https://doi.org/10.51483/IJAIML.1.1.2021.31-46 ISSN: 2789-2557

SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Generative adversarial simulator

Jonathan Raiman[1*]

[1] Paris-Saclay University, Paris, France. Email: jonathanraiman@gmail.com

Article Info

Volume 1, Issue 1, July 2021
Received : 20 November 2020
Accepted : 15 May 2021
Published : 05 July 2021
doi: 10.51483/IJAIML.1.1.2021.31-46

Abstract

Knowledge distillation between machine learning models has opened many new avenues for parameter count reduction, performance improvements, or amortizing training time when changing architectures between the teacher and student network. In the case of reinforcement learning, this technique has also been applied to distill teacher policies to students. Until now, policy distillation required access to a simulator or real world trajectories. In this paper we introduce a simulator-free approach to knowledge distillation in the context of reinforcement learning. A key challenge is having the student learn the multiplicity of cases that correspond to a given action. While prior work has shown that data-free knowledge distillation is possible with supervised learning models by generating synthetic examples, these approaches to are vulnerable to only producing a single prototype example for each class. We propose an extension to explicitly handle multiple observations per output class that seeks to find as many exemplars as possible for a given output class by reinitializing our data generator and making use of an adversarial loss. To the best of our knowledge, this is the first demonstration of simulator-free knowledge distillation between a teacher and a student policy. This new approach improves over the state of the art on data-free learning of student networks on benchmark datasets (MNIST, Fashion-MNIST, CIFAR-10), and we also demonstrate that it specifically tackles issues with multiple input modes. We also identify open problems when distillingagents trained in high dimensional environments such as Pong, Breakout, or Seaquest.

Keywords: *Machine learning, Reinforcement learning, Student networks, Data-free learning*

1. Introduction

The motivation for data-free Knowledge Distillation (KD) in the context of Reinforcement Learning (RL) is two-fold. Firstly, to obtain a compact, portable, student policy from an expert teacher whose network architecture can be changed while retaining high performance on the target environment. This KD is of high importance in RL given that model performance is highly variable and often hard to reproduce (Henderson *et al.*, 2018). Secondly, datafree training: to train the student policy without relying on the original data used to create the teacher. Eliminating the reliance on

* *Corresponding author: Jonathan Raiman, Paris-Saclay University, Paris, France. Email: jonathanraiman@gmail.com*

training data is particularly relevant for RL because they can require thousands of years of experience or 100,000+ CPU cores to train (OpenAI *et al.*, 2019, 2020).

Hinton *et al.* (2015) have shown how Knowledge Distillation (KD) can be used to create student models that obtain higher performance by learning from the teacher's behavior than by training on the original examples. Yet KD requires access to the original data used to train the teacher.

Pioneering work on data-free training (Lopes *et al.*, 2017; Nayak *et al.*, 2019; Chen *et al.*, 2019) has shown how to generate prototypical examples by back-propagating through the teacher into its inputs. These generated examples can be used to train student models solely through synthetic examples. But as we will show in this paper, such techniques can fail to generate the multiplicity of prototypical examples for a given teacher output class. This creates blind spots resulting in poor student performance. Unfortunately, RL policies must often map multiple observations to the same single desired action, thereby requiring that synthetic data reproduce a wide range of cases for each output response.

We propose Generative Adversarial Simulator (GAS), a new algorithm for data-free learning which explicitly seeks multiple input modes for each output class by combining an adversarial loss and periodically reinitializing the generator. In our experiments we show how these changes enable better coverage of the input modes and increase the accuracy of a student network trained on data that has multiple exemplars per output class. We demonstrate how we can distill teacher policies into students without a simulator by using our adversarially trained generators. On experiments with RL benchmarks such as Humanoid, MountainCar, CartPole, we show how this technique enables us to distill a teacher. We also perform experiments on the ALE and identify some areas for future work when dealing with high dimensional observation spaces. We also validate GAS on three supervised learning data-free KD benchmarks and obtain a new state of the art on MNIST, Fashion-MNIST, and CIFAR-10.

This paper is structured as follows: first, we will discuss related work, in Section 2 we will provide a description of the problem, in Section 3 we will describe how the GAS algorithm works, in Section 4 we provide results on our experiments, in Section 5 we discuss these results, and in Section 6 we conclude and provide future work directions.

2. Related work

Our work bridges multiple research areas from knowledge distillation, policy distillation, and imitation learning.

2.1. Knowledge distillation

Knowledge Distillation (Hinton *et al.*, 2015) is a technique for training a neural network by supervising its outputs using the outputs of another teacher neural network. The "Dark Knowledge" (Hinton *et al.*, 2014) or additional information contained in the outputs has been used to produce student models that are more compact, achieve higher accuracy than when trained on the original examples, or retain the teacher's accuracy with a noisier feature set (Watanabe *et al.*, 2017).

Our work is most closely related to recent efforts on distilling teachers without access to the original training examples. Initial attempts used meta-data (Lopes *et al.*, 2017) collected during training to reconstruct examples and guide KD. In ZSKD the authors show that it is possible to optimize the inputs using a DeepDream-like (Mordvintsev *et al.*, 2015) procedure and generate several thousand synthetic examples that are sufficient to train models on MNIST and Fashion-MNIST with high accuracy relative to students trained with access to the original examples. In DAFL (Chen *et al.*, 2019) the authors proprose to instead learn an example generator by using a GAN-like (Goodfellow *et al.*, 2014) objective with the teacher acting as a discriminator, and show that this can be used to achieve good performance on CIFAR-10 and CIFAR-100 as well.

In our work we also train an example generator to perform KD without original examples. Unlike DAFL, we replace the 'one-hot' generator loss by a smoother entropy loss, and use the student during the course of training as a way of informing the generator of areas that require additional supervision. Our training procedure also introduces the use of multiple generators and re-initializations to address vulnerabilities in DAFL when an output has multiple associated proto-examples.

2.2. Policy Distillation and imitation learning

Our work applies ideas from data-free KD to policy distillation. KD has been used before in a reinforcement learning context as a way of compacting a teacher policy, or combining teachers into a multi-task student (Rusu *et al.*, 2015). Several techniques have been proposed to blend actions taken by the teacher and student and enable better

consolidation and transfer of knowledge (Ross *et al.*, 2011; Ho and Ermon, 2016; Duan *et al.*, 2017). Until now, policy distillation has relied on access to a simulator.

Imitation learning is another technique for transferring knowledge to a student policy without a simulator by training offline a student to reproduce expert trajectories. In the work of Abbeel and Ng (2004) the authors try to find offline a reward function that corresponds to expert trajectories, while more recent work uses large amounts of human Go and StarCraft 2 games to bootstrap policies (Silver *et al.*, 2016; Vinyals *et al.*, 2019a; 2019b).

Our proposed approach most resembles work on Policy Distillation (Rusu *et al.*, 2015) but removes the need for a simulator or expert trajectories by replacing rollout data by synthetic examples from a generator.

3. Problem statement

3.1. Overview and notation

We are given a teacher function $f_t(x; \theta_t) \rightarrow \hat{y}$, parameterized by θ_t (for brevity we will omit this term), trained with labeled inputs from an unknown but predetermined teacher training distribution $(x, y) \sim (X, Y)$; $X \subset \mathsf{R}^n$; $Y \subset \mathsf{R}^m$. We know the teacher minimizes some distance D between $\hat{y}$ and the label y: $\theta^*_t = \text{argmint}_{\theta t}\{\mathsf{E}\,[\,D\,(f_t(x;\,\theta_t), y)]\}$, we want to obtain a new student function $f_s(x,;\theta_s)$, parameterized by θ_s, which minimizes the expected distance D between the outputs of f_t and f_s:

$$L_{distill}(\theta_s, X) = E_{x \sim X}\,[D\,(f_s(x;\,\theta_s), f_t(x))] \quad ...(1)$$

$$\theta^*_t = \text{argmint}_{\theta s}\{L_{distill}(\theta_s, X)\}$$

Finding θ^*_s is difficult because the teacher training distribution (X, Y) is unknown. A brute force approach could involve sampling N random points from R^n and iteratively minimizing $L_{\text{distill}}(s, \mathsf{R}^n)$ with respect to θ_s, and as $\mathrm{N} \rightarrow \infty$, we expect to $L_{\text{distill}}(\theta_s, X)$ to be minimized as well because $X \subset \mathsf{R}^n$. However the brute force approach faces several difficulties identified in prior work (Lopes *et al.*, 2017; Nayak *et al.*, 2019; Chen *et al.*, 2019): 1) for large n, it may require an intractable amount of points N to minimize $L_{distill}(s, X)$, 2) the student function f_s may not have the same expressive capacity as f_t, and thus sampling points at random might over-subscribe the student, while limiting the capacity over the points of interest.

3.2. Simplifying Assumptions

To make this problem tractable, some simplifying assumptions have been made in prior work about the output space Y and the distance D: if Y is a simplex, and the teacher was trained to minimize the Kullback-Leibler divergence with respect to vertices of this simplex (e.g., categorical class labels), then we can improve over the brute force approach by only sampling points $x_{fake} \sim X_{fake}$ that minimize the entropy $\mathsf{E}_{xfake \sim Xfake}[H(f_t(x_{fake}))]$. We also assume that Y contains all the vertices of the simplex (e.g. the output spans all output classes), therefore we want to ensure that we maximize the entropy of the expected output: $H(\mathsf{E}[f_t(x_{fake})])$.

Because f_t is not necessarily invertible, we now have a second optimization problem, with hyperparameters, α, β find a distribution X_{fake} that minimizes:

$$L_H = \alpha.\,\mathsf{E}_{xfake \sim Xfake}\,[H(f_t(x_{fake}))] - \beta.\,H(\mathsf{E}_{xfake \sim Xfake}\,[H(f_t(x_{fake})]) \quad ...(2)$$

In the DAFL approach (Chen *et al.*, 2019) the authors also add an "activation loss" weighed by activation, where they assume that $x_{fake} \sim X_{fake}$ should maximize the mean absolute value of the last hidden layer $h_t(x)$ of the (neural network) teacher. Combining these assumptions gives us the following non-adaptive generator loss:

$$L_G = L_H - \lambda_{activation}\,.\,\mathsf{E}_{xfake \sim Xfake}[\,|\,h_t(x_{fake})|] \quad ...(3)$$

3.3. Relation to reinforcement learning

Unlike prior work, we are also interested in distilling teachers that were trained through reinforcement learning by interacting with an environment and using a policy gradient algorithm to learn the parameters to maximize the expected sum of future discounted rewards. Because teacher training is driven by environment interaction, there is no associated fixed teacher training distribution such as MNIST or CIFAR-10. In other regards, the teacher functions learnt by policy gradient (commonly named policies π_t) are functionally identical to the teacher functions f_t described above: π_t maps a state x to a distribution y, from which an action a is chosen.

Some of our earlier assumptions still hold in the reinforcement learning setting: the policy π_t will assign higher probability to actions a that maximize expected discounted future reward. If this policy has no entropy regularization, then the entropy of its action distribution will drop[1]. We can assume the points of interest x_{fake} minimize

[1] We assume the environment is stochastic but fixed, unlike for instance a two-player competitive environment where the opponent might adapt to the current player.

$\mathbb{E}_{xfake}[H(f_t(x_{fake}))]$ and if all actions can be sampled then we should also maximize $\mathbb{H}_{xfake}[E(f_t(x_{fake}))]$, therefore our criteria for X_{fake} from Equation (3) remains applicable.

Our assumption that Y is a simplex may not always be true: policies commonly have continuous action spaces, thus Y is either a probability distribution function or a probability mass function. Furthermore, we previously assumed the teacher's training distribution (X, Y) was predetermined, however in a stateful environment the teacher training distribution is path dependent: X_t, the state distribution after t actions are taken, is a function of the previous state x_{t-1} and the previous actions $\{a_0, \ldots, a_{t-1}\}$. For this reason certain states of the world are only reachable if the policy takes a specific sequence of actions (such as finding a key to open a door). This dependency means that improvements in the overall distillation objective from (1) may not always translate into gains in expected rewards in the student policy since certain sequences of actions matter more than others.

3.4. Space invertibility

The output space of the teacher is not always invertible leading to issues with generators only creating a single proto-example per output class. An explanation for this problem is that the teacher function provide a mapping: $f_t : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where there might exist inputs $x_1; x_2, x_1 \neq x_2$ with $f_t(x_1) = f_t(x_2)$, especially when the output space is a simplex, and the teacher was trained with examples that are simplex vertices (categorical labels). This surjectivity lets distribution X_{fake} that contain only a subset of the input modes but maximize the entropy of the teacher output $H(\mathbb{E}_{xfake \sim Xfake}[f_t(x_{fake})])$ to minimize the objective L_H (2).

We illustrate this problem in Figure 1 where a teacher is trained to classify points originating from 4 quadrants of $\mathbb{R}^2$ into 3 classes. The distribution shown in part C) ignores one of the input modes but is still a minima of L_H (2). As a result, the student might produce outputs that disagree with the teacher when sampling from the true distribution (part D) because it never observed data from the missing mode.

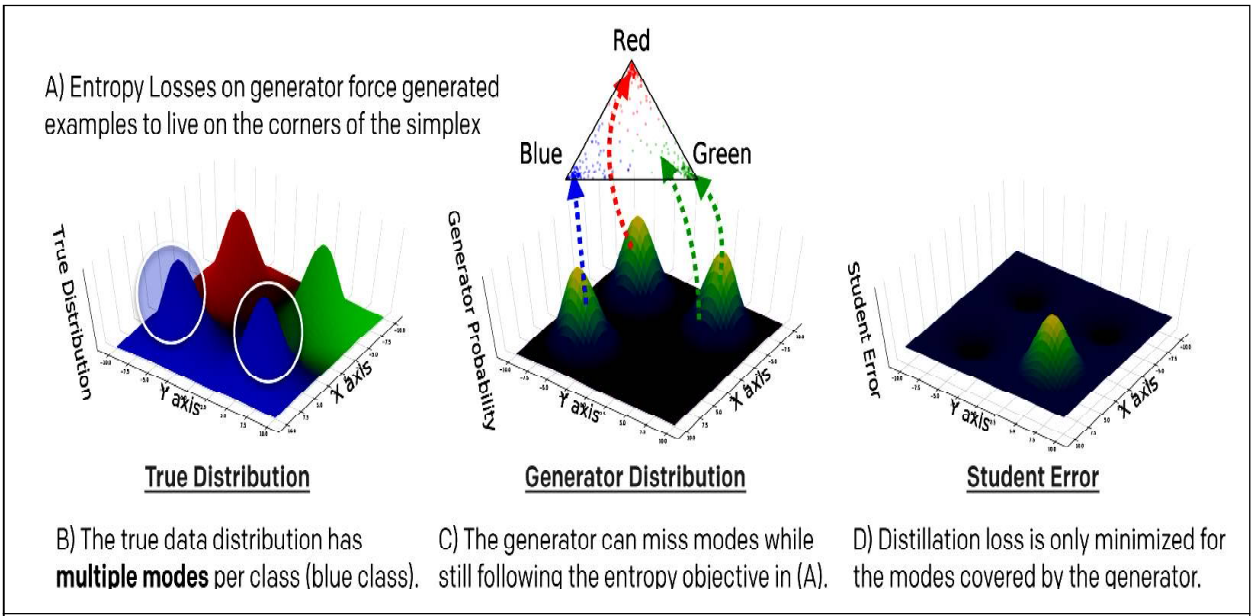

Figure 1: Toy example with 2D data from 3 classes. A generator can minimize the entropy losses despite missing input modes. An adaptive loss biases the generator towards the region where the student disagrees with the teacher, "seeking" out missing modes.

4. Approach

4.1. Goal

We perform KD as introduced in Hinton *et al.* (2014) in a setting where we do not have access to the original examples used to train the teacher (Lopes *et al.*, 2017; Nayak *et al.*, 2019; Chen *et al.*, 2019). Our goal is to learn the parameters θ_s of a student function $f_s(x; \theta_s) \rightarrow \hat{y}$ to minimize the distillation loss defined in Equation (1) given our teacher function $f_t(x)$. We propose to train a Generative Adversarial Simulator (GAS), a model that generates synthetic examples that are maximally informative when distilling the teacher's knowledge into the student. Unlike prior work on data-free KD that

is vulnerable to having only a single proto-example per teacher output, our approach uses an adversarial loss, re-initializations, and multiple generators to seek as many examples as possible for each output class.

4.2. Training objectives

We supervize our student f_s by sampling points from a surrogate distribution $x_{fake} \sim X_{fake}$ and let the teacher label examples $x_{fake}; f_t(x_{fake})$. Since $L_{distill}$ is differentiable with respect to the student parameters θ_s, we can perform stochastic gradient descent to learn θ_s.

To obtain student training data we must construct a source of synthetic examples X_{fake}. Similar to Chen *et al.* (2019), our X_{fake} is a learnt generator that minimizes the objective defined in Equation (3). In Section 3.2 we motivated the use of entropy to obtain a generator that biases the sample generation towards a region of high teacher confidence, as well as ensuring that the generated training data has outputs that span the entire output space (e.g., if f_t outputs a probability mass function, then cover all the vertices of the simplex Y).

Generators X_{fake} that minimize L_G are vulnerable to selecting distributions that only have a single prototype for each output class as we explain in Section 3.4. We propose to extend the generator's objective to explicitly handle multiple input modes by adding a term that biases the generation towards regions of the space where the teacher and the student disagree. As illustrated in Figure 1, a non-adaptive generator distribution produces samples that correspond to vertices of output space simplex Y. We make the generator objective also maximize student error (D in Figure 1), thereby biasing the generation towards areas of the input space that both minimize the entropy of the output, $H(f_t(x_{fake}))$, while also maximize the distance between the student and teacher output. Formally, the adaptive generator loss combines Equations (1) and (3) with a hyperparameter γ:

$$L_{adapt}(\theta_s) = L_G - \gamma L_{distill}(\theta_s; X_{fake}) \quad ...(4)$$

4.2.1. Probability distribution functions

Computing L_H presents a practical challenge in a reinforcement learning setting with a continuous action space policy because we do not always have a closed form expression for $H(\mathbb{E}_{xfake \sim Xfake}[f_t(x_{fake})$. In this work we restrict ourselves to diagonal Gaussian output spaces, making $H(\mathbb{E}_{xfake \sim Xfake}[f_t(x_{fake})$ be the entropy of a Gaussian mixture, that we approximate using the closed-form lower bound given in Huber *et al.* (2008). We reproduce it below, with b the batch size:

$$H_{lower}(x) = -\sum_{i=1}^{b} w_i . \log(\sum_{j=1}^{b} w_j . z_{i,j}) \quad ...(5)$$

with $z_{i,j} = N(\mu_i; \mu_j, C_i + C_j)$

4.3. Neural network architecture

GAS places little requirement on the kinds of neural network architectures used by the teacher and student models. Students and teachers $f_t(x)$ must have outputs that are differentiable with respect to the inputs, and the input and output dimensions for both models must be identical.

Generators in our work change architecture based on the teacher: in image domains we use a DCGAN or another architecture that has transposed-convolutions but does not use Batch Norm (Ioffe and Szegedy, 2015), while in domains without spatial structure we use a deep neural network with Relu nonlinearities but other architecture might work just as well. In both domains the generator receives as input a noise vector, following the approach taken in DAFL (Chen *et al.*, 2019). Moreover, to increase the diversity of the generated outputs we can train multiple generators in parallel that share a single student and teacher. Our noise vector batch is divided among the multiple generators and their outputs are concatenated back.

Algorithm 1: Generative Adversarial Simulator

Input: teacher model $f_t(x)$, $f_t : X \mapsto Y, X \subset \mathbb{R}^n$

Construct generator $g(z, \theta_{g;0})$, Set $i = 0$.

Construct student $f_s(x, \theta_{s;0})$, Set $j = 0$.

for $e = 0$ to E_s **do**

  for step = 0 to N_s **do**

    Sample fake inputs: $\{g(z_1), \ldots, g(z_b)\}$.

Get activations: $\{f_t(g(z_1)),\ f_s(g(z_1)), \ldots\}$

Update student: $\theta_{s,j+1} = \text{Adam}(\nabla\ \theta_{s,j} L_{distill})$.

Set $j = j + 1$.

end for

if e mod R_g is 0 **then**

Reinitialize generator weights $\theta_{g,i}$.

end if

for step = 0 to N_g do

Sample fake inputs $\{g(z_1), \ldots, g(z_b)\}$.

Get activations: $\{f_t(g(z_1)),\ f_s(g(z_1)), \ldots\}$

Update generator: $\theta_{g,i+1} = \text{Adam}(\nabla\ \theta_{g,j} L_{adapt})$.

Set $i = i + 1$.

end for

end for

4.4 Algorithm

GAS involves two losses that must be optimized individually: distillation $L_{distill}$ (1) and an adaptive loss L_{adapt} (4). At the beginning of training, the adaptive loss is not informative because the student is untrained, however when the distance between the student and teacher's output shrinks, it can then be used to improve the generator by identifying missing inputs. We alternate optimization between the student's distillation loss and the generator's adaptive loss.

Because the generator's solution space is very large, it is possible that even with an adaptive loss it does not successfully explore the full input space. We propose to periodically randomly reinitialize weights of the generator to force exploration.

We observe a similar instability phenomenon to GAN training (Goodfellow *et al.*, 2014; Radford *et al.*, 2015; Mescheder *et al.*, 2018) when using an adaptive loss in our generator. For this reason, unlike DAFL that interleaves generator and student training, we alternate the losses and perform a different number of gradient steps on the generator and the student. The exact training procedure for GAS is shown in Algorithm 1.

5. Results

Our experiments focus on three aspects of the problem: first we will examine how input modes impact student accuracy across different data-free distillation techniques, second we compare the effect of our proposed approach to other algorithms on existing benchmarks for data-free KD, thirdly we attempt to distill without a simulator pretrained policies in a variety of reinforcement learning environments and measure the effect of different distillation algorithms on the score of the distilled student.

5.1. Input modes effect on distillation

To isolate the effect of input modes on KD we setup an artificial task where we modify an existing dataset to create experiments where each output class has a set number of input modes. We then train a teacher for each of these datasets and compare the accuracy of students trained with different data-free KD techniques.

Our datasets are constructed by grouping different MNIST digits into a single 'Even' or 'Odd' class. By selecting how many digits we group we can create datasets with 1 to 5 digits per class. We use another MNIST-like dataset (Kuzushiji-MNIST (Clanuwat *et al.*, 2018)) to obtain up to 10 examplars per class.

Our teachers use the LeNet architecture trained using Adam (Kingma and Ba, 2014) for 300 steps with a learning rate of $7e^{-4}$, and our students use the LeNetHalf architecture with half the filters of the teacher. The students are trained using DAFL (Chen *et al.*, 2019), a method that uses only re-initilizations and the generative loss LG, or using GAS, which combines re-initilizations, an adaptive loss, and multiple concurrent generators. Hyperparameters for training the generator and the student are given in Appendix A.

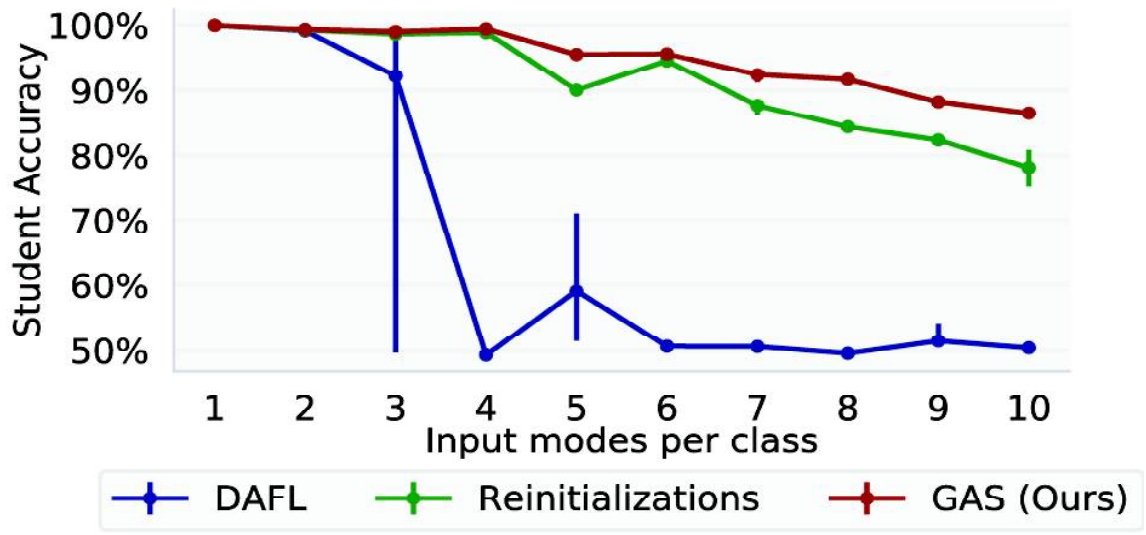

Figure 2: Student accuracy comparison between different data-free KD methods as number of inputs modes per output class is artificially increased. Error bars give 90% confidence interval.

In Figure 2 we observe the student accuracy as the number of input modes for 'Even' and 'Odd' classes is increased. In this plot we notice that past 2 modes per class, the performance of DAFL drops relative to other distillation methods presented. The accuracy of the student decreases with additional input modes. GAS always outperform the other approaches and its accuracy never drops below 85%.

5.2 Data-free supervised learning model distillation

We investigate whether the use of resetting, adaptive losses, and multiple generators improve the performance when distilling supervised learning models. We follow the same experimental setup as Lopes *et al.* (2017), Chen *et al.* (2019), Nayak *et al.* (2019), we train student models from teachers trained on MNIST (LeCun, 1998), Fashion-MNIST (Xiao *et al.*, 2017), and CIFAR-10 (Krizhevsky *et al.*, 2014).

5.2.1. Datasets

We use three different data-free KD benchmark datasets: MNIST, Fashion-MNIST, and CIFAR-10. The MNIST (LeCun, 1998) dataset contains 60,000 training examples and 10,000 test examples showing handwritten digits from 0 to 9. Fashion-MNIST (Xiao *et al.*, 2017) dataset was proposed as an alternative to MNIST, while retaining the same format of 28 x 28 images with 60,000 training examples, and 10,000 test examples. However, the dataset has been known to be more challenging to learn due to the higher inter-class similarity. CIFAR-10 is an RGB image dataset with slightly larger images (32 x 32 x 3). It is composed of 50,000 training examples and 10,000 test examples.

5.2.2. MNIST and fashion MNIST

We study data-free distillation performance by first traininga teacher on this MNIST and Fashion-MNIST. Our teachers are trained using an Adam (Kingma and Ba, 2014) optimizer with learning rate 0.0007, and a Dropout (Srivastava *et al.*, 2014) probability of 0.2. We save the best performing model with a test accuracy of 99.35% on MNIST and 91.17% on Fashion-MNIST. We freeze the teachers and use them to compute generator and distillation losses. To be comparable to prior work we adopt a a LeNet-5 (LeCun *et al.*, 2015) architecture for the teacher, and the student has a LeNet-5- Half architecture (number of filters is halved). Our generator uses the DCGAN architecture from (Radford et al., 2015), which was also used in DAFL (Chen *et al.*, 2019).

Generators are trained with the adaptive loss from Equation (4). At each example generation stage we sample a batch of 512 noise vectors $z \in \mathsf{R}^{100}$, $z_i \sim U(-1, 1)$. These vectors are sliced into 64 sub-batches fed into each of the 8 concurrent generators. We reinitialize the generator once every 4,000 student updates on MNIST, and once every 120,000 student updates on Fashion-MNIST. After each 100 updates to the student, the generators are optimized for 20 steps. The generator and student are trained using the Adam (Kingma & Ba, 2014) optimizer, with a generator learning rate of 0.01 and a student learning rate of 0.001. We repeat our experiment with 3 seeds and report the mean and standard deviation.

Table 1: Classification accuracy on MNIST dataset

Model	Accuracy ($\mu \pm \sigma$)
Teacher	99.35%
Student KD (Hinton *et al.*, 2015) 60,000 original examples	98.92%
Meta data (Lopes *et al.*, 2017)	92.47%
DAFL (Chen *et al.*, 2019)	98.20%
ZSKD (Nayak *et al.*, 2019)	98.77%
GAS w/$\lambda_{activation}$ = 0:1	98:67% ± 0:21
GAS (Ours)	98.79% ± 0:075

Table 2: Classification Accuracy on Fashion-MNIST

Model	Accuracy ($\mu \pm \sigma$)	($\frac{\text{student}}{\text{teacher}}$)
Teacher	**91.17%**	
Student KD (Hinton *et al.*, 2015) *60,000 original examples*	89.66%	98.70%
(Kimura *et al.*, 2019) 200 original examples	72.50%	
ZSKD (Nayak *et al.*, 2019)	79.62%	87.64%
GAS (Ours)	**81.33%** ±0:42	**89.21%** ±0:46

MNIST: In Table 1 we present the performance of several models: the teacher model; Student KD, a model trained using KD with access to the original training data following (Hinton *et al.*, 2015); Meta data, an approach that uses meta data from the teacher during training; (Lopes *et al.*, 2017); and two data-free distillation approaches, ZSKD (Nayak *et al.*, 2019), which uses data-impressions as inputs, and DAFL (Chen *et al.*, 2019), an approach that also trains a generator, but uses a different loss and training algorithm from ours; and our approach, GAS with and without an activation loss. Unlike (Chen *et al.*, 2019), we find that activation losses do not help the generator. We note that our approach outperforms DAFL, and slightly outperforms ZSKD. Despite not using any training data, GAS obtains similar performance to Student KD, which uses the original examples.

Fashion-MNIST: In Table 2 we compare the accuracy of the student obtained through our approach, GAS, to the accuracy from others models. Student KD (Hinton et al., 2015) is trained on the original examples with labels given by the teacher. In (Kimura *et al.*, 2019) the authors use a generative process and a limited number of real examples to train a student. In ZSKD (Nayak *et al.*, 2019) the student is trained using data-impressions without any original examples. To provide a more fair comparison, given that each of these results used a different teacher, we compute the ratio between the student accuracy and the associated teacher accuracy ($\frac{\text{student}}{\text{teacher}}$). GAS outperforms techniques with access to few training examples and other data-free KD approaches such as ZSKD by a large margin.

[2] https://github.com/huawei-noah/ Data-Efficient-Model-Compression.

5.2.3. CIFAR-10

Our setup is identical to the one from (Chen *et al.*, 2019), and we use the implementation released by the authors of (Chen *et al.*, 2019)[2] to train our teacher, implement our adaptive loss, and train students. In this configuration we use a ResNet-34 (He *et al.*, 2016) as the teacher, and ResNet-18 (He et al., 2016) as the student. The student is trained using SGD with momentum 0.9, and a learning rate of 0.1 that is dropped by a factor of 10 every 96,000 gradient steps, for a total of 240,000 steps. The generator is trained using the Adam (Kingma and Ba, 2014) optimizer with a learning rate of 0.015. We perform 1 generator update for each 10 student updates. We set = 0:4 in the generator loss (4).

In Table 3 we compare the accuracy of the student obtained through our approach, GAS, to the accuracy from others models. Student KD (Hinton et al., 2015) is trained on the original examples with labels given by the teacher. DAFL (Chen *et al.*, 2019) is a model where the student is trained without any original examples by using a learnt generator trained by combining several loses similar to the entropy losses in (3), along with an activation loss. We also include our reproduction of the DAFL result obtained when running our experiments. As with Fashion-MNIST, we also compute the ratio between the student accuracy and the associated teacher accuracy ($\frac{\text{student}}{\text{teacher}}$). We observe that GAS is able to reach an accuracy close to that of a student trained with the original examples, and slightly outperforms DAFL.

Table 3: Classification accuracy on CIFAR-10 dataset

Model	Accuracy	($\frac{\text{student}}{\text{teacher}}$)
Teacher	95.36%	
Student KD (Hinton *et al.*, 2015) 50,000 original examples	94.34%	98.70%
DAFL (Nayak *et al.*, 2019)	92.22%	96.48%
GAS (Ours)	**92.35%**	**96.84%**

5.3. Simulator-free policy distillation

Our experiments focus on measuring the score of student policies obtained through simulator-free KD from pretrained teacher policies on a wide range of reinforcement learning environments from classical control benchmarks (Duan *et al.*, 2016) and the Atari Learning Environment (ALE) (Bellemare *et al.*, 2013). Our experiments are conducted using OpenAI gym (Brockman *et al.*, 2016) and utilities from OpenAI baselines (Dhariwal *et al.*, 2017).

5.3.1. Teacher policies

We train policies for the environments listed in Figure 3 using Proximal Policy Optimization (PPO) (Schulman *et al.*, 2017) and a Generalized Advantage Estimator (GAE) (Schulman *et al.*, 2015) to smooth advantages used to update the policies. This approach has been successfully applied to a wide variety of domains in the past (Bansal *et al.*, 2017; Baker *et al.*, 2019; OpenAI *et al.*, 2019; 2020) using both continuous and discrete action spaces. We chose to use a single algorithm for all environments to facilitate reproducibility and simplify comparisons. Because we only use a single teacher algorithm we do not know the exact impact it can have on the final student's performance, and we leave this investigation as future work. We save the parameters of the best performing teacher policy by measuring its average score over 10 episodes over the course of training. Hyperparameters and additional implementation and technical details are given in Appendix A.1.

5.3.2. Classical control

We study several classical control environments that have continuous (Humanoid, Humanoid-Standup, HalfCheetah, MountainCar) and discrete action spaces (CartPole). The environments have input spaces that vary from $\mathbb{R}^2$ to $\mathbb{R}^{376}$, and different dimension output spaces. Using the technique described above we train teachers for all these environments, and then attempt to distill them into student policies.

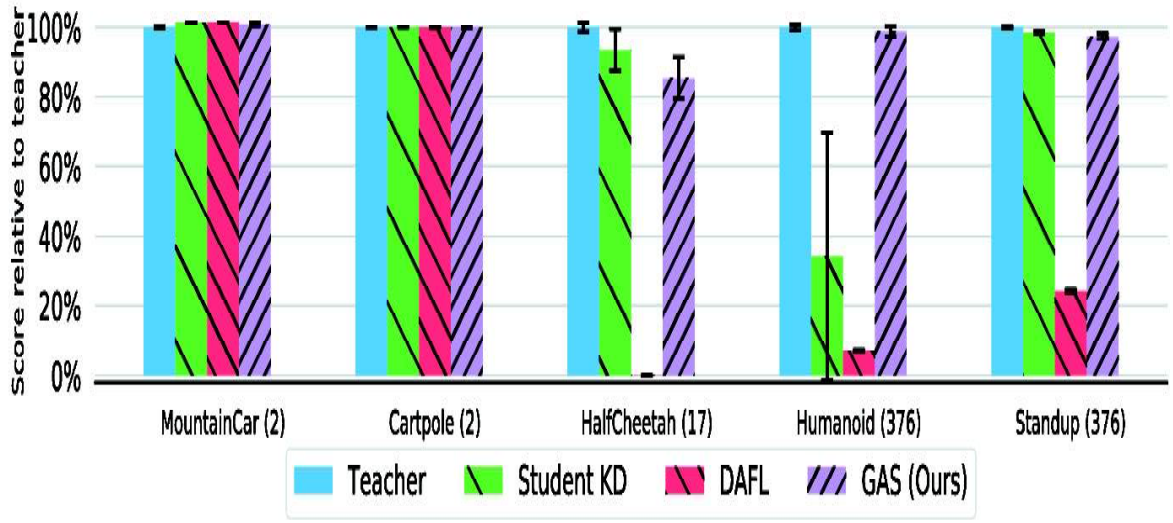

Figure 3: Policy distillation in classical control environments (observation dim). Scores relative to the teachers. Error bars show, N = 5

We compare different KD techniques. Student KD uses real trajectories and minimizes the Kullback-Leibler divergence between the teacher's outputs and those of the student as done in Hinton *et al.* (2015), and Rusu *et al.* (2015). Using DAFL we use a learnt generator to synthesize observations, and apply the distillation objective given in (Chen *et al.*, 2019). Our approach, GAS, uses reinitilizations, the same adaptive loss L_{adapt} (4) used with supervised learning models, and a single generator. Hyperparameters for these experiments are given in Appendix A.2.

In Figure 3 we present the best reward achieved by these different approaches averaged across 5 seeds. We notice that using real trajectories we are able to learn students that approach the teacher's performance. Using DAFL we are able to learn policies with performance matching the teacher only in the environments with the lowest dimensional input space. Finally, GAS is able to learn student policies simulator-free with performance matching the teacher in all environments except HalfCheetah, where we see a slight degradation.

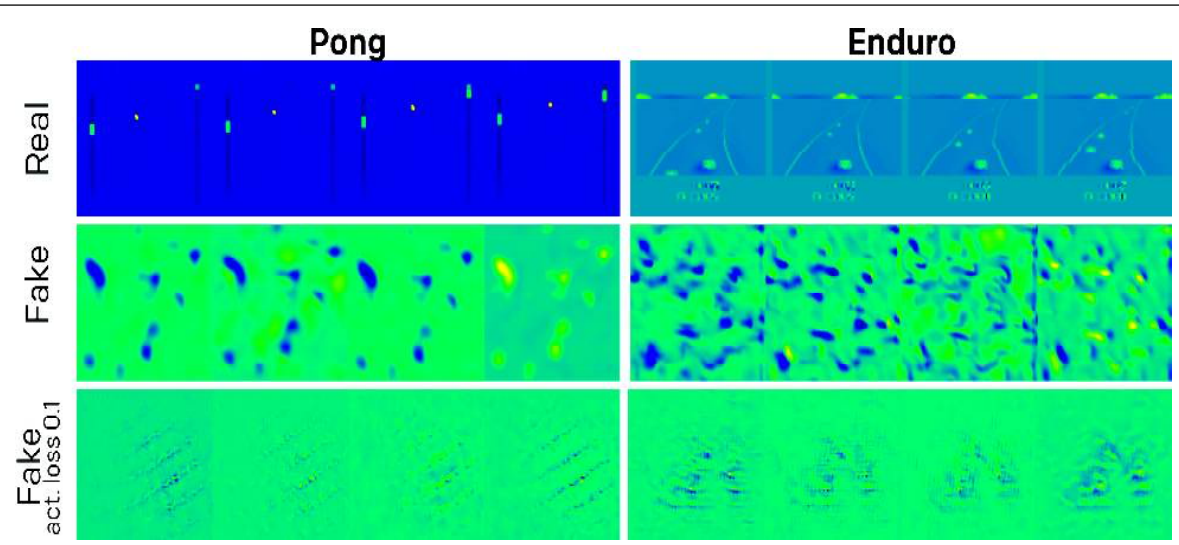

Figure 4: Samples from real inputs and GAS during Enduro and Pong policy distillation. Activation loss leads to recognizable patterns: Enduro road markings and Pong ball trajectories. The presence of multiple ball trajectories suggests the generator learns to produce observation superpositions instead of plausible scenes.

5.3.3. ALE

The Atari Learning Environment (ALE) is considered to be a standard benchmark for reinforcement learning algorithms which presents additional challenges regarding exploration and higher dimensional inputs (160 x 210 RGB frames). We use the Impala (Espeholt *et al.*, 2018) architecture as our teacher policy, and use "Impala-Half" for our student policy (Impala with half-size filters). We obtain teachers by following the steps in Section 5.3.1. We preprocess game frames using the same steps used in Espeholt *et al.* (2018) so our inputs become a stack of 4 grayscale 84 x 84 frames. The inputs to the teachers and students are normalized using a running mean and standard deviation.

Student KD uses real teacher trajectories collected by multiple concurrent workers to train a policy and minimizes the Kullback-Leibler divergence between the student and teacher outputs following (Hinton *et al.*, 2015; Rusu *et al.*, 2015). Our simulator-free approaches, DAFL and GAS, rely on a learnt generator. Here we use a DCGAN architecture (Radford *et al.*, 2015) to generate observations. The generated observations skip the input normalization, so the teacher policy expects those inputs to be 0-mean and standard deviation-1. Both DAFL and GAS fail to learn in these environments. Precise scores in Appendix A.1. With real trajectories Student KD is able to reach performance similar to that of the teacher, confirming the results from Rusu *et al.* (2015). We suspect that the higher dimensional inputs prevented the generators from meaningfully recreating observations that would transfer to the real environment. See Figure 4 for generator samples.

We run an additional experiment with a lower dimensional ALE environment to understand if simplifying the generation task can improve GAS's performance. Generators in the ALE domain produce observations that are 75 x larger than those from Humanoid and 9:2 x larger than CIFAR-10. We propose to use a 2 stacked frame 48 x 48 Pong environment (1:5 x CIFAR-10) and find that the generations now contain paddles but GAS's score remains low (–18:87 $\pm$ 1:17).

6. Conclusion

In this work we considered of data-free KD in the context of RL. We identify a vulnerability in existing data-free KD algorithms that can prevent them from providing a multiplicity of examples for each output class and limits their applicability for simulator-free policy distillation. In an experiment we demonstrate how artificially increasing the number of proto-examples per class lowers the accuracy of the data-free KD technique DAFL.

Our key contribution is Generative Adversarial Simulator (GAS), an algorithm for data-free KD that explicitly seeks to overcome difficulties faced by having multiple examples per output class through the use of adaptive loss, reinitializations, and concurrent generators. We show that this algorithm produces high accuracy students despite increasing input modes. We validate GAS on standard benchmarks and find that it obtains a new state of the art on existing benchmarks MNIST, Fashion-MNIST, CIFAR-10.

We also introduce the task of simulator-free policy distillation. We train teacher policies on 9 different RL environments to compare policy distillation techniques. On ALE we find that neither DAFL nor GAS successfully trains student policies and can compete with using real trajectories. On classical control tasks we demonstrate that GAS successfullytrains student policies without a simulator. Our approach improves over DAFL in higher dimensional environments, providing some evidence that GAS opens up the possibility to distill policies without simulators in larger and more complex environments. Yet, more work is needed to address higher dimensions and would benefit from from understanding the connection between the teacher-driven observationsgenerations from GAS and those found in World Models (Ha and Schmidhuber, 2018) or in model-based RL such as MuZero (Schrittwieser *et al.*, 2019).

References

Abbeel, P., and Ng, A.Y. (2004). Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning, pp. 1.

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autocurricula. arXiv preprint arXiv:1909.07528, 2019.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent complexity via multi-agent competition. arXiv preprint arXiv:1710.03748.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, 253-279.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. (2016). arXiv preprint arXiv:1606.01540.

Chen, H., Wang, Y., Xu, C., Yang, Z., Liu, C., Shi, B., Xu, C., Xu, C., and Tian, Q. (2019). Data-free learning of student networks. In Proceedings of the IEEE International Conference on Computer Vision, 3514-3522.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. (2018). *Deep learning for classical japanese literature.*

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In International conference on machine learning, 1329-1338, 2016.

Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). Oneshot imitation learning. *In Advances in Neural Information Processing Systems*, 1087-1098.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., *et al*. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. arXiv preprint arXiv:1802.01561.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *In Advances in Neural Information Processing Systems*, 2672-2680.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In Advances in Neural Information Processing Systems, pp. 2450–2462, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770-778.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In Thirty-Second AAAI conference on Artificial Intelligence.

Hinton, G., Vinyals, O., and Dean, J. (2014). Dark Knowledge. Presented as the keynote in BayLearn, 2.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In Advances in Neural Information Processing Systems, 4565-4573.

Huber, M. F., Bailey, T., Durrant-Whyte, H., and Hanebeck, U. D. (2008). On entropy approximation for gaussian mixture random vectors. In 2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, 181-188. *IEEE.*

Ioffe, S., and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.arXiv preprint arXiv:1502.03167.

Kimura, A., Ghahramani, Z., Takeuchi, K., Iwata, T., and Ueda, N. (2019). Few-shot learning of neural networks from scratch by pseudo example optimization. In British Machine Vision Conference 2018, BMVC 2018.

Kingma, D. P., and Ba, J. Adam. (2014). A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Krizhevsky, A., Nair, V., and Hinton, G. (2014). The cifar-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html, 55.

LeCun, Y. (1998). The mnist database of handwritten digits. http://yann. lecun. com/exdb/mnist

LeCun, Y. *et al*. (2015). Lenet-5, convolutional neural networks. URL: http://yann. lecun. com/exdb/lenet, 20, 5.

Lopes, R.G., Fenu, S., and Starner, T. (2017). Data-free knowledge distillation for deep neural networks. arXiv preprint arXiv:1710.07535.

Mescheder, L., Nowozin, S., and Geiger, A. (2018). Which training methods for gans do actually converge? In International Conference on Machine Learning (ICML), 2018. Generative Adversarial Simulator.

Mordvintsev, A., Olah, C., and Tyka, M. (2015). Inceptionism: Going deeper into neural networks.

Nayak, G. K., Mopuri, K. R., Shaj, V., Babu, R. V., and Chakraborty, A. (2019). Zero-shot knowledge distillation in deep networks. arXiv preprint arXiv:1905.08114,.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., D²ebiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning.

OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., *et al*. (2020). Learning dexterous inhand manipulation. *The International Journal of Robotics Research*, 39(1), 3-20.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Ross, S., Gordon, G.., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, 627-635.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy Distillation. arXiv preprint arXiv:1511.06295.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., *et al*. (2019). Mastering atari, go, chess and shogi by planning with a learned model. arXiv preprint arXiv:1911.08265.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-dimensional continuous control using Generalized Advantage Estimation. arXiv preprint arXiv:1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al*. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), 1929-1958.

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R., *et al*. (2019a). Alphastar: Mastering the real-time strategy game starcraft ii. DeepMind blog, 2.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., *et al*. (2019b). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782), 350-354.

Watanabe, S., Hori, T., Le Roux, J., and Hershey, J. R. (2017). Student-teacher network learning with enhanced features. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5275-5279. *IEEE*.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747.

Appendix

A. Hyperparameters

In this section we include the experiment hyperparameters. All experiments are run on a 12-core 3.60 GHz Intel i7-6850 K CPU with an NVIDIA Titan X Pascal, using Tensorflow 1.15, CUDA 10.1 and CuDNN 7.5.1.

Table 4: Input Mode Effect Hyperparameters

Model	Optimizer	LR	Architecture
Generator	Adam	0.01	DCGAN without Batch-Norm
Student	Adam	0.001	LeNetHalf
Teacher	Adam	0.0007	LeNet

Table 5: Data-Free Distillation Benchmark Hyperparameters

Task	Optimizer	Student/Generator LR	Teacher	Student
MNIST	Adam	0.0001 / 0.01	LeNet	LeNetHalf
Fashion-MNIST	Adam	0.0001 / 0.01	LeNet	LeNetHalf
CIFAR-10	Adam	Schedule / 0.015	Resnet34	Resnet18

A.1. Reinforcement Learning Teachers

We train reinforcement learning agents asynchronously using a single GPU central optimizer machine connected to a series of workers that update their parameters after a set number of transitions. The workers send their observations to a central actor GPU that can batch and process multiple requests. The workers act in parallel with

Table 6: Common teacher hyperparameters

Parameter	Value
Algorithm	PPO
PPO Clip Ratio	0.2
Gradient steps per update	10
Concurrent environments	512
Entropy penalty	0.0
Optimizer	Adam
GAE	0.95
Concurrent environments	512
Transitions per rollout	32
Max version lag	4

Appendix (Cont.)

the optimization and regularly send their transition data to a queue on the optimizer machine that normalizes the measure advantage and adds this data to the experience buffer. Data is added to the experience buffer if the rollout was produced within some version lag to avoid straggler workers. Observations for all teachers are normalized using a running mean and standard deviation computed on the optimizer machines. We update these statistics alongside the model parameters, thereby ensuring that the models observe mean 0 and standard deviation 1 data.

For each environment we train policies until they have stopped improving or we collect more than 10^7 transitions samples. We save the policy parameters whenever the policy achieves a higher reward across multiple episodes when evaluated.

Table 7: Environment specific teacher hyperparameters

Environment	Model	Replay Buffer Size	Batch Size	LR	Discount Factor (γ)
Humanoid	MLP([64 64])	8192	1024	$3e^{-4}$	0.990
Standup	MLP([64 64])	8192	1024	$3e^{-4}$	0.990
MountainCar	MLP([512])	15000	5000	$1e^{-3}$	0.995
Cartpole	MLP([512])	16384	2048	$1e^{-3}$	0.990
ALE	Impala (Espeholt *et al.*, 2018)	16384	2048	$1e^{-3}$	0.990

A.2. Reinforcement learning distillation

Reinforcement Distillation uses for vector environments (Humanoid, Cartpole, MountainCar) a 100-dimensional uniform [–1, 1] noise vector to drive a single-layer MLP. For ALE environments we use a DCGAN architecture (Radford *et al.*, 2015).

Table 8: Classical control scores ($\mu \pm \sigma$, N = 5) for student policies

Method	Humanoid	Standup (x10^4)	HalfCheetah	MountainCar	Cartpole
Teacher	5896:00 ± 48:29	15:920 ± 0:042	7698:0 ± 103:1	94:30 ± 0:2547	200:0 ± 0:000
Student KD	012 ± 2101	15:710 ± 0:060	7199:0 ± 459:9	95:77 ± 0:3915	200:0 ± 0:000
DAFL	418:10 ± 10:75	3:854 ± 0:056	–0:3870 ± 0:2094	95:87 ± 0:1889	200:0 ± 0:000
GAS	5833:00 ± 78:38	15:52 ± 0:12	6580:0 ± 445:3	95:13 ± 0:3705	200:0 ± 0:000
Input Space	376	376	17	2	4
Action Space	R^{17}	R^{17}	R^6	R^1	Cat(2)

Table 9: ALE scores ($\mu \pm \sigma$, N = 5) for student policies

Method	Pong	Breakout	Seaquest	Enduro
Teacher	20:7000 ± 0:6403	355:5 ± 158:9	2624:0 ± 552:4	643:10 ± 27:22
Student KD	20:2000 ± 0:2683	213:5 ± 144:5 10	15:0 ± 577:6	119:90 ± 63:64
DAFL	–20:6000 ± 0:3033	0:8400 ± 0:4800	110:00 ± 15:49	4:920 ± 4:234
GAS	–20:5400 ± 0:3980	0:8800 ± 0:9683	123:60 ± 21:22	24:220 ± 9:529
Action Space	Cat(6)	Cat(4)	Cat(18)	Cat(9)

A.2.1. Distillation stopping criteria

When doing data-free policy distillation we measure student progress using a periodic evaluation. When the score on these evaluations stops improving for some amount of time or some number of new parameter versions we stop the experiment and save the student with the best recorded performance.

On ALE environments we stop experiments after 10,000 new parameter versions stop improving. On classical control environments we stop after 120 seconds without progress.

A.2.2. Distillation hyperparameters

Student policies use neural network architectures with half the output size and filters of the teacher architectures given in Table 7. Students and generator use the Adam optimizer, and we use $\alpha = 0{:}5$ and $\beta = 5$ in the generator objective L_H (2). Other GAS hyperparameters can be found in Table 10.

Table 10: GAS hyperparameters for policy distillation

Environment	Student LR	Generator LR	Student Steps	Generator Steps	Generator Reset
Humanoid	0.001	0.001	5	2	10
Standup	0.001	0.001	5	2	10
MountainCar	0.001	0.001	5	2	10
Cartpole	0.001	0.001	5	2	10
ALE	0.0003	0.1	7	5	256

A.3. Supervised learning teachers

A.3.1. MNIST and Fashion-MNIST

Our MNIST and Fashion-MNIST teachers are trained using an Adam (Kingma and Ba, 2014) optimizer with learning rate 0.0007, and a Dropout (Srivastava *et al.*, 2014) probability of 0.2. We save the best performing model with a test accuracy of 99.35% on MNIST and 91.17% on Fashion-MNIST. We freeze the teachers and use them to compute generator and distillation losses. To be comparable to prior work we adopt a a LeNet-5 (LeCun *et al.*, 2015) architecture for the teacher, and the student has a LeNet-5-Half architecture (number of filters is halved). Our generator uses the DCGAN architecture from (Radford *et al.*, 2015), which was also used in DAFL (Chen *et al.*, 2019). MNIST and Fashion

MNIST generators use the adaptive loss. sample a batch of 512 noise vectors $z \varepsilon \mathsf{R}^{100}$, $z_i \sim U(-1, 1)$. These vectors are sliced into 64 sub-batches fed into each of the 8 concurrent generators. We reinitialize the generator once every 40,000 student updates on MNIST, and once every 120,000 student updates on Fashion-MNIST. After each 100 updates to the student, the generators are optimized for 20 steps. The generator and student are trained using the Adam (Kingma and Ba, 2014) optimizer, with a generator learning rate of 0.01 and a student learning rate of 0.001. We repeat our MNIST and Fashion-MNIST experiment with 3 seeds.

A.3.2. CIFAR-10

We use a ResNet-34 (He *et al.*, 2016) architecture as the teacher, and ResNet-18 (He *et al.*, 2016) as the student. The student is trained using SGD with momentum 0.9, and a learning rate of 0.1 that is dropped by a factor of 10 every 96,000 gradient steps, for a total of 240,000 steps. The generator is trained using the Adam (Kingma and Ba, 2014) optimizer with a learning rate of 0.015. We perform 1 generator update for each 10 student updates. We set $\gamma = 0{:}4$ in the generator loss (4).

Cite this article as: Jonathan Raiman (2021). Generative adversarial simulator. *International Journal of Artificial Intelligence and Machine Learning*, 1(1), 31-46. doi: 10.51483/IJAIML.1.1.2021.31-46.