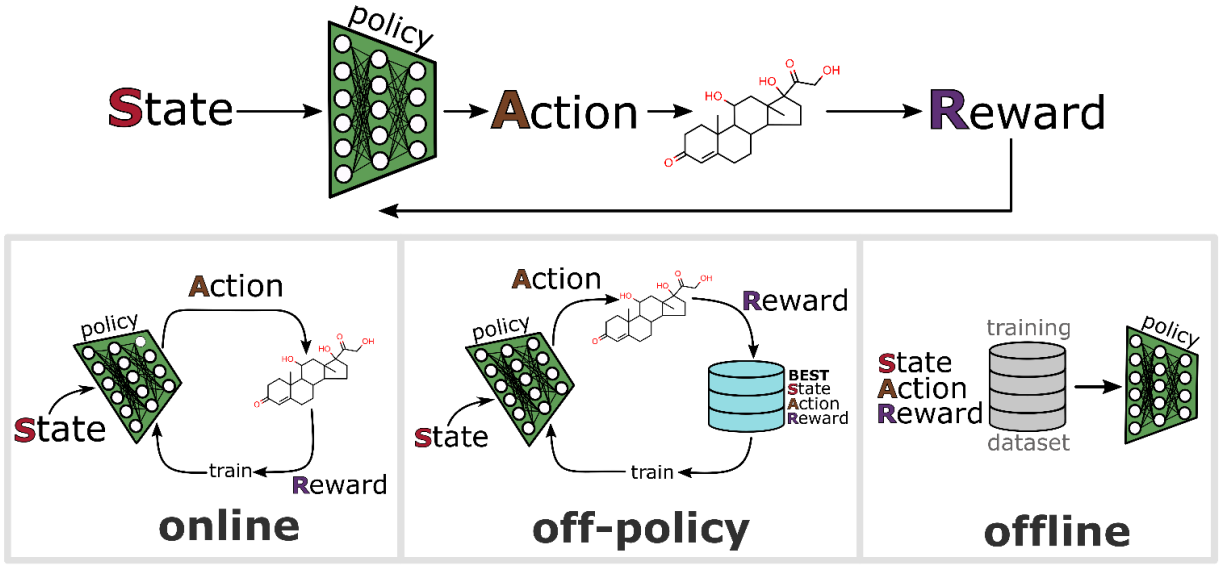
An Introduction on Reinforcement Learning with Examples from Chemical Science

Dylan M. Anstine, Ph.D. Materials Science and Engineering

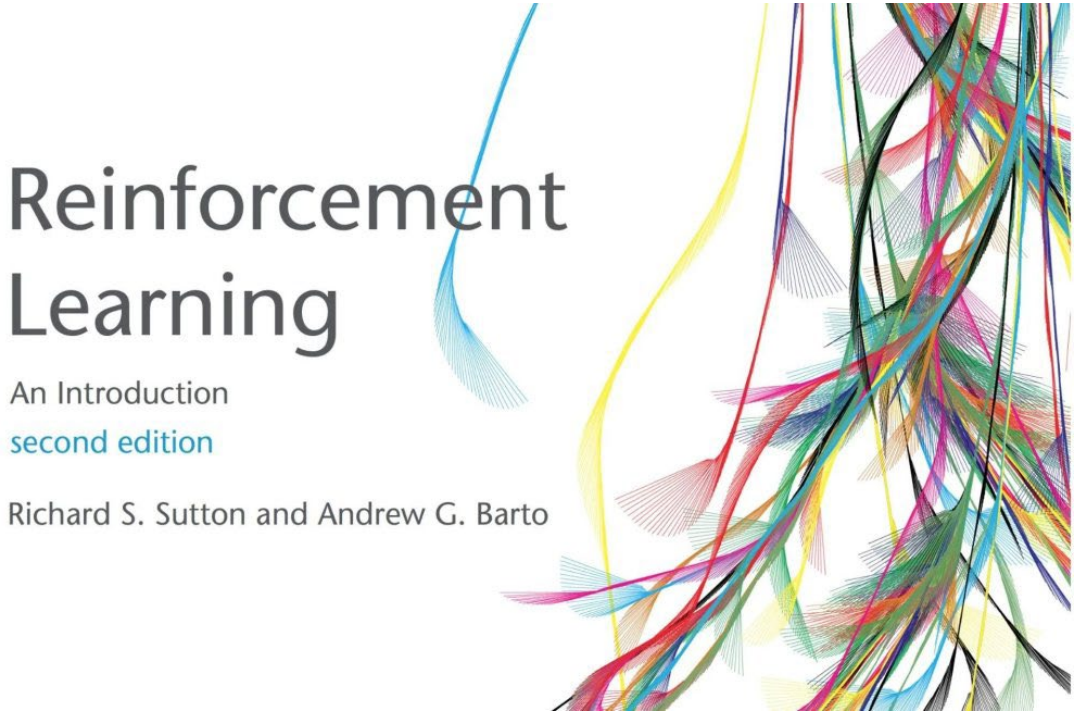
Postdoctoral Fellow, Isayev Research Group

Carnegie Mellon University



Welcome to Reinforcement Learning

- Reinforcement learning has been an active field of study since the 1970s.
  - Computer Science
  - Statistics
  - Psychology
  - Philosophy
  - Ethics
  - Machine Learning



Today we will be talking specifically about a *computational approach* to learning from interaction, which is the underpinning of all reinforcement learning.

A Demonstration of Reinforcement Learning

Deep reinforcement learning is a powerful solution to problems where the goal is clearly defined but algorithmically challenging to implement.

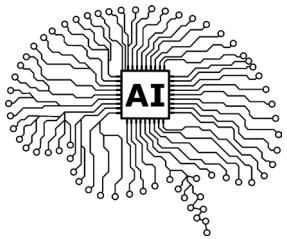
https://openai.com/research/emergent-tool-use

Consider a Few Scenarios

1. “When a gazelle is born it struggles for a few minutes to get to its feet. Half an hour later it is running 20 mph.”

2. A dog that burns its paw on a hot stove knows not to go near the stove again. The dog doesn’t know anything about thermodynamics or the kinetics of heat transfer, but it has acquired an ability to act according to their implications.

3. A machine learning model that learns to play the board game go. Despite there being 10^{360} possible moves, the model can achieve superhuman performance.



In RL, we use the term **agent** to refer to the entity that interacts with the environment. We will describe what *interacts with* means on the next slide.

Let's Tidy up our Language

- "A high-level definition of reinforcement learning (RL) is that it is a framework to describe the process of improving ability through interactions between **states**, **actions**, and **rewards**."
- Improving ability → Achieving a Quantifiable Goal
  - **State:** The set of conditions the agent is in at a particular instance.
  - **Actions:** The set of allowable behaviors that the agent can select from.
  - **Reward:** Is a numeric signal that reflects *how good* or *how bad* the agent is doing at interacting in the environment to achieve its goal.

Revisit: "A dog that burns its paw on a hot stove knows not to go near the stove again. The dog doesn't know anything about thermodynamics or the kinetics of heat transfer, but it has acquired an ability to act according to their implications."

A Universal Goal

Reinforcement learning is based on the idea of the **reward hypothesis**

THE REWARD HYPOTHESIS

All goals can be described by maximization of expected cumulative reward

Do we agree with this idea?

"In this paper, we consider an alternative hypothesis: that the generic objective of maximising reward is enough to drive behaviour that exhibits most if not all abilities that are studied in natural and artificial intelligence."

Artificial Intelligence
Volume 299, October 2021, 103535

Reward is enough

David Silver, Satinder Singh, Doina Precup, Richard S. Sutton

The Nature of Rewards

Rewards can be either *discrete* or *continuous* signals

Flying a Drone Autonomously

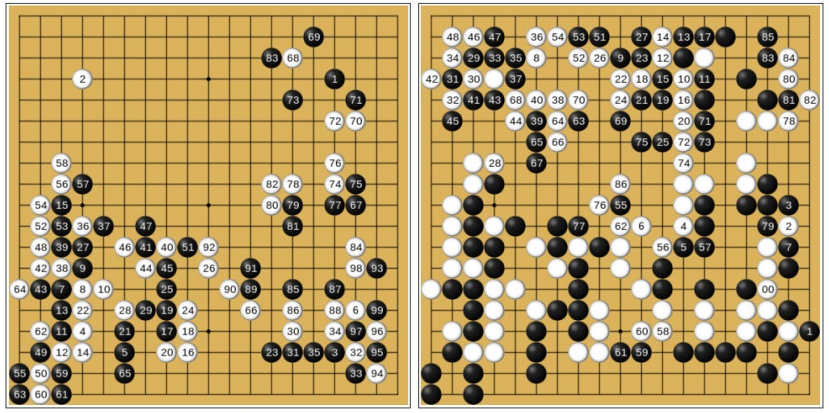
+Reward for following trajectory (RMSD)

-Reward for crashing

Playing Go against world champions

+Reward winning

Nothing for losing



First 99 moves

Moves 100–186 (149 at 131, 150 at 130)

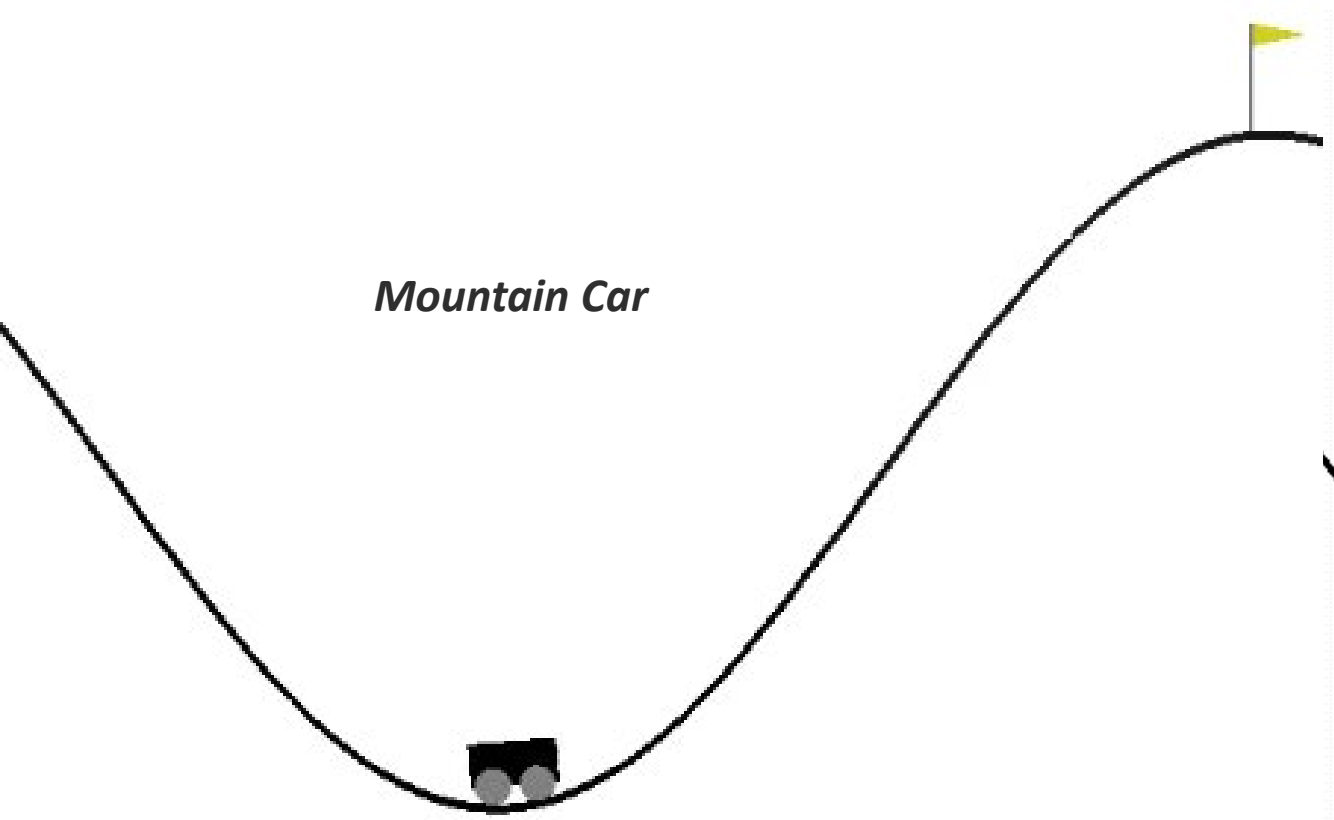
Managing Investments

+Reward for gaining money

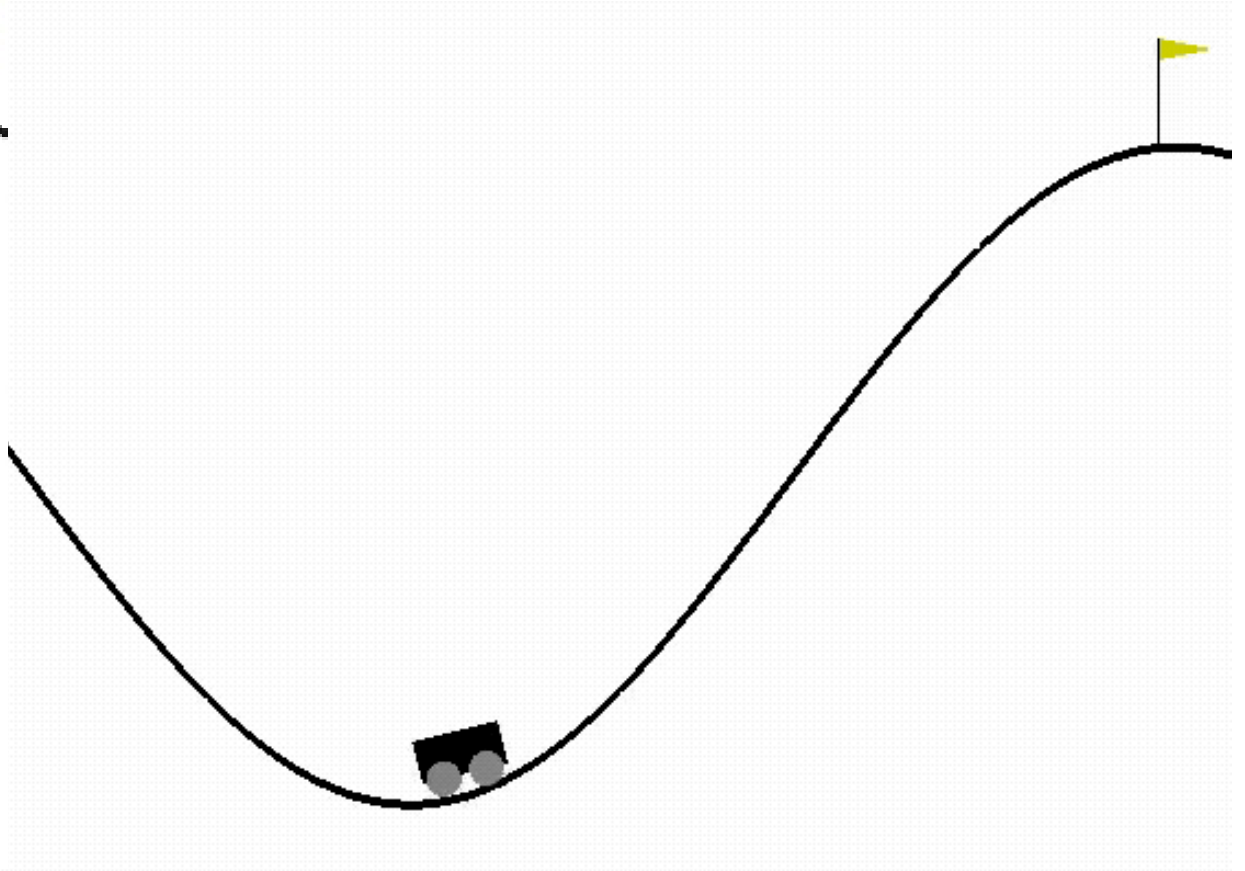
-Reward for going into debt

Delayed Rewards Require Foresight

Delayed rewards is a key concept defining modern RL problems.



Mountain Car



Sequential Decision Making

- The Agent's aim: select actions to maximize the total future reward.
- Actions may have long term consequences (delayed rewards).
- It may be better to sacrifice immediate reward to gain more long-term reward.

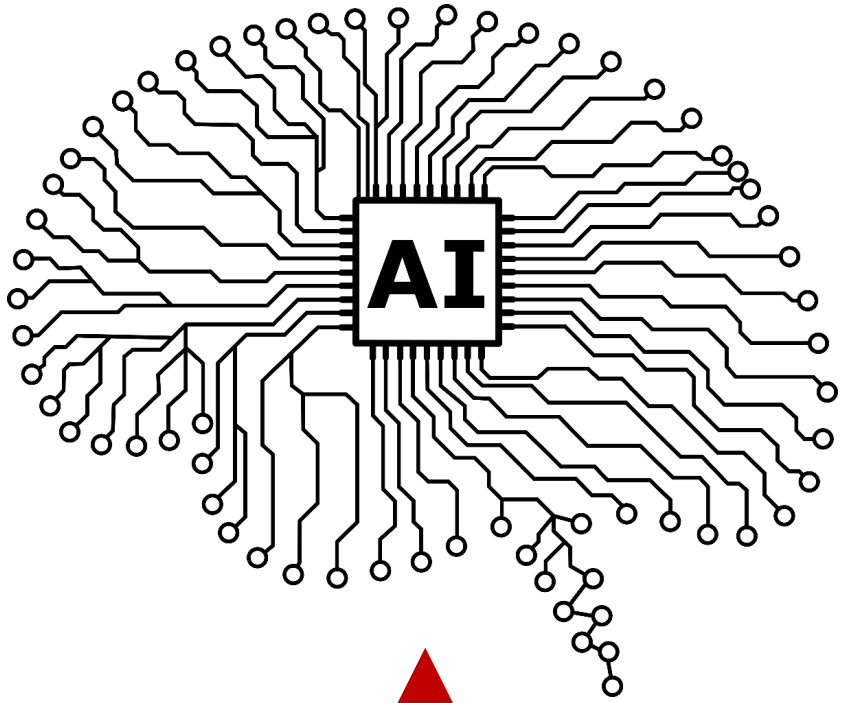
Examples

- A financial investment starts as a negative on your bank account but can bare long-term benefits.
- Refueling a car makes you stop your trip, but it is essential for making it to your destination.
- It can be better to foul your opponent in basketball at the end of the game to save time on the clock, even though this gives them the chance to shoot free throws.

The Agent in Isolation

Observation tells the agent about its state.

O_t



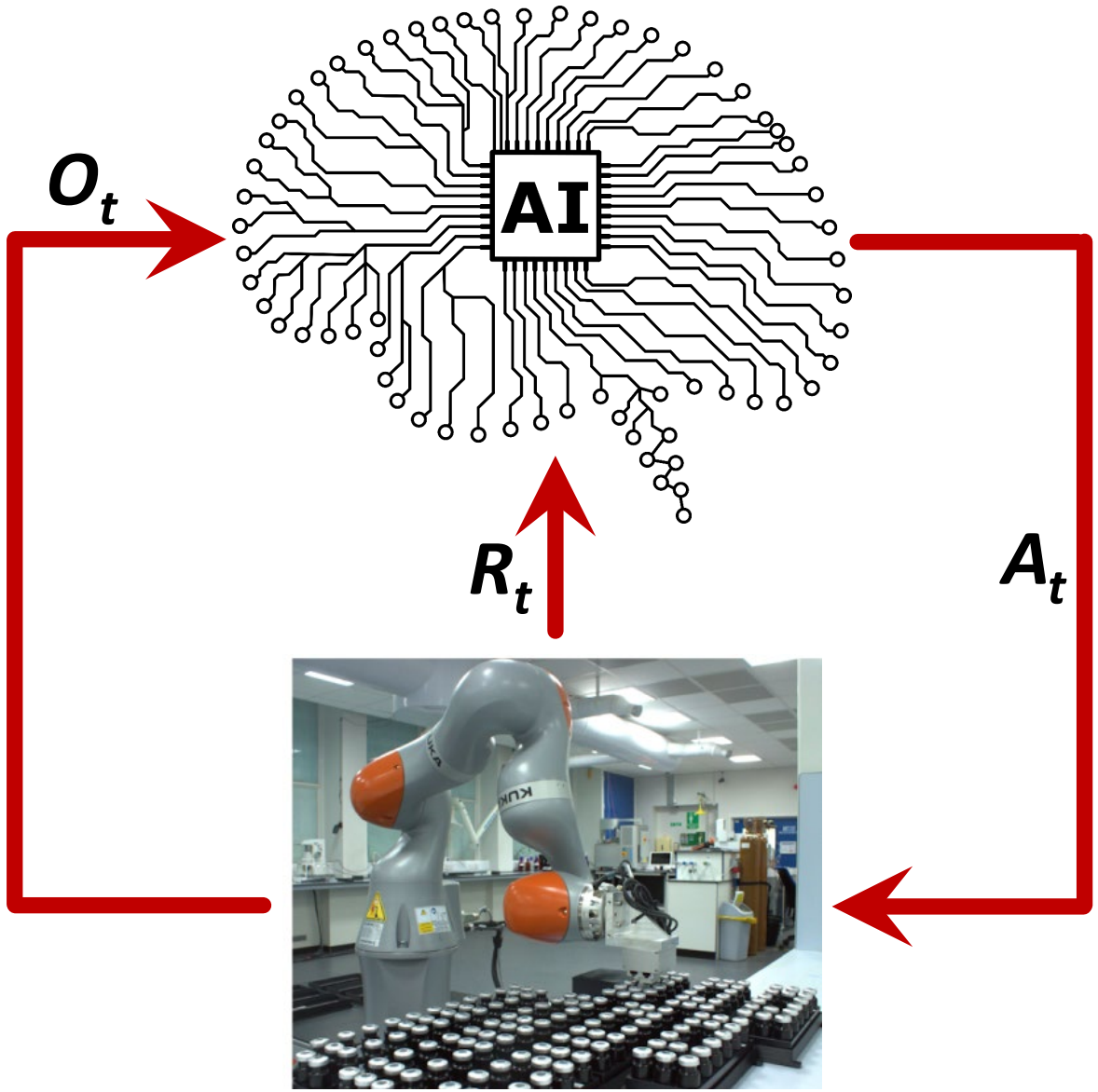
Action is selected to explore or exploit.

A_t

Reward Signal Comes in Depending on the Environment

R_t

Agent and the Environment



Burger *et al.*, *Nature*, **2020**, 583.

<u>At each step (*t*) the agent:</u>

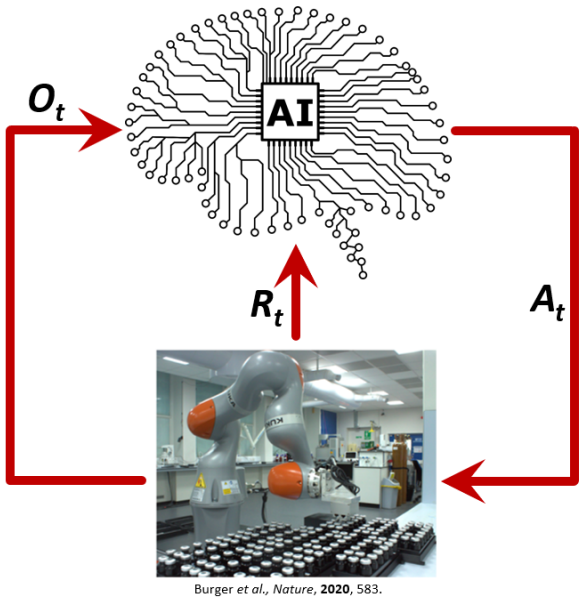
- Executes an action (A_t)
- Observes (O_t) its environment
- Receives a scalar reward (R_t)

<u>At each step (*t*) the environment:</u>

- Receives action (A_t)
- Emits observation (O_t)
- Emits a scalar reward (R_t)

Collectively, we refer to this loop and its history as the *experience*.

A More Formal Definition of S_t



Burger *et al.*, *Nature*, **2020**, 583.

The **history** in a reinforcement learning problem is the sequence of observations, rewards, and actions up to time t.

$$H_t = A_1, O_1, R_1, \ldots, A_t, O_t, R_t$$

i.e., this is all the observable variables that the agent knows about up to time t

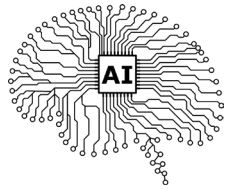
The state (S_t) is a summary of the history that is used to inform future actions.

$$S_t = f(H_t)$$

The next experience (H_{t+1}) depends on how the agent maps the state-action space.

$$H_{t+1} = \text{AI}(S_t)$$

Introduction of Markov States

$H_t = A_1, O_1, R_1, \ldots, A_t, O_t, R_t$ $S_t = f(H_t)$ $H_{t+1} =$  (S_t)

DEFINITION OF MARKOV STATE

A State S_t is said to be a Markov state if and only if

$$\mathbb{P}[S_{t+1}|\, S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

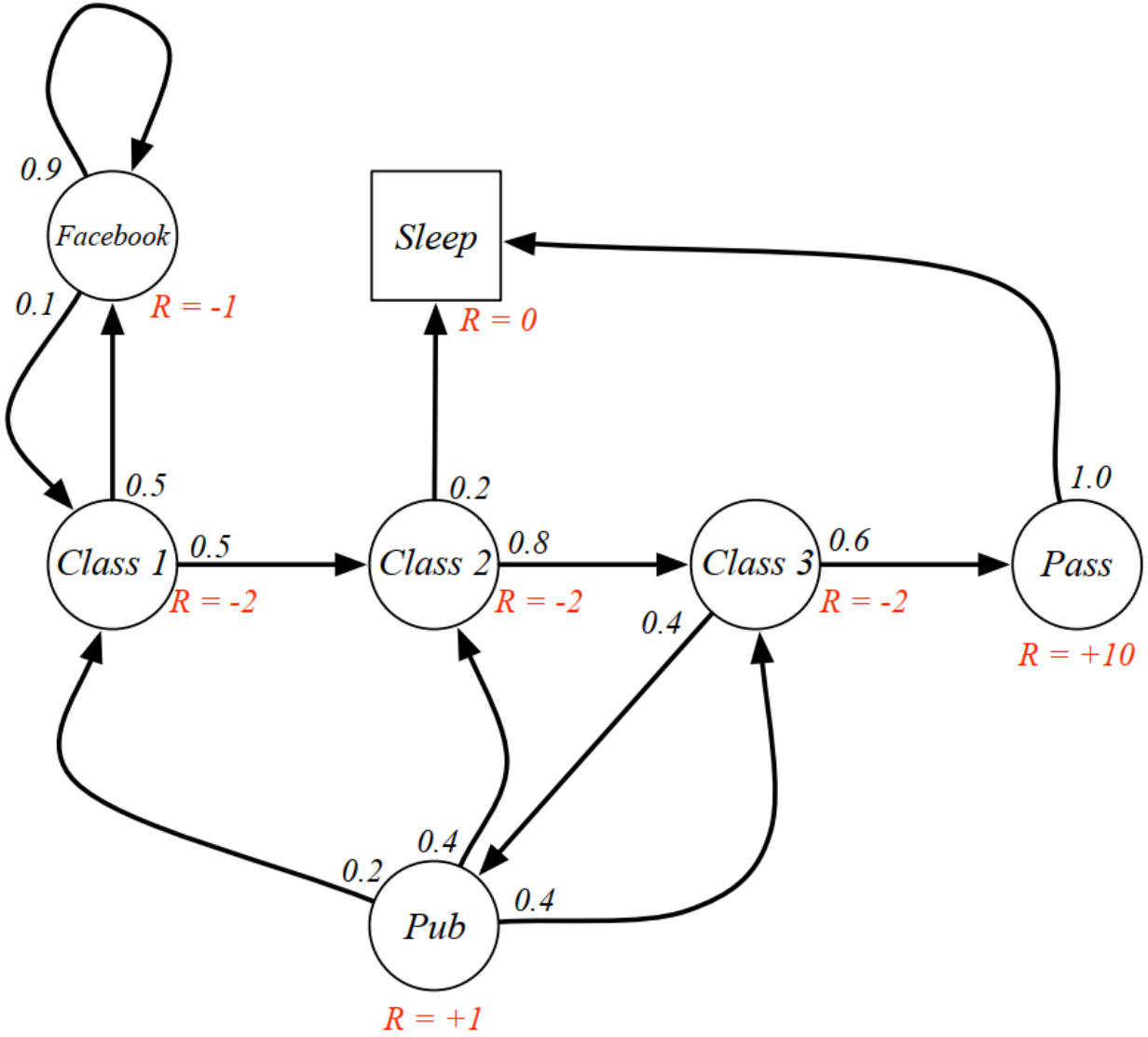
Colloquially: "the future depends only on the present state and not the past."

Why is this important for reinforcement learning?

You can throw away all the history that is not needed to define the current state because the current state is a sufficient statistic to describe the future

Markov Decision Process (MDP)

A **Markov Decision Process (MDP)** is a framework where operations are selected in sequence to transition between Markov states. Most, if not all, reinforcement learning problems can be formulated as MDPs.



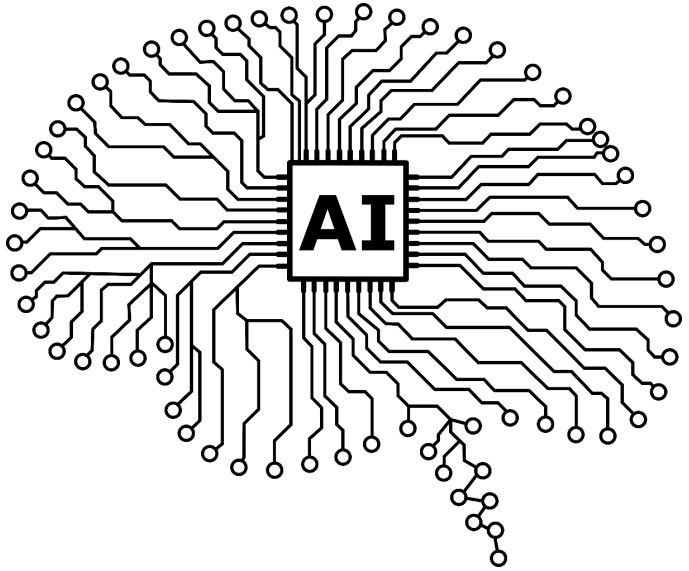
DEFINITION OF MARKOV STATE

A State S_t is said to be a Markov state if and only if

$$\mathbb{P}[S_{t+1}|\, S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$

Colloquially: "the future depends only on the present state and not the past."

What is our Agent made of?



- An RL Agent has one or more of the following components
  1. Policy: Agent's function for making decisions
  2. Value function: Agent's evaluator for how good or how bad each state and/or action is
  3. Model: Agent's representation of the environment

Policy (π) is a function that maps the state-action space

deterministic policy $A_t = \pi(S_t)$ *stochastic policy* $\pi(A_t|S_t)$

Value function (v_π) is a measure of future rewards based on the current state

$$v_\pi(S_t) = \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t)$$

A Quick Aside about the Language

RECALL

Reinforcement learning is based on the idea of the **reward hypothesis**

THE REWARD HYPOTHESIS

All goals can be described by maximization of expected cumulative reward

Reward → instantaneous feedback from the environment

$$R_t$$

Return → the discounted sum of rewards received through a sequence of actions to the end of episode.

$$G_t = \sum_{t=0}^{T} \gamma^t \, R_{t+1} | S_0$$

Value function → an evaluation for the expected return.

$$v_\pi(S_t) = \mathbb{E}(G_t | S_t)$$

The Famous Bellman Equation for MDPs

$$v_\pi(S_t) = \mathbb{E}(G_t|S_t)$$

$$= \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ \ldots |S_t)$$ *write returns in expanded form*

$$= \mathbb{E}(R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \ \ldots)|S_t)$$ *factor out a discount term (gamma)*

$$= \mathbb{E}(R_{t+1} + \gamma G_{t+1}|S_t)$$ *substitute future return at t+1*

$$= \mathbb{E}(R_{t+1} + \gamma\, v_\pi(S_{t+1})|S_t)$$ *apply the law of iterated expectations*

The result: the value function can be understood as the expectation of immediate reward received + the discounted value function at the next state.

From state-value to action-value definitions

Policy: Agent's function for making decisions

state-value function

$$\boldsymbol{v_\pi(S_t) = \mathbb{E}(G_t|S_t)}$$

If I am in state S and follow policy π then I can expect to get $v_\pi(S_t)$ total reward

Informally, "how good is it to be in state S"

action-value function

$$\boldsymbol{q_\pi(S_t, A_t) = \mathbb{E}(G_t|S_t, A_t)}$$

If I am in state S, take action A, and follow policy π then I can expect to get $q_\pi(S_t, A_t)$ total reward

Informally, "if I am in state S, how good is it to take action A"

The MDP (RL problem) is effectively solved once the *optimal policy* ($\boldsymbol{\pi^*}$) that maximizes the action-value function is found

$$\boldsymbol{q_{\pi*}(S_t, A_t) = \mathbb{E}(G_t|S_t, A_t)}$$

What does it mean to be optimal?

- For all MDPs there exists an optimal policy. Therefore, for all reinforcement learning problems formulated as an MDP there exists an optimal policy.
- There can be several policy's that have equivalent optimality.
  - For example, 3-point shooting in basketball
- Policy A is considered optimal over Policy B if the value function is greater than or equal at all possible states. $\pi_A \geq \pi_B \quad if \quad v_{\pi_A}(s) \geq v_{\pi_B}(s) \;\; for \;\; \forall s$

optimal policy $\pi_* \geq \pi \quad for \quad \forall \pi$

All optimal policies have the optimal state-value function $v_{\pi_*} = \; v_*(s)$

All optimal policies have the optimal action-value function $q_{\pi_*}(s,a) = \; q_*(s,a)$

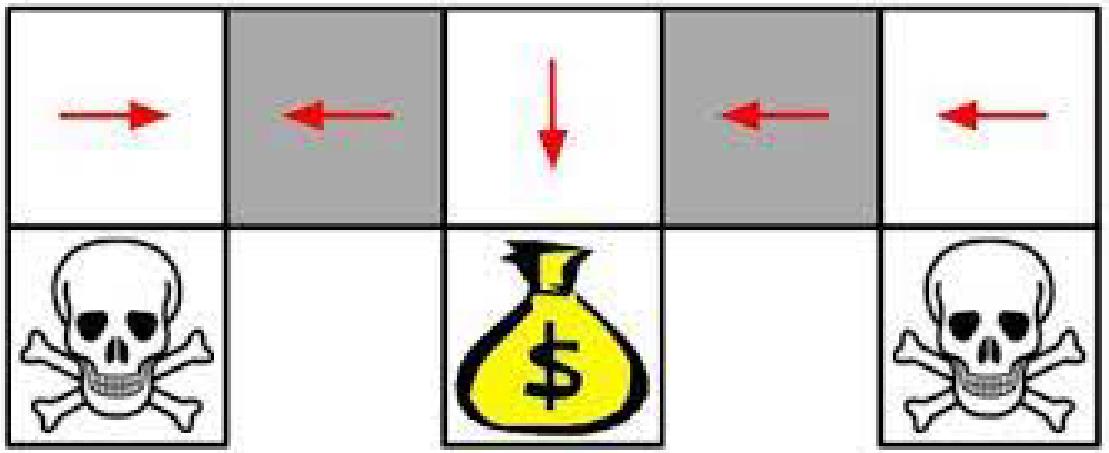
Stochastic Policy vs. Deterministic Policy

- The goal is for the agent to receive the money without falling into the pits.

$$state = (1,0,1,0)$$
$$(N,E,S,W)$$

- The agent cannot distinguish the grey states from each other.

Aliased Gridworld



compare **deterministic policy** vs. **stochastic policy**

π: can approximately do one thing either go east on all grey squares or go west.

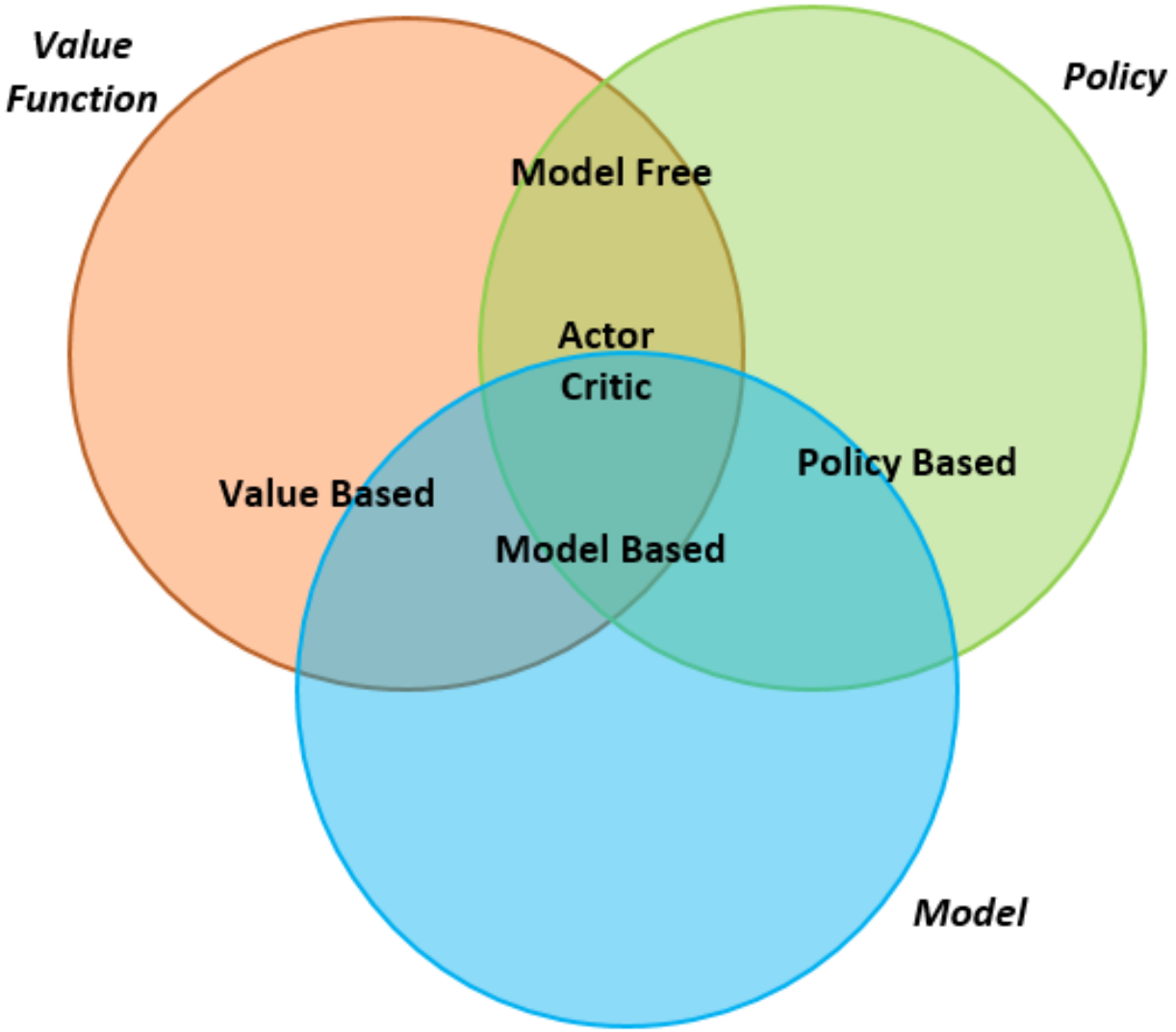
π: can choose to either east or west with 50% probability → converges to a solution

Questions?

At this point we have seen the following concepts

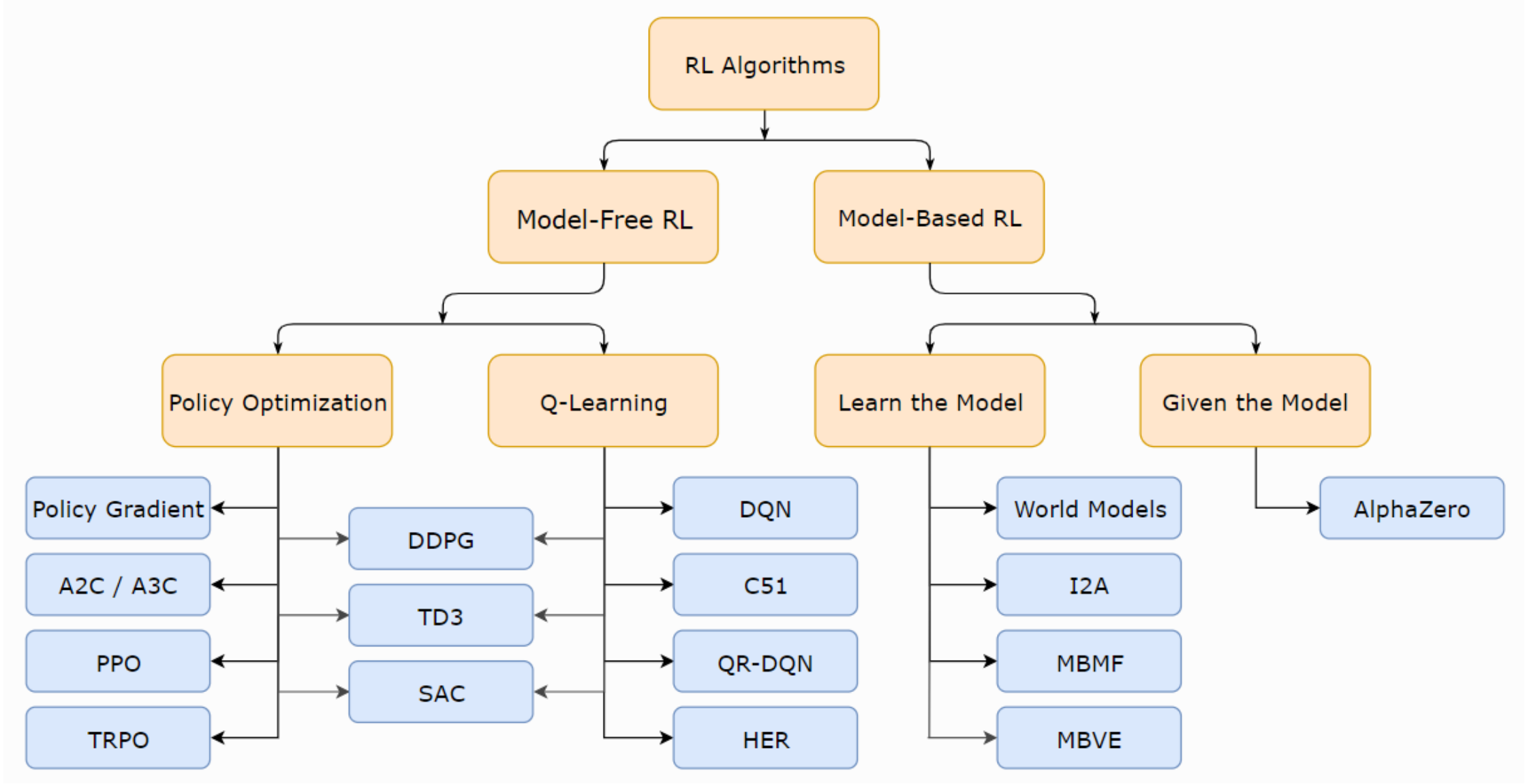
- Agent
- Environment
- States
- Actions
- Rewards
- The reward hypothesis
- Delayed Rewards
- History and Experience
- Returns
- Markov States and Markov Decision Processes
- Policy and values
- Bellman equation
- State-value and Policy-value
- Policy Optimality
- Stochastic vs. Deterministic Policies

How Do we Solve Reinforcement Learning Problems?



- In value-based Reinforcement Learning, we do not store an explicit policy. We seek to learn the value function.
- In policy-based Reinforcement Learning, we explicitly learn the mapping between states and actions using rewards to refine the agent.
- Actor-critic methods are a mix of the two.

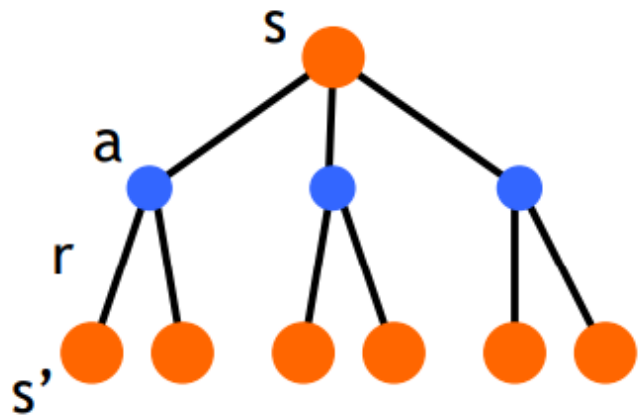
A Non-Exhaustive Taxonomy of Approaches



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Tabular Q-learning

Recall: $Q_\pi(s_t, a_t) = \mathbb{E}(R_{t+1} + \gamma\, Q_\pi(s_{t+1})|s_t, a_t)$



One natural solution to carry out reinforcement learning is iteratively search for the optimal action-value function (Q_*).

I.e., initialize a table of Q values for all state-action possibilities, take trial-and-error moves and update Q depending on the Bellman equation.

Learning Rate

Discount Factor

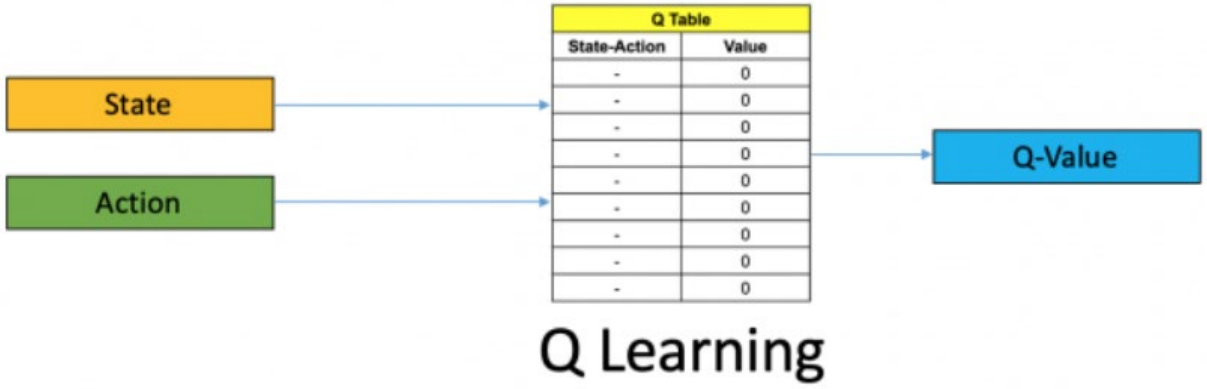
$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

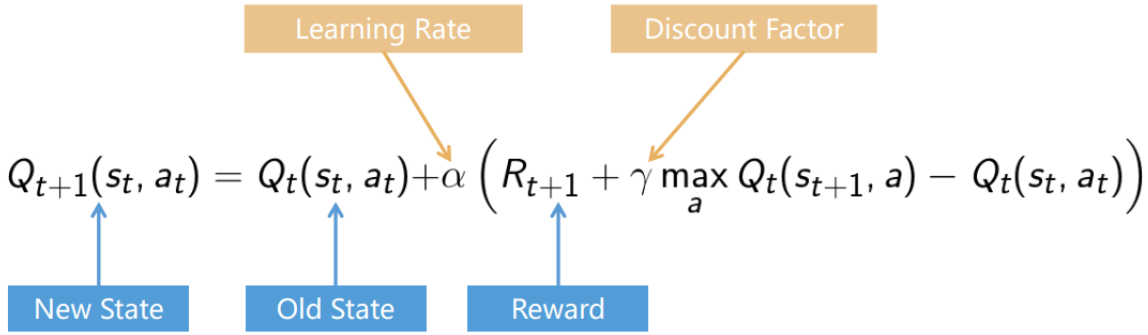
New State

Old State

Reward

How do we use TQL & what are the problems?





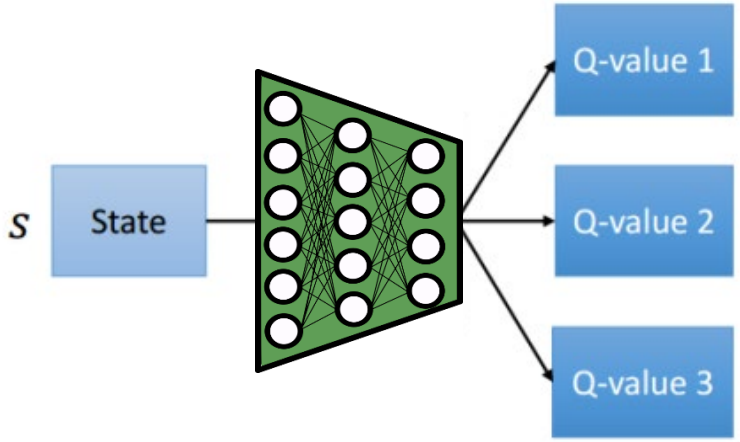
	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

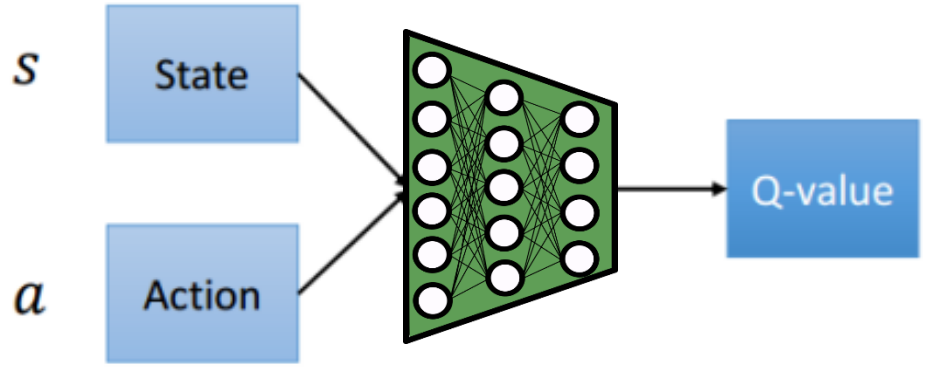
Once we have an optimally parameterized Q table, we can always take the action with the highest Q value. This is the optimal policy

What is the biggest problem with tabular Q learning (TQL)?

Deep Q Learning

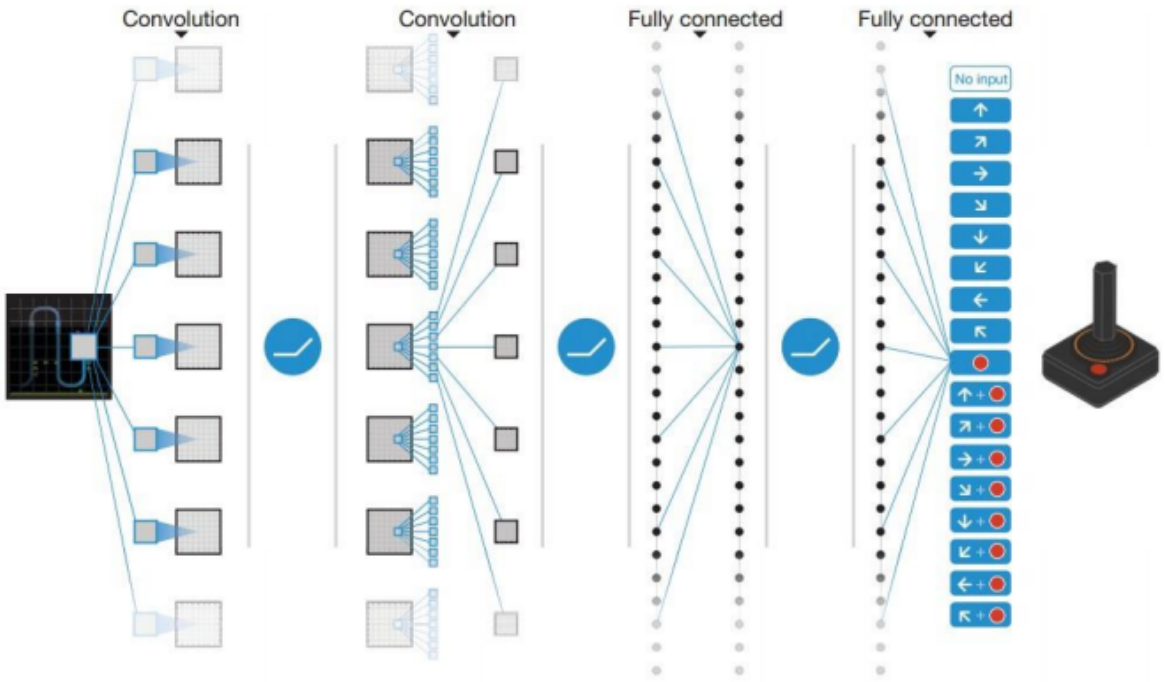
Instead of filling a Q table, use a neural network to directly approximate the action-value function.





1. State space can be continuous and large
2. Action space can be continuous and large
3. We are no longer limited by having to fit a q-table into memory
4. Better learning through generalization

Deep Q Learning



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Catherine Shu @catherineshu / 8:20 PM EST • January 26, 2014 Comment

Mnih et al. "Playing atari with deep reinforcement learning." 2013.

Example of Zhou *et al.*

Optimization of Molecules via Deep Reinforcement Learning

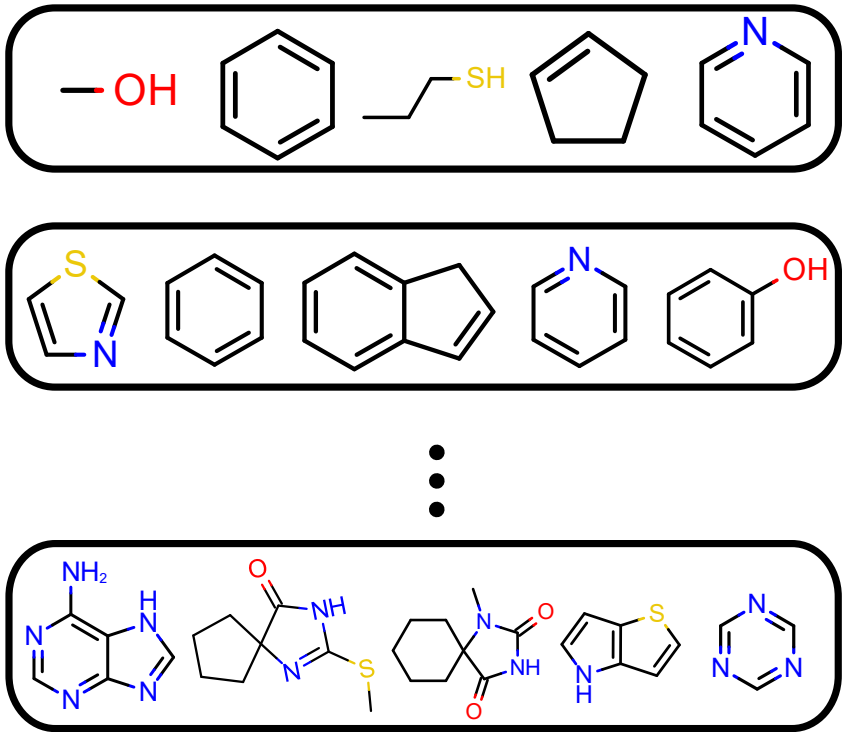
Zhenpeng Zhou[1,3], Steven Kearnes[2], Li Li[2], Richard N. Zare[1] & Patrick Riley[2]

We present a framework, which we call Molecule Deep *Q*-Networks (MolDQN), for molecule optimization by combining domain knowledge of chemistry and state-of-the-art reinforcement learning techniques (double *Q*-learning and randomized value functions). We directly define modifications on molecules, thereby ensuring 100% chemical validity. Further, we operate without pre-training on any dataset to avoid possible bias from the choice of that set. MolDQN achieves comparable or better performance against several other recently published algorithms for benchmark molecular optimization tasks. However, we also argue that many of these tasks are not representative of real optimization problems in drug discovery. Inspired by problems faced during medicinal chemistry lead optimization, we extend our model with multi-objective reinforcement learning, which maximizes drug-likeness while maintaining similarity to the original molecule. We further show the path through chemical space to achieve optimization for a molecule to understand how the model works.

Zhou, Z., Kearnes, S., Li, L. *et al.* Optimization of Molecules via Deep Reinforcement Learning. *Sci Rep* **9**, 10752 (2019). https://doi.org/10.1038/s41598-019-47148-x

The goal of molecular optimization

<u>Molecular optimization</u>
given an initial molecule, what changes need to be made to achieve the target properties



Starting Structures

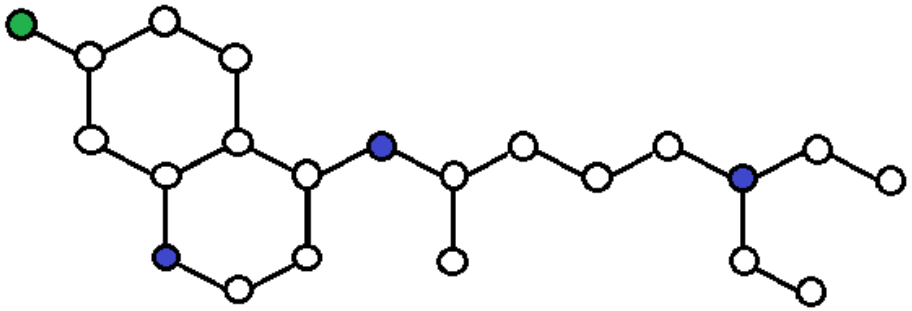
molecular edits

Structures Optimized for Target Property

<u>Question</u>
How do we formulate this as a Markov Decision Process (MDP)?

Molecular Optimization as an MDP

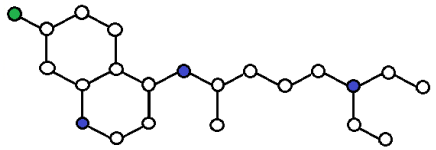
Treat Molecules as Graph Objects
(nodes connected by edges)



State is defined as a tuple {m, t}

m is the molecule

t is the edit step number {0:T}

$s_n = \{$  $: t\}$

The action-value function
assumes the same familiar form

$$Q^{\pi}(s,a) = Q^{\pi}(m,t,a) = \mathbb{E}_{\pi}[\sum_{n=t}^{T} r_n]$$

The action space $a \in \mathcal{A}$ is
defined by the possible edits that
lead to a valid molecule

1. **Add bond**
2. **Delete bond**
3. **Add new atom bonded to m**

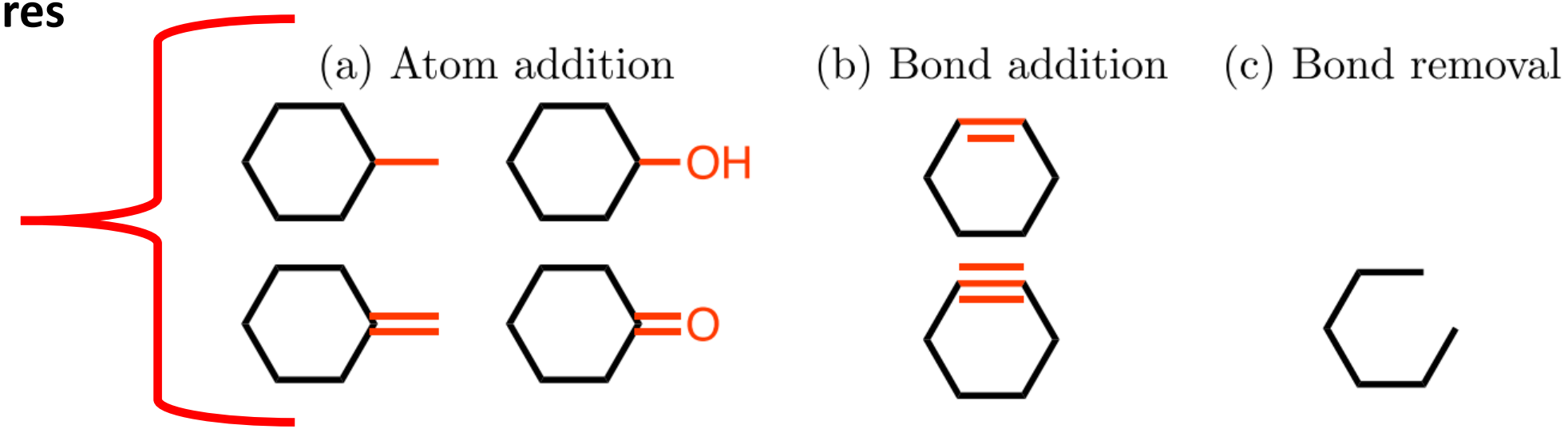
Understanding the Action Space

Possible actions for cyclohexane with an element space of {C,O}

Starting Structures

molecular edits

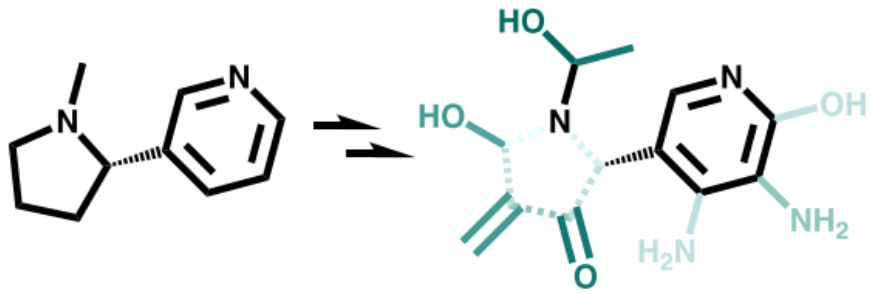
Structures Optimized for Target Property

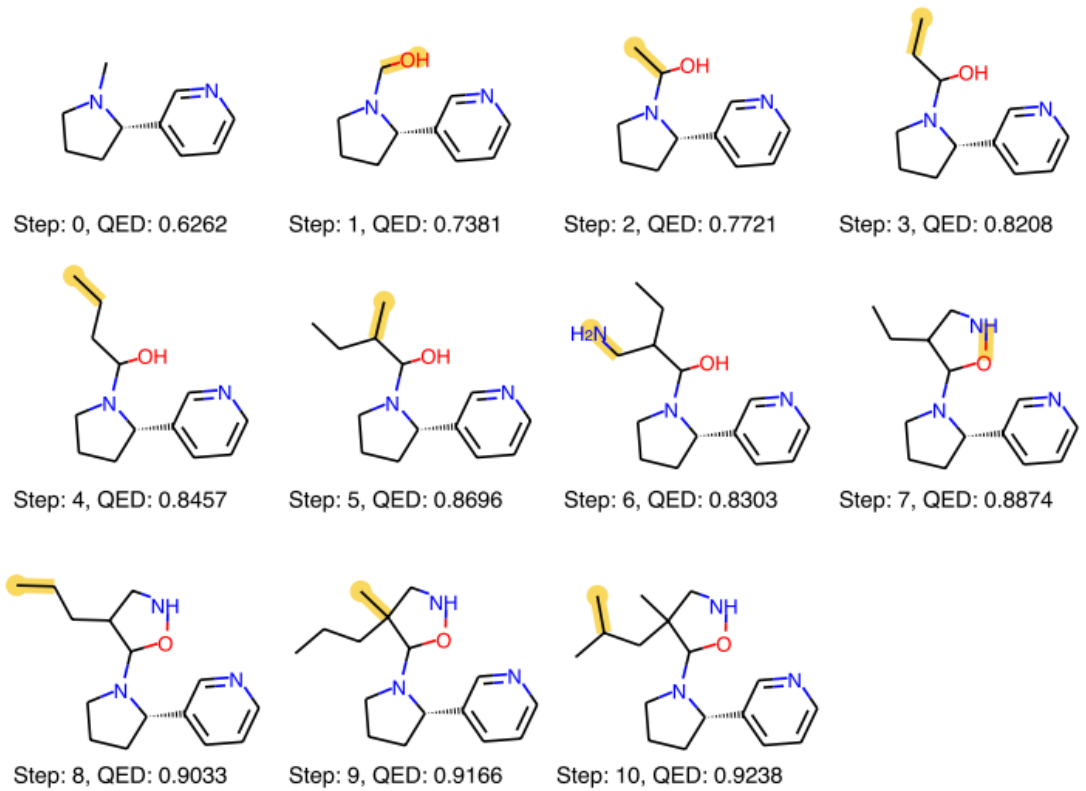


Note: no atom deletion or species changing

why do you think this is?

An Example of Molecular Optimization





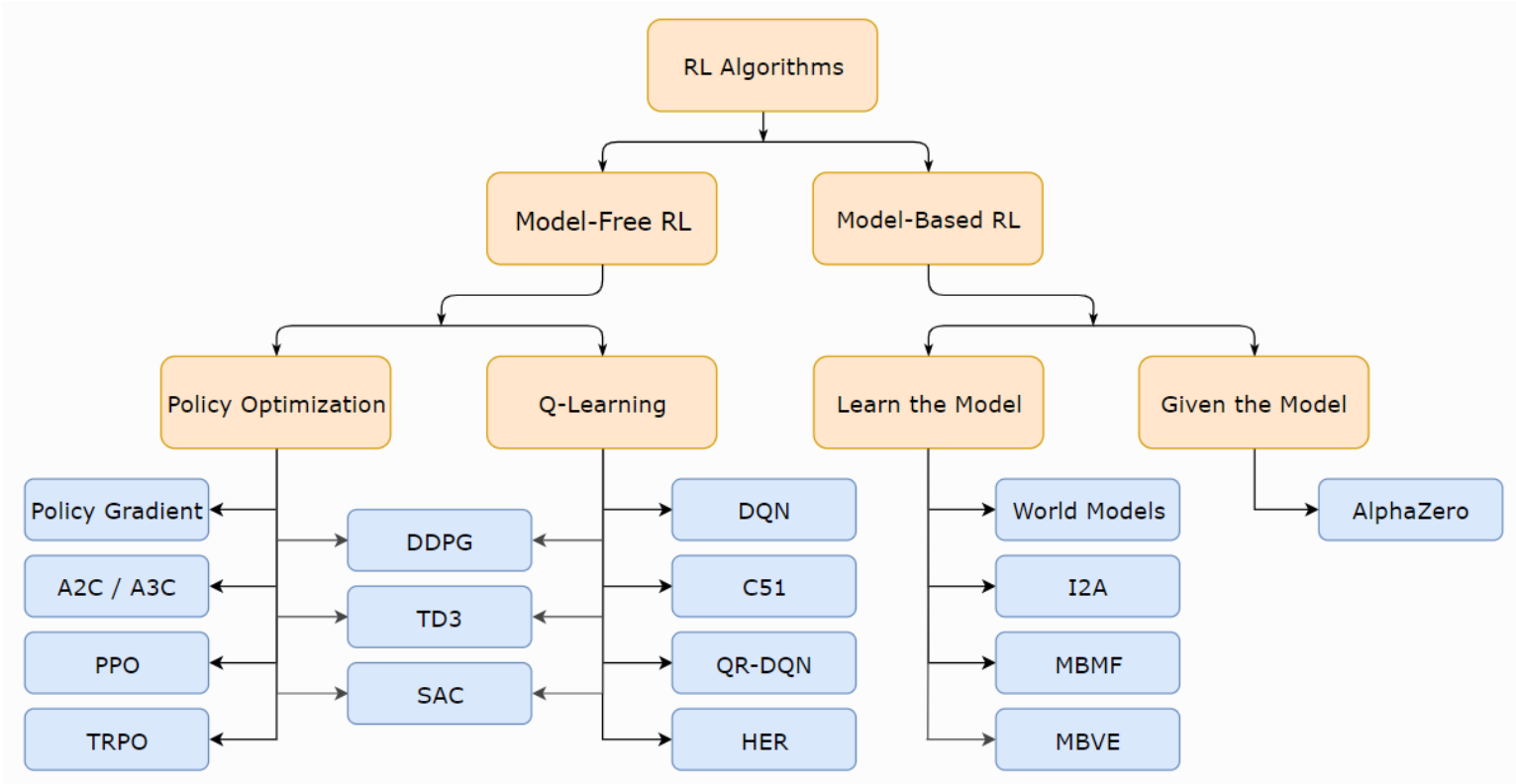
$$\vec{r}_t = QED(m)_t$$

$$Q^{\pi}(s,a) = Q^{\pi}(m,t,a) = \mathbb{E}_{\pi}[\sum_{n=t}^{T} r_n]$$

What do you notice about the 10 optimization steps on the right?
What would you need to do for multi-objective optimization?

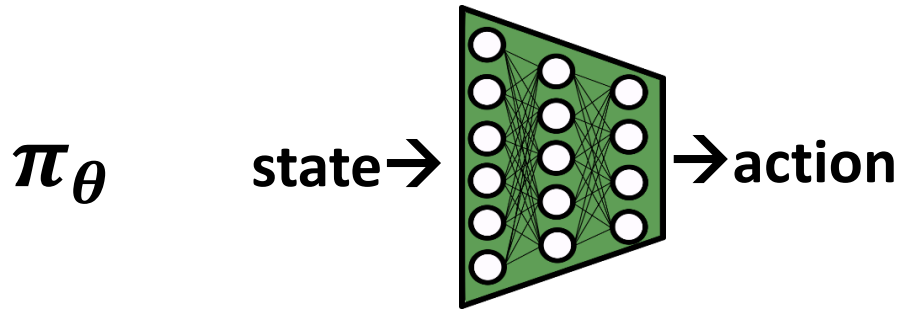
$$\mathcal{R}(s) = w \times \mathrm{SIM}(s) + (1 - w) \times \mathrm{QED}(s)$$

Questions?


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

How do we improve our policy?

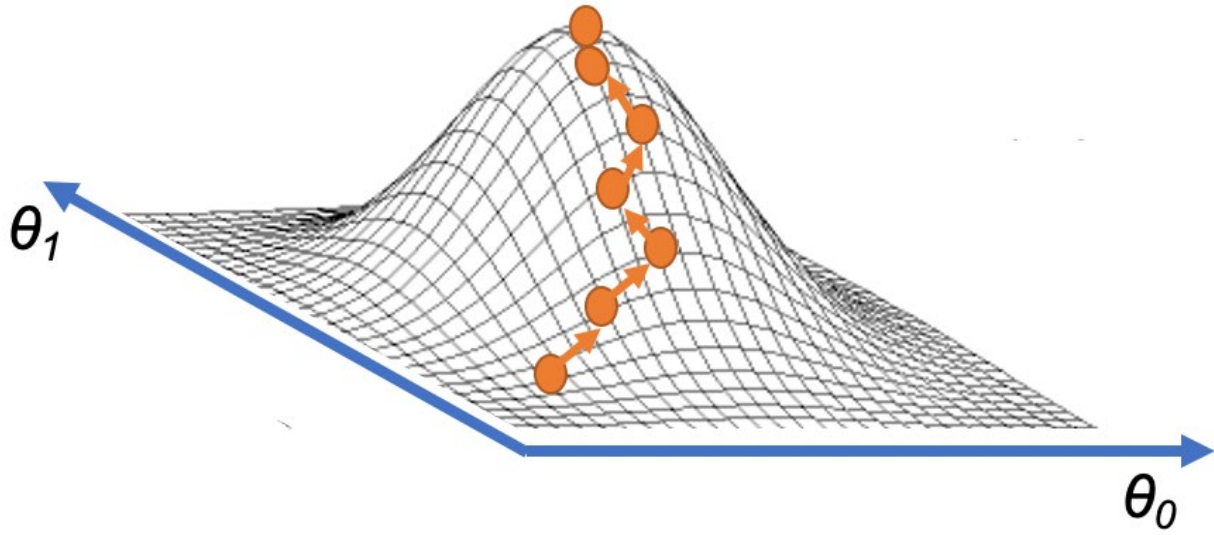
- Consider an example where our policy is a neural network with parameters Θ.



- Our agent's goal is to select actions that maximize the cumulative sum of rewards (i.e., the total return).

- It would be convenient if we could iterate to optimal policy parameters (Θ) by using gradient ascent.

Take a parameter step in the direction of $\nabla \frac{dG_r}{d\theta}$ with a step size of α.



What is the problem with this approach?

Policy-Based Solutions to the RL Problem

Recall: $\pi^*(s) = \arg\max_a Q^*(s, a)$

Policy-based solution methods seek to directly learn a state→action mapping.

For deep-learning approaches, the value function can be thought of as being contained abstractly within the neural network.

An explicit value function is not used, so training typically occurs at episode end. ***Why?***

The most common form of policy-based RL is the REINFORCE algorithm.

$$\theta_{new} = \theta_{old} + \alpha \nabla_\theta J(\theta)$$

steepest ascent

The REINFORCE Algorithm

steepest ascent

New neural network parameters are the old parameters + a change toward J(Θ)

$$\theta_{new} = \theta_{old} + \alpha \nabla_\theta J(\theta)$$

J(Θ) is the objective function: expectation of reward under the current policy

$$J(\theta) = \mathbb{E}[G_{t,\pi}]$$

To perform stochastic gradient ascent, we need ∇_θJ(Θ)

The policy gradient is defined using the Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}[G_\pi(s,a)] = \sum_{t=1}^{T} [\nabla_\theta \log \pi_\theta(a_t|s_t) G_t]$$

Informally: The Policy Gradient Theorem says we are going to scale the gradient of how likely our action is by how much return we will receive.
REINFORCE is sometimes called Monte Carlo Policy Gradient

For a nearly complete derivation see: https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

Example of Popova *et al.*

SCIENCE ADVANCES | RESEARCH ARTICLE

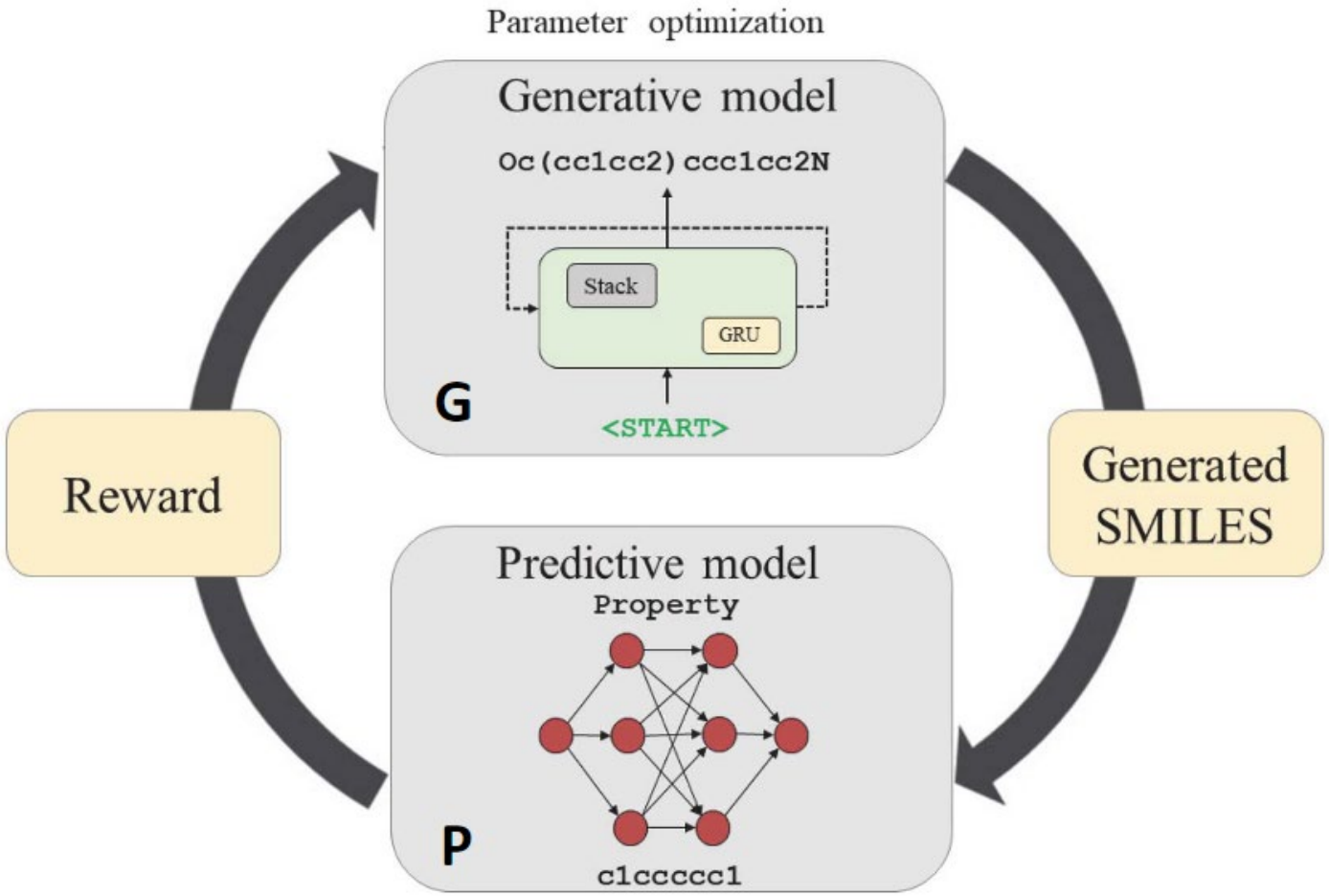
COMPUTATIONAL BIOLOGY

Deep reinforcement learning for de novo drug design

Mariya Popova[1,2,3], Olexandr Isayev[1]*, Alexander Tropsha[1]*

We have devised and implemented a novel computational strategy for de novo design of molecules with desired properties termed ReLeaSE (Reinforcement Learning for Structural Evolution). On the basis of deep and reinforcement learning (RL) approaches, ReLeaSE integrates two deep neural networks—generative and predictive—that are trained separately but are used jointly to generate novel targeted chemical libraries. ReLeaSE uses simple representation of molecules by their simplified molecular-input line-entry system (SMILES) strings only. Generative models are trained with a stack-augmented memory network to produce chemically feasible SMILES strings, and predictive models are derived to forecast the desired properties of the de novo–generated compounds. In the first phase of the method, generative and predictive models are trained separately with a supervised learning algorithm. In the second phase, both models are trained jointly with the RL approach to bias the generation of new chemical structures toward those with the desired physical and/or biological properties. In the proof-of-concept study, we have used the ReLeaSE method to design chemical libraries with a bias toward structural complexity or toward compounds with maximal, minimal, or specific range of physical properties, such as melting point or hydrophobicity, or toward compounds with inhibitory activity against Janus protein kinase 2. The approach proposed herein can find a general use for generating targeted chemical libraries of novel compounds optimized for either a single desired property or multiple properties.

High-Level Overview of Popova *et al.*



Goal is to generate smiles strings of molecules with a target property

↓

Rewards are given as a function of the target property

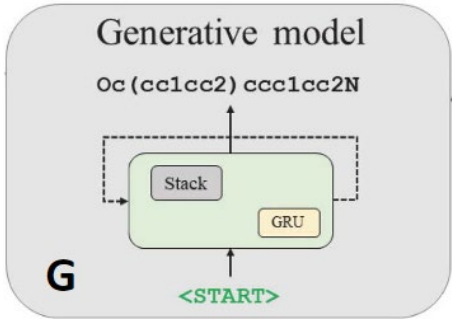
$$r(s_T) = f(P(s_T))$$

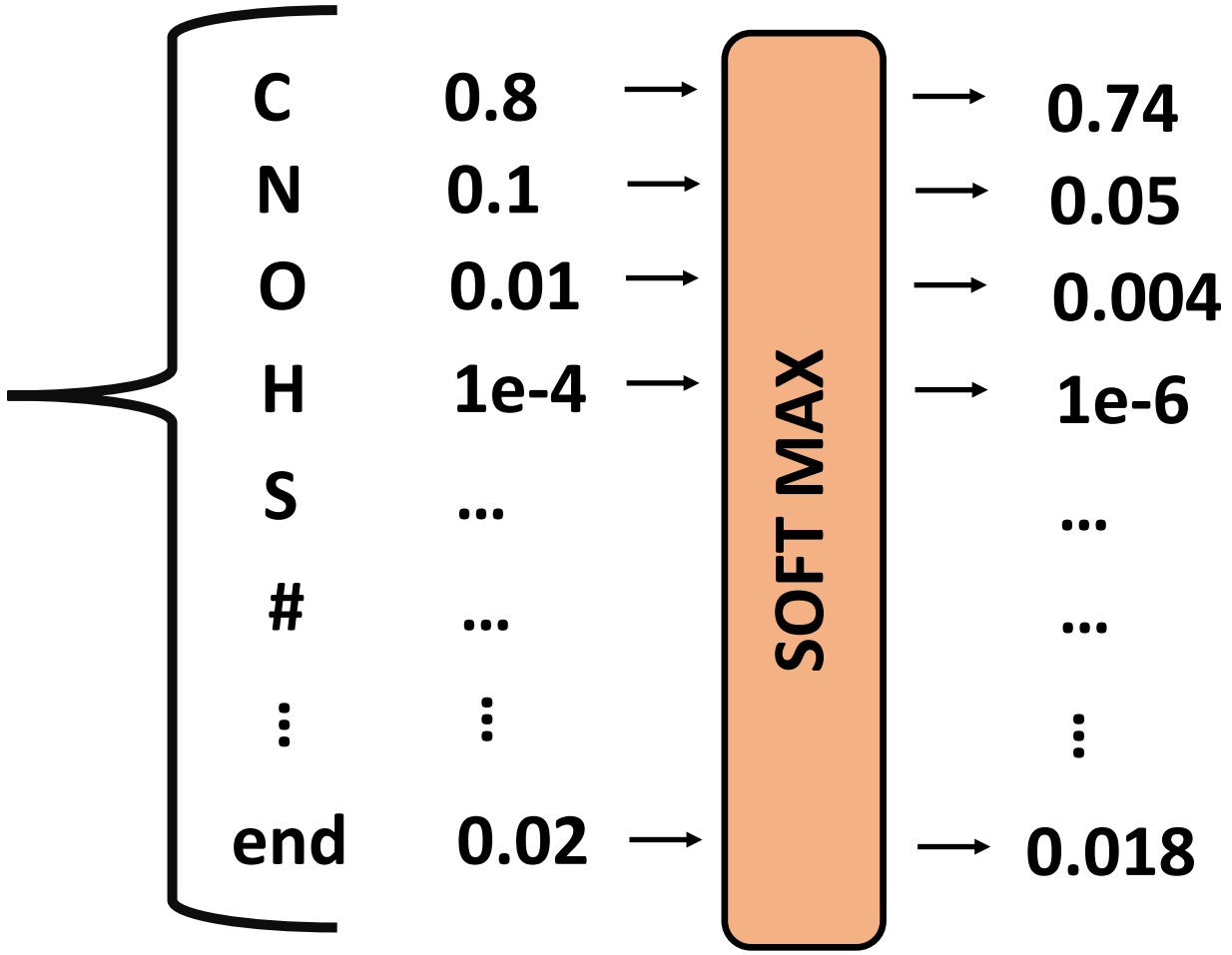
↓

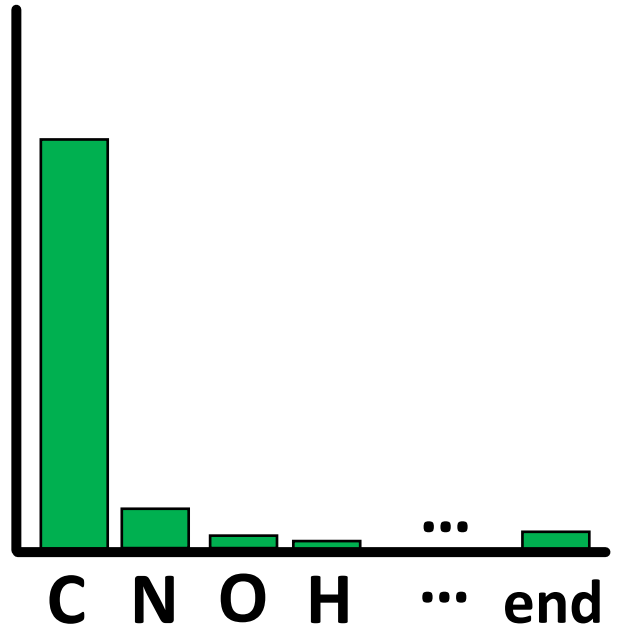
Goal is to generate smiles strings the maximize the total reward

What is the Policy here?

Let's think about how the Generator Works







$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

NEURAL NETWORK OUTPUTS

PROBABILITIES

Low-Level Formalism of Popova *et al.*

Design decision: assume there are no intermediate rewards. This is logical because there is no clear interpretable value in a partially finished smiles string. CC(=O)NC1=CC=C(C=C1)O

$v_\pi(S_0) = \mathbb{E}(R_1 + \gamma R_2 + \gamma^2 R_3 + \ldots | S_0) = \mathbb{E}(R_T | S_0)$

Our policy (π) is a recurrent neural network with parameters θ

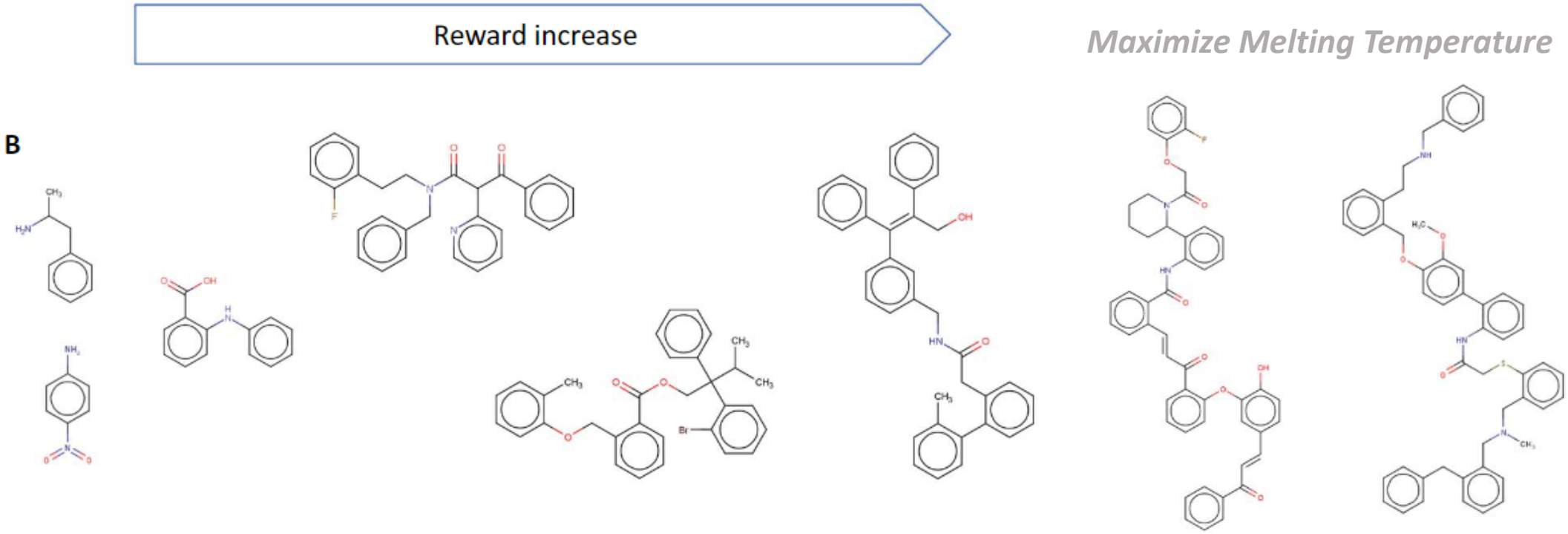
$\pi(s) = RNN_\theta(s)$

We can perform a policy update with the REINFORCE algorithm as

$\theta_{new} = \theta_{old} + \alpha \partial_\theta J(\theta)$

$$\partial_\Theta J(\Theta) = \mathbb{E}_{a_1 \sim p_\Theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\Theta(a_2|s_1)} \cdots \mathbb{E}_{a_T \sim p_\Theta(a_T|s_{T-1})} \left[\sum_{t=1}^{T} \partial_\Theta \log p_\Theta(a_t|s_{t-1}) \right] r(s_T)$$

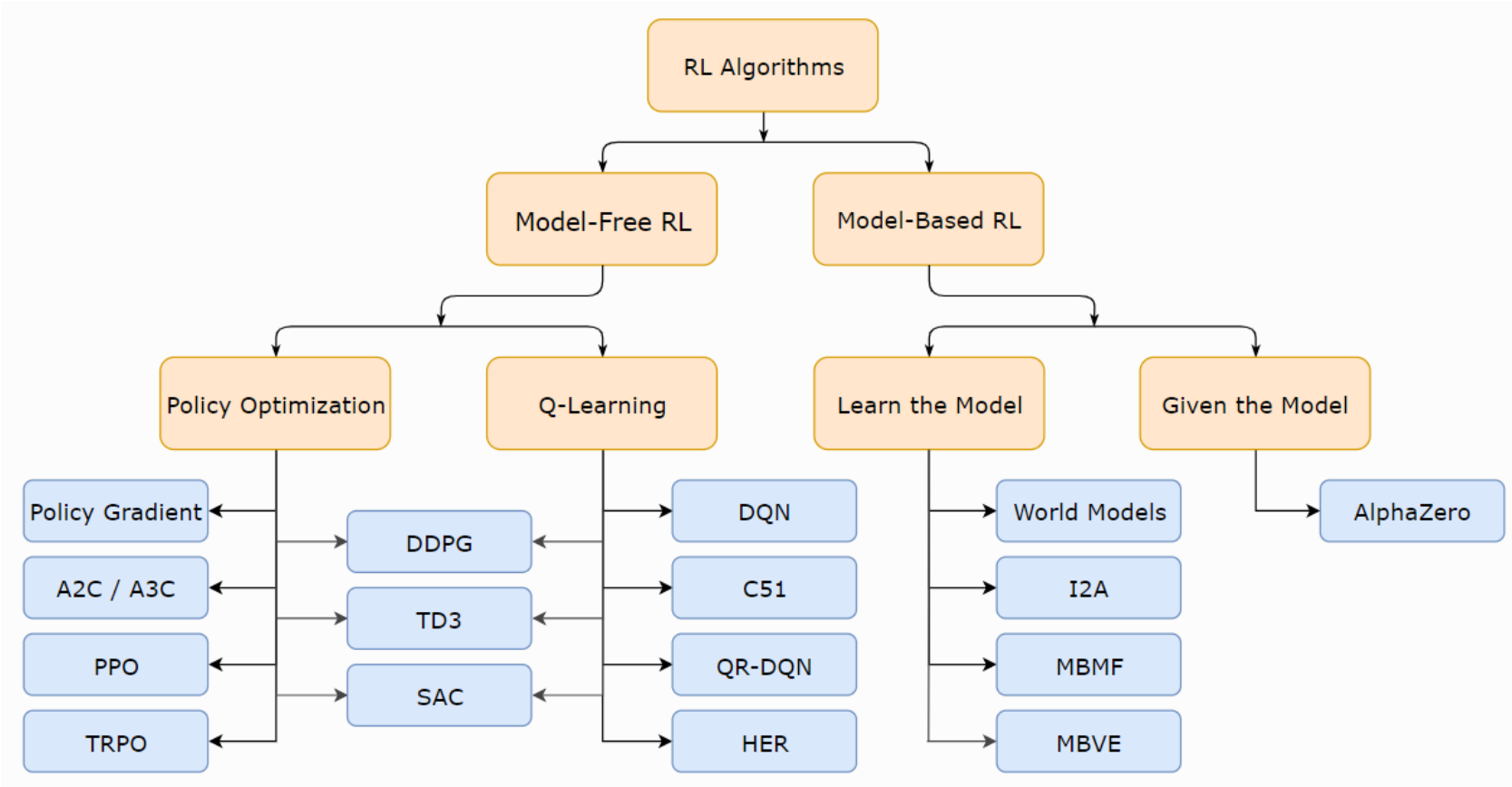
Example Results of Popova *et al.*



The neural network is not given any molecular design rules, yet it discovers that it should be increasing the number of benzene rings. This makes sense from the standpoint of chemical physics.

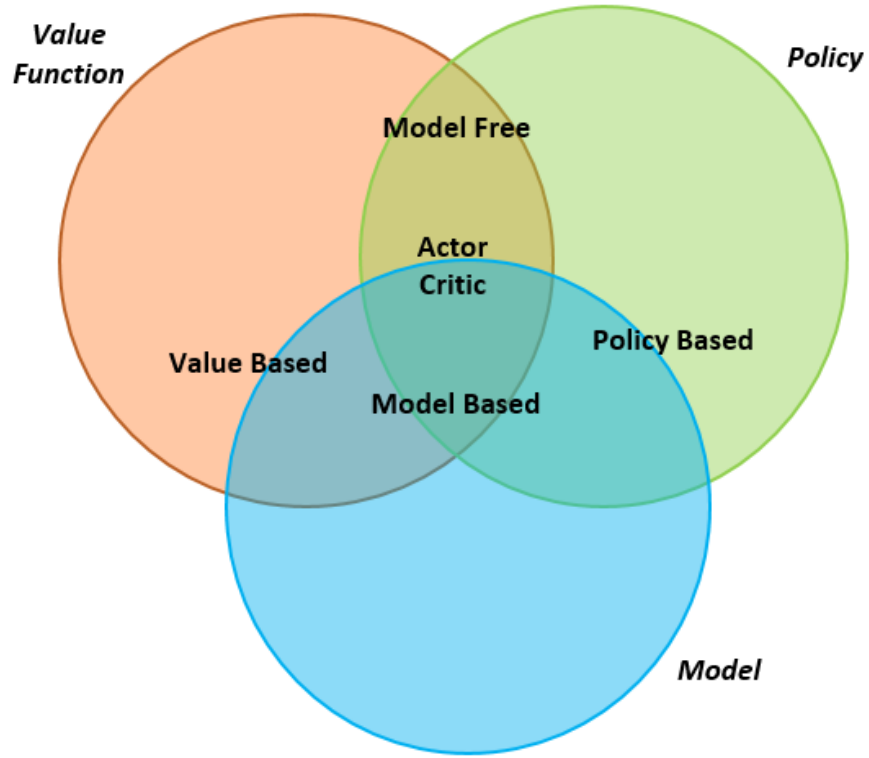
WOW! GO RL!

Questions?


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

Actor-Critic Methods

- Actor-critic methods solve the RL problem by combining value-based solution methods with policy-based solution methods.
- Value-based: learn the optimal value function and then take actions with the highest state-value or action-values.
- Policy-based: learn the policy that maps states-actions. The state-value or action-value function is kept as an internal representation.

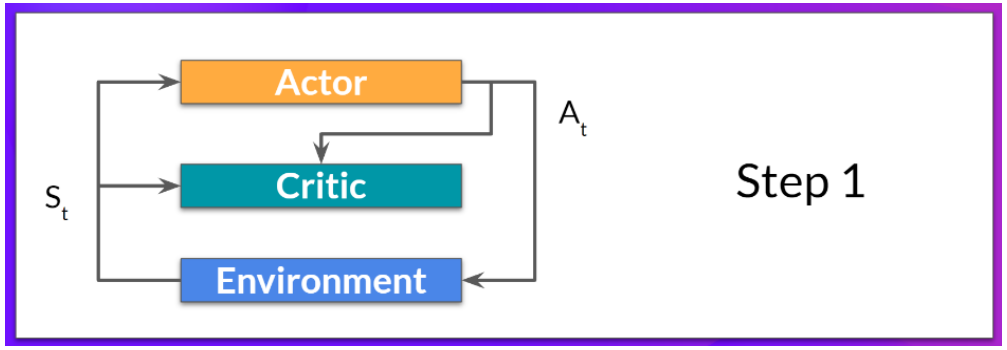


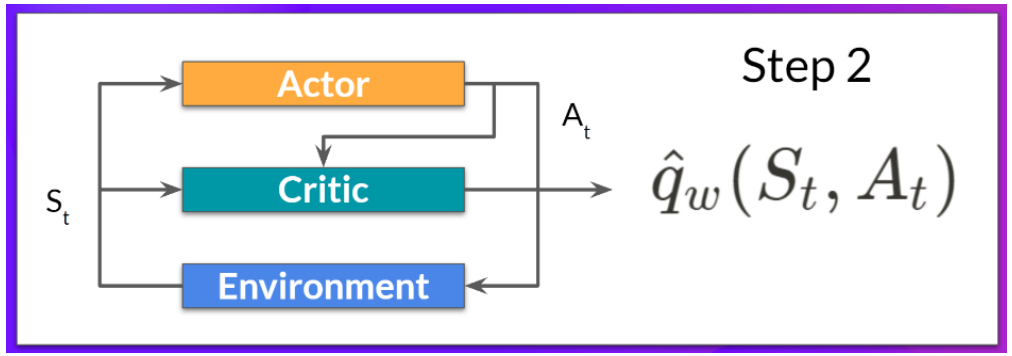
Combining the methods improves upon the limitations of each type of method

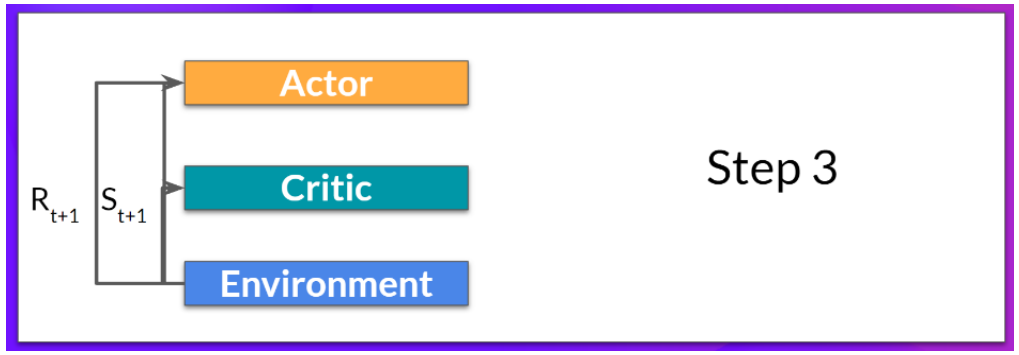
Value-based methods→ can suffer from poor convergence and design decision limitations

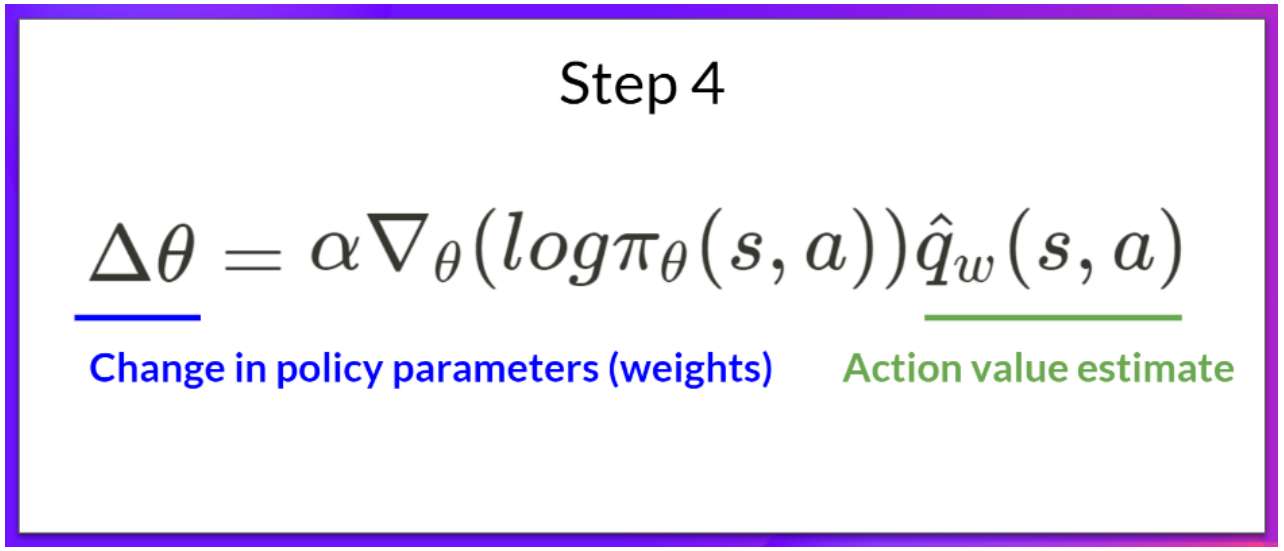
Policy-based methods→ variance in complex environments leads to slow training

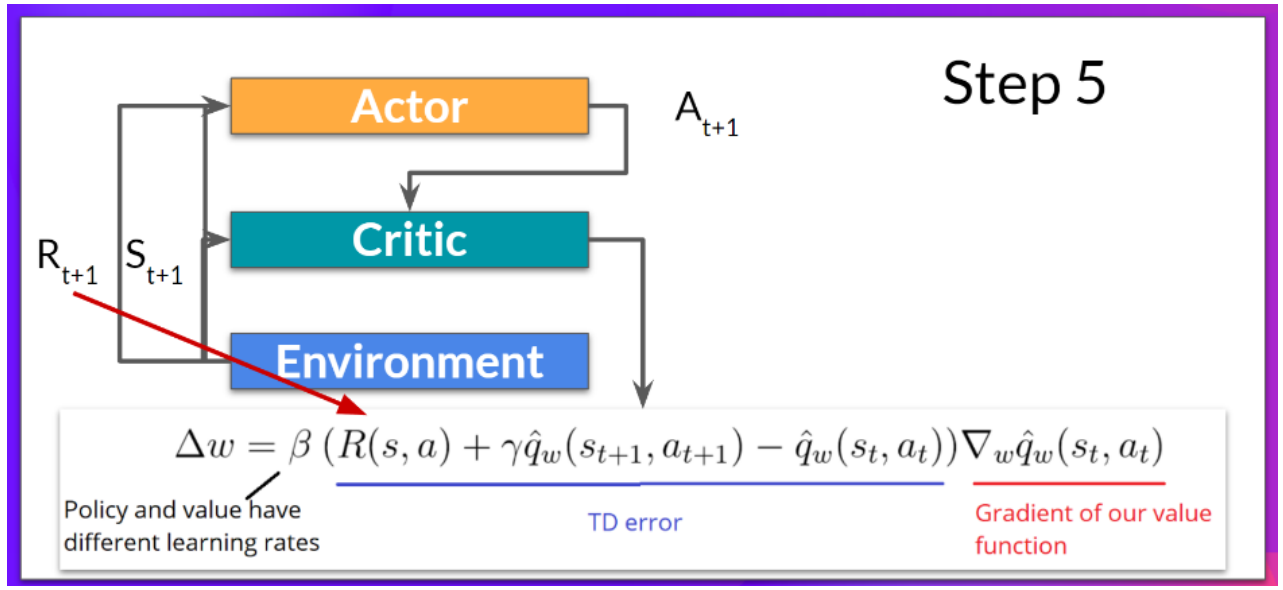
The Actor-Critic Process











https://huggingface.co/blog/deep-rl-a2c

Advantage Actor-Critic (A2C)

Step 4

$$\Delta\theta = \alpha\nabla_\theta(log\pi_\theta(s,a))\hat{q}_w(s,a)$$

Change in policy parameters (weights) — Action value estimate

We can stabilize training even further by replacing our Action-value estimate with the so-called advantage function.

$$A(s,a) = Q(s,a) - V(s)$$

q value for action a in state s

average value of that state

Informally, we can say the advantage function is a measure how much better my action is compared to the current value estimate.

What is the problem here? Hint: # of Networks.

$$A(s,a) = Q(s,a) - V(s)$$

$$r + \gamma V(s')$$

$$A(s,a) = r + \gamma V(s') - V(s)$$

TD Error

Apply the Bellman-equation for the state-value function to arrive at the TD Error as an advantage estimate.

https://huggingface.co/blog/deep-rl-a2c

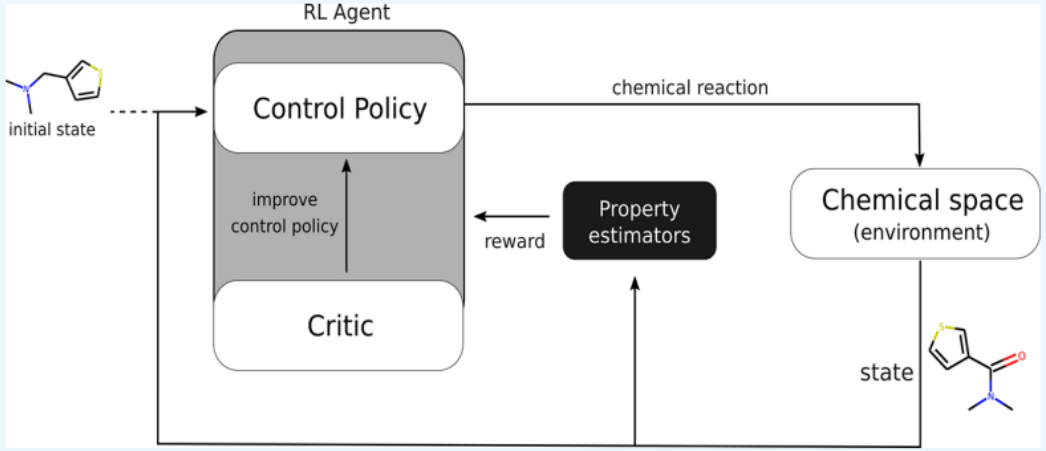
Example of *Horwood* and *Noutahi*

Molecular Design in Synthetically Accessible Chemical Space via Deep Reinforcement Learning

Julien Horwood* and Emmanuel Noutahi*

Cite This: *ACS Omega* 2020, 5, 32984–32994 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information



ABSTRACT: The fundamental goal of generative drug design is to propose optimized molecules that meet predefined activity, selectivity, and pharmacokinetic criteria. Despite recent progress, we argue that existing generative methods are limited in their ability to favorably shift the distributions of molecular properties during optimization. We instead propose a novel Reinforcement Learning framework for molecular design in which an agent learns to directly optimize through a space of synthetically accessible drug-like molecules. This becomes possible by defining transitions in our Markov decision process as chemical reactions and allows us to leverage synthetic routes as an inductive bias. We validate our method by demonstrating that it outperforms existing state-of-the-art approaches in the optimization of pharmacologically relevant objectives, while results on multi-objective optimization tasks suggest increased scalability to realistic pharmaceutical design problems.

Molecular optimization

given an initial molecule, what changes need to be made to achieve the target properties

Starting Structures

↓

molecular edits

↓

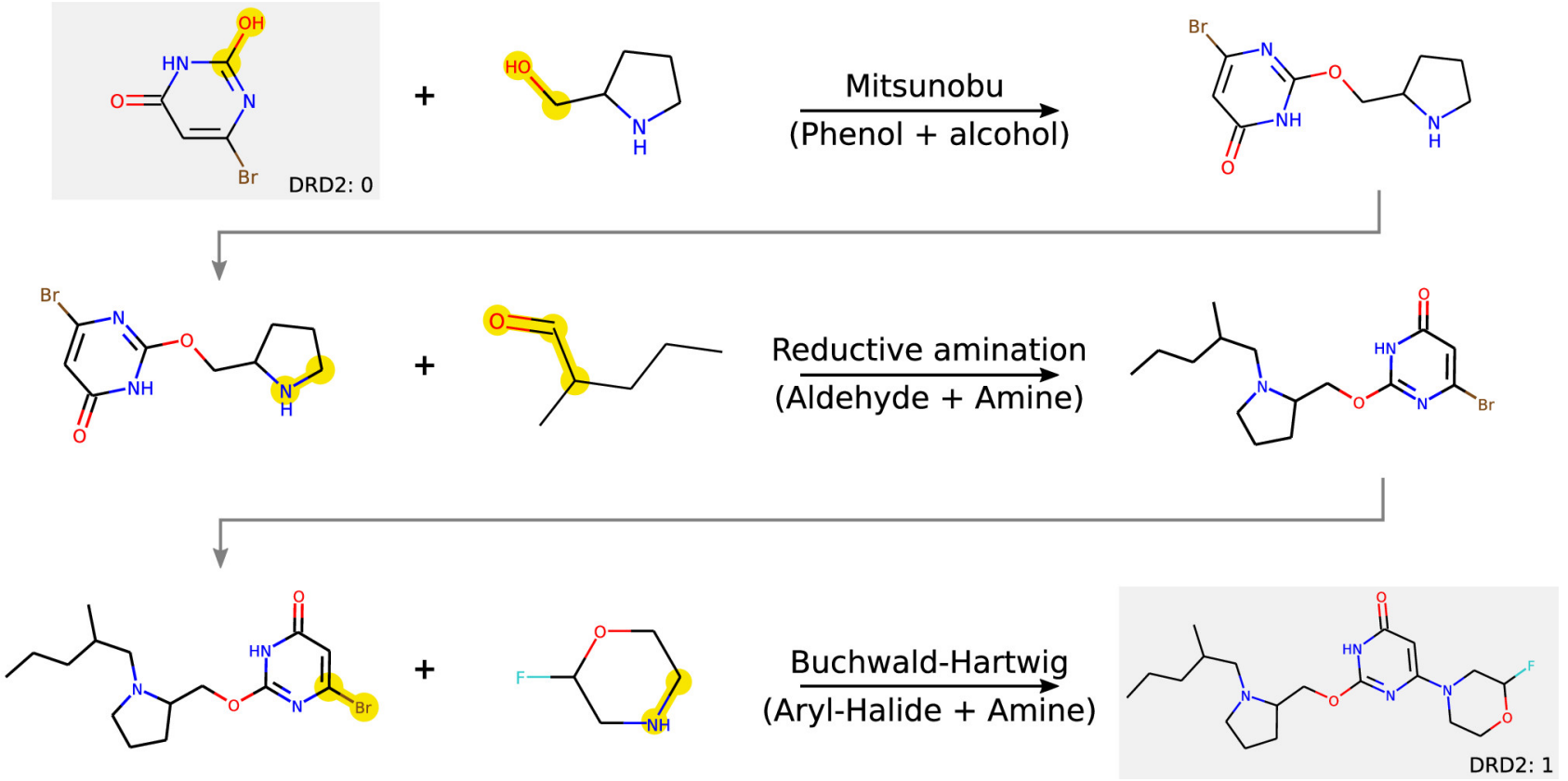
Structures Optimized for Target Property

Synthetically Feasible Optimization with RL

Instead of considering graph-based transformations like Zhou *et al.*, Horwood and Noutahi perform "hypothetical reactions" with known chemical reaction mechanisms.

Why might this strategy be better than the DQN work of Zhou *et al.*?

Is this problem an MDP?



Horwood and *Noutahi* RL Problem Design

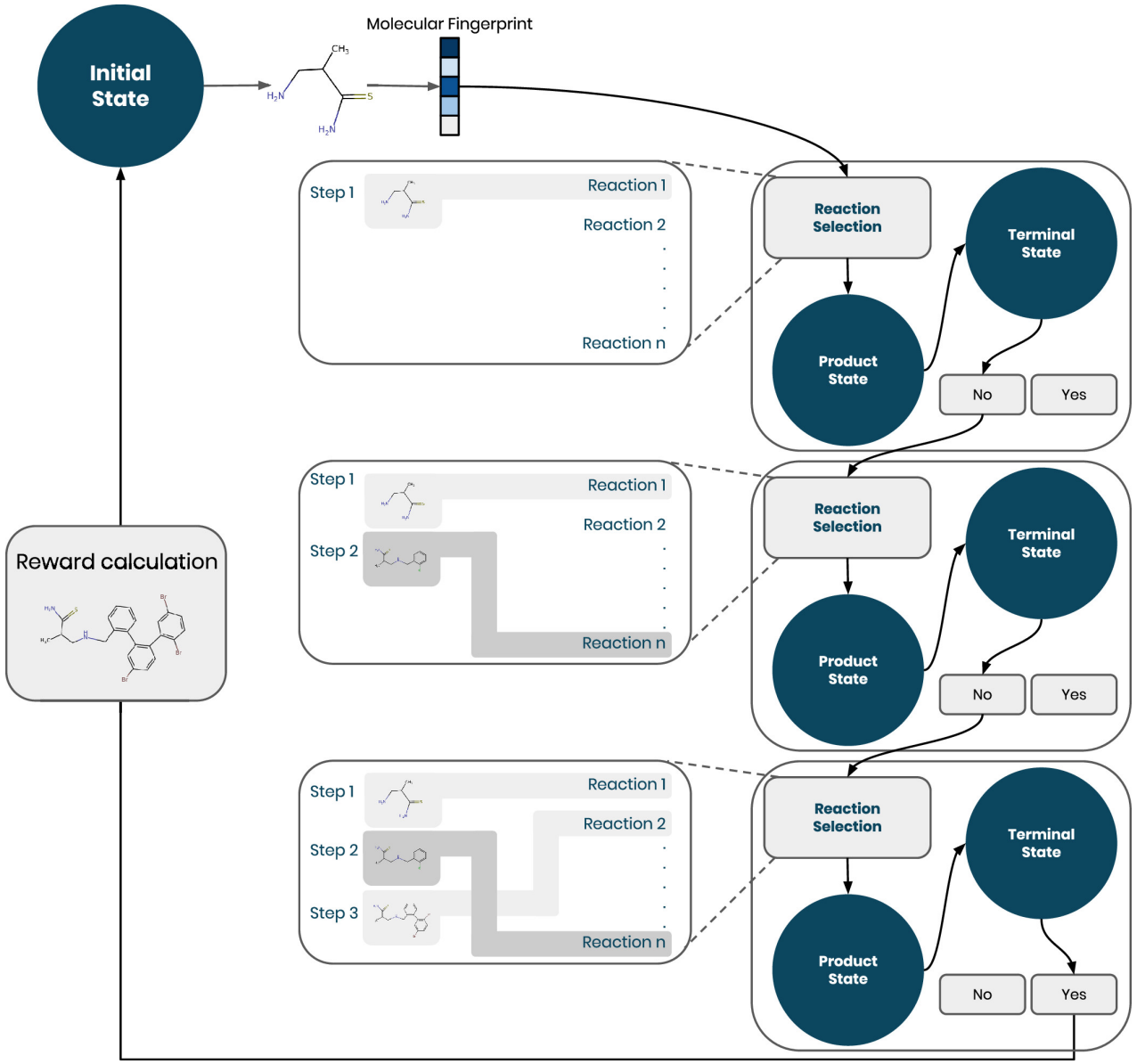
State: molecule species that is reactant 1

Action Space: a collection of predefined templates + reactant 2's

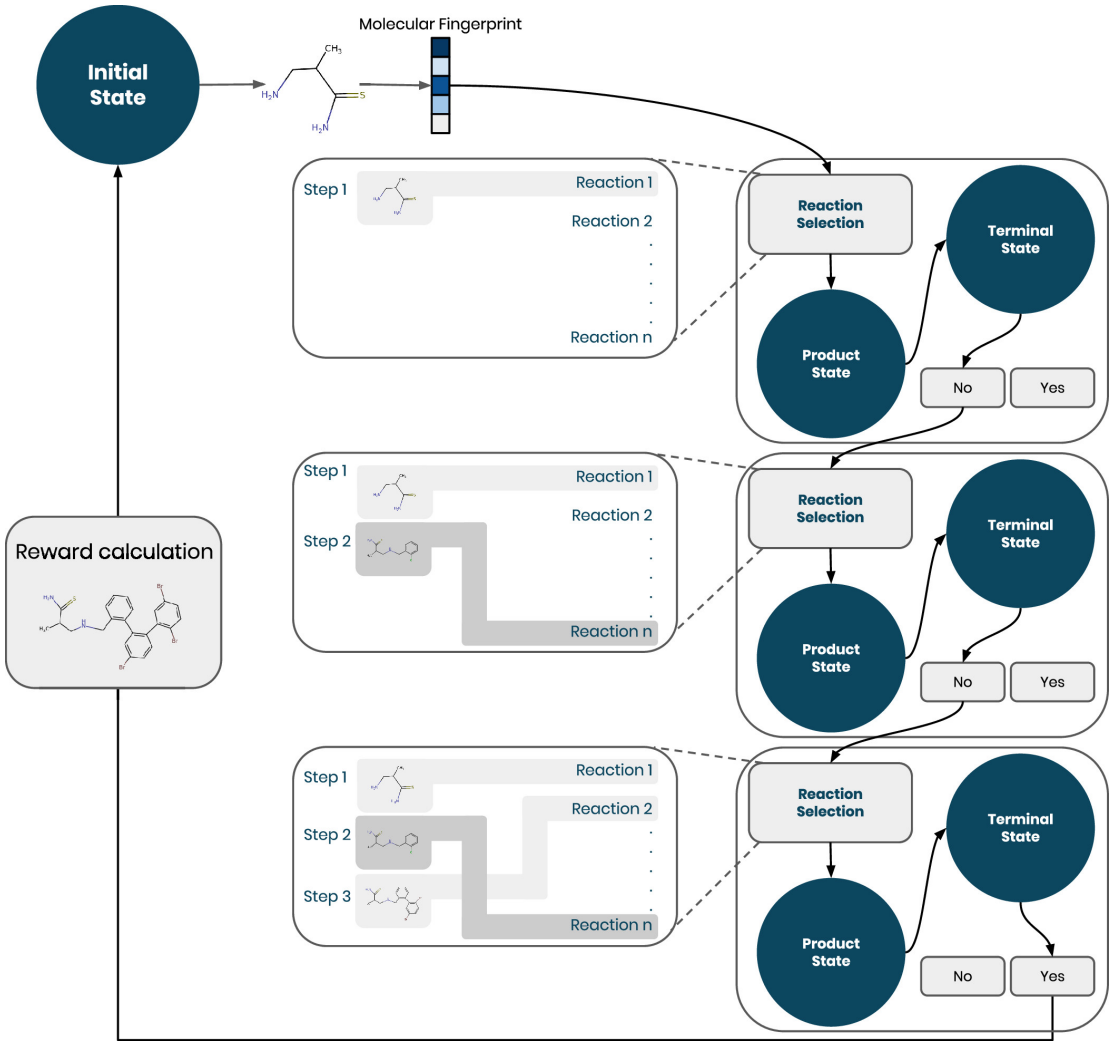
Reward: Calculated at end-of-episode and is proportional to the target property.

Actor: Multilayer perceptron that takes molecular fingerprint as an input and selects action as an output.

Critic: Multilayer perceptron that approximates the state-value function based on molecular fingerprint.



Horwood and *Noutahi* RL Problem Design



policy-objective

$$J(\theta,\theta') = log(\pi_\theta(a_t|s_t)) \sum_{i=0}^{k} (r_{t+i} + \gamma^i v_{\theta'}(s_{t+i}) - v_{\theta'}(s_t))$$

policy-objective rewritten

$$J(\theta,\theta') = log\big(\pi_\theta(a_t|s_t)\big)A(s_t,a_t,\theta',k)$$

The goal of advantage actor-critic is to maximize the policy objective

$$\max J(\theta,\theta')$$

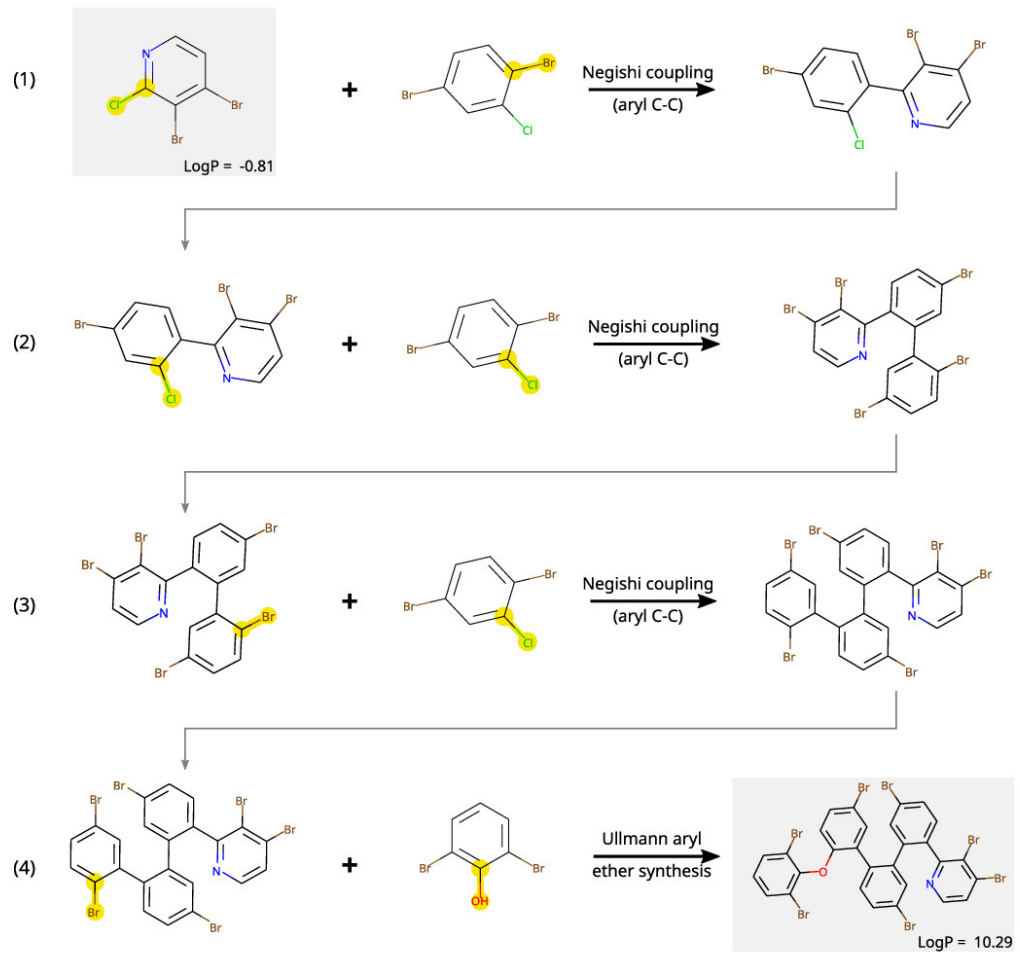
$$iterate\ \theta_{new} = \theta_{old} + \alpha\nabla_\theta J(\theta,\theta')$$

policy gradient theorem

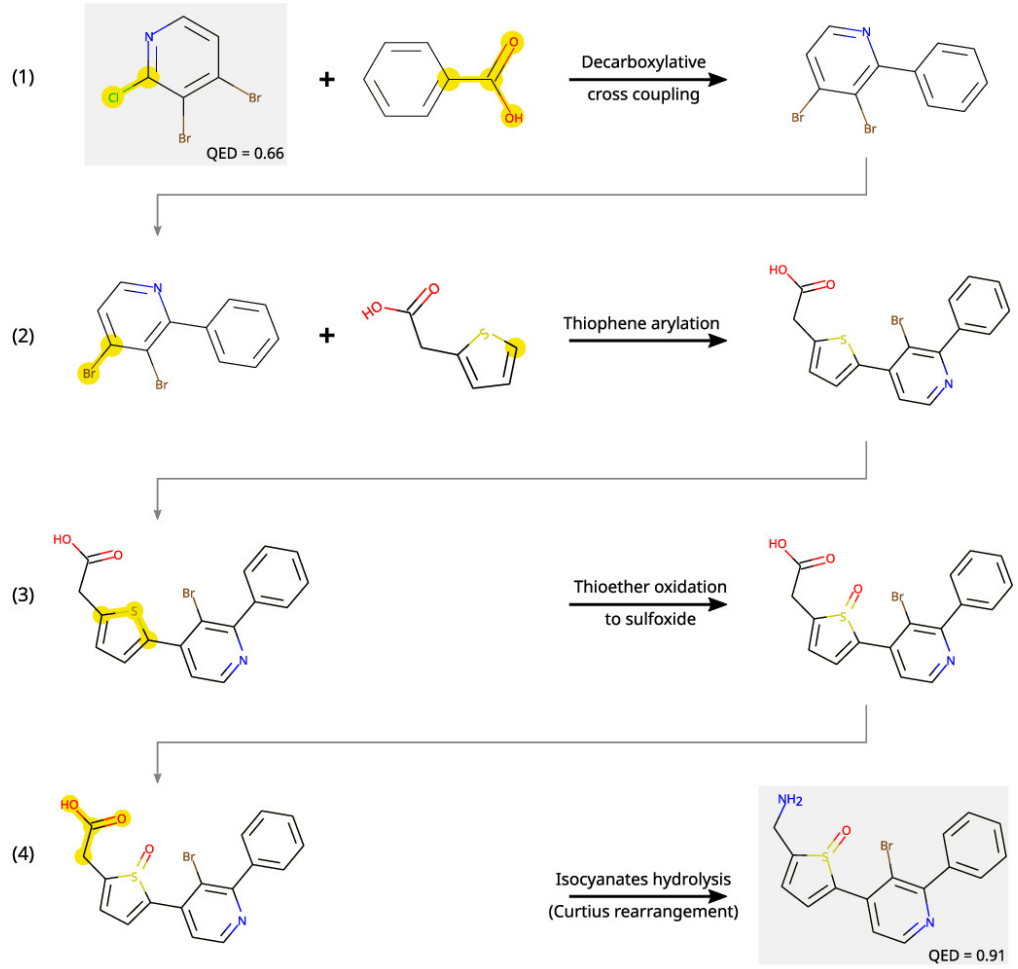
$$\nabla_\theta J(\theta,\theta') = \nabla_\theta log\big(\pi_\theta(a_t|s_t)\big)A(s_t,a_t,\theta',k)$$

Horwood and *Noutahi* RL Example Results

Same starting block → different targets → different sequence of synthetic steps.
No chemist needed, just an RL agent that learns chemical synthesis by trial-and-error experience.

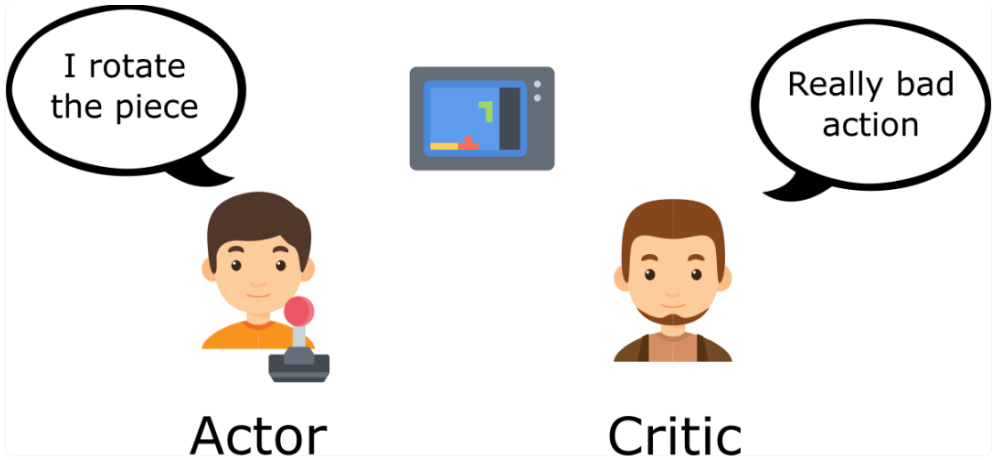


(a) cLogP Trajectory



(b) QED Trajectory

Questions?



FUNCTION: Advantage Actor Critic
initialize s, Θ
initialize actor optimizer and critic optimizer
for t = 0,1,2,… **do**
optimizers.zero_grad()
sample actions ~ $\pi_\theta(s_t)$
collect rewards r_{t+1} and new states s_{t+1}
calculate advantage
calculated policy loss
calculate MSE value loss
policy_loss.backward()
value_loss.backward()
optimizers.step()
end for
end FUNCTION

Can we improve A2C?

What are some potential problems here?

FUNCTION: Advantage Actor Critic
initialize s, Θ
initialize actor optimizer and critic optimizer
for t = 0,1,2,… **do**
optimizers.zero_grad()
sample actions ~ $\pi_\theta(s_t)$
collect rewards r_{t+1} and new states s_{t+1}
calculate advantage
calculated policy loss
calculate MSE value loss
policy_loss.backward()
value_loss.backward()
optimizers.step()
end for
end FUNCTION

1) Hyperparameter choices, like step size, are crucial to performance.
2) If sampled actions give significantly bad results, then the policy can diverge.
3) Advantage estimates can be noisy, especially in stochastic environments.

Importance of Step Size

In supervised learning tasks, if your step size is too large your loss function will have large **oscillations and learn very slowly**.

In policy optimization tasks, if your step size is too large you end up with a terrible policy, and a terrible policy leads to terrible sampling.

1. The most **Naïve approach is to try several fixed step sizes** and see which one works the best.
2. A better approach is to do a **line search** and test different step sizes in the direction of the gradient.
3. Another better approach is to use the maximum step size allowable subject to a constraint. Algorithms of this type are referred to as **trust region** approaches.

Trust Region Policy Optimization (TRPO)

Trust region policy optimization (TRPO) seeks to **limit the amount a policy changes during actor-critic training iteration**.

The direction of policy change is informed by using an ideal update **along the best direction in the trust region**.

The details of implementing TRPO are technical, and we will focus on the big picture for today.

Line search | Trust region

$$\boldsymbol{\theta}\ \textbf{maximize}\quad \mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right] \qquad \textbf{under the constraint}\ \ \mathbb{E}\left[KL\left[\pi_{\theta_{old}}(\cdot|s_t),\pi_\theta(\cdot|s_t)\right]\right] \leq \delta$$

What is KL divergence?

For more information on TRPO see the original publication: https://doi.org/10.48550/arXiv.1502.05477

An Informal Description of TRPO

$$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

How likely is taking a_t from s_t in the current policy compared to the old policy

$$\mathbb{E}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t\right]$$

Scale the ratio the likelihood of taking a_t by the advantage function A_t.

Trust region

If A_t is positive, the action should be more likely under π_θ compared to $\pi_{\theta_{old}}$
If A_t is negative, the action should be less likely under π_θ compared to $\pi_{\theta_{old}}$
If A_t is ~0, we don't want to change our policy

$$\mathbb{E}\left[KL\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right]$$

How similar is the probability distribution of actions under π_θ compared to $\pi_{\theta_{old}}$

$$\mathbb{E}\left[KL\left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \leq \delta$$

Make sure that π_θ remains somewhat close to $\pi_{\theta_{old}}$

For a baseline pytorch implementation of TRPO check out the following GitHub repository: https://github.com/ikostrikov/pytorch-trpo

Additional Resources

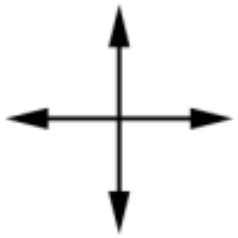
Repositories for loading and interacting with pre-built proof-of-concept environments.
https://github.com/Farama-Foundation/Gymnasium
https://github.com/openai/gym

leaderboards for gym environment!
https://github.com/openai/gym/wiki/Leaderboard

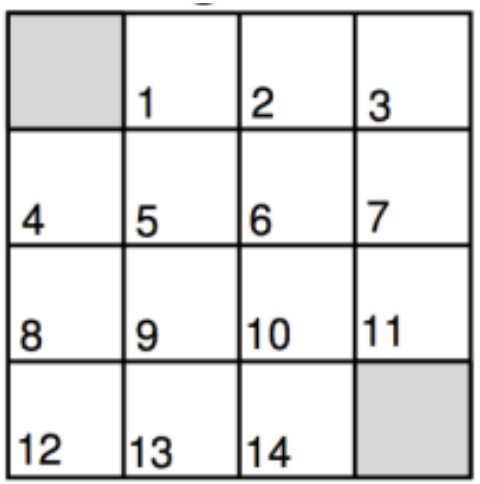
Huggingface is one of the best places for additional information and review materials.
https://huggingface.co/deep-rl-course/unit0/introduction

Pytorch has a great tutorial on DQN for learning to play super mario.
https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html

An Example RL Problem: Small Gridworld



actions



$r = -1$
on all transitions

- Our simulation will be an undiscounted MDP
- The terminal states are boxes 0 and 15 (grey)
- Nonterminal states are 1,…,14
- Actions “leaving” the grid are prohibited
- Reward is -1 until the terminal state is reached
- The Agent follows a random policy

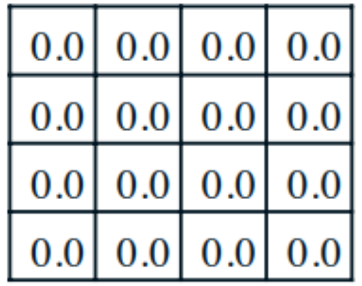
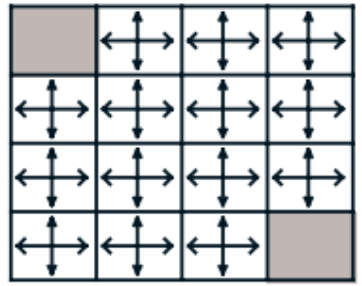
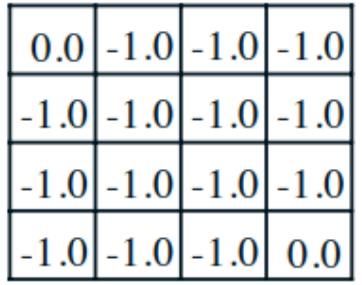
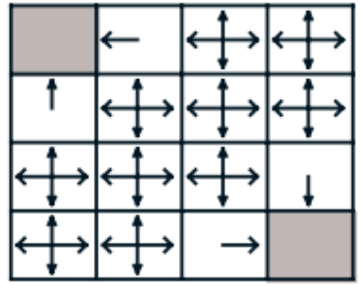
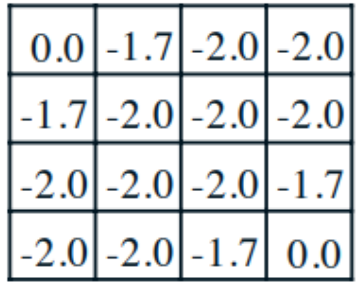
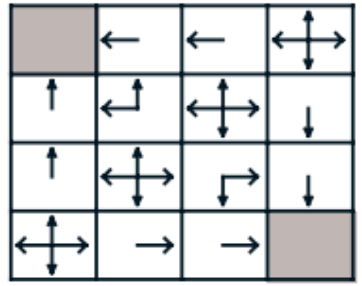
$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

An Example RL Solution: Value Iteration

Step 1: initialize the state-value function.
Step 2: at every state in the environment sample every possible action.
Step 3: update the value function based on the expected return.
Step 4: repeat steps 2-4 until ~convergence.

$$v_{k,s} = r_k + \sum_{a \in A} p(a) \cdot v_{s+1}$$

Note: for most RL problems sampling all actions at all states is unlikely to be a tractable solution.

	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$		
$k = 1$		
$k = 2$		

random policy

An Example RL Solution: Small Gridworld

Notice that the optimal policy does not mean that every state only has one optimal action.

Notice that this simple example arrives at an optimal policy (through greedy actions) before the value function converges to the optimal value function.

$k = 3$

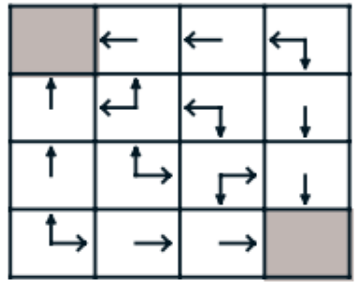
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

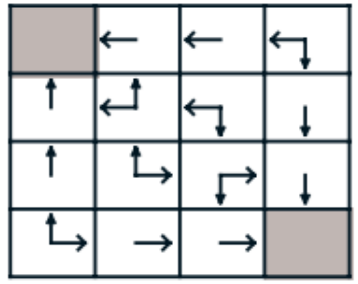
$k = 10$

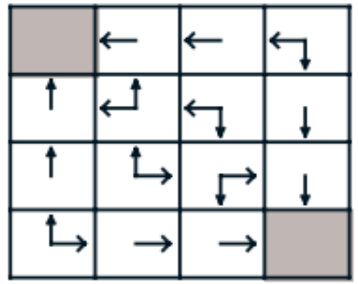
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0







optimal policy

Exploration vs. Exploitation

- Exploration vs Exploitation is the RL flavor of Interpolation vs. Extrapolation.
- RL is a trial-and-error-type of method to solving MDPs.
- At every time step in an episode our agent is faced with a choice:
  - **Exploit** knowledge by choosing the best possible action / best possible state.
  - **Explore** new actions that are closely (or distantly) related to what the agent believes is best.