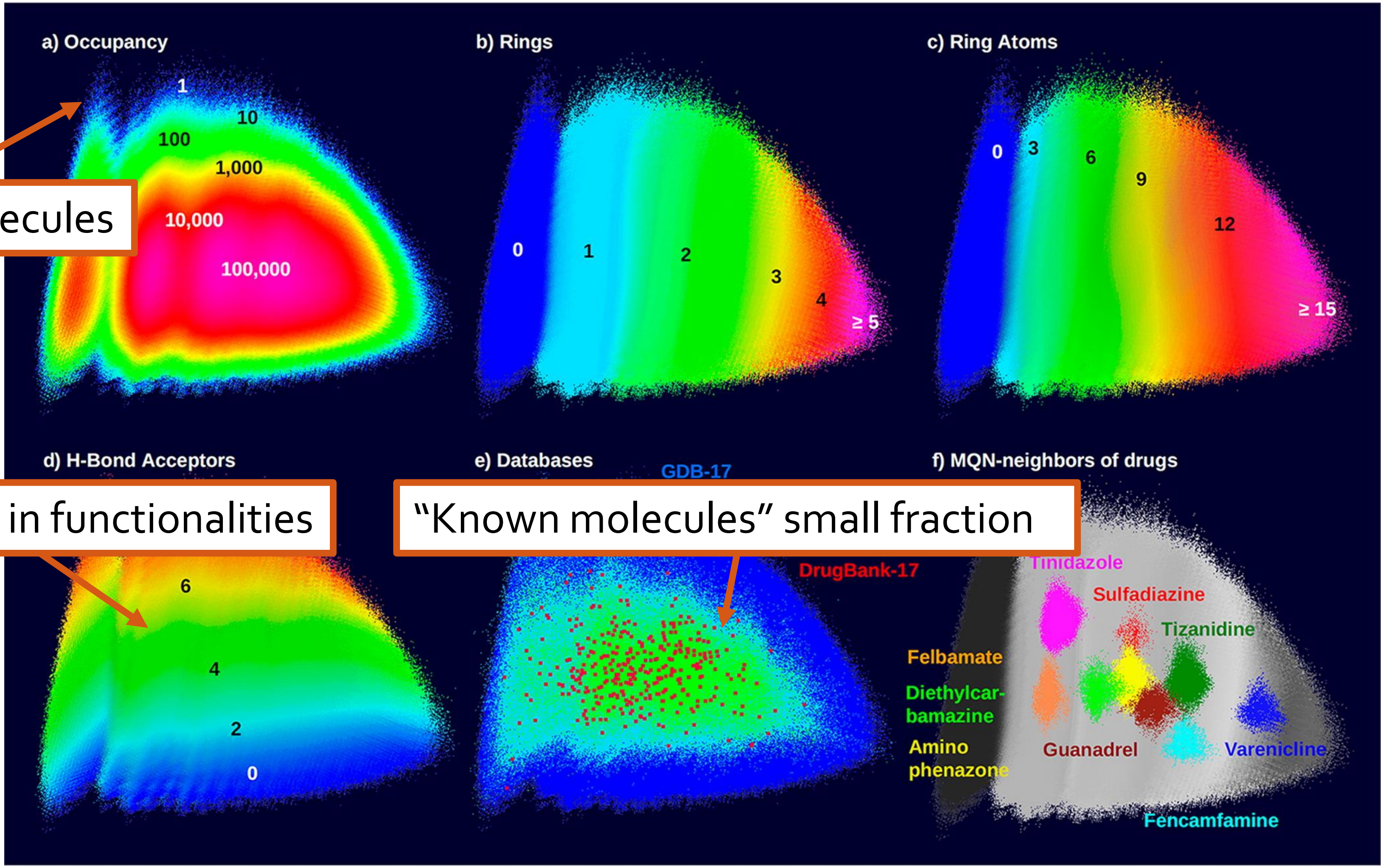
REINFORCEMENT LEARNING FOR MOLECULE GENERATION

Logan Ward
Asst. Computational Scientist
Argonne National Laboratory

14 February 2022

Chemical space is enormous

Searching through even small fraction of "all molecules" is impractical



1.6 x 10^{11} molecules

Wide variety in functionalities

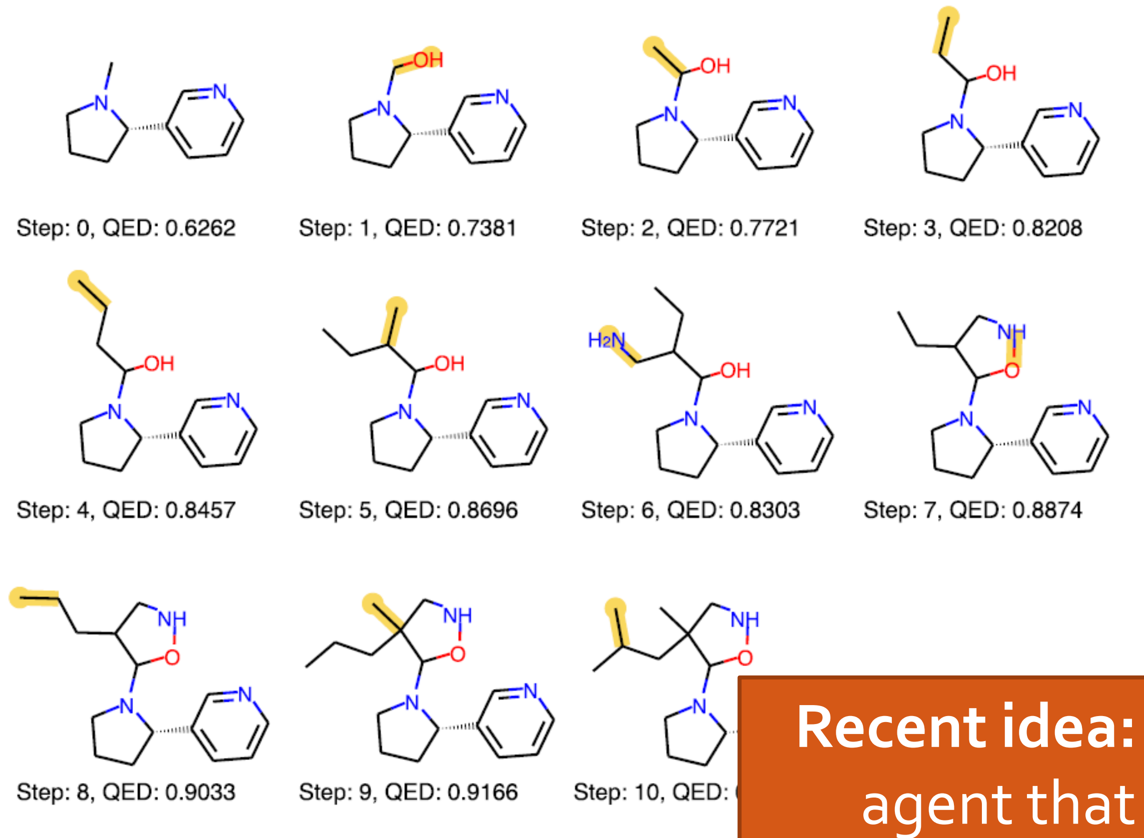
"Known molecules" small fraction

Fig: Ruddigkeit et al. JCIM. (2013)

We need a way to sample through space efficiently

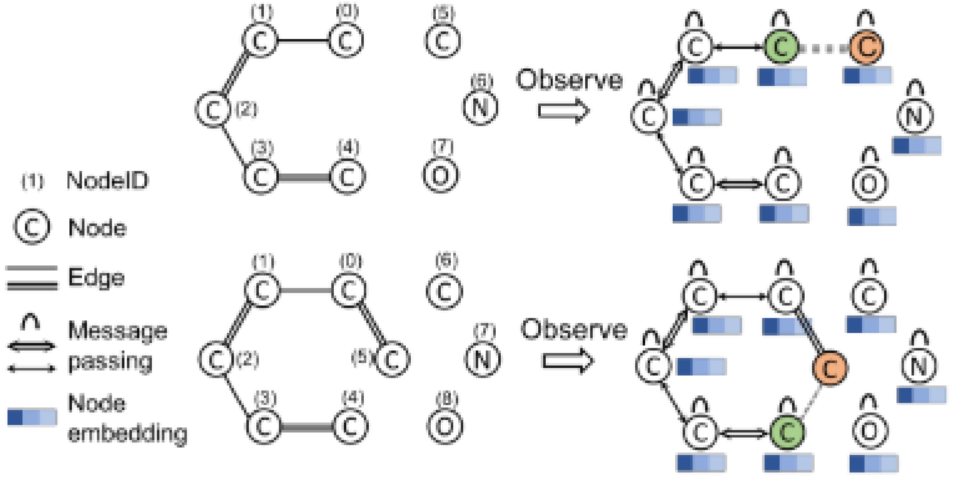
Using RL to sample chemical space

Example #1: MolDQN



Ref: [Zhou et al. Sci Rep. (2019)]

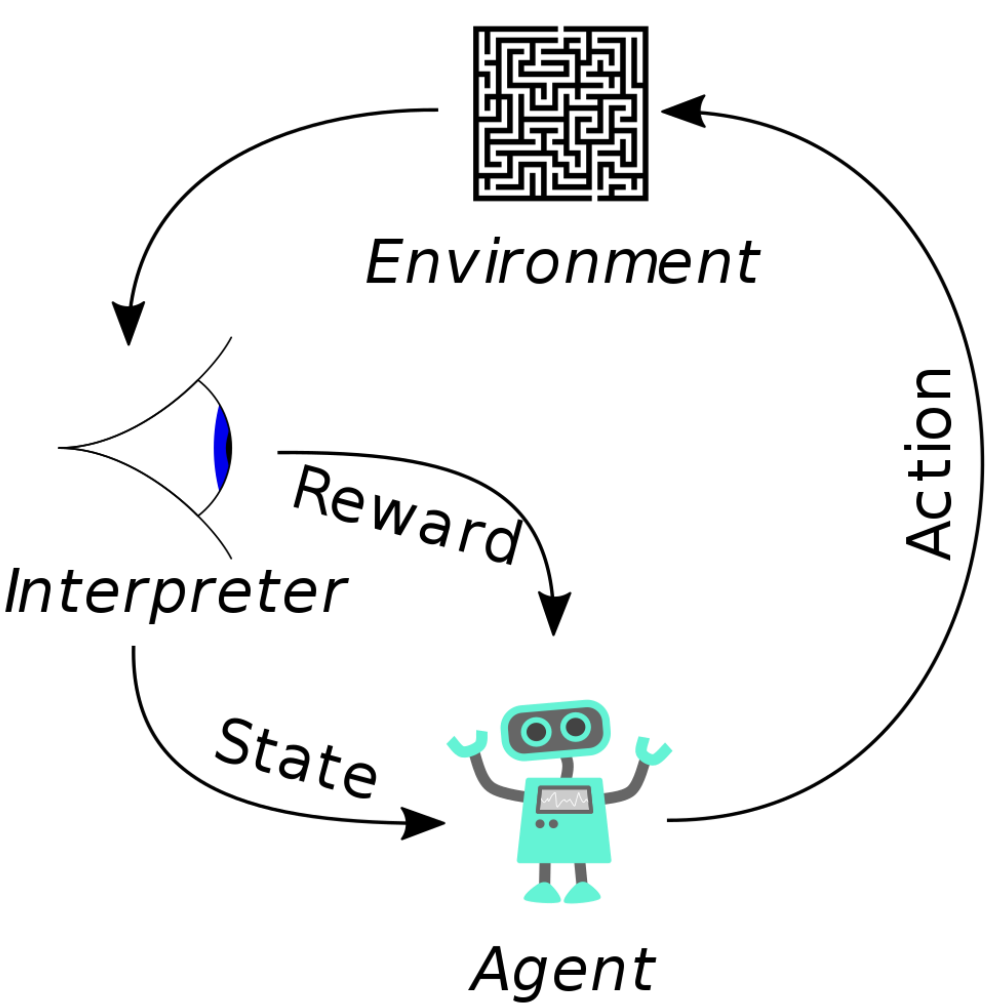
Example #2: GCPN

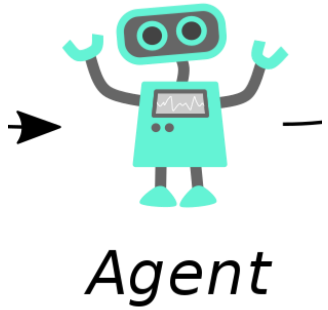


Ref: [You et al. NuerIPS. (2018)]

Recent idea: Make an intelligent agent that builds molecules

What is reinforcement learning?





The agent operates using a policy:

$$\pi(s, a) = \mathrm{P}(\mathrm{a}|\mathrm{s})$$

that produces a distribution of possible actions given the state

Our goal is to learn a policy that "optimizes reward"

How do we learn that policy?

Problem: You cannot just measure policy*

What can we measure?

The "reward" (R) for taking a certain action (a) in state (s)

How can we get those measurements? Interacting with the environment!
Making actions over many *steps* with the algorithm over many *episodes*, guided by the current policy (*rollout*)

New Problem: Ok, so we now have a dataset of many different (s, a, R). Now what?

*Not always true, you could decide an expert (e.g., humans) make optimal policy decisions

Simple Method: “Policy Gradient”

Idea: Update your policy ($\pi_\theta(s, a)$) to make good actions more often

Math*: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{a_t}, \boldsymbol{s_t}) \boldsymbol{V_t^*}$

$\boldsymbol{a_t}$ and $\boldsymbol{s_t}$ are things we measure directory

$\boldsymbol{V_t^*}$ is based on the ($\boldsymbol{r_t}$) rewards we measure for future steps

$$V_t = \sum_{i=0}^{n} \gamma^i r_{t+i}$$

subtracted from the “average reward”: $V_t^* = V_t - \mathbb{E}[V_t]$

so that good actions get positive and bad actions get negative rewards

*this equation is not arbitrary; it has a proof. See Chris Yoon’s excellent blog posts

A less-simple method: Actor Critic

Idea: Learn a better value function, one that looks <u>infinitely</u> far in the future

Math*: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{a_t}, \boldsymbol{s_t}) \boldsymbol{Q}_w(s_t, a_t)$

We now have two models:

1. **Actor ($\boldsymbol{\pi_\theta(a_t, s_t)}$): Generates moves**
2. **Critic ($\boldsymbol{Q_w(a_t, s_t)}$):** Evaluates value of move

Train the "Critic" function by coming a "time-dependent" error from "Bellman Equation":

$$\delta_t = [\boldsymbol{r_t} + \gamma Q_w(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}})] - Q_w(\boldsymbol{s_t}, \boldsymbol{a_t})$$

Error is to make prediction of a moves' reward ($Q_w(\boldsymbol{s_t}, \boldsymbol{a_t})$)

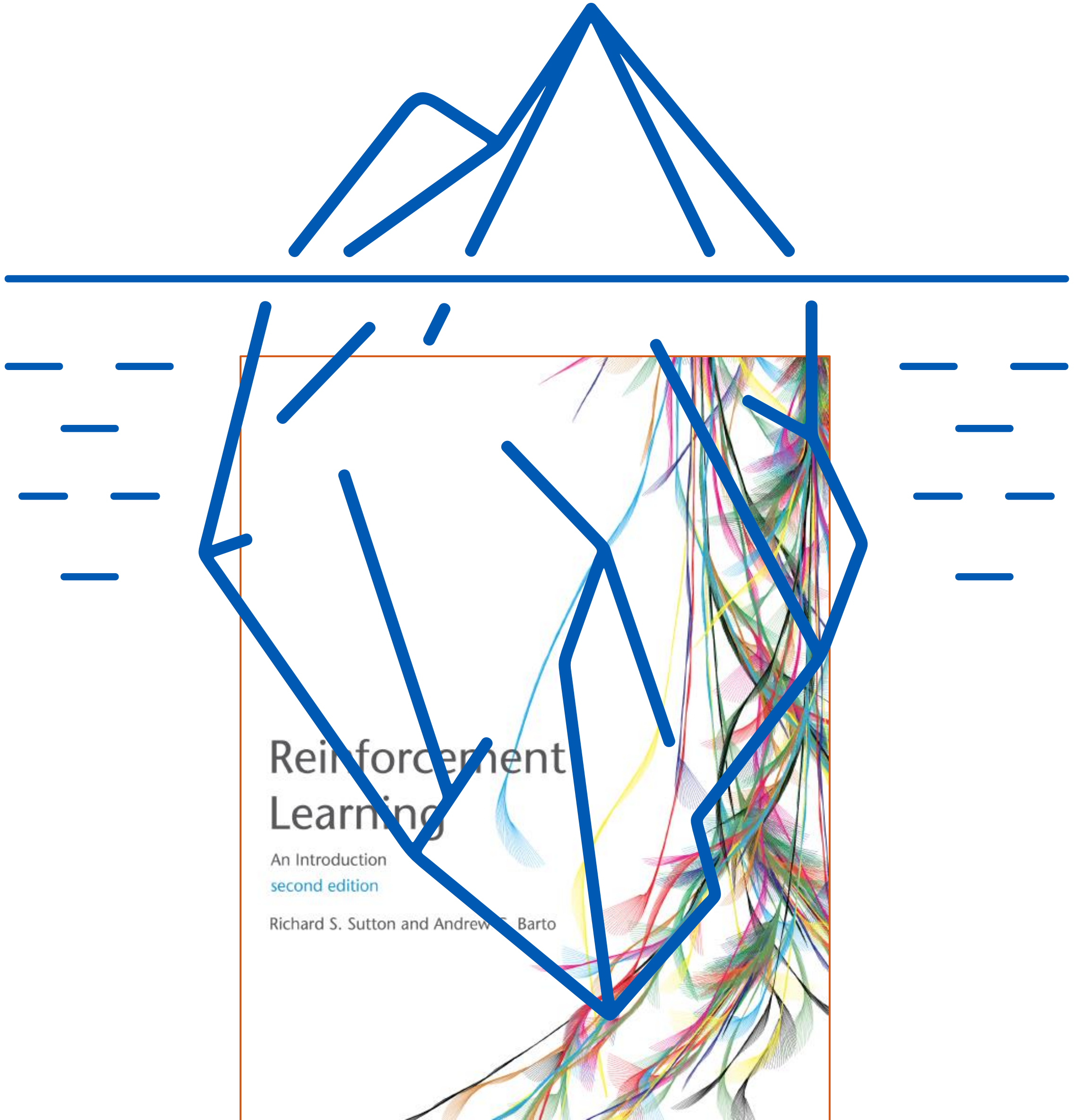
closer to value of the reward (r_t) plus a discounted (γ) value of next move ($Q_w(\boldsymbol{s_{t+1}}, \boldsymbol{a_{t+1}})$)*

Update the weights accordingly

$$w \leftarrow w + \alpha \delta_t \nabla_w Q_w(s_t, a_t)$$

*inductive relationship means after many updates, we achieve infinite lookahead
(step t+1 is influenced by t+2, t+2 by t+3, …)
See Chris Yoon's excellent blog posts

This lecture only scratches the surface



Today's lecture

A field with a [500+ page textbook]

Doing this in practice?

Several interdependent choices:

- **Training Algorithm:** How are we going to express and learn a policy?
- **Environment:** How do we define the state and actions for an environment?
- **Learnable Functions:** How do we represent state and actions? What models to use?

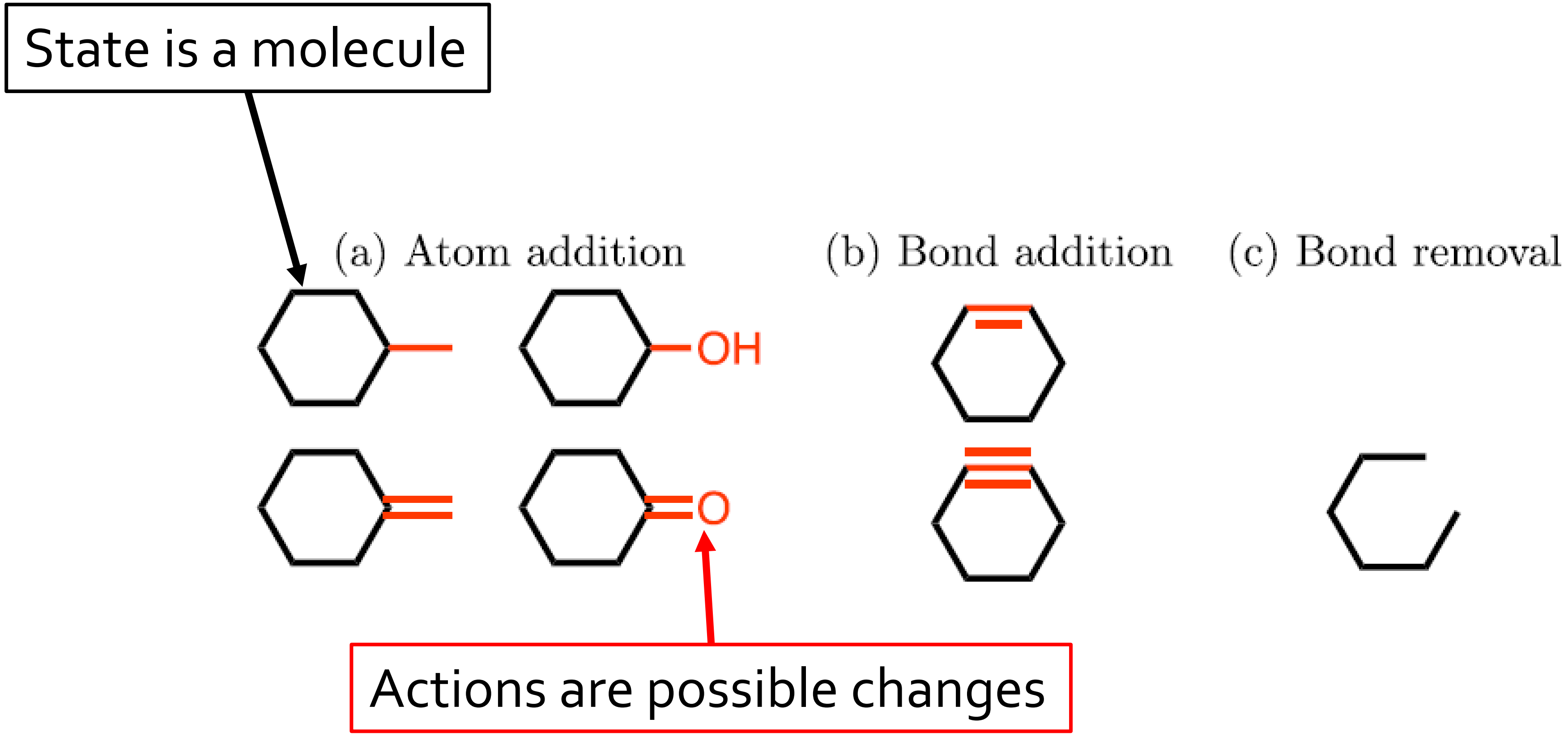
EXAMPLE #1: MOLDQN

Ref: Zhou et al. Sci Rep. (2019)

How do I train it? “Deep Q-Learning”

	Human	Math
What is the “Q function”?		
	“Q” is the value of an action at a certain state	$Q(s, a) = \mathrm{E}_\pi[\sum r_t]$
	Action with best Q is the best choice	$\pi^*(s) = \arg\max_a Q(s, a)$
How do I train it?		
	Minimize the “lookahead/Bellman equation,”	$l(\theta) = \mathrm{E}\left(f_l\left(y_t - Q(s_t, a_t)\right)\right)$
	which updates value to match reward + Q value of the next step	$y_t = r_t + \max_a Q(s_{a+1}, a)$

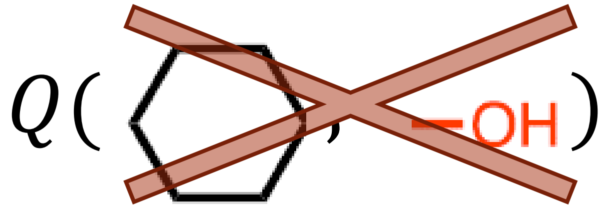
What is the environment?



What are the functions?

What they don't do: Represent current molecule and action separately

How do you represent a bond addition, deletion or atom addition in the same way!?

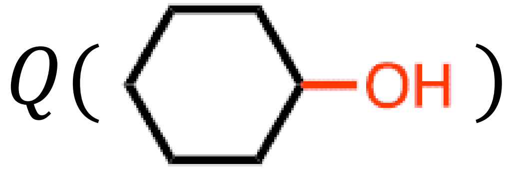


Theory aside:

1. Q-learning takes pairs of state (s) and action (a)
2. State + action do not always predict the next state (s′)
3. Unless the process is deterministic (same state, same action, same result)

Simplification: Use the next state (a whole molecule) [An easy task]

They chose Morgan fingerprints with multi-layer perceptions



EXAMPLE 2: GRAPH CONVOLUTIONAL POLICY NETWORK (GCPN)

Ref: You et al. NuerIPS. (2018)

How do I train it? PPO (at a High-Level)

High-level Goal: Maximize taking 'advantageous moves'

$$L(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\boldsymbol{t}}\right)\right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t, s_t)}{\pi_{\theta_{old}}(a_t, s_t)}$

Adjust θ so that $\pi(s, a)$ becomes bigger for advantageous moves

Hard part is defining the "Advantage Function" ($\widehat{\mathrm{A}}_{\mathrm{t}}$)

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

Something we can compute

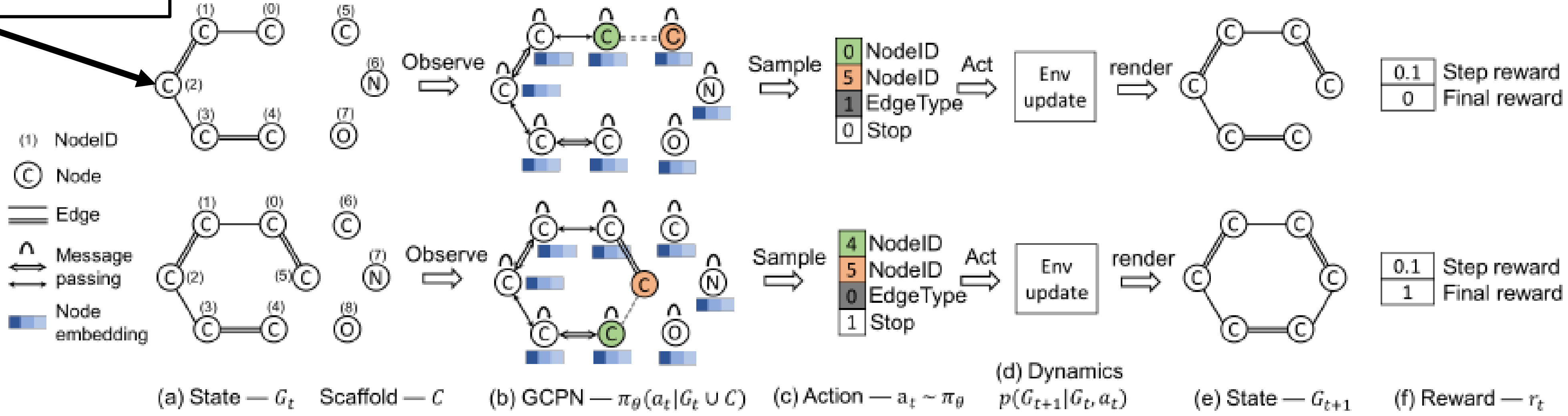
A function we can learn with similar "lookahead" approaches

Another "trick:" Keeping $\pi(a_t, s_t)$ diverse with an "entropy" prediction

Full details: Schman et al.

What does my environment look like?

State is a molecule



Actions are: Atom at beginning and end of bond, bond type and whether to stop

$$a_t = \text{CONCAT}(a_{\text{first}}, a_{\text{second}}, a_{\text{edge}}, a_{\text{stop}})$$

What are the functions?

Message passing networks (graph convolutions) are used to generate atom features (X)

$$f_{\text{first}}(s_t) = \text{SOFTMAX}(m_f(X)), \qquad a_{\text{first}} \sim f_{\text{first}}(s_t) \in \{0,1\}^n$$

First atom: Generate probability given atom features

$$f_{\text{second}}(s_t) = \text{SOFTMAX}(m_s(X_{a_{\text{first}}}, X)), \qquad a_{\text{second}} \sim f_{\text{second}}(s_t) \in \{0,1\}^{n+c}$$

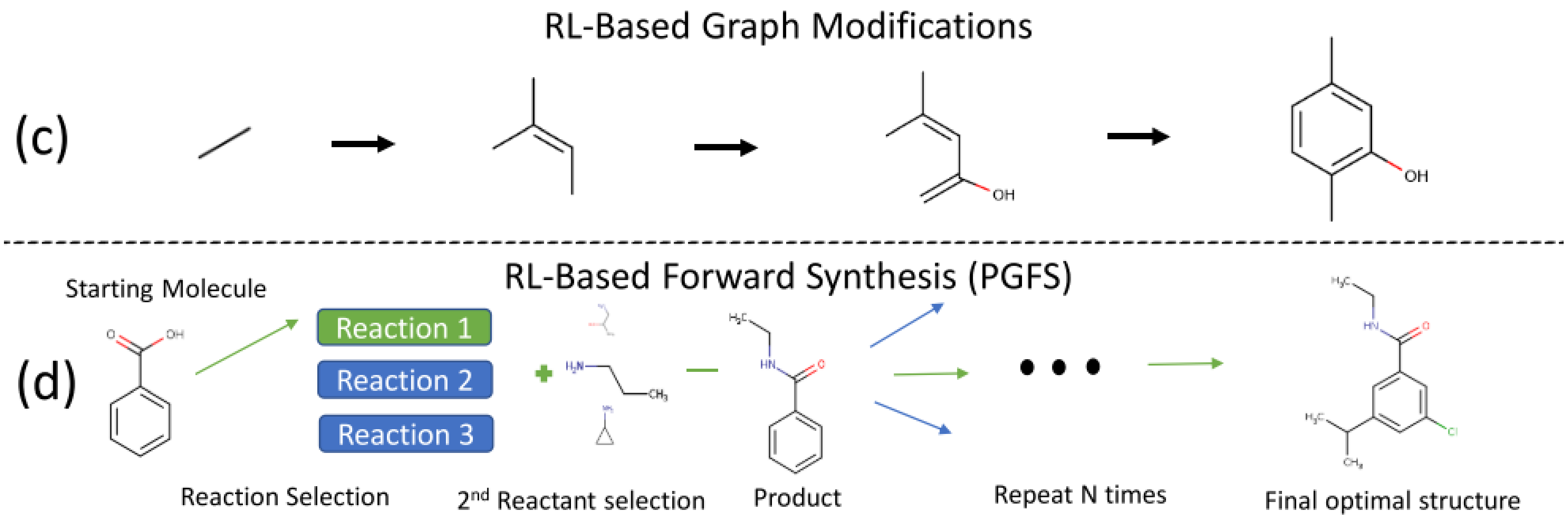
Second atom: Generate probability given first atom's features *and* all node features

$$f_{\text{edge}}(s_t) = \text{SOFTMAX}(m_e(X_{a_{\text{first}}}, X_{a_{\text{second}}})), \qquad a_{\text{edge}} \sim f_{\text{edge}}(s_t) \in \{0,1\}^b$$

Bond type: Generate probabilities given features of first and second atom

Newer Iteration: Reaction-template guided RL

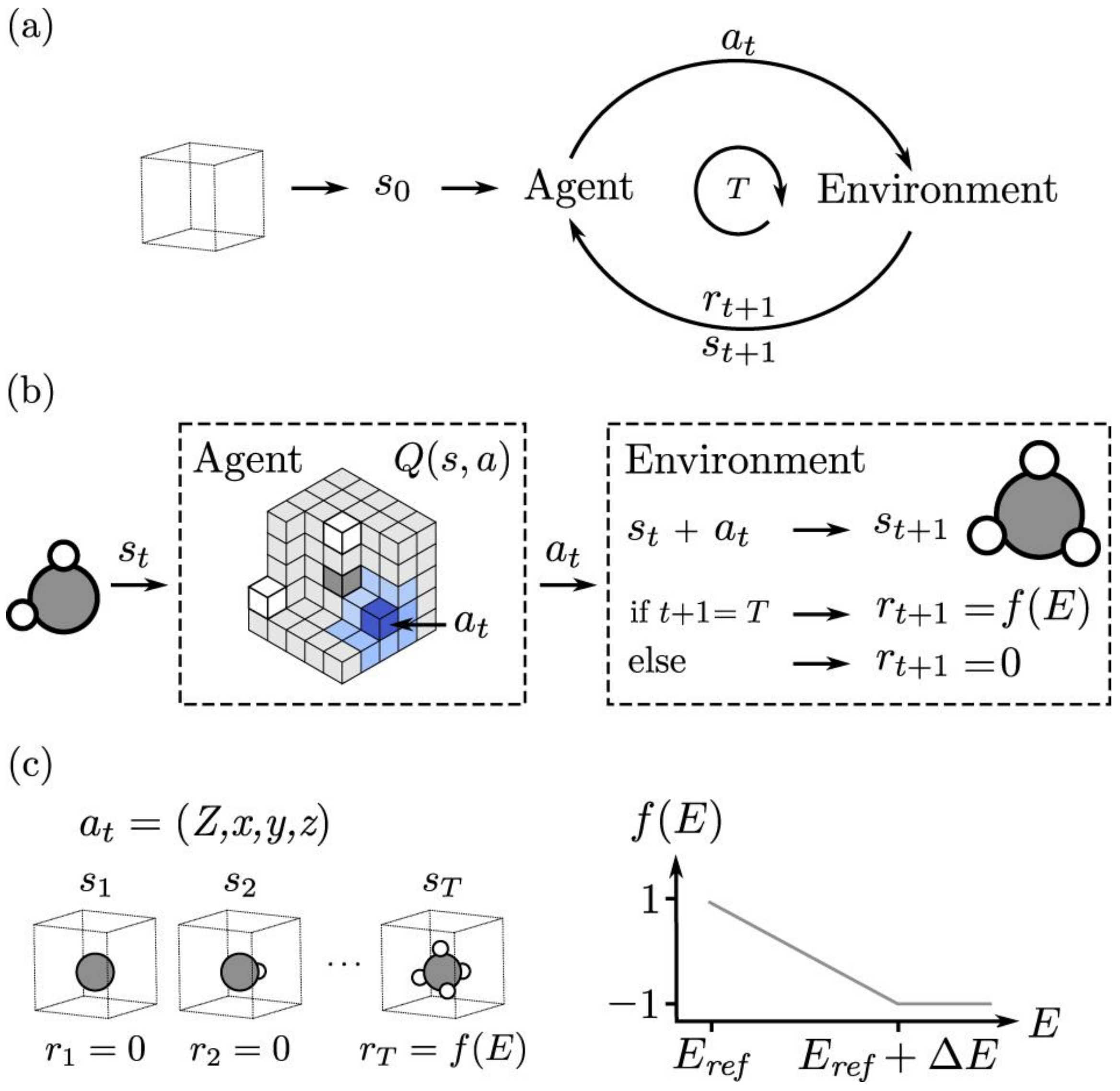
Or example methods chose moves as graph additions



Newer approaches limit choices by selecting reaction products

Ref: Gottipati et al. (2020)

Another Example: Q-Learning for Atomic Structures



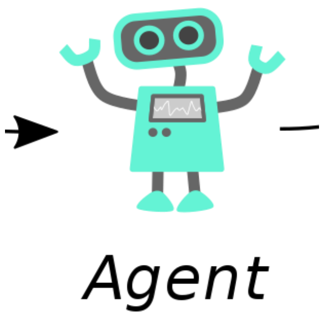
Ref: Meldgaard et al. JP:CM (2020)

Take Away Points

RL at a glance

Concept: Learn a policy by

1. interacting with an environment
2. adjusting based on rewards



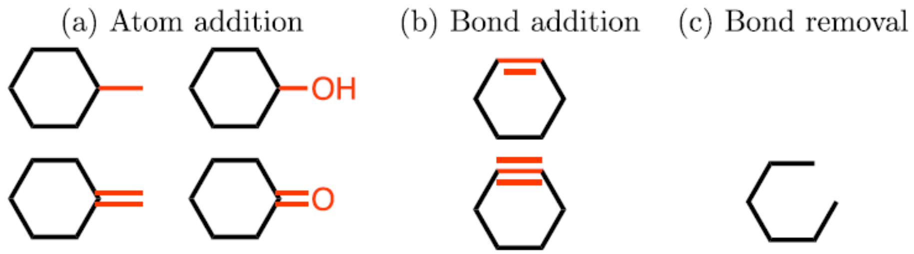
Difficult math:

Many ways to learn policy:

1. Q-learning
2. PPO
3. Actor critic
4. …

Making your own environment

1. Implement an environment



2. Choose learning policy

3. Make representations state and action, implement value/actor/critic/Q functions