Dataset distillation for offline reinforcement learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Offline reinforcement learning often requires a quality dataset that we can train a policy on. However, in many situations, it is not possible to get such a dataset, nor is it easy to train a policy to perform well in the actual environment given the offline data. We propose using data distillation to train and distill a better dataset which can then be used for training a better policy model. We show that our method is able to synthesize a dataset where a model trained on it achieves similar performance to a model trained on the full dataset or a model trained using percentile behavioral cloning.

1 Introduction

A significant challenge in reinforcement learning (RL) is that the data generation process is coupled with the training process, and data generation requires frequent online interaction with the environment, which is not possible in many settings. Offline RL aims to solve this problem by decoupling the two and training the agent on a given a static, fixed dataset ([17],[20]). However, offline RL relies on a dataset generated by a good expert policy. We often do not have access to data generated by good policies, only mediocre ones. Offline training also means we face the distributional shift problem, where the policy trained on the dataset is produces a different data distribution than the one in the dataset.

Instead of taking the usual offline RL approach of finding a better way to train a model given the offline dataset, we take an alternate approach of asking, is there a way to distill a better offline dataset to train on? We believe that this approaches offers several advantages over finding a better training method. First of all, it is easier to *interpret* a distilled dataset vs a better trained model. Secondly, distillation tends to lead to better generalization capabilities since we learn the key features of the input space ([25], [22]). Thirdly, a distilled dataset is much smaller than the original offline dataset, which improves sample efficiency.

We propose using a method from data distillation [31] known as gradient matching to train a smaller synthetic dataset on the offline dataset. We evaluated the effectiveness of such our method, SYNTHETIC, on the Pro We evaluated the students trained using our procedure on the Procgen environment [5], which consists of procedurally generated games. Specifically, the student is only given access to offline expert policy data on some of the procedurally generated maps, and must generalize the knowledge they learn on those maps to other unseen, out-of-distribution settings. We show that students trained using synthetic data are able to perform similarly or better than students trained on the original offline policy dataset or students trained using percentile behavorial cloning both in distribution and out of distribution, despite the fact that they are *trained on a smaller dataset.*

Why does training on a smaller dataset help in RL settings specifically? RL is a learning paradigm that is natural very prone to randomness and over-fitting due to the fact that the agent also has control of the data generation process. The insight that we have here is that a smaller and well controlled dataset can reduce randomness and overfitting. Just like how humans learn more effectively when read a well written book instead of reading many low quality articles, reinforcement learning agents can also learn a better, more generalizable policy by training on a high quality dataset.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

To summarize, our main contributions are as follows:

- We propose a new method SYNTHETIC that synthesizes a new dataset given a offline dataset of trajectories generated by an expert policy using dataset distillation.
- We show that a RL-model trained on a dataset synthesized using our method is able to perform similarly or better in the environment than directly training on the expert data or other techniques such as percentile behavioral cloning
- We demonstrate how we are able to achieve similar performance with a far smaller dataset when training the RL-model on our synthesized dataset

2 Methodology

We describe our general problem setting, the baseline methods of tackling the problem, and our method here.

2.1 Offline reinforcement learning problem setting

In our reinforcement learning setting, the environment is modelled as a **Markov decision process (MDP)** $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, s_0, s_{-1} \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$, $T(s_{t+1}|s_t, a)$ is the probabilistic transition function, $R(s_t, a) \in \mathbb{R}$ is the immediate transition reward, s_0 is the start state, and s_{-1} is the end state since we are in an episodic setting. A policy function $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from a state to a probability distribution over the action space. We will assume that both the $\mathcal{S}$ and $\mathcal{A}$ are discrete in our setting. When a policy can be parameterized by some parameter θ, we denote the policy as π_θ. Since we are using deep reinforcement learning, π_θ will be a neural network with weights θ.

The goal of parametrized reinforcement learning is to learn the optimal θ^* that maximizes the cumulative of an episode. We define the cumulative reward in terms of the trajectory distribution induced by the policy π_θ. A trajectory τ is a sequence of states and actions that starts with s_0 and ends with s_{-1}, i.e. $\tau = ((s_0, a_0), (s_1, a_1), ..., (s_{-1}, a_{-1}))$. Then the trajectory distribution p_π induced by policy π and environment $\mathcal{M}$ is given by

$$p_\pi(\tau) = \prod_{(s_t, a_t) \in \tau} \pi(a_t|s_t) T(s_{t+1}|s_t, a_t)$$

The expected cumulative reward is then

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi} \left[\sum_{(s,a) \in \tau} R(s, a) \right]$$

and our goal is to find

$$\theta^* = \arg\max_\theta J(\pi_\theta), \quad \max_\theta J(\pi_\theta) \approx \max_\pi J(\pi)$$

In offline reinforcement learning, the agent is not allowed to interact with the environment and collect data through that. Instead, we are given a static data set $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i)\}$ of transitions to learn the best policy π_θ from, where $(s_t^i, a_t^i) \in \tau^i$ are part of the trajectory of episode i. Sometimes the dataset also includes the future return $G_t^i = \sum_{(s_k^i, a_k^i) \in \tau^i, k \geq t} R(s, a)$ at both the current state and for the whole episode, so $\mathcal{D} = \{(s_t^i, a_t^i, s_{t+1}^i, r_t^i, G_t^i, G_0^i)\}$. We assume that the data is generated using some policy π_β, i.e. $\tau^i \sim p_{\pi_\beta}$.

2.2 Behavioral cloning

A common way to approach offline RL is to simply attempt to train a policy π_θ to imitate π_β. This is done in a supervised learning fashion by training π_θ to predict a_t^i given s_t^i and minimizing the loss

$$\mathcal{L}_{\mathbf{BC}}(\theta|\mathcal{D}) = \sum_{(s_t^i, a_t^i, ...) \in \mathcal{D}} w(s_t^i, a_t^i, ...) ||\pi_\theta(s_t^i) - a_t^i|| \tag{1}$$

where $||$ is some notion of distance and w is some weighting on the data-points in $\mathcal{D}$. Usually we take the uniform weighting $w(*) = 1$. Since we usually use stochastic gradient descent (SGD) to minimize the loss, in this case we can simply use $p(*) = \frac{w(*)}{\sum_{(s_t^i, a_t^i, \ldots) \in \mathcal{D}} w(s_t^i, a_t^i, \ldots)}$ as the probability of selecting the sample.

Since π_β might not be optimal, we can attempt to train a better policy π_θ by filtering out observations in $\mathcal{D}$ that lead to poor outcomes. In other words, we let $w(s_t^i, a_t^i, \ldots, G_0^i) = \mathbb{I}_{G_0^i \geq b}$ where b is some threshold on good vs bad outcomes. We call the method of choosing b such that we only end up with x% of the initial dataset, and training a policy π_θ using BC on it as BCx%, or **percentile behavior cloning**. In other words, the goal of percentile behavior cloning is to filter out a better training dataset.

2.3 Synthetic dataset

Instead of filtering out bad samples in order to create a good training dataset, our method *directly learns a good training sample* instead. This is done through dataset distillation, a technique used in supervised learning [31]. We described how we 'train' a synthesized dataset $\mathcal{D}_\phi$ here, parameterized by ϕ, given the offline dataset $\mathcal{D}_{\text{real}}$. Our method aims to reduce the gradient matching loss of ϕ with respect to some random initialization of the model weights θ according to some distribution $\theta \sim p_\theta$. Given some model parameters θ, we first get the gradient of θ with respect to the BC loss we defined in 1 on both the real dataset, $\nabla_\theta \mathcal{L}_{\text{BC}}(\theta|\mathcal{D}_{\text{real}})$, and our synthetic one, $\nabla_\theta \mathcal{L}_{\text{BC}}(\theta|\mathcal{D}_\phi)$. Then we define the gradient matching loss as

$$\mathcal{L}_{\text{grad match}}(\phi|\theta_i) = \mathbb{E}_{\theta \sim p_\theta}\left[||\nabla_\theta \mathcal{L}_{\text{BC}}(\theta|\mathcal{D}_{\text{real}}) - \nabla_\theta \mathcal{L}_{\text{BC}}(\theta|\mathcal{D}_\phi)||\right] \quad (2)$$

We then use SDG to minimize this loss. This method helps guarantee that the synthesized dataset $\mathcal{D}_{\text{syn}}$ will produce a gradient similar to that of $\mathcal{D}$ when a model is trained on it.

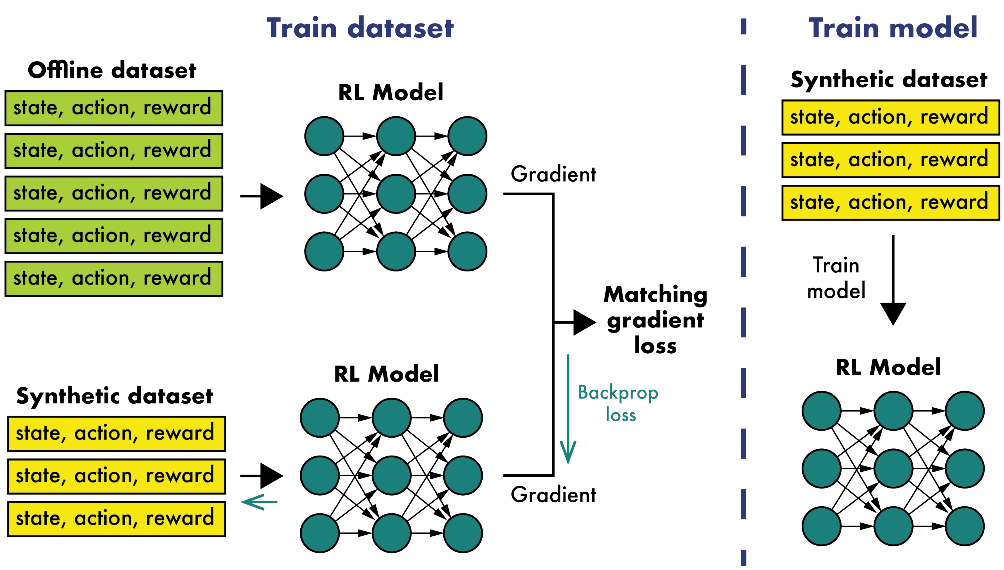

Figure 1: **Overview of our dataset distillation process.** On the left we train the dataset by taking the matching gradient loss between the real offline dataset and our synthetic dataset. On the right we then use the trained synthetic dataset to train a RL model, which we then evaluate on the real environment.

3 Experimental Setup

We provide details on how we implemented our experiments below, including what environments we tested our method on, the architecture for our models, and how we trained the models.

3.1 Environment

We used the *Procgen* environment developed by OpenAI, a suite of 16 procedurally generated environments that allow the creation of adaptive environments with the use of different seeds [5].

The inherent mechanics of these environments serve as an ideal platform for evaluating a student model's ability to learn and adapt to variations introduced by different seeds. Furthermore, the diverse environments help our study as the agent is trained for a variety of challenges that may not occur during standard training procedures. This way we are allowed to scrutinize the agents' performance across different scenarios that could arise in practical implementations. Some sample games from the Procgen environment are shown in Figure 2.

Figure 2: Screenshots of games in Procgen Benchmark [5]

In Procgen environments, the **state space**, consists of 64x64 pixel images (RGB array with three channels and values 0 to 255). The **action space** is discrete in nature and usually includes movements (up, down, left, right) and interactions like collecting items or opening doors. For our experiments, we consider three procedurally generated games: Bigfish, Starpilot, and Jumper. In Starpilot for example, the player must navigate a space ship to avoid being hit by bullets and shoot down enemies in an arcade game fashion. Enemies and obstacles are procedurally generated, so each 'map' is different. The key characteristic of the Procgen benchmark is that, given a different **seed**, the player encounters a different 'map' though the game rules are unchanged. *A key challenge for AI agents lies in how well they can adapt to seeds that they have not seen before*.

3.2 Model architectures and training

3.2.1 Model training

There are two model architectures that we consider – the expert policy model and the student policy model. The **expert model** is the neural network that we use as an expert policy, which is then used to produce the offline dataset $\mathcal{D}$. The **student model** is the neural network which we train on the offline data to produce a policy π_θ. Recall that the goal of offline RL is to optimize the parameters θ in order to produce a good policy, as described in section 2.1.

The expert model is an agent with convolutional architecture found in IMPALA [8], following the convention in Procgen paper [5]. We trained the expert policy on the environment using proximal policy optimization (PPO) [23]. We used the PFRL package and followed one community-created pytorch implementation on GitHub [9, 16]. We trained the expert model for 25 million steps on 200 seeds until it achieved a satisfying level of performance on the environment. Hence, we trained three expert models in total for each of the three environments. This IMPALA network has three convolutional blocks. The first convolutional block has the output channel 16, the next two blocks has the output channel 32. This setup has a total number of $9712+41632+41632+524544+3855+257 = 621632$ trainable parameters. Table 1 shows more details regarding expert model.

For the student (the model that tries to mimic the expert), we use CNN (convolutional neural network) as our base model. The CNN model has 4 convolutional modules followed by a fully connected layer to the logits. The convolutional layers utilize a 3×3 kernel with 3 output channels and coupled with a ReLU activation and a average 2 dimensional pooling layer with pool kernal size 2 and stride 2. For first 2 layers, the dimension of output channel is 4 times larger than that of the input channel. For the last 2 layers, the dimension of output channel is 4 times smaller than that of the input channel. So the output channel of the last layer is the same as the input channel of the first layer. The output of the convolutional layers is then passed to a fully connected layer, which maps to the logits. This leaner setup has a total number of $336 + 5232 + 336 + 327 + 735 = 6966$ trainable parameters, reducing the size of the fully connect layer. Student model has much fewer trainable parameters comparing to that of the expert model. Table 1 contains hyperparameters of training student models.

For each database collection method, we train 10 students and take the average of reward mean and reward standard deviation. We use Adam optimizer with learning rate 5e-3 [12]. For behavioral cloning students, we train them with 1000 steps and batch size 256. For SYNTHETIC students, we train them with only 100 steps and batch size 15.

	Expert	BC Student	SYNTHETIC Student
Model Params	621632	6966	6966
Optimizer	Adam	Adam	Adam
Learning Rate	1e-5	5e-3	5e-3
Batch Size	8	256	15
Steps	25M	1000	100

Table 1: Hyperparameters: Expert V.S. BC Student V.S. SYNTHETIC Student

3.2.2 Data construction

To construct offline RL dataset, we ran 100 episodes of our trained experts on all three environments, and store the data as mentioned in Section 2.1. To construct the synthetic data, given the synthetic data size, we sample data randomly from the offline RL data generated by expert, and then use gradient matching loss to udpate the synthetic data as illustrated in Figure 1. We use our experts to the RL Model to obtain gradients and compute the loss. Here, we use SGD optimizer with learning rate 0.1 and momentum 0.5, training with 1000 epochs.

4 Results

We compare our method to (1) the expert policy that we trained in an online fashion by interacting with the environment until we achieved good performance, which was then used to generate the offline dataset $\mathcal{D}$ and (2) a student trained on datasets with different levels of filtering using percentile behavioral cloning as described in section 2.2. In other words, we want to benchmark how well our method performs at *generating a quality dataset that can be used for training an offline RL model.*

The student model was trained with the same number of stochastic gradient descent steps and same batch size for all baseline methods. We show the in-distribution performance, i.e. the performance on seeds contained in the offline dataset $\mathcal{D}$, of the various methods in table 2 and figure 3. We see here that SYNTHETIC outperforms all percentile behavior cloning methods in both Jumper and Bigfish environments. SYNTHETIC does not outperform percentile BC on Starpilot. We observed that the Starpilot expert mainly takes one action during game – the 'shoot' action – compared to Bigfish and Jumper where the expert takes a more even distribution of actions. Hence the dataset is hard to distill because of the imbalanced samples of different actions.

Since we can procedurally generate out of distribution (OOD) scenarios in Procgen, we also tested the OOD performance of the various dataset generation methods, as shown in table 3 and figure 4. We see here that similar, SYNTHETIC outperforms percentile behavior cloning methods in Jumper environment. SYNTHETIC also matches all percentile BC performances on Bigfish. SYNTHETIC does not outperform percentile BC on Starpilot since the dataset of Starpilot is imbalanced.

Our student model trained by SYNTHETIC only use 150 data samples, as shown in table 4. Given much smaller dataset size (less than half of BC10%) and much fewer training steps as mentioned in table 1, SYNTHETIC achieves competitive results compared to behavioral cloning in different setups. SYNTHETIC also generalizes well out of distribution as the OOD performances matches the ID results, and often times in both Starpilot and Jumper outperforms the ID results.

5 Related work

5.1 Deep Reinforcement Learning

Deep reinforcement learning has seen incredible success recently in tackling wide-ranging problems, from chess and Go to Atari games and robotics [24]. We have also seen great improvements in the

ID Performance						
Environment	**Expert**	**BC 10%**	**BC 25 %**	**BC 40 %**	**BC 100 %**	**SYNTHETIC**
Bigfish	14.27 ± 15.53	0.90 ± 1.44	0.93 ± 1.47	1.01 ± 1.62	1.00 ± 1.67	**1.03 ± 1.99**
Starpilot	28.88 ± 19.41	1.73 ± 2.20	2.10 ± 2.66	**2.17 ± 2.58**	1.85 ± 2.22	1.5 ± 1.96
Jumper	8.79 ± 3.26	1.79 ± 3.54	2.15 ± 4.12	1.95 ± 3.86	2.32 ± 4.13	**2.76 ± 4.40**

Table 2: **Average in distribution performance of student trained on various data collection methods.**

OOD Performance						
Environment	**Expert**	**BC 10%**	**BC 25 %**	**BC 40 %**	**BC 100 %**	**SYNTHETIC**
Bigfish	6.03 ± 9.84	**0.93 ± 1.38**	0.85 ± 1.19	0.83 ± 1.28	0.87 ± 1.32	0.83 ± 1.04
Starpilot	23.34 ± 18.30	1.83 ± 2.39	1.95 ± 2.39	**2.12 ± 2.35**	1.82 ± 2.20	1.54 ± 1.93
Jumper	5.61 ± 4.96	1.81 ± 3.51	1.8 ± 3.73	1.82 ± 3.70	2.50 ± 4.26	**2.86 ± 4.43**

Table 3: **Average out of distribution performance of student trained on various data collection methods.**

Dataset Size					
Environment	**BC 10%**	**BC 25 %**	**BC 40 %**	**BC 100 %**	**SYNTHETIC**
Bigfish	2027	5014	7336	10450	**150**
Starpilot	1116	2796	4192	6830	**150**
Jumper	392	919	1337	4837	**150**

Table 4: **Dataset size used on various data collection methods with respect to different environments.**

architecture used to design such agents, from PPO [23] to decision transformers [4]. As deep neural networks have shown their effectiveness in various tasks, researchers in reinforcement learning have increasingly turned their attention to them. Many complex reinforcement learning situations require these versatile neural networks for tasks such as encoding the states of the agents, learning complex policies, and assessing their performance. [1] give a good overview of various ways deep neural networks were incorporated into reinforcement learning settings.

5.2 Knowledge Distillation

As using large deep neural networks started bringing remarkable success in multiple real-world scenarios related to large-scale data, it became important to deploy deep models within mobile devices and embedded systems. [2] first addressed this issue and proposed compression of large models for transferring the information from large models to train a small model such that accuracy is not hampered. [10] popularized the term 'knowledge distillation' as the process of learning a small model from a large model (teacher) to a small student model. In recent times, there have been many extensions to knowledge distillation where the focus is on compressing deep neural networks. The lightweight student models have paved the way for integrating knowledge distillation in various applications like adversarial attacks [19], security and privacy of data [30], data augmentation [13] etc. KD has been a key instrument in the study of natural language processing (NLP) [7]. [26] and [27] have used some lightweight variations of BERT (called BERT model compression) through knowledge distillation. [11] proposed a TinyBERT, a two-stage transformer knowledge distillation, to make the framework even lighter.

5.3 Policy distillation

There have also been attempts to distill the policy of a expert (teacher) network down to a student network directly [21], known as policy distillation. Policy distillation is a specialized application of knowledge distillation where it adapts the principles of KD in the context of Reinforcement Learning. It is used to transfer knowledge from one policy to another in deep RL. [6]identified three techniques

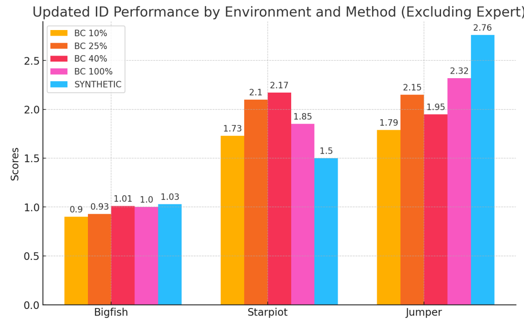

Figure 3: **In distribution performance of various data collection methods**

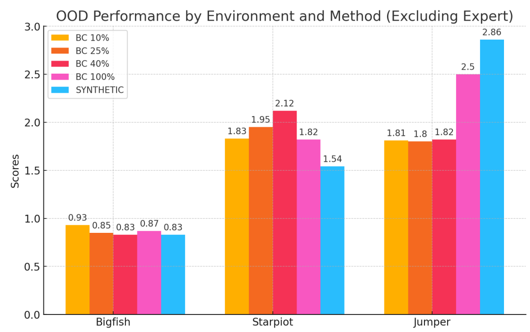

Figure 4: **Out of distribution performance of various data collection methods**

for distillation in DRL, making comparisons of their motivations and strengths through theoretical and empirical analysis, including expected entropy regularized distillation, which ensures convergence while learning quickly. Policy distillation can also be used to extract an RL agent's policy to train a more efficient and smaller network that performs expertly [21].

While our work also teaches a student a policy based on some teacher policy, we take more indirect, offline approach where the student is only allowed to see offline data generated by the expert (teacher).

5.4 Dataset Distillation

Dataset distillation is a dataset reduction method that synthesizes a smaller. In the original work, this is by feeding a randomly initialized model with samples from the real data and samples from the synthetic dataset and taking the gradient of the model with respect to these two data samples [31]. The difference between the two gradients is taken as the loss, and the values of the data in the synthetic dataset are updated using SGD (while keeping the model weights fixed) Since then, a wide variety of different distillation methods have been proposed ([33], [15]). In one such work, instead of matching the gradients for a single sample, the sum of the gradients (total change in model parameters) after training on a series of samples is matched instead ([3]). Despite recent interest in this technique, to the best of the author's knowledge, there have not been any applications of dataset distillation to reinforcement learning yet.

5.5 Task Generalization

The goal of task generalization is to transfer what is learned from doing one task to other tasks, broadening the capabilities of the model. In the ideal scenario, the learned model should be able to apply its knowledge to changing tasks by using the core knowledge learned. ([29]) suggests a new transfer method called "Rule Transfer" which aims to learn the rules of a source task and apply them to other target tasks. ([28]) aims to learn mappings between the transitions from the source to the target task. In the problem suggested in ([18]), agents are required to learn to execute sequences of instructions after mastering subtask-solving skills. The problem gives out a good basis for generalizing to unseen tasks. In ([14]), authors suggest that using reward predictions gives the agents better generalization capabilities.

5.6 Other Works

In the work ([34]), the authors propose a Reinforcement Learning based method for Knowledge Distillation for scenarios where multiple teachers are available. Their work is focused on NLPs and uses large pre-trained models like BERT and RoBERTa, where the framework dynamically assigns weights to different teacher models on each training instance, in order to maximize the performance of the distilled student model. In ([32]), the authors present a novel framework (DRL-Rec) for knowledge distillation between RL-based models in list-wise recommendation, where they introduce one module by which the teacher decides on which lesson to teach to the student and another confidence-guided distillation through which it is determined how much the student should learn from each lesson.

6 Conclusion and limitations

There are several limitations to our study. The first one is that, limited by the computation resources and time we had, we only tested on three environments in Procgen. However, we believe that the experiments on these environments demonstrate the potential of our method, and we look forward to future work on other environments. We also focus on imitation policy learning in our work since our emphasis is on the dataset distillation, not the policy learning method. However, it is also possible to use other RL methods such as q-learning or actor-critic to train policies on the synthetic dataset. We mainly benchmark against percentile behavior cloning, since that is that is the closest method in the existing literature that 'filters' for a better quality training dataset.

In conclusion, we proposed and tested a method that synthesizes a better quality training dataset for offline reinforcement learning. The performance of our method suggests that the quality of the dataset a key component to training a better model, and a smaller but higher quality dataset can lead to similar or better performance compared to a larger one. We believe that these methods can be highly effective in settings with low amounts of data or noisy data, and where data cannot be collected online. There is still much to explore in this research space.

References

[1] Kai Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[2] C Bucilua, R Caruana, and A Niculescu-Mizil. "Model compression, in proceedings of the 12 th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining". In: *New York, NY, USA* 4 (2006).

[3] George Cazenavette et al. "Dataset distillation by matching training trajectories". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 4750–4759.

[4] Lili Chen et al. *Decision Transformer: Reinforcement Learning via Sequence Modeling*. 2021. arXiv: 2106.01345 [cs.LG].

[5] Karl Cobbe et al. "Leveraging Procedural Generation to Benchmark Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 2048–2056. URL: https://proceedings.mlr.press/v119/cobbe20a.html.

[6] Wojciech M Czarnecki et al. "Distilling policy distillation". In: *The 22nd international conference on artificial intelligence and statistics*. PMLR. 2019, pp. 1331–1340.

[7] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[8] Lasse Espeholt et al. *IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures*. 2018. arXiv: 1802.01561 [cs.LG].

[9] Yasuhiro Fujita et al. "ChainerRL: A Deep Reinforcement Learning Library". In: *Journal of Machine Learning Research* 22.77 (2021), pp. 1–14. URL: http://jmlr.org/papers/v22/20-376.html.

[10] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

[11] Xiaoqi Jiao et al. "Tinybert: Distilling bert for natural language understanding". In: *arXiv preprint arXiv:1909.10351* (2019).

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[13] Hankook Lee, Sung Ju Hwang, and Jinwoo Shin. "Self-supervised label augmentation via input transformations". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5714–5724.

[14] Lucas Lehnert, Michael L Littman, and Michael J Frank. "Reward-predictive representations generalize across tasks in reinforcement learning". In: *PLoS computational biology* 16.10 (2020), e1008317.

[15] Shiye Lei and Dacheng Tao. "A comprehensive survey of dataset distillation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[16] Lerrytang. *Lerrytang/train-procgen-PFRL: Pytorch code to train and evaluate Procgen tasks*. URL: https://github.com/lerrytang/train-procgen-pfrl/tree/main.

[17] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[18] Junhyuk Oh et al. "Zero-shot task generalization with multi-task deep reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2661–2670.

[19] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.

[20] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. "A survey on offline reinforcement learning: Taxonomy, review, and open problems". In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[21] Andrei A Rusu et al. "Policy distillation". In: *arXiv preprint arXiv:1511.06295* (2015).

[22] Noveen Sachdeva and Julian McAuley. *Data Distillation: A Survey*. 2023. arXiv: 2301.04272 [cs.LG].

[23] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[24] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.

[25] Samuel Stanton et al. "Does knowledge distillation really work?" In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 6906–6919.

[26] Siqi Sun et al. "Patient knowledge distillation for bert model compression". In: *arXiv preprint arXiv:1908.09355* (2019).

[27] Raphael Tang et al. "Distilling task-specific knowledge from bert into simple neural networks". In: *arXiv preprint arXiv:1903.12136* (2019).

[28] Matthew E Taylor, Gregory Kuhlmann, and Peter Stone. "Autonomous transfer for reinforcement learning." In: *AAMAS (1)*. 2008, pp. 283–290.

[29] Matthew E Taylor and Peter Stone. "Cross-domain transfer for reinforcement learning". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 879–886.

[30] Ji Wang et al. "Private model compression via knowledge distillation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 1190–1197.

[31] Tongzhou Wang et al. "Dataset distillation". In: *arXiv preprint arXiv:1811.10959* (2018).

[32] Ruobing Xie et al. "Explore, filter and distill: Distilled reinforcement learning in recommendation". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 4243–4252.

[33] Ruonan Yu, Songhua Liu, and Xinchao Wang. "Dataset distillation: A comprehensive review". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[34] Fei Yuan et al. "Reinforced multi-teacher selection for knowledge distillation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 16. 2021, pp. 14284–14291.