Supplementary materials:
Deep reinforcement learning for personalized treatment recommendation

1 Reinforcement learning: a brief review

Recently reinforcement learning (RL) has shown great potential in many challenging applications that require dynamic modeling and long term planning, such as game playing [1], real-time ads bidding [2]. It has also been introduced into recommender systems [3, 4, 5, 6, 7]. There are two main categories of RL commonly used: model-based RL and model-free RL [8]. In model-based RL, the model always refers to how the environment reacts to certain actions, or how the transition is made in the environment. Model-based RL, such as POMDP [3], needs to model the dynamic transition, and may not be suitable for complicated recommendation scenarios when the number of candidate drugs is large but with only few cell-lines, as the update of dynamic programming step is too time-consuming. For model-free RL, no dependency on the model of the transition during learning is needed. It can be further divided into two sub-categories: value-based [6] and policy-based [5, 7]. The value-based approaches compute Q-values of all available actions for a given state, and the selection is based on the evaluation of overall actions. Although value-based methods present many advantages such as seamless off-policy learning, they are known to be prone to instability with function approximation [9].

Despite the practical success of many value-based approaches such as deep Q-learning [1], policy convergence of these algorithms are not well-studied. These approaches may become inefficient with a large action space. For policy-based approaches, it does not estimate the transition probability and does not store the Q-values; instead, they aim to generate a policy, of which the input is a state, and the output is an action. Policy-based approaches, on the other hand, remain rather stable with respect to function approximations given a sufficiently small learning rate. However, the popular Policy Gradient method only uses one sample
each step, and after updating the policy, the sample is removed, thus it disables the reuse of samples. The way that the agent grasps the optimal policy and uses the same to act is also referred to as "on-policy" RL. The policy that is used for updating and that used for acting is the same. In this way, every time we simulate a cell-line’s trajectory, it can only be used once and then is discarded, which may result in high variances and gradient explosion. This is also mentioned in training a vanilla policy gradient algorithm REINFORCE [8] with neural networks, as a model with only 46 weight parameters requires more than 10000 epochs to converge, and we have tried with simulated data even with a sample size larger than 1M, the variance is still large; when the sample size $n = 1000$ in simulations, the model’s weights might become "NaN" values during training.

Since this form of gradient has a potentially high variance, a baseline function is typically introduced to reduce the variance whilst not changing the estimated gradient [8]. A natural candidate for this baseline is the state value function. This is the basic framework of Actor-Critic [8], which can be easily used to learn high dimensional or even continuous actions. Typically, we use an Actor Network (Policy Network) to account for policy improvement, where the Actor Network takes the cell-line features (as well as the drug features if any) as input and generates the scores for each action (i.e. each drug), and outputs an action according to the scores, and we update the policy parameters from the Actor Network. Furthermore, the Actor Network also outputs a state representation and feeds it to a Critic Network. We use the Critic Network to evaluate the current policy, where we fit the Critic Network to estimate the state values. We have tried to use a simple linear function to learn the interactions of cell-lines and drugs, which converges fast and performs well on test data*. But it fails to capture non-linear relationships. In order to achieve a stable and convergent algorithm, we leverage deep reinforcement learning (DRL) with adapted artificial neural networks as non-linear approximators to estimate the value function. However, even with the on-policy Actor-Critic framework, the algorithm still cannot get the most out of every sample, namely, it is sample inefficient. In order to improve sample efficiency, we adapt the Proximal Policy Optimization (PPO) algorithm, which enables us to alternate between sampling data from the policy and performing several epochs of optimization on the sampled data. This has been shown to be significantly better in more efficient use of the samples.

More importantly, to account for possible patient (or cell-line) preference/condition changes, we use a GRU layer to memorize the previous preferences (or conditions), and

---

*We tested it on simulated data, in which it achieved 0.830 NDCG as compared to PPORank’s 0.941 NDCG
update the state if the cell-line’s evaluation signal is positive. The state representation learned by the Actor Network is also the input to the Critic Network, which is used to evaluate the policy. Furthermore, without any approximation to the targeted metric as the loss function, policy-based approaches exactly use the gradient methods for policy optimization, without considering the discretization of the loss function.

2 Learning to rank with proximal policy optimization (PPO)

Actor-Critic combines the advantage of value-based methods and policy gradient to achieve accelerated and stable learning, but it still needs further modifications for the dynamic ranking problem. On the other hand, considering the sample complexity, as the on-policy algorithm requires collecting new samples whenever the policy changes, the old sample is not reusable. A good way to make use of old samples is to use importance sampling (IS), in which we can use samples from the old policy to calculate the policy gradient. But if we sample under the old policy to calculate the expected long-term rewards under the new policy, and if the ratio of $\frac{\pi(\theta)}{\pi_{\text{old}}}$ is high, the variance of the estimate may still explode. Thus we still need to re-sample trajectories frequently using the current policy.

We also notice that with the increasing training time, the scores for sensitive drugs will continue increasing, resulting in a final failure. One of the reasons is that the gradient depends on the policy parameterization, instead of the actual policy, as in our case with a softmax policy, the policy can often be reparameterized without changing action probabilities. The steepest ascent in the parameter space does not guarantee the steepest ascent in the policy space. In particular, when the sample size is small, it may not be able to recover from the previous bad policy and thus will collect data under the bad policy.

Hence in our policy optimization algorithm, we want an update step that uses rollouts collected from the most recent policy as efficiently as possible, and takes steps that respect the distance in the policy space as opposed to in the parameter space, which aims to control the changes in policy. In order to stabilize the learning process and improve the sample efficiency, we use the ideas from the trust region policy optimization (TRPO) [10], which bounds the distribution to use the KL divergence and proximal policy optimization (PPO) [11] that clips on the probability ratio. To implement simply, we mainly design the algorithm with PPO. By bounding the policy distribution change, we are able to optimize the expected advantage under the old policy, which aims to get an improved policy from optimization on
the sampled data from the old policy $\pi_{\theta_{old}}$. Here, we use the clip ratios as the constraint for the policy probability change, then the clipped "surrogate" objective is:

$$L_{CLIP}^t(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( r_t(\theta) \hat{A}_{\pi_{\theta_{old}}}^t, \text{clip} \left( r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{\pi_{\theta_{old}}}^t \right) \right],$$

(1)

where $\hat{\mathbb{E}}_t[\ldots]$ indicates the empirical average over a finite batch of samples at time step $t$, and $\hat{A}_{\pi_{\theta_{old}}}^t(s_t, a_t)$ is an estimate of the advantage function at time step $t$ from samples following $\pi_{\theta_{old}}$, and $r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio, $\epsilon$ is the clipping parameter by limiting $r_t(\theta)$ within the interval $[1 - \epsilon, 1 + \epsilon]$. As we can see, the clipping serves as a regularization by removing incentives for the policy to change dramatically, and the hyperparameter $\epsilon$ corresponds to how far away the new policy can go from the old one while still increasing the objective.

As our state is constructed from the cell-line features (and candidate drug features), so we choose to share the parameters between the policy and value networks, denoted as $\theta$. Similar to that in Actor-Critic, we use the supervised value loss function to evaluate the policy, and notice that the value function also explodes as the last layer of the Critic Network is a linear layer, which is not updated with the previous parameters, so we also add a clip penalty on it, then the clipped value loss is:

$$L_{VF}^t(\theta) = \min \left[ \hat{\mathbb{E}}_t \left( V_{\pi_{\theta}}^t(s_t) - V_{targ}^t \right)^2, \hat{\mathbb{E}}_t \left( \text{clip} \left( V_{\pi_{\theta}}, V_{\pi_{\theta_{old}}}^t - \epsilon, V_{\pi_{\theta_{old}}}^t + \epsilon \right) - V_{targ}^t \right)^2 \right].$$

(2)

Here $V_{targ}^t$ is the target value at time step $t$; as used in supervised learning, the easiest case is $V_{targ}^t = r(s_t, a_t) + \gamma V_{\pi_{\theta_{old}}}^t(s_{t+1})$. In our ranking problem, even starting from the same initial state \textsuperscript{†}; according to the randomness of policy, we may sample different trajectories, and with the increase of time step $t$, the variance among these different trajectories will become larger. Furthermore, as the maximum time step is $M = |D|$, we can directly apply the generalized advantage estimation (GAE) [12], which will cut the trajectory before the variance becomes too large. The advantage estimator at time step $t$ is

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

(3)

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

and $M$ is the maximum steps for each trajectory, which may be different across cell-lines. When applying this advantage function into 1, it is calculated based on the sampled batch

\textsuperscript{†}for every cell-line from which we sample episodes, the initial state is always $f([X, D, H_1])$
data from the old policy $\pi_{\theta_{old}}$.

PPO is prone to suffering from lack of exploration, especially with bad initialization, which may lead to the failure of training or being trapped in bad local optima. Thus we introduce a supervised signal, the entropy accounting for efficient exploration similar to that used in the original paper [11], for regularization. The entropy regularization is

$$S(\pi_{\theta}, s_t) = - \left(1 - \hat{I}_t\right) \log (1 - \pi_{\theta}(a_t|s_t)).$$

(4)

At time step $t$, $\hat{I}_t \in \{0, 1\}$ denotes whether the $t$th drug is selected. The idea is similar to Soft Actor-Critic (SAC) [13].

In this way, the surrogate loss at time step $t$ is

$$L_t(\theta) = \hat{E}_t \left[ L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S(\pi_{\theta}, s_t) \right],$$

(5)

where $c_1, c_2$ are coefficients. With $P$ parallel actors each collecting at most $M$-time step data, while the sampling procedure for each actor is the same as in Algorithm 1, we can optimize the surrogate loss on these $P \times M$-time step data with minibatch SGD for $K$ epochs; here $K$ is called the ppo epochs. By optimizing with the proximal policy, we call the resulting ranking algorithm PPORank.

The sampling of one episode is equivalent to ranking the drug list for each cell-line under the current policy. The sampling process for one episode (ranking list) can be found in Algorithm 1. The list-wise ranking process is constructed as follows: given a cell-line $c$, with feature vector $X$, the set of relevant drugs with feature set $D = \{d_j\}_{j=1}^M$, and the true labels $Y$. At each time step $t$, the agent receives state $s_t$, then according to the current policy $\pi_{\theta}$, choose an action $a_t$ of selecting drug $d_{m(a_t)}$, and place it at rank position $t$. Then the environment moves to the next step $t + 1$, and transits to the next state $s_{t+1}$; at the same time, the environment receives reward $r_t = r(s_t, a_t)$ as well as the evaluation signal to decide whether or not to update the hidden state. The process is continued until all the drugs are selected.

### 3 Top $k$ ranking

In drug recommendation, often only top one or few recommendations are needed (as opposed to predicting the exact values of the response to each of all the drugs). Hence we may be more interested in obtaining a top $k$ ranking, instead of ranking all the drugs [14]. When
using $NDCG@k$ as the evaluation metric for ranking, researchers have used different convex upper bounds for non-convex optimization with different $k$ [15, 16]; these upper bounds are functions of $k$. The optimization process requires pre-defined parameter $k$. We can adjust the tuning parameters from two aspects. (1) Cutting the length of each trajectory. Recall that in Algorithm ??, for each cell-line we run the ranking process to the end. Here in order to make more precise prediction on the top $k$ drugs, we want to generate short trajectories for different actions given the initial state. (2) Modifying the discount parameter so that the agent will take into account only the most recent actions. Consider the Advantage function in (3, which is controlled by $\gamma, \lambda$. There are two specific cases with $\lambda = 0$ and $\lambda = 1$ as $\text{GAE}(\gamma, 0) : \hat{A}^{\text{GAE}}_t := \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ and $\text{GAE}(\gamma, 1) : \hat{A}^{\text{GAE}}_t := \sum_{l=0}^{\infty} \gamma^l \delta_{t+l} \equiv \sum_{l=0}^{\infty} \gamma^l r_{t+l} - V(s_t)$. When $\gamma = 0$, the agent acts to maximize the immediate reward. So for different choices of $k$, we choose different discounting factors.

In addition to the tuning process, we also include another supervised loss. As we are ranking the top $k$ drugs, we can decompose this process into two parts: (1) selecting a list of $k$ drugs; (2) re-ranking the $k$ drugs. For selecting a list of $k$ drugs, we have the entropy loss as

$$-(1 - \hat{a}_{t,k}) \log (1 - \pi_\theta(a_{t|s_t})).$$

At time step $t$, $\hat{a}_{t,k} \in \{0, 1\}$ denotes whether the selected drug is among the top $k$ drugs based on the ground truth.
4 Multi-omic data types in GDSC

Table 1: Four types of the omic data in GDSC.

| Data type                      | #cell-lines | #features | missing rate |
|-------------------------------|-------------|-----------|--------------|
| gene expression(GEX)          | 962         | 17737     | 18%          |
| whole-exome sequencing (WES)  | 953         | 300       | 19%          |
| copy number variation(CNV)    | 985         | 425       | 19%          |
| DNA methylation (MET)         | 785         | 378       | 19%          |

1 Note: WES are binary features encoding if the given cell-line carries variants in recurrently mutated sites of one of the 300 candidate cancer genes (CGs) identified in 6,815 patient tumors; CNV are binary features encoding if the given cell-line carries one of the 425 recurrently aberrant copy number segments (RACSs) identified in 8,014 patient tumors; MET are binary features encoding if the given cell-line carries one of the 378 hyper-methylated informative CpG islands (iCpGs) located in the gene promoters identified in 6,166 patient tumors.
References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[2] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. Real-time bidding by reinforcement learning in display advertising. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, WSDM ’17, pages 661–670, Cambridge, United Kingdom, February 2017. Association for Computing Machinery.

[3] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. Journal of Machine Learning Research, 6(Sep):1265–1295, 2005.

[4] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems, pages 191–198, 2016.

[5] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. arXiv:1803.00710 [cs], May 2018.

[6] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW ’18, pages 167–176, Lyon, France, April 2018. ACM Press.

[7] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. Deep reinforcement learning for list-wise recommendations. arXiv:1801.00209 [cs, stat], June 2019.

[8] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT press, 2018.

[9] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems, pages 1057–1063, 2000.
[10] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[14] Chayaporn Suphavilai, Denis Bertrand, and Niranjan Nagarajan. Predicting cancer drug response using a recommender system. *Bioinformatics*, 34(22):3907–3914, 2018.

[15] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484, 2005.

[16] Markus Weimer, Alexandros Karatzoglou, Quoc V Le, and Alex J Smola. Cofi rank-maximum margin matrix factorization for collaborative ranking. In *Advances in neural information processing systems*, pages 1593–1600, 2008.