a couple major types within the first category?
4. What are ways to get a player model?

N-2: PCG concluded

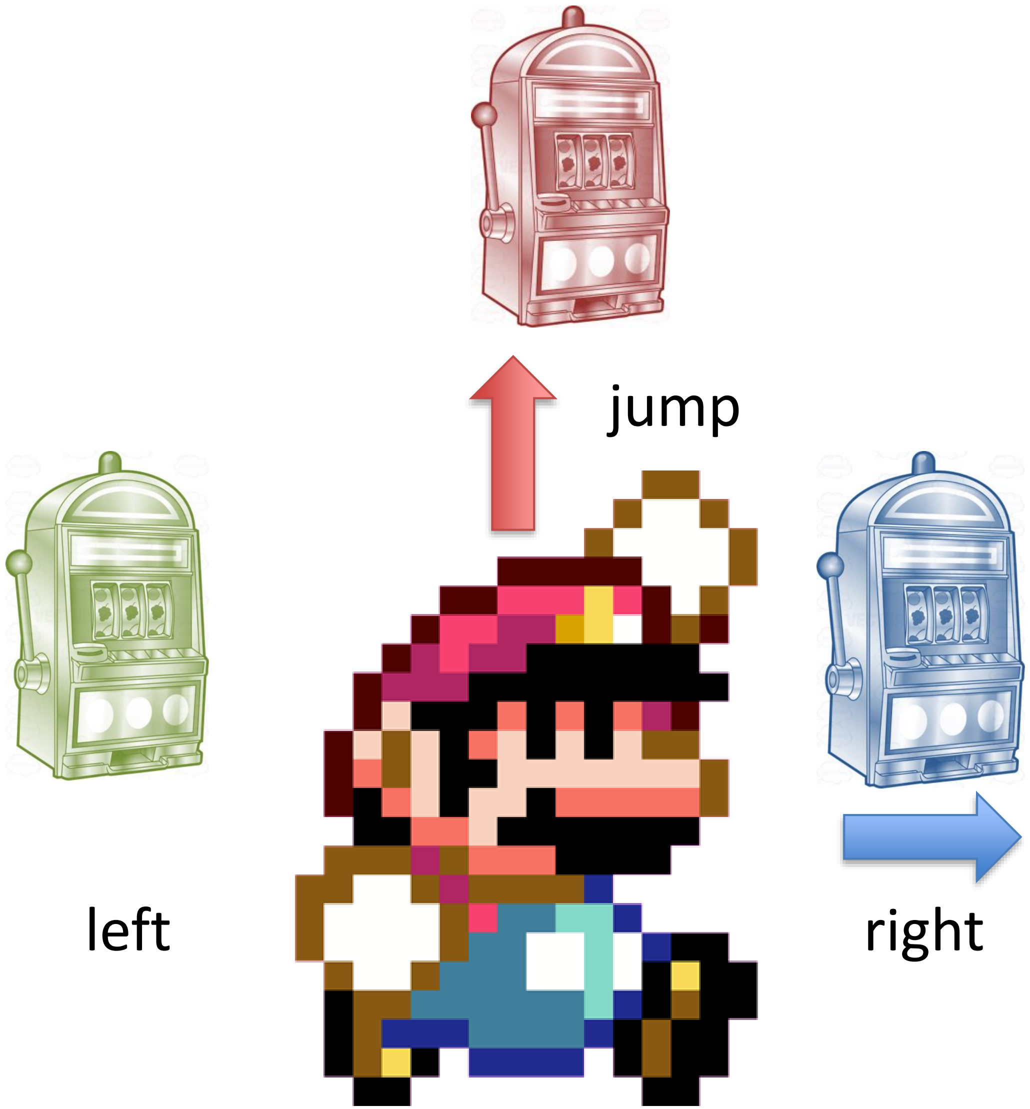
1. What are the 4 high-level forms of PCG we discussed?
2. We described cellular automaton and agents as _______
3. **Discuss the relationship of the evaluation/fitness function and (a) player models, and (b) designer preferences**
4. L-systems are a form of ____, and are particularly useful for ____
5. What is the rewriting order of L-systems, and why does this matter?
6. Explain collapse and propagation in CSP, and the effect on search
7. What class of techniques have we seen before that seem particularly suited to quest and story generation?

Situating RL

- The BLUF: RL is about learning from interaction about how to map situations to actions to maximize reward. Key elements:
  - Trial-and-error search: must discover which actions yield most reward by trying them
  - Delayed reward: actions may affect both immediate reward and also next situation and all subsequent rewards
- Different from:
  - Supervised learning: training set of labeled examples
  - Unsupervised learning: finding latent structure in unlabeled data
- RL: maximize reward signal rather than discover hidden structure

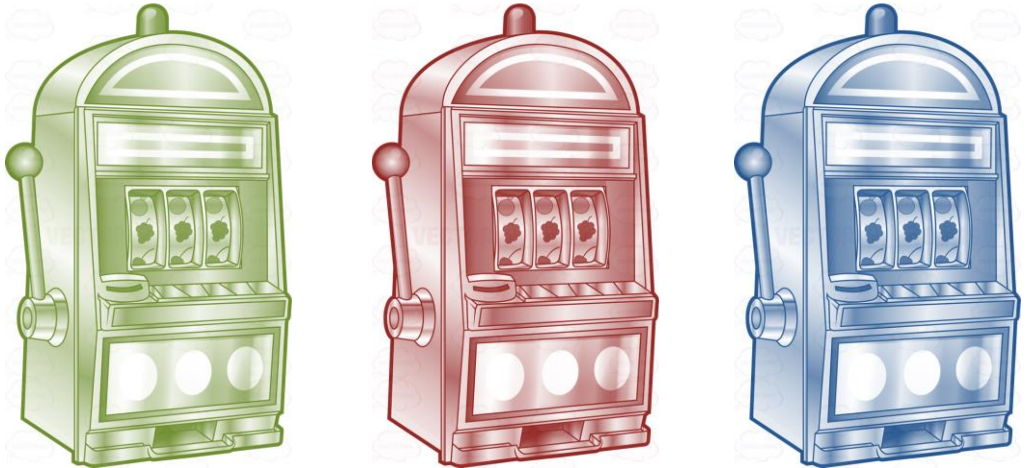
High-level Idea

If the multi-armed bandit problem was a single state MDP, we can think of learning a strategy to play a game as solving this problem for *every state of the game*



Now what?

- We can’t just keep playing one bandit to figure out its potential for reward (money)
- Want to maximize reward across all bandits
- We need to trade off making money with current knowledge and gaining knowledge
  - *Exploitation vs. Exploration*



Epsilon (ϵ) Greedy
(ϵ-Greedy)

(ϵ=0.1)

- 90% of the time try the best machine according to our current beliefs
- 10% of the time take random action

Decaying: Drop ϵ lower and lower according to some schedule

Now we can "escape" from early greedy mistakes!

State Representation Examples

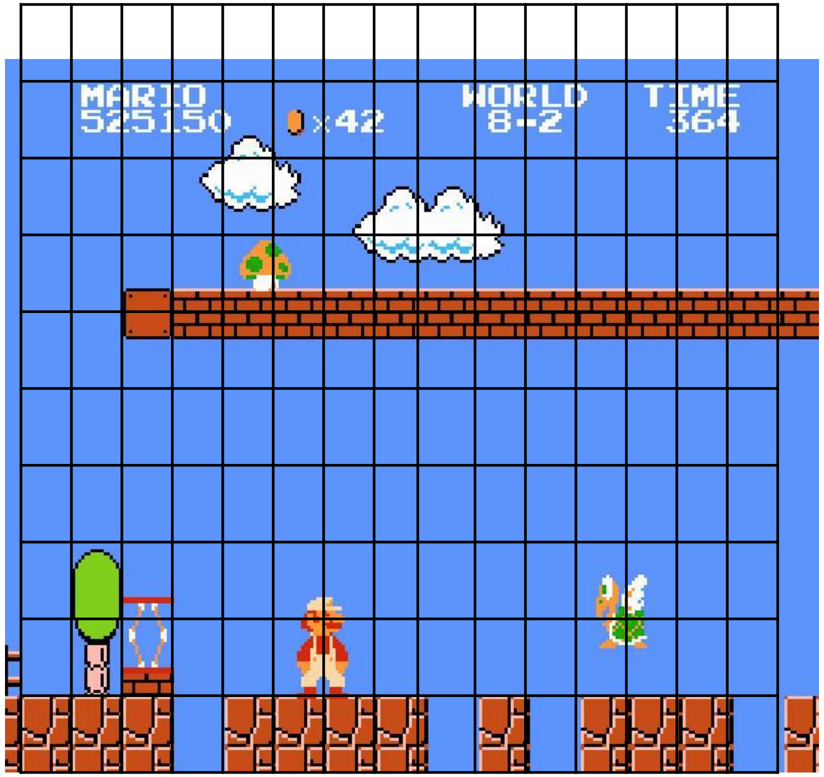
List of facts

- -enemy to right
- -powerup above
- -fire mario
- -on ground
- -etc…

State Representation Examples

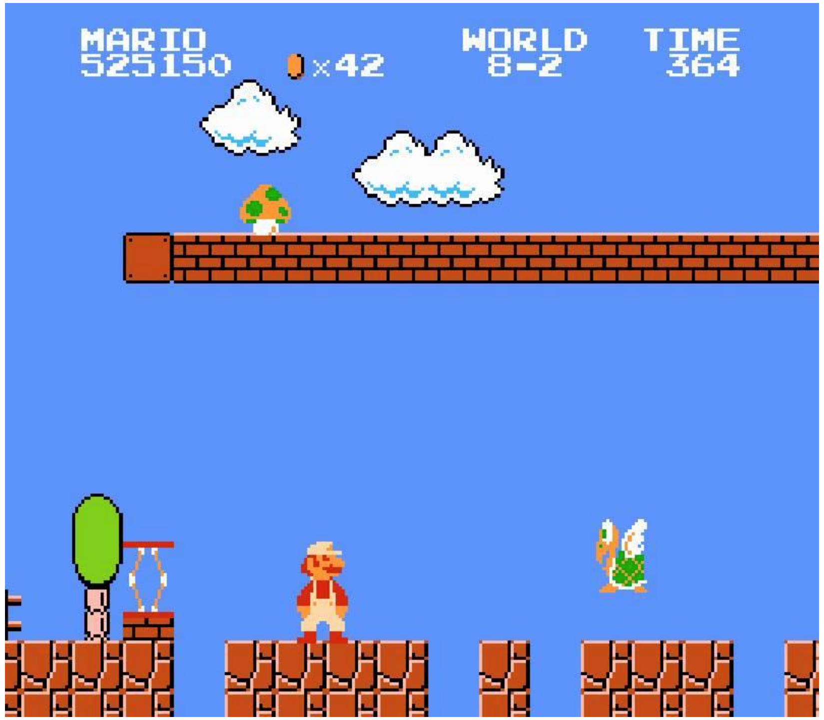
Grid Representations

- Segment all locations to tiles
- All enemies represented individually? Or just as "enemy"
- How much of the level to include in the state? Just screen? Smaller? Larger?



State Representation Examples

What pixels are present on the screen at this time?



State Representation Tradeoffs

- More complex state representations ensure that an agent has all info needed
- Less complex state representations speed up training (more states “look” the same)

- Goal: find the middle ground. Simplest state representation that still allows for optimal performance

Question

What state representations would you use for the following? Why?

- Tic-Tac-Toe
- A simple platformer
- A real time strategy game (Civilization/Starcraft)

Possible Answers

- Tic-Tac-Toe: The Gameboard
- Simple platformer: grid of screen width and height, with values for collideable elements, breakable elements, empty spaces, and enemy types (flying, etc)
- RTS: World map (?) + list of facts about currently running commands (building troops, moving troops, upgrading building, etc) (?)

Reinforcement Learning: An introduction.
Richard Sutton and Andrew Bartow, 2nd ed, MIT Press, 2017.
http://incompleteideas.net/book/the-book-2nd.html

REINFORCEMENT LEARNING

Reinforcement Learning: An Introduction

Richard S. Sutton
and Andrew G. Barto

Second Edition, in progress
MIT Press, Cambridge, MA, 2017

Online draft New Code Solutions Course Materials

16 Applications and Case Studies

http://incompleteideas.net/book/the-book-2nd.html

Some RL Successes

- Learned the world's best player of Backgammon (Tesauro 1995)
- Learned acrobatic helicopter autopilots (Ng, Abbeel, Coates et al 2006+)
- Widely used in the placement and selection of advertisements and pages on the web (e.g., A-B tests)
- Used to make strategic decisions (DD) in Jeopardy! (IBM's Watson 2011)
- Achieved human-level performance on Atari games from pixel-level visual input, in conjunction with deep learning (Google Deepmind 2015)

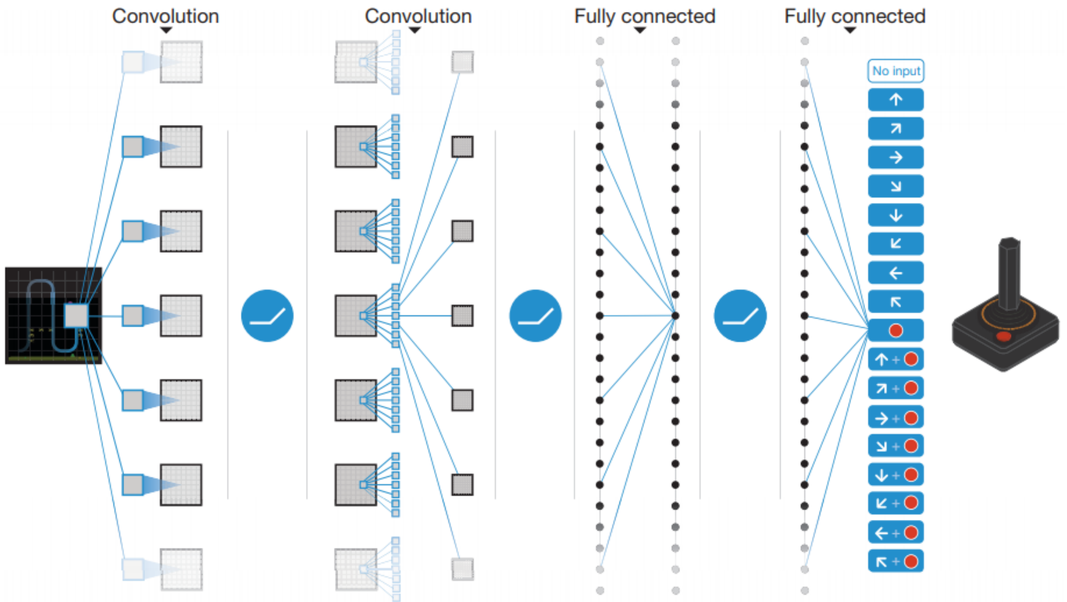
In all these cases, performance was better than could be obtained by any other method, and was obtained without human instruction

Credit: Sutton & Barto
http://incompleteideas.net/609%20dropbox/slides%20(pdf%20and%20keynote)/1-intro-and-singularity.pdf

RL + Deep Learning, applied to Classic Atari Games

Google Deepmind 2015, Bowling et al. 2012

- Learned to play 49 games for the Atari 2600 game console, without labels or human input, from self-play and the score alone
- Learned to play better than all previous algorithms and at human level for more than half the games; same alg applied to all 49, without human tuning



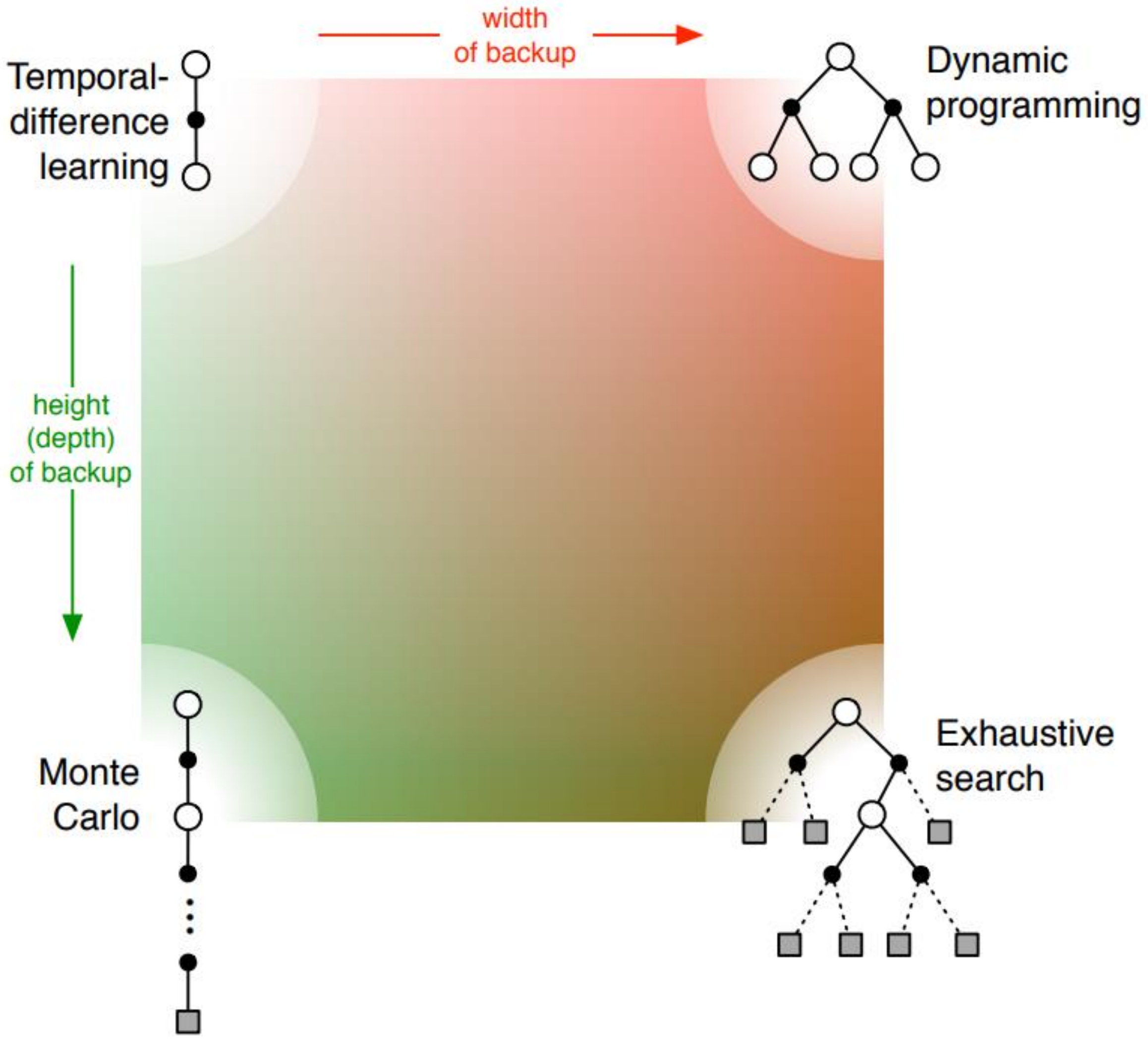
Credit: Sutton & Barto

Fundamental Classes of Methods

- Dynamic programming
  - Mathematically well understood but require a complete and accurate model of the environment
- Monte Carlo methods
  - Model free and conceptually simple, but not well suited for step-by-step incremental computation
- Temporal-difference learning
  - Model free and fully incremental, but difficult to analyze

Also differ in efficiency/speed of convergence

Unified View



Credit: R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction

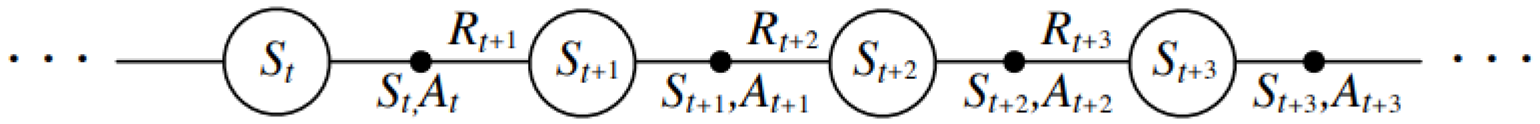
Markovian State

- Multi-armed bandit problem is a “single state” or “stateless” **Markov Decision Process**
- Markov property: given the present, **the future does not depend on the past**. IE ‘memoryless’
- We call a state representation Markovian if it has **all the information we need to make an optimal decision**

Andrey Markov

Markov Decision Process

- *S*: finite set of states
- *A*: finite set of actions
- $P(s_1|s,a)$: Probability action a takes us from state s to state s_1
- $R(s,s_1)$: Reward for transitioning from state s to state s_1
- *y:* Discount factor ([0-1]). Demonstrates the difference in importance between future awards and present awards



http://incompleteideas.net/609%20dropbox/slides%20(pdf%20and%20keynote)/11-12-TD.pdf

Markov Decision Process

- *S*: finite set of states
- ***A*: finite set of actions**
  - **What are they in Tic-Tac-Toe? Platformers? RTS?**
- $P(s_1|s,a)$: Probability action a takes us from state s to state s_1
- $R(s,s_1)$: **Reward** for transitioning from state s to state s_1
- γ: Discount factor ([0-1]). Demonstrates the difference in importance between future awards and present awards

What is the optimal action a *to take for every state* s?

Goal of the MDP

- Find the optimal action *a* to take for every state *s*
- We refer to this strategy as the <u>policy</u>

- <u>Policy</u> is represented as **π(s)**
  - What is the optimal action to take in state s?

MDP: Reward Function

- *$R(s,s_1)$*: **Reward** for transitioning from state s to state s_1
- Better understood as “feedback”
  - An MDP’s reward can be positive or negative

- How to give reward? And what reward to give?

MDP: Transition function

- *$P(s_1|s,a)$*: Probability action a takes us from state s to state s_1
- Also sometimes called a forward model
- Gives probability of transition from one state to another given a particular action
- Typically this is defined at a high level, rather for an individual state
  - E.g. When trying to move forward there is a 50% change of moving forward, a 25% change of moving to the left and a 25% chance of moving to the right
  - Not “when mario is at position (10,15) there is a 80% chance moving right gets mario to (11,15)

Question

Give me all the pieces needed to run an MDP for the *simple* game of your choice

S is the set of all states

A is the set of all actions

P is state transition function specifying P(s'|s,a)

R is a reward function R(s,a,s')

Pros/Cons

Pros

- Once learning is done running we have an agent that can act optimally in any state very, very quickly (table lookup)

Cons:

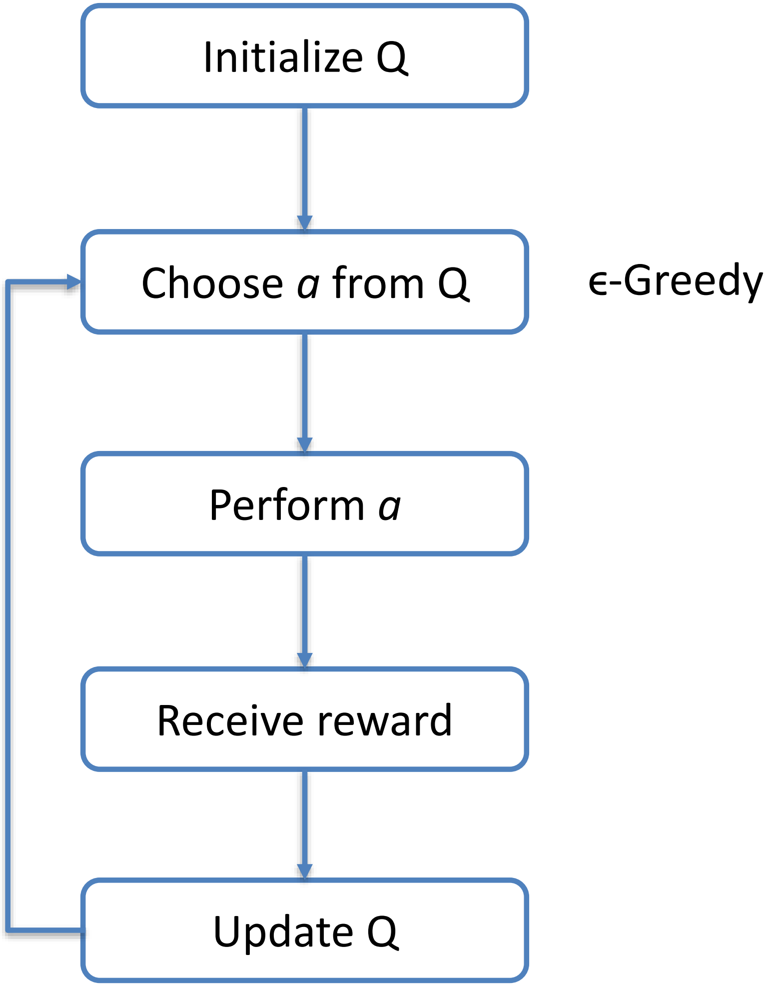
- **What if we don't have a transition function**? IE we don't know probability of getting from s to s'?
  - What if we have a black box that allows us to simulate it…

Q-learning

- Model-free, TD learning
  - Well… states and actions still needed
  - Learn from history of interaction with environment
- The learned action-value function Q directly approximates the optimal one, independent of the policy being followed
- Q: S x A → R
  - This is what we are learning!
  - Iteratively approximating best action a in state s to maximize cumulative reward

Action

$$\begin{array}{c|c} \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$



https://youtu.be/aCEvtRtNO-M?t=205
https://www.youtube.com/watch?v=79pmNdyxEGo

Q-learning

- *S*: finite set of states
- *A*: finite set of actions
- ~~*P(s,*~~$\cancel{s_1}$~~*)*: Probability that action a takes us from state s to state~~ $\cancel{s_1}$
- Environment in which to act (simulated or real)
- *R(s,a)*: Reward function mapping s,a to a real value
- γ*:* Discount factor ([0-1]). Demonstrates the difference in importance between future awards and present awards

What is the optimal action a *to take for every state* s?

See page 131 in S&B book linked earlier

Q-learning algorithm

Assign Q[S,A] arbitrarily

observe current state **s**

repeat

  select and carry out an action **a** ← ϵ-Greedy

  observe reward **r** and state **s'**

  $Q[s,a] \leftarrow Q[s,a] + \alpha * (r+\gamma * \underbrace{\max_{a'} Q[s',a']}_{\text{the max value within } Q[s',*]} - Q[s,a])$

until termination

learning rate, α, ranges (0,1]
  1 if env is deterministic
  often 0.1
discount factor, γ, ranges [0,1]
  0 is myopic, 1 long-term

the learned action-value function, Q, directly approximates the optimal action-value function, independent of the policy being followed

Q Learning

Pros

- We don’t need to have a forward model/transition function
- Means we can go into an environment blind

Cons

- Memory use: we have to hold a |S|*|A| table
- Bigger table means (at times) longer to converge

No free lunch

“One of the greatest challenges in applying reinforcement learning to real-world problems is deciding **how to represent and store value functions and/or policies**. Unless the state set is finite and small enough to allow exhaustive representation by a lookup table […] **one must use a parameterized function approximation scheme**. […]

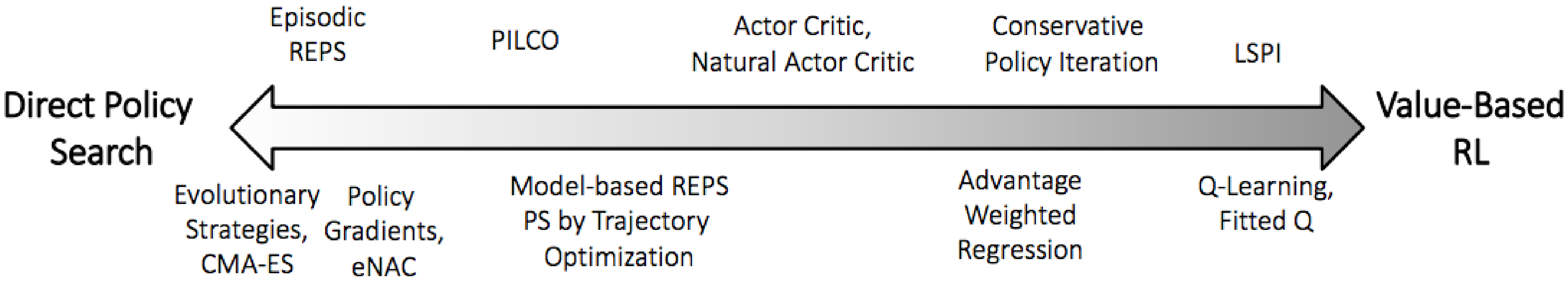
Most successful applications of reinforcement learning **owe much to sets of features carefully handcrafted based on human knowledge** and intuition about the specific problem to be tackled. […]

in all the examples of which we are aware, the most impressive demonstrations required the network's input to be **represented in terms of specialized features handcrafted for the given problem**”

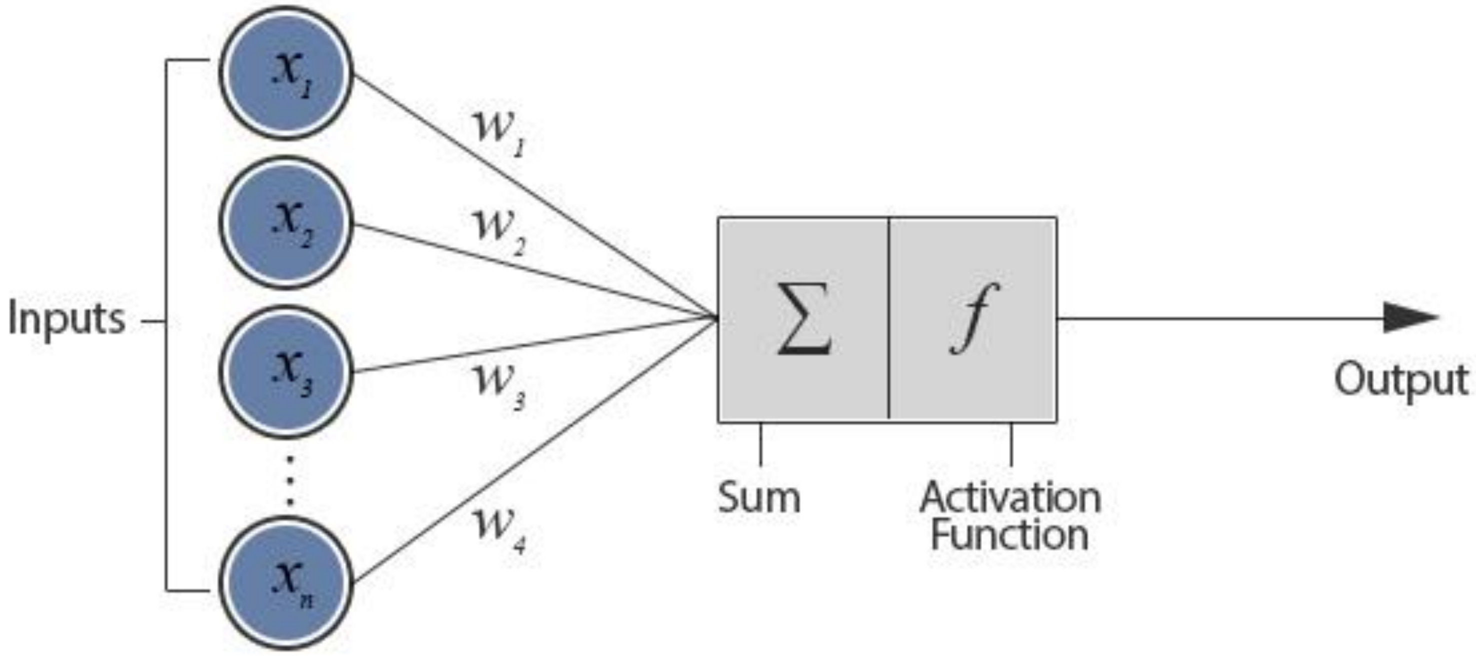
Sutton & Barto, 2nd ed, section 16.5

Q Learning

- The most general game playing/decision making technique we've seen so far
- Only needs a reward function, list of actions, and state representation (besides env & discount)
- But that state space sure is too massive to be useful huh?

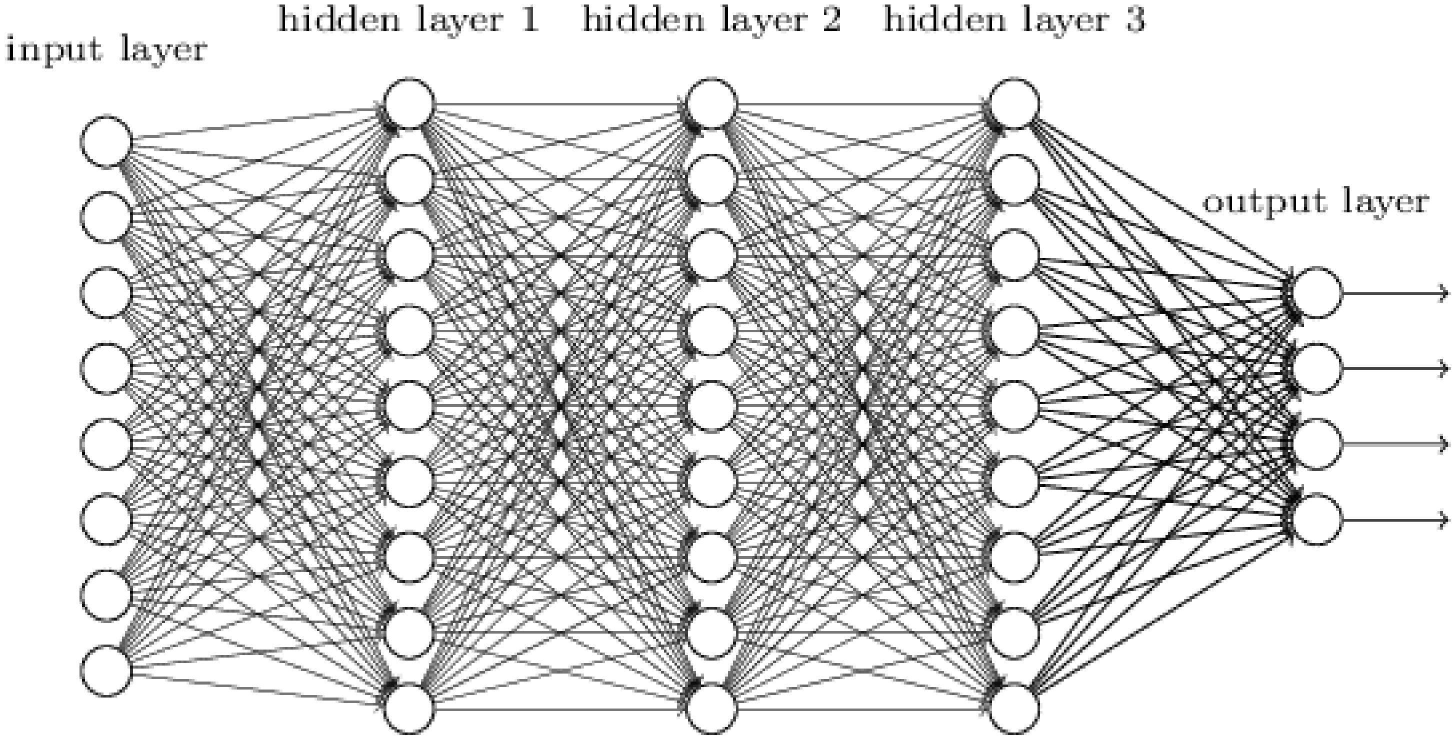


Artificial Neural Nets

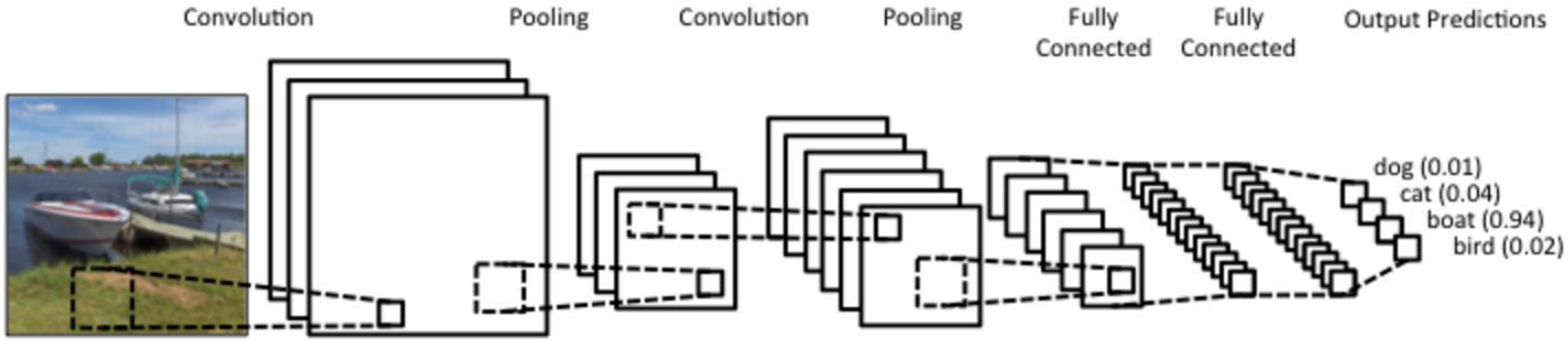


Warren McCulloch and Walter Pitts (1943)

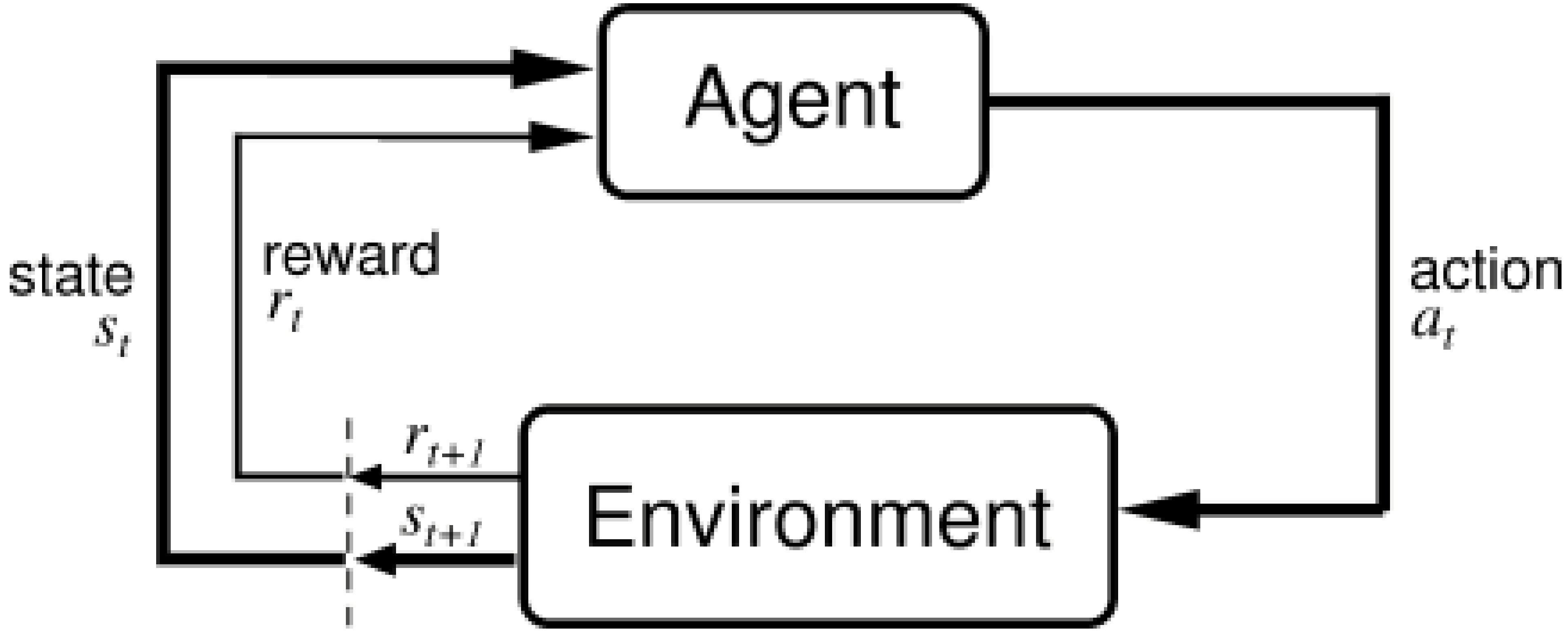
Deep Neural Networks



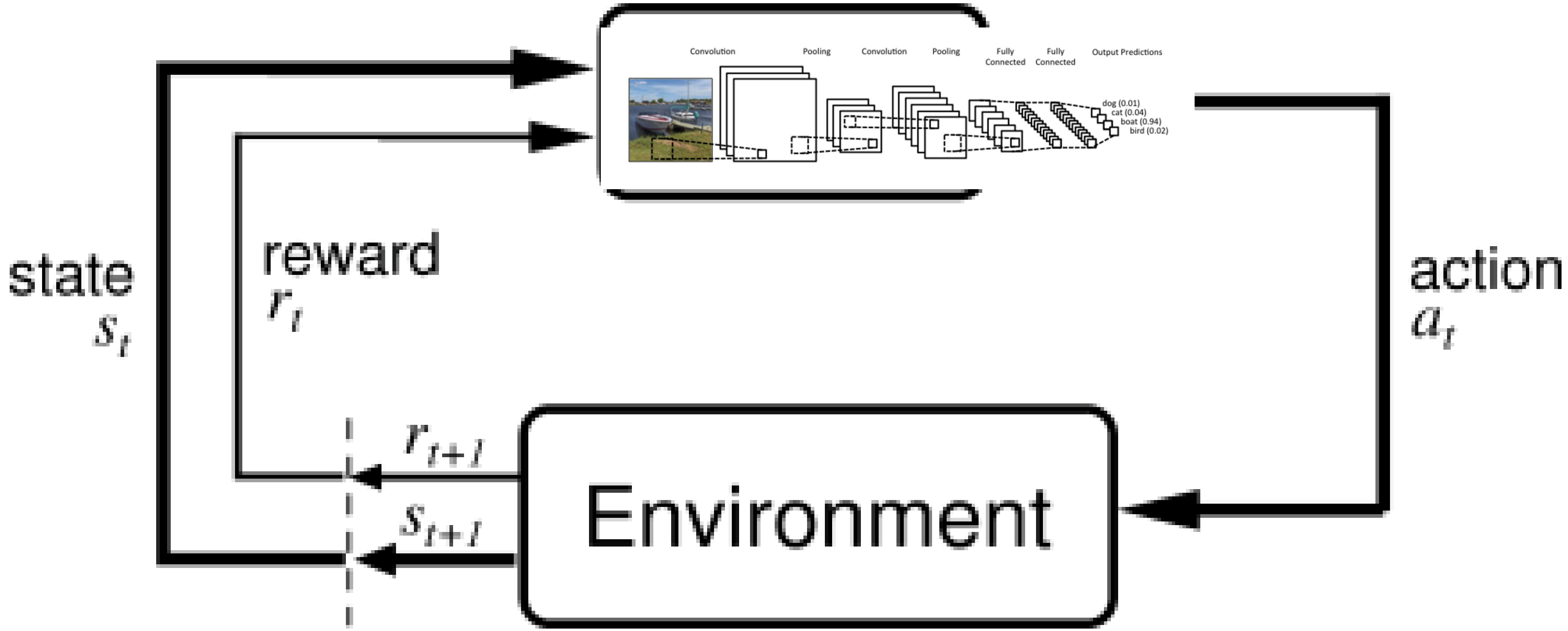
Convolutional Neural Networks



So what if we…



So what if we…


Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)
state
s_t
reward
r_t
action
a_t
r_{t+1}
s_{t+1}
Environment

Deep Q-learning

- Our CNN now acts as our Q table, transforming from the input image/state and giving the action vector for that state
- We can then give it feedback in terms of how off it was from the “true” quality of taking the suggested actions

DeepMind

- https://youtu.be/V1eYniJ0Rnk?t=25
- https://www.youtube.com/watch?v=oo0TraGu6QY

Deep Q Learning Comparisons

Pro:

- Massively cuts back on search space
- Massively speeds up learning
- CNN almost as fast as look up table

Con:

- Needs a *lot* of training data to perform well
- Some state spaces are still too complicated (real life)