DISTILLING REINFORCEMENT LEARNING INTO SINGLE-BATCH DATASETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Dataset distillation compresses a large dataset into a small synthetic dataset such that learning on the synthetic dataset approximates learning on the large dataset. Training on the distilled dataset can be performed in as little as one step of gradient descent. We demonstrate that distillation is generalizable to different tasks by distilling reinforcement learning environments into one-batch supervised learning datasets. This demonstrates not only distillation's ability to compress a reinforcement learning task but also its ability to transform one learning modality (reinforcement learning) into another (supervised learning). We present a novel extension of proximal policy optimization for meta-learning and use it in distillation of both a multi-dimensional extension of the classic cart-pole problem and several Atari games. We demonstrate distillation's ability to compress complex RL environments into one-step supervised learning, explore RL distillation's generalizability across learner architectures, and demonstrate distilling an environment into the smallest-possible synthetic dataset.

1 INTRODUCTION

As deep learning increases in popularity and is used on larger learning tasks, training costs impose a greater cost to the deep learning community and the planet. Methods to reduce training costs are more necessary than ever. We propose a method that can reduce reinforcement learning to a single step of gradient descent using an extension of dataset distillation.

Dataset distillation[1] is a technique in which a large dataset is compressed into a synthetic dataset, learnable in as few as a single step of gradient descent (Wang et al., 2018). The performance of a model when trained on the synthetic dataset should approximate the performance of that model when trained on the original dataset. By compressing the original dataset, distillation greatly reduces the cost of training models; in this work, we demonstrate one-step learning made possible with distillation. The compression provided by distillation can be used for low-resource training, interpretability, and data anonymization (Dong et al., 2022). The reduction of training costs resulting from this compression can accelerate searches such as hyperparameter searches, neural architecture searches, and high-performance initialization searches, and can produce inexpensive ensembles.

We propose a generalization of dataset distillation, which we call *task distillation*. Using techniques similar to those used in dataset distillation, we posit that any learning task can be distilled into a compressed synthetic task. Task distillation allows for transmodal distillations, where a task of one modality is distilled into a synthetic task of another modality, allowing a more complex learning task, such as a reinforcement learning environment, to be reduced to a simpler learning task, such as a supervised classification dataset.

In order to advance task distillation beyond supervised-to-supervised distillation, we explore distilling reinforcement learning environments into supervised learning datasets, which we call *RL-to-SL distillation*. We choose to produce synthetic supervised datasets because supervised learning is the bread-and-butter of machine learning, with decades of research going into improving SL. Thus, we consider transforming an arbitrary learning task into supervised learning to be an important goal to simplify machine learning. We select RL environments to distill because reinforcement learning has

[1] Dataset distillation is related to model/policy distillation, but compresses the data rather than a trained model. Here, we utilize "distillation" to refer only to dataset distillation and its generalization, task distillation.

the added complexity of requiring exploration of the environment. Exploration can cause a host of difficulties in learning, such as poor policy spaces that an agent cannot escape and a general increase in training complexity. Learning a successful policy often requires a great deal of exploration, which may involve learning on many repetitive observations that hold little useful information. By producing a compressed synthetic dataset, agents can learn an RL task quickly without exploration costs, as exploration has already occurred during distillation.

The paper begins by exploring distillation on the cart-pole problem (Barto et al., 1983) as a pedagogical exercise; then demonstrates distillation of more difficult RL environments [*Centipede* (Atari, 1980), *Ms. Pac-Man* (General Computer Corporation, 1982), *Pong* (Atari, 1972), and *Space Invaders* (Taito, 1978)]; and makes the following contributions:

1. Proposes a new formulation of RL task distillation using proximal policy optimization.
2. Proposes an N-dimensional extension of the cart-pole environment to allow for scaling the difficulty of cart-pole.
3. Demonstrates k-shot learning on single-batch datasets distilled from ND cart-pole using various initialization distributions and architectures; demonstrating distillation's generalization to unseen architectures.
4. Empirically validates the theoretical work of Sucholutsky & Schonlau (2021a) by demonstrating the minimum distillation sizes of environments with different numbers of action classes.
5. Provides a method that can be used to scale the difficulty of distillation for complex tasks.
6. Demonstrates distillation of complex reinforcement learning environments: Atari and MuJoCo.

2 Background

2.1 Dataset Distillation

Dataset distillation was originally proposed by Wang et al. (2018), using the "meta-model matching method." This method utilizes meta-gradients to learn a synthetic dataset, based on how well a predictive model trained on the synthetic data performs on the true dataset. This process is split into two nested learning loops: in inner learning, a newly-initialized predictive model is trained on the synthetic dataset. In outer learning, the trained predictive model is tested against the true dataset, and the loss is backpropagated through the inner-learning process to the synthetic dataset. The synthetic dataset is updated with gradient descent, and the updated synthetic dataset is used to train a new predictive model in the next iteration.

The goal of dataset distillation is to produce a synthetic dataset $\{X_d, Y_d\}_\theta$ such that predictive models sampled from a given distribution $\lambda_\phi \in \Lambda$ can train on the synthetic dataset to reach high performance on a targeted training task T_0. The original formulation defines a fixed synthetic label vector Y_d; however, in this work we use the soft label formulation of Sucholutsky & Schonlau (2021b), in which Y_d is a parameterized vector of real numbers that can be learned along with the synthetic data instances. The sampled learner is trained on the synthetic dataset using an appropriate loss measure; we utilize mean-squared error between the prediction and soft labels. This inner optimization must be differentiable, as outer learning requires backpropagating through the inner optimization. The trained model can be tested against a sample of T_0 with an appropriate loss metric (generally one that would be used in learning T_0 directly) to produce the outer meta-loss.

We provide a visualization of this formulation of dataset distillation, extended to be task-agnostic, in Figure 1.

2.2 Related Works in Dataset Distillation

A similar method to the meta-model matching method was proposed contemporaneously by Such et al. (2020). This method uses meta-gradient to train a generator model that produces the synthetic dataset, rather than directly learning the dataset. This method, the generative teaching network, was tested on supervised learning tasks and on the simple RL cart-pole environment using A2C. Our

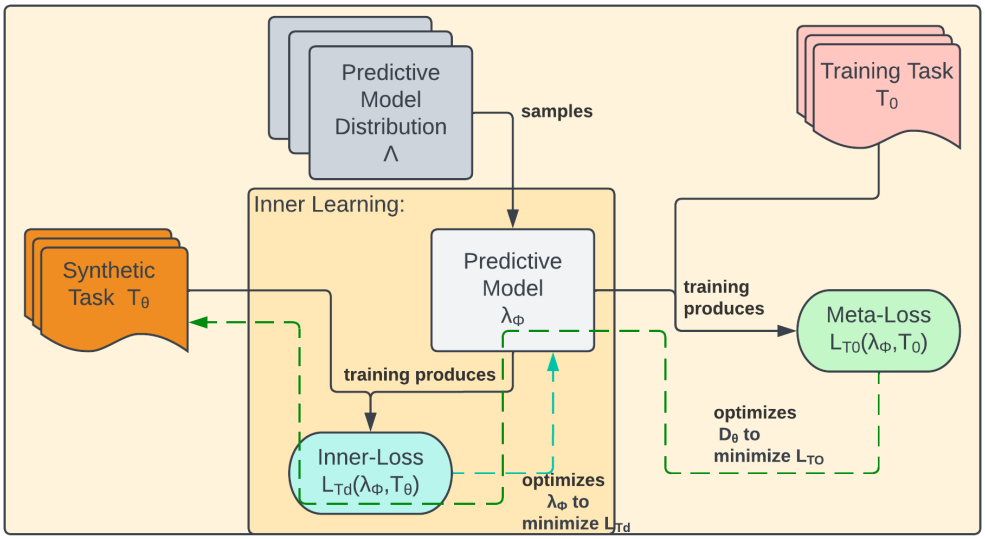


Figure 1: Training process for task-agnostic distillation based on Wang et al. (2018)

experiments are inspired by this work, though we base our method on that of Wang et al. (2018) as our preliminary experiments indicated that parameterizing the dataset provided better results.

Other distillation methods exist, such as gradient matching (Zhao et al., 2021), trajectory matching (Cazenavette et al., 2022), distribution matching (Zhao & Bilen, 2023), differentiable Siamese augmentation (Zhao & Bilen, 2021), and kernel ridge-regression (Nguyen et al., 2021). We base our formulation of that of Wang et al. (2018) because of its simplicity and its independence from expert models and trajectories, though we posit that our method could be adapted to any of these formulations. Other non-distillation methods for simplifying learning, such as behavioral cloning, also require expert examples, and unlike distillation cannot feasibly be used to produce many models due to high training costs.

Lupu et al. (2024) has similar motivations to our work and also tackles distilling reinforcement learning environments. Their approach uses evolutionary strategies to optimize the dataset, while we utilize meta-gradients calculated with PPO loss. Our work distinguishes itself by focusing on minimum-sized datasets and single-step learning, which is feasible given the direction provided by the meta-gradients.

2.3 Proximal Policy Optimization

PPO is a reinforcement learning algorithm that combines two policy gradient methods: actor-critic and trust region optimization. PPO uses two networks: the actor π_ϕ to learn the optimal policy, and the critic V_ψ to approximate the value function (the return at a given state) given policy π_ϕ. The actor determines the actions taken at each timestep at training and evaluation time, while the critic evaluates the actions taken during training only; it performs no function at evaluation time. PPO actor loss over a batch of a given trajectory of observations is as follows (Schulman et al., 2017):

$$L_\pi = -\frac{1}{b}\sum_{1}^{n} \min\left[\frac{\pi_\phi^i(a_t|s_t)}{\pi_\phi^0(a_t|s_t)}A_t, \text{clip}\left(\frac{\pi_\phi^i(a_t|s_t)}{\pi_\phi^0(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A_t\right] \tag{1}$$

where b is the batch size, π_ϕ^0 is the policy network with the parameters used to gather the observation sequences from the environment, π_ϕ^i is the policy network with its current parameters, a_t is the action taken at time t, s_t is the state at time t, A_t is the advantage at time t, and ϵ is the trust region

width. A_t can be calculated in various ways, but we use generalized advantage estimation in our implementation (Schulman et al., 2016).

PPO requires an auxiliary critic network, trained alongside the actor, to determine policy loss. The critic is not used at evaluation time and does not affect the policy after training concludes. The critic is trained to approximate the return R_t expected from timestep t, given the agent is at state s_t and is following the policy π_ϕ. Batched critic loss is as follows (Schulman et al., 2017):

$$L_V = \frac{1}{n}\sum_{1}^{n}(V_\psi(s_t) - R_t)^2 \tag{2}$$

where n is batch size, V_ψ is the critic network, s_t is the state at time t, and R_t is the return at time t.

3 RL-TO-SL DISTILLATION

The learner, or distillation-trained predictive model, must be capable of performing on both the true task T_0 (i.e. act as an RL agent) and the distilled task T_d (i.e. act as a classifier). In RL-to-SL distillation, a learner λ must be capable of both producing a policy $\hat{p}$ for acting in the outer RL environment E_0 based on a state observation: $\hat{p} = \lambda(s)$ for all state observations s in E_0, and label prediction in the inner synthetic SL task based on a data instance: $\hat{y} = \lambda(x)$, for all $x \in X_d$. Once distillation is complete, a new learner $\lambda \in \Lambda$ trained on $\{X_d, Y_d\}_\theta$ should solve the environment E_0, or achieve an acceptable episodic reward on E_0 (without any direct training on E_0).

We define the synthetic data instances and labels to match the dimensionality of the states and actions of the RL environment, respectively. Thus, training the learner, the RL policy network, on the synthetic task is equivalent to learning to classify synthetic states into an action or regressing a state-dependent policy. However, the distiller is not guaranteed to generate data instances that are in or near the state space, nor label a reasonably state-like instance with a reasonable policy. The only constraint on the distiller is how well the dataset it produces can train the learners to perform on E_0.

3.1 PROXIMAL POLICY META-OPTIMIZATION

We consider PPO to be a great candidate for the distillation outer-learning objective due to the benefits of its trust-region protections. However, a direct adaptation of dataset distillation using PPO as the outer objective is not sufficient to utilize these trust region protections. PPO's trust region is created by the clip function, which sets the gradient to 0 when the change in policy throughout learning on a given RL trajectory exceeds a threshold set by ϵ. This requires that the policy network π_ϕ be altered throughout training on the RL trajectory. However, only the synthetic dataset is changed by the policy loss during distillation. Thus, we alter dataset distillation such that the agents' initializations do not change throughout training on a given trajectory (i.e. over a single PPO epoch) to ensure that the only changes reflected in the change in policies is due to optimization on PPO loss, rather than factors such as parameter initialization or model architecture.

We propose a new formulation for task distillation using PPO as the outer objective, and introduce it as *Proximal Policy Meta-Optimization* (PPMO) for RL-to-SL distillation. This blurs the standard formulation's distinction between outer and inner tasks; however, it incorporates the policy trust region protections of PPO.

This process is detailed in Algorithm 1. Training is split into 3 nested loops: meta-epochs, policy epochs, and batched iterations, respectively. At each meta-epoch, a new agent/policy network initialization $\lambda_{\phi_{init}}$ is sampled and trained on the synthetic dataset. This training uses mean-squared error loss on the model's prediction of synthetic data X against the synthetic label Y, and we perform this learning in a single step of gradient descent. The model resulting from this inner-training, π^0_ϕ, defines the policy that is run on the RL environment E_0, creating an RL trajectory τ. Just as in PPO, this trajectory is used for (outer) learning over multiple "policy epochs". For each batch drawn from the shuffled trajectory, PPO policy and value loss is calculated using the formulas defined in Equations 1 and 2, taking in the current parameters of the agent π^i_ϕ, the parameters of the agent used to gather the experiences π^0_ϕ, the value network V_ψ, and the batch of experience data. Note that A_t is calculated using GAE in our implementation. The policy loss is our meta-loss: it is backprop-

Algorithm 1: PPMO for RL-to-SL Distillation($\{X_d, Y_d\}_\theta$, Λ, V_ψ, E_0, e, n, b)

input : initialized synthetic dataset $\{X_d, Y_d\}_\theta$, learner distribution Λ, value network V_ψ, RL environment E_0, number of episodes per iteration e, number of policy epochs n, RL batch size b

output: learned synthetic dataset $\{X_d, Y_d\}_\theta$ distilled from E_0

```
while {X_d, Y_d}_θ has not converged do
    λ_{φ_init} := Sample(Λ);
    π^0_φ, ∇_φ := Train(λ_{φ_init}, {X_d, Y_d}_θ);
    τ = PerformEpisodes(E_0, π^0_φ, e);
    i := 0;
    for policy epoch from 1 to n do
        for (s, a, r) ∈ τ of batch size b do
            L_π, L_V := PPOLoss(π^i_φ, π^0_φ, V_ψ, s, a, r);
            ∇θ := BackpropagateWithMetaGradients(L_π, θ, ∇_φ);
            ∇ψ := Backpropagate(L_V, ψ);
            Optimize θ and ψ w.r.t. ∇θ and ∇ψ;
            π^{i+1}_φ, ∇_φ := Train(λ_{φ_init}, {X_d, Y_d}_θ);
            i := i + 1;
        end
    end
end
return {X_d, Y_d}_θ;
```

agated through the inner-learning process to update the synthetic dataset $\{X_d, Y_d\}_\theta$. The critic is updated with standard optimization, as in standard PPO. A new agent π_ϕ^{i+1} is trained from the same initialization $\lambda_{\phi_{init}}$ as the previous agents in this meta-epoch, and used in the next sampled batch. Once all trajectory batches have been iterated through n times, the next meta-epoch iteration begins by sampling a new agent initialization. By the end of this process, the synthetic dataset can be used to train any agent initialization in Λ to perform on E_0.

4 ND Cart-Pole Experiments

To control the difficulty of cart-pole and the size of its state and action spaces, we expand the cart-pole system into multiple dimensions. In ND cart-pole, the cart and pole have N degrees of freedom, though the cart is limited to accelerating along one axis at each timestep, creating an action space of size $2N$. We utilize this extension of cart-pole to examine k-shot learning and minimum distillation size in RL-to-SL distillation. See Appendix B for a full description of ND cart-pole.

Our cart-pole experiments utilize a standard set of hyperparameters unless otherwise stated. We defined the learner distribution Λ to contain a single architecture, α_c, with random initialization. The architecture, initializations, and PPO hyperparameters used in this work are in line with the standard of Huang et al. (2022), except our removal of learning rate annealing and value loss clipping.

4.1 k-Shot Learning

Once a synthetic dataset has been fully distilled, it can be used for k-shot learning. New agent models, including those not used in distilling, can be trained on the synthetic task using k learning instances. In our experiments, these k instances are learned as a single batch in one gradient descent step. As the distilled dataset was trained to teach agents sampled from Λ, these agents are expected to outperform other agents on the task after k-shot learning—outliers within Λ and out-of-distribution agents may perform significantly worse on T_0 than the average performance of learners from Λ.

We verify the success of our distiller by performing k-shot learning on randomly sampled agent architectures and initializations in Λ on the cart-pole environment (see summary of results in Table 1). Note that, because the agents in Λ used in our experiments share the same architecture but are

Experiment	Λ	α_c: Ortho std=1	α_c: Xe	α_c: Xe std=1	Random h	Random L
1D 2-Shot	500 ± 0	269 ± 204	202 ± 212	278 ± 212	500 ± 0	500 ± 0
1D 512-Shot	500 ± 0	291 ± 227	178 ± 211	280 ± 204	499 ± 6.4	500 ± 0
2D 3-Shot	364 ± 151	15.4 ± 12.6	68.1 ± 58.6	69.1 ± 60.9	433 ± 124	417 ± 160
2D 512-Shot	339 ± 144	17.2 ± 15.2	71.3 ± 62.2	73.2 ± 61.6	445 ± 122	445 ± 117
3D 4-Shot	261 ± 118	12.4 ± 3.7	41.7 ± 27.9	43.3 ± 30.1	375 ± 141	331 ± 171
3D 512-Shot	245 ± 114	12.6 ± 3.7	43.5 ± 29.3	45.9 ± 28.5	421 ± 134	387 ± 135
4D 5-Shot	128 ± 40.8	12.2 ± 2.3	36.5 ± 19.1	36.0 ± 18.6	164 ± 58.5	159 ± 65.6
4D 512-Shot	129 ± 41.5	12.3 ± 2.8	39.6 ± 18.7	39.3 ± 18.4	200 ± 71.1	189 ± 52.3
5D 6-Shot	88.4 ± 29.5	12.7 ± 2.5	36.6 ± 14.6	36.3 ± 14.3	109 ± 35.1	125 ± 50.9
5D 512-Shot	85.4 ± 28.1	13.0 ± 2.5	42.9 ± 15.5	42.0 ± 15.2	142 ± 15.2	160 ± 37.1

Table 1: Mean and standard deviation of rewards over 1,000 agents on distilled ND cart-pole sampled from a variety of distributions. The distributions are as follows: Λ is the agent distribution used in training, consisting of a fixed architecture α_c and using orthogonal initialization with a standard deviation of 0.01 on the final layer and $\sqrt{2}$ on the other layers. "α_c: Ortho std=1" is the same as Λ but with a standard deviation of 1 on all layers. "α_c: Xe" uses architecture α_c but is initialized using Xe initialization with the same standard deviations as Λ. "α_c: Xe std=1" uses Xe initialization with standard deviation = 1 for all layers. "Random h" modifies architecture α_c by randomly sampling the hidden layer size from the set $[32, \ldots, 256]$ (in α_c the size is fixed at 64). "Random L" modifies architecture α_c by randomly sampling the number of hidden layers (number of layers not counting the output layer) from the set $[1, \ldots, 6]$ (in α_c it is fixed at 2). Both Random h and Random L used the same initialization as Λ. For a more thorough examination of h and L values, see Table 6 in Appendix D.

produced by randomly initializing the parameters, we can assume that no agent used in verification was used in training. We also explore deviations from Λ to test k-shot learning on out-of-distribution learners. For each distribution tested, we sample 1,000 model architecture/initialization values, and train each model on the distilled dataset. The model is tested against the RL environment, and the means and standard deviations of rewards over all the tested models is reported. Note that we do not set the random seed of the environment, ensuring any randomness inherent in the environment is preserved for a fair evaluation. For cart-pole, the only random element is the initial state.

Our experiments showed little difference in k-shot learning based on the size of k: learning on the minimum-sized distillation (see Section 4.2) generally performed better than 512-shot with agents from Λ, but performed slightly worse than 512-shot with the other tested distributions. This implies that higher batch sizes are less likely to overfit to Λ. Our experiments show that k-shot learning on T_d appears to be more sensitive to the parameter initialization function (see Ortho and Xe experiments) than to architecture changes (see Random h and L experiments). While this distillation could be used for neural architecture search due to its good generalization to architectures outside of Λ, it may still be biased to architectures close to those in Λ; using a wider array of architectures during training would ensure a fairer evaluation of varied neural architectures.

4.2 Minimum Distillation Size

Distillation compresses a learning task, transforming it from many-shot learning to finite k-shot learning. k, the number of distilled instances to produce, is an important hyperparameter in distillation. Our experiments have determined that varying k has only a small effect on the quality of the distillation, past a certain threshold. Increasing k increases the computation costs of the distillation training non-negligibly, as all k instances are updated in each outer learning step and a distiller with a larger k requires more epochs on average to converge. However, the end-learning costs are negligibly increased, so long as k fits into a single batch on the provided hardware.

There is, however, a minimum threshold k_{min}, such that distillation fails when $k < k_{min}$. This number is determined by the geometry of the label space (Sucholutsky & Schonlau, 2021a). The distilled instances and labels must distinguish the classes or actions from each other. This can be done in fewer instances than the number of classes using soft labels. Using soft labels, the minimum

number of instances to distinguish the classes of a label space with c discrete values is:

$$k_{min} = \lceil c/2 \rceil + 1 \tag{3}$$

This allows for what Sucholutsky & Schonlau (2021a) call "less than one"-shot learning,[2] or learning on less than one instance per class.

Equation 3 was one of multiple demonstrated by Sucholutsky & Schonlau (2021a). We validate this equation experimentally on ND cart-pole and Atari. Distilling to $k \geq k_{min}$ instances results in a successful distillation—the expected reward for $\forall \lambda \in \Lambda$ trained on the distillation is approximately equal to the reward achieved by λ after direct RL training. As an example, 2D cart-pole distillation achieves a mean reward of 339.2 when $k = 512$ and 364.1 mean reward when $k = k_{min} = 3$. When $k < k_{min}$, the distillation is unsuccessful in teaching learners to solve the task, resulting in low average rewards consistent with random performance in cart-pole. 2D cart-pole distilled to $k = k_{min} - 1 = 2$ achieves an average of 40.6 reward.

The equation $k_{min} = \lceil c/2 \rceil + 1$ determines how many instances are required to distinguish the classes of the action space, assuming all actions are necessary. In ND cart-pole, all actions are needed to solve the task, and all actions are performed approximately at the same frequency. Thus, the distilled dataset must provide distinction between each action in its soft labels. That is, there must be at least one instance in which the action probability label is high, and one where it is low, relative to the other action probabilities in the label.

See Figure 4 in Appendix C for a visualization of 1D and 2D cart-pole distilled to k_{min} instances.

5 Atari Environments

We perform RL-to-SL distillation on Atari environments to demonstrate task distillation scaling to more complex and difficult reinforcement learning tasks. The Atari environments have a much larger state and action space than cart-pole, as well as more complex reward functions. Thus, they provide a significant challenge for RL-to-SL distillation.

We have chosen a small selection of Atari tasks for brevity. We have avoided selecting tasks that are difficult for a standard PPO agent to learn, as finding tasks that can be learned by distillation but not by direct-task learning is beyond the scope of this work. As Atari tasks are mostly open-ended reward maximization problems, we determine the success of the distillation by comparing the average reward achieved by randomly sampled agent initializations from Λ trained on a fully distilled dataset $\{X_d, Y_d\}_\theta$, against the average reward achieved by randomly sampled learners from Λ trained directly on the environment using PPO. This is in line with the distillation ratio metric proposed by Sucholutsky & Schonlau (2021b), which also compares performance after direct-task learning versus distilled-task learning.

For the Atari experiments, we define Λ as the distribution of random initializations of a single architecture α_a, derived from the standardized PPO implementation of Huang et al. (2022), as with parameter initialization and all PPO hyperparameters. For the sake of feasibility, we limit the amount of training time for the baseline RL experiments. While it may be possible for the agents to achieve a higher reward given more training time, it is difficult to determine whether an RL agent has fully converged, or if it has reached a temporary plateau in performance; thus, we consider time limits and apparent convergence are a reasonable early-stop heuristic. To provide a baseline for our RL agents themselves, we ensured our baseline PPO agents reached comparable performance to the results of DQN agents in the work of (Mnih et al., 2015).

Due to the increase in complexity in Atari environments versus cart-pole environments, the cost of distillation increases greatly as well. It may be unreasonable to commit the large amount of required resources for a full distillation of an Atari environment without knowing how long distillation will take or if distillation can succeed given the experimental setup; thus, we provide a method for controlling the difficulty of distillation by providing intermediate steps between direct-task learning and distillation. We do so by providing an algorithm that generalizes both direct-task learning and distillation. Then, we demonstrate this method on four Atari environments. We perform these experiments by distilling with $k = k_{min}$ for each of the four chosen Atari environments.

[2] The term k-shot learning is not standardized. In this work, we define k as the number of training instances.

Algorithm 2: Meta-gradient Task Distillation with Encoding($\{X_d, Y_d\}_\theta$, Λ^l, ε^l_ξ, n, T_0)

input : distilled dataset $\{X_d, Y_d\}_\theta$, learner distribution Λ^l, encoder ε^l_ξ, number of inner iterations n, target training task T_0 with data instances X_0 and loss function L
output: distilled dataset $\{X_d, Y_d\}_\theta$ and trained encoder ε^l_ξ

1 **while** $\{X_d, Y_d\}_\theta$ *and* ε^l_ξ *have not converged* **do**
2 $\quad \lambda^l_\phi$:= Sample(Λ^l);
3 $\quad e^l_{X_d} := \varepsilon^l_\xi(X_d)$;
4 $\quad \phi, \nabla_\phi$, := Train($\lambda^l_\phi, e^l_{T_d}, Y_d$);
5 $\quad e^l_{X_0} := \varepsilon^l_\xi(X_0)$;
6 $\quad L_{T_0} := L(\lambda^l_\phi, e^l_{X_0})$;
7 $\quad \nabla_\theta, \nabla_\xi$:= Backpropagate(∇_ϕ, L_{T_0});
8 $\quad$ Optimize θ and ξ w.r.t. ∇_θ and ∇_ξ;
9 **end**
10 return $\{X_d, Y_d\}_\theta$ ε^l_ξ;

5.1 Encoder Rollback for Variable Difficulty

Due to the computational expense of distilling Atari games, we provide a method to scale the complexity of distillation training, providing evidence for the feasibility of full distillations. We utilize an encoder network to simplify the complex state space of the environment without reducing the dimensionality of the synthetic dataset. This formulation with the encoder as all but the final layer is defined and explored by Zhou et al. (2022) as a combination of dataset distillation and neural feature regression. While they propose this formulation to train downstream task learners, we expand this method for difficulty scaling.

An encoder network ε with parameters ξ can be utilized to transform the learner model's input into a smaller space. We use our standard 5-layer Atari learner architecture α_a and split it along layer l, such that the encoder ε^l_ξ consists of layers $[0, l)$ of α_a and the learner λ^l_ϕ consists of layers $[l, 5)$. The encoder's output is fed directly into the learner, just as if they were still a single network. The difference between learner and encoder lies in training: the learner's parameters are still updated by inner learning on the distiller's synthetic task T_d, and the learner is resampled from Λ^l each outer iteration. The encoder, however, is only initialized at the start of training and is trained through outer meta-learning on T_0, rather than inner learning T_d. The encoder is trained alongside the distiller and the auxiliary critic network, rather than with the learner.

We provide the encoder formulation of generalized task distillation as Algorithm 2. The encoder is initialized with the distiller and, unlike the learner, is not reinitialized in the outer loop. The encoder produces encodings of the data associated with both the synthetic task (line 3) and the real task (line 5), which in RL-to-SL distillation includes the synthetic data instances and the environment's states, respectively. The learner predicts on the encoded data, rather than directly on the tasks' data (lines 4, 6). While the learner's parameters are updated through inner learning, the encoder's parameters are updated through outer learning, alongside the distiller (line 8). Note that the distiller could create synthetic data in the encoded space, rather than utilizing the encoder in inner learning. We utilize the encoder in inner learning, as that allows the distiller's output to be the same dimensionality throughout our encoder rollback experiments; thus, we can ensure that the distiller's output space is large enough to hold all information needed at any value of l.

While distilling with an encoder limits the use of the distillation to training only new learners that utilize that encoder, it does allow scaling the difficulty of distillation for tasks where a full distillation may be too computationally expensive. When $l = 0$, the encoder is 0 layers long, and thus training is equivalent to distillation without using an encoder, which we refer to as *full distillation*. When $l = 5$ (the number of layers of α_a), the learner is 0 layers long, and thus the algorithm is equivalent to directly training the encoder network on T_0, i.e. direct-task learning. Since direct-task learning is generally more stable to train and computationally cheaper to perform, higher l values (larger encoders) represent easier distillation tasks closer to direct-task learning, while lower l values

	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	8160 ± 4207	8597 ± 4769	8168 ± 4607	8243 ± 4760	8121 ± 4386	8083 ± 4811
Space Inv.	1947 ± 464	2153 ± 804	$1591^{*} \pm 652$	$710^{*\dagger} \pm 277$		$285^{*} \pm 16$
Ms. P.M.	3894 ± 839	3352 ± 455	$2980^{\dagger} \pm 956$	$2563^{\dagger} \pm 1323$		$403^{*} \pm 281$
Pong	17.3 ± 3.8	$21.0^{\dagger} \pm 0.1$	$12.4^{*} \pm 6.4$			

(a) Average End-Episode Reward Achieved at Convergence

	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	1000	3000	4000	5000	40,000	8000
Space Inv.	115,000	90,000	175,000*	670,000*†		205,000*
Ms. P.M.	53,000	50,000	140,000†	1,300,000†		190,000*
Pong	5000	17,000†	50,000*			

(b) Outer Epochs to Apparent Convergence

Table 2: Results of distillation training with $k = k_{min}$, rolling back an encoder from $l = 5$ to $l = 0$. † indicates that the run was performed with parallelization (6 gpus), so the number of epochs required may not be directly comparable to non-parallelized runs. $*$ indicates the distillation has not fully converged. Empty cells represent experiments that did not exceed random performance due to high computational costs.

(smaller encoders) represent harder distillation tasks. Thus, if we can successfully distill a task with a given encoder, and show that we can roll the encoder back by reducing l and successfully distill the task, we can show how the cost of distillation increases as l decreases. In addition, we can also provide evidence for the feasibility of performing full distillation even for tasks for which the cost of distillation is prohibitive given fixed resources.

5.2 Experiments

In the Atari experiments, a 5-layer agent (with architecture α_a) is used. These agents are standard for PPO learning and are capable of learning the tested Atari environments directly. We demonstrate that the environments can also be learned with an encoder with $l = 4$, where the learner consists of a single linear probe, and for decreasing values of l, at the cost of an increase in computational complexity.

In general, as l decreases, the number of outer epochs to convergence increases, though the amount by which it increases varies across the environments and through different layers of the network. This makes it difficult to accurately predict the cost of a full distillation; however, this method does provide a smoother increase in computation cost per step than jumping from reinforcement learning to distillation. We were able to fully distill *Centipede*, due to its relatively cheap cost of full distillation. The other environments quickly became prohibitively expensive for our resources; however, we assert that full distillations of these environments is possible given sufficient resources (see Table 2).

While these experiments provide evidence for the feasibility of full distillations of these environments, the costs may be prohibitive. However, distillation frontloads computation costs and allows for cheaper training on downstream tasks—as low as one optimization step per model. Given a large enough search space for neural architecture search, distillation is a cost-effective solution. Training Atari agents is expensive, and each agent must explore the environment on its own. Distilling an Atari environment allows for training these agents in a single optimization step without exploration.

For example, our distillation of the Atari *Centipede* environment took approximately 8 times the number of epochs to train than direct-RL on *Centipede*. The meta-optimization and inner training slightly increase the cost of each distillation epoch: on our hardware using one NVIDIA A100 GPU, distillation took on average 3.73 seconds per epoch versus 3.25 seconds per epoch for RL; distillation training took 1.15 times longer per epoch than direct RL training. The benefits of distillation come in training multiple models on the trained distiller. Training on the synthetic dataset is relatively cheap, taking one step of SGD on a single batch of data without any interaction with the environment. Each model takes an average of 0.18 seconds to train on the distilled task versus 3,250 seconds

	RL	Full Distillation
Ant	4919 ± 1488	3485 ± 953
Half Cheetah	4486 ± 1359	2526 ± 693
Hopper	2204 ± 765	1858 ± 1012
Humanoid	1602 ± 1097	543.3 ± 127.7
Humanoid Standup	$212,265 \pm 86,359$	$139,717 \pm 17,699$
Inverted Double Pendulum	8781 ± 1973	1365 ± 2145
Reacher	-5.23 ± 1.83	-8.00 ± 3.43
Walker2D	4299 ± 1156	200.8 ± 257.3

Table 3: Results of distillation on the MuJoCo continuous environments.

of RL, and requiring much less data: 10 instances versus 8,000,000 instances; 1 optimization step versus 15,625. While producing the distilled dataset takes 29,840 seconds with our resources, the entire process of training the distiller and performing k-shot learning on the distilled task is cheaper than direct-RL if one is training more than 9 models sequentially. To demonstrate how quickly this scales: in the time it takes to produce 10 *Centipede* agents, we can distill *Centipede* from scratch and initialize and train 14,777 agents on the distillation. For each additional model trained directly on *Centipede*, we could initialize and train an additional 18,055 agents on the distilled task in the same amount of time. If one is willing to pay the costs of distilling the dataset, they are rewarded with the ability to train 20,000 agents per hour without parallelization.

6 Continuous Environments

Due to its importance in fields such as robotics, we test our distillation method on continuous RL environments. We use the MuJoCo-v4 (Multi-Joint dynamics in Contact) environments (Todorov et al., 2012), which have continuous state and action spaces. Continuous PPO uses the state to predict the mean value of each action and learns a fixed log standard deviation for each action, independent of the state. The distilled dataset is treated similarly to the discrete environments, with the only difference being the dataset trains the learner's mean predictions, while the distiller creates a single standard deviation target. In addition, due to the continuous action space, we opted to distill to $k = 64$ instances, rather than determining k_{min} through trial-and-error. We report our results in Table 3.

7 Discussion

We have demonstrated RL-to-SL distillation, distilling cart-pole and Atari environments into compact, single-batch datasets. We have demonstrated its potential benefits in reducing cost of training. While the up-front expenses of distillation are not insignificant, for applications that require training many models, the per-model speedup may be worth the cost. The cost may be reduced by using better reinforcement learning algorithms, better hyperparameters, and parallelization. Performing more complex experiments on Atari distillation, such as k-shot learning on different learner distributions, could further increase our understanding of RL-to-SL distillation principles explored in our cart-pole experiments; decreased distillation costs would increase the feasibility of such experiments.

Distillation has been proposed for use in downstream tasks such as data anonymity (Dong et al., 2022) and neural architecture search (Wang et al., 2018), and could be viable for other search problems, such as finding high-performance models or good parameter initializations. In addition, methods that require multiple end-task models, such as ensembling, may benefit from distillation. However, distillation has not yet been widely adopted for these use cases, likely due to applications being an afterthought in distillation research and to the computational expense of distillation. Generalized task distillation, and RL-to-SL distillation in particular, opens avenues for real-world application by expanding the types of problems that distillation can be used on. We intend for our future work to examine use cases of distillation to prove its viability for real-world applications.

REFERENCES

Atari. Pong. Arcade, Atari 2600, 1972.

Atari. Centipede. Arcade, Atari 2600, 1980.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

George Cazenavette, Tongzhou Wang, Antonio Torralba, Alexei A. Efros, and Jun-Yan Zhu. Dataset distillation by matching training trajectories. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 4749–4758, 2022.

Tian Dong, Bo Zhao, and Lingjuan Lyu. Privacy for free: How does dataset condensation help privacy? In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 5378–5396, 2022.

General Computer Corporation. Ms. Pac-Man. Arcade, Atari 2600, 1982.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/`. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Andrei Lupu, Chris Lu, Jarek Liesen, Robert Tjarko Lange, and Jakob Foerster. Behaviour distillation. In *International Conference on Learning Representations*, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *International Conference on Learning Representations*, 2021.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

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

Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 9206–9216, 2020.

Ilia Sucholutsky and Matthias Schonlau. "Less than one"-shot learning: Learning N classes from M $<$ N samples. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, volume 35, pp. 9739–9746, 2021a.

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 2220–2227, 2021b.

Taito. Space Invaders. Arcade, Atari 2600, 1978.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei Efros. Dataset distillation. *arXiv*, 1811.10959, 2018.

Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, 2021.

Bo Zhao and Hakan Bilen. Dataset condensation with distribution matching. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 6503–6512, 2023.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

Yongchao Zhou, Ehsan Nezhadarya, and Jimmy Ba. Dataset distillation using neural feature regression. *arXiv*, 2206.00719, 2022.

A Appendix - Hyperparameters and Additional Details

Distillation Hyperparameters	Values
distiller optimizer	Adam
critic optimizer	Adam
inner optimizer	SGD
distiller lr	$2.5 * 10^{-4}$
critic lr	$2.5 * 10^{-4}$
initial inner lr	$2 * 10^{-2}$
outer objective	meta-PPO
inner objective	mean squared error
episodes per epoch	10
reward discount γ	0.99
policy epochs	4
outer batch size	512
rollout length	200

(a) Hyperparameters Used in Cart-pole and Atari

	Cart-pole	Atari
state space	4N	(4x84x84)
action space	2N	18 max (env dependent)

(b) Hyperparameters that Differ in Cart-pole vs Atari

Table 4: Hyperparameters utilized in experiments.

Environment	Number of Actions	k_{min}
Centipede	18	10
Ms. Pac-Man	9	6
Pong	6	4
Space Invaders	6	4

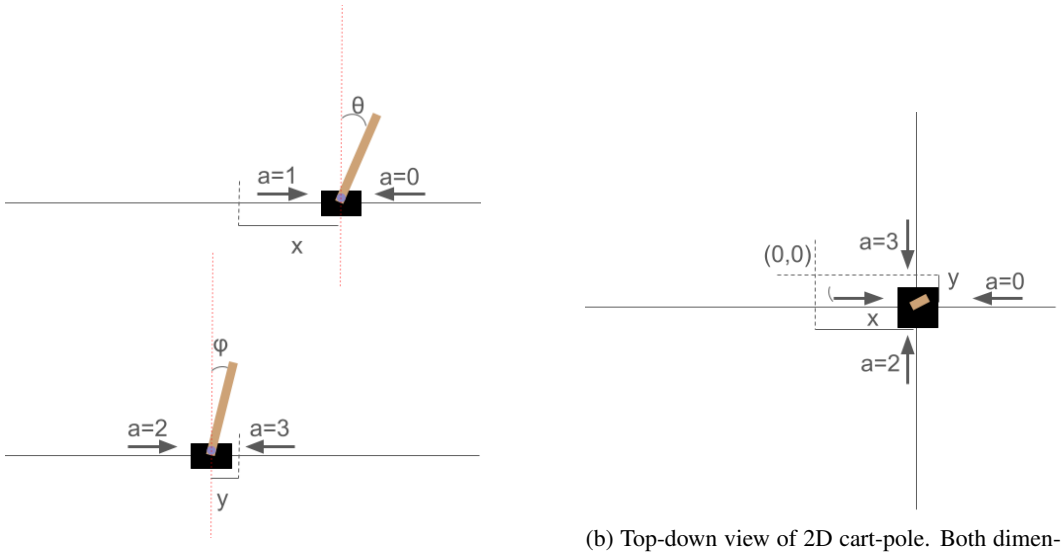
Table 5: Action spaces of Atari environments used.

B ND Cart-pole

In an effort to create a reinforcement environment with a controllable state and action space, we have expanded the classic control cart-pole problem from a 1 degree-of-freedom problem to an arbitrary N degree-of-freedom problem. This problem contains N dimensions in which the cart can be moved and in which the pole can fall. The observation state consists of N sets of quadruples: the cart's position, the cart's velocity, the pole's angle, and the pole's angular velocity in the corresponding dimension; or as 4 vectors of size N. We provide a diagram showing two views of 2D cart-pole in Figure 2

At each timestep, a force of magnitude F is applied to the cart in a direction selected by the agent. To limit the action space to $2N$, we limit the applied force to $\pm F$ along one of the N axes, resulting in a force of 0 along all other axes (other than gravity pulling the pole downward). This is similar to the classic 1D cart-pole problem, where the agent can select to apply a force of $\pm F$ to the cart along the dimension of the track. Another way of framing ND cart-pole is as N independent cart-pole environments, in which the agent applies force $\pm F$ in exactly one environment per timestep, and the other environments receive a force of 0. If the agent fails in any of the parallel cart-poles, it fails in all. Like in standard cart-pole, each iteration provides a reward of +1 until the agent fails. In all ND cart-pole experiments, we terminate the environment once an accumulated reward of 500 is reached, an arbitrary reward ceiling often used in standard cart-pole (as unbounded cart-pole can lead to extremely long episodes).

This environment allows for easy difficulty scaling for reinforcement learning. However, the scaling does not appear to be linear. With a sufficiently high N, the agent cannot act in all dimensions



(a) Two perpendicular side views of 2D cart-pole. The solid lines represent the degrees of freedom of the cart.

(b) Top-down view of 2D cart-pole. Both dimensions representing the 2 degrees of freedom are visible. θ and ϕ are not clearly visible, as they represent the angle between the pole and the axes.

Figure 2: Two views of 2D cart-pole.

before the pole has a chance to fall in one of them. This can be counteracted by altering the time scale (i.e. scaling down velocity and acceleration), giving the agent a quicker reaction time. In our experiments, however, we only utilize up to 5D cart-pole, which can be solved by our PPO agent without changing the time scale.

C Additional Figures

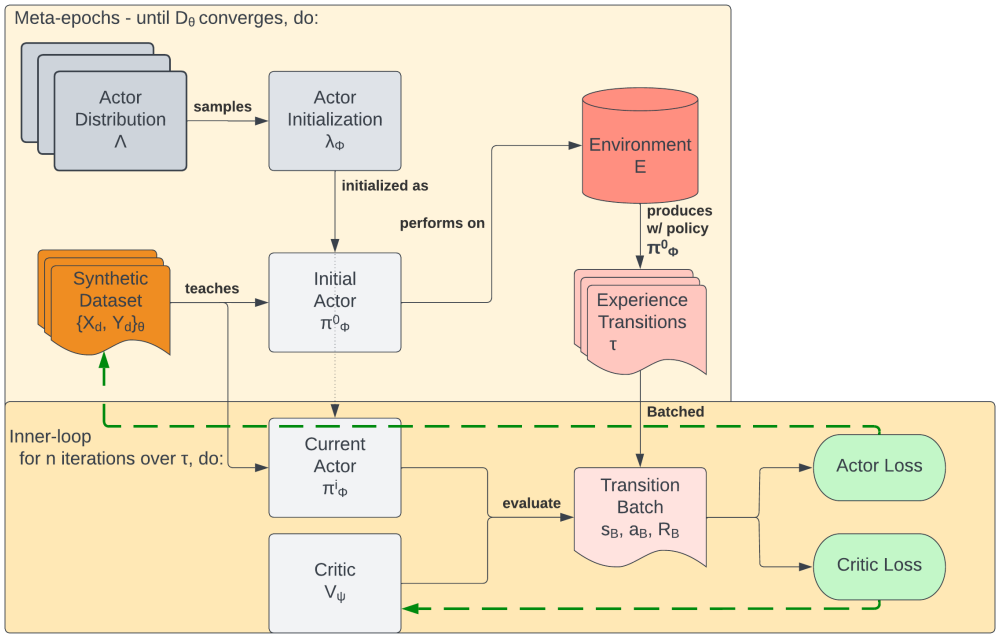


Figure 3: Training process for RL-to-SL distillation with generic actor-critic method.

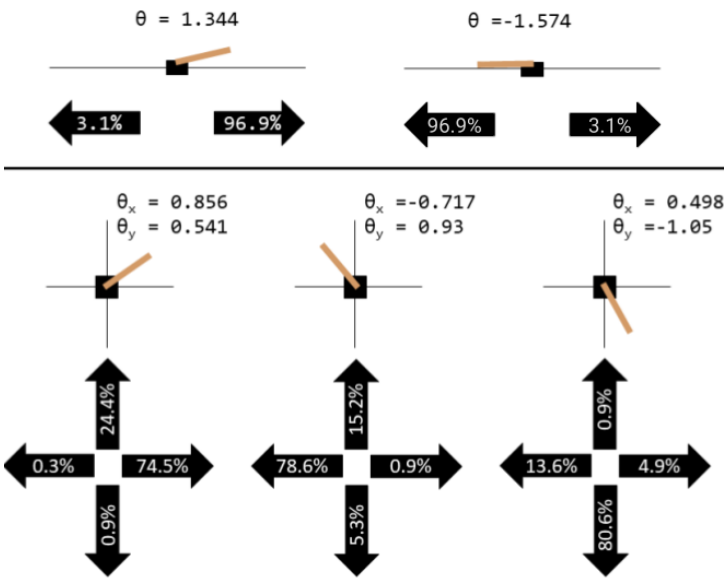


Figure 4: Visualizations of a distillation of 1D (above) and 2D (below) cart-pole with simplified information (only the angle and softmaxed labels are shown). While a full examination of interpretability is outside the scope of this work, this demonstrates that some tasks when distilled provide clear interpretability. In both distillations, the action label demonstrates that the agent should move the cart in the direction in which the pole is leaning. In 2D cart-pole, which was distilled to 3 states despite having 4 actions, the "up" action never has the highest probability mass in any of the labels; instead, the mass is split between two instances. This demonstrates how distillations with less than c instances can teach a model to distinguish all c classes or actions.

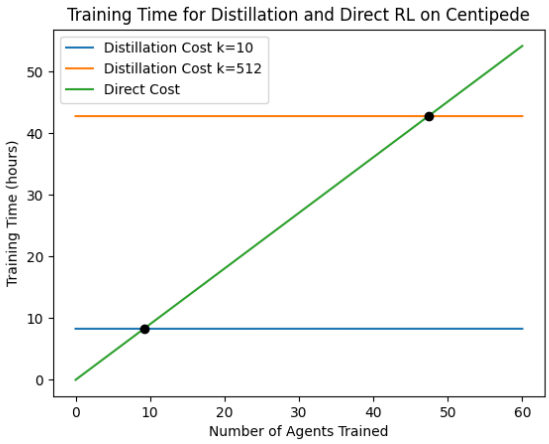


Figure 5: Comparing *Centipede* training time based on number of agents trained between direct-task learning (reinforcement learning), distillation with $k = 512$, and distillation with $k = 10$ ($k_{min} = 10$ for *Centipede*). Note that both distillation lines are parallel with a slope of 0.18 seconds (the cost of training each learner with the final distilled set). Direct-task learning is cheaper than distillation training; however, training each agent after distillation using the distilled set is significantly cheaper than direct-task learning. Distillation training with $k = 10$ is significantly cheaper than distillation training with $k = 512$.

D Additional Results

	h=32	h=64	h=128	h=256
L=0	500.0	500.0	500.0	500.0
L=1	500.0	500.0	500.0	500.0
L=2	500.0	500.0	500.0	500.0
L=3	500.0	500.0	500.0	500.0
L=4	499.9	500.0	500.0	500.0
L=5	494.9	499.5	500.0	500.0

(a) 2-shot 1D

	h=32	h=64	h=128	h=256
L=0	94.4	90.4	93.1	93.2
L=1	490.5	490.5	491.8	491.8
L=2	500.0	500.0	500.0	500.0
L=3	492.7	500.0	500.0	500.0
L=4	447.8	490.4	499.2	500.0
L=5	371.3	448.3	489.4	499.6

(b) 3-shot 2D

L=0	78.7	76.9	78.6	80.4
L=1	430.7	468.7	482.7	488.2
L=2	382.8	447.6	486.6	498.2
L=3	292.7	368.4	441.5	488.1
L=4	220.7	284.8	366.0	437.1
L=5	170.7	221.8	287.0	374.9

(c) 4-shot 3D

L=0	49.1	50.7	51.1	50.9
L=1	177.7	193.1	200.6	208.0
L=2	182.0	210.2	235.7	261.0
L=3	159.1	186.5	217.8	251.3
L=4	135.6	162.1	189.6	225.2
L=5	109.5	136.8	161.9	198.8

(d) 5-shot 4D

L=0	38.0	39.6	39.1	39.4
L=1	125.0	134.5	140.0	143.4
L=2	149.3	165.5	183.2	196.8
L=3	135.1	153.8	175.1	197.5
L=4	113.0	136.3	157.2	181.5
L=5	79.5	115.1	136.0	160.9

(e) 6-shot 5D

Table 6: Extended results for Table 1, obtained by performing k-shot learning and validating on ND cart-pole, using a grid search over L ($[0, 5]$ hidden layers) and h ($2^{[5,8]}$ hidden layer width). Error bars are omitted for brevity, but were of similar magnitude to those of Table 1.

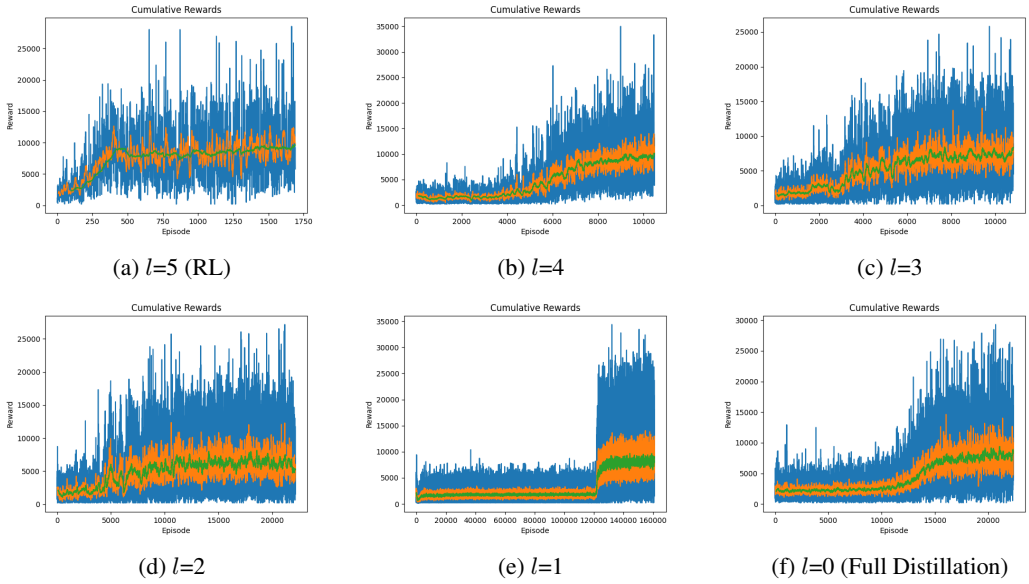
	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	8160	10,955	8251	7874	8327	7694
Space Invaders	1947	2941	1732			277*
Ms. Pac-Man	3894	2987	731*	699*	342*	518*
Pong	17.3	20.5				

(a) Average End-Episode Reward Achieved at Convergence

	RL	$l = 4$	$l = 3$	$l = 2$	$l = 1$	Full Distillation
Centipede	1,000	1,500	3,000	2,000	55,000	40,000
Space Invaders	115,000	195,000	250,000			30,000*
Ms. Pac-Man	53,000	110,000	55,000*	75,000*	90,000*	85,000*
Pong	5,000	40,000				

(b) Outer Epochs to Apparent Convergence

Table 7: Results for $k = 512$ distillation, compare to Table 2.



(a) l=5 (RL) (b) l=4 (c) l=3

(d) l=2 (e) l=1 (f) l=0 (Full Distillation)

Figure 6: End-episode rewards for outer learning on *Centipede* distillation with $k = 10$ (k_{min}).