Reading Group on Reinforcement Learning - Hi! Paris

Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation

Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, Jure Leskovec

Stanford University
Published @NeurIPS 2018 (∼700 citations)

Outline

1. Brief introduction to RL (C. Laclau)
2. Presentation of the paper (C. Laclau)
3. Demo (G. Brison)

Machine Learning Settings[1]

Supervised Learning

Input: (x, y) where x is the **predictor** and y is a given **label**.
Goal Learn a function $f : x :\rightarrow y$
Examples: classification, regression

Unsupervised Learning

Input: x, **no labels**
Goal: Learn some underlying hidden structure of the data
Examples: clustering, dimensionality reduction, feature learning

Reinforcement Learning

Input: an **agent** interacting with an **environment**, which provides numeric **reward** signals
Goal : learn how to take action in order to maximize rewards

[1] inspired from Fei-Fei Li (Standford course)

Machine Learning Settings[1]

Supervised Learning

Input: (x, y) where x is the **predictor** and y is a given **label**.
Goal Learn a function $f : x :\rightarrow y$
Examples: classification, regression

Unsupervised Learning

Input: x, **no labels**
Goal: Learn some underlying hidden structure of the data
Examples: clustering, dimensionality reduction, feature learning

Reinforcement Learning

Input: an **agent** interacting with an **environment**, which provides numeric **reward** signals
Goal : learn how to take action in order to maximize rewards

[1] inspired from Fei-Fei Li (Standford course)

Machine Learning Settings[1]

Supervised Learning

Input: (x, y) where x is the **predictor** and y is a given **label**.
Goal Learn a function $f : x :\rightarrow y$
Examples: classification, regression

Unsupervised Learning

Input: x, **no labels**
Goal: Learn some underlying hidden structure of the data
Examples: clustering, dimensionality reduction, feature learning

Reinforcement Learning

Input: an **agent** interacting with an **environment**, which provides numeric **reward** signals
Goal : learn how to take action in order to maximize rewards

[1] inspired from Fei-Fei Li (Standford course)

Where to start? Some useful resources

- Course by David Silver
  `https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf`
- Course UCL $\times$ DeepMind by Hado van Hasselt (available on Youtube)
- Reinforcement Learning: an Introduction [Sutton & Barto]
  `https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf`
- Lilian Weng's blog post - A (Long) Peek into Reinforcement Learning
  `https://lilianweng.github.io/posts/2018-02-19-rl-overview/#key-concepts`
- Andrej Karpathy's blog post - Deep Reinforcement Learning: Pong from Pixels
  `http://karpathy.github.io/2016/05/31/rl/`

What is Reinforcement learning? [2]

Observations

- People (and animals) learn by **interacting with our environment**
- We are **goal oriented**
- We can learn **without examples** of optimal behaviour
- Instead we optimize some **reward signal**

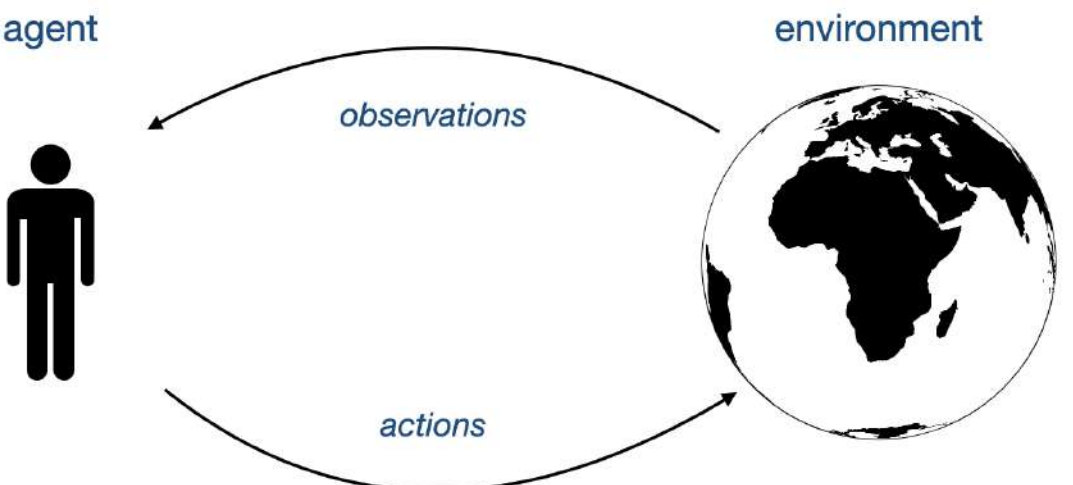
Differs from other learning scenario

- RL is **active** rather than passive
- Interactions are often **sequential** - future interaction can depend on earlier ones

[2] Slide from DeepMind x UCL RL Lecture Series

Reinforcement Learning

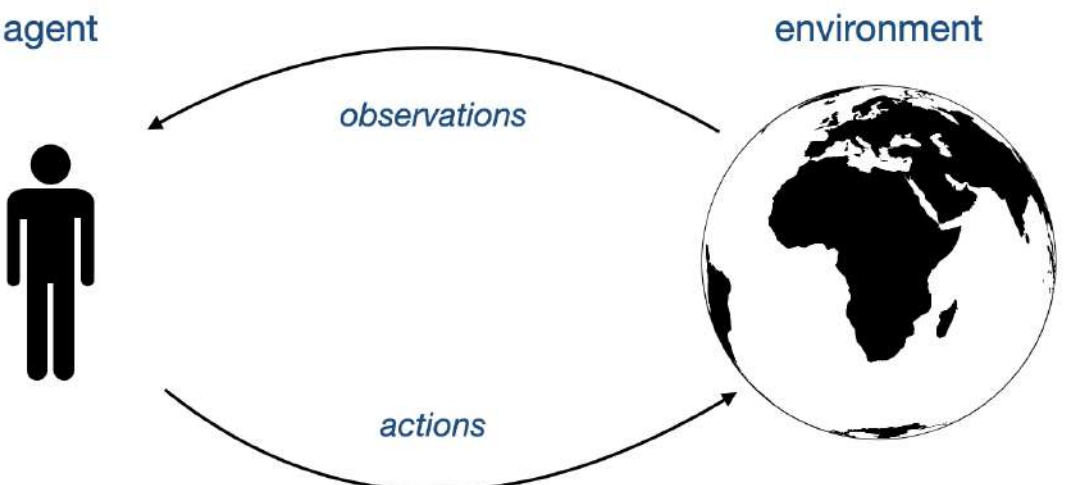
The Interaction Loop [3]



[3] Slide from DeepMind x UCL RL Lecture Series

Reinforcement Learning

The Interaction Loop [3]



Goal: optimize sum of rewards, through repeated interactions

[3] Slide from DeepMind x UCL RL Lecture Series

Reward Hypothesis

Any goal can be formalized as the outcome of maximizing a cumulative reward

Examples of RL problems

- Fly a helicopter
- Manage an investment portfolio
- Control a power station
- Make a robot walk
- Play video or board games

Rewards

→ air time, inverse distance
→ gains, gains minus risk
→ efficiency
→ distance, speed
→ win, maximize scores

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case
- This requires us to think about
  - ... time

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case
- This requires us to think about
  - ... time
  - ... (long term) consequences of actions

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case
- This requires us to think about
  - ... time
  - ... (long term) consequences of actions
  - ... actively gathering experiences

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case
- This requires us to think about
  - ... time
  - ... (long term) consequences of actions
  - ... actively gathering experiences
  - ... predicting the future

Reinforcement Learning

General definition

Science and framework of **learning to make decisions** from **interactions**

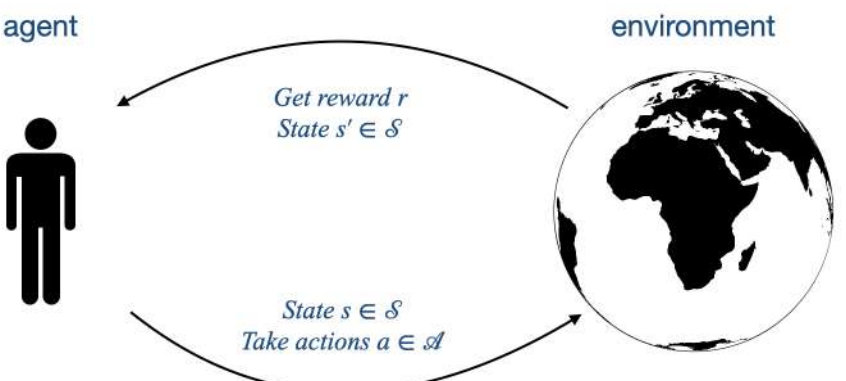
There are different reasons to learn

1. Find **solutions**
   - a program that plays chess very well
   - a robot that can learn to navigate unknowns terrains
2. **Adapt online**, deal with unseen circumstances
   - a chess program that can learn how to adapt to you
   - a robot that can learn to navigate unknown terrains

- Reinforcement learning can provide algorithms for both case
- This requires us to think about
  - ... time
  - ... (long term) consequences of actions
  - ... actively gathering experiences
  - ... predicting the future
  - ... dealing with uncertainty

Reinforcement Learning

Formalising the RL problem[4]



[4] Section 3.2 in the paper

Reinforcement Learning

Formalising the RL problem

- $\mathcal{S} = s_t$ is the set of states
- $\mathcal{A} = a_t$ is the set of actions
- P is the transition dynamics $P(s_{t+1}|s_t, \cdots, s_0, a_t)$
- R_t the reward, is a scalar feedback signal
  - Indicated how well (or bad) the agent is doing at each step t - defines the goal
  - The agent's job is to maximize the cumulative reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots$$

- Episode (also known as *trial* or *trajectory*) fully describe an interaction sequence up to a terminal state T

$$s_0, a_0, r_0, s_1, \cdots, s_T$$

Policy

- A policy defines the agent's behaviour: it maps from an agent's state to an action
  - Deterministic policy: $A = \pi(S)$
  - Stochastic policy: $\pi(A|S) = p(A|S)$

Goal: select actions to maximise value

- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward

Examples:

- Refueling a helicopter (might prevent a crash in several hours)
- Defensive moves in a game (may help chances of winning later)
- Learning a new skill (can be costly & time-consuming at first)

All the RL problems can be framed as Markov Decision Processes (MDPs).

- All states in MDP has **Markov** property, referring to the fact that the future only depends on the current state, not the history:

$$\mathbb{P}(S_{t+1}|S_t] = \mathbb{P}(S_{t+1}|S_1, \cdots, S_t)$$

- A Markov decision process consists of five elements $< \mathcal{S}, \mathcal{A}, P, R, \gamma >$ where
  - $\mathcal{S}$ is a set of states
  - $\mathcal{A}$ is a set of actions
  - P is the transition probability function
  - R is a reward function
  - γ is a discounting factor for future rewards

Value Function

- Measures the goodness of a state or how rewarding a state or an action is by a prediction of future reward, i.e. the return

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discounting factor γ penalize the rewards in the future
- The state-value of a state s is the expected return if we are in this state at time t

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S = s_t]$$

- The action-value (Q-value) of a state-action pair as:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S = s_t, A = a_t]$$

Value Function

- Measures the goodness of a state or how rewarding a state or an action is by a prediction of future reward, i.e. the return

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discounting factor γ penalize the rewards in the future
- The state-value of a state s is the expected return if we are in this state at time t

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S = s_t]$$

- The action-value (Q-value) of a state-action pair as:

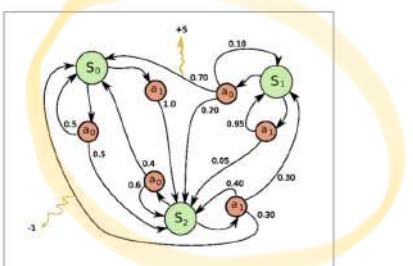
$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S = s_t, A = a_t]$$

Popular algorithms to learn the optimal policy

- Q-learning, Policy gradient, Proximal Policy Gradient

Deep Reinforcement Learning [5]

- Let's consider the example of Pong
- Input : an image frame (a 210x160x3 byte array)
- Actions: move up or down
- Reward: +1 if the ball went past the opponent (win), -1 if we missed the ball (loose), 0 otherwise (game continue)

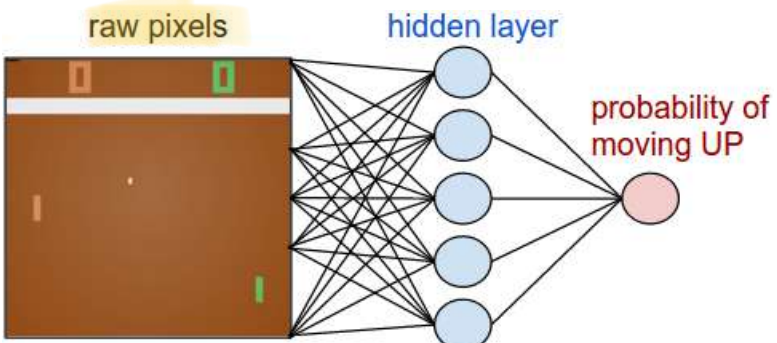


Left: The game of Pong. **Right:** Pong is a special case of a Markov Decision Process (MDP): A graph where each node is a particular game state and each edge is a possible (in general probabilistic) transition. Each edge also gives a reward, and the goal is to compute the optimal way of acting in any state to maximize rewards.

[5] Example is taken from the really good blog post of `http://karpathy.github.io/2016/05/31/rl/`

Deep Reinforcement Learning [6]

Let's implement our agent as a **policy network**

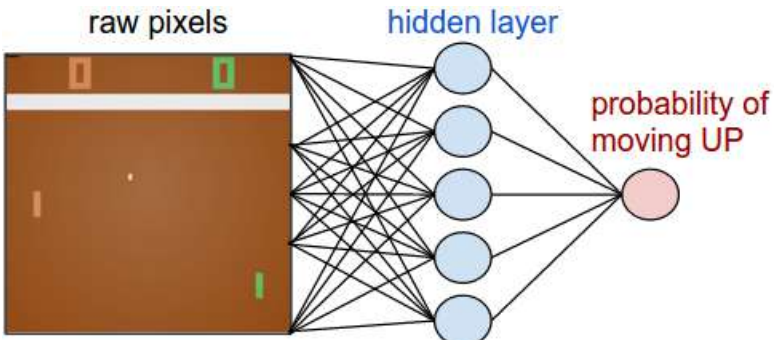


- Sample an action from the output distribution of the network $\rightarrow$ execute that action in the game

[6] Example is taken from the really good blog post of `http://karpathy.github.io/2016/05/31/rl/`

Deep Reinforcement Learning [6]

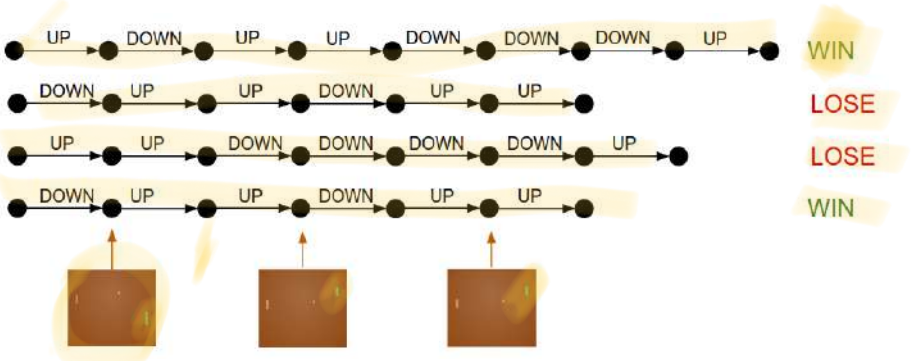
Let's implement our agent as a **policy network**



- Sample an action from the output distribution of the network $\rightarrow$ execute that action in the game
- Wait until the end of the game, then take the reward we get (either +1 if we won or -1 if we lost), and update the gradients accordingly

[6] Example is taken from the really good blog post of `http://karpathy.github.io/2016/05/31/rl/`

Deep Reinforcment Learning [7]



- With Policy Gradients we would take the two games we won and slightly encourage every single action we made in that episode.
- Conversely, we would also take the two games we lost and slightly discourage every single action we made in that episode.
- Similar to supervised learning, but on a continuously changing dataset (the episodes), scaled by the advantage, and we only want to do one (or very few) updates based on each sampled dataset.

[7] Example is taken from the really good blog post of `http://karpathy.github.io/2016/05/31/rl/`

Today's paper

Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation

Jiaxuan You[1]*
jiaxuan@stanford.edu

Bowen Liu[2]*
liubowen@stanford.edu

Rex Ying[1]
rexying@stanford.edu

Vijay Pande[3]
pande@stanford.edu

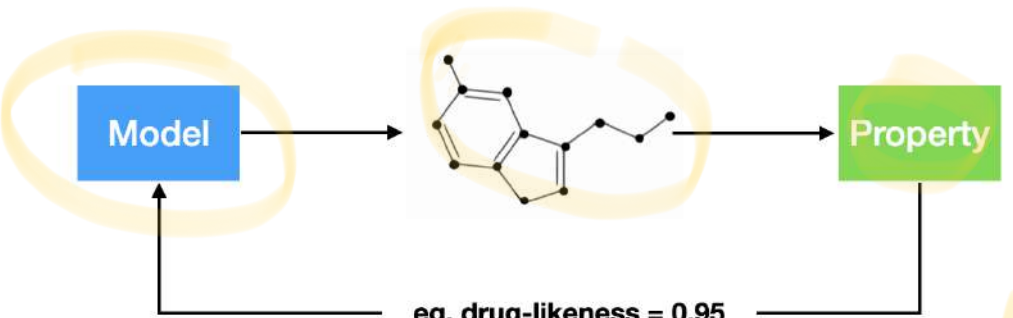
Jure Leskovec[1]
jure@cs.stanford.edu

[1]Department of Computer Science, [2]Department of Chemistry, [3]Department of Bioengineering
Stanford University
Stanford, CA, 94305

RL for Drug Discovery

Research Question

Can we learn a model that can generate **valid** and **realistic** molecules with **optimized** property scores ?



- valid = obeys the rules of chemistry
- realistic = looks like a drug (no Frankenstein type molecule)
- optimize: e.g. maximize a score

Key Components

Graph Convolution Policy Network

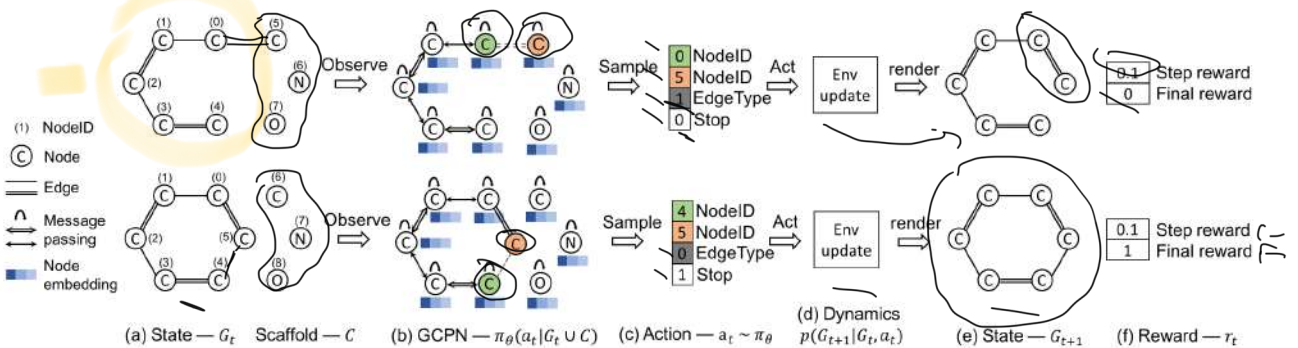
Combine Graph Neural Network and Reinforcement Learning

- **Graph Neural Networks** captures the graph structural information
- **Reinforcement Learning** guides the generation toward the desired objectives
- **Supervised Learning** imitates examples from a given dataset

Graph Generation as a RL problem

- **Agent** = Graph Neural Network (policy network)
- **Actions** = add a bond (of a certain type) between two atoms
  - **validity**: *infeasible actions proposed by the policy network are rejected and the state remains unchanged*
- **States** = all intermediate graphs and the final graph $\{G_0, \cdots, G_n\} \rightarrow$ fully observable
- **State transition dynamic**: domain-specific rules are incorporated in the state transition dynamics $\rightarrow$ the environment carries out actions that obey the given rules
- Primary objective = optimize a given property $S(G)$, ie, $\mathbb{E}_{S'}[G(S')]$
- Secondary objective = add prior knowledge using training examples

GCPN - OVERALL PICTURE [8]



(a) Insert new nodes

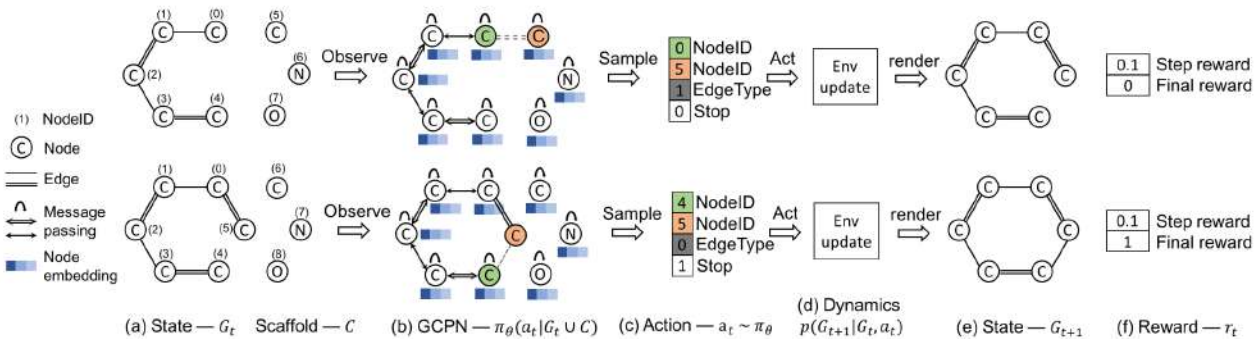
(b), (c) Use GNN to predict which nodes to connect

(d) Take an action (check validity)

(e), (f) Compute rewards

[8] Slide from Leskovec

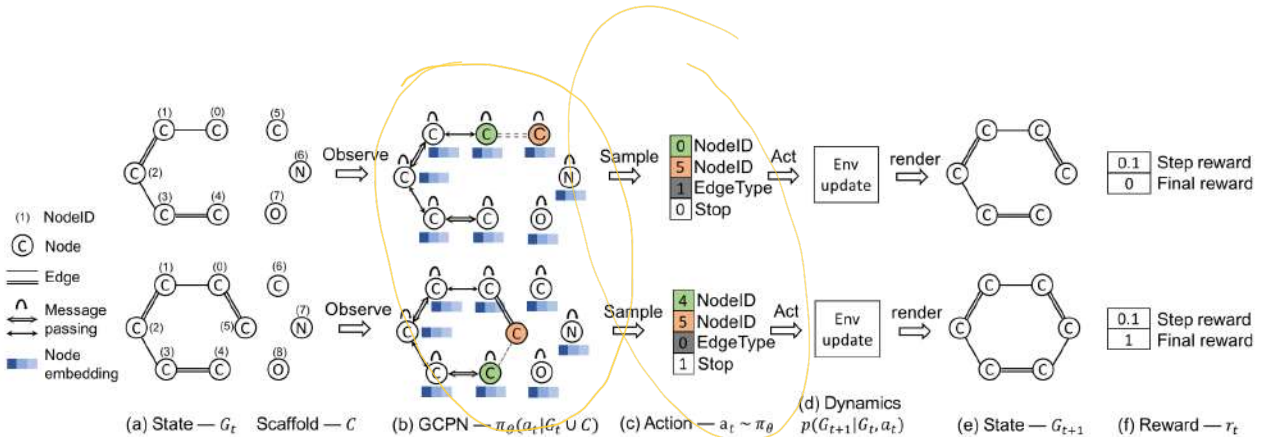
GCPN - OVERALL PICTURE [9]



- **Step rewards**: learn to take valid actions
  - At each step, assign small positive reward for valid action
- **Final reward**: optimize desired properties
  - At the end, assign positive reward for high desired property
  - **Reward**: step rewards + final rewards

[9] Slide from Leskovec

GCPN - OVERALL PICTURE [10]

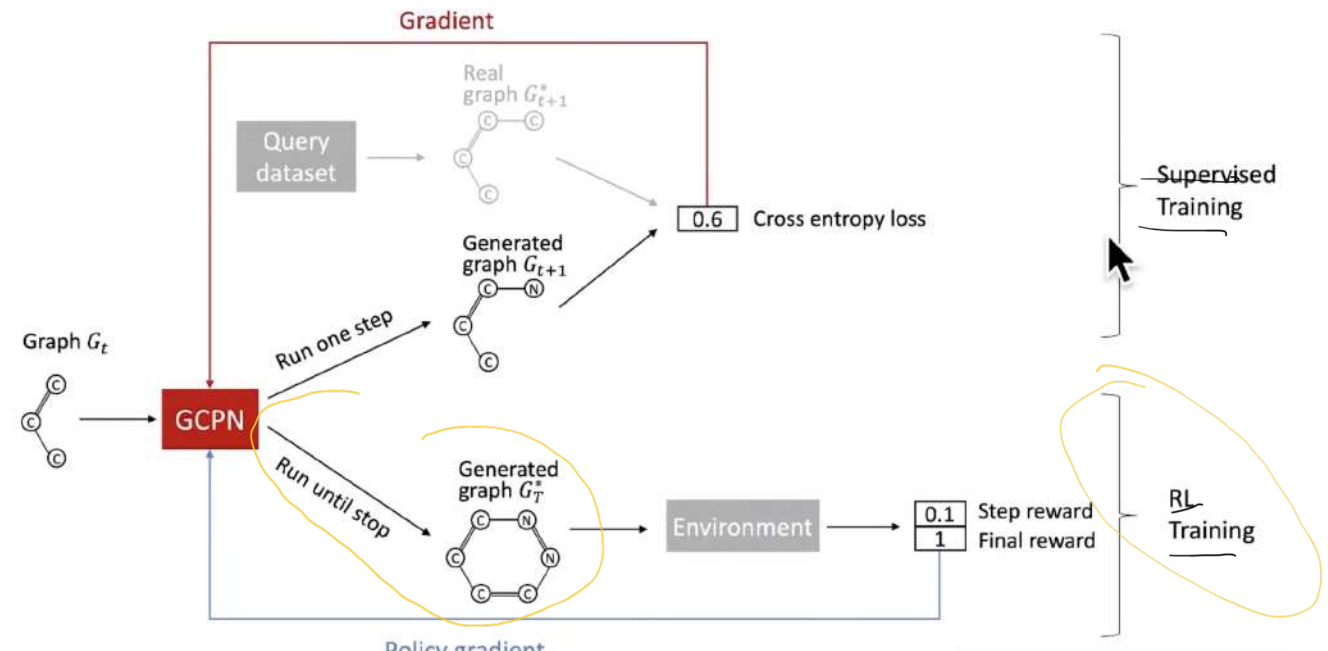


Training in two parts

- **Supervised Training**: train policy by imitating the action given by a real observed graph (gradient based)
- **RL training**: Train policy to optimize rewards (standard policy gradient algorithm)

[10] Slide from Leskovec

GCPN OVERALL PICTURE [11]



[11] Slide from Leskovec

Details on Reward design

WARNING: I'm not sure of this part!

Intermediate Reward

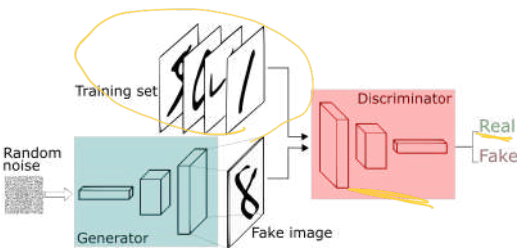
- Small reward for **chemical validity** (valency rule) otherwise a small negative reward is applied
- Adversarial reward for realistic molecules (w.r.t to some training set)

Final Reward

- Calculated when the trajectory stops
- Domain-specific rewards: combination of final property scores + penalization of unrealistic molecules (eg. excessive steric strain)
- Final reward = Domain-specific rewards + Adversarial rewards

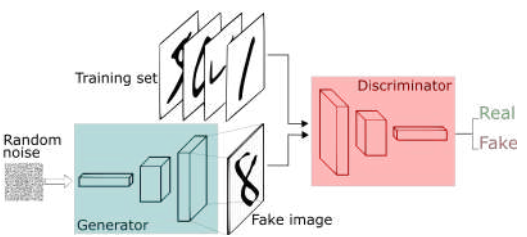
THE ADVERSARIAL REWARD

- Idea: use a training set of molecules to integrate prior knowledge with Generative Adversarial Network (GAN)



THE ADVERSARIAL REWARD

- Idea: use a training set of molecules to integrate prior knowledge with Generative Adversarial Network (GAN)



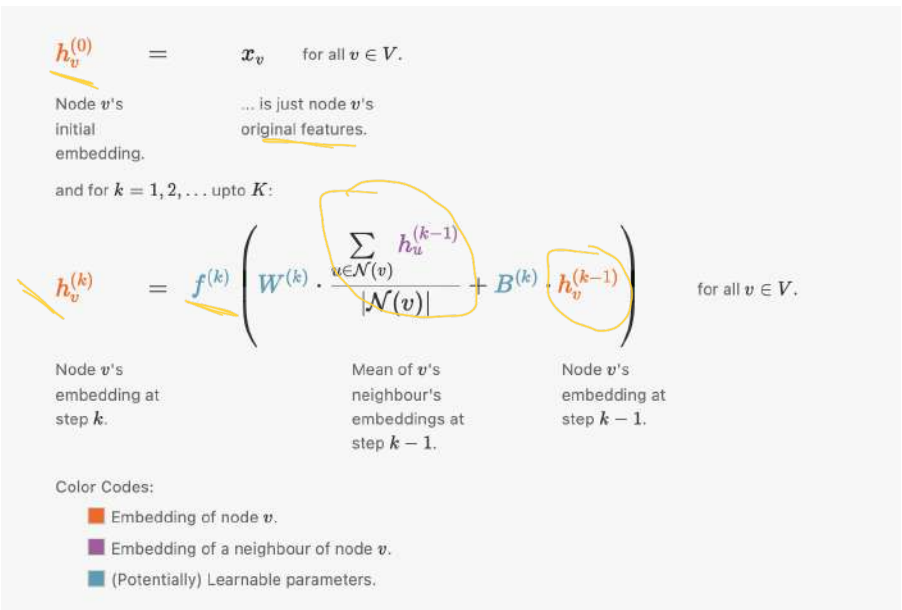
To ensure that the generated molecules resemble a given set of molecules, we employ the Generative Adversarial Network (GAN) framework [10] to define the adversarial rewards $V(\pi_\theta, D_\phi)$

$$\min_\theta \max_\phi V(\pi_\theta, D_\phi) = \mathbb{E}_{x\sim p_{data}}[\log D_\phi(x)] + \mathbb{E}_{x\sim \pi_\theta}[\log D_\phi(1-x)] \quad (1)$$

where π_θ is the policy network, D_ϕ is the discriminator network, x represents an input graph, p_{data} is the underlying data distribution which defined either over final graphs (for final rewards) or intermediate graphs (for intermediate rewards). However, only D_ϕ can be trained with stochastic gradient descent, as x is a graph object that is non-differentiable with respect to parameters ϕ. Instead, we use $-V(\pi_\theta, D_\phi)$ as an additional reward together with other rewards, and optimize the total

Details on (b)GCPN

Graph Convolutional Network (GCN) to learn node embedding



Details on (b)GCPN

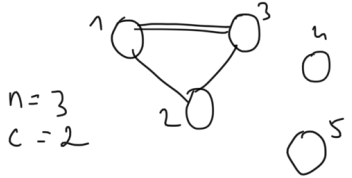
From node embeddings to taking actions

- An action a_t is a concatenation of four components: selection of two nodes, prediction of edge type, and prediction of termination.
- Each component is sampled according to a predicted distribution governed

$$a_t = \text{CONCAT}(a_{\text{first}}, a_{\text{second}}, a_{\text{edge}}, a_{\text{stop}}) \tag{3}$$

$$\begin{aligned}
&f_{\text{first}}(s_t) = \text{SOFTMAX}(m_f(X)), && a_{\text{first}} \sim f_{\text{first}}(s_t) \in \{0,1\}^n \\
&f_{\text{second}}(s_t) = \text{SOFTMAX}(m_s(X_{a_{\text{first}}}, X)), && a_{\text{second}} \sim f_{\text{second}}(s_t) \in \{0,1\}^{n+c} \\
&f_{\text{edge}}(s_t) = \text{SOFTMAX}(m_e(X_{a_{\text{first}}}, X_{a_{\text{second}}})), && a_{\text{edge}} \sim f_{\text{edge}}(s_t) \in \{0,1\}^b \\
&f_{\text{stop}}(s_t) = \text{SOFTMAX}(m_t(\text{AGG}(X))), && a_{\text{stop}} \sim f_{\text{stop}}(s_t) \in \{0,1\}
\end{aligned} \tag{4}$$

EXAMPLE FOR PREVIOUS SLIDE



$n = 3$
$c = 2$

① Sample a 1rst node

input $X \in \mathbb{R}^{(n+c) \times k} \longrightarrow \mathbb{R}^n$ eg $[0.3, 0.2, 0.5]$

probability for each node to be sample.

② Assume node 3 has been picked as first node

Concatenate embedding X_3 with X

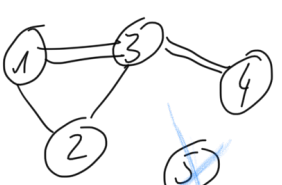
$$X' \in \mathbb{R}^{(n+c) \times 2k} \begin{bmatrix} X_3 & X_3 & X_3 & X_3 & X_3 \\ X_1 & X_2 & X_3 & X_4 & X_5 \end{bmatrix} \longrightarrow \mathbb{R}^{(n+c) \times k}$$

eg of type: [0.1 0.2 0 0.6 0.1] ← proba of an edge between nodes 3 and 4

③ $[X_3, X_4]$ $\xrightarrow{\text{edge type}}$ $[0.1\ 0.9] \in \mathbb{R}^2$

④ $Agg(X) \longrightarrow \mathbb{R}^2$ $[0.7, 0.3]$ stop?

↑ eg. mean



Policy Gradient Training

Policy Gradient objective

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right].$$

Expected

Policy Loss

log(probabilities) from the output of the policy network

Estimate of the relative value of selected action

- The advantage $\hat{A}_t$ is an estimation of the relative value for a_t in current state s_t.

$$A_t = Q(s_t, a_t) - V(s_t)$$

- Quantify the extra reward that could be obtained by the agent by taking a particular action
- $\hat{A}$ positive means that the actions the agent took in the sample trajectory resulted in better than average return
  - policy gradient would be positive to increase the probability of selecting these actions again when we encounter a similar state

Policy Gradient Training

Policy Gradient objective

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log \pi_\theta(a_t \mid s_t)\hat{A}_t\right].$$

Expected

Policy Loss

log(probabilities) from the output of the policy network

Estimate of the relative value of selected action

- The advantage $\hat{A}_t$ is an estimation of the relative value for a_t in current state s_t.

$$A_t = Q(s_t, a_t) - V(s_t)$$

- Quantify the extra reward that could be obtained by the agent by taking a particular action
- $\hat{A}$ positive means that the actions the agent took in the sample trajectory resulted in better than average return
  - policy gradient would be positive to increase the probability of selecting these actions again when we encounter a similar state
- Destructively large policy updates: risk of updating the parameters too far

Policy Gradient Training

Proximal Policy Gradient (implemented by OpenAI)

- Idea: limit the policy gradient step so it does not move too much away from the original policy – Trust Region Policy Optimization (Schulman et al, 2015)

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

- PPO clip objective

3.5 Policy Gradient Training

Policy gradient based methods are widely adopted for optimizing policy networks. Here we adopt Proximal Policy Optimization (PPO) [35], one of the state-of-the-art policy gradient methods. The objective function of PPO is defined as follows

$$\max L^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \quad (5)$$

SUPERVISED TRAINING: EXPERT POLICY

It is known that pretraining a policy network with expert policies if they are available leads to better training stability and performance [24]. In our setting, any ground truth molecule could be viewed as an expert trajectory for pretraining GCPN. This expert imitation objective can be written as $\min L^{\text{EXPERT}}(\theta) = -\log(\pi_\theta(a_t|s_t))$, where (s_t, a_t) pairs are obtained from ground truth molecules. Specifically, given a molecule dataset, we randomly sample a molecular graph G, and randomly select one connected subgraph G' of G as the state s_t. At state s_t, any action that adds an atom or bond in $G \setminus G'$ can be taken in order to generate the sampled molecule. Hence, we randomly sample $a_t \in G \setminus G'$, and use the pair (s_t, a_t) to supervise the expert imitation objective.

Experiments (more details to come)

To demonstrate effectiveness of goal-directed search for molecules with desired properties, we compare our method with state-of-the-art molecule generation algorithms in the following tasks.

Property Optimization. The task is to generate novel molecules whose specified molecular properties are optimized. This can be useful in many applications such as drug discovery and materials science, where the goal is to identify molecules with highly optimized properties of interest.

Property Targeting. The task is to generate novel molecules whose specified molecular properties are as close to the target scores as possible. This is crucial in generating virtual libraries of molecules with properties that are generally suitable for a desired application. For example, a virtual molecule library for drug discovery should have high drug-likeness and synthesizability.

Constrained Property Optimization. The task is to generate novel molecules whose specified molecular properties are optimized, while also containing a specified molecular substructure. This can be useful in lead optimization problems in drug discovery and materials science, where we want to make modifications to a promising lead molecule and improve its properties [2].