COMP6247(2019/20): Reinforcement and Online Learning Final Assignment (Project)

DAJI LI
Student ID: 31326234
E-mail: dl5n19@soton.ac.uk

June 2020

1 Part I: Radial Basis Functions.

1.1 Familiar with the RBF model.

The method of Radial Basis Functions (RBF) consists of nonlinear basis functions (usually local functions) and learnable weights. The basis functions are using some sensible set of rules and the weights are the only parts estimated, thus making the problem linear in parameters. An RBF model of J basis functions is defined by

$$f(\boldsymbol{x}) = \sum_{j=1}^{J} w_j \phi\left(\left\|\boldsymbol{x}_\mathbf{j} - \mathbf{m_j}\right\| / \sigma_\mathbf{j}\right) \tag{1}$$

where J is the number of basis function and $\mathbf{m_j}$ is the mean of input $\boldsymbol{x}_\mathbf{j}$. The non-linear function $\phi(.)$ is usually a Gaussian function and the σ_j offers the flexibility to control the region of influence of the basis functions. Given a training set, $\{\boldsymbol{x}_n, y_n\}_{n=1}^{N}$, the RBF model will construct an $N \times J$ design matrix U, which elements u_{ij} are given by

$$u_{ij} = \phi\left(\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\| / \sigma_j\right), i = 1, \ldots, N, j = 1, \ldots, J \tag{2}$$

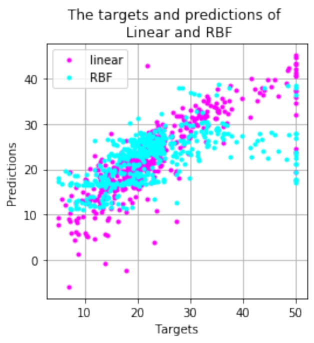


Figure 1: Regression with Linear and RBF Models

In the following part, I implement RBF sampling for a small dataset `housing.data` and compare the regression between Linear model and RBF model with $J = 20$. Both methods use pseudo-inverse to calculate weights $\mathbf{w}$, which linear model use the original data X while RBF model use the design matrix U. The results of true targets and predictions shows in Fig. 1

We also can use Stochastic Gradient Descent (SGD) to calculate the $\mathbf{w}$ of RBF models given by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha\left[y - f(\boldsymbol{x}, \boldsymbol{w}_t)\right] \nabla f(\boldsymbol{x}, \boldsymbol{w}_t) \tag{3}$$

where the α is the step size which need to be small to avoid affecting the entire model and the $f(\boldsymbol{x}, \boldsymbol{w}_t)$ is the prediction of RBF model. The Fig. 2 (left) shows the convergence of the error and the Fig. 2 (right) shows the performance of regression which is similar with using pseudo-inverse directly.

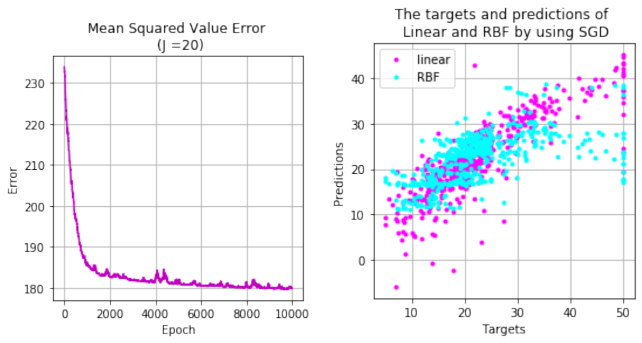


Figure 2: Train the RBF models by SGD

1.2 The mountain car learning control problem.

1.2.1 Obtain the value function learned by the tabular method and build an RBF approximation to it with different numbers of basis functions.

In this part, we need to consider the mountain car learning problem. We firstly use tabular method to train the action value function $q_\pi(s, a)$. We split the state space equally by 40 shown in Fig. 3. And implement 10 times episodes by Q learning rule updated by

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\Big[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) \\ -Q\left(S_t, A_t\right)$$

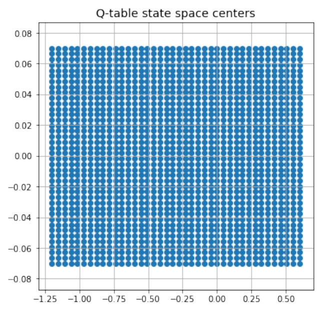


Figure 3: Q table state space center.

I choose the max value action to build the optimal policy $q_*(s,a)$ and plot the mountain 'Cost to Go' function shown in Fig. 4 (left). I also record the episode length of each time and plot the episode length over time shown in Fig. 4, we can see the episode length show a downward trend.

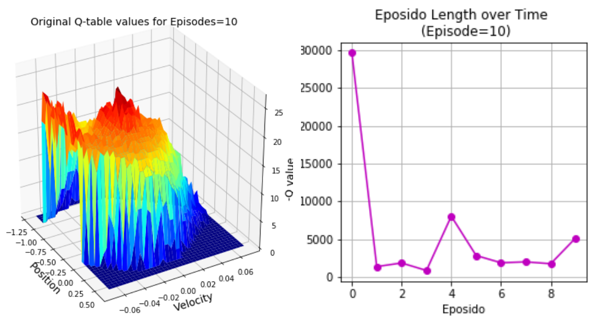


Figure 4: Left: Optimal policy $q_*(s,a)$; Right: Episode Length over Time.

In fact, if we implement more episodes, we can get a more accurate policy to mountain 'Cost to Go' function shown in Fig. 5.

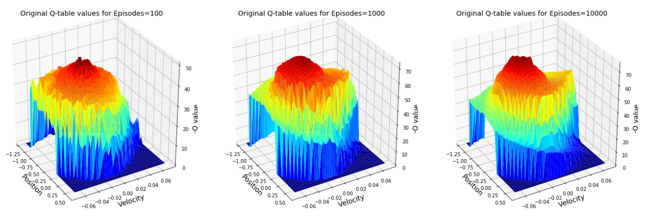
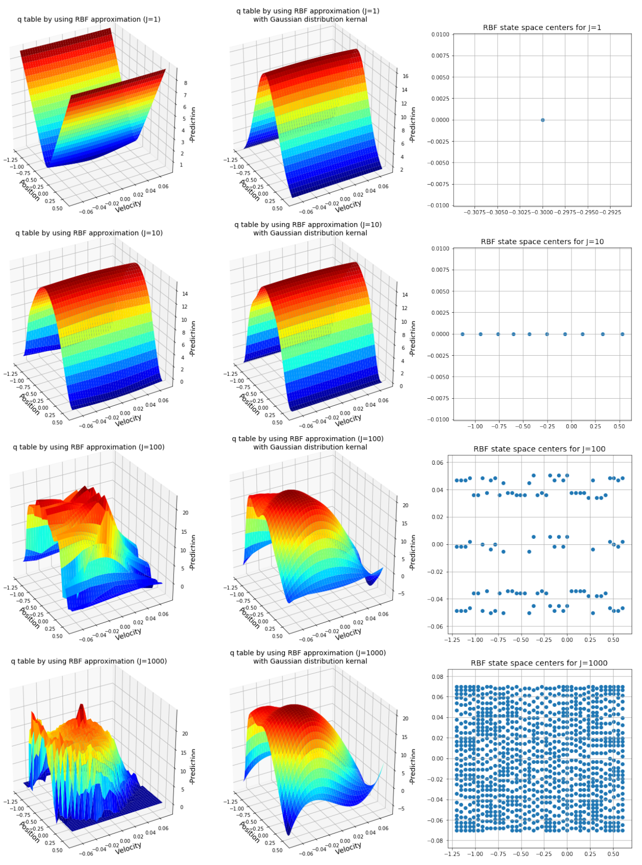


Figure 5: Optimal policy $q_*(s,a)$ of episodes = 100, 1,000, and 10,000.

Next steps, we need to build an RBF approximation to it with different numbers of basis functions (episodes = 10) and I use two different kernels to build design matrix U. One is the kernel function used in last Section 1.1, another is Gaussian function given by

$$u_{ij} = \phi\left(\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_j^2}\right)\right), i = 1, \ldots, N, \quad j = 1, \ldots, J \tag{4}$$

Like before, I plot the optimal policy $q_*(s,a)$ transformed by RBF approximation and show the state space center generated by k mean (see in Fig. 6).



Euclidean Gaussian State Space Center

Figure 6: RBF approximation with different numbers of basis functions ($J = 1, 10, 100$ and $1,000$).

We can find the when $J = 100$, the optimal policy shows a similar shape with its generated by tabular method and the optimal policy $q_*(s,a)$ transformed by RBF approximation with Gaussian function looks like more smoothed. While the number of basis function increases, the shape of mountain 'Cost to Go' function is more similar with q table shown in Fig. 4.

If we change $J = 1600$ which equal the number of state space center in tabular method, the optimal policy will show a same shape of original q table when we use Euclidean kernel. However, it still shows smoothed if we use Gaussian function kernel.

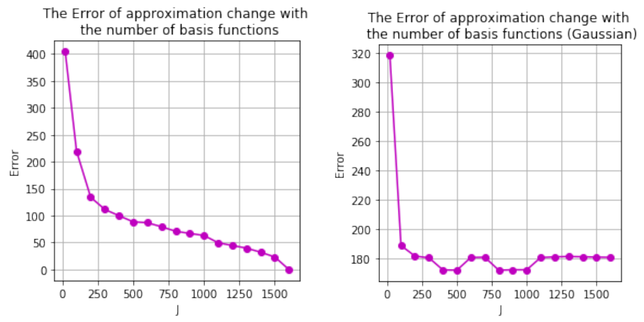


Figure 7: The error of RBF approximation change with the number of basis functions. Left: Euclidean; Right: Gaussian.

I calculate the L2-norm error between the original value function and the value function generated by RBF approximation. The error changes with the number of basis functions shown in Fig. 7. We can find if we use Euclidean function kernel, the error will go down to zero when $J = 1600$. However, if we use Gaussian function kernel, the error will be stable about 180 after the number of basis function more than 200.

If we use the policy derived from the approximation to driving the car to the top of the hill, We can find the number of basis function more than 10, the policy is capable of driving the car to target with shorter episode length.

1.2.2 Learn the weights of the RBF approximation on-line using a Q-learning rule.

In a reinforcement learning setting RBF models can be exploited to derive a value function approximation. Tabular methods are impractical when dealing with problems that have big state spaces (e.g go has 10^{170} possible states) or that have continuously valued features (tabular methods need pre-processing such tile coding)[1]. So in this part, we try to use RBF approximation on-line to solve the mountain car learning control problem by following steps:

- Obtain the value function used by tabular discretization;
- Construct feature vector state representation using RBF;
- Use tabular discretization value function as an unbiased estimate (U_t);
- Compute the weight vector using the pseudo-inverse and get an optimal policy $q_*(s,a)$ by maximising the value function according to actions.

The problem now resembles a supervised learning setting and, thus, we can solve it by updating the weight vector exploiting gradient descent. The weights update using SGD is calculated in a similar manner to linear regression given by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha \left[v_\pi (S_t) - \widehat{v}(S_t, \boldsymbol{w}_t) \right] \nabla \widehat{v}(S_t, \boldsymbol{w}_t) \qquad (5)$$

where $v_\pi(S_t)$ is the the value function used by tabular discretization and $\widehat{v}(S_t, \boldsymbol{w}_t)$ is the prediction given by the design matrix (U_t).

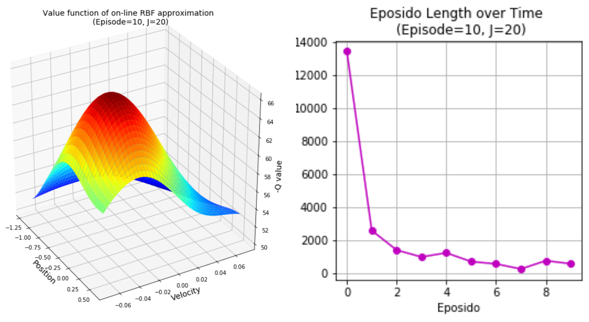


Figure 8: Left: Optimal policy $q_*(s,a)$; Right: Episode Length over Time.

I implement RBF approximation with Gaussian function kernel ($J = 20$) for 10 episodes and generate the weight $\mathbf{w}$ randomly (see Fig. 8). We can see the optimal policy $q_*(s,a)$ is more smoothed than its generated by tabular method. The episode length goes down strongly in first two episodes and keeps stable about 1000 times.

If we just charge the number of basis function J (see Fig. 9), we can find the optimal policy $q_*(s,a)$ is similar. But if we use too small number of basis function, it will lead an unsuccessful training since the design matrix does not have enough solutions to interpret the policy.

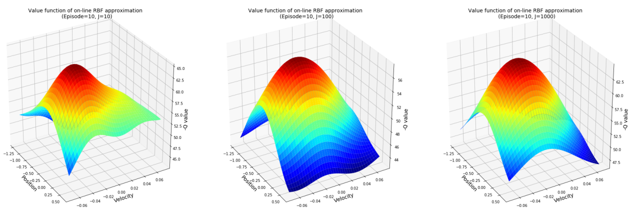


Figure 9: Some results of different number of basis function with same number of episodes ($Episodes = 10; J = 10, 100, 1,000$).

Moreover, I fix the number of basis functions and just charge the number of episode (see in Fig. 10). We can find a more precise optimal policy model with smoothed surface.

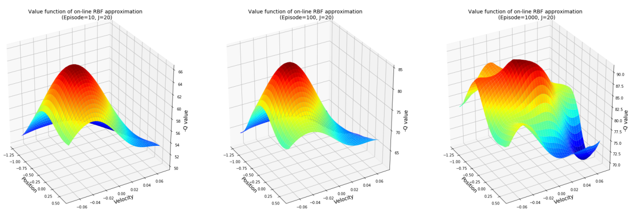


Figure 10: Some results of different number of episode with same number of basis function ($Episodes = 10, 100, 1,000$; $J = 20$).

In my observation, the function approximation method shows a more stable performance in testing as it will complete some information of states which never appear and fix the big state space with continuously valued features problem.

1.3 Fixed the size of the RBF network arbitrarily

In order to overcome NP-complete problems in learning with fixed size network, Platt designed Resource-allocating network (RAN) [2] to fix input-output pairs observation $(\mathbf{x}, \mathbf{y})$ with a smaller size network. The RAN is a growing network with one hidden layer, which the output response to the input is a linear combination of the hidden unit response given by

$$f(\mathbf{x}) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x}) \qquad (6)$$

where the $\phi_k(\mathbf{x})$ are the hidden unit response to input $\mathbf{x}$ and the coefficients α_k are the weights of the hidden to output layers and α_0 is the bias term. The RAN hidden unit response by a growing Gaussian radial basis function (GaRBF) network [3] given by

$$\phi_k(\mathbf{x}) = \exp\left\{-\frac{1}{\sigma_k^2}\left\|\mathbf{x}-\mathbf{u}_k\right\|^2\right\} \tag{7}$$

where $\mathbf{u}_k$ is the space center of GaRBF, σ_k is the standard deviation to control the width of basis function and will be updated while add a new hidden unit, given by

$$\sigma_{k+1} = \kappa\left\|\mathbf{x}-\mathbf{u}_{\text{nearest}}\ \right\| \tag{8}$$

where κ is an overlap factor that determines the overlap of the responses of the hidden units in the input space.

The network starts without hidden unit and the bias define by $\mathbf{x_0}$ and allocate the new unit according to the distance criterion (see Equation 9) and the prediction error criterion (see Equation 10) given by

$$\mathbf{u}_{\text{nearest}}\ = \min_n\left\|\mathbf{u}_n-\mathbf{x}\right\| > \epsilon_n \tag{9}$$

$$e_n = y_n - f(\mathbf{x}) > e_{min} \tag{10}$$

where $\mathbf{y}_n$ is the prediction of model, $\mathbf{u}_n$ is the space center of RBF, ϵ_n is the distance limitation controlled the resoluton of model and $_{min}$ is the prediction limitation controlled the resolution of limitation, both are thresholds. The distance ϵ_n will be controlled dynamically, states with $\epsilon(t) = \epsilon_{\max}$ and update given by

$$\epsilon_n = \max\left\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\right\} \tag{11}$$

where the $0 < \gamma < 1$ is a decay constant.

When adding a new hidden unit to the network, the parameters or weights associated with this unit are assigned as follows:

$$\alpha_{K+1} = e_n \tag{12}$$

$$\mathbf{u}_{K+1} = \mathbf{x}_n \tag{13}$$

$$\sigma_{K+1} = \kappa\left\|\mathbf{x}_n-\mathbf{u}_{\text{nearest}}\right\| \tag{14}$$

where κ is an overlap factor that determines the overlap of the responses of the hidden units in the input space.

If the observation does not satisfy criteria, the LMS algorithm can use this model to adapt the network parameters $\mathbf{w}$, updated by

$$\mathbf{w}^{(n)} = \mathbf{w}^{(n-1)} + \eta e_n \mathbf{a}_n \tag{15}$$

where the η is a small step size, e_n is error between prediction and target and $\mathbf{a}_n = \nabla_w f(\mathbf{x})$ is the gradient.

The RAN algorithm can be used in `MountainCar-v0` problem with function approximation in section 1.2.2 mentioned before. The RAN algorithm can be used as a function to suit the size of weights according to the distance criterion and the prediction error criterion. The RAN algorithm pseudo-code shows in Algorithm 1.

Algorithm 1 Resource-allocating network (RAN)

```
Input: Input input-output pairs observation ($\mathbf{x}$,$y$) which
    obtains by Q learning rule;
Output: The weights $\mathbf{w}$ of RBF function approximation and the suitable size of function approximation
    model.
    the network output $f(\mathbf{x})$ to the input $\mathbf{x}$ is given by
    $f(\mathbf{x}) = \alpha_0 + \sum_{k=1}^{K}\alpha_k\phi_k(\mathbf{x})$
    $\phi_k(\mathbf{x}) = \exp\left\{-\frac{1}{\sigma_k^2}\left\|\mathbf{x}-\mathbf{u}_k\right\|^2\right\}$
    initial the criteria $\epsilon_n = \epsilon_{\max}, \quad \alpha_0 = y_0$
    loop over presentations of observation $(\mathbf{x}_n, y_n)$
        $\epsilon_n = \max\left\{\epsilon_{\max}\gamma^n, \epsilon_{\min}\right\}$
        compute error $e_n = y_n - f(\mathbf{x})$
        if $e_n > e_{\min}$ and $\min_n\left\|\mathbf{u}_n-\mathbf{x}\right\| > \epsilon_n$
            allocate a new hidden unit
            $\alpha_{K+1} = e_n$
            $\mathbf{u}_{K+1} = \mathbf{x}_n$
            $\sigma_{K+1} = \kappa\left\|\mathbf{x}_n-\mathbf{u}_{\text{nearest}}\right\|$
        else
            $\mathbf{w}^{(n)} = \mathbf{w}^{(n-1)} + \eta e_n \mathbf{a}_n$
```

It is strange for us to use it on-line with RBF function approximation using Q-learning or SARSA rules. In Q-learning or SARSA rules, we can get the next observation by the value function or get next observation randomly by $\epsilon - greedy$. The RAN algorithm response to fix the size of weights $\mathbf{w}$ (a.k.a.. the value of J). As a result, we can run the RBF function approximation with small number of basis function and increase the size of model automatically which can reduce the time complexity and get better generalisation theoretically. I try to implement, but it shows an unsatisfactory convergence at the beginning and I will try to fix it in continues. And refer to Kadirkamanathan and Niranjan [3], the RAN algorithm still has some limitations.

- The bias term α_0 might be not in the solution, this might be not considered in other ANN algorithm;
- The distance threshold ϵ_n could reduce gradually until reach the ϵ_{min}, this will lead the limitation of a new unit lower. When the number of observation increases, some long distance state must appear. It will lead to add some unnecessary hidden units and influence the generalisation.

In the paper of Kadirkamanathan and Niranjan [3], they use the extended Kalman filter (EKF) to improve the performance of RAN. Using EKF to replace LMS algorithm can increase the speed.

2 Summarising a recent scientific study (drug design)

In this paper, the authors try to use Reinforcement Learning for Structural Evolution (ReLeaSE) as a cimputation strategy for de novo design of molecules. [4] In new drug discovery, one important part is hypothesis for new from available and in last 15 years [5] and people try to use computer science technologies to assign this process. Traditional method is using recurrent neural networks (RNNs) [6] but the properties of produced molecules could not be controlled well and used to predict the compounds in large library, not actually design. As reinforcement learning (RL) used for the data to involve and analyse the possible action and possible outputs. So, the authors use RL algorithm to design new compounds or molecules base on the desired physical, chemical, and/or bio-activity properties de novo [4]. It would be seen as an analogy with chemical space exploration. And one more important part is the sample representation which uses molecular-input line-entry system (SMILES) strings in the ReLeaSE approach.

The ReLeaSE approach have two models named generative models and predictive models. In the first phase, both generative models and predictive models need to be trained separately with a supervised learning algorithm.

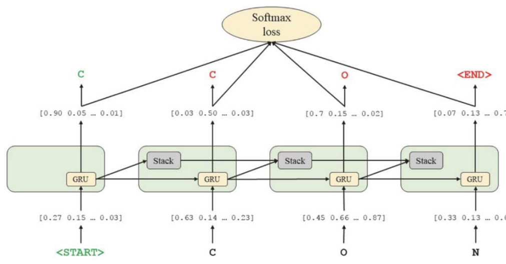


Figure 11: Training step of the generative Stack-RNN.

For the generative model, it is RNN model (GRU) with a stack which allow to learn meaningful long-range inter-dependencies. The training process shows in Fig. 11.

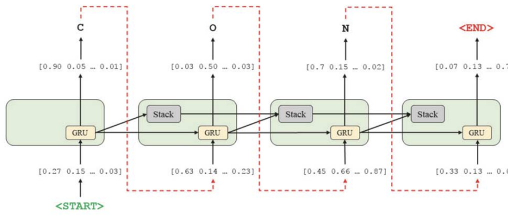


Figure 12: Generator step of the generative Stack-RNN.

The Fig. 12 is the process how the generative model to generate SMILES strings. The RAN-stack model will generate the state step by step t until reaching the terminal state T. At each step, $0 < t < T$, generative model will take the last state s_{t-1} as an input and estimates the probability distribution $p(a_t|s_{t-1})$ for the next action. This process likes the RNN model using in text generation.

For the predictive model, it is a neural network to predict property, which the SMILES string as input to the model and the target is the property.

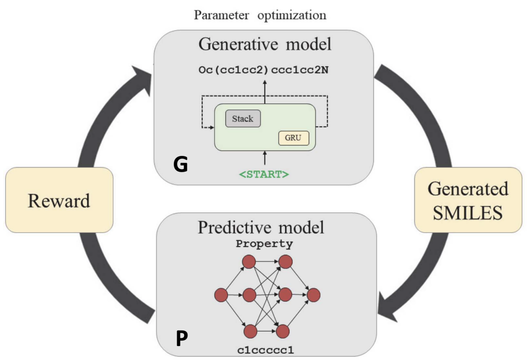


Figure 13: General pipeline of RL system for novel compound generation.

In the second phase, both models are trained together with the RL algorithm (see Fig. 13). The generative model as the agent to generate the possible SMILES string to the predictive model. And the predictive model as the critic to give a predicted property and according the desired property to give a reward to generative model for every molecule. The main components show below.

- **States:** SMILES strings with lengths from zero to some value T, which represent the chemical structures;
- **Rewards:** a function of the numerical property came from predictive model given by:

$$r(s_T) = f(P(s_T)) \tag{16}$$

- **State transition:** generate a length T states S_T step by step and each time step t accords to last probability distribution to choose the next action;
- **Policy models:** REINFORCE policy gradient algorithm with desired property which will be formulated as a task of finding a vector of parameters Θ to maximizes the expected reward given by

$$\begin{aligned} J(\Theta) &= \mathbb{E}[r(s_T)|s_0, \Theta] \quad (17)\\ &= \sum_{s_T \in S^*} p_\Theta(s_T) r(s_T) \rightarrow \max \quad (18) \end{aligned}$$

where this sum iterates over the set S^* of terminal states. The unbiased estimation $J(\Theta)$ is a cost function and its gradient is ∂_Θ given by

$$\begin{aligned} \partial_\Theta f(\Theta) &= f(\Theta)\frac{\partial_\Theta f(\Theta)}{\partial\Theta} \quad (19)\\ &= f(\Theta)\partial_\Theta \log f(\Theta) \quad (20) \end{aligned}$$

which can be written down as

$$\partial_\Theta J(\Theta) = \mathbb{E}_{a_1 \sim p_\Theta(a_1|s_0)} \mathbb{E}_{a_2 \sim p_\Theta(a_2|s_1)} \cdots \mathbb{E}_{a_T \sim p_\Theta(a_T|s_{T-1})} \left[\sum_{t=1}^{T} \partial_\Theta \log p_\Theta \left(a_t|s_{t-1}\right) \right] r\left(s_T\right) \tag{21}$$

which gives an algorithm $\partial_\Theta J(\Theta)$ estimation. We also need to calculate the expectation by Monte Carlo algorithm.

Refer to the Chapter 13 of [1], the right hand of Equation 21 is a sum over states weighted by how often the states occur under the target policy with all actions. If we define the target policy π which is a soft-max distribution and differentiated, then states will be encountered in these proportions.

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi\left(S_t,a\right) \nabla \pi\left(a|S_t,\boldsymbol{\theta}\right) \right] \end{aligned} \tag{22}$$

Replace a sum over the random variable's possible values by an expectation under π, and then sampling the expectation. Thus

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \mathbb{E}_\pi \left[\sum_a \pi\left(a|S_t,\boldsymbol{\theta}\right) q_\pi\left(S_t,a\right) \frac{\nabla \pi\left(a|S_t,\boldsymbol{\theta}\right)}{\pi\left(a|S_t,\boldsymbol{\theta}\right)} \right] \\ &= \mathbb{E}_\pi \left[q_\pi\left(S_t,A_t\right) \frac{\nabla \pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)} \right] \\ &= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi\left(A_t|S_t,\boldsymbol{\theta}\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}\right)} \right], \end{aligned} \tag{23}$$

where G_t is the return as usual (average). And we can use SGD to update the REINFORCE given by

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi\left(A_t|S_t,\boldsymbol{\theta}_t\right)}{\pi\left(A_t|S_t,\boldsymbol{\theta}_t\right)} \tag{24}$$

In proof-of-concept studies, the authors used the ReLeaSE method to design chemical libraries that favored structural complexity or compounds with a maximum, minimum, or specific range of physical properties (such as melting point or hydrophobicity), or favored Janus protein kinase 2 compounds that inhibit activity. The method proposed herein can be used to generate a targeted chemical library of novel compounds optimized for a single desired characteristic or multiple characteristics. The ReLeaSE approach benefits from the latest developments in the ML community's application of natural language processing and machine translation. These new algorithms allow learning the mapping from input sequence to output sequence.

Conclusion This paper is using the RL and DL algorithm for for de novo design of molecules with desired properties, which want to learn the latent policy in chemical space. As this area used to develop difficulty, obviously the ReLeaSE method can provide efficiency in deep and the paper also shows an achievement in single task regime. I think it can use to the drug for COVID-19.

And as the feature of RL which focus on policy learning, it is no doubt that it can implement in fiance market which can choose a wise action in stock market.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[2] J. Platt, "A resource-allocating network for function interpolation," *Neural computation*, vol. 3, no. 2, pp. 213–225, 1991.

[3] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural computation*, vol. 5, no. 6, pp. 954–975, 1993.

[4] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Science advances*, vol. 4, no. 7, p. eaap7885, 2018.

[5] G. Schneider and U. Fechner, "Computer-based de novo design of drug-like molecules," *Nature Reviews Drug Discovery*, vol. 4, no. 8, pp. 649–663, 2005.

[6] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, "Generating focused molecule libraries for drug discovery with recurrent neural networks," *ACS central science*, vol. 4, no. 1, pp. 120–131, 2018.