Reinforcement Learning

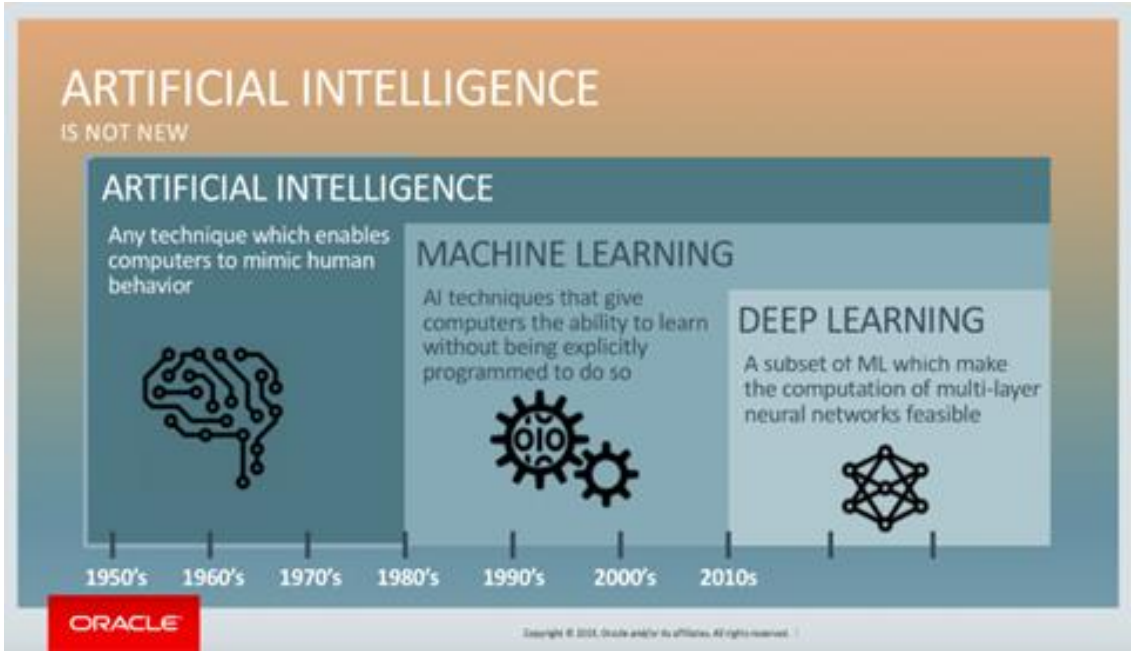
2019-11-18

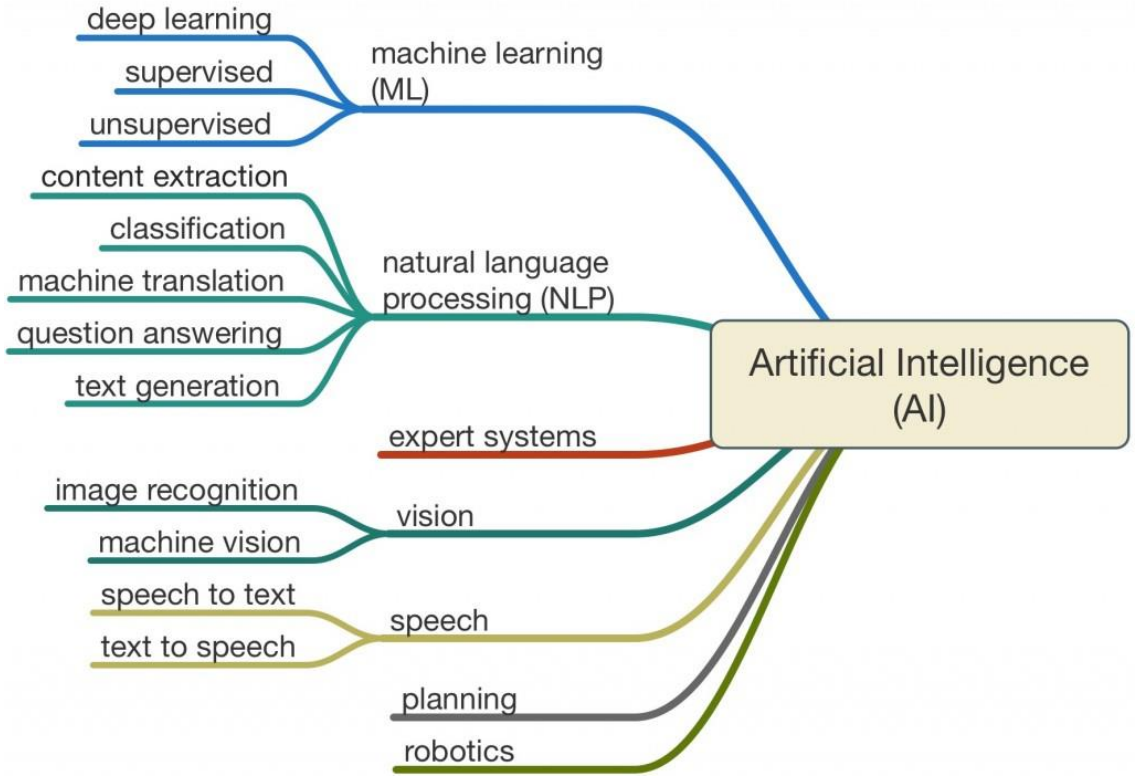
Andrey Markov

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

Organizing our thoughts

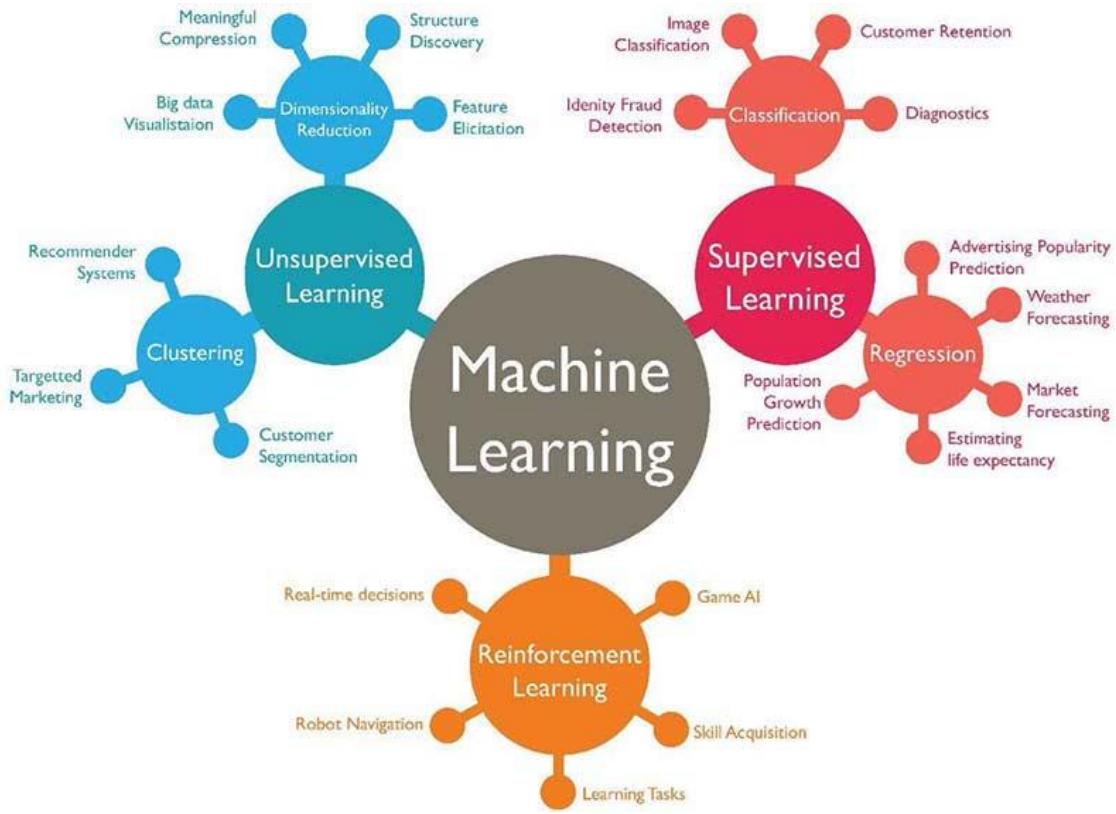


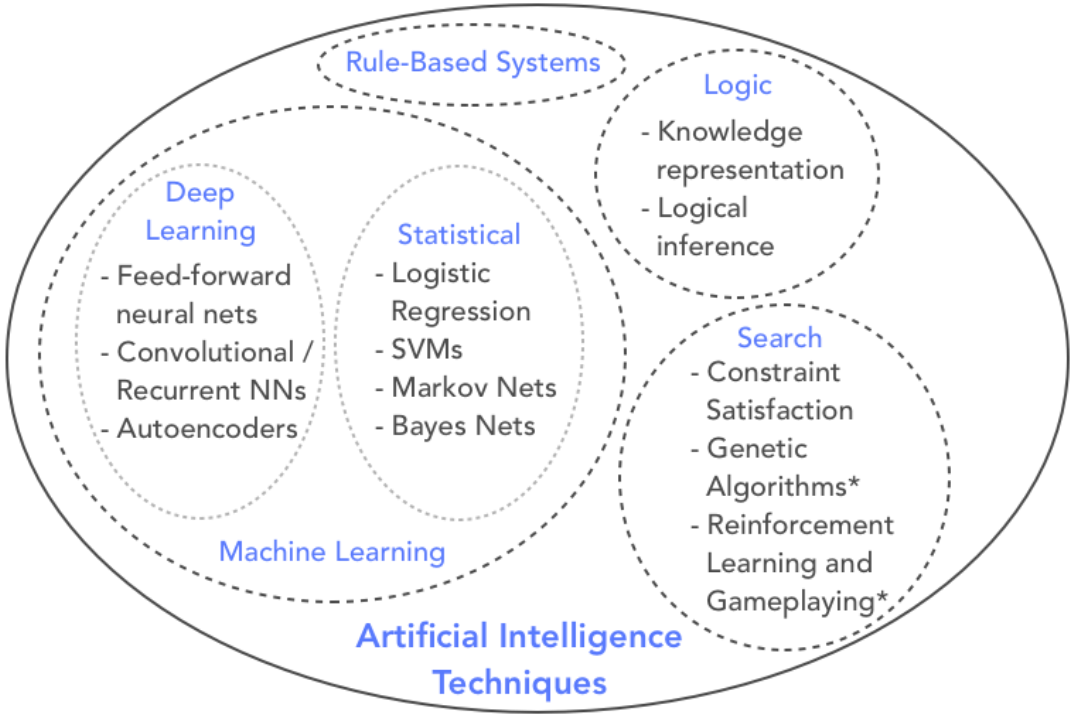


https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning

http://www.legalexecutiveinstitute.com/artificial-intelligence-in-law-the-state-of-play-2016-part-1/

Mind Map / Family Tree



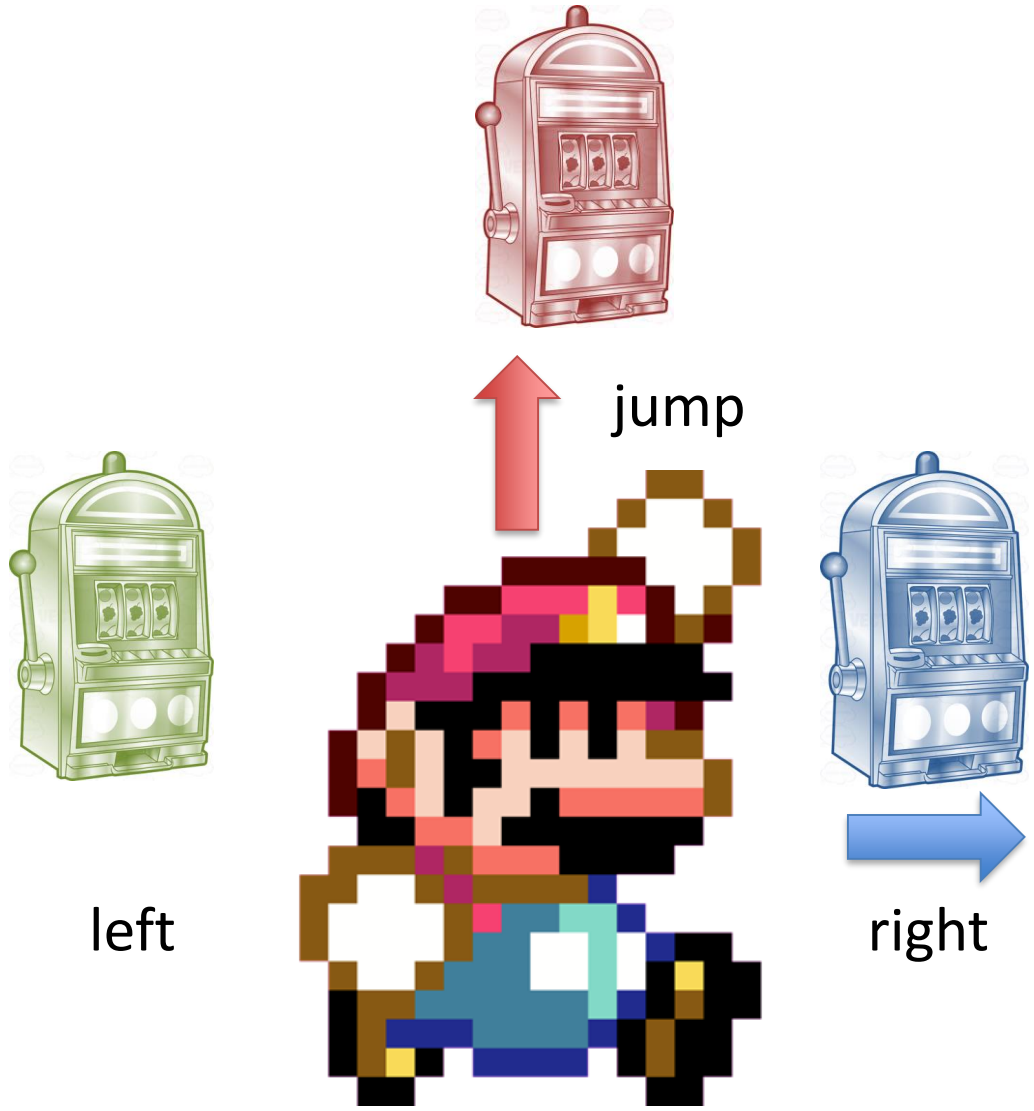


https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/

https://blog.eloquent.ai/2018/08/15/the-ai-landscape/

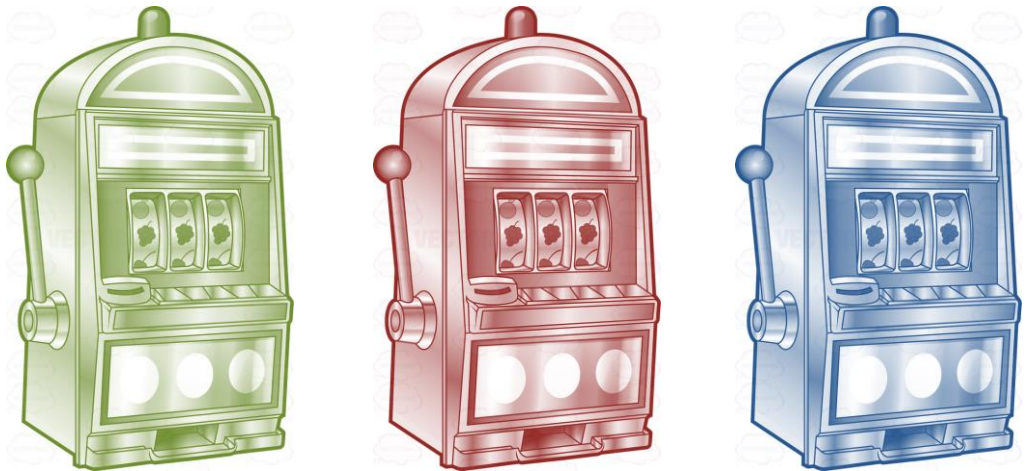
High-level Idea

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)

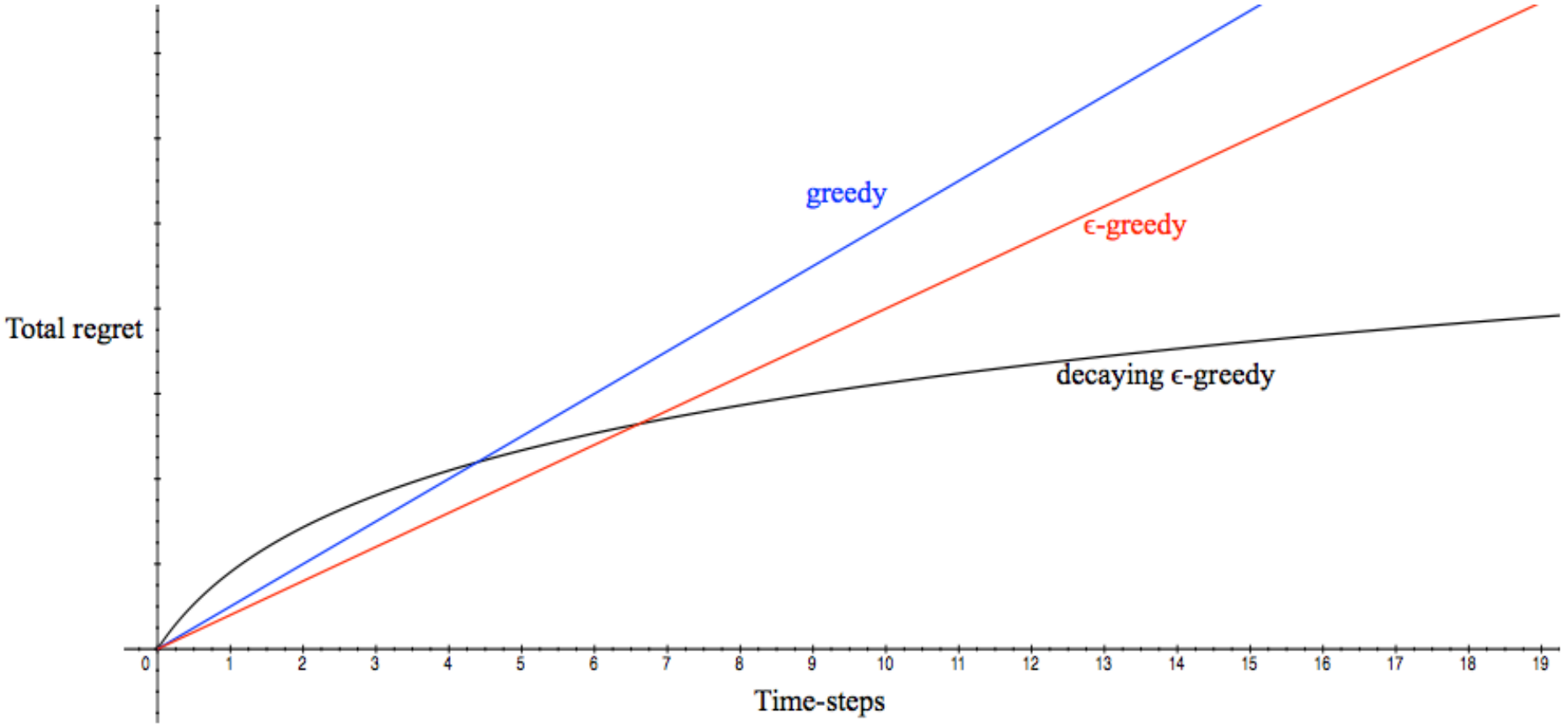
(ϵ=0.1)

- 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!

Different Strategies and Regret



State Representation Examples

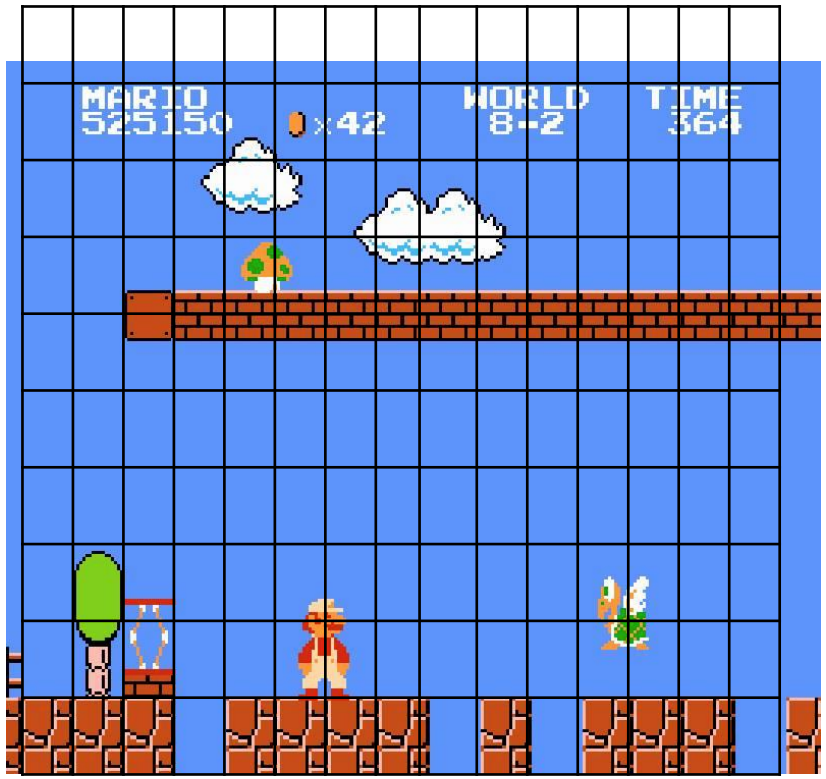
List of facts

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

State Representation Examples

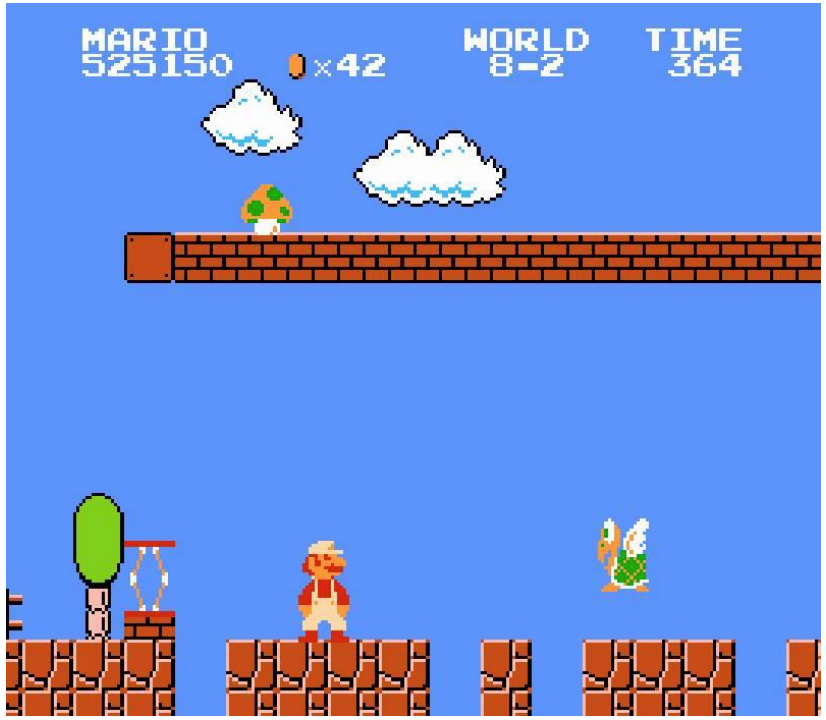
Grid Representations

- 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

Scholarly articles for **sutton and barto reinforcement learning**

Reinforcement learning: An introduction - **Sutton** - Cited by 25899

Reinforcement learning is direct adaptive optimal ... - **Sutton** - Cited by 348

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

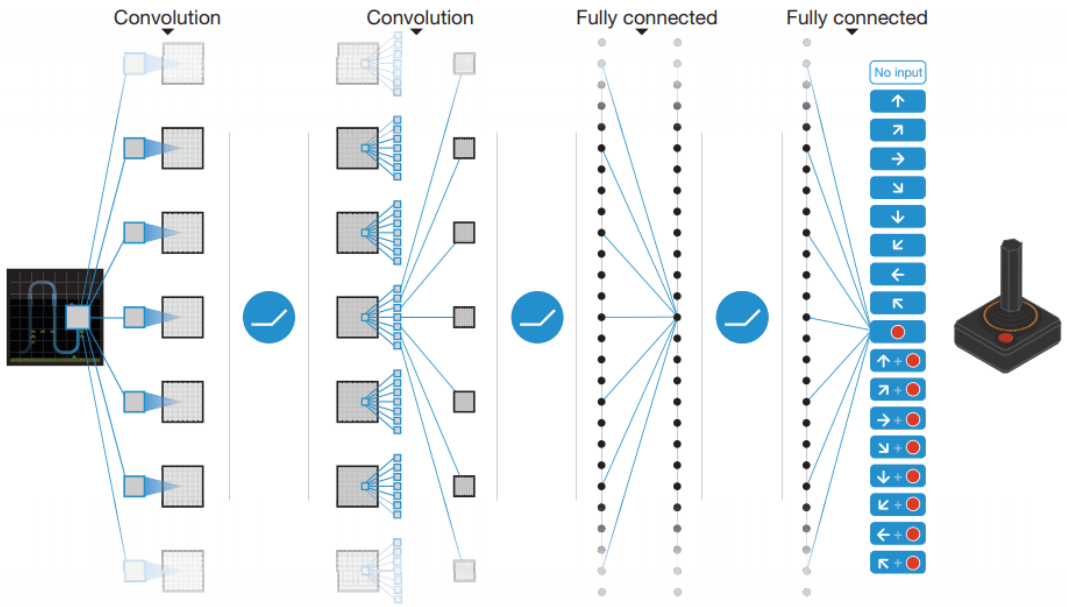
Credit: Sutton & Barto
http://incompleteideas.net/609%20dropbox/slides%20(pdf%20and%20keynote)/1-intro-and-singularity.pdf

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

Components of the AlphaGo

The core parts of the Alpha Go comprise of:

- **Monte Carlo Tree Search:** AI chooses its next move using MCTS
- Residual **CNNs (Convolutional Neural Networks):** AI assesses new positions using these networks
- **Reinforcement learning:** Trains the AI by using the current best agent to play against itself

In this blog, we will **focus on the working of Monte Carlo Tree Search** only. This helps AlphaGo and AlphaGo Zero smartly explore and reach interesting/good states in a finite time period which in turn helps the AI reach human level performance.

It's application extends beyond games. MCTS can theoretically be applied to any domain that can be described in terms of {*state*, *action*} pairs and simulation used to forecast outcomes. Don't worry if this sounds too complex right now, we'll break down all these concepts in this article.

https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/

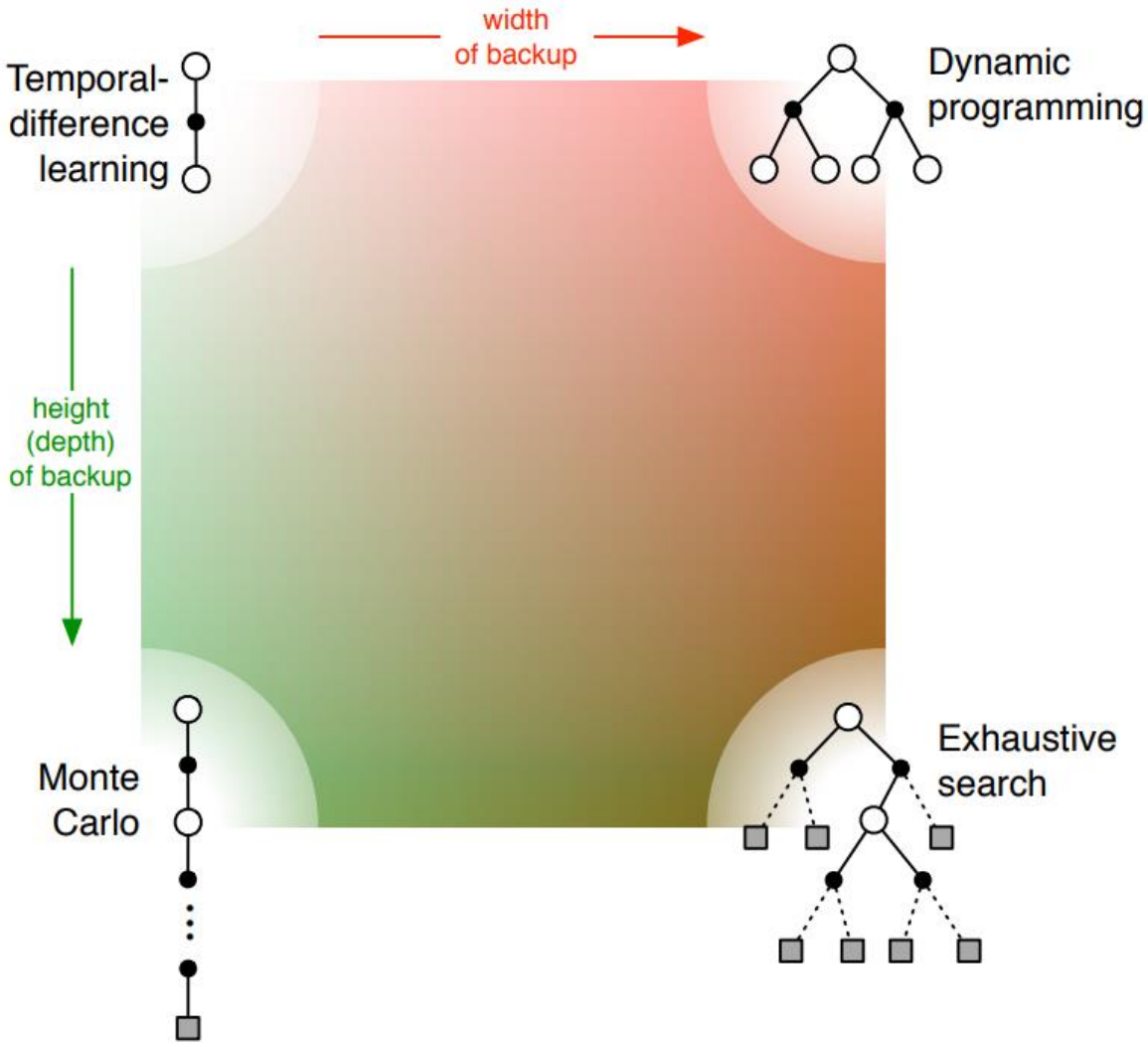
https://www.youtube.com/watch?v=0g9SlVdv1PY

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

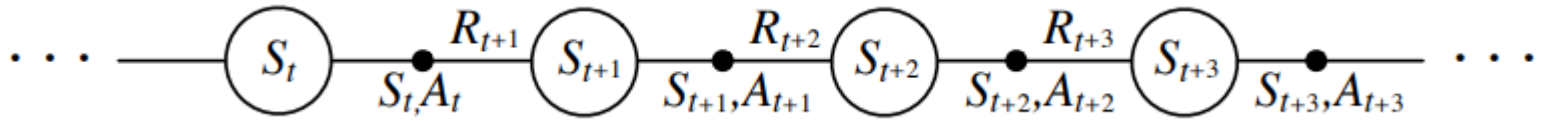
Markovian State

- 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 policy

- Policy 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)

Value Iteration

Value Iteration:

- every state has a value V(s), generally represented as a table
- Known transition probabilities
- Known reward for every transition

Algorithm

- Start with arbitrary values for all V(s)
- Every iteration we update the value of every state according to the rewards we can get from nearby states
- Stop when values are no longer changing

See page 83 in S&B book linked earlier

Value Iteration Update

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_k(s')\right)$$

Value Iteration Update

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V_k(s'))$$

$$V_k(s) = \max_a Q_k(s,a)$$

Value Iteration Update

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma V_k(s'))$$

Translation:

The value of a particular action a in state s =

- the sum across all neighboring states of the
- the probability of going from state s to state s’ given action a multiplied by
- The reward of entering state s’ by action a from state s plus the discount factor γ multiplied by the current value of state s’

Value Iteration Update

$V_k(s) = \max_a Q_k(s,a)$

The new value of state s is then

- the max value within $Q_k(s,a)$

Value Iteration Algorithm

assign V0[S] arbitrarily

$k \leftarrow 0$

repeat

$k \leftarrow k+1$

for each state s **do**:

$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a) (R(s,a,s')+ \gamma V_k(s'))$

$V_k(s) = \max_a Q_k(s,a)$

until $V_k(s) - V_{k+1}(s) < \theta$:

Get Policy From Value Table

Remember we want the optimal action *a* for each state *s*

$\pi[s] = \text{argmax}_a \sum_{s'} P(s'|s,a) (R(s,a,s')+ \gamma V_k[s'])$

Translation:

Take the value of a that gives the max value…

- For each neighboring state and action a
  - Multiply the transition probability from state s to state s’ with action a by
  - The sum of
    - the reward for entering s’ from s with a and
    - The product of the discount value by the Value table for s’

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:

- *Once learning is done running*
- **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…

Seems like we have some extra stuff we don't really need huh?

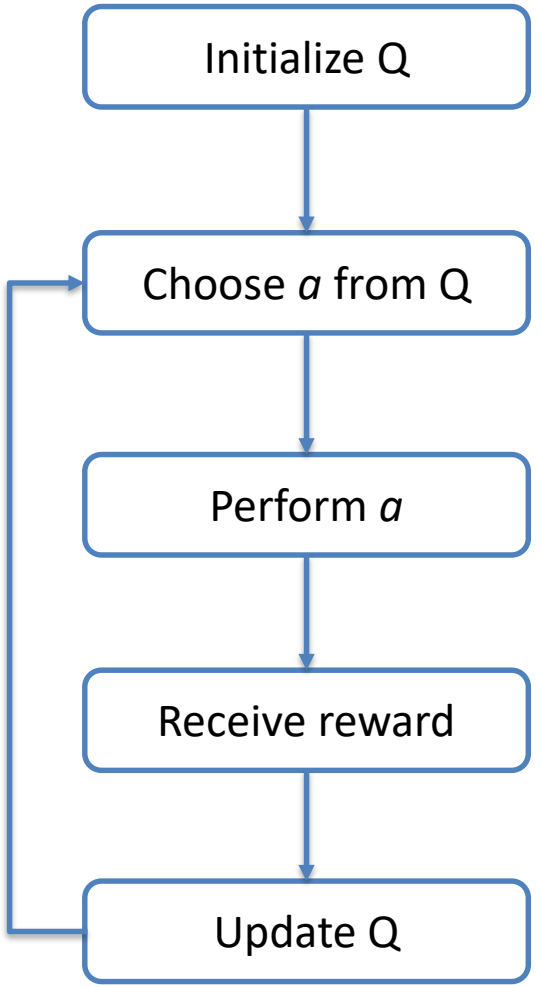
$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\,(R(s,a,s') + \gamma \mathbf{V_k(s')})$$

$$\mathbf{V_k(s) = \max_a Q_k(s,a)}$$

Traditionally, there was a preference for keeping track of V(s) over Q(s,a) as it was smaller

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



State	Action 0	1	2	3	4	5
0	−1	−1	−1	−1	0	−1
1	−1	−1	−1	0	−1	100
2	−1	−1	−1	0	−1	−1
3	−1	0	0	−1	0	−1
4	0	−1	−1	0	−1	100
5	−1	0	−1	−1	0	100

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,s_1)*: Probability that action a takes us from state s to state 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?

Q-table Update

$$Q[s,a] \leftarrow Q[s,a] + \alpha * (r+\gamma * \max_{a'} Q[s',a']-Q[s,a])$$

A Q-table enumerates all possible states and all possible actions and gives the utility of each action in each state

States	Action 1	Action 2	Action 3	Action 4	Action 5
s1	0.1	0.5	0.9	0.4	0.0
s2	0.8	0.2	0.1	0.0	-1.0
s3	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...

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'**

$Q[s,a] \leftarrow Q[s,a] + \alpha * (r+\gamma * \max_{a'} Q[s',a'] - Q[s,a])$

until termination

the max value within Q[s',*]

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

- Takes up more memory than Value Iteration as we have to hold a |S|*|A| table instead of just an |S|-sized table
  - Can be ameliorated with function approximation
- 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**"

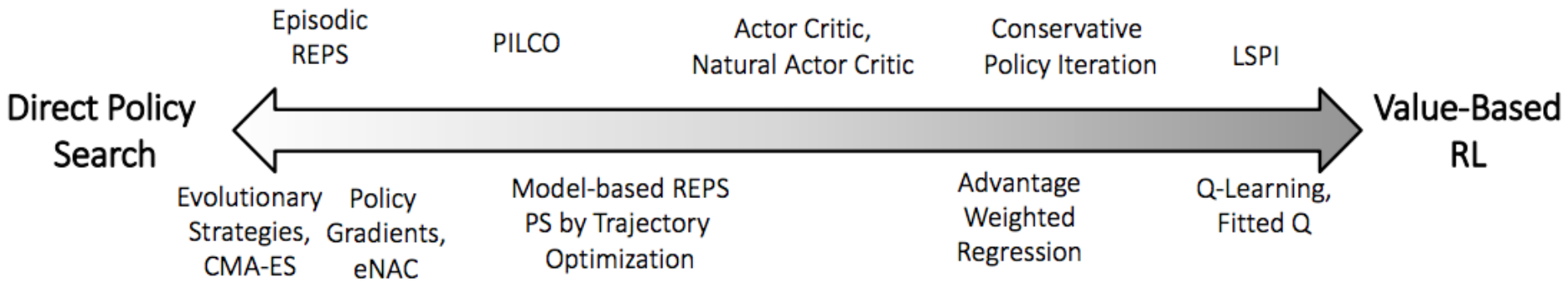
Sutton & Barto, 2nd ed, section 16.5

We can’t always iterate through all spaces

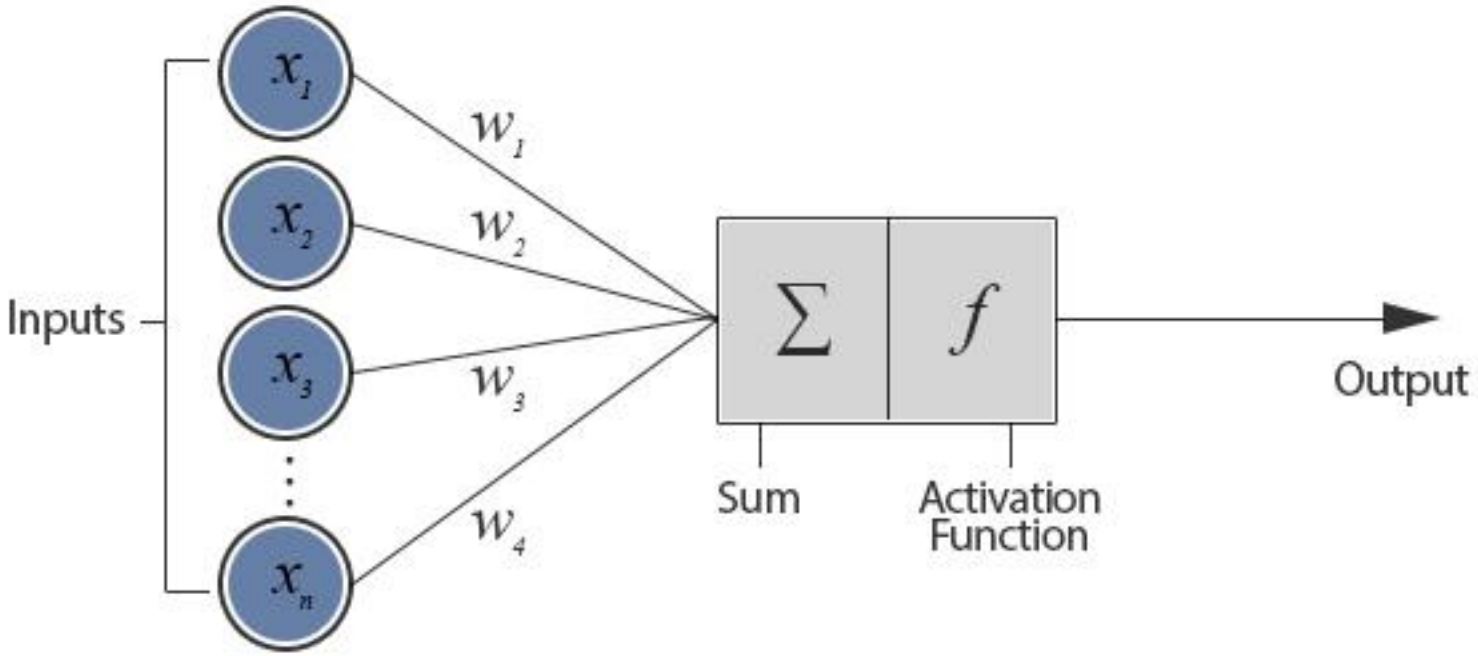
- Start at a possible start state
- Follow this current policy to the end (or almost perfectly follow it)
- Once we hit an end (or horizon) return all the way back up to the start

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?

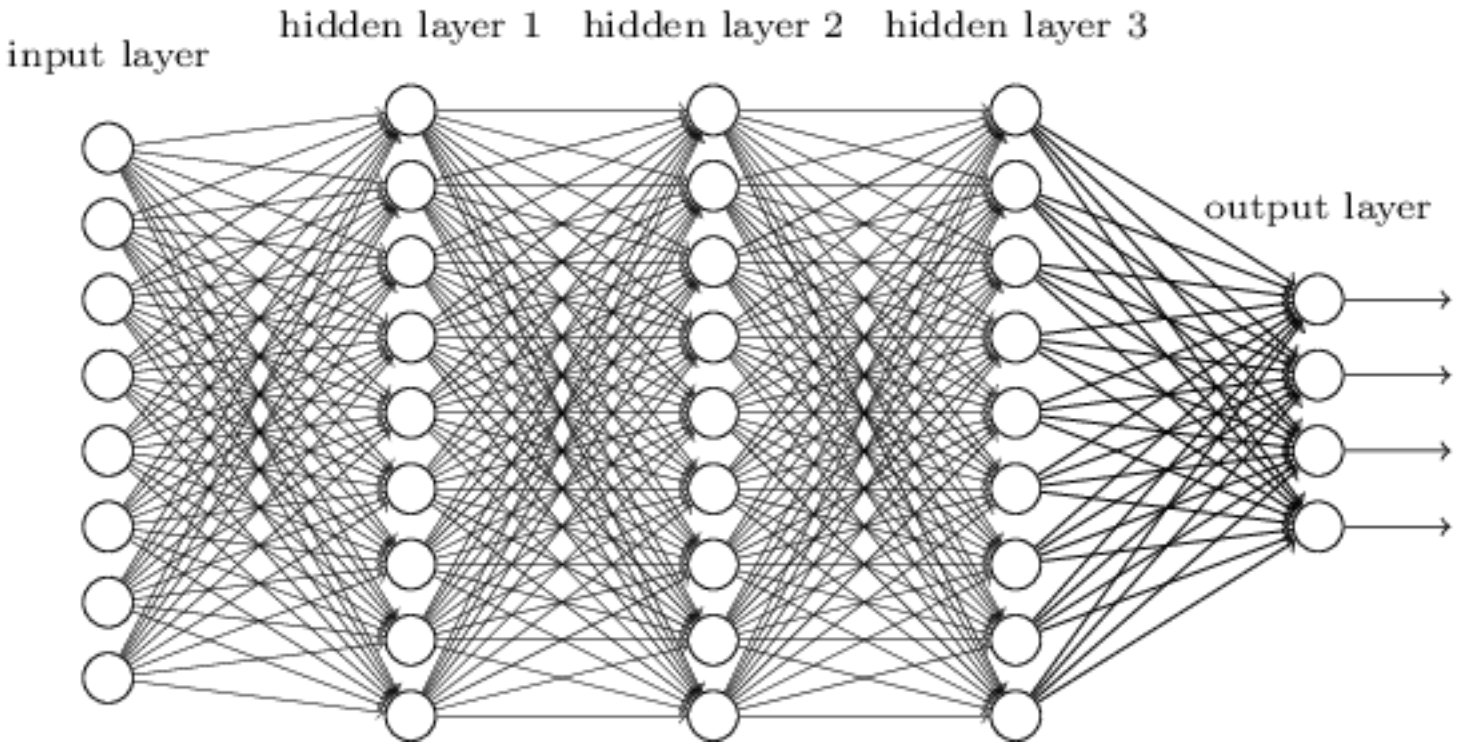


Artificial Neural Nets

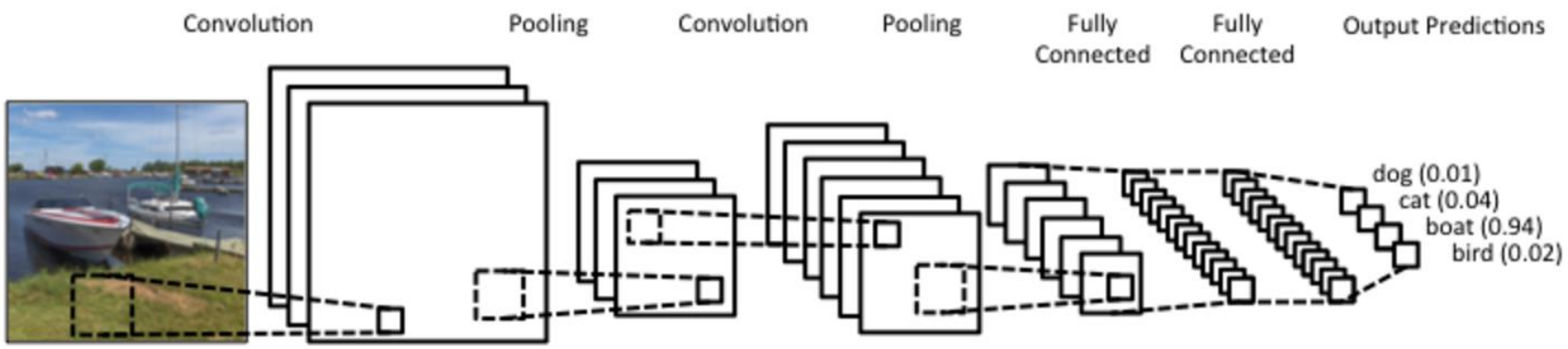


Warren McCulloch and Walter Pitts (1943)

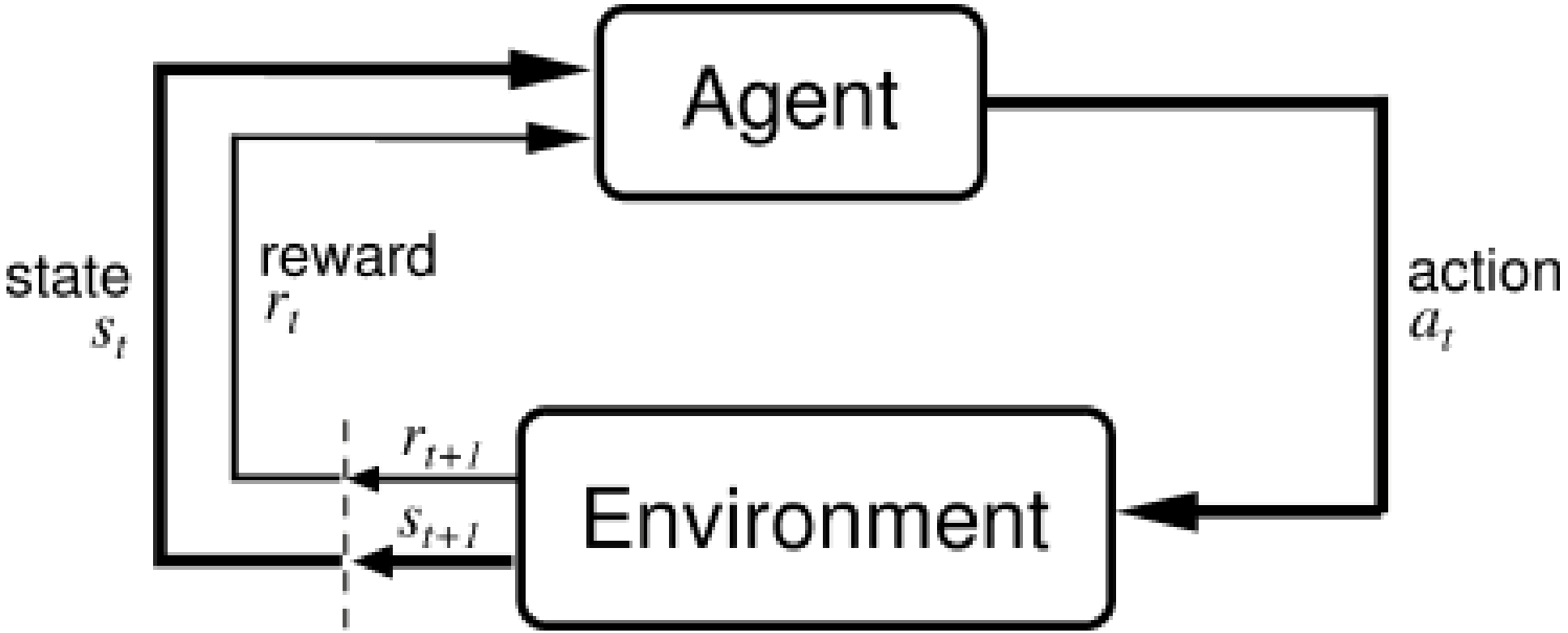
Deep Neural Networks



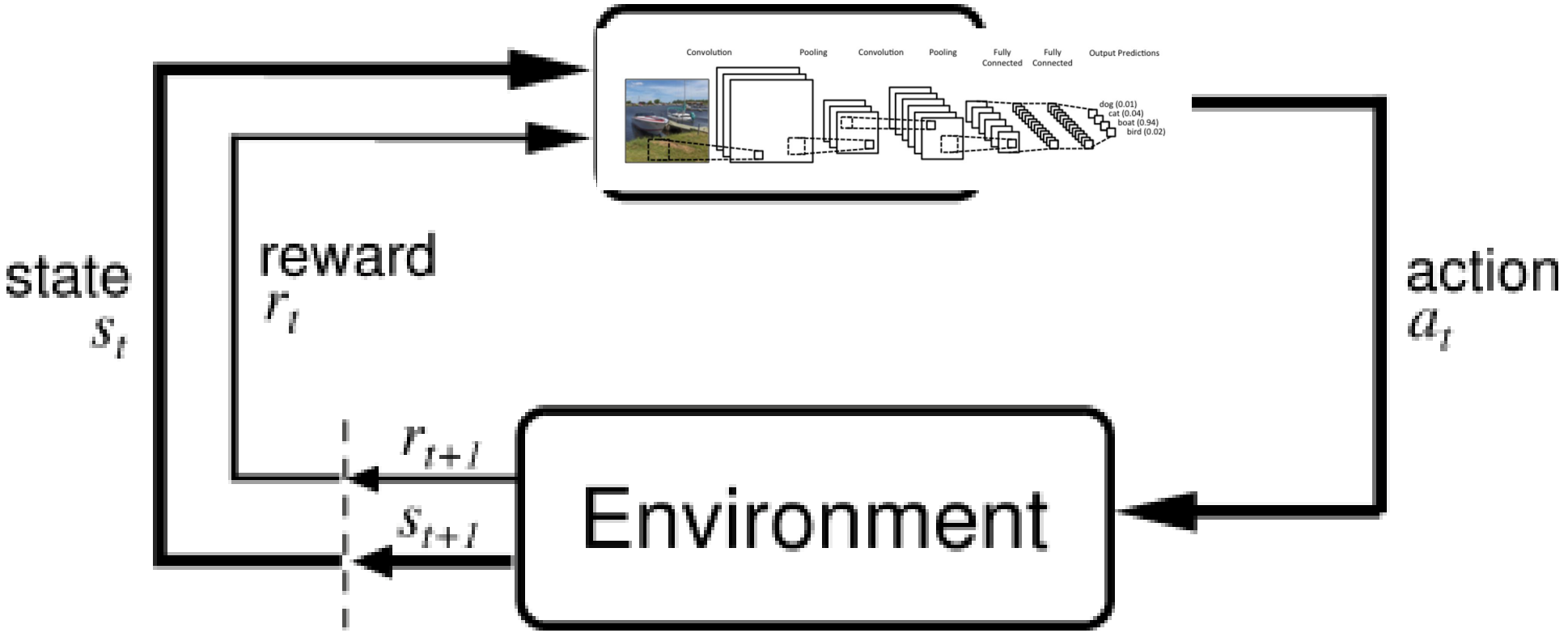
Convolutional Neural Networks



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

Learning via self play

- DeepMind
  - https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
  - https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning
  - https://youtu.be/V1eYniJ0Rnk?t=25
- https://www.youtube.com/watch?v=oo0TraGu6QY
- Google's self-learning AI AlphaZero masters chess in 4 hours
  - https://www.youtube.com/watch?v=0g9SlVdv1PY
- OpenAI
  - https://www.theverge.com/2018/8/28/17787610/openai-dota-2-bots-ai-lost-international-reinforcement-learning
  - https://venturebeat.com/2019/04/22/openais-dota-2-bot-defeated-99-4-of-players-in-public-matches/
  - OpenAI Five Beats World Champion DOTA2 Team 2-0

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)