Applying Reinforcement Learning to De Novo Design of Bioactive Compounds

Final Project Presentation

Gabriel Costanzo

University of Trieste -ICTP

October 31, 2024

Outline

Project Overview

This project builds on the research presented in the article:

[1] Korshunova et al. (2022)

Korshunova, M., Huang, N., Capuzzi, S., et al. (2022). Generative and reinforcement learning approaches for the automated de novo design of bioactive compounds. *Communications Chemistry, 5*, 129. https://doi.org/10.1038/s42004-022-00733-0

The objective of this project is to gain insights and evaluate methods for generating de novo molecules with specific biological properties using a Reinforcement Learning approach.
The original code from the repository was modified (with the assistance of AI) to extract metrics and rewards across various experimental settings.

Introduction

The proposed RL system uses:

- A deep generative neural networks (RNN) to de novo design molecules with desired properties.
- A predictive model (Random Forest) to predict the bioactivity of the new generated molecules.
- A Reinforcement learning algorithm for optimizing the bioactivity of the generated molecules.

Introduction: Molecular Representation

- SMILES (Simplified Molecular Input Line Entry System): A notation that allows a user to represent a chemical structure in a way that can be used by the computer.
- Example: Aspirin is represented as `CC(=O)OC1=CC=CC=C1C(=O)O`.
- SMILES strings are used to encode molecules in RL to generate new chemical structures

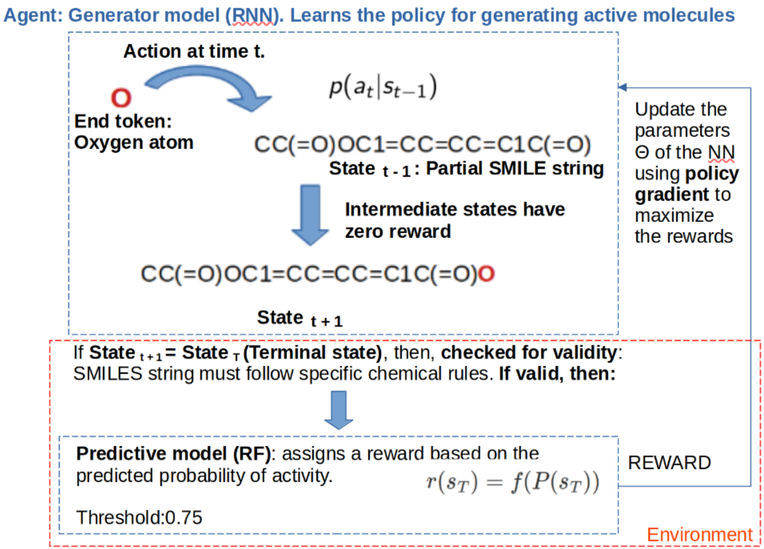
Problem Definition

Sparse Rewards and Exploration-Exploitation Trade-off Problem

An encounter of a molecule active against a specific target is a rare event.

This will result in over-exploration, a situation when the model mostly experiences low rewards for inactive molecules. At the same time, the model should not over-exploit information about known active molecules from the historical data, so that it can generate novel active molecules.

Modeling the problem as an MDP (using Model-Free RL)



Agent: Generator model (RNN). Learns the policy for generating active molecules

Action at time t.

O

End token: Oxygen atom

$p(a_t|s_{t-1})$

CC(=O)OC1=CC=CC=C1C(=O)

State $_{t-1}$: Partial SMILE string

Intermediate states have zero reward

CC(=O)OC1=CC=CC=C1C(=O)O

State $_{t+1}$

Update the parameters Θ of the NN using **policy gradient** to maximize the rewards

If **State $_{t+1}$ = State $_T$ (Terminal state)**, then, **checked for validity**: SMILES string must follow specific chemical rules. **If valid, then:**

Predictive model (RF): assigns a reward based on the predicted probability of activity.

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

Threshold:0.75

REWARD

Environment

Modeling the problem as an MDP (using Model-Free RL)

- The generative model is treated as the policy network. It refers to the strategy used to determine the next action (adding a new character to the SMILES string prefix) without an explicit model of the environment's transition probabilities, so, it is a Model Free approach
- The generative model directly learns through experience, generating sequences and adjusting its policy based on the received rewards.
- The set of actions are limited to the SMILES alphabet.
- The Set of states are limited to all strings in the SMILES alphabet with lengths up to limit N, a hyperparameter.
- N is defined by the maximum length of SMILES strings from the training dataset.

Objective function to be maximized: Expected Reward

$$L(\theta) = \sum_{i=1}^{N} r(s_N)\gamma^i \log p(s_i|s_{i-1}, \theta) \tag{1}$$

- Where s_N is the generated SMILES string.
- $s_i, i = 1, \dots, N$ is the prefix of s_N of length $0 < i < N$.
- γ is the discount factor.
- $p(s_i|s_{i-1}, \theta)$ is the transition probability obtained from the generative model.
- $r(s_N)$ is the value of the reward function for the generated SMILES string based on the output of the predictive model of active class probability.

REINFORCE algorithm: Objective function maximization

- Train agents to make sequential decisions in an environment.
- Policy gradient method that belongs to the family of Monte Carlo algorithms.
- A neural network is employed to present a policy.
- Updates the parameters (θ) to maximize expected rewards.

REINFORCE update equation:

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t \tag{2}$$

The term log $p(s_i|s_{i-1}, \theta)$ in the objective function corresponds to the policy component π_θ in the REINFORCE update equation,$r(s_N)$ corresponds to v_t and α is the learning rate (size of the updates made to the policy parameters (θ) during training).

REINFORCE algorithm

Algorithm 1 REINFORCE

```
1: Initialize $\theta$ arbitrarily
2: repeat
3:     for all episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\}$ sampled from $\pi_\theta$ do
4:         for $t = 1$ to $T - 1$ do
5:             $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
6:         end for
7:     end for
8: until convergence
9: return $\theta$
```

Computing rewards: `get_reward` Function

The `get_reward` function evaluates the reward for a generated molecule based on its predicted activity score.

Function Overview:

- Takes in a SMILES string (a textual representation of a molecule) and a predictor model.
- Returns a reward value based on whether the molecule's activity prediction meets a certain threshold.

Explanation:

- `predictor.predict([smiles])` returns the predicted activity score for the molecule.
- If the prediction meets the threshold, it returns a high reward (10.0); otherwise, it returns `invalid_reward`(1.0).

Understanding the `simple_moving_average` Function

The function calculates the moving average of a series of values to smooth out fluctuations and highlight longer-term trends, helping track reward stability and trends over training iterations.

Function Overview:

- Takes a list of previous values, a new value, and the window size *n*.
- Returns the average of the last *n* values, including the new value.

Example:

- Suppose `prev_values = [2, 4, 6, 8, 10]` and `new_value = 12`.
- With `ma_window_size = 3`, we calculate:

$$\texttt{moving\_average} = \frac{8 + 10 + 12}{3} = 10$$

- The function returns 10, which is the simple moving average over the last 3 values.

De Novo Molecule Generation Pipeline

Pipeline Overview

- **Model Pretraining:** The generative model is pre-trained using a dataset from ChEMBL which consists of 2 million bioactive molecules.
- **Predictive model :** Trained on historical experimental data of activities for EGFR extracted from ChEMBL.
- **Reinforcement Learning Setup:** Define the reward function based on molecular properties (e.g., activity).
- **Policy Optimization:** Utilize reinforcement learning methods (e.g., Policy Gradient) to improve the generation policy, focusing on generating molecules with high activity.
- **Sampling and Evaluation:** Generate novel molecules, evaluate them using predefined criteria (e.g., validity, uniqueness, activity), and refine the model based on feedback.
- **Iterative Improvement:** Repeat the reinforcement learning steps to progressively enhance molecule quality and novelty.

Exploration and Exploitation trade-off improvement

Heuristics:

- **Fine-tuning by transfer learning on high-reward examples.**
- **Experience replay on high-reward molecules.**
- **Real-time reward shaping.**

Experience Replay in Reinforcement Learning

Experience Replay on High-Reward Molecules

- **Purpose:** Addresses sparse rewards while balancing exploration and exploitation.
- **Replay Buffer:** Stores high-reward molecules for later use.
- **Training Process:**
    - Randomly draw high-reward examples from the buffer.
    - Calculate reward and apply policy gradient updates.
- **Intermittent Use of High-Reward Molecules:**
    - The model occasionally selects and trains on high-reward molecules from the replay buffer, but not exclusively.
    - The generative model also explores and generates new molecules outside the buffer.
    - This approach allows balancing learning from successful past examples and exploring new possibilities, maintaining a mix of exploitation and exploration during training.

Fine-Tuning in Reinforcement Learning

Fine-Tuning by Transfer Learning on High-Reward Examples

- **Purpose:** Optimize the generative model (policy) by focusing on high-reward molecules to improve performance.
- **Training Process:**
  - Fine-tuning is done by minimizing cross-entropy loss, similar to the pretraining stage.
  - Uses generated molecules with high rewards as training samples, rather than historical experimental data.
- **Characteristics:**
  - **High Exploitation, Low Exploration:** Model focuses on successful scaffolds, producing molecules similar to high-reward examples.
- **Difference from Historical Data:**
  - Historical data may limit discovery to known chemical scaffolds.
  - Fine-tuning on generated high-reward scaffolds allows potential discovery of novel structures.

Real-Time Reward Shaping in policy network

- **Purpose:** Efficiently train a neural network when high-reward molecules are rare.
- **Concept:** Dynamically adjust the reward function over training based on molecule generation results.

Threshold-Based Reward Function

- **Threshold Reward Function:**

$$R(s) = \begin{cases} r_{\text{pos}} & \text{if } p(s) > p_0 \\ r_{\text{neg}} & \text{otherwise} \end{cases}$$

- Where:
  - s: Generated molecule
  - $p(s)$: Active class probability (from predictive model).
  - p_0: Probability threshold (starts small, dynamically increases).
  - r_{pos}: Reward for "good" examples ($p(s) > p_0$).
  - r_{neg}: Reward for "bad" examples ($p(s) \leq p_0$)

Dynamic Adjustment of Threshold p_0

- **Initialization:** Start with $p_0 = 0.05$.
- **Update Condition:** Increase p_0 by 0.05 when:
  .At least 15% of a batch (e.g., 3000 molecules) has $p(s) > p_0$
- **Result:** The model learns to exploit "good" examples even when few high-reward molecules are initially available.

Benefits of Real-Time Reward Shaping

- Improves **training efficiency** by rewarding molecules with non-zero predicted active class probabilities in the absence of good examples.
- Allows the model to **progressively learn** from increasingly selective thresholds.
- Enables the discovery of high-reward molecules over time.

Model Evaluation Metrics

Fraction of Active Chemical Structures

The extent of model learning is evaluated by measuring the fraction of generated trajectories resulting in active chemical structures:

$$\text{Active Fraction} = \frac{\text{Valid SMILES with Predicted EGFR Activity}}{\text{Valid and Unique SMILES}}$$

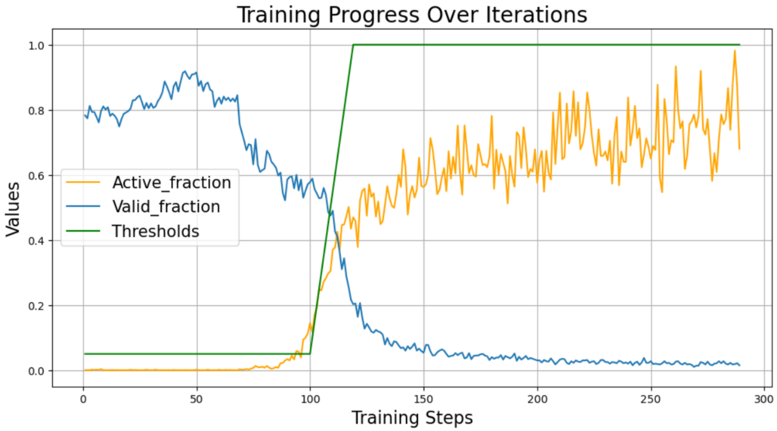
This metric records the fraction of molecules with predicted activity (threshold $= 0.75$), indicating effective learning.

Two Types of Threshold

- **Fixed Threshold (e.g., threshold = 0.75)**:
  - Used during evaluation to determine the *active fraction* of generated molecules.
  - Measures the fraction of molecules exceeding a fixed activity threshold, assessing generator performance.
- **Dynamic Threshold**:
  - Used in reinforcement learning to increase rewards for molecules with high predicted activity.
  - Encourages the generator to produce progressively higher activity molecules as training progresses.

Dynamic Threshold Setting

Purpose: Define the best setting for the Dynamic Threshold.
Initial setting: p_0 is increased by steps of 0.05 if at least 15% of a batch has $p(s) > p_0$. The maximum value that the threshold can reach is 1.
This run was without experience in replay and Fine-tuning.



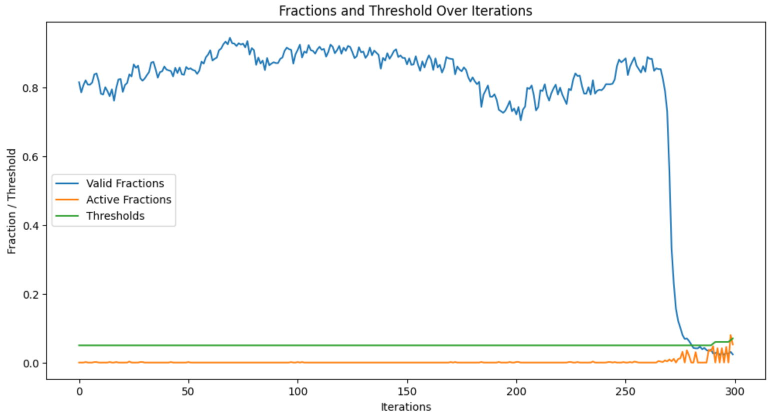
Results

- Setting the threshold too high (1.0 in this case) results in a very selective generation process, which limits the diversity in generated molecules and could lead to fewer rewards if only a limited number of molecules achieve the maximum score.

- If we reduce the threshold step and maximum value this may lead to a more balanced progression, as the model isn't forced into strict thresholds too quickly, allowing more room for generating viable and active molecules over time.

Results

Setting: p_0 is increased by steps of 0.005 and the maximum value is 0.8, **without experience replay**.

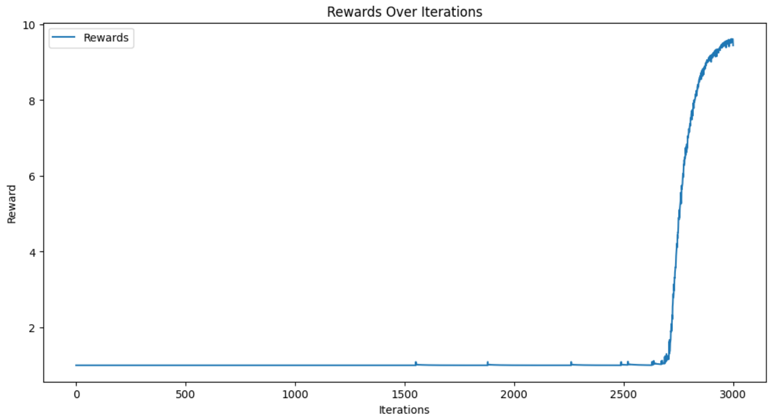


Results

- The smaller step size and lower max threshold (0.8) allows for a higher and constant valid fraction at the expense of producing near zero-activity molecules.
- In the original setting (steps of 0.05 and the maximum value of 1.0) a higher proportion (0.4) of valid and active fraction is obtained at iteration 110.

Results

- 300 Epochs x 10 policy updates

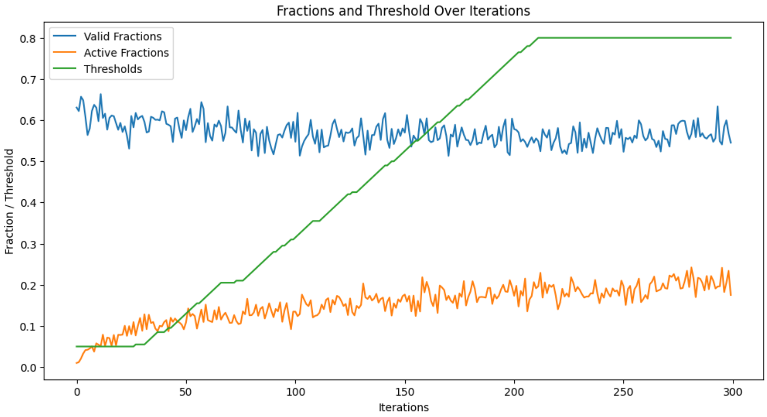


- The rewards initially remain very low over the first 2500 iterations. Then, after iteration 2500, there is a steep increase, eventually reaching a reward close to 10 by iteration 3000. Which coincide with the increase of the active fraction. This steep increase tells us that most of generated molecules meet or surpass the activity threshold.

- This steep increase might be due to cumulative learning effects as the generator network slowly aligns its molecule generation with the predictor's requirements.

Results

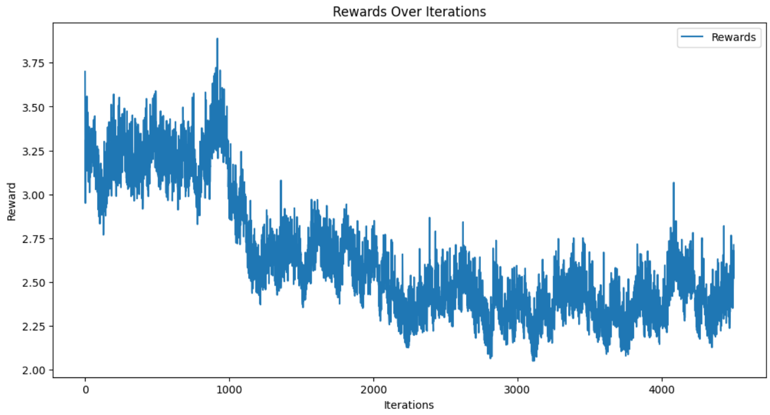
Setting: p_0 is increased by steps of 0.005 and the maximum value is 0.8, **with experience replay**.



- With experience replay we obtain the result expected when decided to change the dynamic threshold setting.

- Experience replay allows the model to "remember" useful molecule patterns from previous iterations, which likely leads to a more robust learning process. This is reflected in higher active fractions and more stable valid fractions.

Results


Rewards Over Iterations
Rewards
Reward
Iterations
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
0
1000
2000
3000
4000

Results

- With this setting, both the valid and active transactions increase. As the valid fraction increases, so do the non-active molecules; hence, we observe lower average rewards.

Conclusion

- Applying a dynamic threshold in the policy gradient optimization can effectively control the trade-off between exploring new, diverse chemical structures and generating molecules that meet a specific activity threshold.

- Dynamic threshold with smaller steps (0,005) and a capped maximum (like 0.8) reduces the initial reward sparsity, accelerating early learning and allowing the model to gain momentum in generating valid and active molecules.

- **The optimal policy** was achieved implementing dynamic threshold and experience replay. This setting let the model to produce a constant and moderately high proportion of valid and active fraction which could lead to a higher proportion of novel molecules with high biological activity.

Future Steps

Future directions include:

- Test additional settings, including fine-tuning via transfer learning on high-reward trajectories and combining dynamic thresholds with experience replay.
- Change the policy optimization algorithm:
    - **Soft Actor-Critic (SAC):** SAC is an off-policy actor-critic algorithm that promotes exploration by maximizing entropy. This algorithm may be well-suited for sparse reward settings, as its exploratory nature (due to the entropy term) encourages the agent to try different actions and explore more states.

Bibliography I

[1] Korshunova et al. (2022)

Korshunova, M., Huang, N., Capuzzi, S., et al. (2022). Generative and reinforcement learning approaches for the automated de novo design of bioactive compounds. *Communications Chemistry, 5*, 129. https://doi.org/10.1038/s42004-022-00733-0

[2] Segler et al. (2018)

Segler, M. H. S., Preuss, M., Waller, M. P. (2018). Planning chemical syntheses with deep neural networks and symbolic AI. *Science Advances, 4*(2), eaap7885. https://doi.org/10.1126/sciadv.aap7885

[3] Isayev Lab (2023)

Isayev Lab. (2023). Reinforcement Learning Experiments. GitHub repository. https://github.com/isayevlab/rl$_e$*xperiments*/*tree*/*main*

Bibliography II

[4] Celani (2024)

Celani, A. (2024). Reinforcement Learning course notes. University of Trieste.

[4] Haarnoja et al. (2018)

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv preprint arXiv:1801.01290*. https://doi.org/10.48550/arXiv.1801.01290