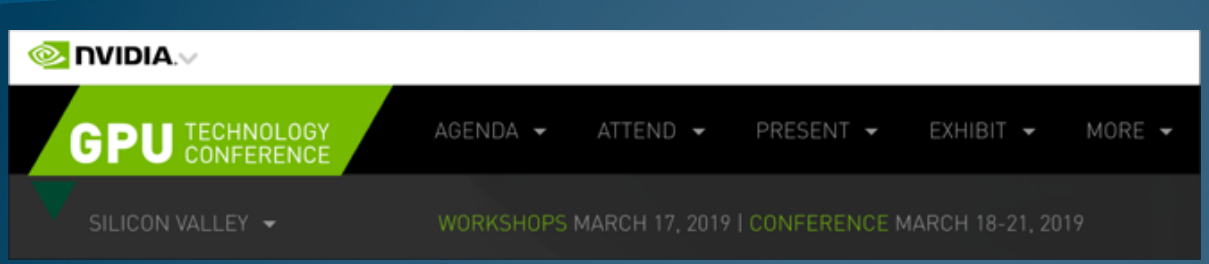
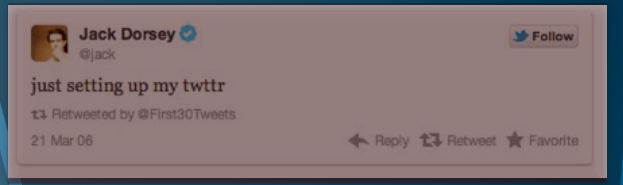


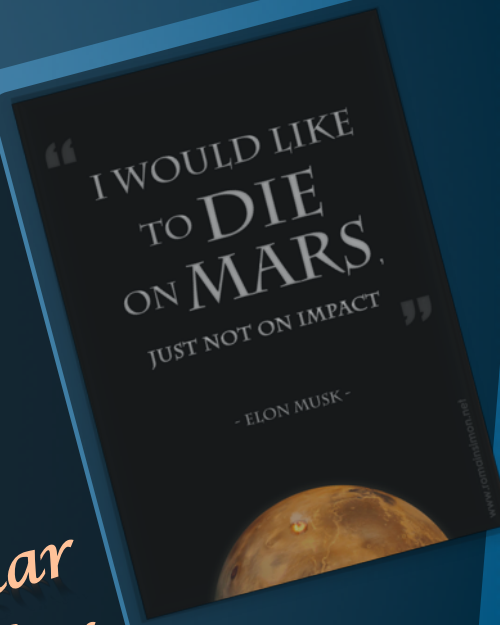


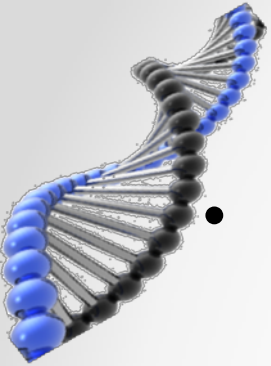
"A good traveler has no fixed plans and is not intent on arriving." – Lao Tzu

Demystifying Deep Reinforcement Learning – Part 1 : DQN

All observations are my own & do not represent my employer's or anyone else's

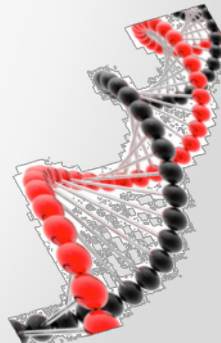
krishna sankar
@ksankar

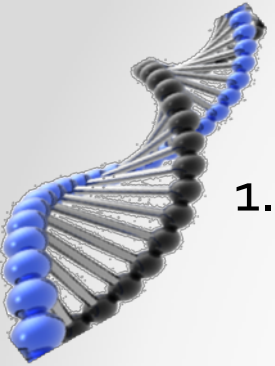




GTC 2019 SV : Lab Overview

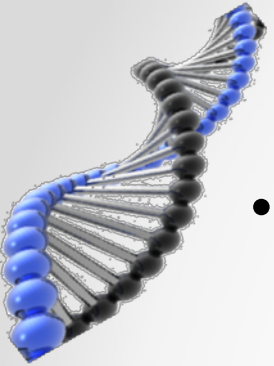
- Deep Reinforcement Learning
  - Principles, Paradigms & Equations in 2 Parts
  - https://gputechconf2019.smarteventscloud.com/connect/sessionDetail.ww?SESSION_ID=279058
  - https://gputechconf2019.smarteventscloud.com/connect/sessionDetail.ww?SESSION_ID=279064
- Part 1 : Start from the Fundamentals
  - Program Value Methods
  - Monte Carlo, TD & Q-learning
  - DQN and it's extensions
- Part 2 : Policy Gradients
  - REINFORCE
  - A3C & DDPG
  - Intro to PPO, TRPO & the rest





DRL Lab Goals

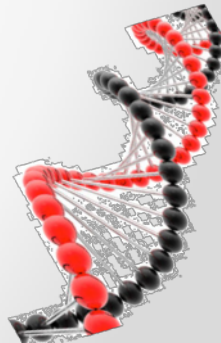
1. Get a reasonably good introduction to DRL (as much as we can do in 4 hrs)
   - Sometimes the ideas take time to rattle around the brain and sink in; they need reflection; hence we stop Part 1 at DQN and later start Policy Gradients as Part 2
2. Alleviate the Initial Barrier
   - Discuss intuition and formal notations
   - Understand the underpinnings
   - Do hands-on programming and finally,
   - *Motivate to learn, explore on your own & be proficient in Deep Reinforcement Learning*
3. We will use Autonomous Driving & AlphaGo as background examples
4. Hands on programming
   - Gridworld = Frozen Lake
   - Continuous = Cartpole
   - We will run these 2 examples throughout the lab for all the algorithms
     - With limited time, we don't want to add new examples every time

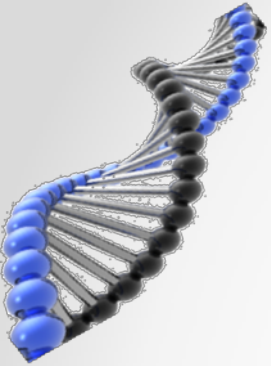


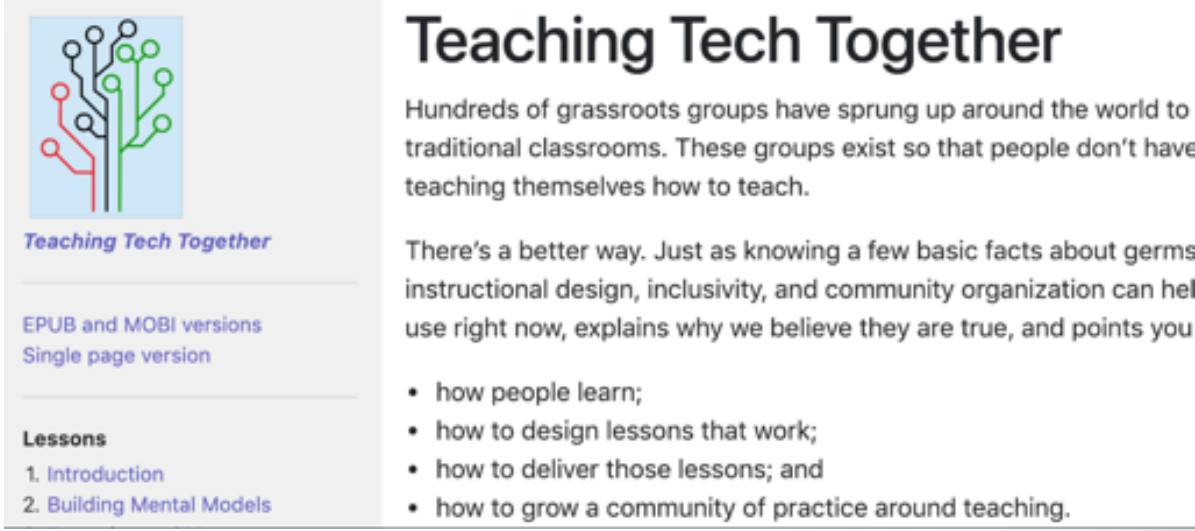
Please Do …

- The Programming Exercises & Quizzes
  - This is a good time to do these - you have a captivated time away from the daily trifles
- Write the formulas, at least 3 times !
  - Why write down formula ? Levine UCB classes ! Kuru Toga icon !

- Read the essential papers
  - Good comprehensive intro:
    - https://arxiv.org/pdf/1810.06339.pdf
    - An Introduction to Deep Reinforcement Learning https://arxiv.org/pdf/1811.12560.pdf
  - Sutton Book http://incompleteideas.net/book/the-book-2nd.html
  - AlphaGo nature papers, movies, AlphaZero et al https://deepmind.com/research/alphago/
  - Rainbow Paper https://arxiv.org/pdf/1710.02298.pdf
  - Minh Paper : https://arxiv.org/abs/1602.01783
  - DDPG Paper : https://arxiv.org/pdf/1509.02971.pdf

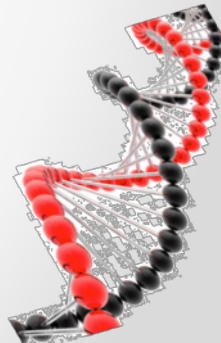


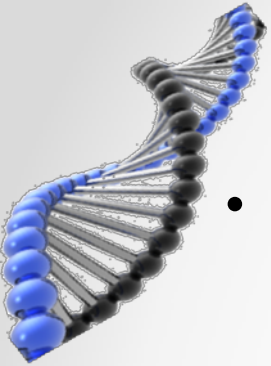




The Rules for the Teacher

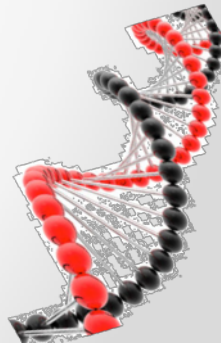
1. Remember that you are not your learners…
2. Never hesitate to sacrifice truth for clarity 😇
3. Remember that …
   - … no lesson survives first contact with learners…
   - … that every lesson is too short from the teacher's point of view and too long from the learner's…
   - … and that nobody will be more excited about the lesson than you are.

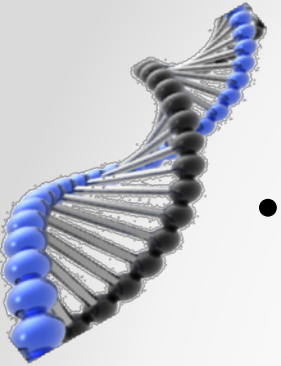




You have …

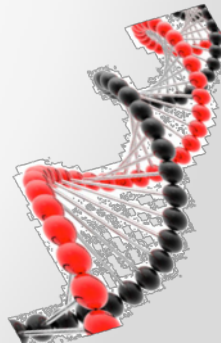
- Programming experience
  - Know about DeepLearning,
  - Experience using Tensorflow or pytorch,
  - Python programming and familiar with Jupyter/iPython notebook
- No Reinforcement Learning experience is expected or assumed
- Will cover enough of Deep Learning & Pytorch
- Why Pytorch ?
  - Easier & compact, those of you from the world of Tensorflow, this will be a new experience
  - DL is only a small part and are using simpler primitives
- Have installed the prerequisites
- Have downloaded the materials from github
  - https://github.com/xsankar/GTC_2019_DRL_LAB
- The notebooks in (somewhat) working order

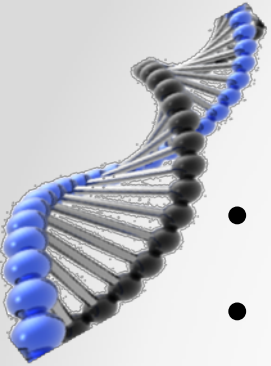




Lab notebooks & presentation

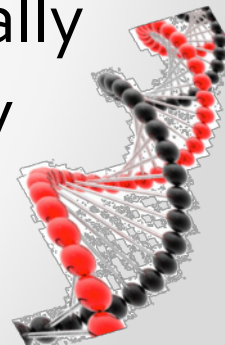
- Labs have working examples *good enough*
  - Look at them as starting frameworks - for you to tweak & improve
  - Use as a study tool
  - Share your comments, clarifications, functions and algorithms as pull requests
  - And when you have mastered an algorithm or an environment, add your successes to the OpenAI Gym leaderboard
    - Consider leaderboard as a level of performance
- Lab pragmatics
  - In github : https://github.com/xsankar/GTC_2019_DRL_LAB
  - No esoteric requirements
  - Requirements
    - python 3.6, pytorch, openAI gym, numpy, matplotlib, jupyter notebook
  - Miniconda works fine



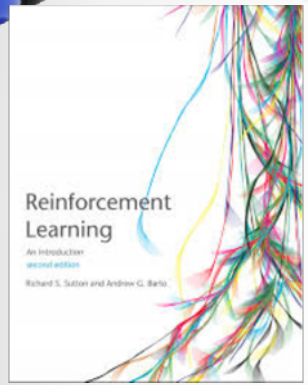


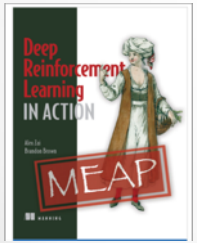
Pragmas - Deep RL Experimentation

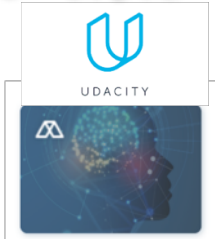
- Be ready for (a lot of) trial-and-error
- There will be many false starts
- Hyper-parameter tuning is important
- Have faith; the network will learn - eventually !!
- Learning is not linear
  - Even after 90% is done, you still might have 90% to go
- Keep a log and try things (somewhat) systemically
- Too many (simultaneous) moving parts will only create chaos



Thanks to the Giants, whose work helped me to prepare this lab !!





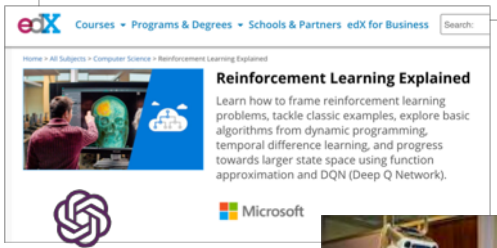


School of Artificial Intelligence

Deep Reinforcement Learning

In Collaboration With

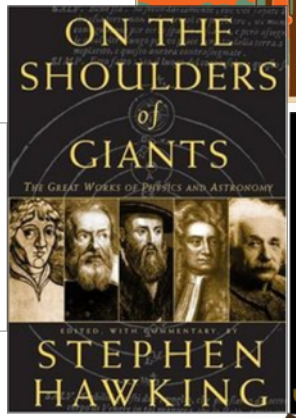
Unity, NVIDIA Deep Learning Institute

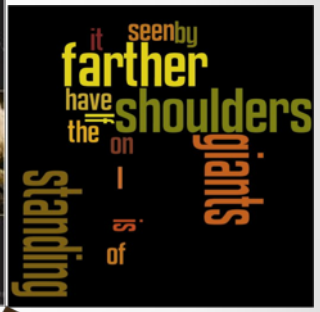


OpenAI

CS 294-112 at UC Berkeley

Deep Reinforcement Learning



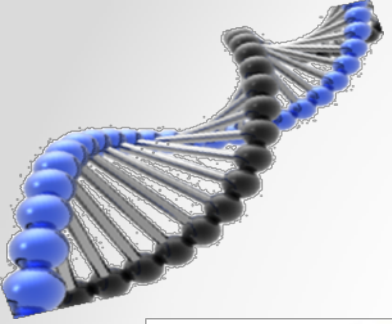


Abbeel & Levine

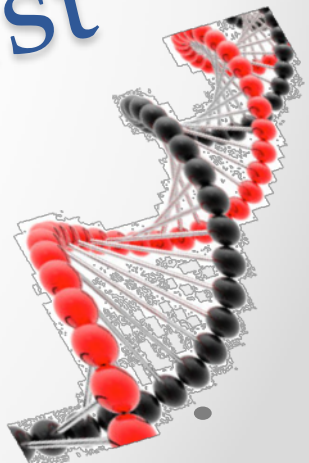
Agenda

- Framing
- Reinforcement learning – A Formal Expedition
- Algorithms & Programming
  - Monte Carlo, TD, SARSA, Q-Learning & DQN
- The world of AlphaGo
- Policy Gradients – REINFORCE, A3C & DDPG
- Conclusion & Forward
- Emergency Time Reserve – Overflow !!

https://github.com/xsankar/GTC_2019_DRL_LAB

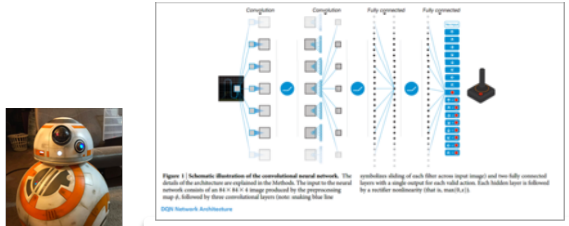


1. Framing : Open This First



What exactly is this Reinforcement Learning ?

RL = Formal framework
✓ Computationally efficient approach



Agent – Learn, Decide & Act
✓ Nothing less, nothing more !
✓ Interacts with the environment
✓ Collects Information

Agent

Goal : Maximize rewards

3

1

State

Reward

Action

2

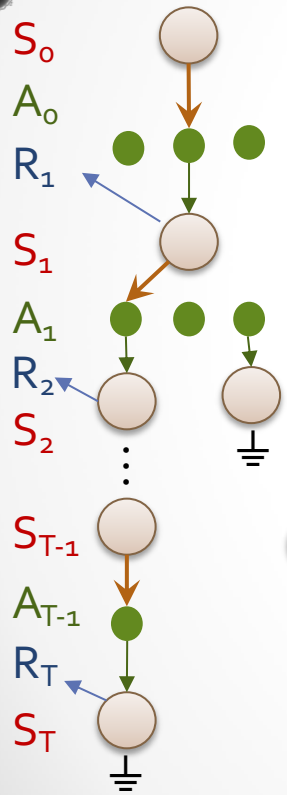
Environment

Acquires new behaviors & skills
incrementally, using trial-and-error experiences
- a priori batch is OK, in some cases

Uncertain; fully observable at best

RL Sequence

- Agent starts at State S_o , chooses & takes Action A_o,
- The environment transitions to S_1 & emits a reward R_1
- The environment also sends and observation of S_1
- Now the agent is at S_1 and can take action A_1
- The cycle continues, till the end is reached
- *Can be in 1 step, n steps or even continuous (=infinite horizon)*

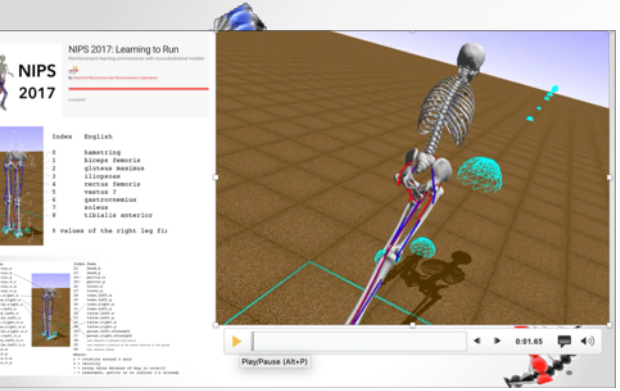


= State

= Action

= Terminal State

DRL in Action (?) – Video Time !!



1. Walking Humanoid figures - UC Berkeley

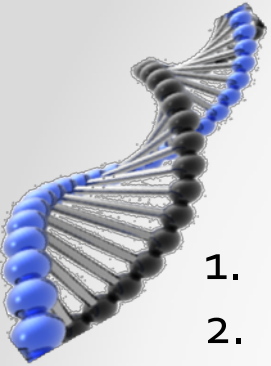
2. Humanoid avoiding obstacles - UCB

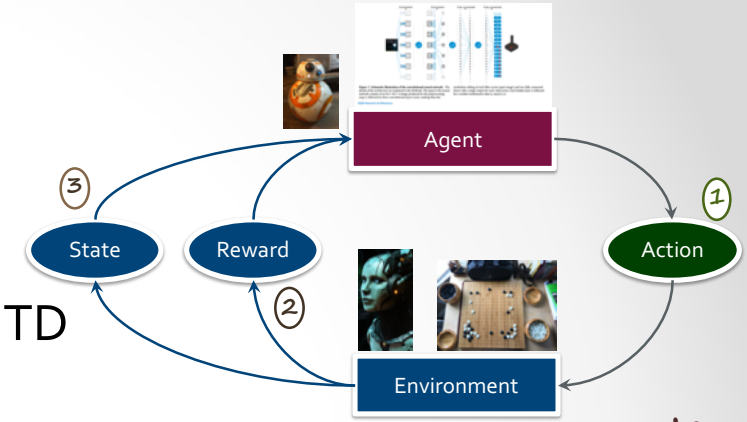
3. Navigating Complex Roundabouts

Goal of the videos : Motivation for DRL

- *While, in this lab, we look at relatively smaller problems, the algorithms are very powerful and are solving much larger problems*
- *As you go through the lab, think of how these problems can be solved using different Reinforcement Learning mechanisms*

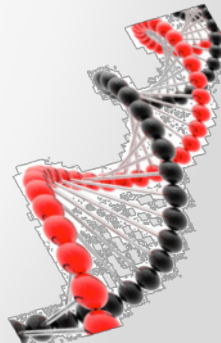
DRL Challenges

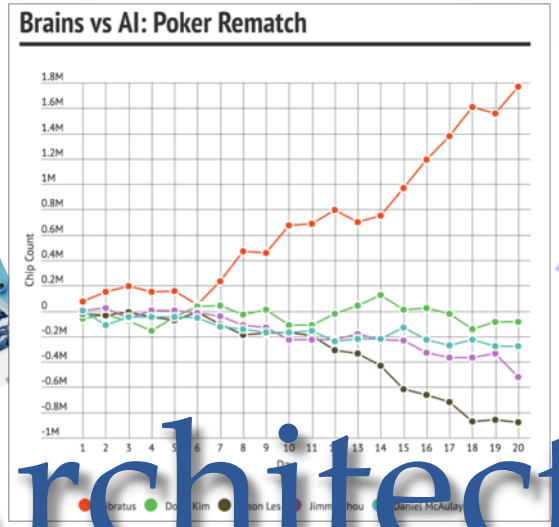


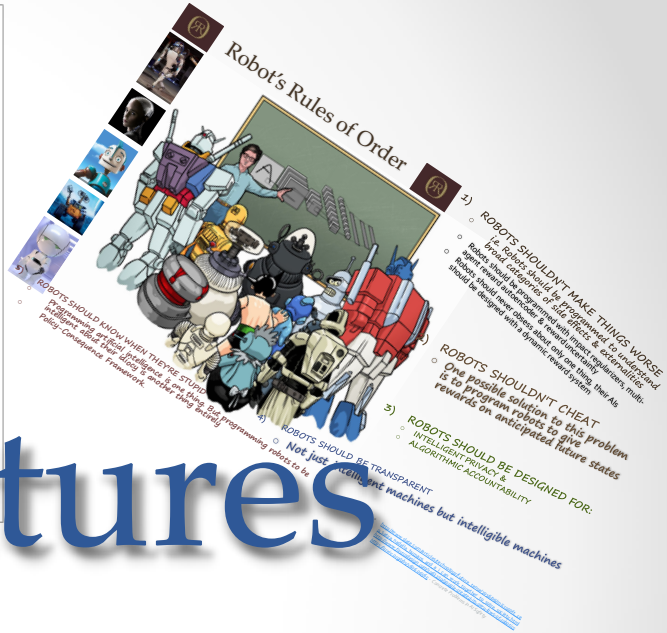


1. Exploration & Convergence – Bandit
2. Reward Shaping & Credit Assignment – TD
3. Representation
   - DQN, CNN (LSTM,RNN Not yet) Capsule Networks/Transformer Architectures (?)
   - Lower dimensional than tabular representation
   - Compact as possible, but can capture the full complexity of the underlying state space
   - Sample efficiency - replay buffer w/ DQN
   - Atari representation
4. Generalization - DQN
   - Atari generalization - same architecture, but trained for each game
5. Context & Content - Knowledge Graph
   - Encode expert knowledge
   - Atari - pixels - no knowledge
   - If it's Tuesday, this must be Belgium - you need Tuesday Belgium 1000 times !
6. Common Sense Reasoning - Logic Representation & Traversal
7. Conversation Chain - Jarvis/Jeeves

If you are planning to do PhD (I urge all to earn a doctorate degree in AI) there are fertile research topics here !!!







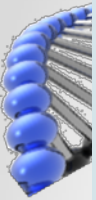
AI Architectures

. . .

Quick Context

- What is AI in our context ?
- A Robot's Rules of Order
- AI Frameworks
- …

- AI is :
  - Contextual & Autonomous i.e. sense the environment, propagate current state across goals, knowledge & actions and respond appropriately,
  - Rational i.e. The ability to infer/perceive value/payoff of a set of actions in the current context/ beliefs, and plan accordingly
  - Optimal i.e. The capacity to choose one or more actions over the space of behaviors, based upon a set of policies including short & long term optimization of cost functions & goals (bounded rationality as the system can only see a limited set)
  - Remembering i.e. the capability to hold episodic and long term beliefs, which adds representational formalisms and retrieval mechanisms
  - Reflective, Adaptive & Self-evolving i.e. the ability to learn from past actions & incorporate the lessons in future behavior
  - Interactive i.e. the capability to interact with humans with different roles (e.g. passengers, other drivers, pedestrians et al) via multiple interfaces including NLP & NLU
- We also add the constraints of reactivity, redundancy, efficiency & scalability for the infrastructure & the framework stack where the AI system is deployed



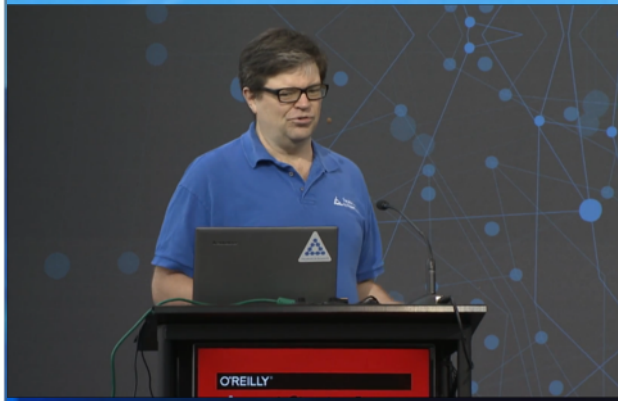

Obstacles to Progress in AI

Y LeCun

- Machines need to learn/understand how the world works
  - Physical world, digital world, people,....
  - They need to acquire some level of common sense
- They need to learn a very large amount of background knowledge
  - Through observation and action
- Machines need to perceive the state of the world
  - So as to make accurate predictions and planning
- Machines need to update and remember estimates of the state of the world
  - Paying attention to important events. Remember relevant events
- Machines neet to reason and plan
  - Predict which sequence of actions will lead to a desired state of the world

- Intelligence & Common Sense =

  Perception + Predictive Model + Memory + Reasoning & Planning



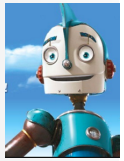
Yann LeCun Retweeted

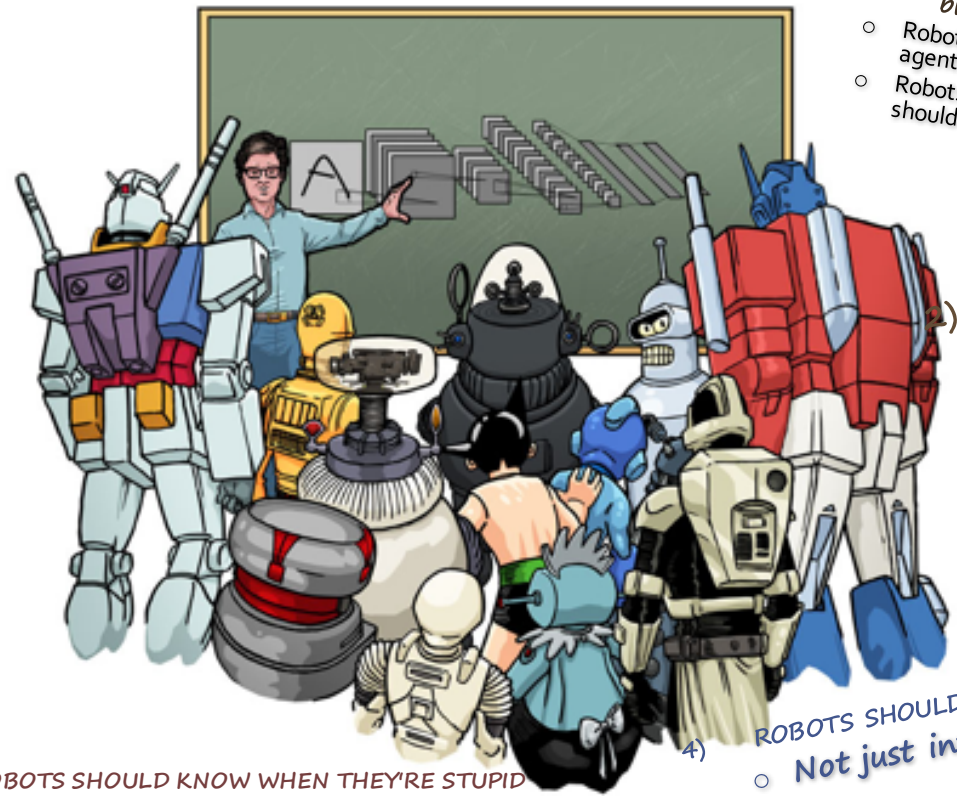
Andrew McAfee @amcafee · 17h

.@ylecun Teaching machines common sense is a "Big mountain. And we don't know how many mountains are behind it." #WHFrontiers

Robot's Rules of Order







1) ROBOTS SHOULDN'T MAKE THINGS WORSE
   - i.e. Robots should be programmed to understand broad categories of side effects & externalities
   - Robots should be programmed with impact regularizers, multi-agent reward autoencoder & reward uncertainty
   - Robots should never obsess about only one thing, their AIs should be designed with a dynamic reward system

2) ROBOTS SHOULDN'T CHEAT
   - One possible solution to this problem is to program robots to give rewards on anticipated future states

3) ROBOTS SHOULD BE DESIGNED FOR:
   - INTELLIGENT PRIVACY &
   - ALGORITHMIC ACCOUNTABILITY

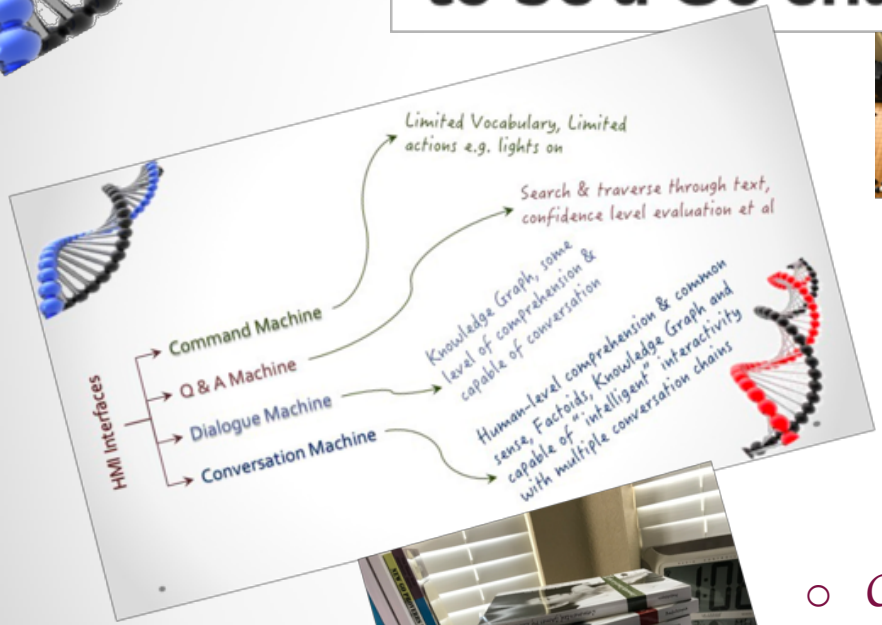
4) ROBOTS SHOULD BE TRANSPARENT
   - Not just intelligent machines but intelligible machines

5) ROBOTS SHOULD KNOW WHEN THEY'RE STUPID
   - Programming artificial intelligence is one thing. But programming robots to be intelligent about their idiocy is another thing entirely
   - Policy-Consequence Framework.

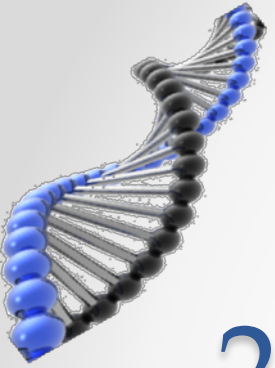
- http://www.slate.com/articles/technology/future_tense/2016/06/microsoft_ceo_satya_nadella_humans_and_a_i_can_work_together_to_solve_society_html
- http://www.fastcodesign.com/3061230/google-created-its-own-[illegible]-robots
- https://arxiv.org/abs/1606.06565 - Concrete Problems in AI Safety

Ask not if AlphaZero can beat humans in Go—Ask if AlphaZero can teach humans to be a Go champion !!!

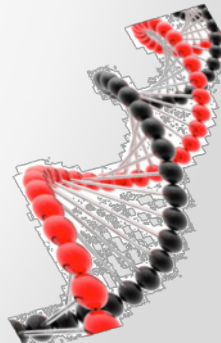
https://goo.gl/vPzN9B



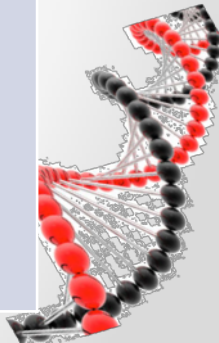
- *Can a program understand playstyles (e.g. Lee Sedol) & imitate ?*
- *Can it adapt to it's student's level ?*



2. Reinforcement Learning
A Formal Expedition



	Supervised Machine Learning	Unsupervised Machine Learning	Reinforcement Learning
Data	Curated	Curated	Generates data while exploring
Labels	Curated	Machine Generated	Learn via interactions
Signals & Feedback			Curated Reward Signals and Goals
Actions			Actions instead of labels
Defining Characteristics	Classification, Linear Regression	Human Meaning e.g. clusters, survival analysis	End result = Decision making - winning a game, driving cars, manufacturing robots, cleaning robots, conversations, …

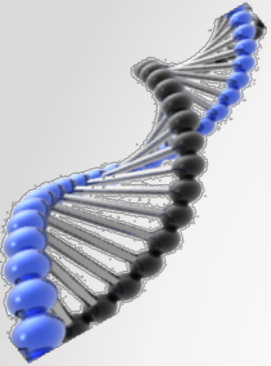


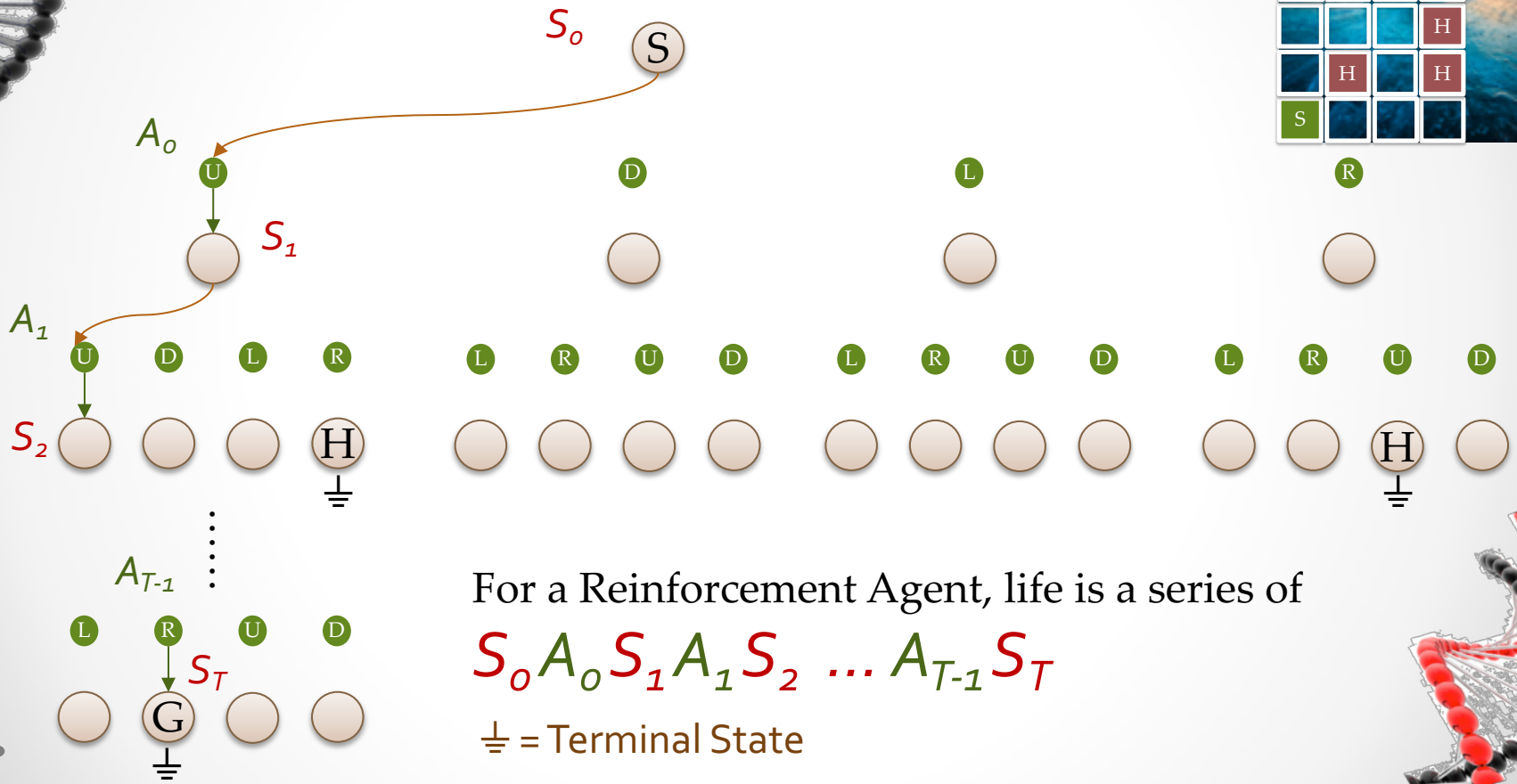
Frozen Lake !

- You are in Siberia and want to visit the foothill under the rock, which is the Goal (G), starting at S
- But the lake is a little treacherous with holes(H)

https://psmag.com/magazine/a-frozen-lake-in-siberia

Notation : State, Action



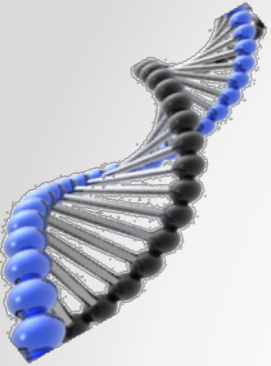


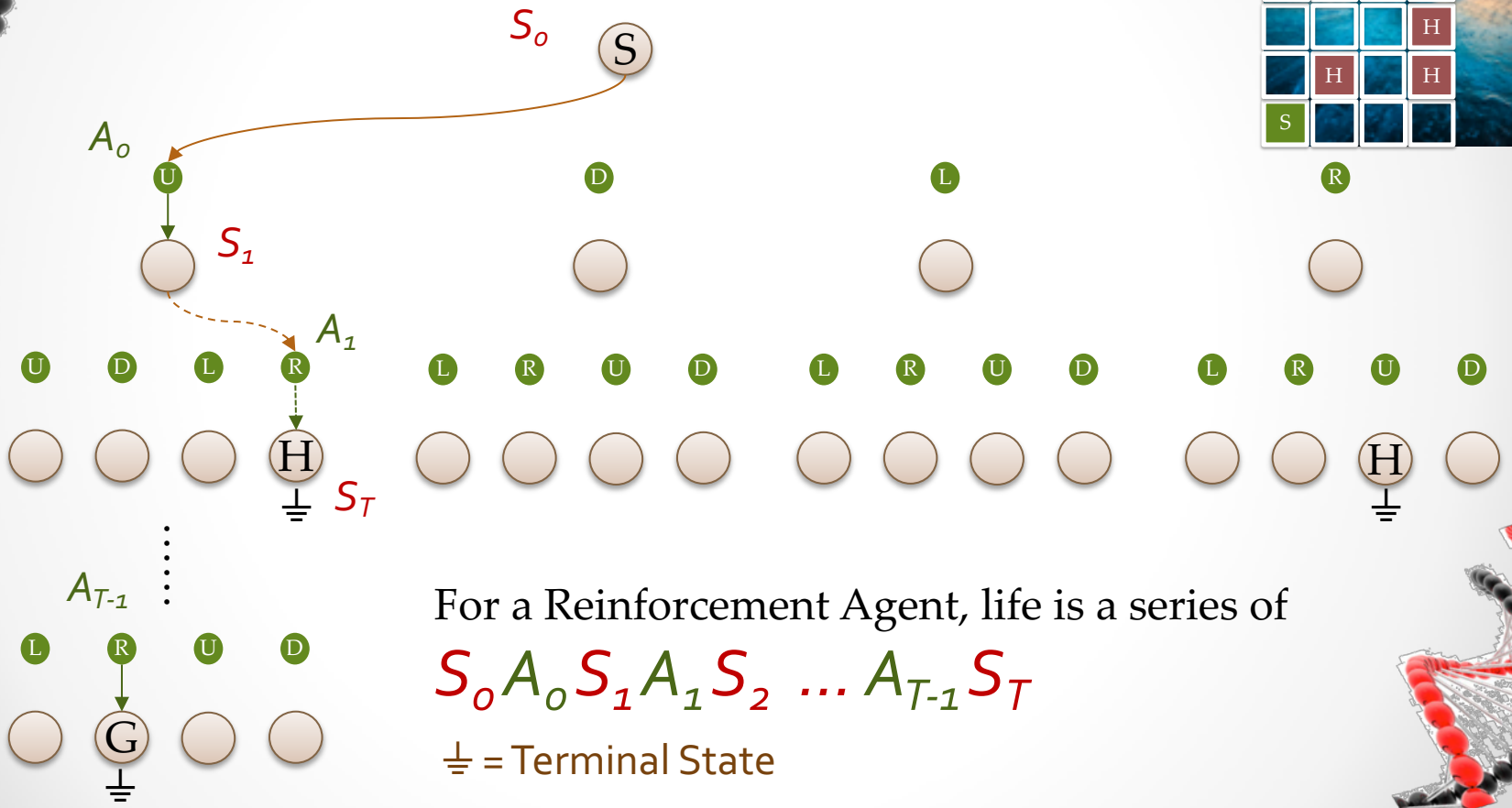
For a Reinforcement Agent, life is a series of

$S_0 A_0 S_1 A_1 S_2 \ldots A_{T-1} S_T$

⏚ = Terminal State

Notation : State, Action



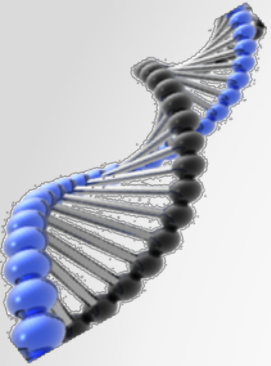


For a Reinforcement Agent, life is a series of

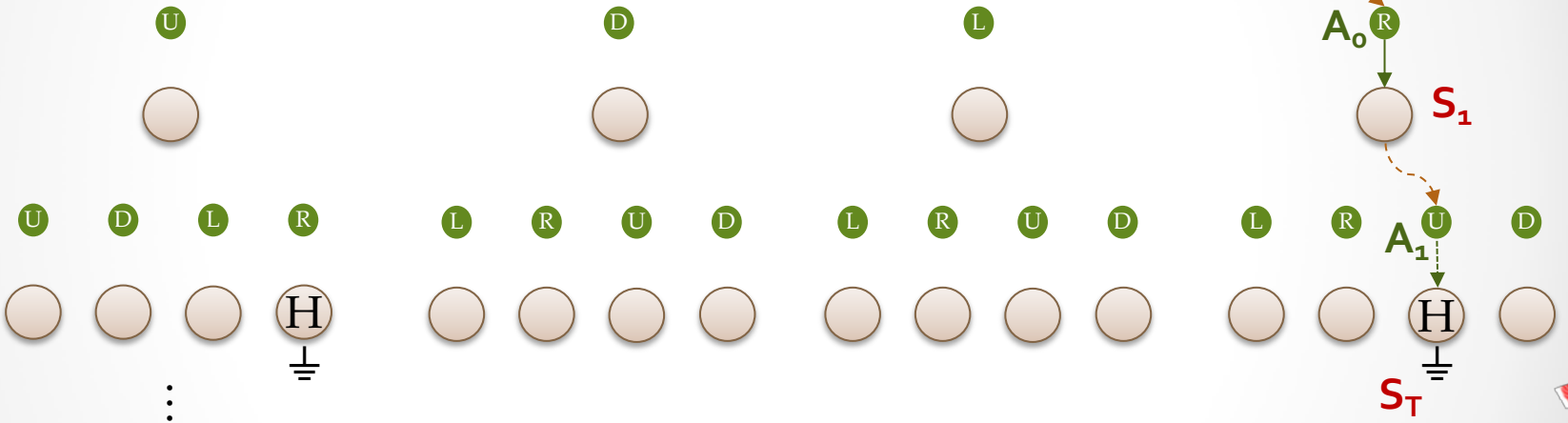
$$S_0 A_0 S_1 A_1 S_2 \ \ldots \ A_{T-1} S_T$$

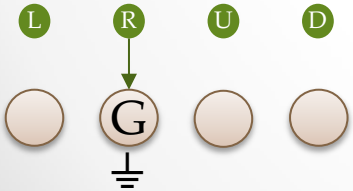
⏚ = Terminal State

Notation : State, Action



S_0

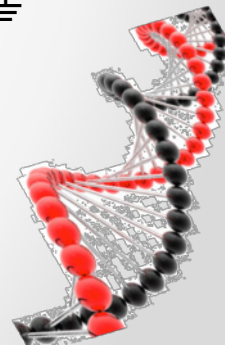




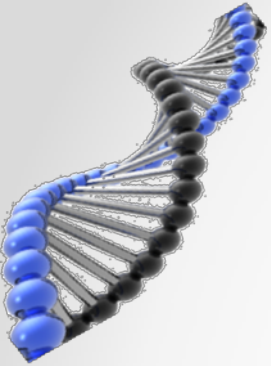
For a Reinforcement Agent, life is a series of

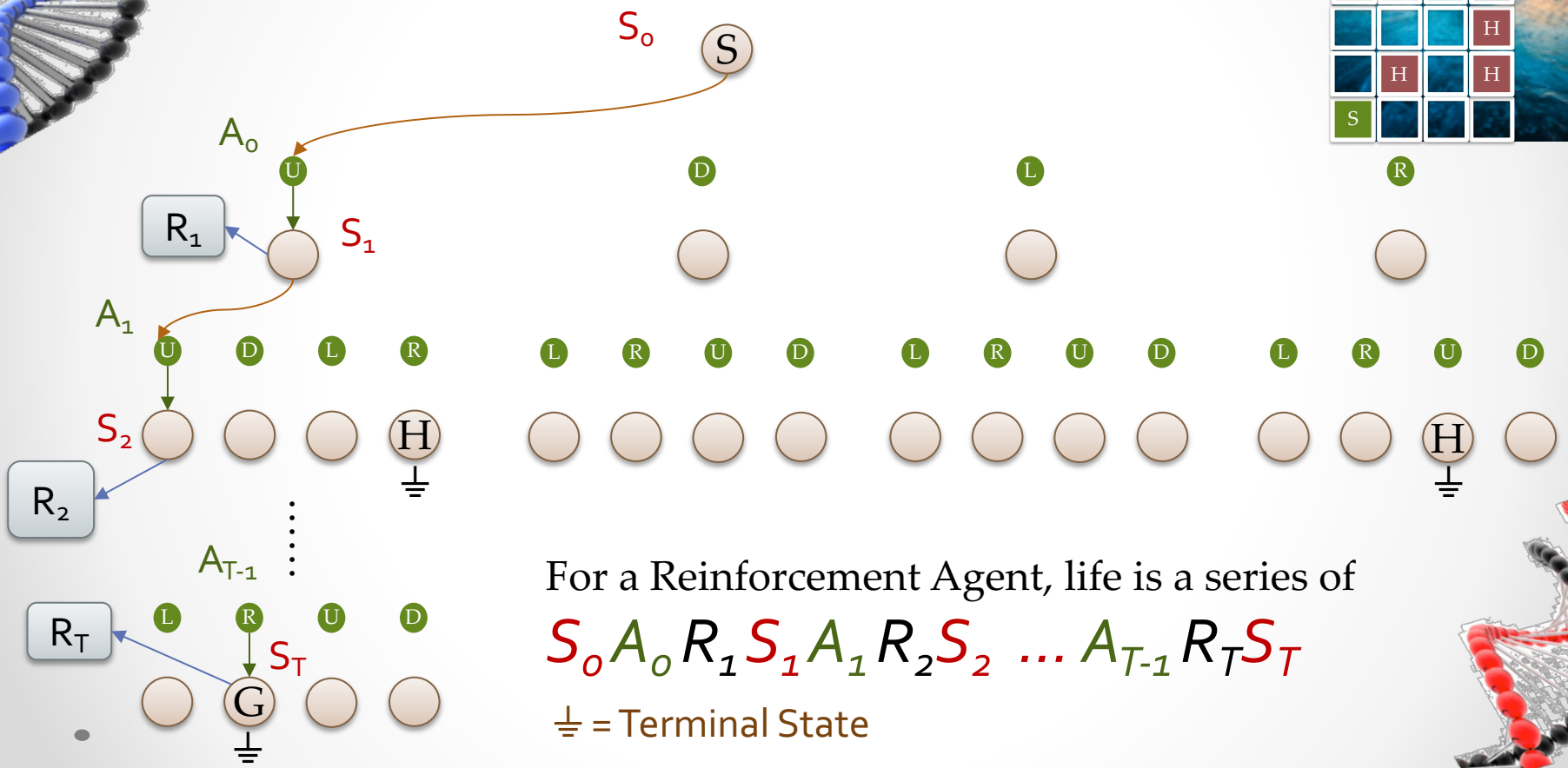
$S_0 A_0 S_1 A_1 S_2 \ldots A_{T-1} S_T$

⏚ = Terminal State



Notation : State, Action, Reward



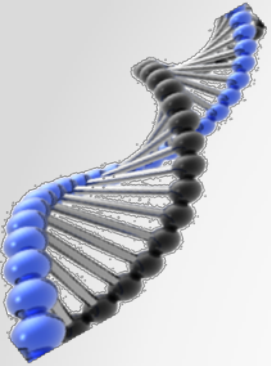


For a Reinforcement Agent, life is a series of

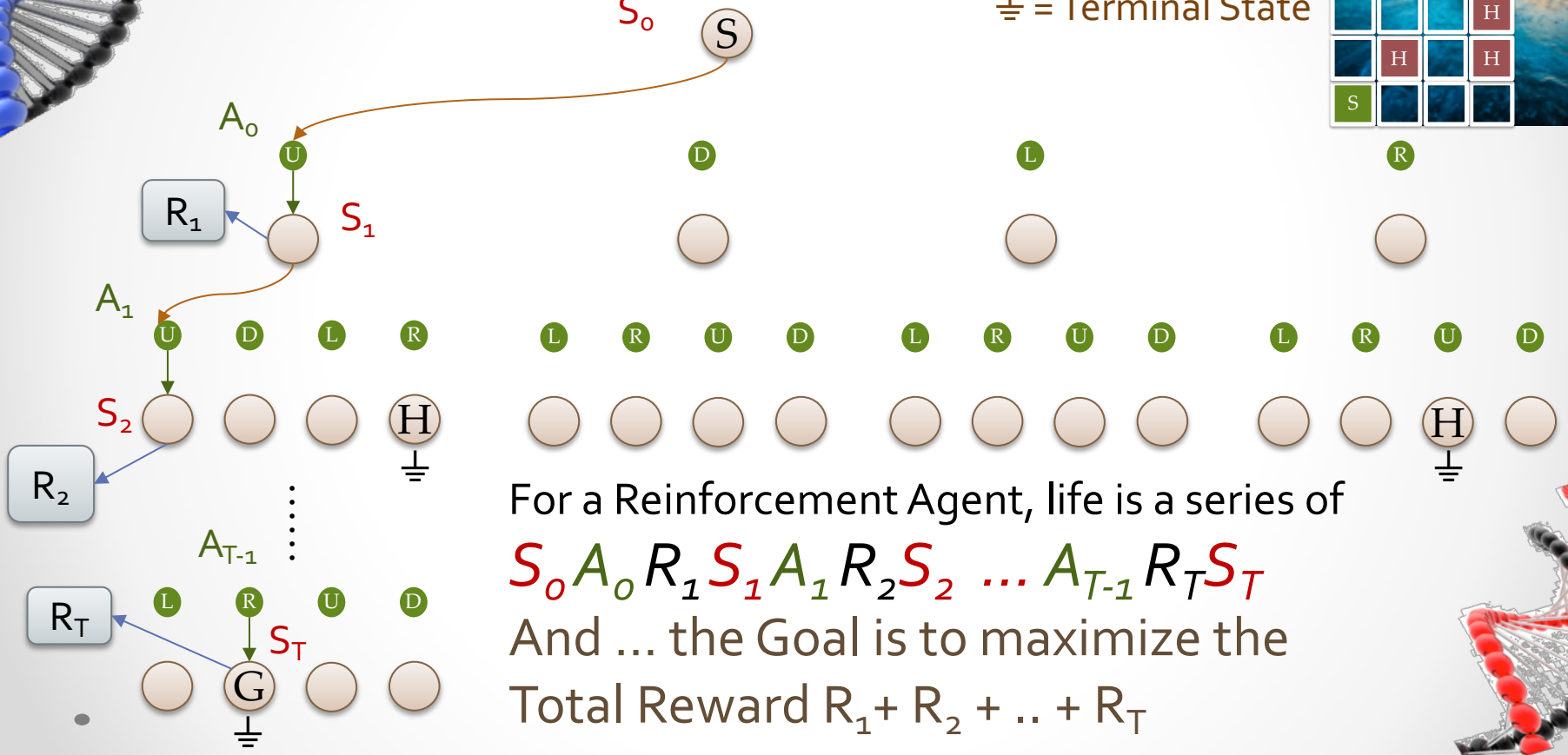
$$S_0 A_0 R_1 S_1 A_1 R_2 S_2 \ \ldots\ A_{T-1} R_T S_T$$

⏚ = Terminal State

Notation : State, Action, Reward



⏚ = Terminal State

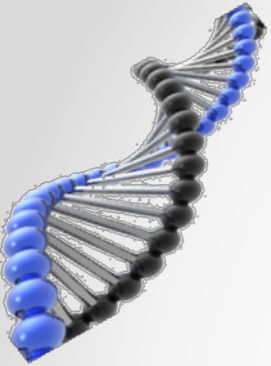


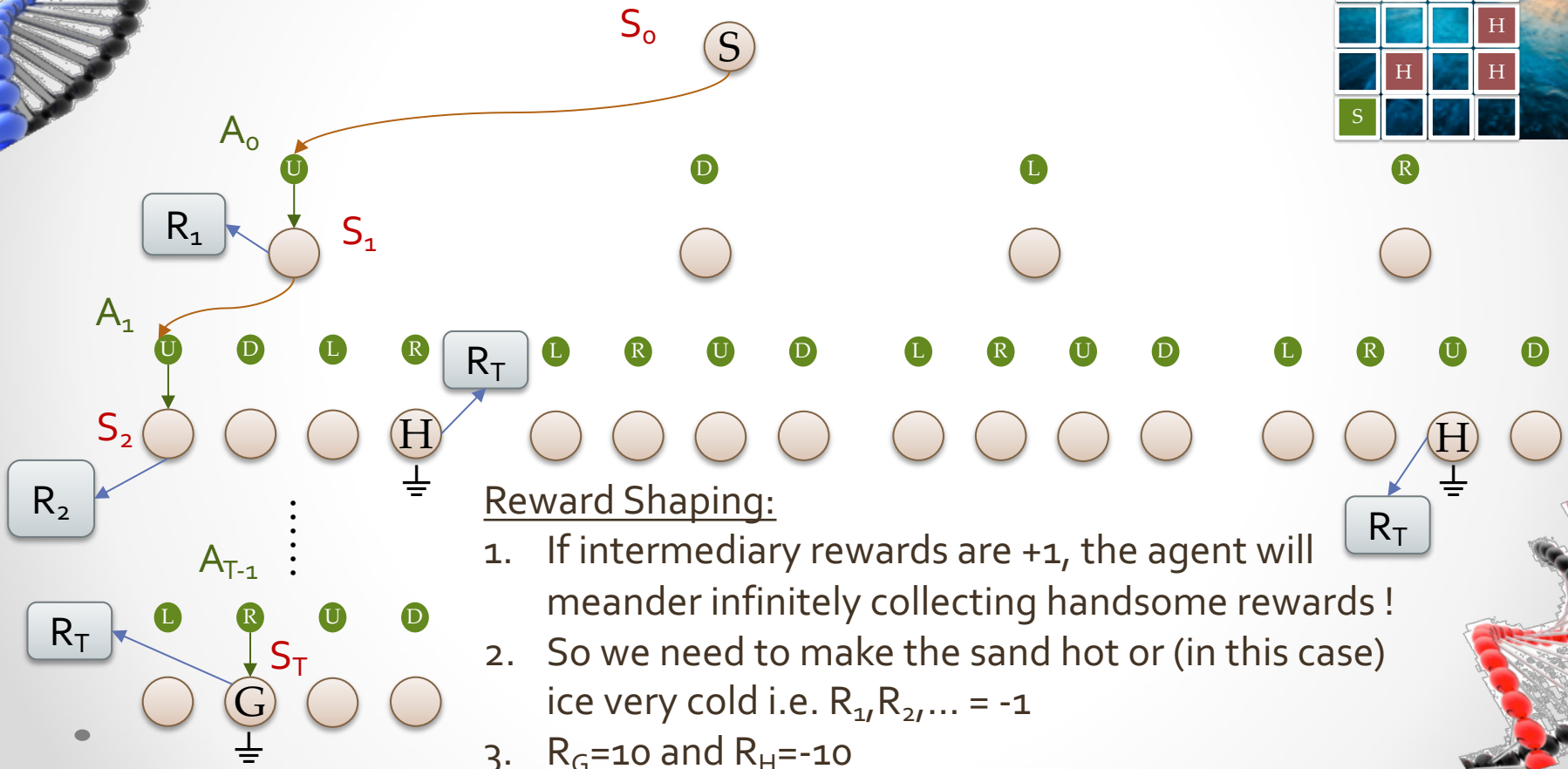
For a Reinforcement Agent, life is a series of

$S_0 A_0 R_1 S_1 A_1 R_2 S_2 \ \ldots \ A_{T-1} R_T S_T$

And … the Goal is to maximize the Total Reward $R_1 + R_2 + .. + R_T$

Notation : State, Action, Reward

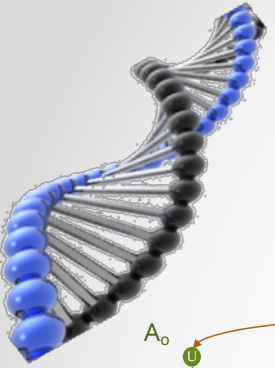


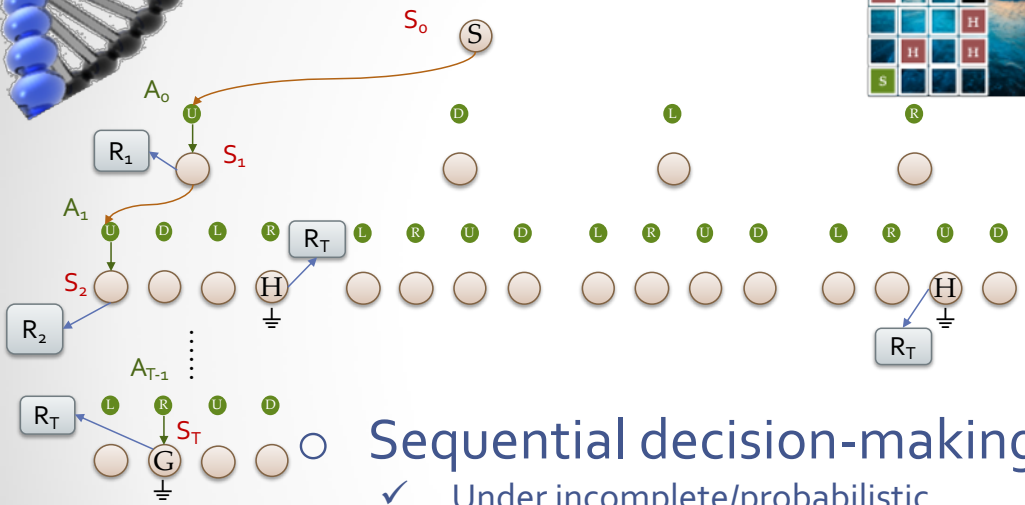


Reward Shaping:

1. If intermediary rewards are +1, the agent will meander infinitely collecting handsome rewards !
2. So we need to make the sand hot or (in this case) ice very cold i.e. $R_1, R_2, \ldots = -1$
3. $R_G=10$ and $R_H=-10$

In short DRL is …

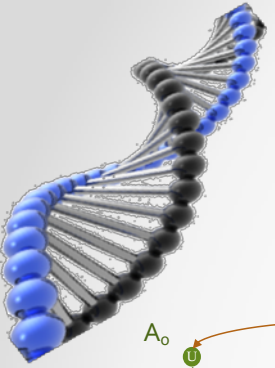


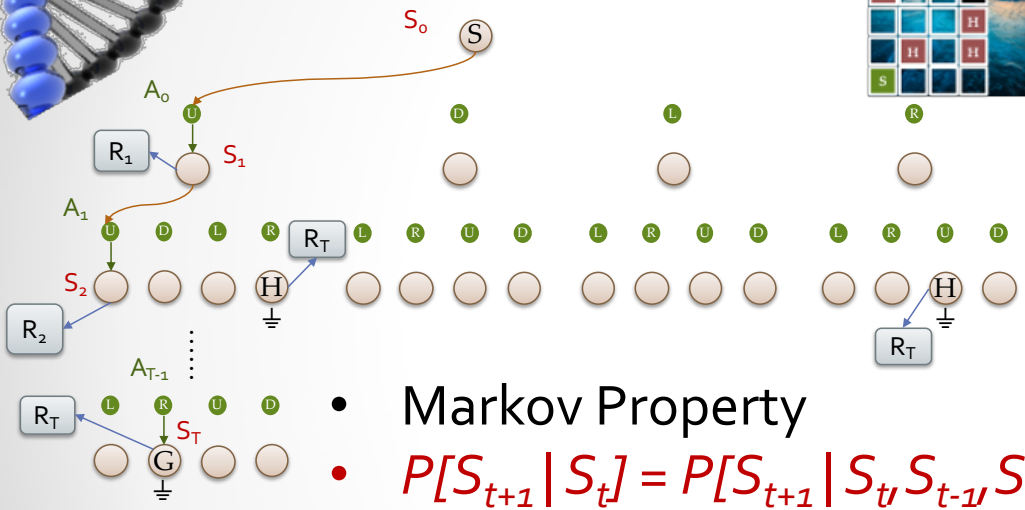


- Sequential decision-making
  - ✓ Under incomplete/probabilistic information, even if we have full observability
  - ✓ Uncertain environment
- Evaluative feedback
- Exhaustive sampling

- There is a State Space
  - State is actually an observation, need not represent the environment
  - *Encoding state requires some thought*
- There is an action space
- There are rewards
  - Goal is to maximize total rewards (= Return)
  - R_t = One step Return
- Same action need not result in same return
- Action to state transition might be stochastic

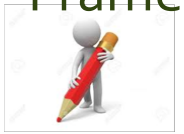
Markov Process



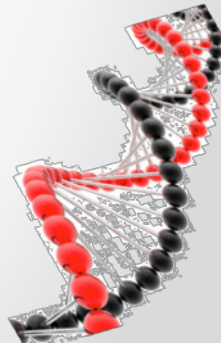


- Markov decision process formally describes an environment for Reinforcement Learning (Silver)
- Powerful Mathematical Framework for RL

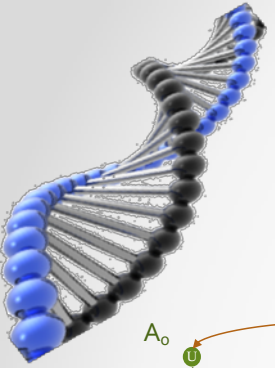
- Markov Property
- $P[S_{t+1} | S_t] = P[S_{t+1} | S_t, S_{t-1}, S_{t-2}, \ldots]$

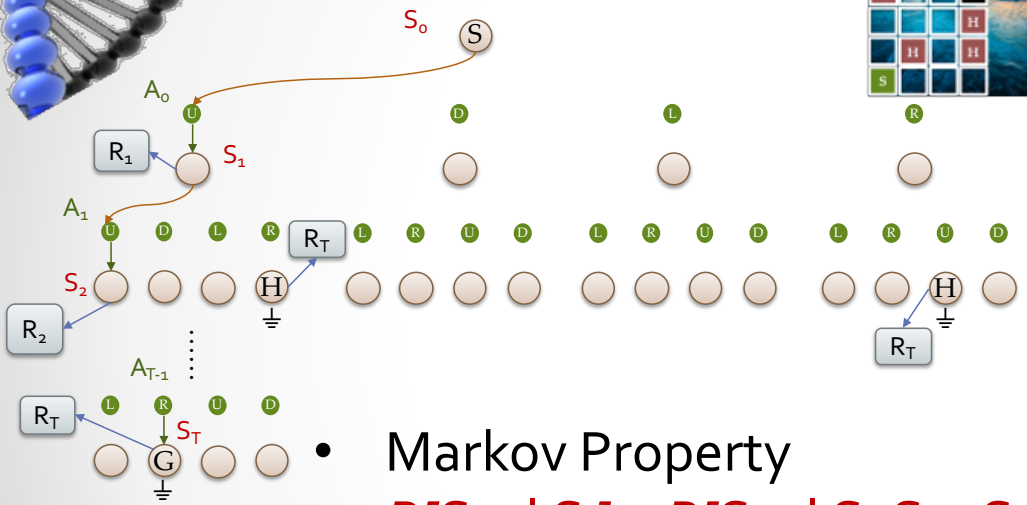


- Current state is sufficient to predict the future
- A Markov process is a *memoryless sequence* of states $S_1, S_2, S_3, \ldots$ with the Markov Property



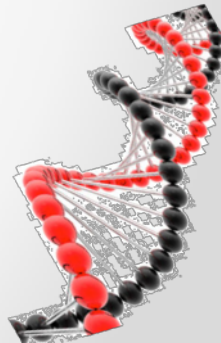
Markov Process



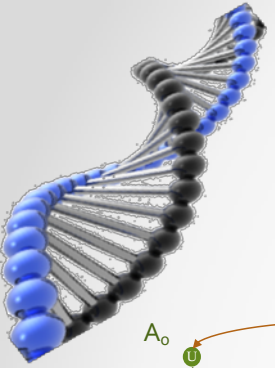


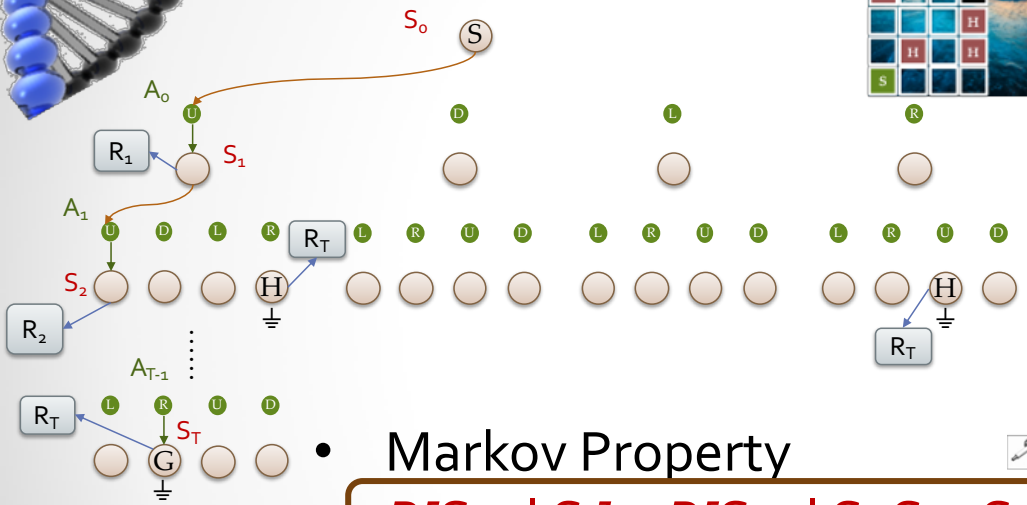
- Good
  - Don't have to remember all trajectories & paths
  - Don't have to consider them for decisions
- Bad
  - The state has to be constructed s.t it encodes all the relevant information e.g. 4 frames in Atari RL agent

- Markov Property
- $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_t, S_{t-1}, S_{t-1}, \ldots]$
- Current state is sufficient to predict the future
- A Markov process is a *memoryless sequence* of states $S_1, S_2, S_3, \ldots$ with the Markov Property

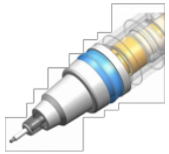


Markov Process

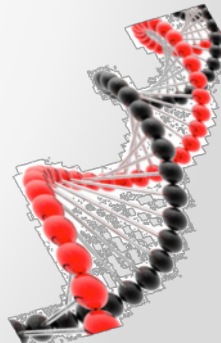




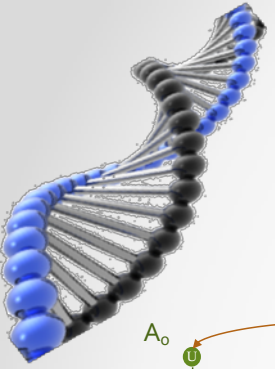
- Good
  - Don't have to remember all trajectories & paths
  - Don't have to consider them for decisions
- Bad
  - The state has to be constructed s.t it encodes all the relevant information e.g. 4 frames in Atari RL agent

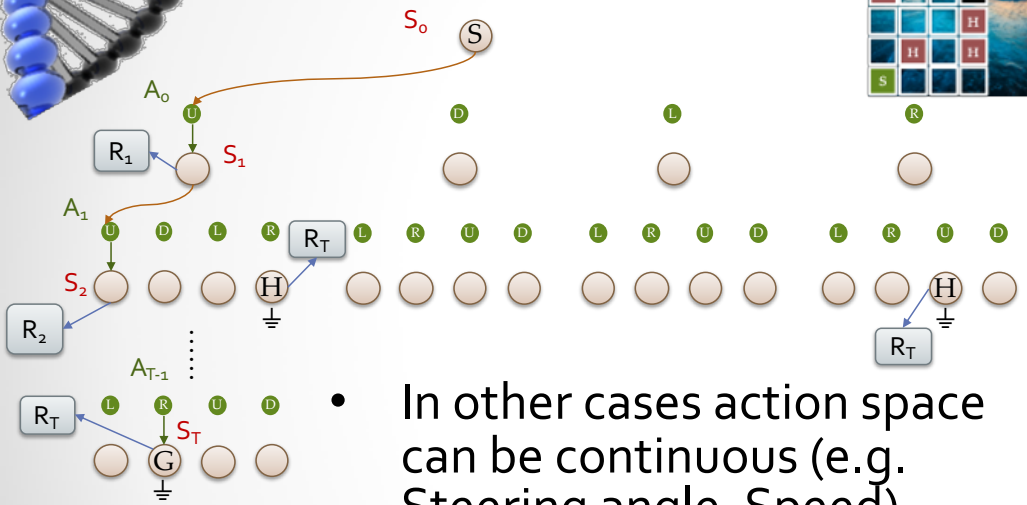


- Markov Property
- $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_t, S_{t-1}, S_{t-1}, \ldots]$
- Current state is sufficient to predict the future
- A Markov process is a *memoryless sequence* of states $S_1, S_2, S_3, \ldots$ with the Markov Property



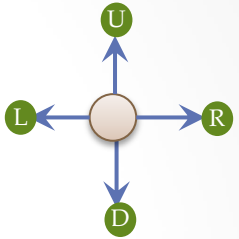
Action Space



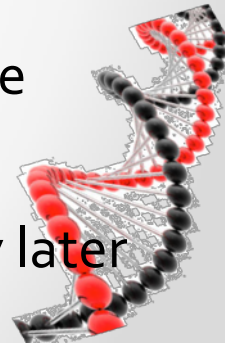


- In other cases action space can be continuous (e.g. Steering angle, Speed)
- Different actions at different states (games, parking vs. merging)
- Large Action space

- 4 actions



- How does an agent choose the actions ?
  - Equi-probable
  - ε-greedy
  - Greedy
  - Softmax distribution
- Best action need not be deterministic
  - E.g. Rock-Paper-Scissor
- We will examine policy later



Reward Space & Discounting

R_t = Reward at a time step

G_t = Total Return

Goal is to maximize this

$$G_{t(t=0)} = R_1 + R_2 + .. + R_T$$

$$G_t = R_{t+1} + R_{t+2} + .. + R_T$$

Add Discount factor γ to account for uncertainties,

$$G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1} \quad ..(3.8)$$

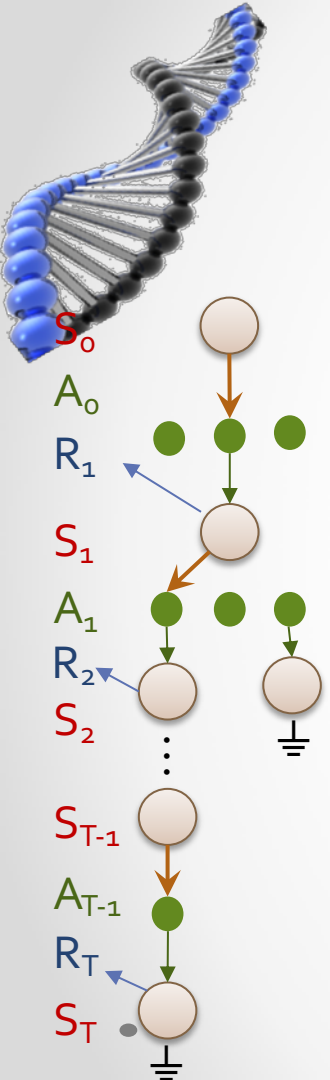
$\gamma \in (0,1)$

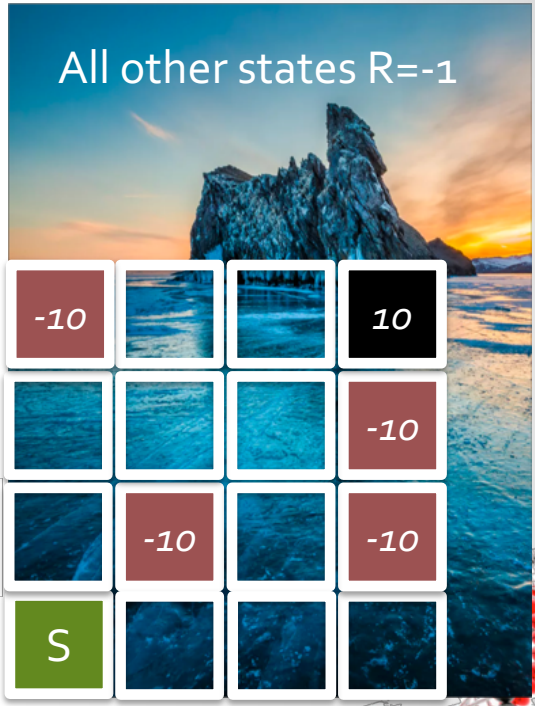
close to 0 gives "myopic"

close to 1 gives "far-sighted"

γ = 0.9 for continuous,

= 1.0 for sequences that terminate





Reward Space & Discounting

R_t = Reward at a time step

G_t = Total Return

Goal is to maximize this

$$G_{t(t=0)} = R_1 + R_2 + .. + R_T$$

$$G_t = R_{t+1} + R_{t+2} + .. + R_T$$

Add Discount factor γ to account for uncertainties

$$G_t = \sum_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1} \quad ..(3.8)$$

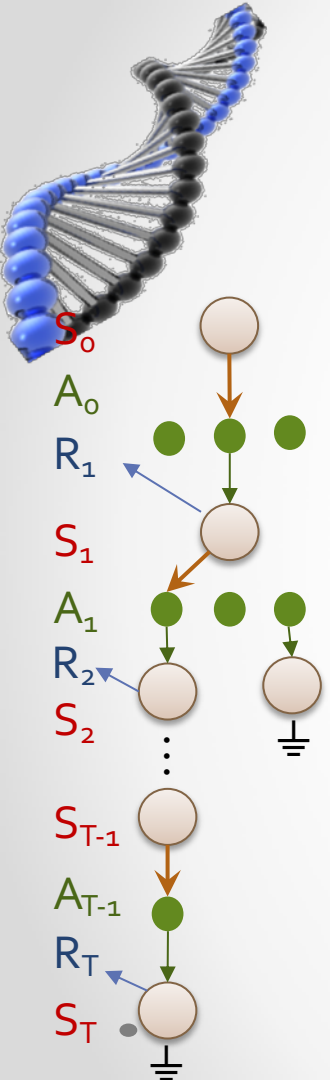
$\gamma \in (0,1)$

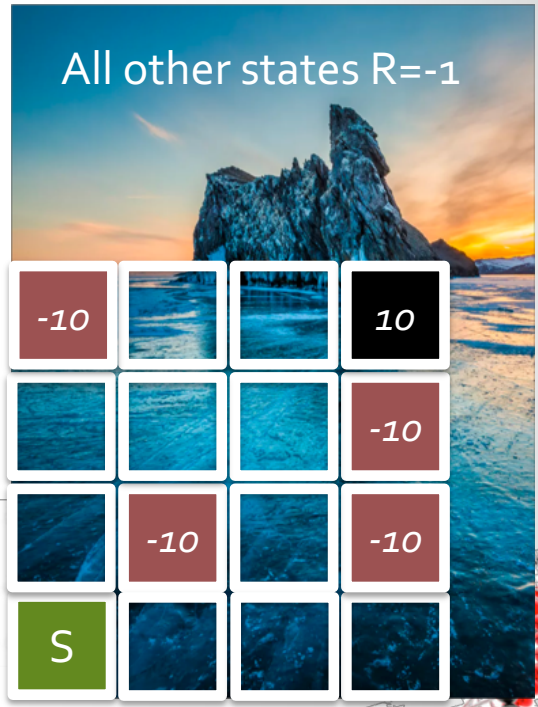
close to 0 gives "myopic"

close to 1 gives "far-sighted"

γ = 0.9 for continuous,

= 1.0 for sequences that terminate

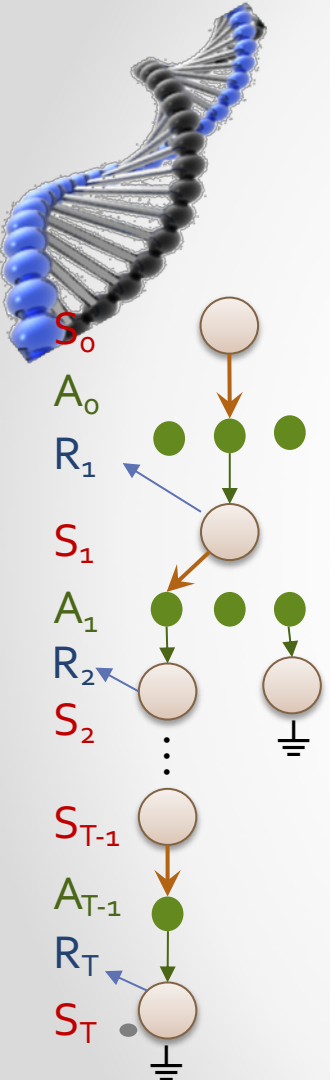


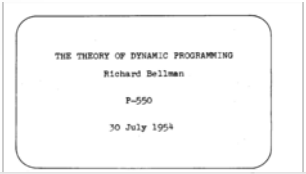


Richard Bellman

Richard E. Bellman invented Dynamic Programming

His formulations form the foundation of Reinforcement Learning





THE THEORY OF DYNAMIC PROGRAMMING

Richard Bellman

P-550

30 July 1954

Turning to a more precise discussion, let us introduce a small amount of terminology. A sequence of decisions will be called a policy, and a policy which is most advantageous according to some preassigned criterion will be called an optimal policy.

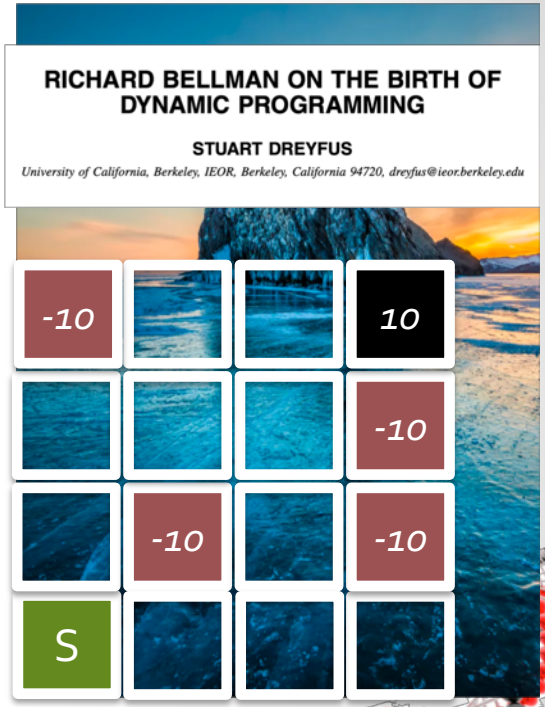
Dynamic Programming – refactor into sub problems

Decomposition & Aggregation

$$G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1}$$

$$= R_{t+1} + \Sigma_{k=1 \text{ to } \infty} \gamma^k R_{t+k+1}$$

$$= R_{t+1} + \gamma G_{t+1} \ldots (3.9)$$

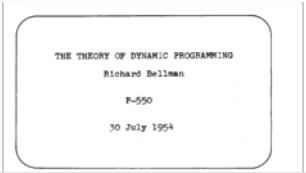


An interesting paper :
http://smo.sogang.ac.kr/doc/dy_birth.pdf

Richard Bellman

Richard E. Bellman invented Dynamic Programming

His formulations form the foundation of Reinforcement Learning


THE THEORY OF DYNAMIC PROGRAMMING

Richard Bellman

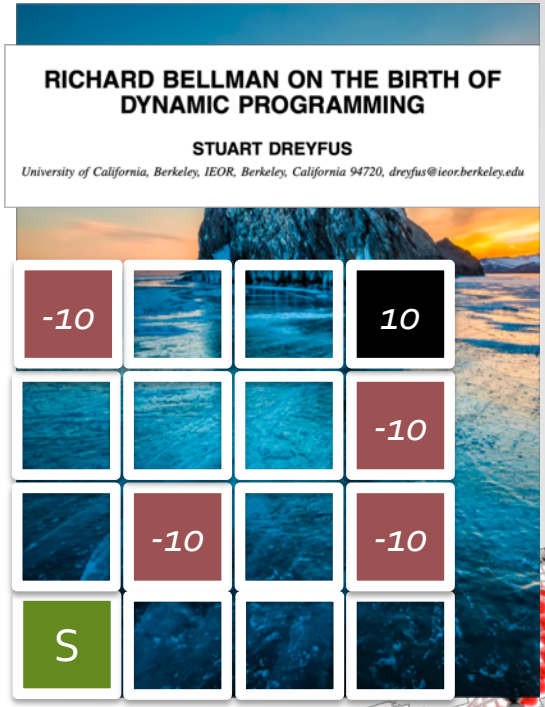
P-550

30 July 1954


The RAND Corporation

1700 MAIN ST. • SANTA MONICA • CALIFORNIA

Turning to a more precise discussion, let us introduce a small amount of terminology. A sequence of decisions will be called a policy, and a policy which is most advantageous according to some preassigned criterion will be called an optimal policy.

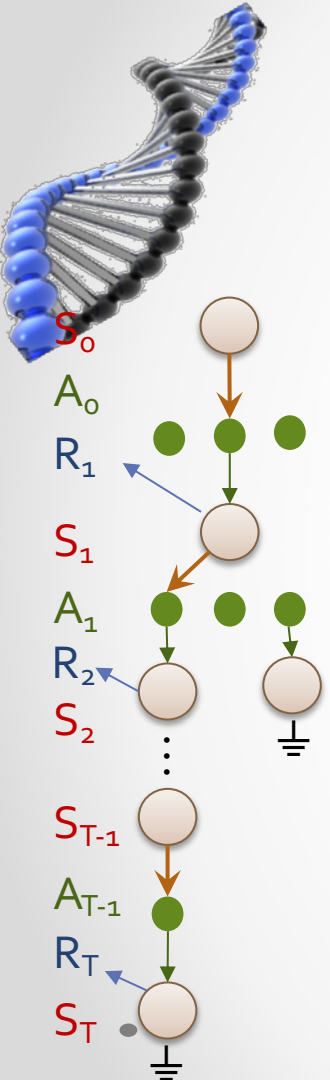


Dynamic Programming – refactor into sub problems

Decomposition & Aggregation

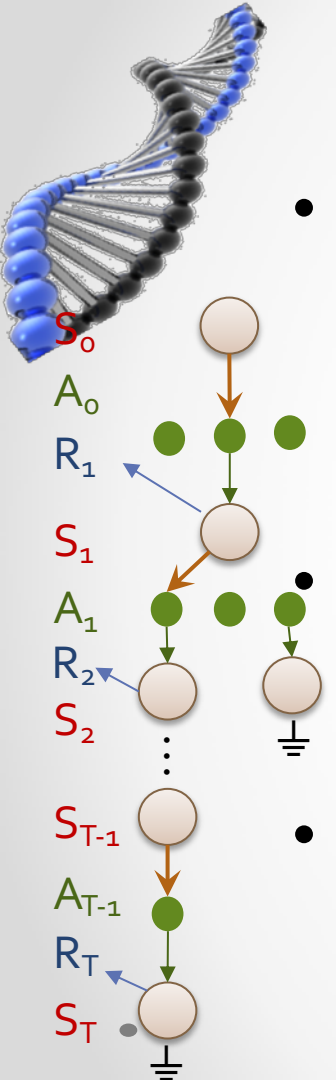
$$G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1}$$
$$= R_{t+1} + \Sigma_{k=1 \text{ to } \infty} \gamma^k R_{t+k+1}$$
$$= R_{t+1} + \gamma G_{t+1} \ldots (3.9)$$

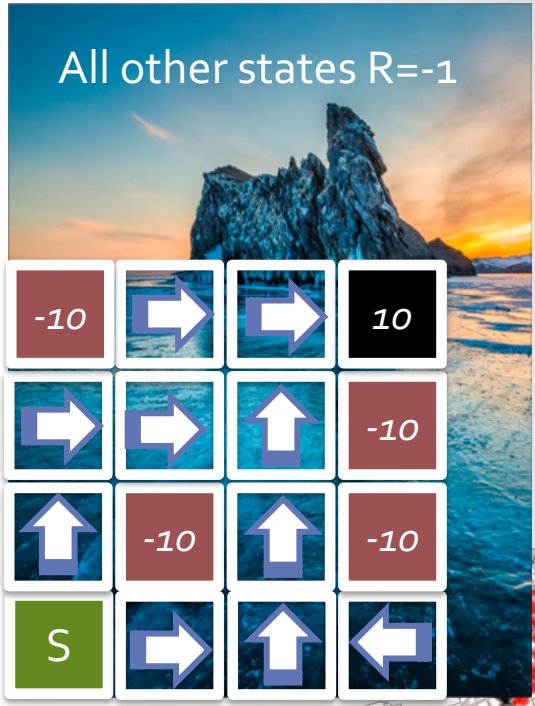
An interesting paper :
http://smo.sogang.ac.kr/doc/dy_birth.pdf



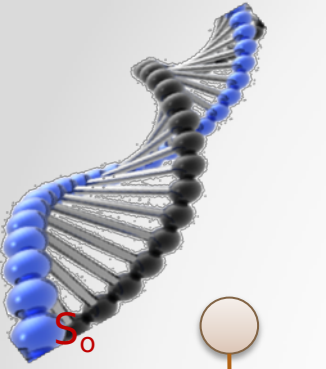
Policy

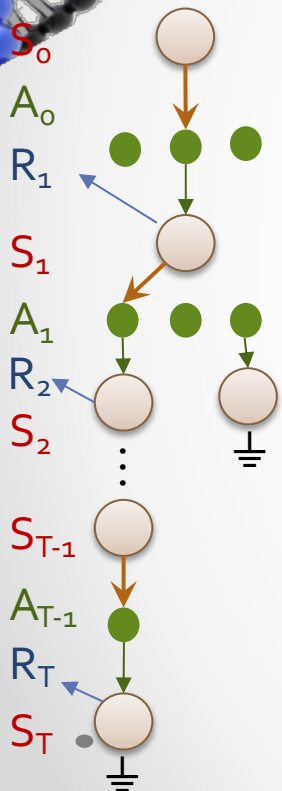
- How does an agent choose an action ?
  - Equi-probable Random
  - ε-Greedy
  - Greedy
  - Softmax distribution
- Or A deterministic policy as given here
  - Which is possible if we have the full model of the environment
- Policy *guides* the decisions
  - For example it could order actions as preferences that reflect the current understanding or belief

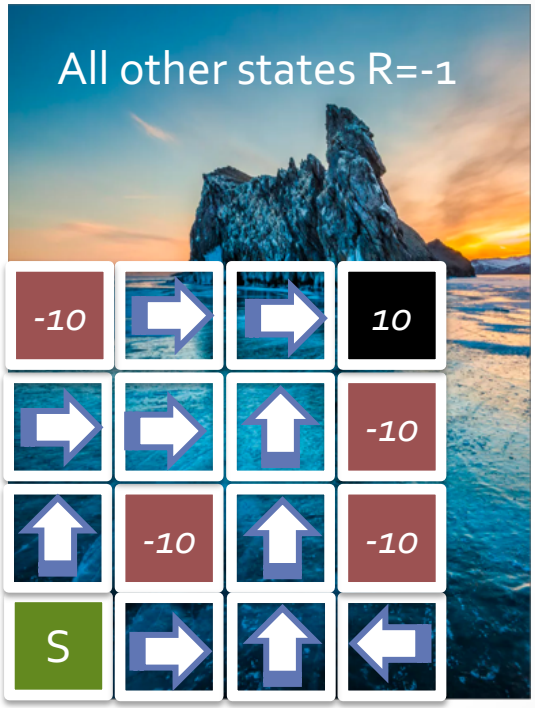




State Space & Policy

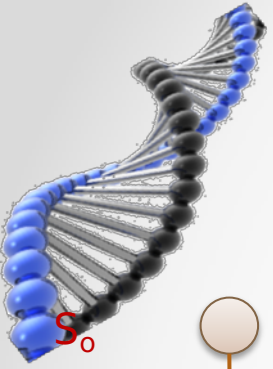


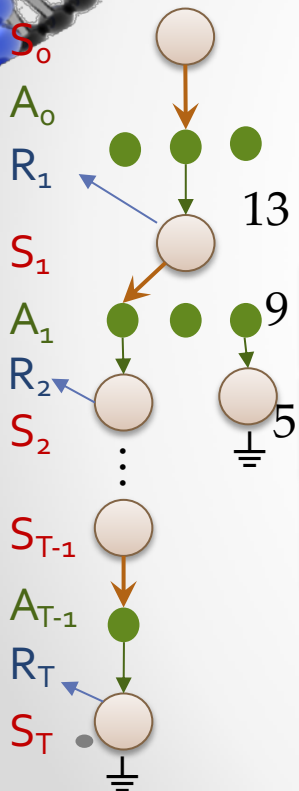


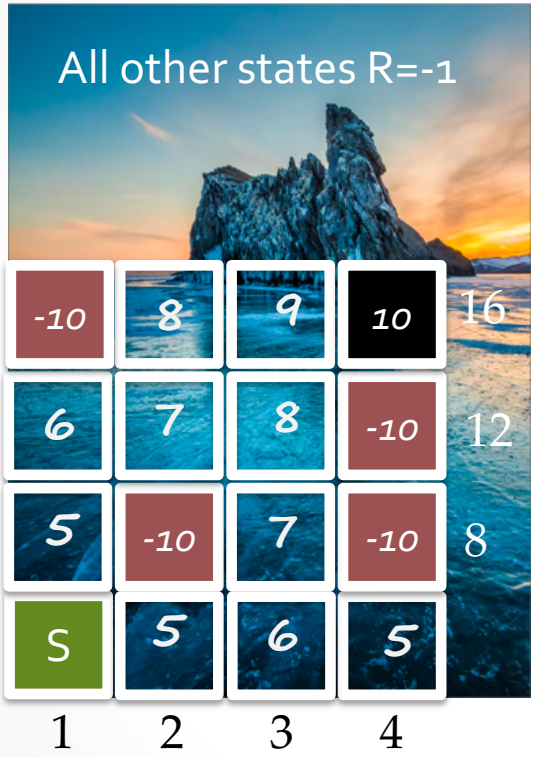


State	Action	Probability
1	U	1.0
2	R	1.0
3	U	1.0
4	L	
5	U	
6		
7	U	
8		
9	R	
10	R	
11	U	
12		
13		
14	R	
15	R	
16		

State-Value Function







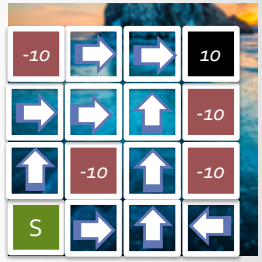
Given a policy, we can compute the value at each state using the Bellman Equation

$G_t = R_{t+1} + G_{t+1}$

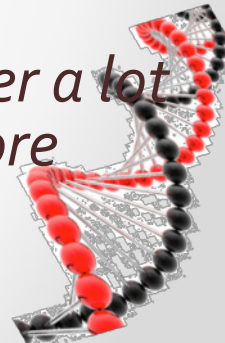
Or

$V_\pi(s) = R_{t+1} + V_\pi(s_{t+1} | s_t=s)$

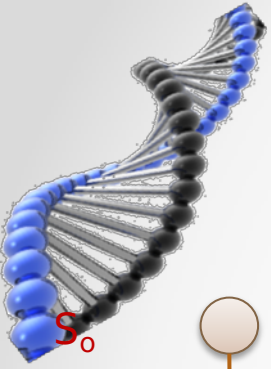
As shown in the diagram

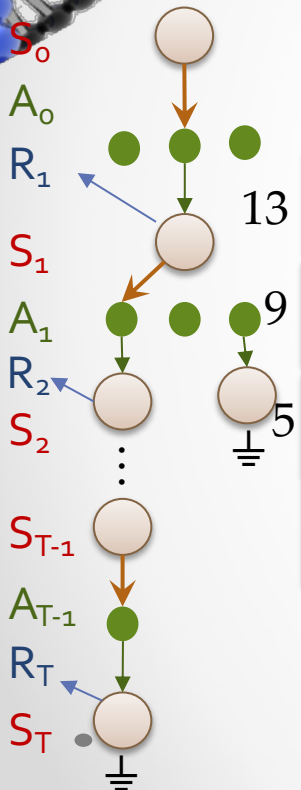


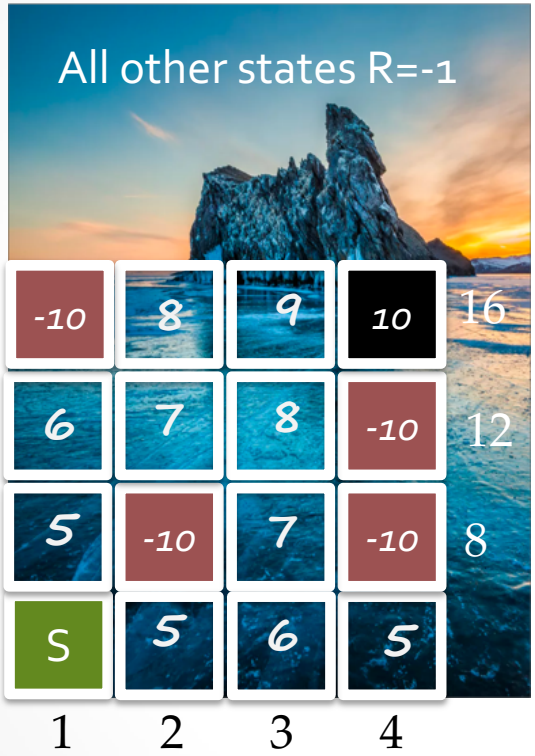
Of course, we are glossing over a lot of details, which we will explore during the rest of the lab



State-Value Function





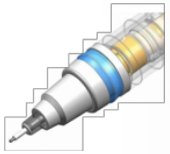


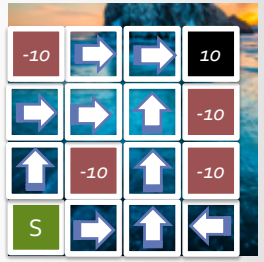
Given a policy, we can compute the value at each state using the Bellman Equation

$$G_t = R_{t+1} + G_{t+1}$$

Or

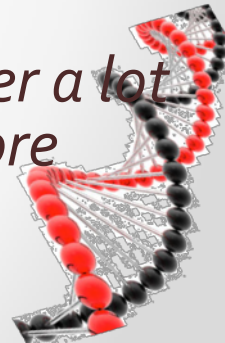
$$V_\pi(s) = R_{t+1} + V_\pi(s_{t+1} | s_t = s)$$





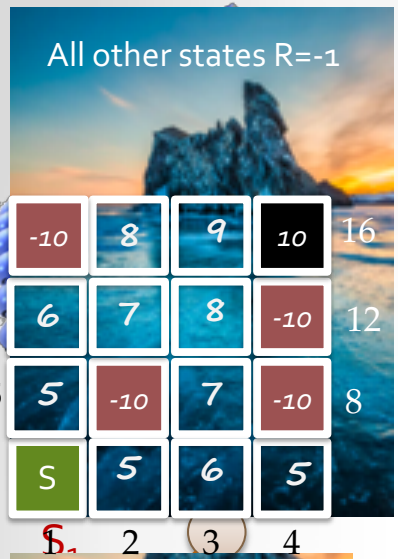
As shown in the diagram

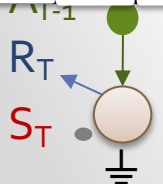
Of course, we are glossing over a lot of details, which we will explore during the rest of the lab

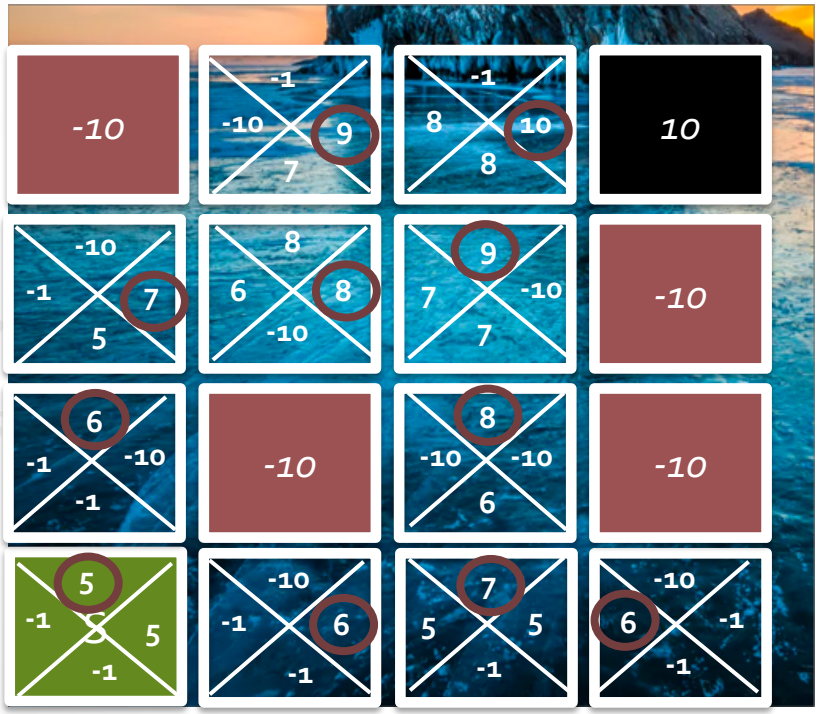


State-Action-Value Function

Iterative Policy Evaluation & Improvement







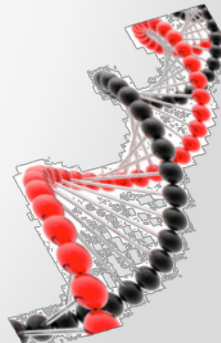
$q_\pi(s,a) = v_\pi(s_{t+1}, a_t | s_t=s, a_t=a)$

$\pi(a|s) = q_*(s,a) = \text{argmax}_{a'}(q(s,a'))$

$v_\pi(s) = R_s + q_\pi(s, \pi(s))$

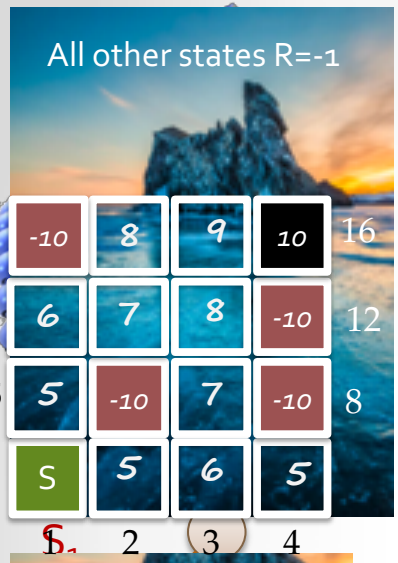
Q-Table

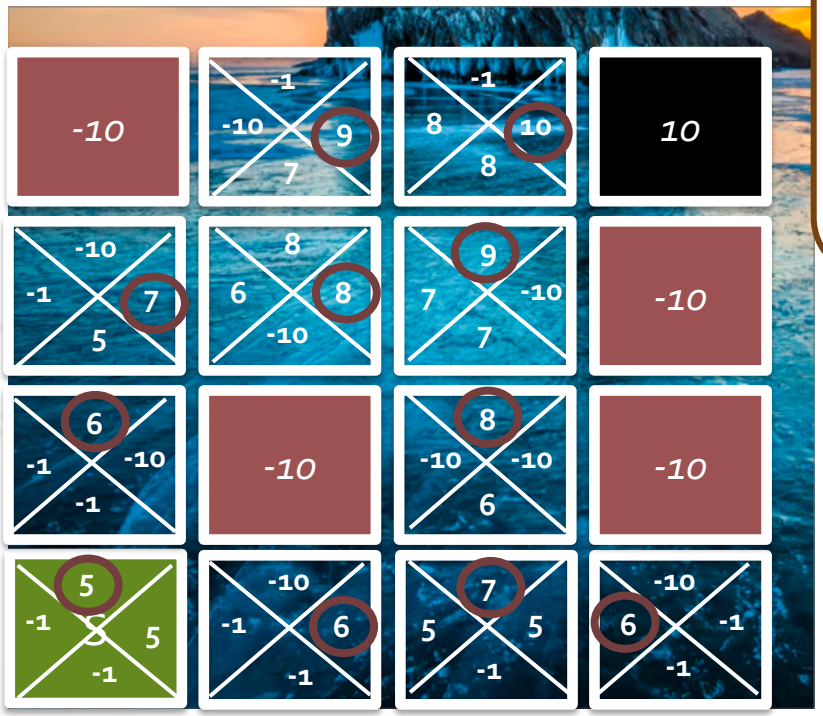
s,a	Q-Value
2,L	-1
2,D	-1
2,U	-10
2,R	6



State-Action-Value Function

Iterative Policy Evaluation & Improvement





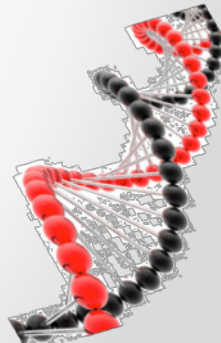
$q_\pi(s,a) = v_\pi(s_{t+1}, a_t | s_t=s, a_t=a)$

$\pi(a|s) = q_*(s,a) = \text{argmax}_{a'}(q(s,a'))$

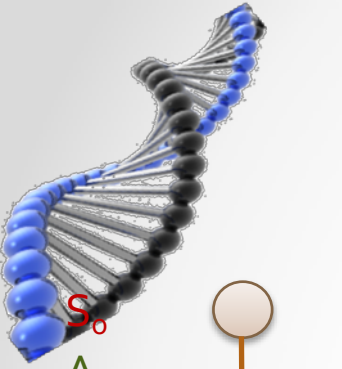
$v_\pi(s) = R_s + q_\pi(s, \pi(s))$

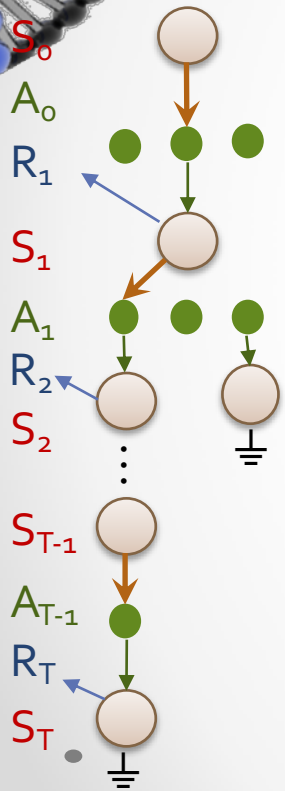
Q-Table

s,a	Q-Value
2,L	-1
2,D	-1
2,U	-10
2,R	6

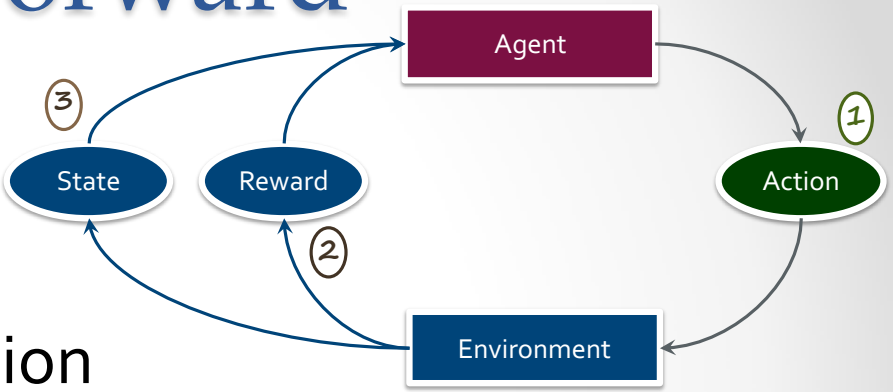


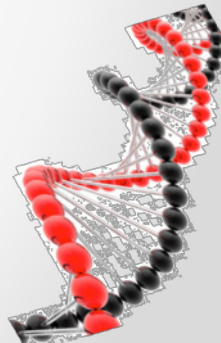
Recap & Forward



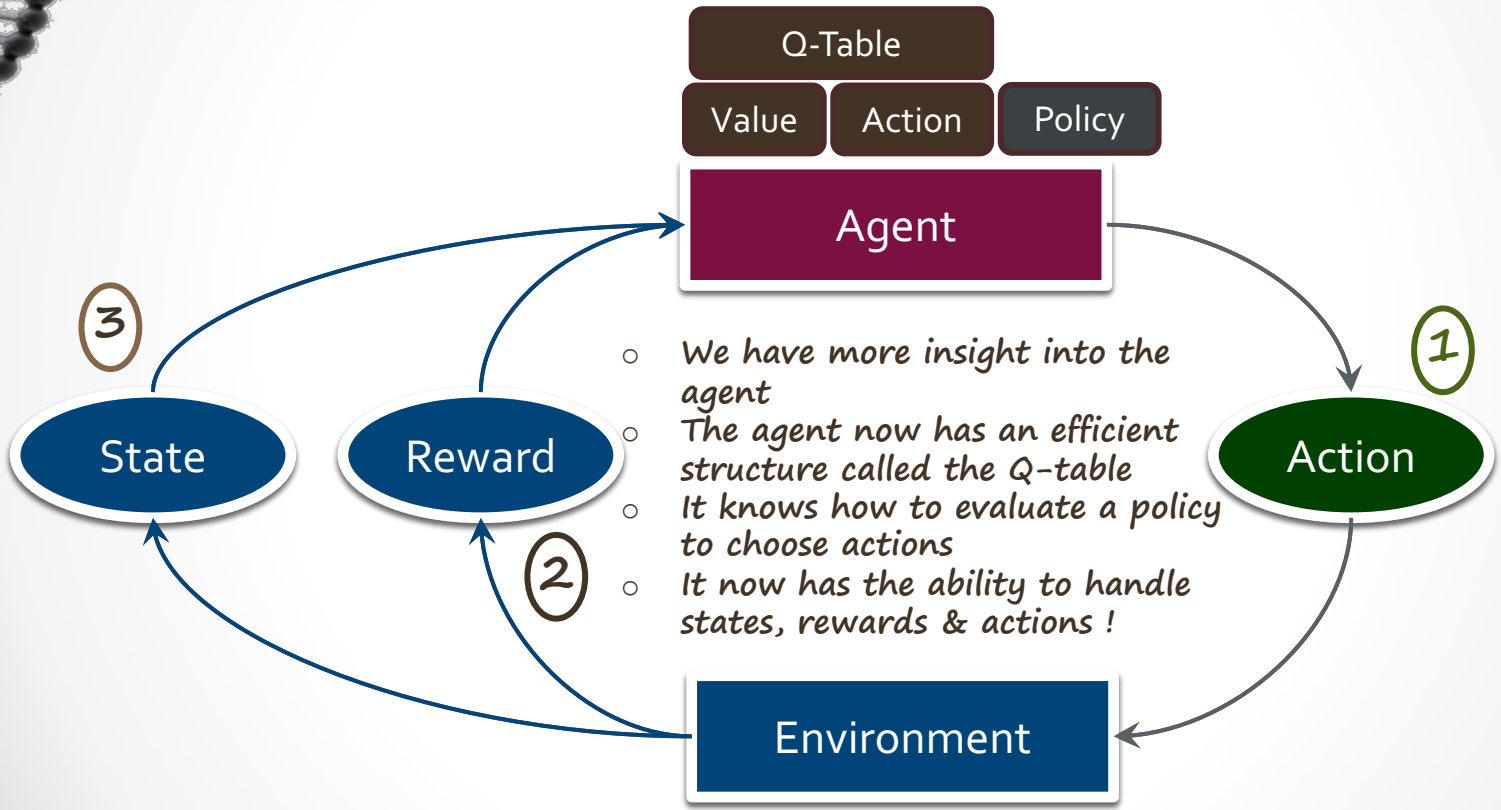


- RL Cycle
- Backup Diagram
- Value Function
- Action-Value Function
  - Value functions define a partial ordering of policies
- Policy – greedy policy ($argmax_{a'}$)
  - state sweep
- Grid World as an example
- Used Dynamic Programming in a model-based fashion
  - Decomposition & Aggregation



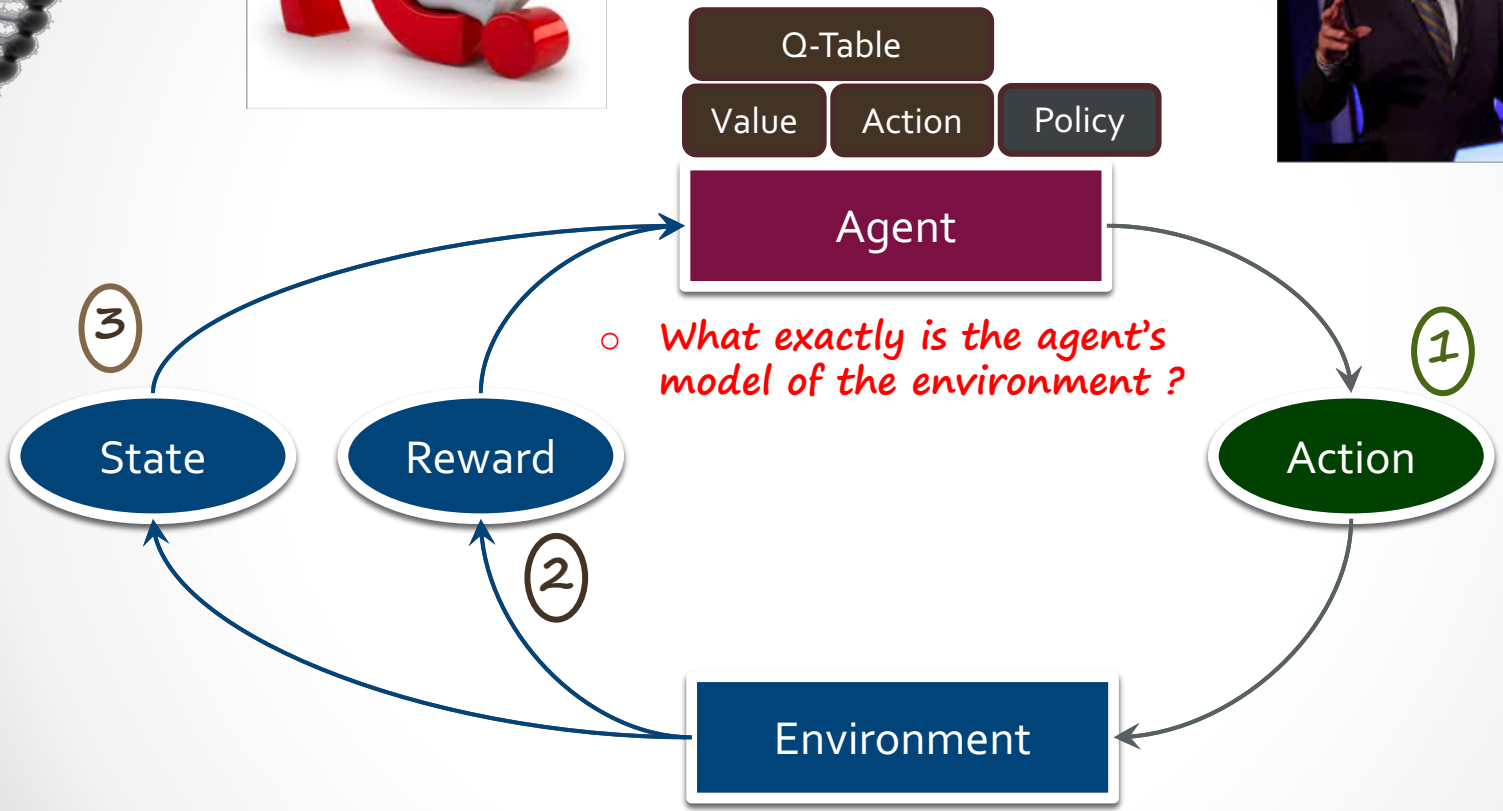


RL Flow



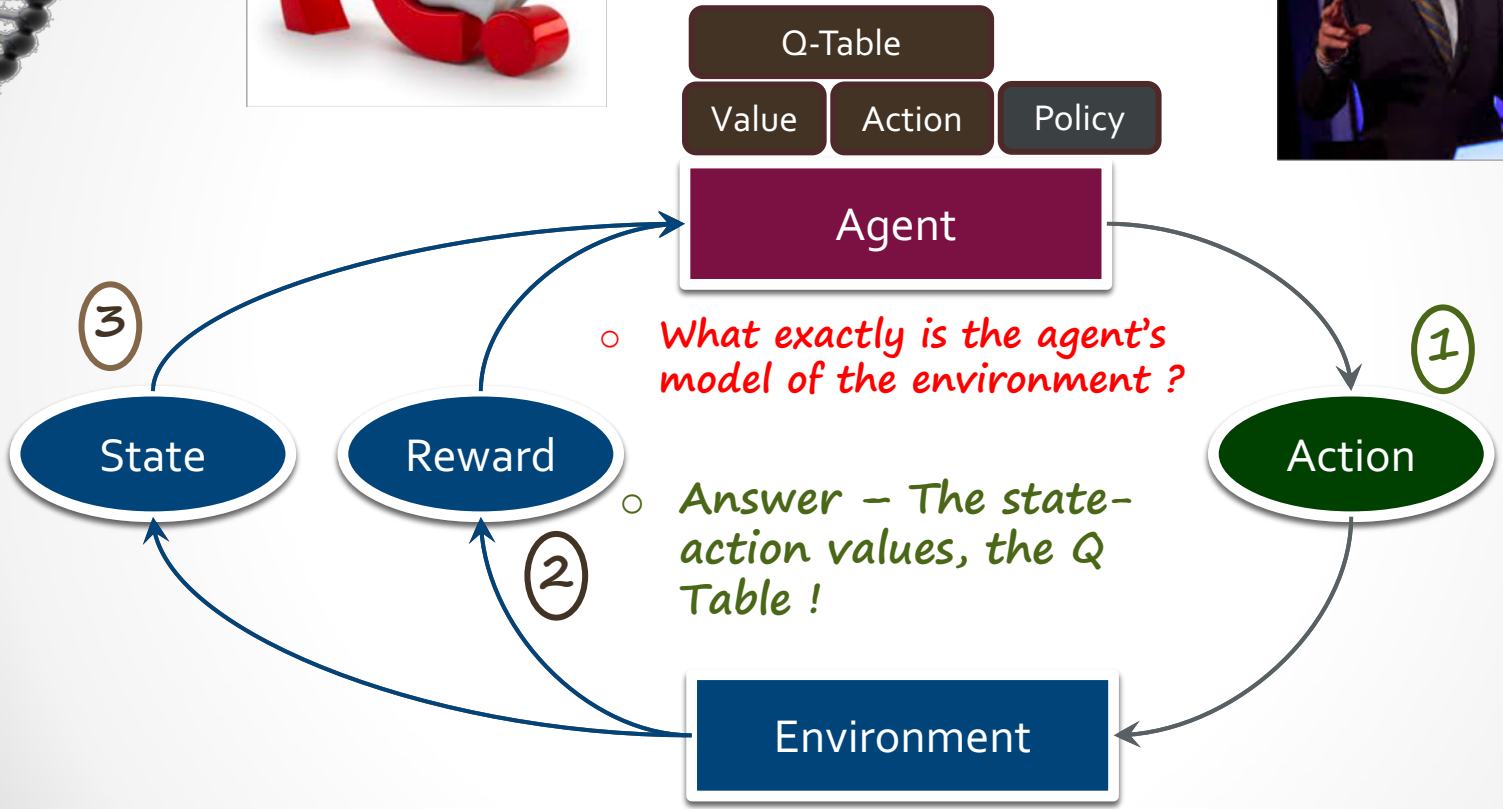
- We have more insight into the agent
- The agent now has an efficient structure called the Q-table
- It knows how to evaluate a policy to choose actions
- It now has the ability to handle states, rewards & actions !

Quiz



- What exactly is the agent's model of the environment ?

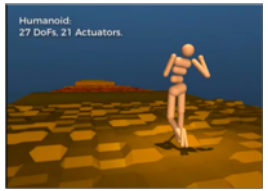
Quiz

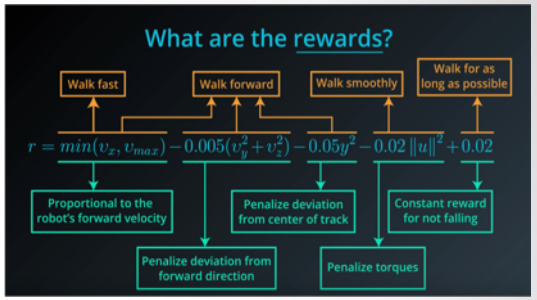


- What exactly is the agent's model of the environment ?
- Answer – The state-action values, the Q Table !

Examples of representation

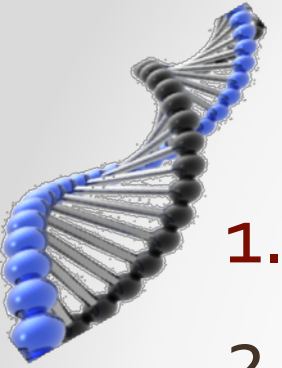
- Atari
  - Grey-scale, 84 X 84,
  - Stacking 4 frames for context
  - CNN as feature extractor





- Walking Humanoids
  - https://www.theverge.com/tldr/2017/7/10/15946542/deepmind-parkour-agent-reinforcement-learning
  - Emergence of Locomotion Behaviors in Rich Environments
  - https://arxiv.org/abs/1707.02286
  - Algorithms : PPO, DPPO,A3C
  - States – Position & Velocities of the joints, the height of the walker body above the ground, the difference between the agents position and the next sampling grid center
  - Actions : Forces and directions on the 21 actuated joints with 28 DoF
  - Rewards :
    - *The reward consists of a main component proportional to the velocity along the x-axis, encouraging the agent to make forward progress along the track, plus a small term penalizing torques. For the walker the reward also includes the same box constraints on the pose as in section 2. For the quadruped and humanoid we penalize deviations from the center of the track, and the humanoid receives an additional reward per time-step for not falling*

$$r = \min(v_x, v_{\max}) - 0.005(v_x^2 + v_y^2) - 0.05y^2 - 0.02\|u\|^2 + 0.02$$

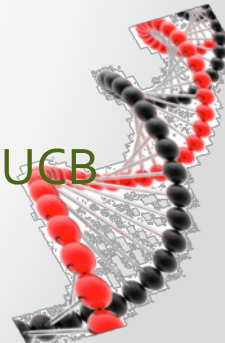


General Observation on Rewards

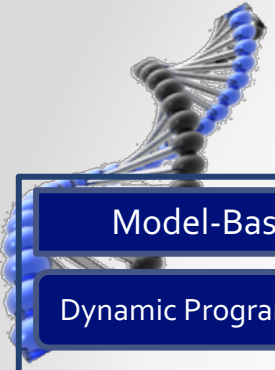
1. Specify the goals, not how to do it
2. Do not over specify, even a hint of how

Enable the agent to pursue novel strategies (e.g. Go game) and discover winning strategies

3. Rewards can change over time
    - This can break stationary assumptions of mechanisms like UCB action selection and optimistic initialization



RL Algorithm Taxonomy

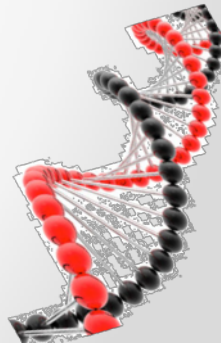


- Model-Based
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration
- Model-Free
  - Monte Carlo
  - TD Learning
  - Value Methods
    - SARSA
    - Q-Learning
    - Deep Q-Learning
      - Fixed Q Targets
      - Experience replay
      - Double DQN
      - Prioritized replay
      - Dueling DQN
      - *Increase robustness*
      - *Reduce bias/variance*
      - *Sample efficiency*
  - Policy Search
    - Hill Climbing
      - Steepest Hill Climbing
        - Cross-entropy
        - Evolution Strategies
      - Random Restart
      - Simulated Annealing
      - Adaptive noise scaling
  - Policy Gradients
    - *Natural Policy Gradients*
    - REINFORCE
    - TRPO
    - Actor-Critic
    - PPO
    - DDPG

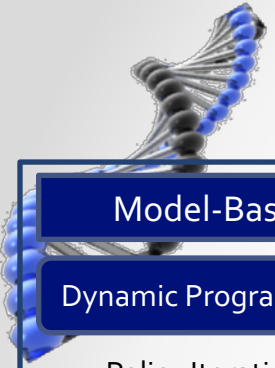
RL ⟷ DRL

Non-linear Function Approximation

- Why Policy Based ?
  - Simplicity
  - Stochastic Action Space
    - Rock-Paper-scissor
  - Continuous
    - Can output a # without exploring values in a continuous space
  - Aliased States



RL Algorithm Taxonomy

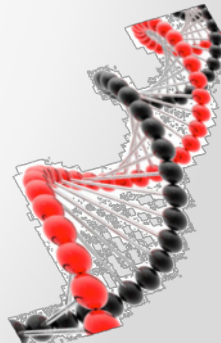


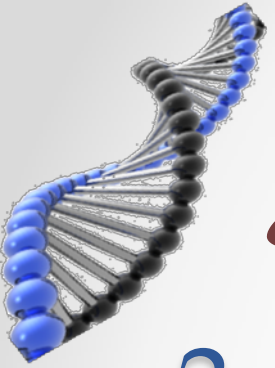
- Model-Based
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration
- Model-Free
  - Monte Carlo
  - TD Learning
  - Value Methods
    - SARSA
    - Q-Learning
    - Deep Q-Learning
      - Fixed Q Targets
      - Experience replay
      - Double DQN
      - Prioritized replay
      - Dueling DQN
      - *Increase robustness*
      - *Reduce bias/variance*
      - *Sample efficiency*
  - Policy Search
    - Hill Climbing
    - Steepest Hill Climbing
      - Cross-entropy
      - Evolution Strategies
    - Random Restart
    - Simulated Annealing
    - Adaptive noise scaling
  - Policy Gradients
    - *Natural Policy Gradients*
    - REINFORCE
    - TRPO
    - Actor-Critic
    - PPO
    - DDPG

RL ⟷ DRL

Non-linear Function Approximation

- Why Policy Based ?
  - Simplicity
  - Stochastic Action Space
    - Rock-Paper-scissor
  - Continuous
    - Can output a # without exploring values in a continuous space
  - Aliased States



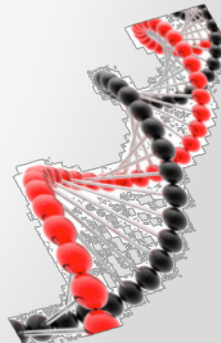


Algorithm Time

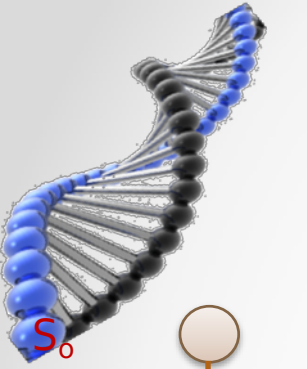
PyTorch Time

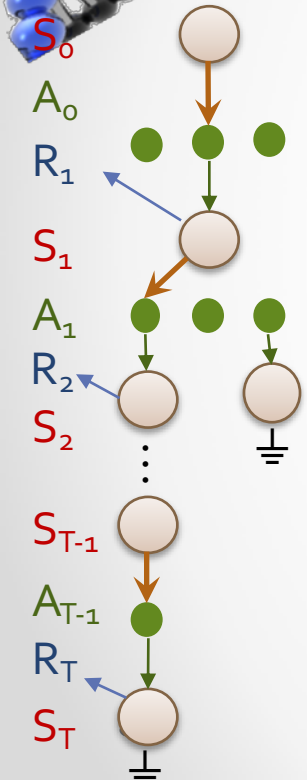
3. Algorithms & Programming : From Monte Carlo to DQN

Let us implement four algorithms and in the process discuss Monte Carlo, TD(), SARSA and Q-Learning



Notations & Definitions - Review





- Episode

  Sequence of State-Action-Reward until an episode ends in T steps

- Return

  $G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1} = R_{t+1} + \Sigma_{k=1 \text{ to } \infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$

- Policy

  Mapping $\pi : S \times A \mapsto [0,1]$ $\quad \pi(a|s) = \mathbb{P}(A_t=a, S_t=s)$

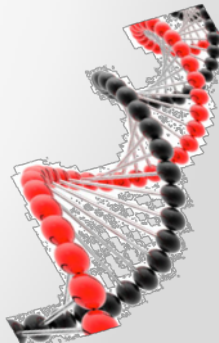
- Value

  $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t=s]$ … state-value Function

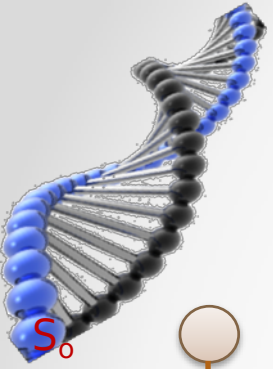
  $= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}(s_{t+1}| s_t=s)]$

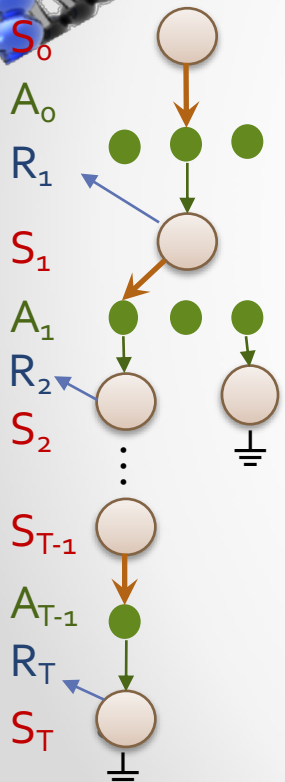
  $= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1}| s_t=s)]$

  $q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t=s, A_t=a]$ … action-value function



Notations & Definitions - Review





- Episode

  Sequence of State-Action-Reward until an episode ends in T steps

- Return

$$G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1} = R_{t+1} + \Sigma_{k=1 \text{ to } \infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma G_{t+1}$$

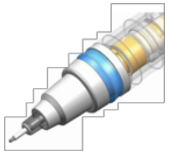
- Policy

  Mapping $\pi : SxA \mapsto [0,1]$ $\quad \pi(a|s) = \mathbb{P}(A_t=a, S_t=s)$

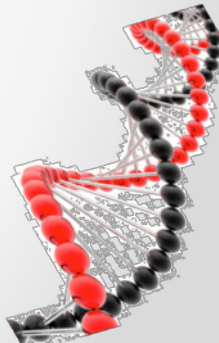
- Value

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t=s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}(s_{t+1}|\, s_t=s)] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1}|\, s_t=s)] \end{aligned}$$

… state-value Function



$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t=s, A_t=a]$$ … action-value function



Hands On #1 - Skating the Frozen Lake

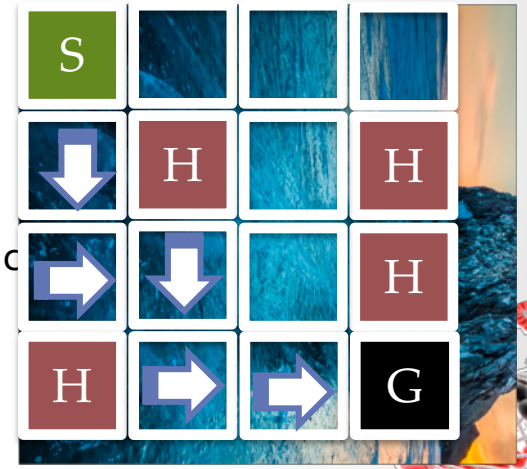
Goal:

- Get familiar with programming OpenAI Gym & it's interfaces
- Solve OpenAI Gym : Frozen Lake, given an optimum deterministic policy
- Frozen Lake deterministic : episodes before solve

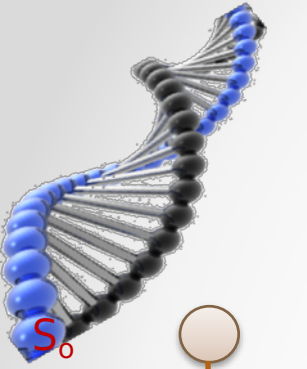
Steps:

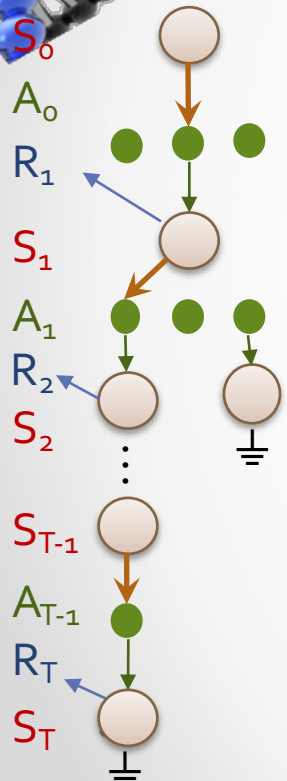
Notebook : frozen_lake_01.ipynb

1. Understand OpenAI Setup & interface
2. Examine Random Policy Run
3. Run the cells upto *5. Hands-on ToDo*
4. Create aPolicy which captures the deterministic polic
5. Use deterministic policy to skate to the goal
6. Metrics that can be programmed:
   - Episodes before solve
   - Solved 0.78 over 100 consecutive runs
   - Max score over 1000 episodes
   - Avg over 1000 episodes

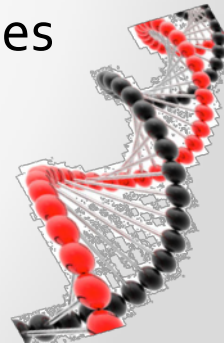


Monte Carlo





- Model-free method
- Usually we do not have the model or the transition matrix. Then how do we arrive at G_t, $q_\pi(s,a)$ *et al ?*
- Monte Carlo brings deterministic expectation by repeated random sampling. i.e. *even though the underlying problem involves a great degree of randomness, we can infer useful information we can trust by collecting lots of samples*
- *Monte Carlo estimates a value by averaging across a large number of episodes*
- Remember, a state-action might be visited several times and the q-value can and will be different.
  - *Averaging gives the best estimate across a large number of episodes*
  - *Either average (across episodes) considering only during 1st visit in an episode or*
    - *Consider all visits and average the value*



Monte Carlo Algorithm

1. Initialize variables, set policy π
2. Generate episodes using policy π - $S_0 A_0 R_1 S_1 A_1 R_2 S_2 \ \ldots \ A_{T-1} R_T S_T$
3. For each episode
   4. For each time step
      5. For each state, action pair
         6. If 1st visit to (S_t, A_t)
            7. Calculate the return G_t (S_t, A_t) - $G_t = \Sigma_{k=0 \text{ to } \infty} \gamma^k R_{t+k+1}$
            8. Add G_t to the return column - $\Sigma q(s,a)$
            9. Increment $N(S_t, A_t)$
10. Average $G_t(S_t, A_t) = \Sigma q(s,a) / N(S_t, A_t)$ and
11. Return the Q table q(s,a)

$Q(s,a) \mapsto Q_\pi(s,a)$ *for a larger number of episodes*

(S,A)	Σq(s,a)	N
1,0	-4	4
1,1	5	1
1,2	10	2
1,3	-2	2
2,0	-20	20
2,1	-10	1
2,2	4	24
..		

Algorithm 9: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer *num_episodes*
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $returns\_sum(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$
  end
end
$Q(s,a) \leftarrow returns\_sum(s,a)/N(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
return Q

Algorithm 9: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer *num_episodes*
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $returns\_sum(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
    for $t \leftarrow 0$ ***to*** $T-1$ **do**
        if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
            $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
            $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$
    end
end
$Q(s,a) \leftarrow returns\_sum(s,a)/N(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
return Q

Couple of optimizations

1. Instead of

Increment counter $N(s) \leftarrow N(s) + 1$

Increment total return $S(s) \leftarrow S(s) + G_t$

Value is estimated by mean return $V(s) = S(s)/N(s)$

   - Do this

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

2. Track a running mean

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

Incremental Mean

$$\mu_k = \frac{1}{k} \Sigma_{j=1,k} x_j$$

$$= \frac{1}{k} \left[x_k + \Sigma_{j=1,k-1} x_j \right]$$

$$= \frac{1}{k} \left(x_k + (k-1)\, \mu_{k-1} \right)$$

$$= \frac{1}{k} \left(x_k + k\, \mu_{k-1} - \mu_{k-1} \right)$$

$$= \mu_{k-1} + \frac{1}{k} \left(x_k - \mu_{k-1} \right)$$

Policy !

- We glossed over 2 points
- Where does the π come from ?
- How do we get to

- From a Q table ?

(S,A)	Σq(s,a)	N
1,0	-4	4
1,1	5	1
1,2	10	2
1,3	-2	2
2,0	-20	20
2,1	-10	1
2,2	4	24
..		

Algorithm 9: First-Visit MC Prediction (*for action values*)

Input: policy π, positive integer *num_episodes*
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize $returns\_sum(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $returns\_sum(S_t, A_t) \leftarrow returns\_sum(S_t, A_t) + G_t$
  end
end
$Q(s,a) \leftarrow returns\_sum(s,a)/N(s,a)$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
return Q

i.e. how does all this help us to make better actions ?

Policy Evaluation-Improvement Cycle

Update agent's behavior based on experience

Policy Evaluation

The q(s,a) tells us the utility of a policy

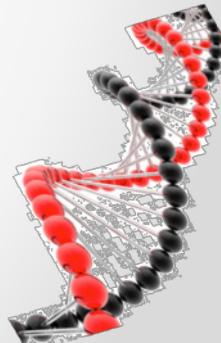
- Collect experience using π
- Create Q Table

- Construct better policy π' based on the Q values
- (optionally) Set $\pi = \pi'$
- Take action using π'

Policy Improvement

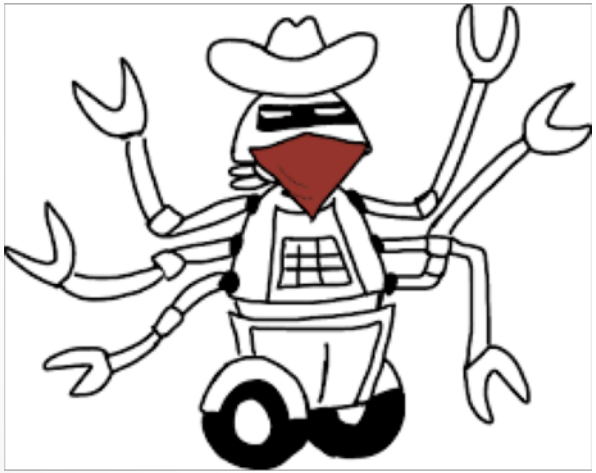
$$\pi_0 \mapsto v_{\pi 0} \mapsto \pi_1 \mapsto v_{\pi 1} \mapsto .. .. .. \mapsto \pi_* \mapsto v_*$$

- π – Behavior Policy (sample experience)
- π' – Target Policy (to optimize behavior)
- Most Common Policies
  - Equiprobable
  - Greedy
  - ε-Greedy (Greedy + Equiprobable)
  - Softmax distribution



TTD (Things To Do)

- Read & Explore
  - Multi-arm bandits
  - Contextual bandits



Policy Algorithms

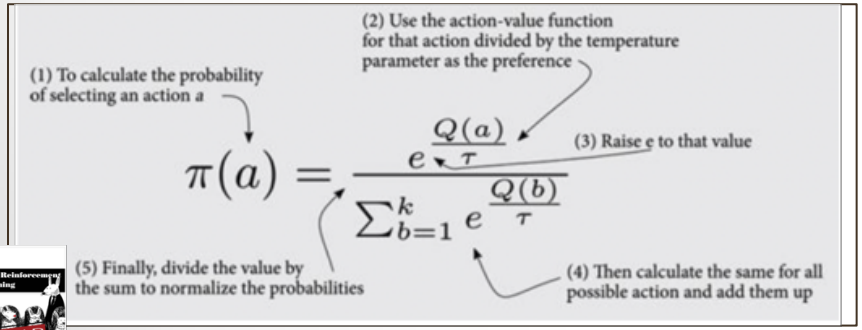
Equiprobable

- Random Uniform
- All actions equally possible
- *Good starting point, in an unknown environment*

Greedy

- Follow the action with the max value
- $\pi(a \mid s) = q_*(s,a) = \text{argmax}_{a'}(q(s,a'))$
- *Pure exploit*

Softmax



ε-Greedy

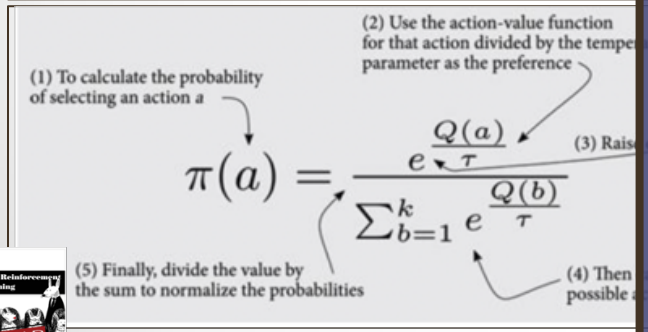
- With ε follow equiprobable
- With 1-ε follow greedy
- *Gives exploration*
- Decay ε as the gent learns more

Policy Algorithms

Equiprobable

- Random Uniform
- All actions equally possible
- *Good starting point, in an unknown environment*

Softmax



Softmax

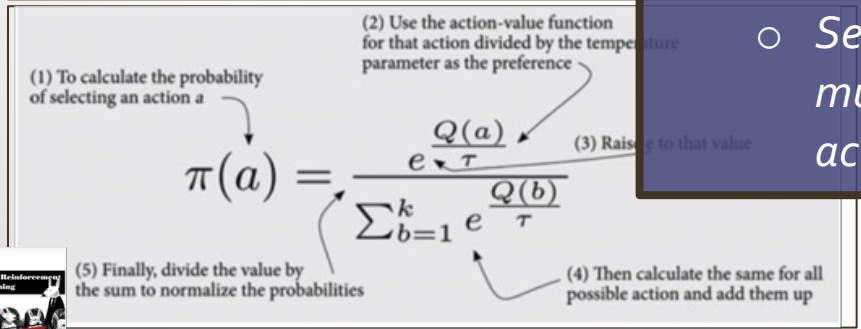
- *A weighted probability across actions, establishing a partial ordering*
  - *A weighted choice will establish the order during action selection e.g. random.choice(arr, p=Pr_array)*
- *τ = temperature, that scales the probability distribution of actions.*
  - *A high temperature will tend the probabilities to be very similar, whereas*
  - *A low temperature will exaggerate differences in probabilities between actions*
  - *Initially, when the values are inaccurate, we are Ok with small difference between the values i.e. uniform random*
  - *Decay τ such that as our values get better we want meaningful difference, yet have a small exploration possibility*

Policy Algorithms

Equiprobable

- Random Uniform
- All actions equally possible
- *Good starting point, in an unknown environment*

Softmax



Softmax

- *We all would have encountered softmax-based image classifier where there is a single correct classification to an image and there is no temporal association between each prediction.*
- *In our RL case, we want more control over how we make these updates.*
  - *First, we want to make small, smooth updates because we want to maintain some stochasticity in our action sampling to adequately explore the environment.*
  - *Secondly, we want to be able to weight how much we assign credit to each action for earlier actions.*

ε-Greedy

- With ε follow equiprobable
- With 1-ε follow greedy
- *Gives exploration*
- Decay ε as the gent learns more

Quiz

Equip

Random Uniform

Quiz:

1. *Which ε will always be greedy ?*
2. *Always not greedy ?*
3. *Is equiprobable a special case of ε-Greedy ? If so, what is ε for ε-Greedy Equiprobable ?*
4. *Which guarantees non greedy ?*
   ε=0 ? ε=0.25 ? ε=0.75 ? ε=0.99 ? ε=1 ?
5. *What is a good ε_{start} ? ε_{min} for decay ?*
6. *What is a good policy for Rock-Paper-Scissor ? How will it work ?*
7. *When will ε-Greedy stop learning ?*

$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_{a' \in \mathcal{A}(s)} Q(s, a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

value

- $\pi(a \mid s) = q_*(s,a) = \text{argmax}_{a'}(q(s,a'))$
- *Pure exploit*

ε-Greedy

- With ε follow equiprobable
- With 1-ε follow greedy
- *Gives exploration*
- Decay ε as the agent learns more

Quiz

Equipr

Random Uniform

$$\pi(a|s) \leftarrow \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|} & \text{if } a = \arg\max_{a' \in \mathcal{A}(s)} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

Quiz:

1. *Which ε will always be greedy ?* $\varepsilon = 0$
2. *Always not greedy ?* Trick Q. Not possible
3. *Is equiprobable a special case of ε-Greedy ? If so, what is ε for ε-Greedy Equiprobable ?* Yes. $\varepsilon = 1$
4. *Which guarantees non greedy ?* $\varepsilon > 0$
   ε=0 ? ε=0.25 ? ε=0.75 ? ε=0.99 ? ε=1 ?
5. *What is a good* ε_{start} *?* ε_{min} *for decay ?* ε = 1; decaying over say 10,000 steps to ε = 0.1
6. *What is a good policy for Rock-Paper-Scissor ? How will it work ?* ε = 1; equiprobable
7. *When will ε-Greedy stop learning ?* ε = 0; Greedy

value

- $\pi(a|s) = q_*(s,a) = \text{argmax}_{a'}(q(s,a'))$
- *Pure exploit*

ε-Greedy

- With ε follow equiprobable
- With 1-ε follow greedy
- Gives exploration
- Decay ε as the agent learns more

Algorithm 10: First-Visit GLIE MC Control

Input: positive integer *num_episodes*, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Initialize $N(s,a) = 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon$-greedy(Q)
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
  end
end
return π

GLIE - Greedy in the Limit with Infinite Exploration (GLIE) means that the behavior policies are required to become greedy (no exploration) in the limit of an online learning setting where the agent has gathered an infinite amount of experience

1. Greedy with limited Infinite Exploration
2. Construct a policy that is greedy w.r.t the Q table
3. Decay ε as the agent learns more
4. But sill maintain a small exploration

Algorithm 11: First-Visit Constant-α (GLIE) MC Control

Input: positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: policy π ($\approx \pi_*$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  $\pi \leftarrow \epsilon$-greedy(Q)
  Generate an episode $S_0, A_0, R_1, \ldots, S_T$ using π
  for $t \leftarrow 0$ ***to*** $T-1$ **do**
    if (S_t, A_t) *is a first visit (with return* G_t*)* **then**
      $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$
  end
end
return π

α Mean

$\alpha = 0$, never learns

$\alpha = 1$, never remembers !

α = small, larger history, learns slowly

α = large, focuses more on recent experience, but potentially difficult to converge

Hands On #2 – Monte Carlo

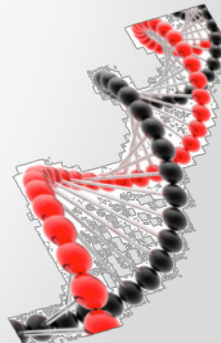
Goal:

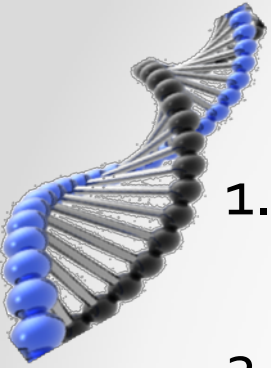
- Implement Monte Carlo
- Explore different policies solving OpenAI Gym : FrozenLake

Steps:

Notebook : Monte_Carlo_01.ipynb

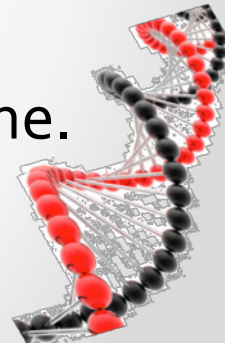
1. Program Monte Carlo – 1st Visit or every visit
2. Run the code and understand how it all comes together
3. How is the exploration policy implemented ?
4. How do we finally get the learned policy from the agent ?
5. Does it correspond to our deterministic policy ?

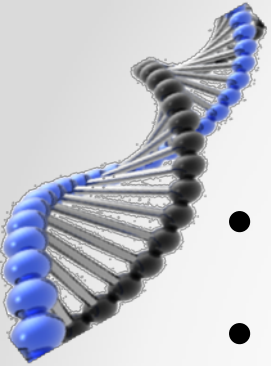




Points to Ponder

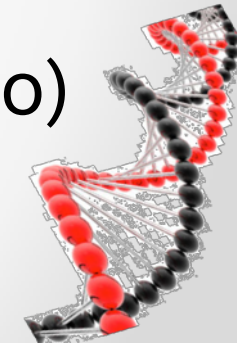
1. There are different ways of decaying ε
   - Or we could just keep ε =1, making it uniformly random; but this won't work. We will see some interesting results
2. GLIE – decay ε as decreasing function of episodes
3. Exponentially decay ε
   - The ε decay is to incorporate the learning & make the action more deterministic
4. We could use softmax & keep the policy constant i.e. weighted random.
   - As it learns, it's belief of the grid world changes and that shows up in the preferences
5. After 1,000,000 episodes, it usually wins all the time.
   - I had the ε_{min} = 0.1. Probably it should be reduced



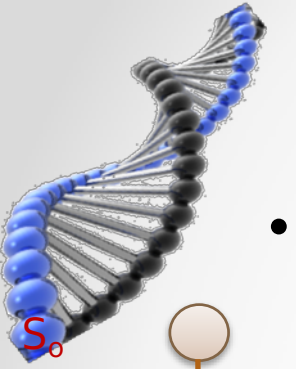


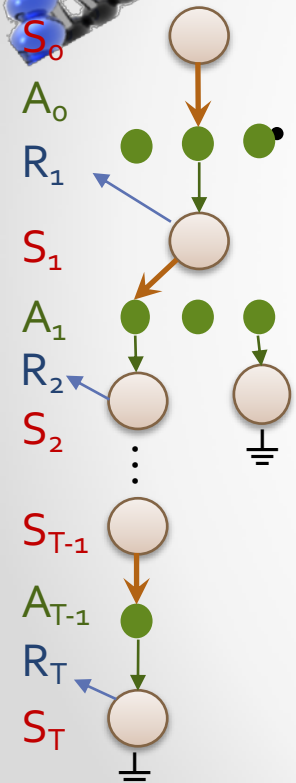
TD

- Temporal Difference Algorithm
- Motivation
  - Monte Carlo requires full episodes
  - MC has high Variance (but low Bias)
  - MC : Slow to converge
  - MC : Easy to understand
- So instead of waiting for a full episode, update the values after every step – TD(0) or every n steps – TD(ƛ)



TD vs. MC





- MC : $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

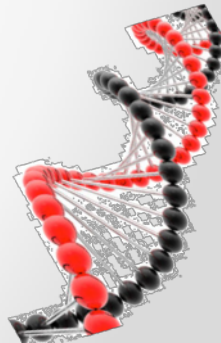
We observe from a full episode & update all

- TD : $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

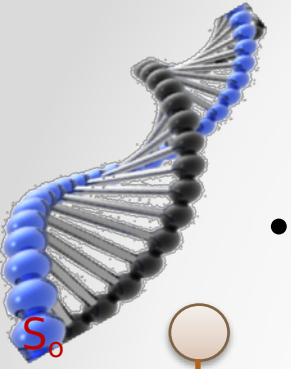
TD Target = Alternate Estimate | Current Estimate

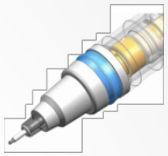
TD Error

  - *TD Target*
  - *TD Error*
  - *At every timestep, so we have R_{t+1} not G_t*
  - *We bootstrap $Q(S_{t+1}, A_{t+1})$, but as we take more steps and more episodes it all converges*
  - *TD learns from incomplete episodes by bootstrapping*
  - *TD lends itself to function approximation*
    - *And scale to large state space*
  - *Remember, we have to select S_{t+1} & A_{t+1}*
    - *That is policy & more later*



TD vs. MC





- MC : $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$

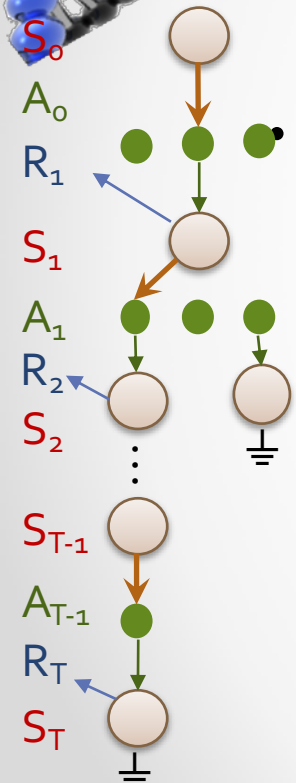
We observe from a full episode & update all

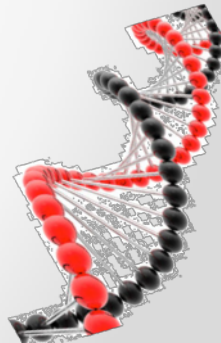
- TD : $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

TD Target = Alternate Estimate | Current Estimate

TD Error

  - *TD Target*
  - *TD Error*
  - *At every timestep, so we have R_{t+1} not G_t*
  - *We bootstrap $Q(S_{t+1}, A_{t+1})$, but as we take more steps and more episodes it all converges*
  - *TD learns from incomplete episodes by bootstrapping*
  - *TD lends itself to function approximation*
    - *And scale to large state space*
  - *Remember, we have to select S_{t+1} & A_{t+1}*
    - *That is policy & more later*





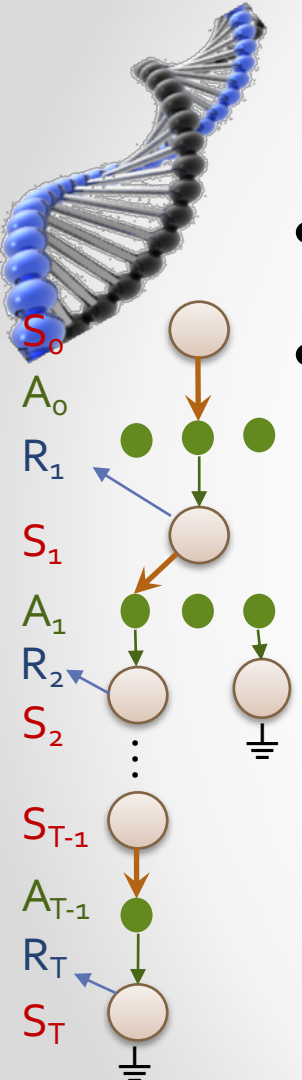
TD Intuition from Sutton Book/D.SLiver

Driving Home Example

State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

SARSA(0)

- Initialize $Q(s,a)\ \forall\ s \in \mathbb{S},\ a \in \mathbb{A}$
- π = ε-Greedy(Q) { $argmax_{a'}(q(s,a'))$ }



$S_0 A_0 R_1 S_1 A_1$ $R_2 S_2 A_2$ $\ldots A_{T-1} R_T S_T$

Update $Q(S_0,A_0)$
π = ε-Greedy(Q)

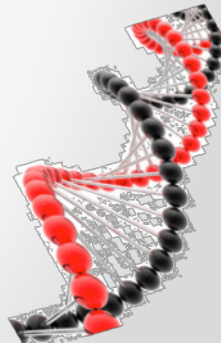
Update $Q(S_1,A_1)$
Update $Q(S_0,A_0)$
π = ε-Greedy(Q)

Update $Q(S_{T-1},A_{T-1})$
..
Update $Q(S_0,A_0)$
π = ε-Greedy(Q)

Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\,(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

A_{t+1} *selected using* ε*-Greedy(Q)*



Algorithm 13: Sarsa

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    Choose action A_0 using policy derived from Q (e.g., ϵ-greedy)
    $t \leftarrow 0$
    repeat
        Take action A_t and observe R_{t+1}, S_{t+1}
        Choose action A_{t+1} using policy derived from Q (e.g., ϵ-greedy)
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

Hands On #3 – SARSA

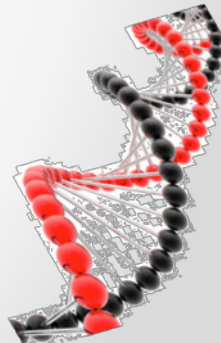
Goal:

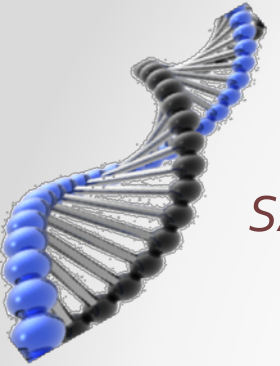
- Implement SARSA for Frozen lake

Steps:

1. Program SARSA
2. Plot Value

Skipping - we will work on $SARSA_{max}$ next





Q-Learning

SARSA Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

$Q(S_{t+1}, A_{t+1})$: *Selected using ε-Greedy(Q) for update*

A_{t+1} : *Selected using ε-Greedy(Q) for evaluation*

Q Learning Update Rule:

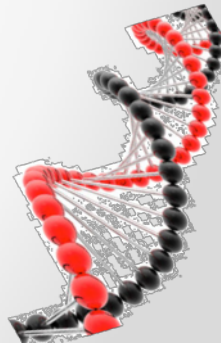
$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

$\max_{a \in \mathbb{A}} Q(S_{t+1}, a)$: *Selected using Greedy(Q) for update*

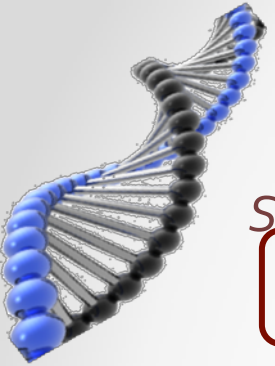
A_{t+1} : *Selected using ε-Greedy(Q) for evaluation*

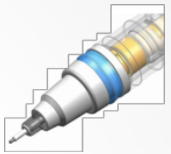
- SARSA : Behavior = Target ↦ On-Policy
- Q-Learning : Behavior ≠ Target ↦ Off-Policy

- *Learn Optimal policy while following exploratory policy*
- *Reuse data collected/Experience Replay/Prioritized Replay*



Q-Learning





SARSA Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

$Q(S_{t+1}, A_{t+1})$: *Selected using ε-Greedy(Q) for update*

A_{t+1} : *Selected using ε-Greedy(Q) for evaluation*

Q Learning Update Rule:

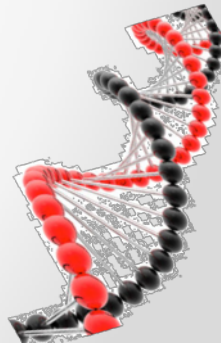
$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

$\max_{a \in \mathbb{A}} Q(S_{t+1}, a)$: *Selected using Greedy(Q) for update*

A_{t+1} : *Selected using ε-Greedy(Q) for evaluation*

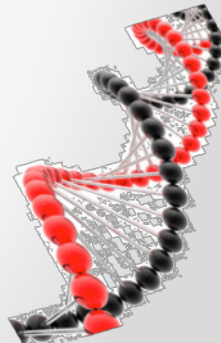
- SARSA : Behavior = Target ↦ On-Policy
- Q-Learning : Behavior ≠ Target ↦ Off-Policy

- *Learn Optimal policy while following exploratory policy*
- *Reuse data collected/Experience Replay/Prioritized Replay*



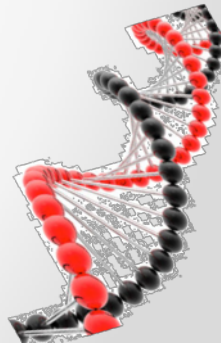
Quiz

- This Algorithm implements TD(0)
  - What is ?
- Q-Learning in terms of SARSA
  - What is ?
- This algorithm learns the best exploratory policy
  - What is ?
- This algorithm learns the optimum policy
  - What is ?



Quiz

- This Algorithm implements TD(0)
  - What is SARSA ?
- Q-Learning in terms of SARSA
  - What is $SARSA_{max}$?
- This algorithm learns the best exploratory policy
  - What is SARSA ?
- This algorithm learns the optimum policy
  - What is Q-Learning ?

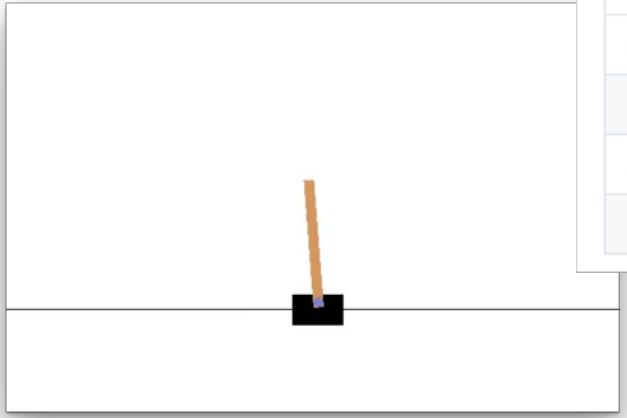


Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
    $\epsilon \leftarrow \epsilon_i$
    Observe S_0
    $t \leftarrow 0$
    repeat
        Choose action A_t using policy derived from Q (e.g., ϵ-greedy)
        Take action A_t and observe R_{t+1}, S_{t+1}
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
        $t \leftarrow t + 1$
    until S_t *is terminal*;
end
return Q

Introducing the CartPole

- *Balance an Inverted Pendulum by applying force on the cart*



Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Num	Action
0	Push cart to the left
1	Push cart to the right

Episode Termination

1. Pole Angle is more than ±12°
2. Cart Position is more than ±2.4 (center of the cart reaches the edge of the display)
3. Episode length is greater than 200

State: $\{x, \dot{x}, \theta, \dot{\theta}\}$

2 Discrete Actions: Apply force to cart left/right

Rewards: +1 for each step

Solved after 211 episodes. Best 100-episode average reward was 195.27 ± 1.53. (CartPole-v0 is considered "solved" when the agent obtains an average reward of at least 195.0 over 100 consecutive episodes.)

211	500	✓	83s	↓	
Episodes to solve	Total episodes	Solved	Time to solve	Download	Tweet

https://gym.openai.com/envs/CartPole-v0/

https://github.com/openai/gym/wiki/CartPole-v0

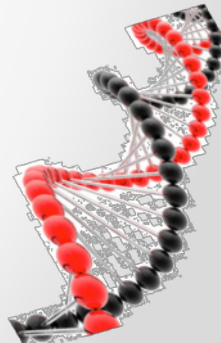
Hands On #4 – Q Learning

Goal:

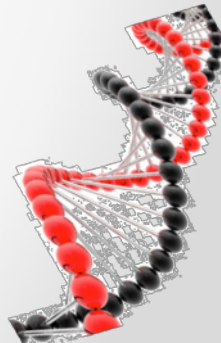
- Introduce the CartPole Environment
  - *More complex, Continuous*
- Implement Q-Learning for digitized CartPole
  - *Later we will use function approximation & remove the requirement for digitization*

Steps:

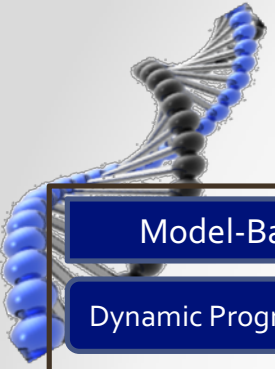
1. Notebook : Q_Learning_01.ipynb
2. Get familiar with Cartpole environment
   - np.linspace() and np.digitize() for state space aggregation
3. Program Q Learning – 4 To Dos
4. Track & Plot Metrics to solve Cart Pole



Down the DRL-Land



RL Algorithm Taxonomy

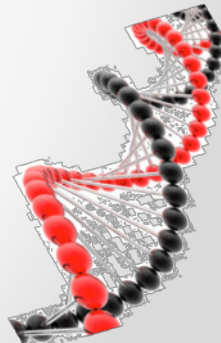


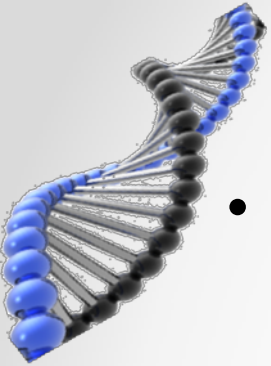
- Model-Based
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration
- Model-Free
  - Monte Carlo
  - TD Learning
  - Value Methods
    - SARSA
    - Q-Learning
    - Deep Q-Learning
      - Fixed Q Targets
      - Experience replay
      - Double DQN
      - Prioritized replay
      - Dueling DQN
      - *Increase robustness*
      - *Reduce bias/variance*
      - *Sample efficiency*
  - Policy Search
    - Hill Climbing
      - Steepest Hill Climbing
        - Cross-entropy
        - Evolution Strategies
      - Random Restart
      - Simulated Annealing
      - Adaptive noise scaling
  - Policy Gradients
    - REINFORCE
    - TRPO
    - PPO
  - Actor-Critic
  - DDPG

RL ⟷ DRL

Non-linear Function Approximation

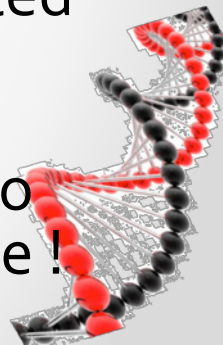
- Why Policy Based ?
  - Simplicity
  - Stochastic Action Space
    - Rock-Paper-scissor
  - Continuous
    - Can output a # without exploring values in a continuous space
  - Aliased States



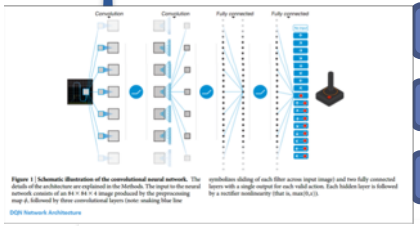


Deep reinforcement Learning

- We have interesting algorithms, but they won't scale easily for large state spaces or for continuous state spaces
  - For example, GO has possibly 10^{170} states, more than the number of atoms in the observable universe (10^{80})
  - We want something that can sample and generalize; *that gets better with more data*
- Fortunately, Reinforcement Learning research has progressed so far that machines are able to win over Go world champions - 10 years ahead of the predicted timeframe !
  - Methodological breakthroughs in Deep Learning !!
- The latest DOTA 5 uses 120,000 CPUs & 256 GPUs to win over human players in a complex strategic game !



DeepRL Schematic



Value

Action

Policy

Agent

State

Reward

Actions

Environment

- We will first look at the simple mechanism
  - Replace our Q-Table with a function approximation ie a Neural Network, to make it work on the top of DL structures
  - Then we can get values, actions or even policies
- Then we will look at extensions
- And finally, we will see how these ideas scale to games with video frames, games like Go with huge state spaces and complex semantics

DQN Intuition

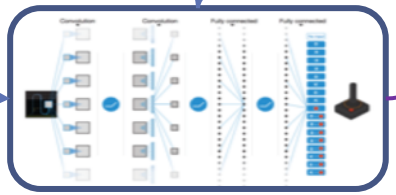
Q Learning Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\,(R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

Deep Q Learning Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\,(R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

Pl. note the direction of the arrows



DQN Intuition

Deep Q Learning Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

$$Y_t = R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t)$$

Action to get Max next state value = Target

Gradient Descent

$$\theta_{t+1} = \theta_t + \alpha\left(Y_t - Q(S_t, A_t; \theta_t)\right) \nabla_\theta Q(S_t, A_t; \theta_t)$$

DQN Parameter changed

TD Error

Gradient

DQN Intuition

Deep Q Learning Update Rule:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

Constantly shifting values with the same parameters θ_t

".. this approach can build large errors at different places in the state-action space .."

$$Y_t = R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t)$$

Action to get Max next state value = Target

Gradient Descent

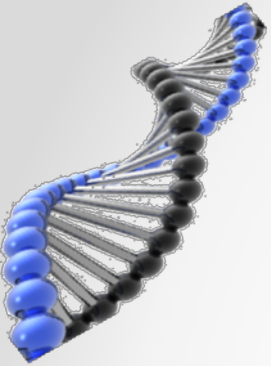
$$\theta_{t+1} = \theta_t + \alpha\left(Y_t - Q(S_t, A_t; \theta_t)\right) \nabla_\theta Q(S_t, A_t; \theta_t)$$

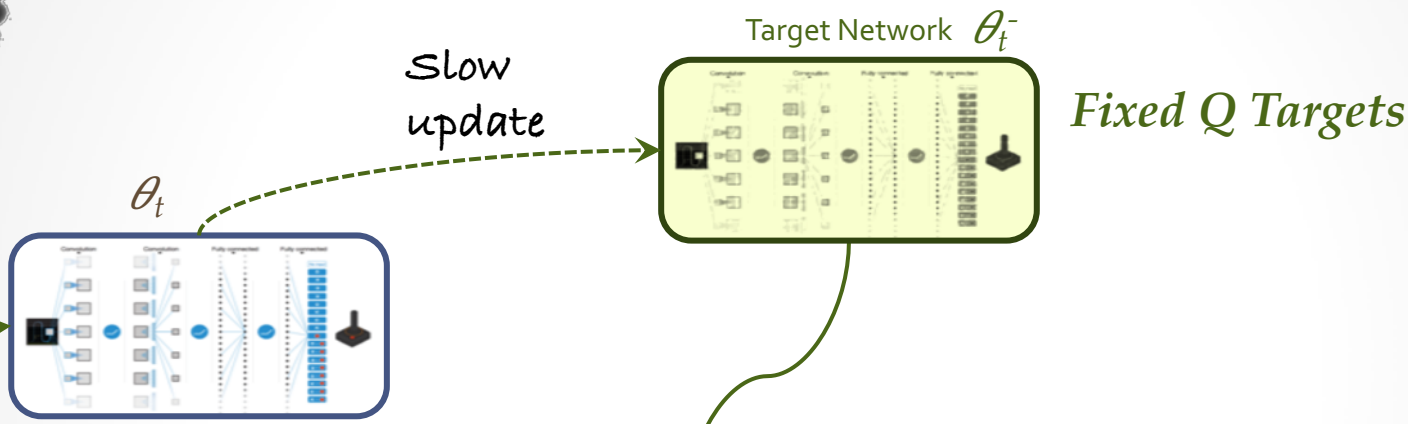
DQN Parameter changed

TD Error

Gradient

DQN Intuition





$$Y_t^Q = R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t^-)$$

Action to get Max next state value = Target

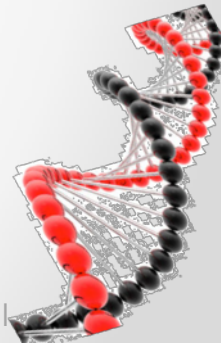
Gradient Descent

$$\theta_{t+1} = \theta_t + \alpha (Y_t^Q - Q(S_t, A_t; \theta_t)) \nabla_\theta Q(S_t, A_t; \theta_t)$$

DQN Parameter changed

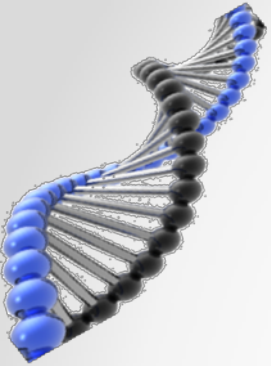
TD Error

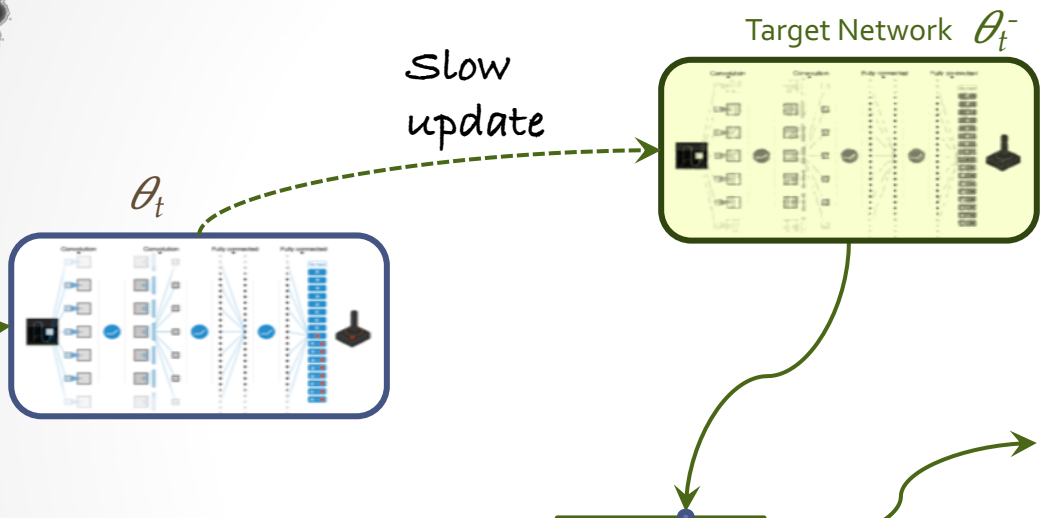
Gradient



https://paulispace.com/inference/2017/08/29/dqn.html

DQN Intuition





Fixed Q Targets

https://arxiv.org/abs/1509.06461

Over estimation (upward) Bias: Over optimism w.r.t estimation errors - Selection & Evaluation with the same network θ_t^-
Solution : Double DQN

$$Y_t^Q = R_{t+1} + \gamma \max_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t^-)$$

Action to get Max next state value = Target

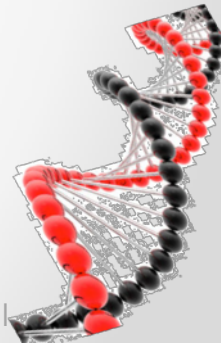
Gradient Descent

$$\theta_{t+1} = \theta_t + \alpha (Y_t^Q - Q(S_t, A_t; \theta_t)) \nabla_\theta Q(S_t, A_t; \theta_t)$$

DQN Parameter changed

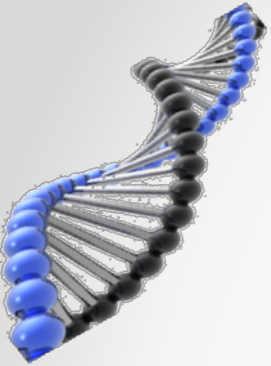
TD Error

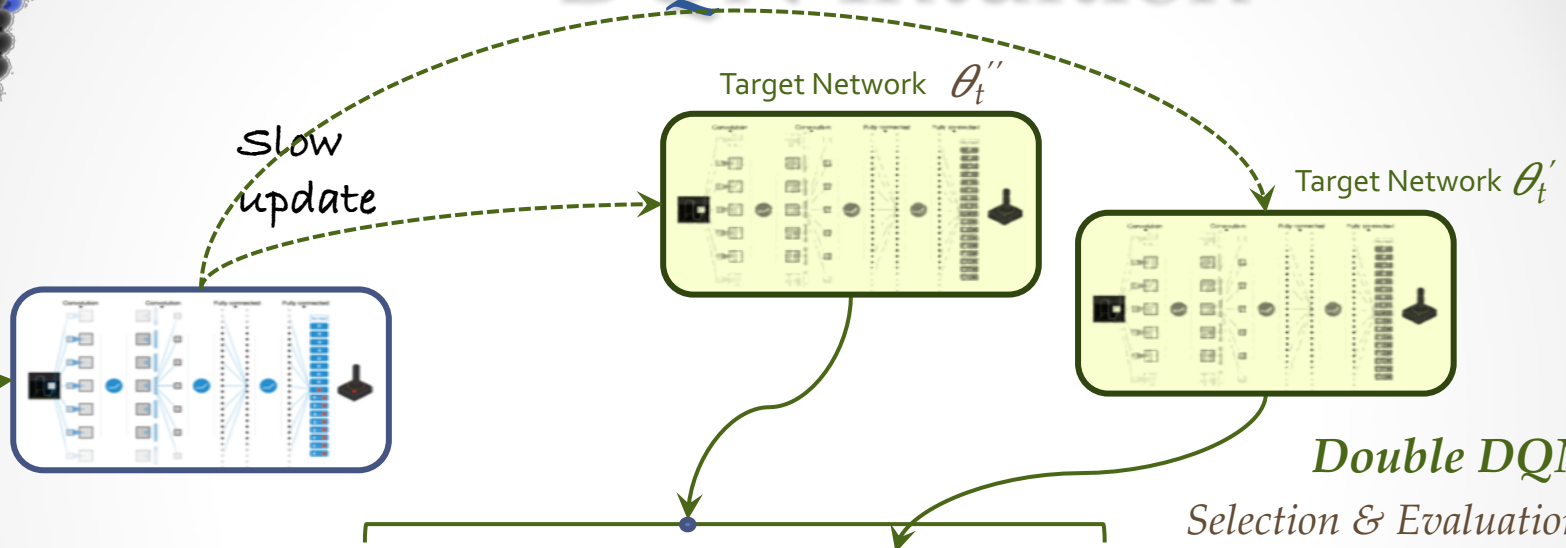
Gradient



https://paulispace.com/inference/2017/08/29/dqn.html

DQN Intuition





Double DQN

Selection & Evaluation in two separate target networks, θ_t' and θ_t'' which alternate their roles

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \underbrace{argmax_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t')}_{\text{Action to get Max next state value = Target}}; \theta_t'')$$

Gradient Descent

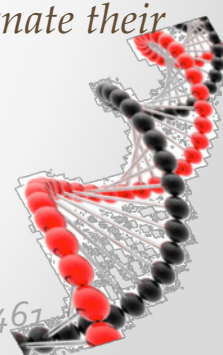
$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t)) \nabla_\theta Q(S_t, A_t; \theta_t)$$

DQN Parameter changed

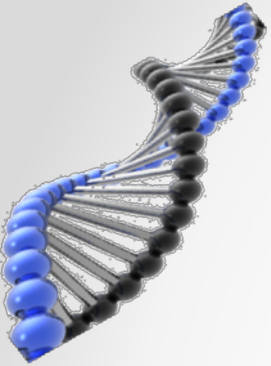
TD Error

Gradient

Double DQN paper https://arxiv.org/abs/1509.06461

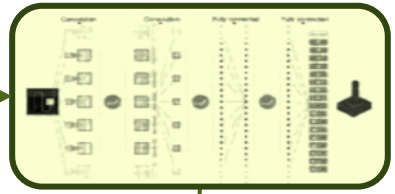


DQN Intuition

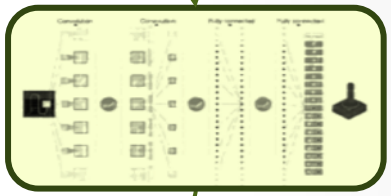


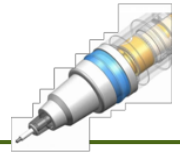
Slow update

Target Network θ_t''



Target Network θ_t'





$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \underbrace{argmax_{a \in \mathbb{A}} Q(S_{t+1}, a; \theta_t')}; \theta_t'')$$

Action to get Max next state value = Target

Double DQN

Selection & Evaluation in two separate target networks, θ_t' and θ_t'' which alternate their roles

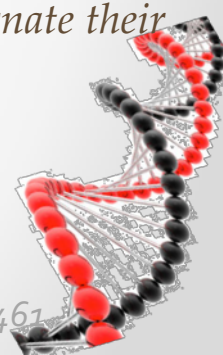
Gradient Descent

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t)) \nabla_\theta Q(S_t, A_t; \theta_t)$$

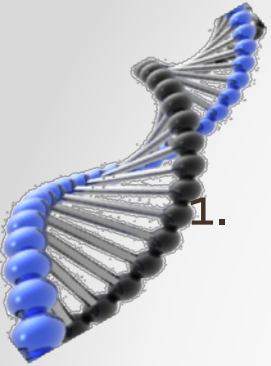
DQN Parameter changed

TD Error

Gradient



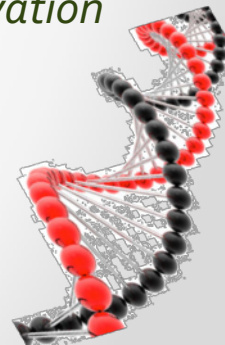
Double DQN paper https://arxiv.org/abs/1509.06461



Experience Replay

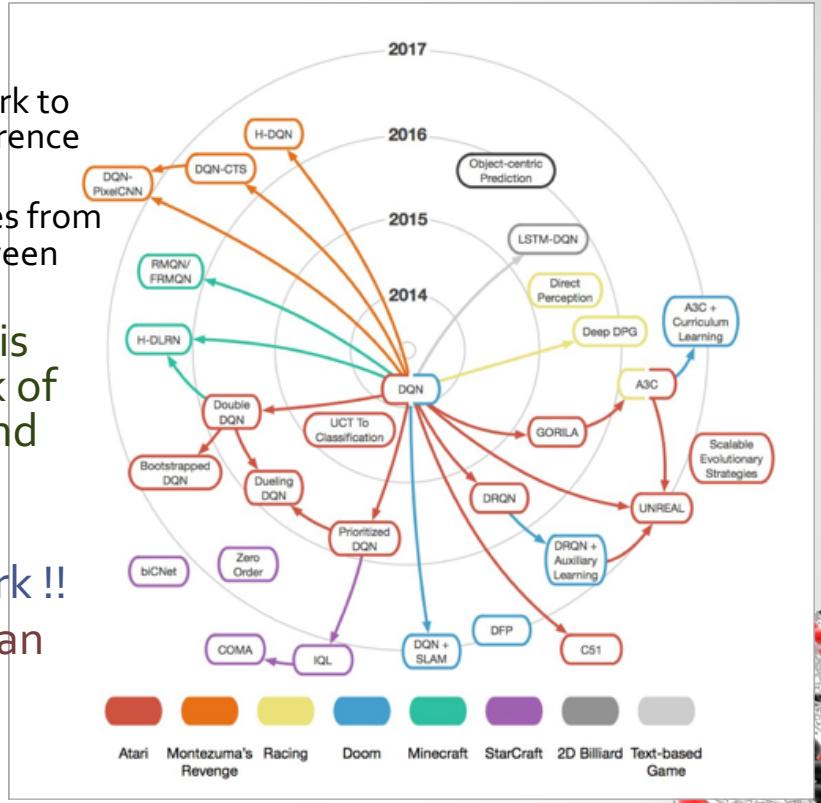
1. Most optimization algorithms assume that the samples are independently and identically distributed(i.i.d).
    - Obviously, when the samples are generated from exploring sequentially in an environment this assumption no longer holds
2. To make efficient use of hardware optimizations, it is essential to learn in minibatches, rather than online

- Solution : Experience Replay
    - Buffer the (s_t, a_t, r_t, s_{t+1}) tuple in a "*finite-sized circular cache*"
    - Sample a mini batch of appropriate size & train !
    - "*randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution*"
    - Decorrelate & Data efficiency
    - Prevent Catastrophic Forgetting



A word on the evolution

- DQN has two main innovations:
  1. The network is trained with a target Q network to give consistent targets during temporal difference backups
  2. The network is trained off-policy with samples from a replay buffer to minimize correlations between samples
- It looks a lot easier when we look from this side, but this is the result of a good chunk of research by David Silver, Volod, Dennis and team …
- Am sure they spent tons of hours on mechanisms and schemes that didn't work !!
- And, I think, this work paved the way for an explosion of research and pragmatic applications based on Reinforcement Learning



https://arxiv.org/abs/1708.07902

http://fzruniverse.life/2018/03/24/Modular-Architecture-for-Implementing-RL-Agent/

Deep Q Networks

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

- We tested this agent on the challenging domain of classic Atari 2600 games
- We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to achieve a professional human level across a set of 49 games
- *Same algorithm, network architecture and hyperparameters*

https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
https://eecs.wsu.edu/~taylorm/17_580/Yunshu_DQNpresentation_10102016.pdf

Input: the pixels and the game score
Output: Q action value function (from which we obtain a policy and select actions)

initialize replay memory D
initialize action-value function Q with random weight $\boldsymbol{\theta}$
initialize target action-value function $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
for *episode* = *1 to* M **do**
  initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for t = *1 to* T **do**

Sample

following ϵ-greedy policy, select $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_a Q(\phi(s_t), a; \boldsymbol{\theta}) & \text{otherwise} \end{cases}$
execute action a_t in emulator and observe reward r_t and image x_{t+1}
set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Learn

// experience replay
sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \boldsymbol{\theta}^-) & \text{otherwise} \end{cases}$
perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \boldsymbol{\theta}))^2$ w.r.t. the network parameter $\boldsymbol{\theta}$
// periodic update of target network
in every C steps, reset $\hat{Q} = Q$, i.e., set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$

  end
end

https://arxiv.org/pdf/1810.06339.pdf
Minh Paper : https://arxiv.org/abs/1602.01783

Algorithm 8: Deep Q-Nework (DQN), adapted from Mnih et al. (2015)

A detour – to DL machinery with *pytorch*

Introducing PYTORCH

*Adam Paszke, Sam Gross, **Soumith Chintala**, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Christian Sarofeen, Alban Desmaison, Andreas Kopf, Edward Yang, Zach Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy*

- *Origin* https://classroom.udacity.com/courses/ud188/
  - *Started early, but matured at FAIR –*
  - *Initially Soumith, Adam Paszke & Sam Gross*
  - *slowly opened for "power users"*
  - *Adopted by the "fun, long tail" (research) people, not the enterprise crowd … so the core design is (more) research-friendly … pythonic debugging capabilities*
  - *But also is production-ready .. Function annotation … IR export to C++ run-time or any other VM and so forth*
  - *Hybrid frontend – parts of the model compiled – like the core RESNET, but leave the other parts not compiled*
  - *JIT compiler – long term goal is to make the model faster leveraging hardware, optimizing fusing operations and so forth*

- We exist!
  - Launched Jan 2017
- An imperative programming model
- Eager evaluation (unlike Dynet which is lazy)
- Simplicity as a philosophy

- *"Large part of pytorch live in C++*
- *Whatever user facing is in python …"*
- *Giving it flexibility ("hackable") and performance …*

PYTORCH

With ❤ from

http://pytorch.org

Released Jan 2017

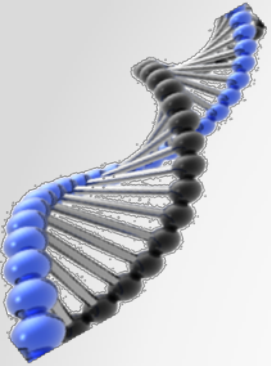
500,000+ downloads

2700+ community repos

17,200+ user posts

351 contributors

facebook · NVIDIA · salesforce · ParisTech · Carnegie Mellon University · Digital Reasoning · Stanford University · OXFORD · NYU · Inria · ENS · EPFL · Berkeley · UBER



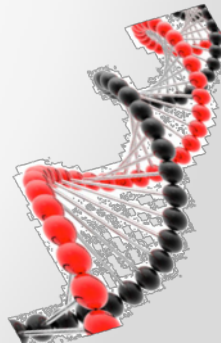
Code Review – pytorch DL Machinery

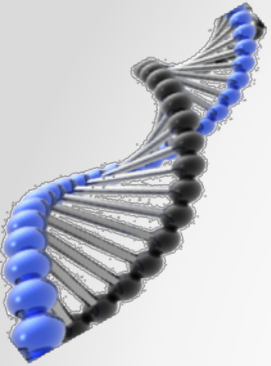
Goal:

- Get familiar with a basic pytorch neural network

Steps:

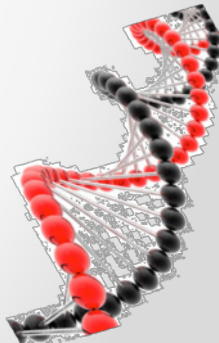
1. Define Network, loss & optimizer
2. Get Dataset - MNIST
3. Construct minibatch dataloader
4. Forward pass
5. Calculate error/loss
6. Do backprop
7. Update weights using Gradient Descent
8. Train & Test





pytorch DL Machinery

- Much simpler and easy to create a network
- Intuitive pythonic interface
- Things to keep in mind
  - Use with torch.no_grad() for inferencing
    - Disables autograd which is needed only for backpropaga
  - Use model.eval() for testing
    - Disables dropout
    - Can enable dropout by model.train()
  - Do optimizer.zero_grad() to clear the gradient
    - To reset the accumulation of gradient – very important !!
  - Enabling GPU is easy and idiomatic
    - device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    - ..
    - model.to(device) <- Move model to GPU
    - ..
    - inputs, labels = inputs.to(device), labels.to(device) <- Move data to GPU
    - ..



Hands On #5 – Deep Q Learning

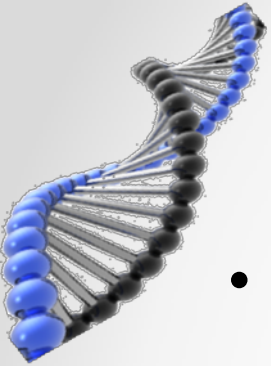
Goal:

- Implement Deep Q-Learning for CartPole
  - No digitization
  - Handle continuous state

Steps:

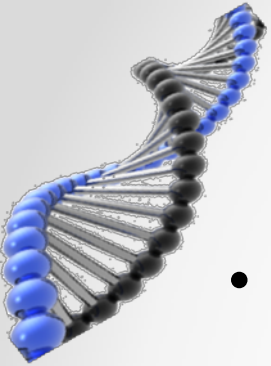
1. Notebook : DQN_01.ipynb
2. Understand the Deep Q Learning Code
3. Understand DNN implementation in pytorch
4. Metrics to solve Cart Pole
5. Plot Values
6. Answer the quiz in next page

"Confidence is recursive ! You have to come thru a few times to be certain and confident that you can beat it !"



Quiz

- What is the DQN DNN topology ?
  -
- What is the input size ? Output size ?
  -
- How does the output translate to an action ?
  -
- How is the target network updated ?
  -
- Did it solve the environment ?
  -
- Can it be tuned to solve with lesser number of steps ?
  -



Quiz

- What is the DQN DNN topology ?
  - FC16-FC2
- What is the input size ? Output size ?
  - Input = 4 (state), output = 2 (actions) – *notice that we get all actions in one shot!*
- How does the output translate to an action ?
  - np.argmax(action_values,..)
- How is the target network updated ?
  - Soft_update with τ=0.001
- Did it solve the environment ?
  - Yep, in ~ 3300 steps, with a particular set of hyperparameters
- Can it be tuned to solve with lesser number of steps ?
  - May be, probably, approximately definitely !

Deep Q Networks

LETTER

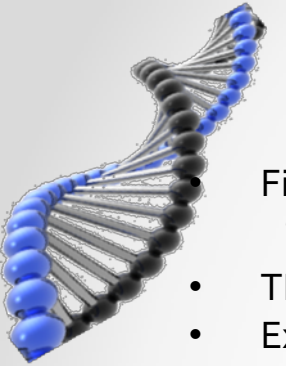
doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

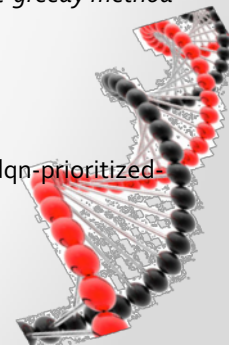
- Hyperparameters
  - Minibatch size = 32
  - Replay buffer = 1 Million
  - Agent history = 4 frames
  - Target Network update frequency = 10,000
  - ε = 1 to 0.1 ; linearly annealed in 1 million steps

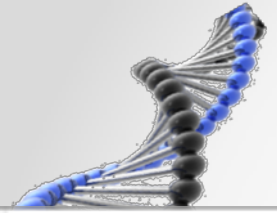
https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
https://eecs.wsu.edu/~taylorm/17_580/Yunshu_DQNpresentation_10102016.pdf



Deep Q Learning - Extensions

- Fixed Q Targets – improve stability with Target Networks
  - "*an iterative update that adjusts the action-values (Q) towards target values that are only periodically updated, thereby reducing correlations with the target*"
- TD Error clipping to [-1,1]
- Experience replay
  - "*randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution* "
  - Decorrelate & Data efficiency
  - Prevent Catastrophic Forgetting
- Prioritized replay
  - Pay more attention to interesting experience
  - Replaying more often transitions from which there is more to learn
- Double DQN
- Dueling DQN
- Noisy nets - add parametric noises to network weights (we will see this later)
  - Without ϵ-greedy - "*Adding noise to a deep network is often equivalent or better than adding noise to an action like that in the ϵ-greedy method*"
- Rainbow
  - To address challenges robustness & stability
  - Increase robustness, reduce bias/variance, sample efficiency
  - The Integrated Agent – Rainbow !! https://arxiv.org/abs/1710.02298
    - Another interesting paper : https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682
  - Homework : Can Rainbow solve Pong ? Breakout ?



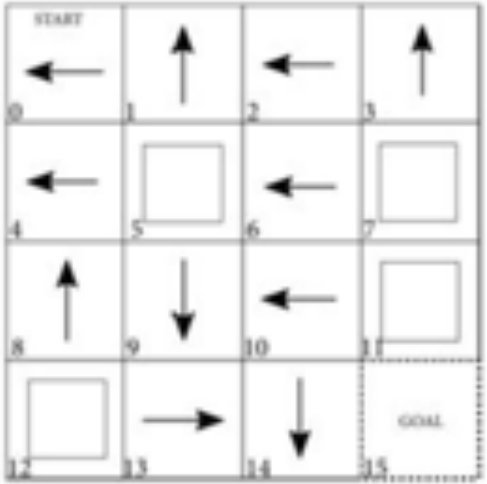


Hands On #6 – Skating slippery Frozen Lake with DQN

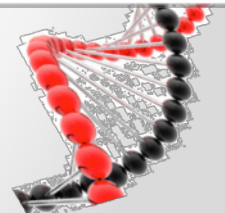
Yes, this is Frozen Lake's optimal policy!

And I wish you at least question the validity of it. How is it possible that in state 14 the optimal action is to go DOWN? Shouldn't it be RIGHT?

Since the Frozen Lake environment stochasticity is so large, it makes it a great example to illustrate the difficulty of sequential decision making under uncertainty of action effects. The fact is, the RIGHT action would send us to state 10, 14 or 15 with equal probability. While DOWN would send us to state 13, 14 or 15. If you think about it, state 13 is preferred to 10 because we can only get to state 15 safely going LEFT in 10, DOWN in 9, RIGHT in 13 and DOWN in 14. On the other hand, going DOWN in state 14 keeps us only a few steps away from 15 regardless of where the environment sends us.

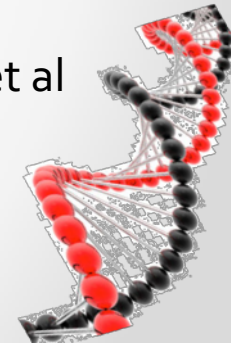


Homework : See if DQN-Rainbow finds this policy
Compare with DDPG



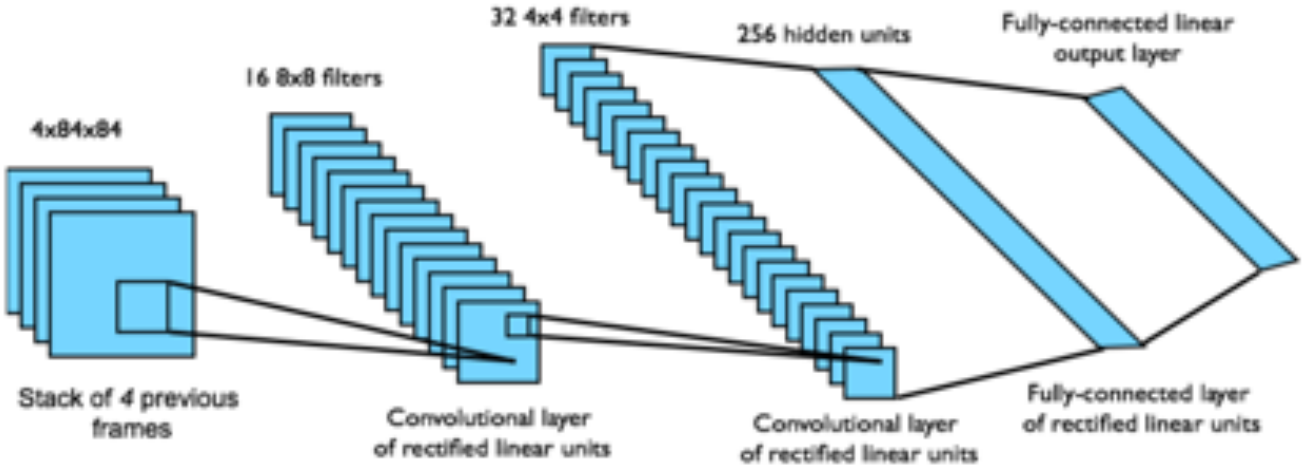
References

1. Human Level Control Through Deep Reinforcement Learning, Volodymyr Mnih et al https://deepmind.com/research/publications/human-level-control-through-deep-reinforcement-learning/
   - *We demonstrate that the deep Q-network agent, receiving only the pixels and the game score as inputs, was able to surpass the performance of all previous algorithms and achieve a level comparable to that of a professional human games tester across a set of 49 games, using the same algorithm, network architecture and hyperparameters*
2. Neural Fitted Q Iteration by Martin Riedmiller et al
   - https://qiita.com/Rowing0914/items/3d1b9a6ad4fe0a53cf5b
3. Deep Reinforcement Learning with Double Q-learning by Hado van Hasselt, Arthur Guez, David Silver
   - https://arxiv.org/abs/1509.06461
4. Prioritized Experience Replay by Tom Schaul, John Quan, Ioannis Antonoglou, David Silver
   - https://arxiv.org/abs/1511.05952
5. A Brief Survey of Deep Reinforcement Learning by Kai Arulkumaran et al
   - https://arxiv.org/abs/1708.05866

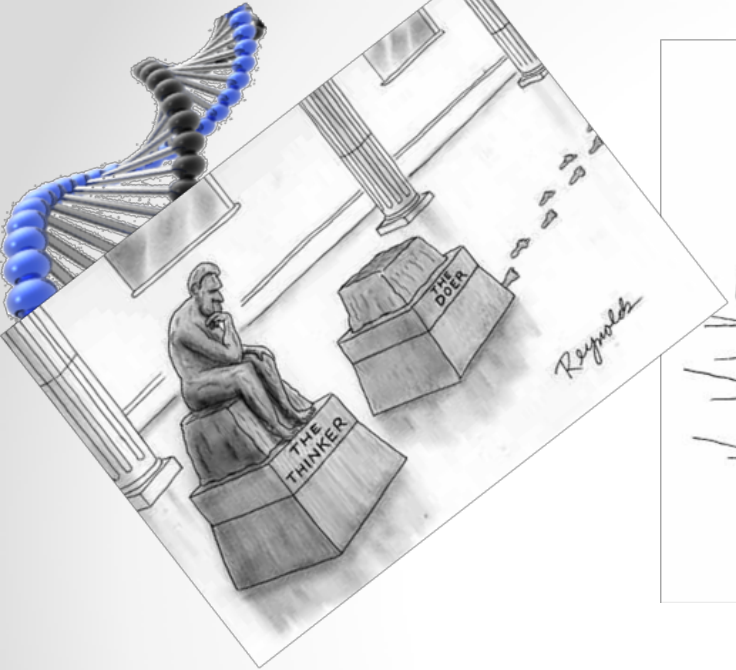


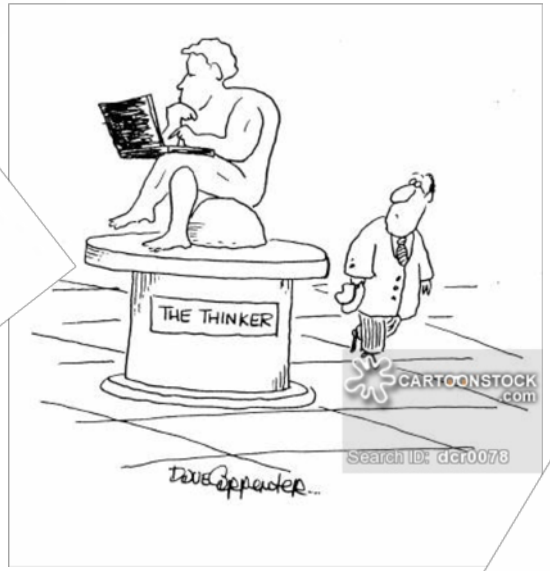
DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

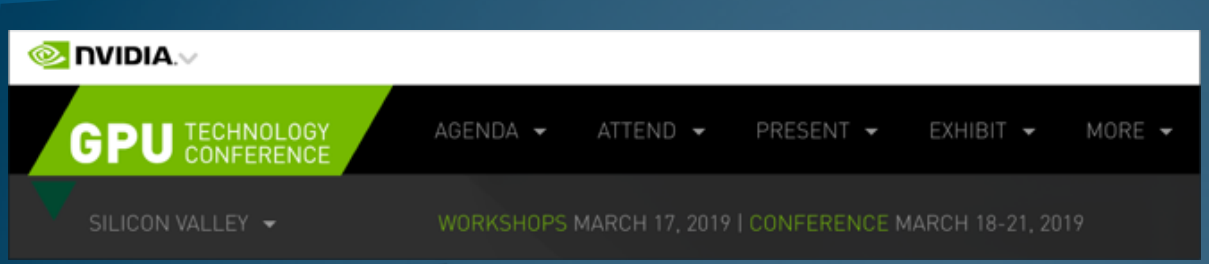


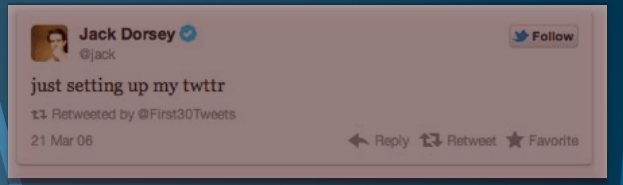
Network architecture and hyperparameters fixed across all games





Pause & Reflect



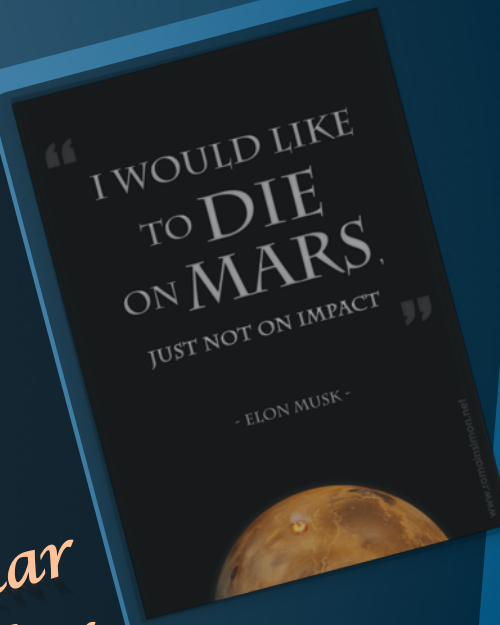


"A good traveler has no fixed plans and is not intent on arriving." – Lao Tzu

Demystifying Deep Reinforcement Learning – Part 2 : Policy Gradients

All observations are my own & do not represent my employer's or anyone else's

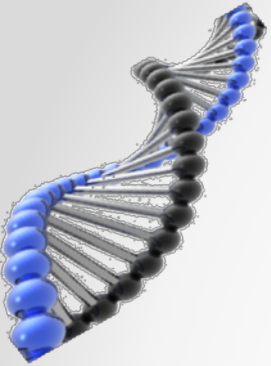
krishna sankar
@ksankar



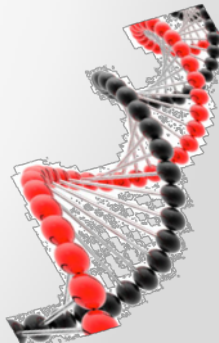
Agenda

- Framing
- Reinforcement learning – A Formal Expedition
- Algorithms & Programming
  - Monte Carlo, TD, SARSA, Q-Learning & DQN
- The world of AlphaGo
- Policy Gradients – REINFORCE, A3C & DDPG
- Conclusion & Forward
- Emergency Time Reserve – Overflow !!

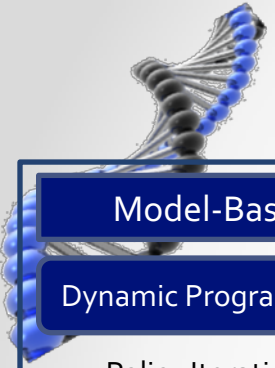
https://github.com/xsankar/GTC_2019_DRL_LAB



5. Policy Gradients



RL Algorithm Taxonomy

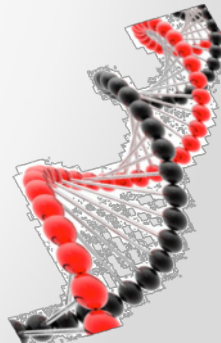


- Model-Based
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration
- Model-Free
  - Monte Carlo
  - TD Learning
  - Value Methods
    - SARSA
    - Q-Learning
    - Deep Q-Learning
      - Fixed Q Targets
      - Experience replay
      - Double DQN
      - Prioritized replay
      - Dueling DQN
      - *Increase robustness*
      - *Reduce bias/variance*
      - *Sample efficiency*
  - Policy Search
    - Hill Climbing
    - Steepest Hill Climbing
      - Cross-entropy
      - Evolution Strategies
    - Random Restart
    - Simulated Annealing
    - Adaptive noise scaling
  - Policy Gradients
    - *Natural Policy Gradients*
    - REINFORCE
    - TRPO
    - Actor-Critic
    - PPO
    - DDPG

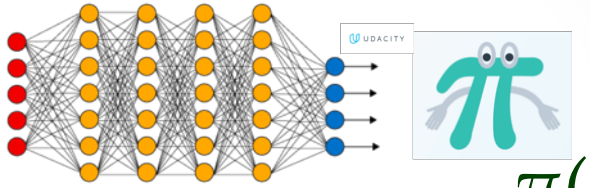
RL ⟷ DRL

Non-linear Function Approximation

- Why Policy Based ?
  - Simplicity
  - Stochastic Action Space
    - Rock-Paper-scissor
  - Continuous
    - Can output a # without exploring values in a continuous space
  - Aliased States



Policy in Policy Gradient



$\pi(a|s;\theta)$

Agent

$\pi(a|s;\theta) = \mathbb{P}\{A_t = a \mid S_t = s, \theta_t = \theta\}$

State

Reward

Parameterized Policy

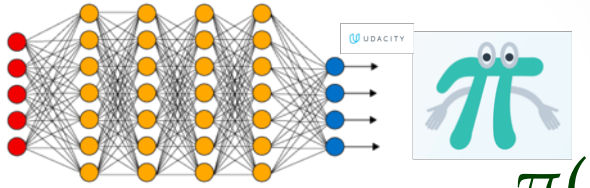
Action

Discovered via Gradient Ascent

$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$

Environment

Policy in Policy Gradient



$\pi(a|s; \theta)$

Agent

$\pi(a|s; \theta) = \mathbb{P}\{A_t = a \mid S_t = s, \theta_t = \theta\}$

State

Reward

Parameterized Policy

Action

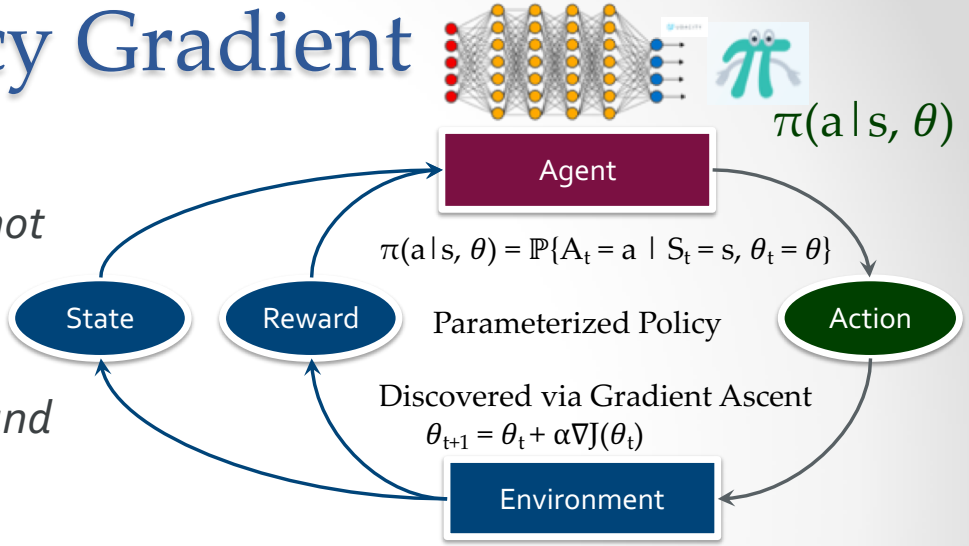
Discovered via Gradient Ascent

$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$

Environment

Policy in Policy Gradient

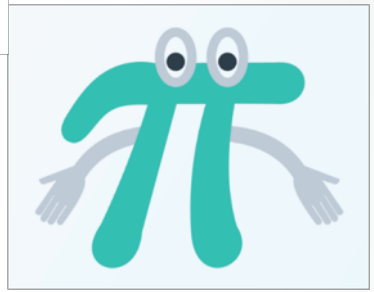
- *All methods that follow this general schema we call policy gradient methods, whether or not they also learn an approximate value function.*
- *Methods that learn approximations to both policy and value functions are often called actor–critic methods*



- Stochastic Policy $\pi(a|s; \theta) = \mathbb{P}\{a|s; \theta\}$
  - e.g. Softmax
  - Probability - A degree of belief (i.e. Credence), a subjective assessment of how much I can predict an event given the knowledge I currently possess. Bayesian prior-ish !
- Deterministic Policy $\pi : s \mapsto a$
  - Our 1st exercise was a symbolic deterministic policy injecting domain knowledge

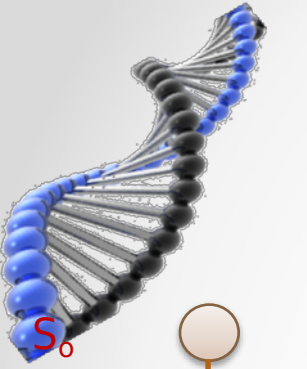
Policy Gradients

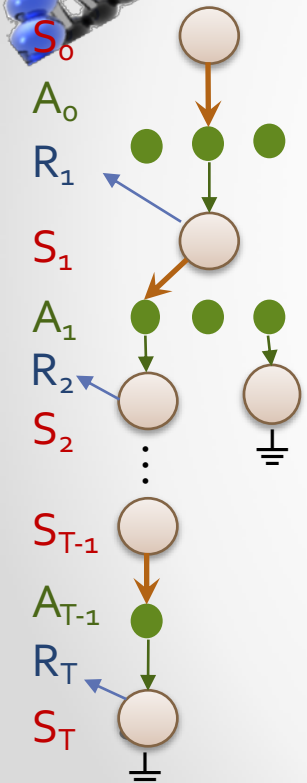
UDACITY



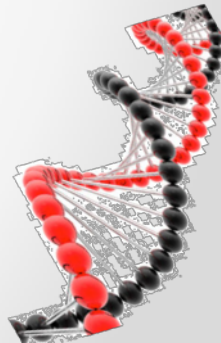
- Why Policy Gradients
  - Compact Representation
  - Better Convergence properties
  - Can handle Continuous space
  - Better for High Dimensional space
  - Stochastic Policies (rock-paper-scissor !)
  - Aliased States (Using uniform Stochastic Policy)
  - Simple
    - Enables the agent to devise robust strategies for navigating the environment directly (e.g. Left or Right)
- Challenges
  - Susceptible to converging to local minima
  - High Variance

Policy Gradients - Nomenclature

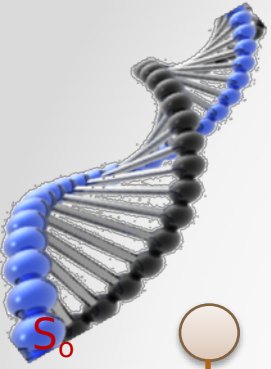


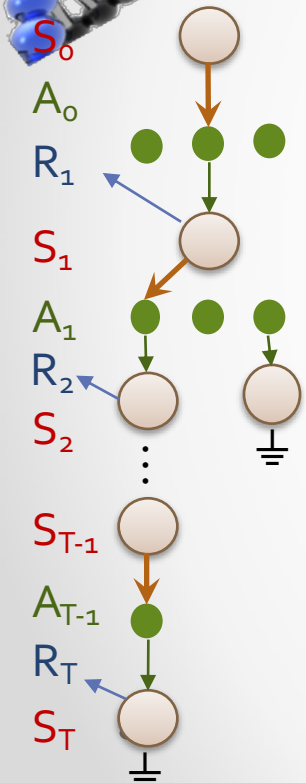


- Trajectory $\boldsymbol{\tau}$ over a horizon H
  - $S_0 A_0 R_1 S_1 A_1 R_2 S_2 \ \ldots \ A_{H-1} R_H S_H$
- Reward $R(\boldsymbol{\tau})$
  - $R(\boldsymbol{\tau}) = \Sigma_{t=0 \text{ to } H} R(s_t, a_t)$
- Trajectory can be an episode (H=T), but doesn't dictate it

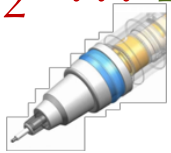


Policy Gradients - Nomenclature



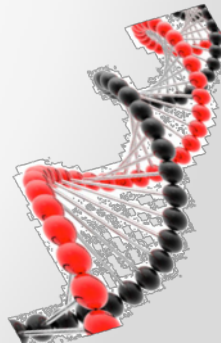


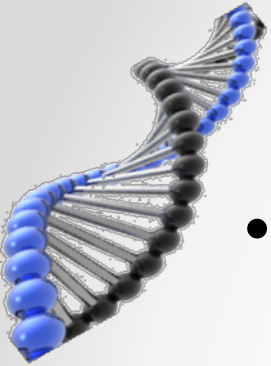
- Trajectory τ over a horizon H
  - $S_0 A_0 R_1 S_1 A_1 R_2 S_2 \ \ldots \ A_{H-1} R_H S_H$
- Reward $R(\tau)$
  - $R(\tau) = \sum_{t=0 \text{ to } H} R(s_t, a_t)$



This the G_t we are used to.
Some literature calls it R_τ

- Trajectory can be an episode (H=T), but doesn't dictate it





Unrolling the Math

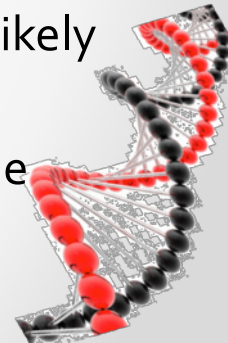
- Expected Return

  $U_\theta = \mathbb{E}[\, \Sigma_{t=0 \text{ to } \tau} R(s_t, a_t) \,;\, \pi(\theta)] = \Sigma_\tau \, \mathbb{P}(\tau; \theta) \, R(\tau)$ *… some literature calls it* J_θ

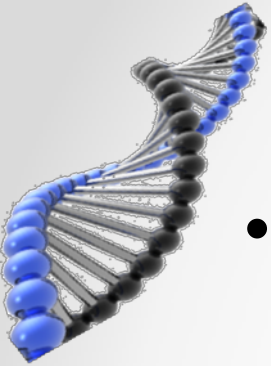
- Goal : Maximize J_θ

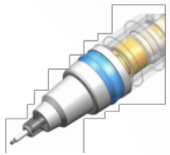
  $\max_\theta(U_\theta) = \max_\theta \{ \mathbb{E}[\, \Sigma_{t=0 \text{ to } \tau} R(s_t, a_t); \pi(\theta)] \} = \max_\theta \{ \Sigma_\tau \, \mathbb{P}(\tau; \theta) \, R(\tau) \}$

- Likelyhood Ratio Method
  - Forms the foundation for Policy Gradients, e.g. REINFORCE
  - In essence, it means our policy makes some trajectories more likely to happen than others
    - With weighting those returns from those trajectories by the probability that they actually take place



Unrolling the Math





- Expected Return

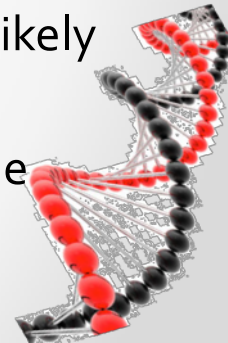
$$U_\theta = \mathbb{E}[\, \Sigma_{t=0 \text{ to } \tau} R(s_t, a_t) \,;\, \pi(\theta)] = \Sigma_\tau \, \mathbb{P}(\tau; \theta) \, R(\tau)$$

... some literature calls it J_θ

- Goal : Maximize J_θ

$$\max_\theta(U_\theta) = \max_\theta \{ \mathbb{E}[\, \Sigma_{t=0 \text{ to } \tau} R(s_t, a_t); \pi(\theta)] \} = \max_\theta \{ \Sigma_\tau \, \mathbb{P}(\tau; \theta) \, R(\tau) \}$$

- Likelyhood Ratio Method
  - Forms the foundation for Policy Gradients, e.g. REINFORCE
  - In essence, it means our policy makes some trajectories more likely to happen than others
    - With weighting those returns from those trajectories by the probability that they actually take place



Likelyhood Policy Gradient

$\max_\theta(U_\theta) = \max_\theta \{ \mathbb{E}[\Sigma_{t=o \text{ to } \tau} R(s_t, a_t); \pi(\theta)] \} = \max_\theta \{ \Sigma_\tau \mathbb{P}(\tau; \theta) R(\tau) \}$

Discovered via Gradient Ascent : $\theta_{t+1} = \theta_t + \alpha \nabla U(\theta_t)$

$$\nabla_\theta U(\theta) = \nabla_\theta \Sigma_\tau \mathbb{P}(\tau; \theta) R(\tau)$$

$$= \Sigma_\tau \nabla_\theta \mathbb{P}(\tau; \theta) R(\tau)$$

$$= \Sigma_\tau \frac{\mathbb{P}(\tau; \theta)}{\mathbb{P}(\tau; \theta)} \nabla_\theta \mathbb{P}(\tau; \theta) R(\tau)$$

$$= \Sigma_\tau \mathbb{P}(\tau; \theta) \frac{\nabla_\theta \mathbb{P}(\tau; \theta)}{\mathbb{P}(\tau; \theta)} R(\tau)$$

Def:

$$\nabla_x \log f(x) = \frac{\nabla_\theta f(x)}{f(x)}$$

$$\nabla_\theta U(\theta) = \Sigma_\tau \mathbb{P}(\tau; \theta) \nabla_\theta \log \mathbb{P}(\tau; \theta) R(\tau)$$

Likelyhood Policy Gradient

$\max_\theta(U_\theta) = \max_\theta \{ \mathbb{E}[\Sigma_{t=o \text{ to } \tau} R(s_t, a_t); \pi(\theta)] \} = \max_\theta \{ \Sigma_\tau \mathbb{P}(\tau; \theta) R(\tau) \}$

Discovered via Gradient Ascent : $\theta_{t+1} = \theta_t + \alpha \nabla U(\theta_t)$

$$\nabla_\theta U(\theta) = \nabla_\theta \Sigma_\tau \mathbb{P}(\tau; \theta) R(\tau)$$

$$= \Sigma_\tau \nabla_\theta \mathbb{P}(\tau; \theta) R(\tau)$$

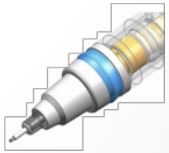
$$= \Sigma_\tau \frac{\mathbb{P}(\tau; \theta)}{\mathbb{P}(\tau; \theta)} \nabla_\theta \mathbb{P}(\tau; \theta) R(\tau)$$

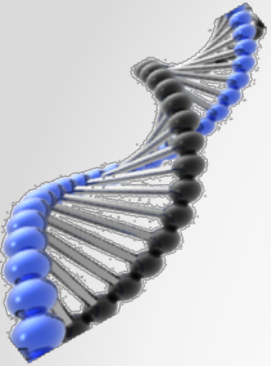
$$= \Sigma_\tau \mathbb{P}(\tau; \theta) \frac{\nabla_\theta \mathbb{P}(\tau; \theta)}{\mathbb{P}(\tau; \theta)} R(\tau)$$

Def:

$$\nabla_x \log f(x) = \frac{\nabla_\theta f(x)}{f(x)}$$

$$\nabla_\theta U(\theta) = \Sigma_\tau \mathbb{P}(\tau; \theta) \nabla_\theta \log \mathbb{P}(\tau; \theta) R(\tau)$$





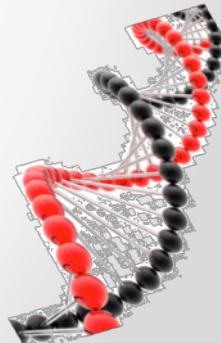
Likelyhood Policy Gradient

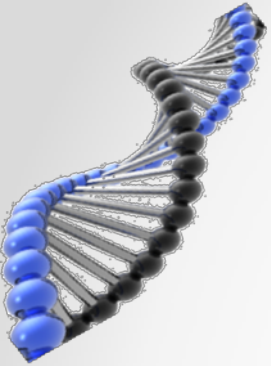
$$\nabla_\theta U(\theta) = \Sigma_{\boldsymbol{\tau}}\, \mathbb{P}(\boldsymbol{\tau};\theta)\, \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau};\theta)\, R(\boldsymbol{\tau})$$

Now we can perturb our parameter space θ and it will ascend or descend depending on $R(\boldsymbol{\tau})$

But we have 2 problems:

1. We don't have probabilities in terms of trajectories; we have states and actions !
2. We assume infinite trajectories; not feasible for mere mortals
    - *Even though OpenAI 5 is using 120,000 CPUs and 256 GPUs, learning the equivalent of 180 years of play per day !*



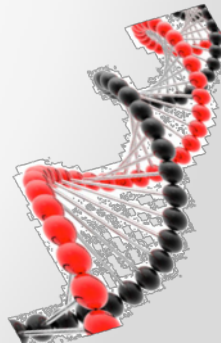


Unrolling the Math

$$\nabla_\theta U(\theta) = \Sigma_{\boldsymbol{\tau}} \mathbb{P}(\boldsymbol{\tau}; \theta) \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}; \theta) R(\boldsymbol{\tau})$$

- Considering all possible trajectories is an impossible task. While in the math world, we can image as elegant as possible.
- Down in the real world, we estimate empirically by sampling m trajectories(paths) under π_θ and average

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \Sigma_{i=1 \text{ to } m} \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}^i; \theta) R(\boldsymbol{\tau}^i)$$

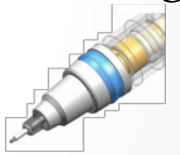


Unrolling the Math

$$\nabla_\theta U(\theta) = \Sigma_{\boldsymbol{\tau}} \mathbb{P}(\boldsymbol{\tau}; \theta) \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}; \theta) R(\boldsymbol{\tau})$$

- Considering all possible trajectories is an impossible task. While in the math world, we can image as elegant as possible.
- Down in the real world, we estimate empirically by sampling m trajectories(paths) under π_θ and average

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \Sigma_{i=1 \text{ to } m} \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}^i; \theta) R(\boldsymbol{\tau}^i)$$



Decomposing trajectories to states & actions

$$\nabla_\theta U(\theta) = \frac{1}{m} \Sigma_{i=1 \text{ to } m} \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}^i ; \theta) R(\boldsymbol{\tau}^i)$$

- Let us reason about states and actions than probabilities of trajectories

$$\nabla_\theta \log \mathbb{P}(\boldsymbol{\tau} ; \theta) = \nabla_\theta \log [\Pi_{t=0 \text{ to } T} \mathbb{P}(s_{t+1} | s_t, a_t) \pi(a_t | s_t, \theta)]$$

$$= \nabla_\theta [\Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) + \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta)]$$

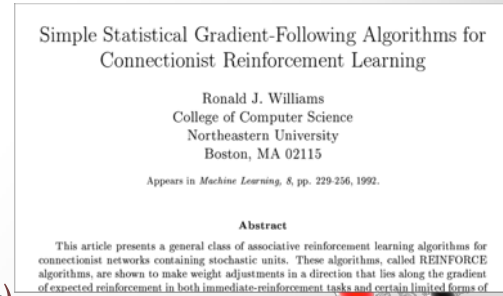
$$= \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) + \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta)$$

$$= \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta) \ \ldots \text{ since } \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) = 0$$

$$\nabla_\theta U(\theta) = \frac{1}{m} \Sigma_{i=1 \text{ to } m} \Sigma_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) R(\boldsymbol{\tau}^i)$$

… and this is the REINFORCE algorithm !

(http://www-anw.cs.umass.edu/~barto/courses/cs687/williams92simple.pdf)



Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning

Ronald J. Williams
College of Computer Science
Northeastern University
Boston, MA 02115

Appears in *Machine Learning*, 8, pp. 229-256, 1992.

Abstract

This article presents a general class of associative reinforcement learning algorithms for connectionist networks containing stochastic units. These algorithms, called REINFORCE algorithms, are shown to make weight adjustments in a direction that lies along the gradient of expected reinforcement in both immediate-reinforcement tasks and certain limited forms of

Decomposing trajectories to states & actions

$$\nabla_\theta U(\theta) = \frac{1}{m} \Sigma_{i=1 \text{ to } m} \nabla_\theta \log \mathbb{P}(\boldsymbol{\tau}^i ; \theta) R(\boldsymbol{\tau}^i)$$

- Let us reason about states and actions than probabilities of trajectories

$$\nabla_\theta \log \mathbb{P}(\boldsymbol{\tau} ; \theta) = \nabla_\theta \log [\Pi_{t=0 \text{ to } T} \mathbb{P}(s_{t+1} | s_t, a_t) \pi(a_t | s_t, \theta)]$$

$$= \nabla_\theta [\Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) + \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta)]$$

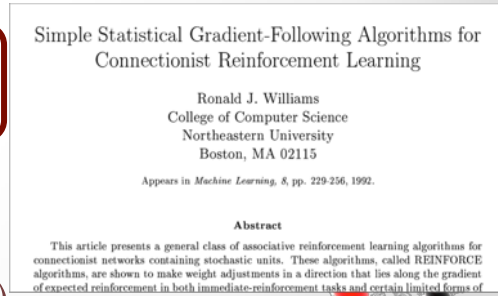
$$= \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) + \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta)$$

$$= \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \pi(a_t | s_t, \theta) \ \ldots \text{ since } \nabla_\theta \Sigma_{t=0 \text{ to } T} \log \mathbb{P}(s_{t+1} | s_t, a_t) = 0$$

$$\nabla_\theta U(\theta) = \frac{1}{m} \Sigma_{i=1 \text{ to } m} \Sigma_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) R(\boldsymbol{\tau}^i)$$

… and this is the REINFORCE algorithm !

(http://www-anw.cs.umass.edu/~barto/courses/cs687/williams92simple.pdf)



Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning

Ronald J. Williams
College of Computer Science
Northeastern University
Boston, MA 02115

Appears in *Machine Learning*, 8, pp. 229-256, 1992.

Abstract

This article presents a general class of associative reinforcement learning algorithms for connectionist networks containing stochastic units. These algorithms, called REINFORCE algorithms, are shown to make weight adjustments in a direction that lies along the gradient of expected reinforcement in both immediate-reinforcement tasks and certain limited forms of

REINFORCE = Monte Carlo PG

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

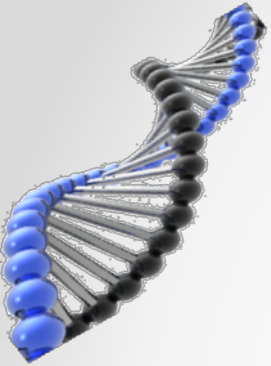
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Discounted return for variance reduction

α = 0.1 or 0.2

- Reinforce = MonteCarlo PG
  - i.e. full episode as Trajectory
  - *Just nudges the policy depending on the reward from a trajectory (could be a partial trajectory)*
- Low bias, but high variance
- REINFORCE Paper : http://www-anw.cs.umass.edu/~barto/courses/cs687/williams92simple.pdf



REINFORCE

1. Create initial policy π_θ
2. Act in the environment $a_t \sim \pi(s_t)$ for an episode storing states, actions, rewards: $s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
3. Compute return $R = \sum_{t=0}^{T} r_t$
4. Compute policy gradient $\nabla_\theta J(\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R$
5. Take a step in the direction of the gradient: $\theta = \theta + \alpha \nabla_\theta J(\theta)$
6. Repeat until convergence

$\alpha = 0.1$ or 0.2

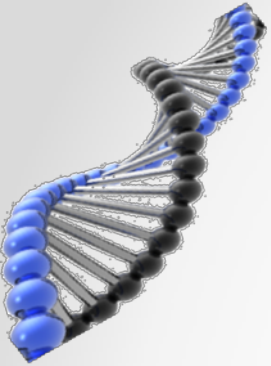
Input: policy $\pi(a|s, \boldsymbol{\theta})$, $\hat{v}(s, \boldsymbol{w})$
Parameters: step sizes, $\alpha > 0, \beta > 0$
Output: policy $\pi(a|s, \boldsymbol{\theta})$

initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\boldsymbol{w}$
for *true* **do**
  generate an episode $s_0, a_0, r_1, \cdots, s_{T-1}, a_{T-1}, r_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  for *each step t of episode* $0, \cdots, T-1$ **do**
    $G_t \leftarrow$ return from step t
    $\delta \leftarrow G_t - \hat{v}(s_t, \boldsymbol{w})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta \delta \nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t \boldsymbol{\delta} \nabla_{\boldsymbol{\theta}} log \pi(a_t|s_t, \boldsymbol{\theta})$
  end
end

Algorithm 6: REINFORCE with baseline (episodic), adapted from Sutton and Barto (2018)

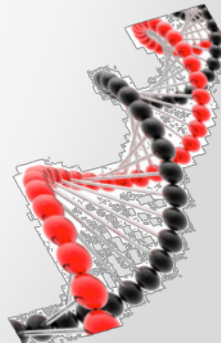
https://arxiv.org/pdf/1810.06339.pdf

The algorithm from the edX course is a lot easier to understand !



Hands-On #10 : Balancing the Cart Pole w/ Policy Gradient (REINFORCE)

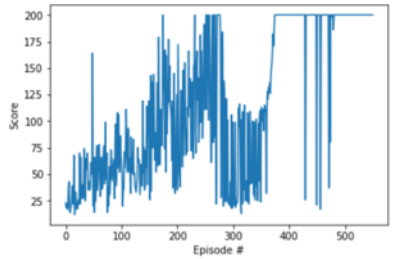
- Goal:
  - Implement Policy Gradient Network for CartPole
    - It is a lot simpler !!
- Steps:
  - Notebook : REINFORCE_01.ipynb
  - Understand the REINFORCE code
  - Plot Values



Implementation Lessons

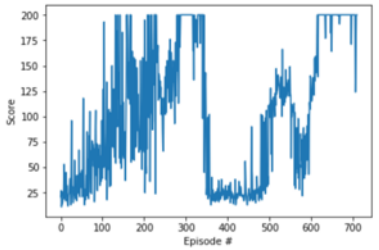
1. Very Simple
2. No digitization – neural network takes care of generalization
3. Solves the environment in less number of steps
4. It does oscillate a bit, but converges reasonably well

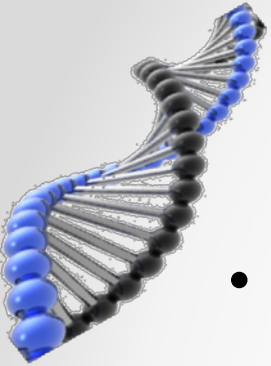
```
Episode 100     Average Score: 47.22
Episode 200     Average Score: 80.94
Episode 300     Average Score: 116.19
Episode 400     Average Score: 112.00
Episode 500     Average Score: 191.65
Environment solved in 450 episodes!     Average Score: 195.18
Elapsed : 0:00:23.645272
2019-01-12 11:47:30.619834
```




```
Max Score 200.000000 at 174
Percentile [25,50,75] : [ 51. 111. 200.]
Variance : 4779.687
```

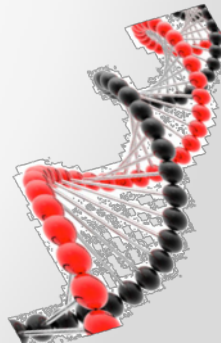
```
Episode 100     Average Score: 41.11
Episode 200     Average Score: 102.92
Episode 300     Average Score: 142.24
Episode 400     Average Score: 102.35
Episode 500     Average Score: 31.79
Episode 600     Average Score: 98.43
Episode 700     Average Score: 189.09
Environment solved in 610 episodes!     Average Score: 195.07
Elapsed : 0:00:22.520751
2019-02-08 13:10:04.801406
```

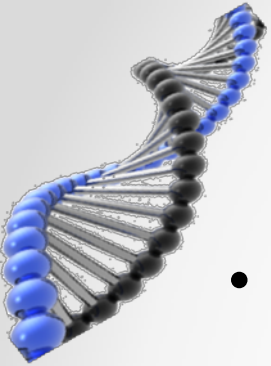




Quiz

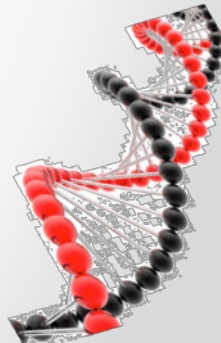
- What is the DQN DNN topology ?
  -
- What is the input size ? Output size ?
  -
- What is the output of the network ?
  -
- How does the output translate to an action ?
  -
- Is the policy deterministic or stochastic ?
  -
- How do you characterize this w.r.t others ?
  -

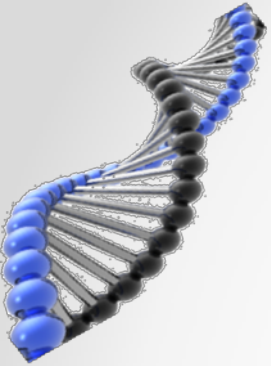




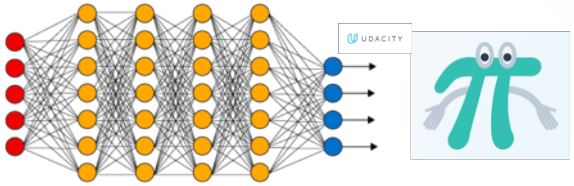
Quiz

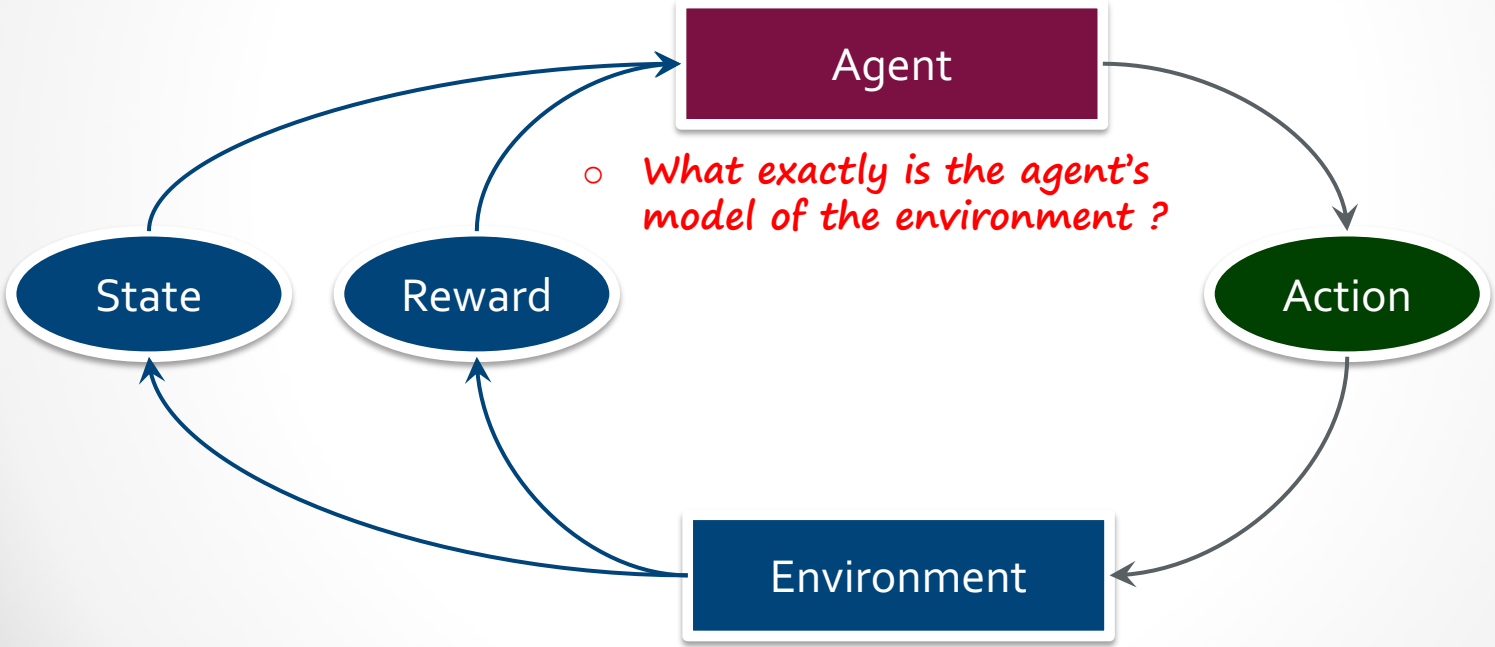
- What is the DQN DNN topology ?
  - FC16-FC2
- What is the input size ? Output size ?
  - Input = 4 (state), output = 2 (actions)
- What is the output of the network ?
  - softmax
- How does the output translate to an action ?
  - sample
- Is the policy deterministic or stochastic ?
  - Stochastic ! We get it as a part of the architecture !
- How do you characterize this w.r.t others ?
  - A better solution ! Probably the right solution for Cartpole.

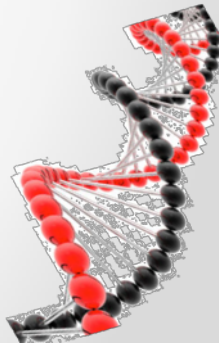


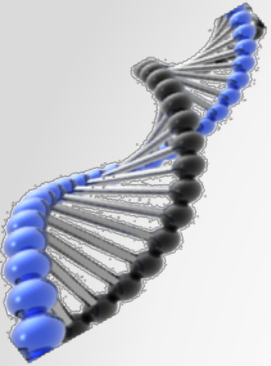


Quiz

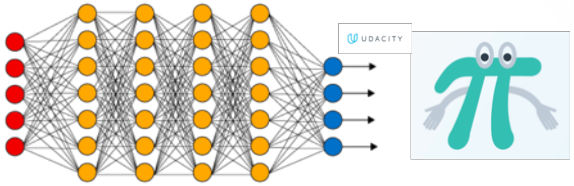


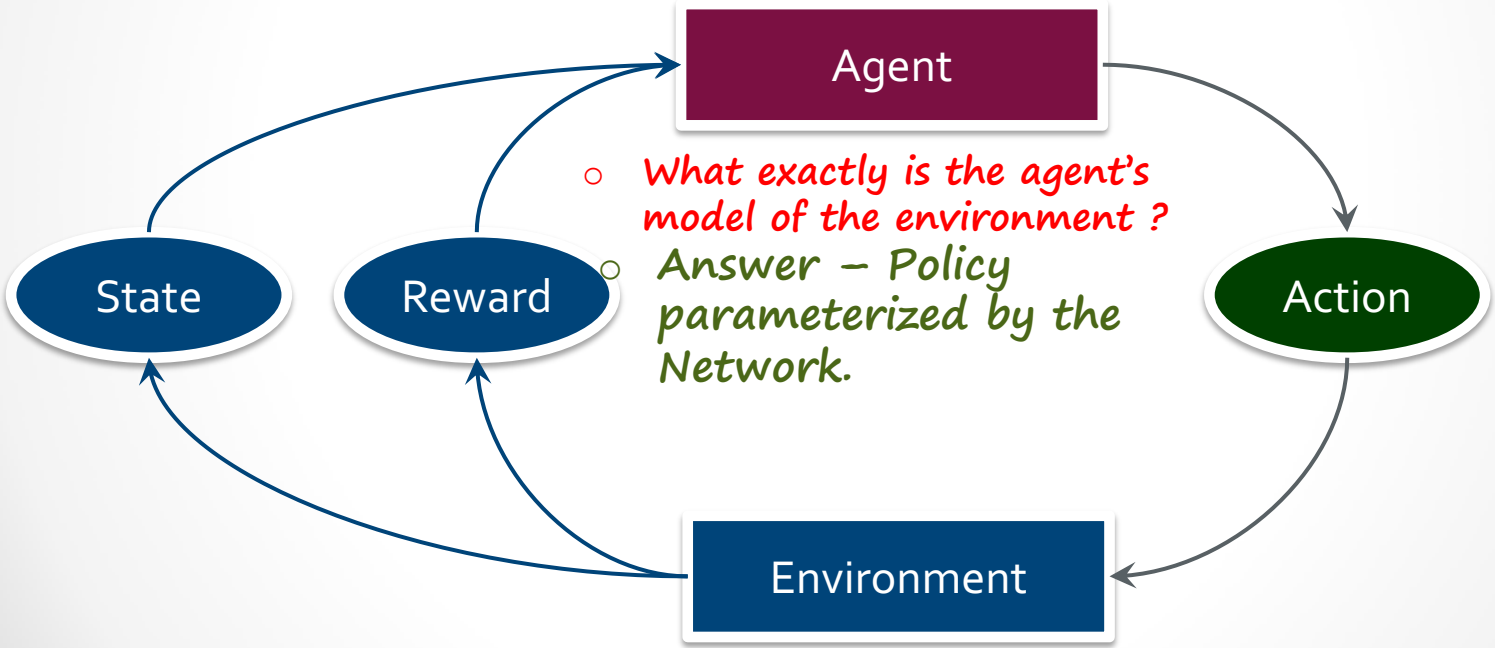

Agent
What exactly is the agent's model of the environment ?
State
Reward
Action
Environment



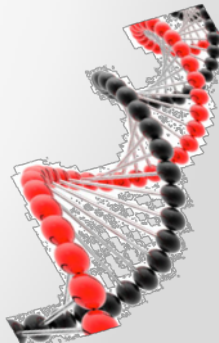


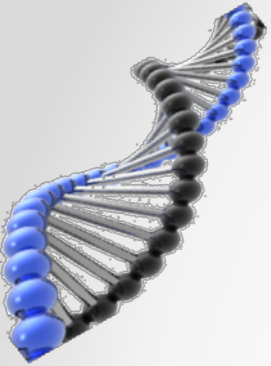
Quiz



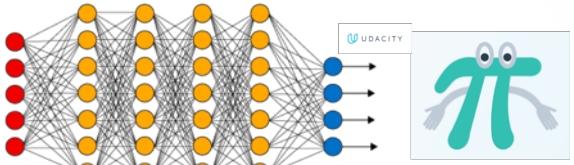


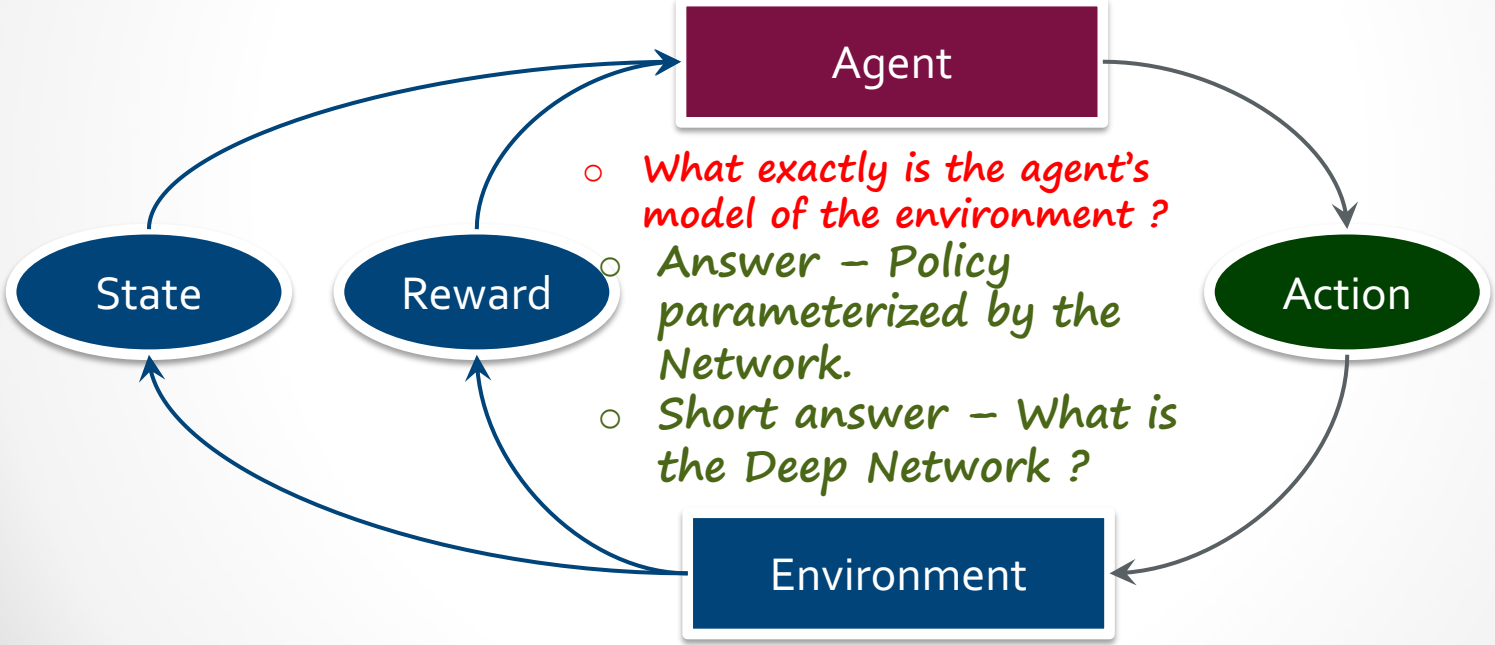
- What exactly is the agent's model of the environment ?
- Answer – Policy parameterized by the Network.



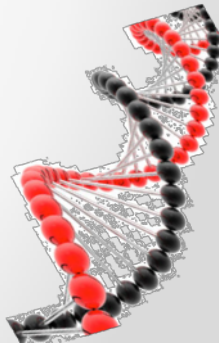


Quiz





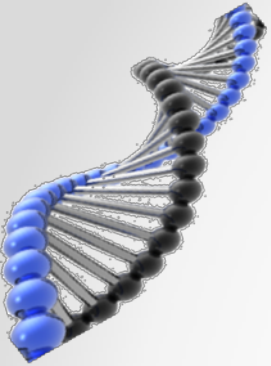
- What exactly is the agent's model of the environment ?
  - Answer – Policy parameterized by the Network.
  - Short answer – What is the Deep Network ?

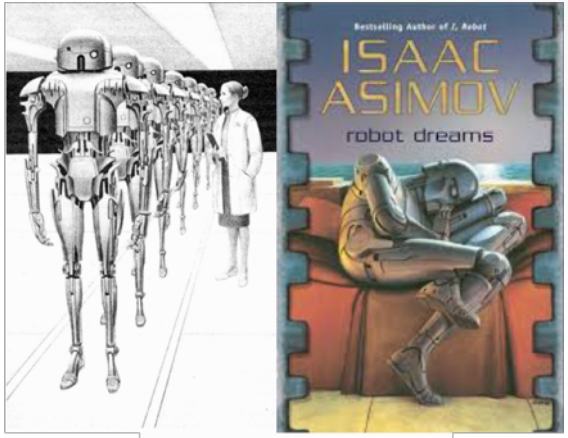


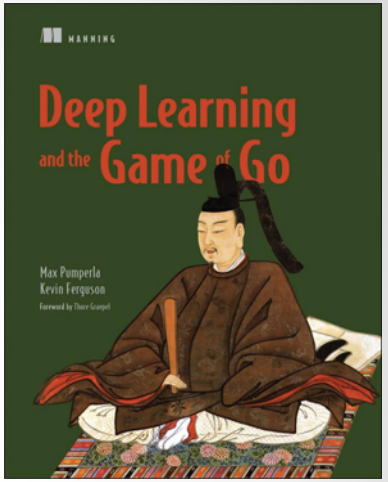
The Deadly Triad--Sutton

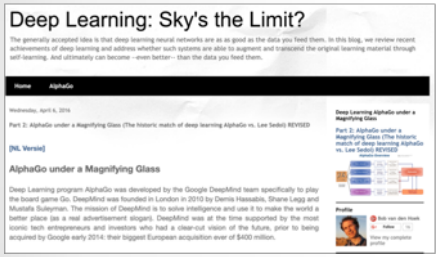
- Function Approximation
  - Generalization from examples
- Bootstrapping
- Off-Policy learning

- REINFORCE
  - Unbiased, very noisy
    - We are sampling one out of millions of trajectories, for each iteration
  - Sample in-efficient
  - Credit Assignment
- Fixes
  - Baseline
  - Temporal Structure
  - KL-Divergence trust Region



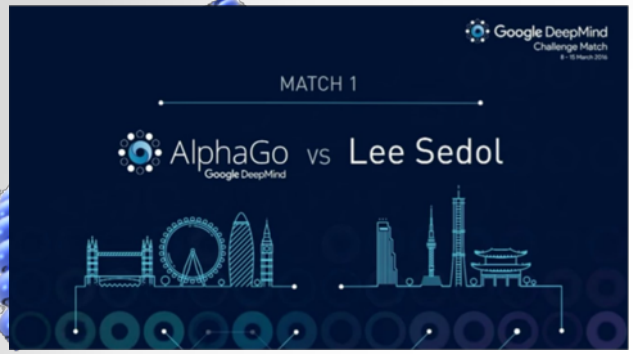






. . .

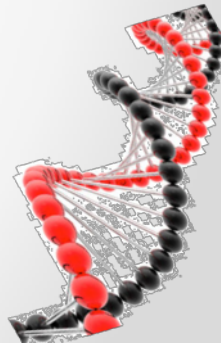
The world of α_{Go}



- AlphaGo AI program from Google's DeepMind defeated Go Champion
  - Lee Sedol (9-dan professional with 18 world titles)
- AlphaGo has the ability to look "globally" across a board—and find solutions that humans either have been trained not to play or would not consider.
  - This has huge potential for using AlphaGo-like technology to find solutions that humans don't necessarily see in other areas
- Search & Optimization
  - Policy Network with Values & search
  - "This ability of neural networks to bottle intuition / pattern recognition from one environment & apply to other contexts "

GENERATIONAL GAP !

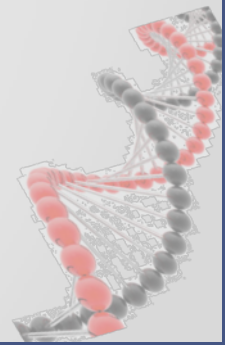
- *Unlike IBM's Deep Blue, which defeated chess champion Garry Kasparov in 1997, AlphaGo was not programmed with decision trees, or equations on how to evaluate board positions, or with if-then rules. "*
- *AlphaGo learned how to play go essentially from self-play and from observing big professional games*
  - *During training, AlphaGo played a million go games against itself.*

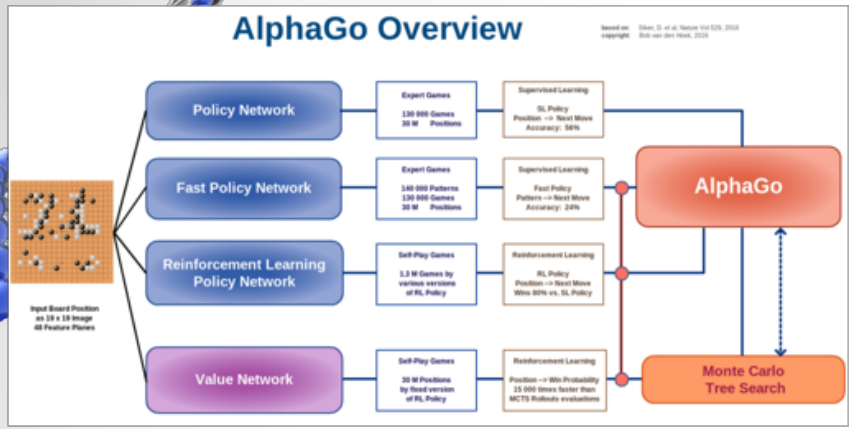


- AlphaGo AI program from Google's DeepMind defeated Go Champion
  - Lee Sedol (9-dan professional with 18 world titles)
- AlphaGo has the ability to look "globally" across a board—and find solutions that humans either have been trained not to play or would not consider.
  - This has huge potential for using AlphaGo-like technology to find solutions that humans don't necessarily see in other areas
- Search & Optimization
  - Policy Network with Values & search
  - "This ability of neural networks to bottle intuition / pattern recognition from one environment & apply to other contexts "

GENERATIONAL GAP !

- *Unlike IBM's Deep Blue, which defeated chess champion Garry Kasparov in 1997, AlphaGo was not programmed with decision trees, or equations on how to evaluate board positions, or with if-then rules. "*
- *AlphaGo learned how to play go essentially from self-play and from observing big professional games*
  - *During training, AlphaGo played a million go games against itself.*

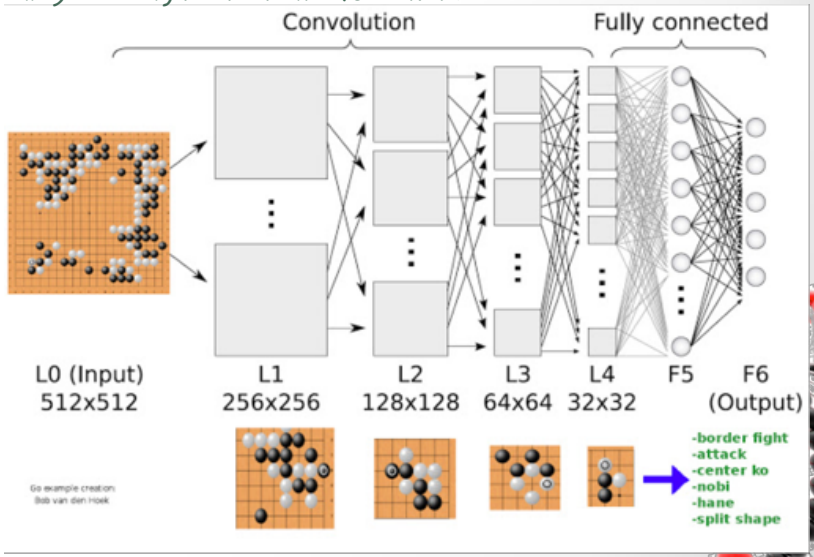


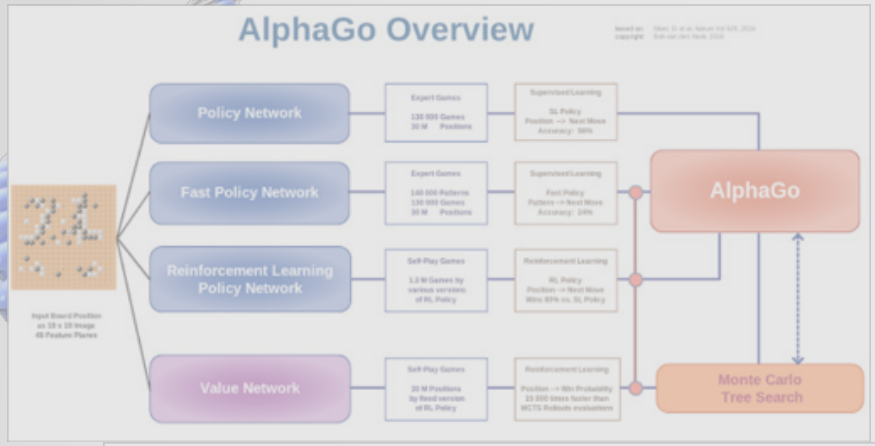


- AlphaGo is not a revolutionary breakthrough in itself, but rather a leading edge of an extremely important development
- #AlphaGo appeared to have a weakness in responding to unorthodox moves; Toward the end of the game, #AlphaGo played desperate moves similar to those of humans facing defeat …
- Ultimately, the scary thing about the rise of intelligent machines is not that they could someday have a mind of their own, but they could someday have a mind that we humans – with all our flaws and complexity – design and build for them

- It uses 4 networks :
- **Fast Rollout Policy Network** (P-Network) :It rolls out a quick plan for the game.
- **Supervised Learning Policy Network** (SL-Network) :The P-Network and the SL-Network are trained to predict human expert moves in a data set of positions.
- **Reinforcement Learning Policy Network** (RL-Network) :The RL Network is initialized as the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network.
- **Value network** : a Value Network V is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set.

https://medium.com/@arnabghosh93/deep-reinforcement-learning-driving-alphago-9881f1a90be4#.f462gnebl

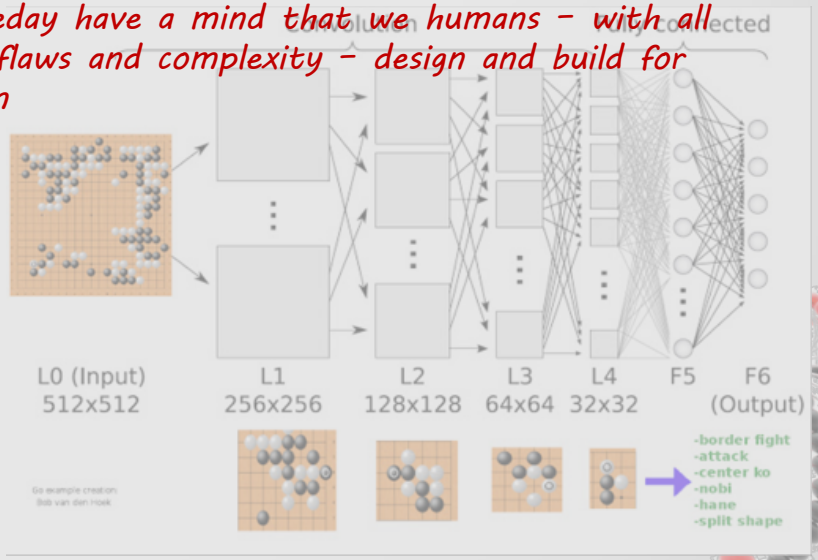




- It uses 4 networks :
- **Fast Rollout Policy Network** (P-Network) :It rolls out a quick plan for the game.
- **Supervised Learning Policy Network** (SL-Network) :The P-Network and the SL-Network are trained to predict human expert moves in a data set of positions.
- **Reinforcement Learning Policy Network** (RL-Network) :The RL Network is initialized as the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network.
- **Value network** : a Value Network V is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set.

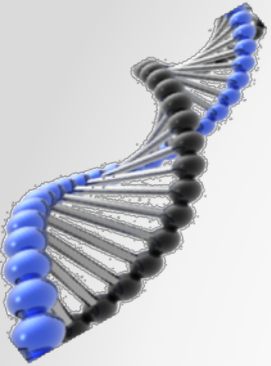
https://medium.com/@arnabghosh93/deep-reinforcement-learning-driving-alphago-9881f1a90be4#.f462gnebl

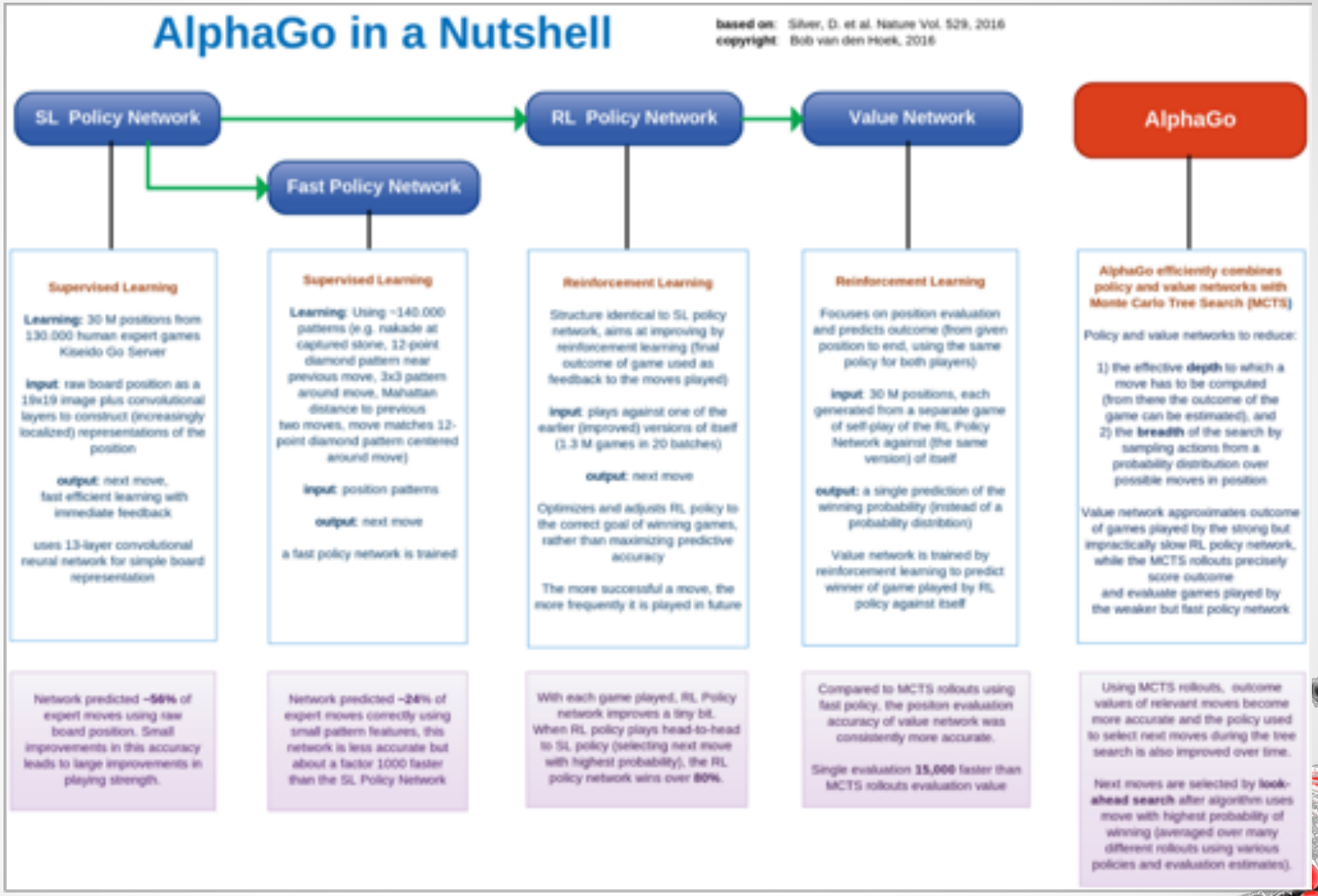
- *AlphaGo is not a revolutionary breakthrough in itself, but rather a leading edge of an extremely important development*
- *#AlphaGo appeared to have a weakness in responding to unorthodox moves; Toward the end of the game, #AlphaGo played desperate moves similar to those of humans facing defeat …*
- *Ultimately, the scary thing about the rise of intelligent machines is not that they could someday have a mind of their own, but they could someday have a mind that we humans – with all our flaws and complexity – design and build for them*

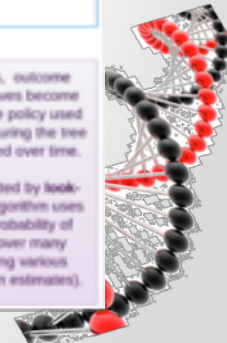


AlphaGo in a Nutshell

based on: Silver, D. et al. Nature Vol. 529, 2016







α_{Go}

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[3], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

- Combination of model-free and model-based approach
- Look ahead search using MCTS

https://storage.googleapis.com/deepmind-media/alphago/AlphaGoNaturePaper.pdf
https://pdfs.semanticscholar.org/2088/c55ec3e52a8be99cd93390702oc70b281383.pdf

Human Instinct

Positional Judgement

Reading Capability

Policy Network

Next Move

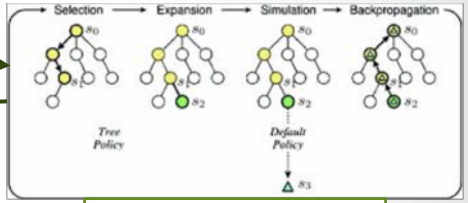
Self Play

Value Network

$v_\theta(s')$

Expected Outcome

Monte Carlo Search Tree



Rollout

$p_\sigma(a|s)$

Strong Policy

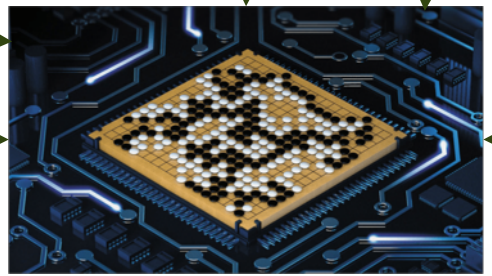
RL Policy Network

$p_\rho(a|s)$

Improved Strong Policy

Fast Rollout Policy

$p_\pi(a|s)$



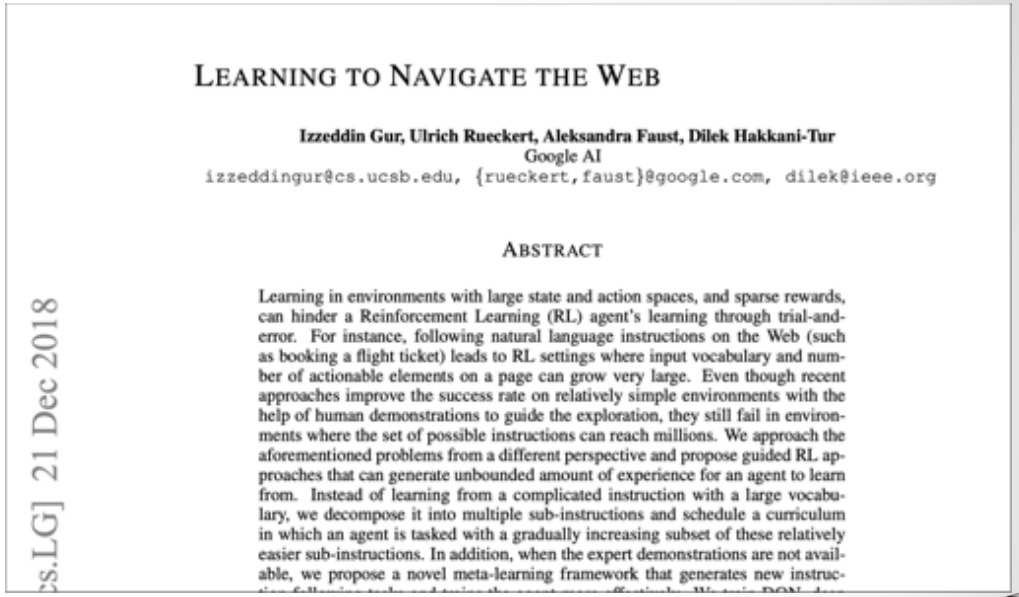
- Human Instinct : 29.4 Million positions from 160K actual human games from KGS Go server
- 19 X 19 X 48 i.e. 48 features for 19 X 19 board
- 40 threads, 48 CPUs, 8 GPUs
- Training 3 weeks for 340M steps

- 13 Layer Policy Network
- RL : REINFORCE algorithm w/baseline

- 'value networks' to evaluate board positions/*reduce depth* & 'policy networks' to select moves/*reduce breadth*

curriculum-DQN to book tickets !

- DQN can even book tickets !
- Given an instruction "Book a flight from WTK to LON on 21-Oct-2016" , the DQN agent navigates through the web form and selects the cheapest flight



LEARNING TO NAVIGATE THE WEB

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, Dilek Hakkani-Tur
Google AI
izzeddingur@cs.ucsb.edu, {rueckert,faust}@google.com, dilek@ieee.org

ABSTRACT

Learning in environments with large state and action spaces, and sparse rewards, can hinder a Reinforcement Learning (RL) agent's learning through trial-and-error. For instance, following natural language instructions on the Web (such as booking a flight ticket) leads to RL settings where input vocabulary and number of actionable elements on a page can grow very large. Even though recent approaches improve the success rate on relatively simple environments with the help of human demonstrations to guide the exploration, they still fail in environments where the set of possible instructions can reach millions. We approach the aforementioned problems from a different perspective and propose guided RL approaches that can generate unbounded amount of experience for an agent to learn from. Instead of learning from a complicated instruction with a large vocabulary, we decompose it into multiple sub-instructions and schedule a curriculum in which an agent is tasked with a gradually increasing subset of these relatively easier sub-instructions. In addition, when the expert demonstrations are not available, we propose a novel meta-learning framework that generates new instruc-

https://arxiv.org/abs/1812.09195

Will leave this section with 2 thoughts …

Should we fear the machine's strengths and (try to) control them ?

Is #AlphaGo's 3rd win a prelude of things to come ? "it played so well that it was almost scary"; "We're now convinced that AlphaGo is simply stronger than any known human Go player"; "Perhaps it's time for broader discussion about how human society will adapt to this emerging technology !!!" https://goo.gl/htmOzR show less

AlphaGo shows its true strength in 3rd victory against Lee Sedol

gogameguru.com · AlphaGo made history once again on Saturday, as the first computer program to defeat a top profes…

[Update 3/6/16] An interesting post from Tom Devenport about Cognitive Computing.

- Good insights into what Cognitive Computing is, as a combination of Intelligence(Algorithms), Inference(Knowledge) and Interface (Visualization, Recommendation, Prediction,…)
- IMHO, Cognitive Computing is more than Analytics over unstructured data, it also has touches of AI in there.
- Reason being, Cognitive Computing understands humans – whether it is about buying patterns or the way different bodies reacts to drugs or the various forms of diseases or even the way humans work and interact
- And that knowledge is the difference between Analytics and Cognitive Computing !

> I like Cognitive Computing as an important part of AI, probably that is where most of the applications are … again understanding humans rather than being humans !

Or should we work with the machines - who understand us, rather than being us ?

Will leave this section with 2 thoughts …

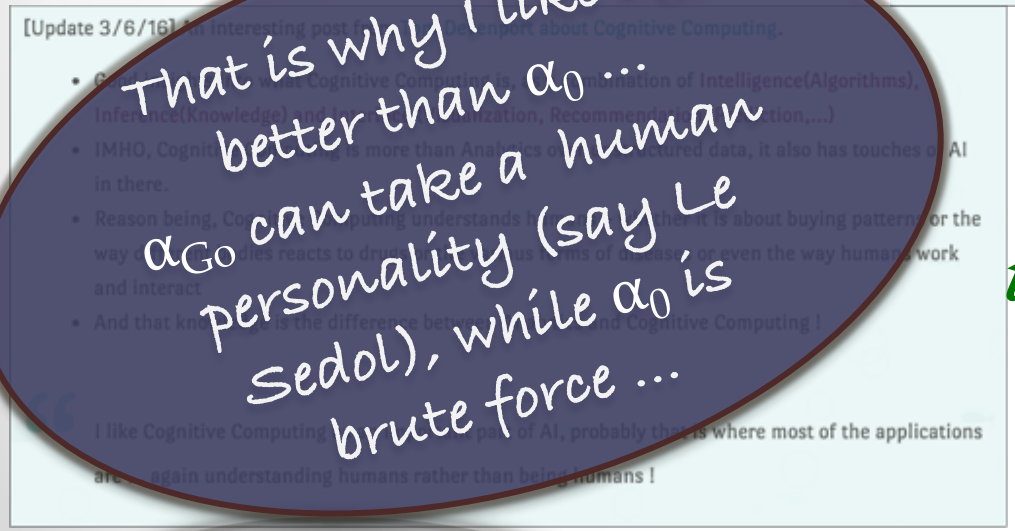
Should we fear the machine's strengths and (try to) control them ?

Is #AlphaGo's 3rd win a prelude of things to come ? "it played so well that it was almost scary"; "We're now convinced that AlphaGo is simply stronger than any known human Go player"; "Perhaps it's time for broader discussion about how human society will adapt to this emerging technology !!!" https://goo.gl/htmOzR show less

AlphaGo shows its true strength in 3rd victory against Lee Sedol

gogameguru.com · AlphaGo made history once again on Saturday, as the first computer program to defeat a top profes…

That is why I like α_{Go} better than α_0 … α_{Go} can take a human personality (say Le Sedol), while α_0 is brute force …



Or should we work with the machines - who understand us, rather than being us ?

Will leave this section with 2 thoughts …

Should we fear the machine's strengths and (try to) control them ?

Is #AlphaGo's 3rd win a prelude of things to come ? "it played so well that it was almost scary"; "We're now convinced that AlphaGo is simply stronger than any known human Go player"; "Perhaps it's time for broader discussion about how human society will adapt to this emerging technology !!!" https://goo.gl/htmOzR show less

AlphaGo shows its true strength in 3rd victory against Lee Sedol

gogameguru.com · AlphaGo made history once again on Saturday, as the first computer program to defeat a top profes…

Self-awareness between self & Task – interesting paper from Columbia

https://www.eurekalert.org/pub_releases/2019-01/cuso-asc012819.php

[Update 3/6/16] An interesting post from Tom Davenport on Cognitive Computing.

- Good insights into what Cognitive Computing is, as a combination of Intelligence(Algorithms), Interaction and Interface (Recommendation, Prediction,…)
- Indeed, Cognitive Computing is more than Analytics over unstructured data, it also has touches of AI in there.
- Reason being, Cognitive Computing emulates humans – whether it is about buying patterns or the way different people react to drugs or the various symptoms of diseases or even the way humans work and interact.
- And that knowledge is the difference between Analytics and Cognitive Computing !

I like Cognitive Computing as an important part of AI, probably that is where most of the applications are – again understanding humans rather than being humans !

Or should we work with the machines – who understand us, rather than being us ?

Will leave this section with 2 thoughts …

Should we fear the machine's strengths and (try to) control them ?

Is #AlphaGo's 3rd win a prelude of things to come ? "it played so well that it was almost scary"; "We're now convinced that AlphaGo is simply stronger than any known human Go player"; "Perhaps it's time for broader discussion about how human society will adapt to this emerging technology !!!" https://goo.gl/htmOzR show less

AlphaGo shows its true strength in 3rd victory against Lee Sedol

gogameguru.com · AlphaGo made history once again on Saturday, as the first computer program to defeat a top profes…

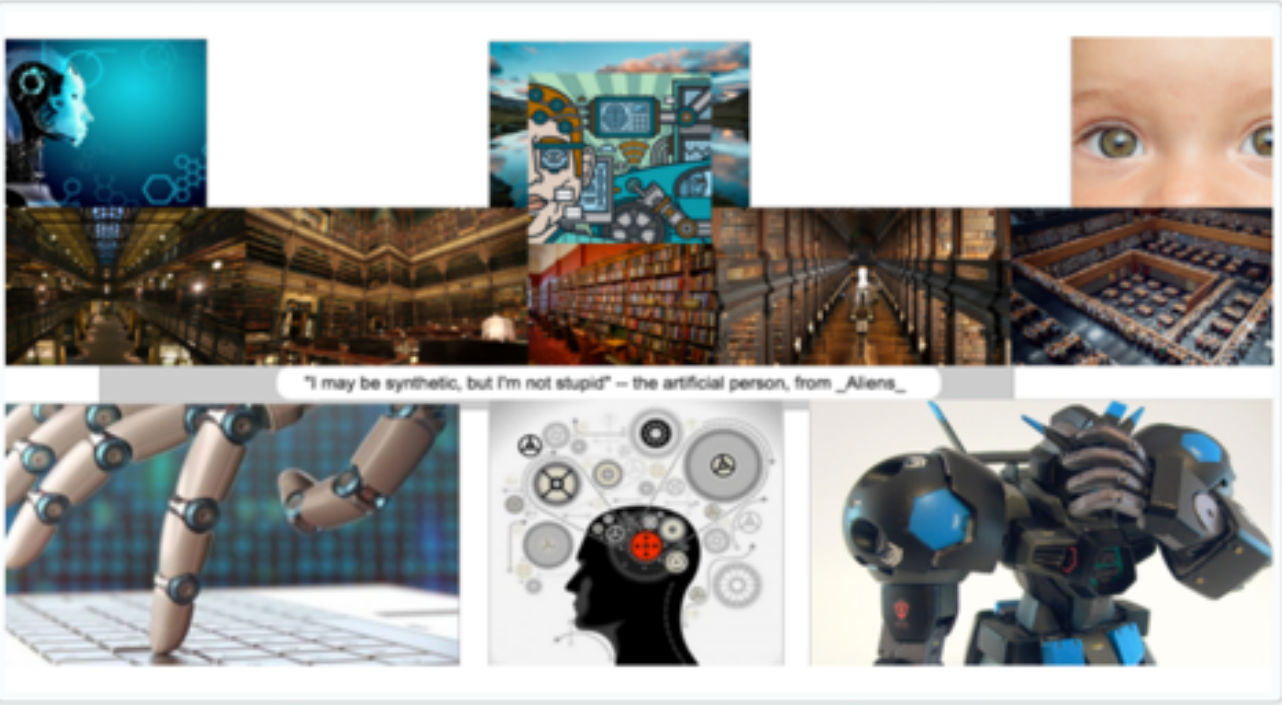
[Update 3/6/16] An interesting post from Tom Devenport about Cognitive Computing.

- Good insights into what Cognitive Computing is the combination of Intelligence(Algorithms), Inference(Knowledge) and Visualization(Presentation, Prediction,…)
- IMHO, Cognitive Computing is more than Analytics over unstructured data, it also has touches of AI in there.
- Reasoning is Cognitive Computing understands human behavior, it is about buying patterns or the way different objects to diagnosis of diseases or even the way humans work and interact.
- And the big challenge is the difference between Analytics and Cognitive Computing !

"I like Cognitive Computing as an important part of AI, probably that is where most of the applications are again understanding humans rather than being humans !

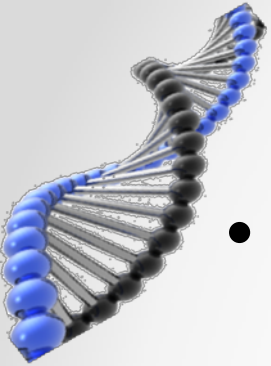
"The researchers are now exploring whether robots can model not just their own bodies, but also their own minds, whether robots can think about thinking"

Or should we work with the machines - who understand us, rather than being us ?



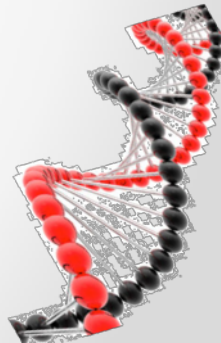
WHAT WOULD YOU WANT AI TO DO, IF IT COULD DO WHATEVER YOU WANTED IT TO DO ?

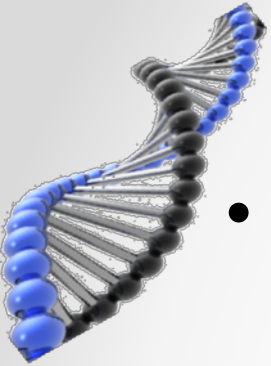
https://goo.gl/6pKvgY



Back to the main feature …

- REINFORCE
  - High variance
  - Difficult to converge
  - The dependence on $R(\tau^i)$
    - Pushes probabilities for all actions in that trajectory
    - Which is not fully meaningful
  - What if we have a better yardstick ?
    - i.e. better or worse than what you expect ?
    - i.e. compare with a baseline



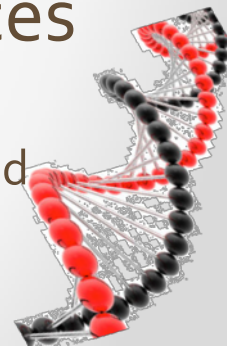


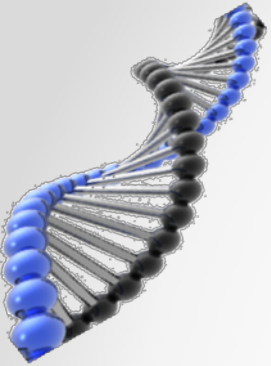
Policy Gradients with a Baseline

- Bias and variance
  - MonteCarlo has *low bias* – because it accumulates an estimate of reward from actual episodes
    - But because a state can participate in many different trajectories, there is the *high variance* of the cumulated rewards
  - TD has low variance, because it works with a few steps, single step in many cases, instead of full rollout
    - But it has a high bias because it depends on estimates of estimates !
- Adding a TD baseline to REINFORCE reduces the variance
  - Thus improves the convergence properties - results in better and consistent convergence
  - And because of TD, speeds up learning as well !

UDACITY

Ref: Miguel Morales





Reinforce with Baseline

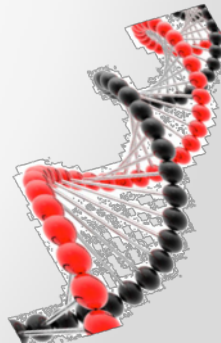
$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, R(\tau^i)$$

$$= \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, (R(\tau^i) - b)$$

Many interesting choice for a baseline:

1. Average of all rewards so far i.e. $\mu\{R(\tau)\}$ – constant moving average
2. Optimal Baseline
3. Time dependent baseline
4. State dependent expected baseline i.e. $V_\pi(s)$

$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, (R(\tau^i) - V_\phi(s))$$



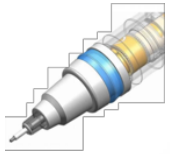
Reinforce with Baseline

$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, R(\boldsymbol{\tau}^i)$$

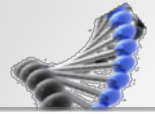
$$= \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, (R(\boldsymbol{\tau}^i) - b)$$

Many interesting choice for a baseline:

1. Average of all rewards so far i.e. $\mu\{R(\boldsymbol{\tau})\}$ – constant moving average
2. Optimal Baseline
3. Time dependent baseline
4. State dependent expected baseline i.e. $V_\pi(s)$



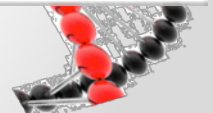
$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t | s_t, \theta) \, (R(\boldsymbol{\tau}^i) - V_\phi(s))$$



REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

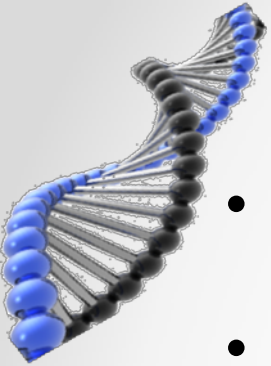
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
    $\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$



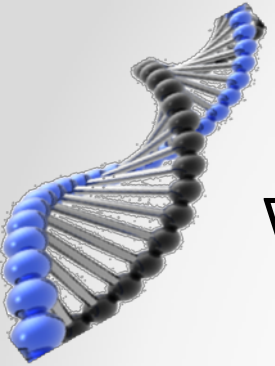
A Simpler Algorithm from edX

1. Initialize policy π_θ and value function V_ϕ
2. For episode 1,2,...,n do:
    1. Collect trajectory τ by running current policy
    2. For each step in τ compute the discounted return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
    3. Re-fit the baseline minimizing $\left\| V_\phi(s_t) - R_t \right\|^2$ over all steps
    4. Update π_θ using gradient $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t))$



Morphing to A3C

- Would a better baseline reduce the variance for REINFORCE ? And generalize ?
- Can we use a Neural Network to establish this baseline ?
- Of course, Yes to both
  - A3C & it's synchronous twin A2C
- And, … if we use a TD-Critic, instead of a MonteCarlo baseline
  - Faster Learning than MC alone
  - Consistent Convergence than value based methods
  - And, we gain generalization
- And that in essence leads us to A3C & then to DDPG !



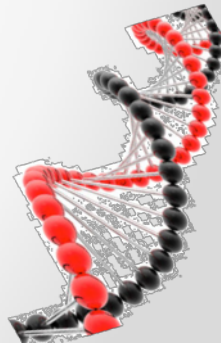
Advantage

$$\nabla_\theta U(\theta) = \frac{1}{m} \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \, (R_t - V_\phi(s_t))$$

Actor

Critic

- Interestingly, R_t is nothing but $Q_\pi(s_t, a_t)$
- So $R_t - V_\phi(s_t)$ becomes $Q_\pi(s_t, a_t) - V_\phi(s_t)$
  - Which says how much better the new state-action value is over the current value i.e. Advantage
- $A(s_t, a_t) = Q_\pi(s_t, a_t) - V_\phi(s_t)$

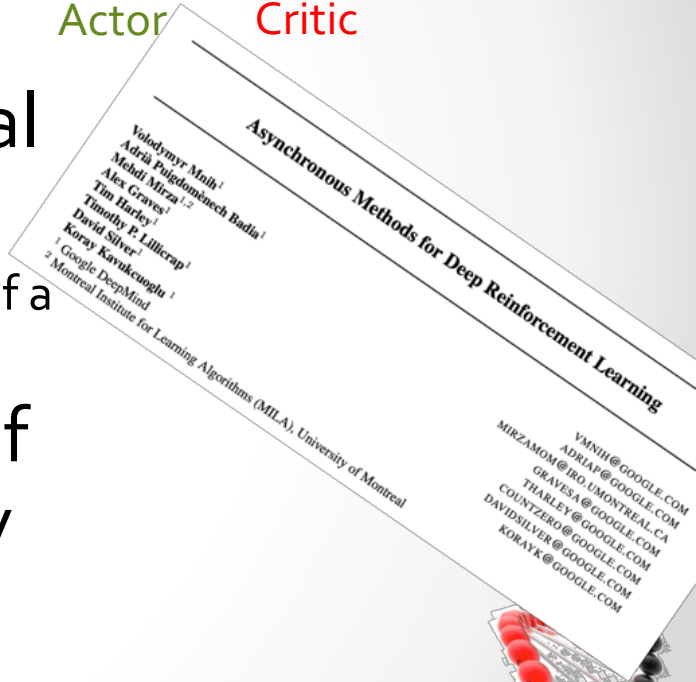


A Basic Actor-Critic Agent

$$\nabla_\theta U(\theta) = \sum_{i=1 \text{ to } m} \sum_{t=0 \text{ to } T} \nabla_\theta \log \pi(a_t \mid s_t, \theta) A_t$$

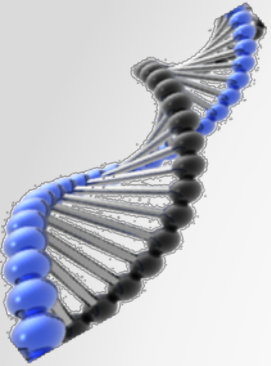
Actor Critic

- Function Approximation ie neural network for Actor and Critic
  - i.e. the Policy and the Predictor of Advantage of a policy using TD estimate
- Once you know the advantage of a policy, we can tweak the policy up or down
  - https://arxiv.org/abs/1602.01783



A3C Intuition

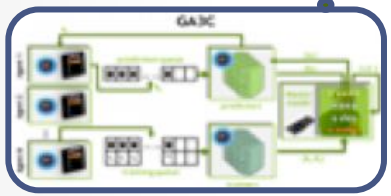
$$\pi_\theta = \pi_\theta + \alpha \Sigma \nabla \log \pi(a_t | s_t, \theta_\pi) (R(\tau) - V(s_t;\theta_v))$$



Train Actor 7

$A(s_t,a_t) = r_{t+1}+\gamma V(s_{t+1};\theta_v) - V(s_t;\theta_v)$

Actor θ_π



s_t

$\pi(a_t | s_t;\theta_\pi)$

a_t

2

(r_{t+1}, s_{t+1})

4

1

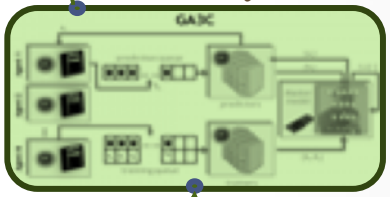
3

$(s_t,a_t,r_{t+1}, s_{t+1})$

$r_{t+1}+\gamma V(s_{t+1};\theta_v)$

6

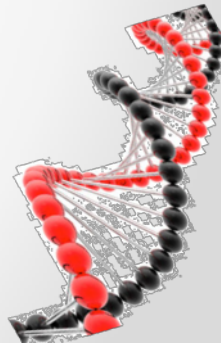
Critic θ_v



s_t

$V(s_t;\theta_v)$

5 Train Critic



One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

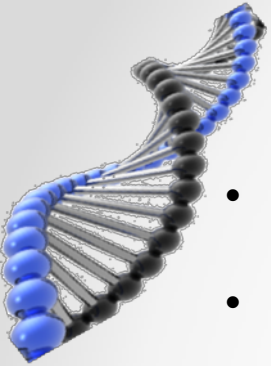
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Input: policy $\pi(a|s, \boldsymbol{\theta})$, $\hat{v}(s, \boldsymbol{w})$
Parameters: step sizes, $\alpha > 0$, $\beta > 0$
Output: policy $\pi(a|s, \boldsymbol{\theta})$

initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\boldsymbol{w}$
for *true* **do**
  initialize s, the first state of the episode
  $I \leftarrow 1$
  for s *is not terminal* **do**
    $a \sim \pi(\cdot|s, \boldsymbol{\theta})$
    take action a, observe s', r
    $\delta \leftarrow r + \gamma \hat{v}(s', \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w})$ (if s' is terminal, $\hat{v}(s', \boldsymbol{w}) \doteq 0$)
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \beta \delta \nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I \delta \nabla_{\boldsymbol{\theta}} log \pi(a_t|s_t, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $s \leftarrow s'$
  end
end

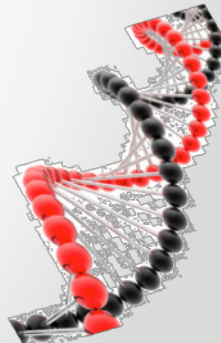
Algorithm 7: Actor-Critic (episodic), adapted from Sutton and Barto (2018)

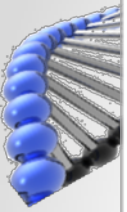
https://arxiv.org/pdf/1810.06339.pdf



A3C

- A3C can share networks between the Actor and Critic.
  - But harder to train – so start with 2 networks, do hyperparameter and architecture tuning and then combine for performance
- N-step bootstrapping TD(lambda) with lambda = n
  - More n, less bias while keeping variance under control
  - Usually 4-5 step bootstrapping are the best
    - Update last n steps
- A3C doesn't use experience replay but gets diversity from parallel training ie running multiple agents to a common buffer
- On or off policy ? On ! Stable and consistent convergence property !
  - Off policy often unstable and can diverge with nn
  - But there is work to make off-policy critic better
    - Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic https://arxiv.org/abs/1611.02247
- A2C – Synchronous A3C !
  - Synchronous point, update network and update gradients of all agents
  - A3C CPU is Ok. A2C – better on a GPU
- UC Berkeley Soft Actor Critic ! New development as of Jan'19
  - https://bair.berkeley.edu/blog/2018/12/14/sac/





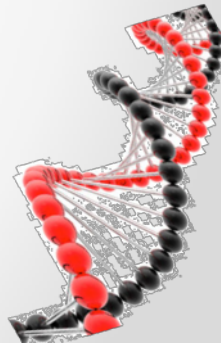
global shared parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_v$, thread-specific parameter vectors $\boldsymbol{\theta}'$ and $\boldsymbol{\theta}'_v$
global shared counter $T = 0$, T_{max}
initialize step counter $t \leftarrow 1$
for $T \leq T_{max}$ **do**
    reset gradients, $d\boldsymbol{\theta} \leftarrow 0$ and $d\boldsymbol{\theta}_v \leftarrow 0$
    synchronize thread-specific parameters $\boldsymbol{\theta}' = \boldsymbol{\theta}$ and $\boldsymbol{\theta}'_v = \boldsymbol{\theta}_v$
    set $t_{start} = t$, get state s_t
    for s_t *not terminal and* $t - t_{start} \leq t_{max}$ **do**
        take a_t according to policy $\pi(a_t|s_t;\boldsymbol{\theta}')$
        receive reward r_t and new state s_{t+1}
        $t \leftarrow t+1, T \leftarrow T+1$
    end

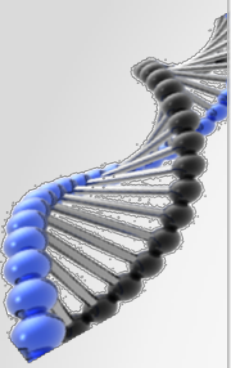
$$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \boldsymbol{\theta}'_v) & \text{otherwise} \end{cases}$$

    for $i \in \{t-1, ..., t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        accumulate gradients wrt $\boldsymbol{\theta}'$: $d\boldsymbol{\theta} \leftarrow d\boldsymbol{\theta} + \nabla_{\boldsymbol{\theta}'} \log \pi(a_i|s_i;\boldsymbol{\theta}')(R - V(s_i;\boldsymbol{\theta}'_v))$
        accumulate gradients wrt $\boldsymbol{\theta}'_v$: $d\boldsymbol{\theta}_v \leftarrow d\boldsymbol{\theta}_v + \nabla_{\boldsymbol{\theta}'_v}(R - V(s_i;\boldsymbol{\theta}'_v))^2$
    end
    update asynchronously $\boldsymbol{\theta}$ using $d\boldsymbol{\theta}$, and $\boldsymbol{\theta}_v$ using $d\boldsymbol{\theta}_v$
end

Algorithm 10: A3C, each actor-learner thread, based on Mnih et al. (2016)

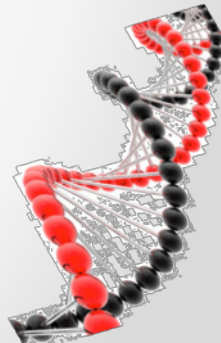
https://arxiv.org/pdf/1810.06339.pdf





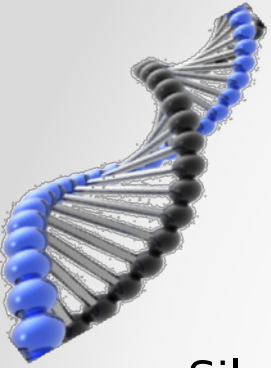
Algorithm S2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared parameter vector θ.
// Assume global shared target parameter vector θ^-.
// Assume global shared counter $T = 0$.
Initialize thread step counter $t \leftarrow 1$
Initialize target network parameters $\theta^- \leftarrow \theta$
Initialize thread-specific parameters $\theta' = \theta$
Initialize network gradients $d\theta \leftarrow 0$
repeat
  Clear gradients $d\theta \leftarrow 0$
  Synchronize thread-specific parameters $\theta' = \theta$
  $t_{start} = t$
  Get state s_t
  repeat
    Take action a_t according to the ϵ-greedy policy based on $Q(s_t, a; \theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \frac{\partial \left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
  end for
  Perform asynchronous update of θ using $d\theta$.
  if $T \mod I_{target} == 0$ **then**
    $\theta^- \leftarrow \theta$
  end if
until $T > T_{max}$



Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$



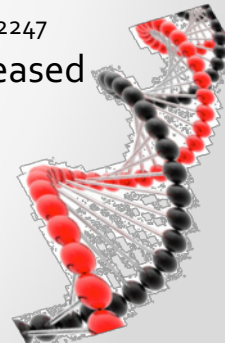
Evolution of Policy Gradients

DDPG

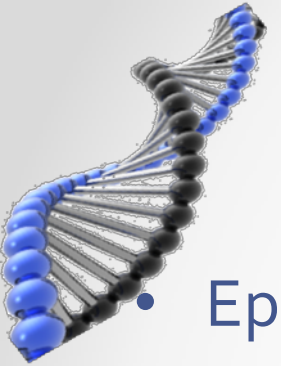
- Silver et al. released the DPG algorithm in 2014
  - http://proceedings.mlr.press/v32/silver14.pdf
- Lillicrap et al. iterated DPG with DDPG in 2015
  - https://arxiv.org/abs/1509.02971
- Ornstein–Uhlenbeck process
  - https://en.wikipedia.org/wiki/Ornstein–Uhlenbeck_process
- A3c vs DDPG
  - https://yodahuang.github.io/articles/DDPG-vs-A2C/

PPO

- In 2016, Schulman et al. released TRPO() and GAE algorithms(GAE on TRPO)
  - TRPO : https://arxiv.org/abs/1502.05477
  - GAE : https://arxiv.org/abs/1506.02438
- PPO (Proximal Policy Optimization Algorithms)
  - https://arxiv.org/abs/1707.06347
- Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic
  - Gu, Lillicrap, Levine : https://arxiv.org/abs/1611.02247
- Sergey Levine, Chelsea Finn, et al. released the GPS algorithm
  - Guided Policy Search http://rll.berkeley.edu/gps/



A3C vs DDPG

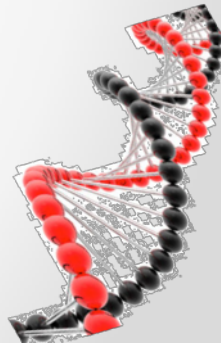


A3C

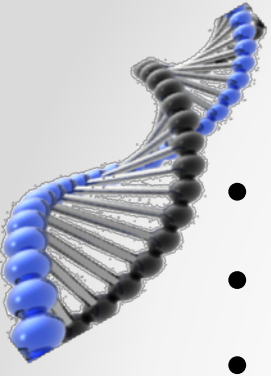
- Episode (~MC paradigm)
- On-line, on-policy
- Loss = loglikelihood
- Continuous action can be modelled as a Gaussian vector

DDPG

- TD() & TD(ƛ)
- Off-Policy
- Experience Replay
- Target network mix-in & blending via soft update
- Better for continuous state & action space

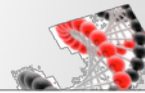


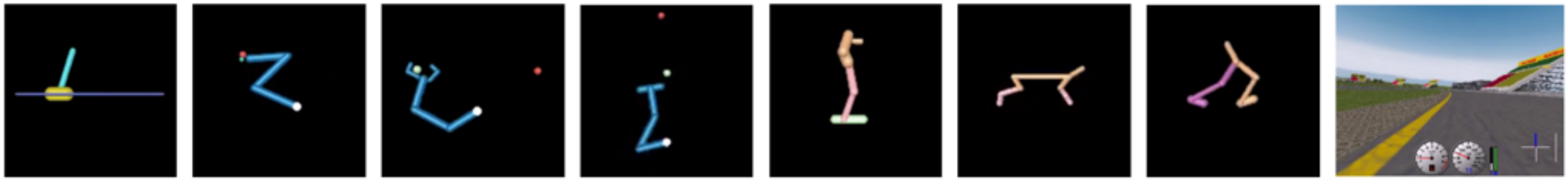
They both use the Actor-Critic design – approximately ! - finer semantics might differ



DDPG

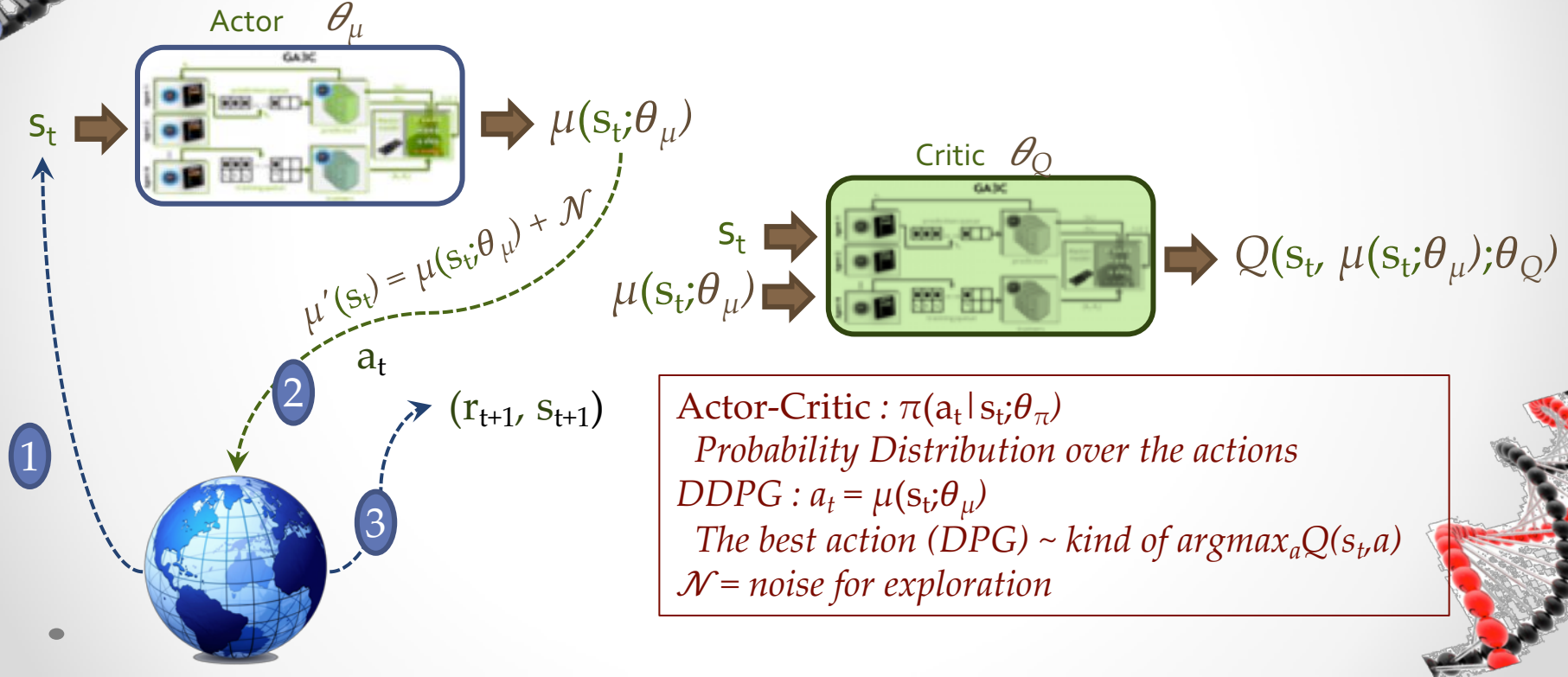
- Paper https://arxiv.org/abs/1509.02971
- Deterministic Policy : $a_t = \mu(s_t|\theta^\mu)$
- Paper says it is actor-critic
  - Morales – it is more of an approx. DQN for continuous *action* space similar to NAF(https://arxiv.org/abs/1603.00748)
  - The critic is a maximizer not a baseline
- A key feature of the approach is its simplicity
  - *It requires only a straightforward actor-critic architecture and learning algorithm with very few "moving parts", making it easy to implement and scale to more difficult problems and larger networks*



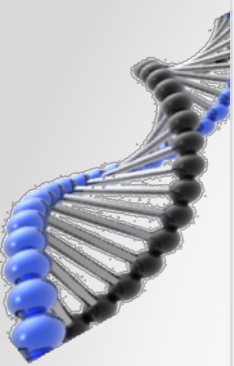


DDPG Intuition

$$\mu_{\theta(t+1)} = \mu_\theta + \alpha \Sigma \nabla_a Q(s_t, a; \theta_Q) \mid (a=\mu(s_t)\ \nabla_{\theta\mu}\ \mu(s_t;\theta_\mu)$$



Actor-Critic : $\pi(a_t \mid s_t;\theta_\pi)$
Probability Distribution over the actions
DDPG : $a_t = \mu(s_t;\theta_\mu)$
The best action (DPG) ~ kind of $argmax_a Q(s_t,a)$
$\mathcal{N}$ *= noise for exploration*



Algorithm 1 DDPG algorithm

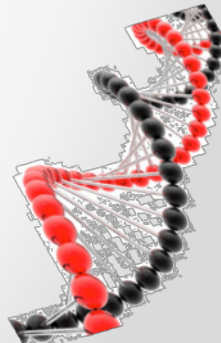
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

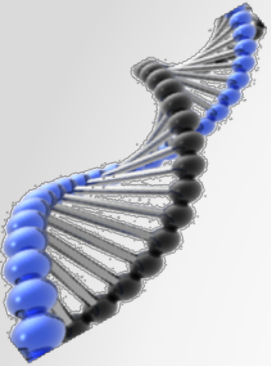
$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for





Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

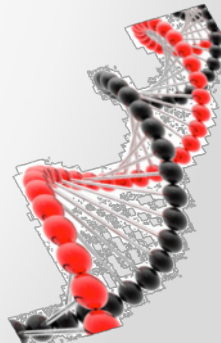
      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

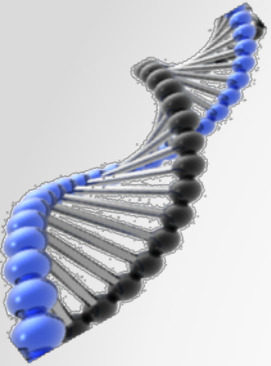
      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

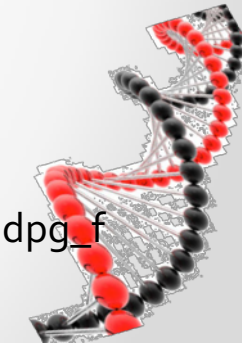


https://spinningup.openai.com/en/latest/algorithms/ddpg.html



Hands-On #11 : Balancing the Cart Pole w/ DDPG

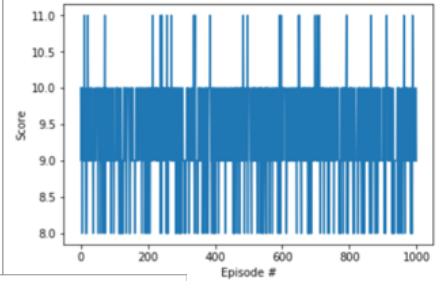
- Goal:
  - Implement DDPG on the CartPole Environment
  - It is an overkill, but we will get a good understanding and also can compare against other algorithms
- Steps:
  - Program DDPG Algorithm
  - Run and Optimize
  - Plot Values, like we did in other exercises
- DDPG for discrete actions
  - DDPG for Discrete Action Domains https://www.reddit.com/r/reinforcementlearning/comments/8k8bye/ddpg_for_discrete_action_domains/



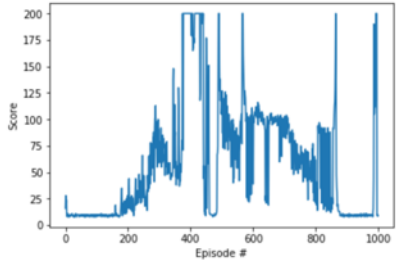
Bad Runs ..

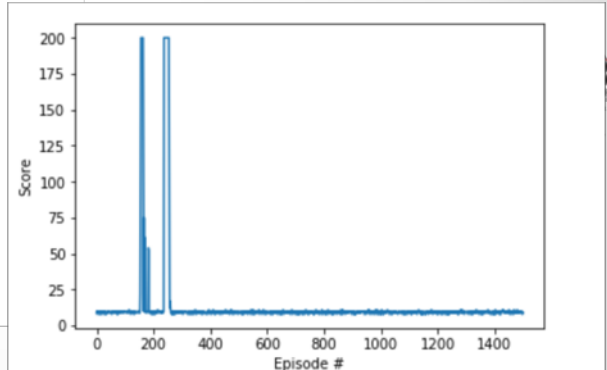
- Just meanders around
- Doesn't go beyond a score of 11
- Or learns well
- Then unlearns

```
Episode 100     Average Score: 9.3131   Score:  9.00   Max_steps :  8
Episode 200     Average Score: 9.2929   Score: 10.00   Max_steps :  9
Episode 300     Average Score: 9.4242   Score:  8.00   Max_steps :  7
Episode 400     Average Score: 9.3232   Score: 10.00   Max_steps :  9
Episode 500     Average Score: 9.3333   Score: 10.00   Max_steps :  9
Episode 600     Average Score: 9.3333   Score:  9.00   Max_steps :  8
Episode 700     Average Score: 9.4343   Score: 10.00   Max_steps :  9
Episode 800     Average Score: 9.3838   Score: 10.00   Max_steps :  9
Episode 900     Average Score: 9.2929   Score: 10.00   Max_steps :  9
Episode 1000    Average Score: 9.2828   Score:  9.00   Max_steps :  8
Elapsed : 0:00:37.123424
2019-02-02 09:10:54.042185
```

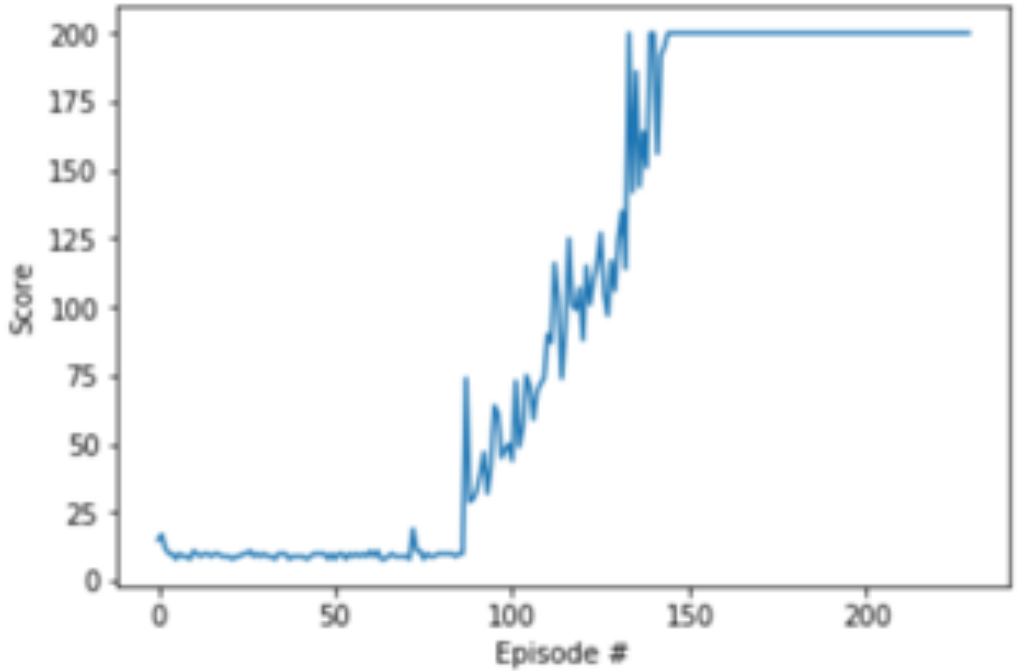



```
Episode 100     Average Score: 9.8080   Score:   8.00   Max_steps :   7
Episode 200     Average Score: 11.433   Score:  18.00   Max_steps :  17
Episode 300     Average Score: 42.622   Score:  59.00   Max_steps :  58
Episode 400     Average Score: 98.055   Score: 200.00   Max_steps : 199
Episode 500     Average Score: 113.16   Score: 110.00   Max_steps : 109
Episode 600     Average Score: 89.211   Score: 100.00   Max_steps :  99
Episode 700     Average Score: 94.122   Score:  85.00   Max_steps :  84
Episode 800     Average Score: 65.322   Score:  30.00   Max_steps :  29
Episode 900     Average Score: 44.555   Score:  10.00   Max_steps :   9
Episode 1000    Average Score: 24.555   Score:   9.00   Max_steps :   8
Elapsed : 0:03:46.871175
2019-02-02 09:18:48.363542
```

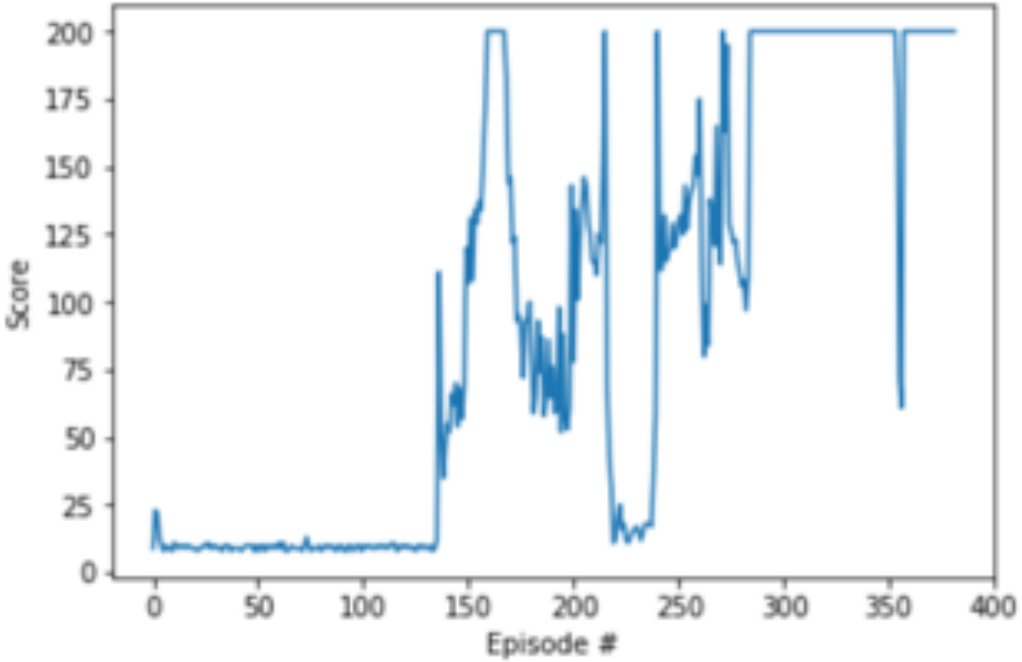




```
Episode 100     Average Score: 14.377   Score:  50.00   Max_steps :  49
Episode 200     Average Score: 162.22   Score: 200.00   Max_steps : 199
Episode 230     Average Score: 195.04   Score: 200.00   Max_steps : 199
Environment solved in 130 episodes!     Average Score: 195.04
Elapsed : 0:01:16.801558
2019-02-08 12:59:29.923779
```



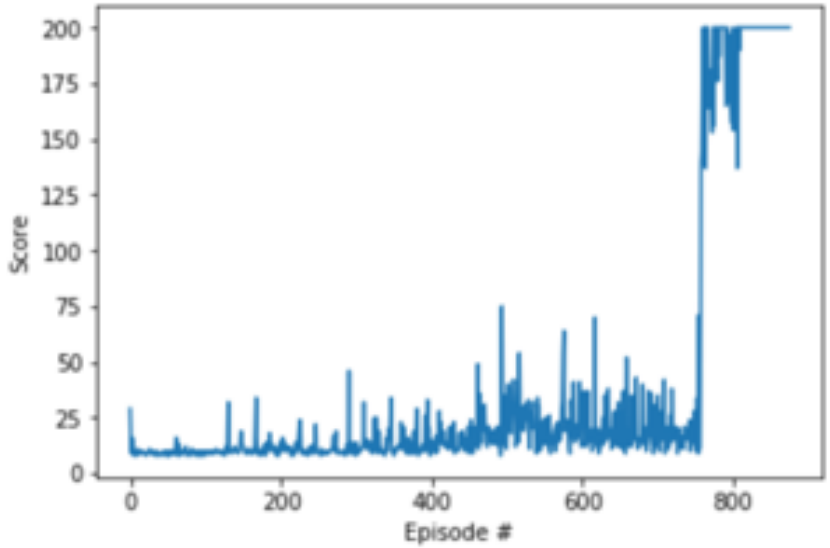
```
Episode 100     Average Score: 9.61     Score: 8.00     Max_steps :   79
Episode 200     Average Score: 71.03    Score: 143.00   Max_steps : 142
Episode 300     Average Score: 114.96   Score: 200.00   Max_steps : 199
Episode 382     Average Score: 195.17   Score: 200.00   Max_steps : 199
Environment solved in 282 episodes!     Average Score: 195.17
Elapsed : 0:02:29.415469
2019-01-13 18:14:48.885969
```



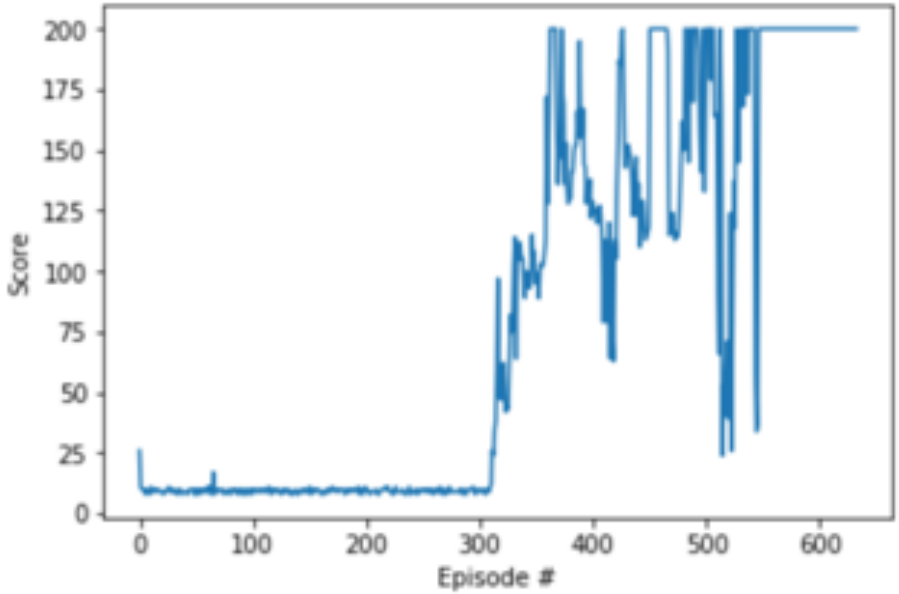
Perfect Test Score !

```
Episode :  1 Score : 200.00 Steps : 199
Episode :  2 Score : 200.00 Steps : 199
Episode :  3 Score : 200.00 Steps : 199
Episode :  4 Score : 200.00 Steps : 199
Episode :  5 Score : 200.00 Steps : 199
Episode :  6 Score : 200.00 Steps : 199
Episode :  7 Score : 200.00 Steps : 199
Episode :  8 Score : 200.00 Steps : 199
Episode :  9 Score : 200.00 Steps : 199
Episode : 10 Score : 200.00 Steps : 199
Mean of 10 episodes = 200.0
Elapsed : 0:00:00.517737
2019-01-13 18:18:07.700678
```

```
Episode 100     Average Score: 9.8282   Score:   8.00   Max_steps :   7
Episode 200     Average Score: 11.033   Score:  14.00   Max_steps :  13
Episode 300     Average Score: 11.300   Score:   8.00   Max_steps :   7
Episode 400     Average Score: 13.588   Score:  11.00   Max_steps :  10
Episode 500     Average Score: 16.977   Score:  11.00   Max_steps :  10
Episode 600     Average Score: 22.666   Score:  26.00   Max_steps :  25
Episode 700     Average Score: 20.444   Score:  12.00   Max_steps :  11
Episode 800     Average Score: 89.944   Score: 200.00   Max_steps : 199
Episode 875     Average Score: 195.33   Score: 200.00   Max_steps : 199
Environment solved in 775 episodes!     Average Score: 195.33
Elapsed : 0:02:12.733915
2019-01-18 09:19:16.580979
```

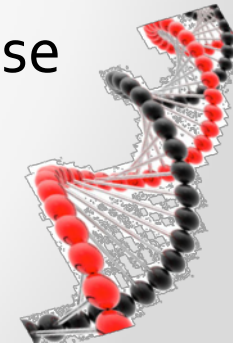


```
Episode 100     Average Score: 9.5858    Score:   8.00    Max_steps :   7
Episode 200     Average Score: 9.4444    Score:  11.00    Max_steps :  10
Episode 300     Average Score: 9.3535    Score:  10.00    Max_steps :   9
Episode 400     Average Score: 104.71    Score: 129.00    Max_steps : 128
Episode 500     Average Score: 149.30    Score: 170.00    Max_steps : 169
Episode 600     Average Score: 173.89    Score: 200.00    Max_steps : 199
Episode 633     Average Score: 195.01    Score: 200.00    Max_steps : 199
Environment solved in 533 episodes!     Average Score: 195.01
Elapsed : 0:03:13.291465
2019-02-02 09:44:52.322328
```

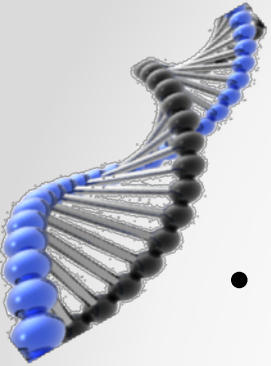


Lessons from Implementation

1. Ornstein-Uhlenbeck process for exploration noise !
   - ".. Deterministic policy gradient might not explore the full state and action space"
   - Need a noise process to encourage exploration : $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
   - "Generates temporally correlated exploration"
   - "Exploration efficiency with inertia"
2. Unlearn & diverge / slow
   - We can see what Lillicrap, Silver* et al refers to as "*instability in learning wherein progress on a problem is either destroyed by subsequent learning updates, or else learning is too slow to be practical*"
3. A good solution for the cartpole
4. Very sensitive to hyperparameters – may be because DDPG is an overkill for the cartpole
5. Remember to restart the Kernel between tries

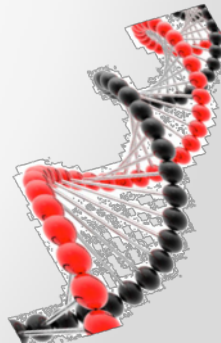


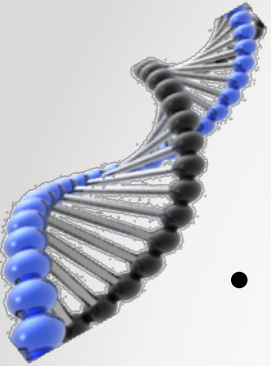
*https://arxiv.org/pdf/1509.02971.pdf



Quiz

- What is the DQN DNN topology ?
  -
- What is the input size ? Output size ?
  -
- What is the output of the actor network ?
  -
- How does the output translate to an action ?
  -
- Is the policy deterministic or stochastic ?
  -
- How do you characterize this w.r.t others ?
  -

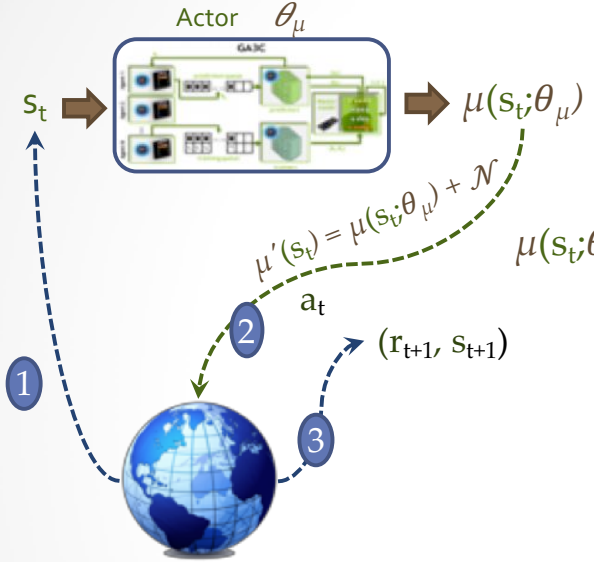


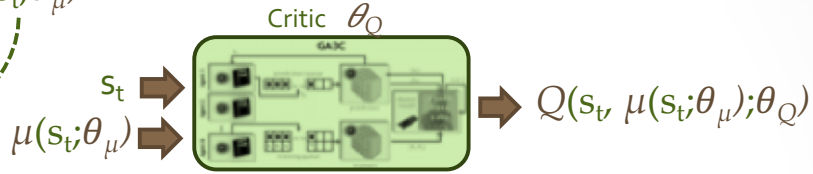


Quiz

- What is the DQN DNN topology ?
  - FC16-FC8-FC2
- What is the input size ? Output size ?
  - Input = 4 (state), output = 2 (actions)
- What is the output of the actor network ?
  - softmax
- How does the output translate to an action ?
  - sample
- Is the policy deterministic or stochastic ?
  - Stochastic ! We get it as a part of the architecture !
- How do you characterize this w.r.t others ?
  - A good solution ! But sensitive to hyperparameters – [probably an overkill

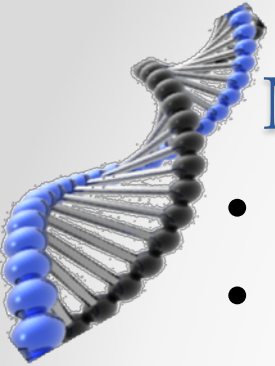
Notice the Critic Network





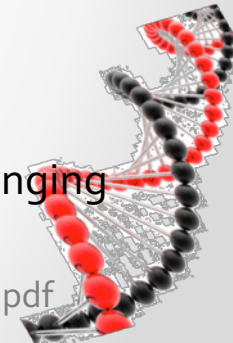

```
Actor(
  (model): Sequential(
    (0): Linear(in_features=4, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=2, bias=True)
    (5): Softmax()
  )
)
Critic(
  (hc_1): Sequential(
    (0): Linear(in_features=4, out_features=16, bias=True)
    (1): ReLU()
  )
  (hc_2): Sequential(
    (0): Linear(in_features=18, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

The actions are fed into the network after the FC16 layer

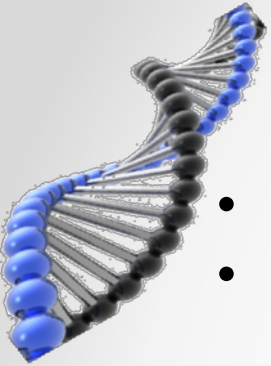


Natural Policy Gradients & Trust Region variants

- Policy Gradients tend to get stuck in local plateaus
- Natural Policy Gradients operate in a different space than the parameter space, the Fisher metric
  - Instead of following the usual steepest direction in the parameter space, NPG algorithms follow the steepest direction with respect to the Fisher metric
  - The caveat with natural gradients is that, in the case of neural networks and their large number of parameters, it is usually impractical to compute, invert, and store the Fisher information matrix
- Trust Region algorithms leverage the natural gradients
  - As a modification to the natural gradient method, policy optimization methods based on a trust region aim at improving the policy while changing it in a controlled way.

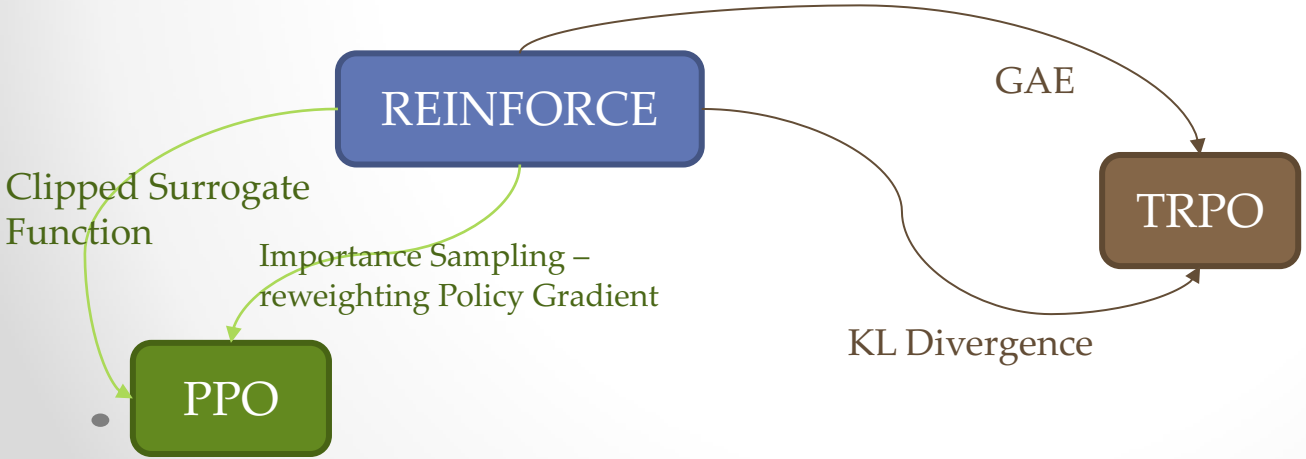


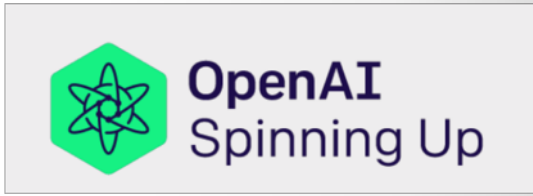
https://arxiv.org/pdf/1811.12560.pdf

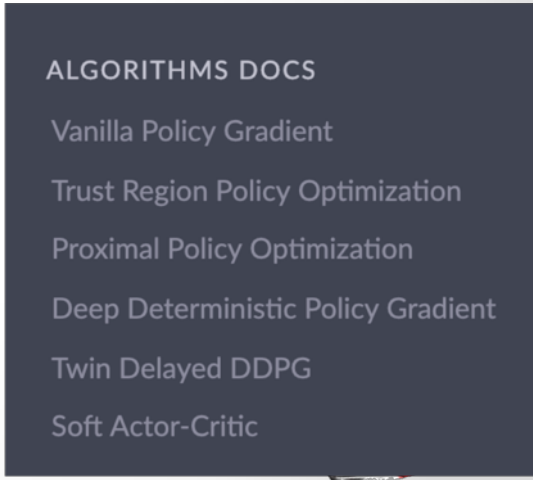


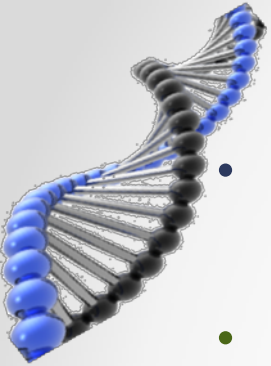
PPO & TRPO

- Another interesting set of ideas that have become mainstream
- "*PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance*"
  - https://blog.openai.com/openai-baselines-ppo/
  - PPO paper https://arxiv.org/abs/1707.06347
- TRPO
  - https://spinningup.openai.com/en/latest/algorithms/trpo.html



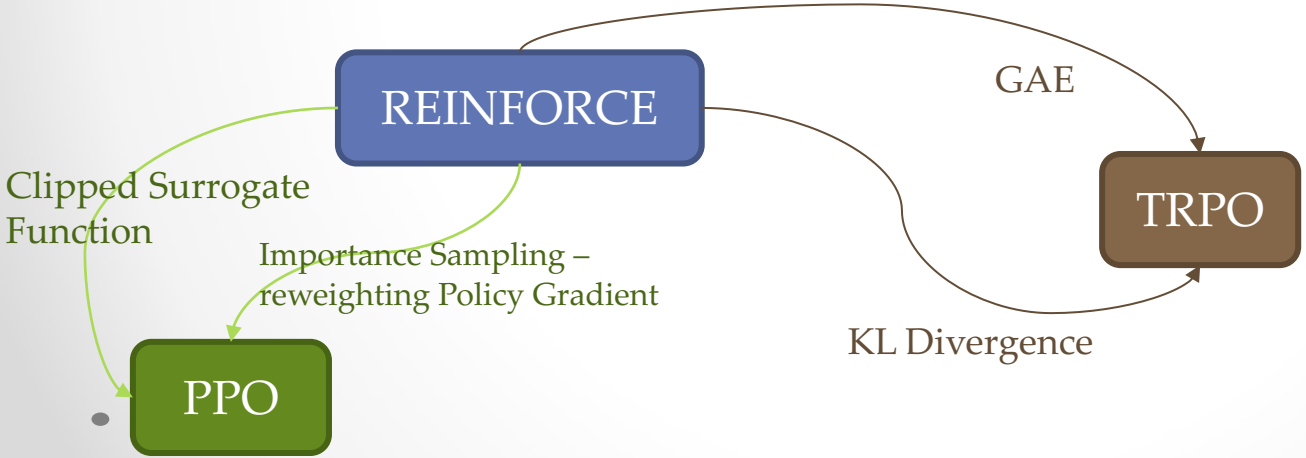


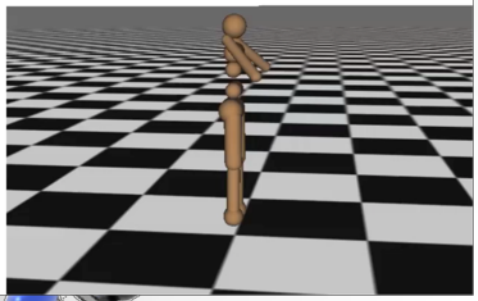




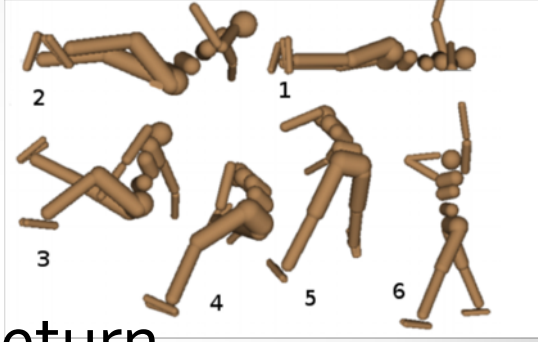
PPO & TRPO

- These constraint-based policy optimization methods focus on restricting the changes in a policy using mechanisms like the KL divergence between the action distributions
- TRPO uses constrained updates and advantage function estimation to perform the update, resulting in the reformulated optimization
- Proximal Policy Optimization (PPO)is a variant of the TRPO algorithm, which formulates the constraint as a penalty or a clipping objective, instead of using the KL constraint





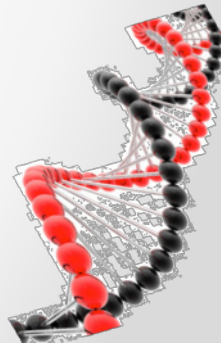
GAE



- GAE (is a mechanism) – mixture of lambda return. Difficult to say which n is better
  - Combination of all n step weighted by exponentially decaying lambda
- High-Dimensional Continuous Control Using Generalized Advantage Estimation https://arxiv.org/abs/1506.02438
- Learning LocomoQon (TRPO + GAE)
  - Schulman, Moritz, Levine, Jordan, Abbeel, 2016

An interesting paper experiments with the agent's physical structure as a parameter

- Reinforcement Learning for Improving Agent Design https://arxiv.org/pdf/1810.03779.pdf



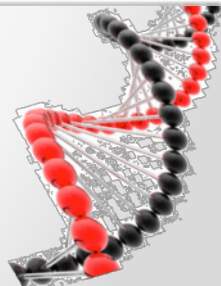
Actor-Critic with A3C or GAE

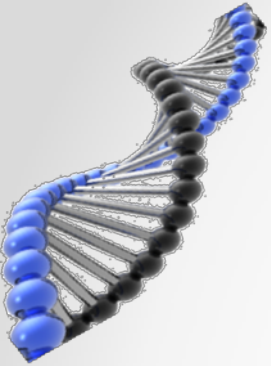
- Policy Gradient + Generalized Advantage Estimation:
  - Init π_{θ_0} $V^{\pi}_{\phi_0}$
  - Collect roll-outs {s, u, s', r} and $\hat{Q}_i(s,u)$
  - Update: $\phi_{i+1} \leftarrow \min_{\phi} \sum_{(s,u,s',r)} \|\hat{Q}_i(s,u) - V^{\pi}_{\phi}(s)\|_2^2 + \kappa\|\phi - \phi_i\|_2^2$

$$\theta_{i+1} \leftarrow \theta_i + \alpha \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_{\theta_i}(u_t^{(k)}|s_t^{(k)}) \left(\hat{Q}_i(s_t^{(k)}, u_t^{(k)}) - V^{\pi}_{\phi_i}(s_t^{(k)})\right)$$

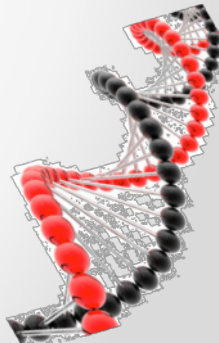
Async Advantage Actor Critic (A3C) *[Mnih et al, 2016]*

one of the above choices (e.g. k=5 step lookahead)

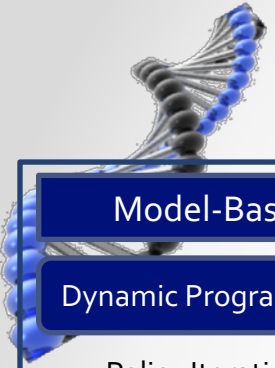




6. Conclusion



RL Algorithm Taxonomy

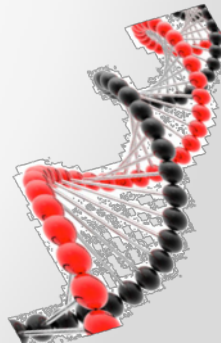


- Model-Based
  - Dynamic Programming
    - Policy Iteration
    - Value Iteration
- Model-Free
  - Monte Carlo
  - TD Learning
  - Value Methods
    - SARSA
    - Q-Learning
    - Deep Q-Learning
      - Fixed Q Targets
      - Experience replay
      - Double DQN
      - Prioritized replay
      - Dueling DQN
      - *Increase robustness*
      - *Reduce bias/variance*
      - *Sample efficiency*
  - Policy Search
    - Hill Climbing
    - Steepest Hill Climbing
      - Cross-entropy
      - Evolution Strategies
    - Random Restart
    - Simulated Annealing
    - Adaptive noise scaling
  - Policy Gradients
    - *Natural Policy Gradients*
    - REINFORCE
    - TRPO
    - Actor-Critic
    - PPO
    - DDPG

RL ⟷ DRL

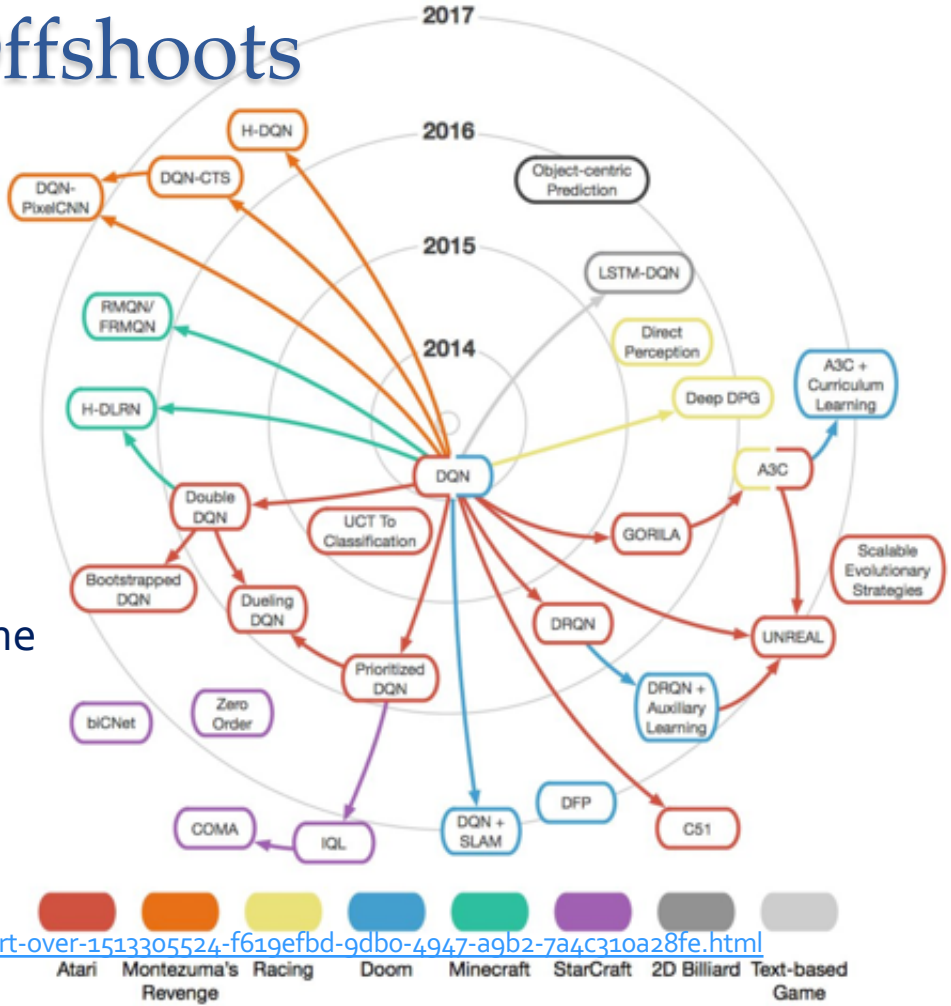
Non-linear Function Approximation

- Why Policy Based ?
  - Simplicity
  - Stochastic Action Space
    - Rock-Paper-scissor
  - Continuous
    - Can output a # without exploring values in a continuous space
  - Aliased States

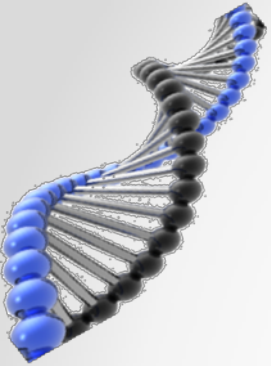


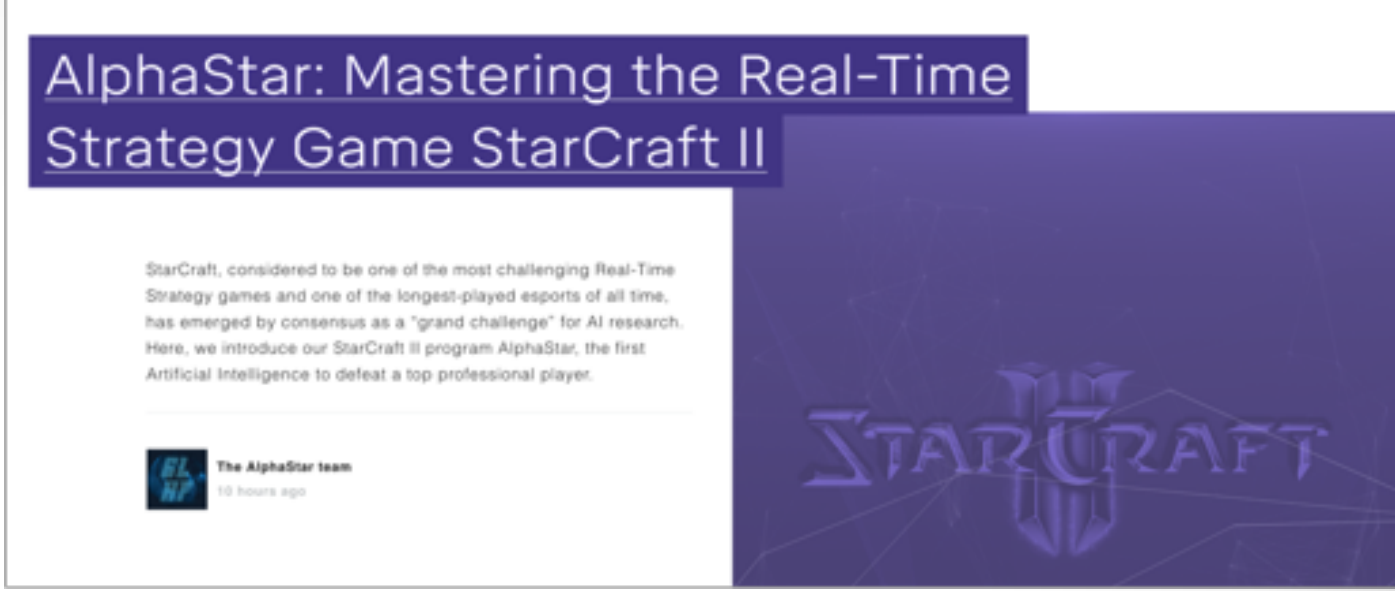
Influences & Offshoots

- My view is throw it all away and start again,…
- Max Planck said, 'Science progresses one funeral at a time'
- *The future depends on some graduate student who is deeply suspicious of everything I have said*.

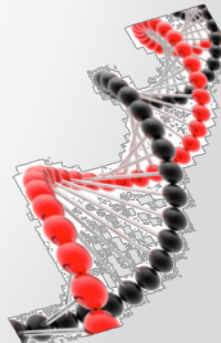


https://www.axios.com/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524-f619efbd-9db0-4947-a9b2-7a4c310a28fe.html
https://arxiv.org/abs/1708.07902





Summary, technologies and networks



- https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/
  https://www.theverge.com/2019/1/24/18196135/google-deepmind-ai-starcraft-2-victory

We use our visual system not only to survive & navigate, but also to socialize, entertain, understand & learn the world @drfeifei

Can a machine understand the nuances and humor in this picture ?

I had a good time researching & preparing for this Tutorial.

I hope you learned a few new things & inspired to follow a few vectors