ML Meeting

Idea Proposals

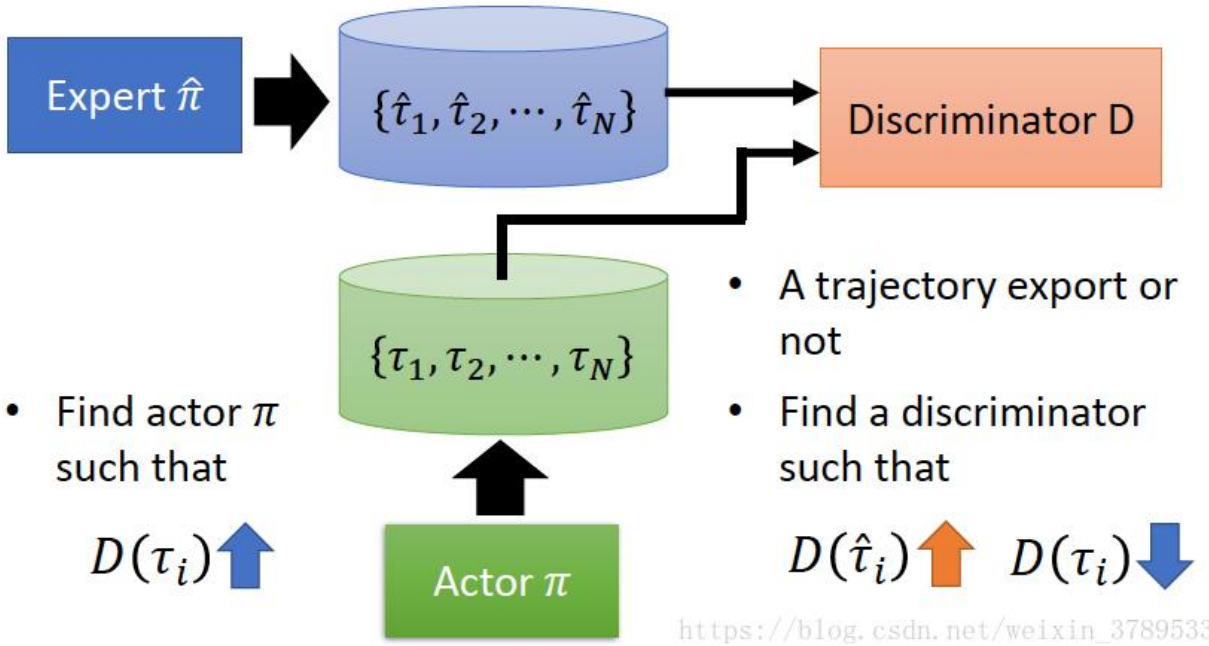
Idea Proposals

- GA-NTK + GAIL
- NTK + policy-constrained offline RL
- NTK + uncertainty-based offline RL
- Transformer RL + NTK

Generative Adversarial Imitation Learning(GAIL)

Given expert trajectory $\hat{\tau}$ generated by expert policy $\hat{\pi}$ as ground truth and the generated trajectory τ by generator(agent) π.

Discriminator D need to distinguish which trajectory is generated by the generator(fake).



Generative Adversarial Imitation Learning(GAIL)

Algorithm 1 Generative adversarial imitation learning

Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
for $i = 0, 1, 2, \ldots$ **do**
    Sample trajectories $\tau_i \sim \pi_{\theta_i}$
    Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

    Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}\left[\log(D_{w_{i+1}}(s,a)) \,|\, s_0 = \bar{s}, a_0 = \bar{a}\right] \end{aligned} \tag{18}$$

end for

Issues of GAIL

Most of them are te same as GANs

- Mode collapse, unstable training, low sample efficiency
- Recent search What Matters for Adversarial Imitation Learning? argues that
  - GAILs only synthetic demonstrations may lead to algorithms which perform poorly in the more realistic scenarios with human demonstrations, which means that GAILs doesn't generalize well on the reality.
  - But performs well on low-dimension tasks.
- Challenged by Offline RL, which usually can perform well on non-expert dataset

Idea: GA-NTK + GAIL

Advantages

- Avoid mode collapse, unstable training, low sample efficiency

Disadvantages

- Interpolation on small training dataset, same as GAIL

Offline RL

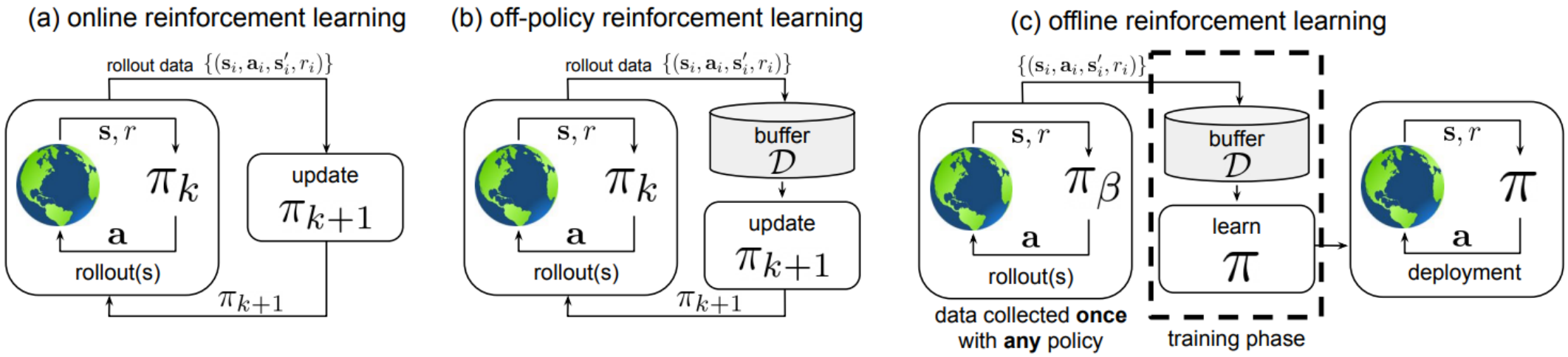


Figure 1: Pictorial illustration of classic online reinforcement learning (a), classic off-policy reinforcement learning (b), and offline reinforcement learning (c). In online reinforcement learning (a), the policy π_k is updated with streaming data collected by π_k itself. In the classic off-policy setting (b), the agent's experience is appended to a data buffer (also called a replay buffer) $\mathcal{D}$, and each new policy π_k collects additional data, such that $\mathcal{D}$ is composed of samples from $\pi_0, \pi_1, \ldots, \pi_k$, and all of this data is used to train an updated new policy π_{k+1}. In contrast, offline reinforcement learning employs a dataset $\mathcal{D}$ collected by some (potentially unknown) behavior policy π_β. The dataset is collected once, and is not altered during training, which makes it feasible to use large previous collected datasets. The training process does not interact with the MDP at all, and the policy is only deployed after being fully trained.

Extrapolation Error

- **Final Buffer**: train a DDPG agent for 1 million time steps, adding $\mathcal{N}(0, 0.5)$ Gaussian noise to actions for high exploration.
- **Concurrent**: We concurrently train the off-policy and behavioral DDPG agents, for 1 million time steps. A standard $\mathcal{N}(0, 0.1)$ Gaussian noise is added to action
- **Imitation**: A trained DDPG agent acts as an expert, and is used to collect a dataset of 1 million transitions.

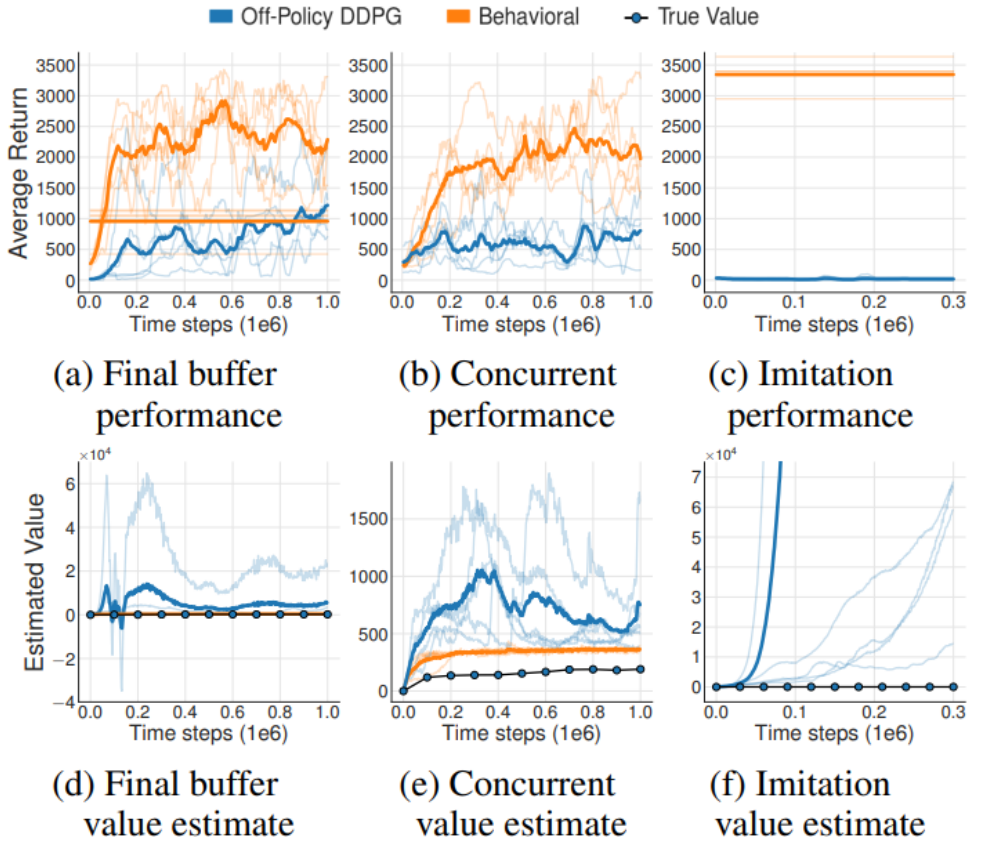


Figure 1. We examine the performance (top row) and corresponding value estimates (bottom row) of DDPG in three batch tasks on Hopper-v1. Each individual trial is plotted with a thin line, with the mean in bold (evaluated without exploration noise). Straight lines represent the average return of episodes contained in the batch (with exploration noise). An estimate of the true value of the off-policy agent, evaluated by Monte Carlo returns, is marked by a dotted line. In all three experiments, we observe a large gap in the performance between the behavioral and off-policy agent, even when learning from the same dataset (*concurrent*). Furthermore, the value estimates are unstable or divergent across all tasks.

Extrapolation Error

- In each task, the off-policy agent performances significantly worse than the behavioral agent, even in the concurrent experiment.
- Because the agent's policy isn't the same as the behavioral policy. The agent may overestimate unseen $Q^{\pi}(s, a)$.

Solutions for Extrapolation Error

- Policy Constraints: Align agent's policy $\pi(a|s)$ close to behavioral policy $\pi_\beta(a|s)$ (the policy that generate the training dataset)
    - Challenges: Hard to measure the distance between the distribution of $\pi(a|s)$ and $\pi_\beta(a|s)$. Currently, the SOTA algorithm use f-divergence(ABM algorithm) and MMD(BEAR algorithm) to measure the distance.
    - However, MMD just provides sufficient enough for constraining supports when finite samples are used.
- Uncertainty Estimation: Penalize the unseen action on $Q^\pi(s,a)$
    - Challenges: Hard to measure the uncertainty. The SOTA algorithm EDAC trains 10 ~ 500 Q-networks to estimate the uncertainty.
- NTK-GP may help.

Offline RL

$$\hat{Q}^{\pi}_{k+1} \leftarrow \arg\min_{Q} \mathbb{E}_{(\mathbf{s},\mathbf{a},\mathbf{s}')\sim\mathcal{D}} \left[\left(Q(\mathbf{s},\mathbf{a}) - \left(r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{a}'\sim\pi_k(\mathbf{a}'|\mathbf{s}')}[\hat{Q}^{\pi}_{k}(\mathbf{s}',\mathbf{a}')] \right) \right)^2 \right]$$

$$\pi_{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}[\hat{Q}^{\pi}_{k+1}(\mathbf{s},\mathbf{a})] \right] \text{ s.t. } D(\pi,\pi_\beta) \leq \epsilon.$$

Policy Constraints Offline RL

Integral probability metrics (IPMs). Another choice of the policy constraint D is an integral probability metric, which is a divergence measure with the following functional form dependent on a function class Φ:

$$D_\Phi(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \sup_{\phi \in \Phi, \phi: S \times A \to \mathbb{R}} \left| \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s})}[\phi(\mathbf{s}, \mathbf{a})] - \mathbb{E}_{\mathbf{a}' \sim \pi_\beta(\cdot|\mathbf{s})}[\phi(\mathbf{s}, \mathbf{a}')] \right|. \quad (21)$$

Distance measured in MMD

$$\text{MMD}^2(\pi(\cdot|\mathbf{s}), \pi_\beta(\cdot|\mathbf{s})) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi(\cdot|\mathbf{s})}[k(\mathbf{a}, \mathbf{a}')] - 2\mathbb{E}_{\mathbf{a} \sim \pi(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi_\beta(\cdot|\mathbf{s})}[k(\mathbf{a}, \mathbf{a}')] + \mathbb{E}_{\mathbf{a} \sim \pi_\beta(\cdot|\mathbf{s}), \mathbf{a}' \sim \pi_\beta(\cdot|\mathbf{s})}[k(\mathbf{a}, \mathbf{a}')],$$

where $k(\cdot, \cdot)$ represents any radial basis kernel, such as the Gaussian or Laplacian kernel. As another

IDEA: NTK instead of MMD?

Uncertainty-Based Offline RL

$$\pi_{k+1} \leftarrow \arg\max_{\pi} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}} \underbrace{\left[\mathbb{E}_{\mathbf{a}\sim\pi(\mathbf{a}|\mathbf{s})}\left[\mathbb{E}_{Q^{\pi}_{k+1}\sim\mathcal{P}_{\mathcal{D}}(Q^{\pi})}[Q^{\pi}_{k+1}(\mathbf{s},\mathbf{a})] - \alpha \mathrm{Unc}(\mathcal{P}_{\mathcal{D}}(Q^{\pi}))\right]\right]}_{\text{conservative estimate}}, \quad (24)$$

IDEA: Use NTK-GP measure uncertainty?

Transformer RL

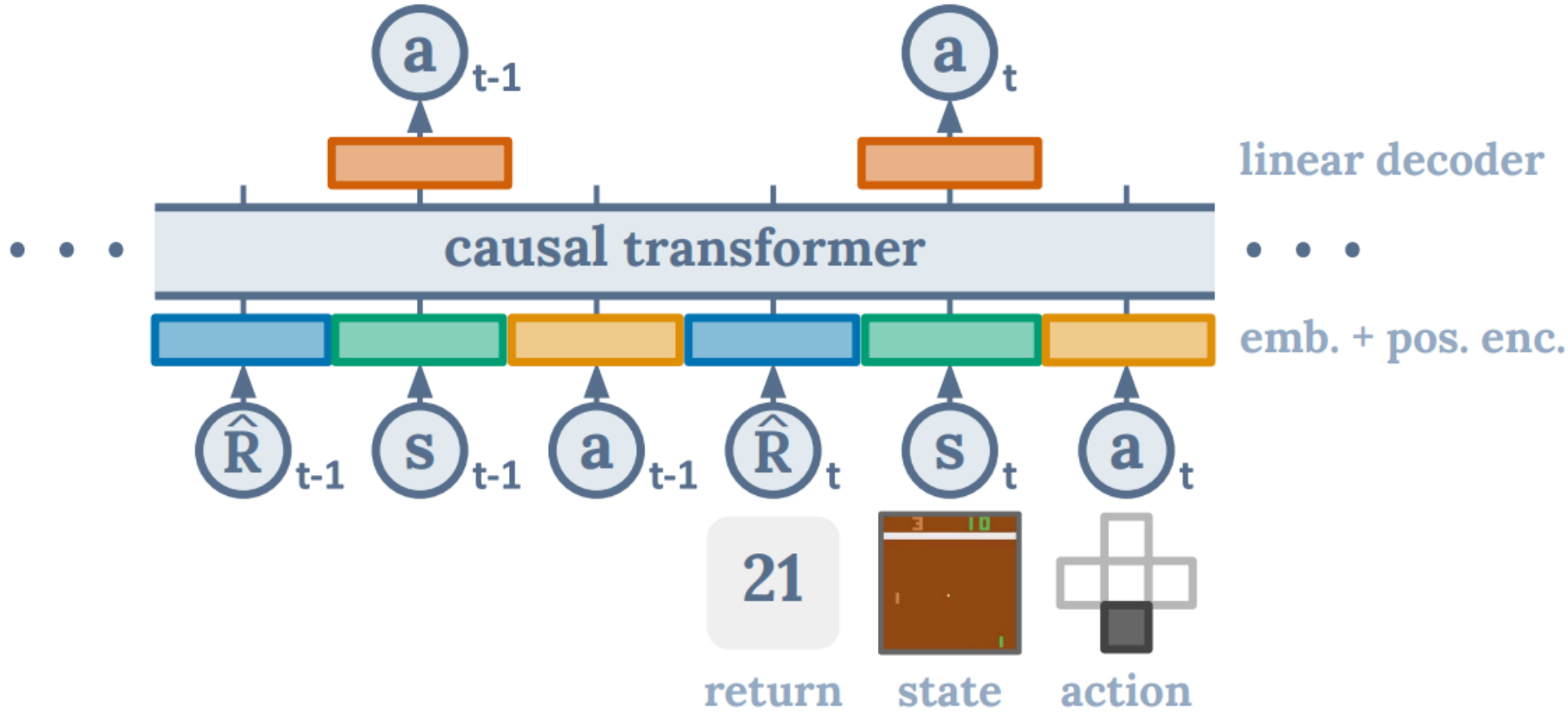


Figure 1: Decision Transformer architecture[1]. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

Performance

Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	**86.8 ± 1.3**	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	**108.1 ± 0.2**	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	**89.1 ± 1.3**	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	**67.6 ± 1.0**	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	**51.2 ± 3.4**	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	**82.7 ± 7.0**	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	**66.6 ± 3.0**	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	**18.0 ± 2.4**	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

Table 2: Results for D4RL datasets[3]. We report the mean and variance for three seeds. Decision Transformer (DT) outperforms conventional RL algorithms on almost all tasks.

Issues

- Cannot use the model online, since we need to produce the reward, compute the reward-to-go and feed it into the model.

Idea

- Predict action a_{t-1} and next reward-to-go $\hat{R}_t$ at time step $t-1$.
- Then kernelize it.

Advantage

- Since the loss is defined as MSE(in the source code). We can use NTK-GP compute the prediction deterministically.
- Can be applied on both offline RL and real-time situation, since it don't need to train it iteratively.
- The next reward-to-go is predicted and we only need to give the first reward-to-go.

But I think I need to double check the architecture. It sounds crazy.