Jointly Pre-training with Supervised, Autoencoder, and Value Losses for Deep Reinforcement Learning

Gabriel V. de la Cruz, Jr.
Washington State University
Pullman, Washington
gabriel.delacruz@wsu.edu

Yunshu Du
Washington State University
Pullman, Washington
yunshu.du@wsu.edu

Matthew E. Taylor
Washington State University
Pullman, Washington
matthew.e.taylor@wsu.edu

ABSTRACT

Deep Reinforcement Learning (DRL) algorithms are known to be data inefficient. One reason is that a DRL agent learns both the feature and the policy *tabula rasa*. Integrating prior knowledge into DRL algorithms is one way to improve learning efficiency since it helps to build helpful representations. In this work, we consider incorporating human knowledge to accelerate the asynchronous advantage actor-critic (A3C) algorithm by pre-training a small amount of non-expert human demonstrations. We leverage the supervised autoencoder framework and propose a novel pre-training strategy that jointly trains a weighted supervised classification loss, an unsupervised reconstruction loss, and an expected return loss. The resulting pre-trained model learns more useful features compared to independently training in supervised or unsupervised fashion. Our pre-training method drastically improved the learning performance of the A3C agent in Atari games of Pong and MsPacman, exceeding the performance of the state-of-the-art algorithms at a much smaller number of game interactions. Our method is light-weight and easy to implement in a single machine. For reproducibility, our code is available at github.com/gabrieledcjr/DeepRL/tree/A3C-ALA2019

KEYWORDS

Reinforcement Learning; Deep learning; Learning from Humans

1 INTRODUCTION

Deep Reinforcement Learning (DRL) has been an increasingly popular general machine learning technique, significantly contributing to the resurgence in neural networks research. Not only can DRL allow machine learning algorithms to learn appropriate representations without extensive hand-crafting of input, it can also achieve record-setting performance across multiple types of problems [19, 20, 28, 29]. However, one of the main drawbacks of DRL is its data complexity. Similar to classic RL algorithms, DRL suffers from slow initial learning as it learns *tabular rasa*. While acceptable in simulated environments, the long learning time of DRL has made it impractical for real-world problems where bad initial performance is unaffordable, such as in robotics, self-driving cars, and health care applications [3, 15, 17].

There are two components of learning in DRL: feature learning and policy learning. While DRL is able to directly extract features using a deep neural network as its nonlinear function approximator, this process adds additional training time on top of policy learning and consequently slows down DRL algorithms. In this work, we propose several *pre-training* techniques to tackle the feature learning problem in DRL. We believe that by aiding one of the learning components, a DRL agent will be able to focus more on the policy learning thus improve the overall learning speed.

Many techniques have been proposed to address the data inefficiency of DRL. Transfer learning has been shown to work well for RL problems [31]. The intuition is that knowledge acquired from previously learned *source tasks* can be transferred to related *target tasks* such that the target tasks learn faster since they are not learning from scratch. Learning from demonstrations (LfD) [1, 9, 10, 24] is also an effective way to accelerate learning. In particular, demonstration data of a task can be collected from either a human demonstrator or a pre-trained agent; a new agent can start with mimicking the demonstrator's behavior to obtain a reasonable initial policy quickly, and later on move away from the demonstrator and learns on its own. One can also leverage additional auxiliary losses to gather extra information about a task [7, 11, 18, 27]. For example, an agent can jointly optimize the policy loss and an unsupervised reconstruction loss; doing so explicitly encourages learning the features. In this work, we combine the flavor of the methods above and propose a pre-training strategy to speed up learning. Our method jointly pre-trains a supervised classification loss, an unsupervised reconstruction loss, and a value function loss.

2 PRELIMINARIES

2.1 Deep Reinforcement Learning

We consider a reinforcement learning (RL) problem that is modeled using a Markov Decision Process (MDP), represented by a 5-tuple $\langle S, A, P, R, \gamma \rangle$. A *state* S_t represents the environment at time t. An agent learns what *action* $A_t \in \mathcal{A}(s)$ to take in S_t by interacting with the environment. A *reward* $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ is given based on the action executed and the next state reached, S_{t+1}. The goal is to maximize the expected cumulative return $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where $\gamma \in [0, 1]$ is a discount factor that determines the relative importance of future and immediate rewards [30].

In value-based RL algorithms, an agent learns the state-action value function $Q^\pi(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^\pi(s', a')|s, a]$, and the optimal value function $Q^*(s, a) = max_\pi Q^\pi(s, a)$ gives the expected return for taking an action a at state s and thereafter following an optimal policy. However, directly computing Q values is not feasible when the state space is large. The deep Q-network (DQN) algorithm [20] uses a deep neural network (parameterized as θ) to approximate the Q function as $Q(s, a; \theta) \approx Q^*(s, a)$. At each iteration i, DQN minimizes the loss

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[(y - Q(s, a; \theta_i))^2\right]$$

where $y = r + \gamma max_{a'} Q(s', a'; \theta_i^-)$ is the *target network* (parameterized as θ_i^-) that was generated from previous iterations. The key component that helps to stabilize learning is the *experience replay memory* [16] which stores past experiences. An update is performed by drawing a batch of 32 experiences (minibatch) uniformly random from the replay memory—doing so ensures the $i.i.d.$ property of the sampled data thus stabilizes the learning. *Reward clipping* also helps to make DQN work. All rewards are clipped to $[-1, 1]$ thus avoids the potential instability brought by various reward scales in different environments.

2.2 Asynchronous Advantage Actor-Critic

The asynchronous advantage actor-critic (A3C) algorithm [19] is a policy-based method that combines the actor-critic framework with a deep neural network. A3C learns both a *policy function* $\pi(a_t|s_t;\theta)$ (parameterized as θ) and a *value function* $V(s_t;\theta_v)$ (parameterized as θ_v). The policy function is the *actor* that decides which action to take while the value function is the *critic* that evaluates the quality of the action and also bootstraps learning. The policy loss given by Mnih et al. [19] is

$$\begin{aligned} L_{policy}^{a3c} =& \nabla_\theta log(\pi(a_t|s_t;\theta))\big(Q^{(n)}(s_t, a_t; \theta, \theta_v) - V(s_t;\theta_v)\big) \\ & - \beta^{a3c}\mathcal{H}\nabla_\theta\big(\pi(s_t;\theta)\big) \end{aligned}$$

where $Q^{(n)}(s_t, a_t; \theta, \theta_v) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}; \theta_v)$ is the n-step bootstrapped value that is bounded by a hyperparameter t_{max} ($n \leq t_{max}$). $\mathcal{H}$ is an entropy regularizer for policy π (weighted by β^{a3c}) which helps to prevent premature convergence to sub-optimal policies. The value loss is

$$L_{value}^{a3c} = \nabla_{\theta_v}\Big(\big(Q^{(n)}(s_t, a_t; \theta, \theta_v) - V(s_t;\theta_v)\big)^2\Big)$$

The A3C loss is then

$$L^{a3c} = L_{policy}^{a3c} + \alpha L_{value}^{a3c} \tag{1}$$

where α is a weight for the value loss. A3C runs k actor-learners in parallel and each with their own copies of the environment and parameters. An update is performed using data collected from all actors. In this work, we use the feed-forward version of A3C [19] for all experiments. The architecture consists of three convolutional layers, one fully connected layer (*fc1*), followed by two branches of a fully connected layer: a policy function output layer (*fc2*) and a value function output layer (*fc3*).

2.3 Transformed Bellman Operator for A3C

While the reward clipping technique helped to reduce the variance and stabilize learning in DQN, Hester et al. [10] found that clipping all rewards to $[1, -1]$ hurts the performance in games where the reward has various scales. For example, in the game of MsPacman, a single dot is worth 10 points, while a cherry bonus is worth 100 points; when both are clipped to 1, the agent becomes incapable of distinguishing between small and large rewards, resulting in reduced performance. Pohlen et al. [24] proposed the *transformed Bellman operator* to overcome this problem in DQN. Instead of changing the magnitude of rewards, Pohlen et al. [24] considers reducing the scale of the action-value function, which enables DQN to use raw rewards instead of clipped ones. In particular, a transform function

$$h : z \mapsto sign(z)\left(\sqrt{|z|+1} - 1\right) + \varepsilon z \tag{2}$$

is applied to reduce the scale of $Q^{(n)}(s_t, a_t; \theta, \theta_v)$ and Q is transformed as

$$Q_{TB}^{(n)}(s_t, a_t; \theta, \theta_v) = \sum_{k=0}^{n-1} h\left(\gamma^k r_{t+k} + \gamma^n h^{-1}\left(V\left(s_{t+n}; \theta_v\right)\right)\right) \tag{3}$$

In this work, we apply the transformed Bellman operator in the A3C algorithm and use the raw reward value (instead of clipped) to perform updates. We denote this method as *A3CTB*.

2.4 Self-Imitation Learning

The *self-imitation learning* (SIL) algorithm aims to encourage the agent to learn from its own past good experiences [21]. Built on the actor-critic framework [19], SIL adds a replay buffer $\mathcal{D} = (s_t, a_t, G_t)$ to store the agent's past experiences. The authors propose the following off-policy actor-critic loss

$$\begin{aligned} L_{policy}^{sil} &= -log(\pi(a_t|s_t;\theta))\big(G_t - V(s_t;\theta_v)\big)_+ \\ L_{value}^{sil} &= \frac{1}{2}||\big(G_t - V(s_t;\theta_v)\big)_+||^2 \end{aligned}$$

where $(\cdot)_+ = max(\cdot, 0)$ and $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is the discounted sum of rewards. The SIL loss is then

$$L^{sil} = L_{policy}^{sil} + \beta^{sil} L_{value}^{sil} \tag{4}$$

In this work, we leverage this framework and incorporate SIL in *A3CTB* (see Section 3.1).

2.5 Supervised Pre-training

In our previous work, supervised pre-training consists of a two-stage learning hierarchy [6]: 1) pre-training on human demonstration data, and 2) initializing a DRL agent's network with the pre-trained network $\theta \leftarrow \theta_s$. It uses non-expert human demonstrations as its training data where the game states are the neural network's inputs and assume the non-optimal human actions as the true labels for each game state. The network is pre-trained with the cross-entropy loss

$$L_s = -\sum_{\forall x} p(x) log\big(q(x;\theta_s)\big)$$

where x is the image game state s and $p(x)$ is the distribution over discrete variable x represented here as a one-hot vector of the human action a; while $q(x;\theta_s)$ is the output distribution of the supervised learning neural network with the weights θ_s.

2.6 Supervised Autoencoder

An autoencoder learns to reconstruct its inputs and has been used for unsupervised learning of features. A supervised autoencoder (SAE) is an autoencoder with an additional supervised loss that can better extract representations that are tailored to the class labels. For example in Le et al. [14], the authors consider a supervised learning setting where the goal is to learn a mapping between some inputs $X \in \mathbb{R}$ and some targets $Y \in \mathbb{R}$. Instead of learning with only

a supervised loss, an auxiliary reconstruction loss is integrated and the following SAE objective is proposed:

$$L^{sae} = L_s^{sae}(W_s F x_i, y_i) + L_{ae}^{sae}(W_{ae} F x_i, x_i) \quad (5)$$

where F is the weights of a neural network; W_s and W_{ae} are the weights for the supervised output layer and the autoencoder output layer respectively. Here, L_s^{sae} and L_{ae}^{sae} can be any loss functions (e.g., MSE).

Existing work have considered training using an SAE loss from scratch. Our method is different in that i) we consider the SAE loss as a pre-training method instead of training from scratch and ii) we jointly pre-train a supervised loss and a reconstruction loss and then use the learned parameters as initialization (i.e., the two-stage hierarchy described in Section 2.5). In this work, we explore if incorporating an unsupervised loss in supervised pre-training can further boost the learning of an agent.

3 METHODOLOGIES

This section describes our proposed algorithms. First, we show that by incorporating the SIL framework (see Section 2.4), we can further improve the performance of the original A3C algorithm [19] and the *A3CTB* variant from our previous work [6]; we term this new method as *A3CTB+SIL*. Then, we introduce our proposed pre-training methods and show that, after pre-training, *A3CTB+SIL* can achieve superior results; its performance on MsPacman exceeds or is comparable to some state-of-the-art algorithms that use human demonstrations (e.g., Hester et al. [10], Oh et al. [21]), and is also much lower on computational demands.

3.1 A3C with Self-Imitation Learning

We incorporate the self-imitation learning (SIL) framework (see Section 2.4) in *A3CTB* with the following modifications. To enable using raw rewards (as was done in *A3CTB*), we apply the transformation function h (Equation (2)) to the returns as $G_t = h\big(r_{t+1} + \gamma h^{-1}(G_{t+1})\big)$. We also add a SIL-learner in parallel with the k actor-learners in A3C (i.e., there are a total of $k+1$ parallel threads). The SIL-learner does not have its own copy of the environment; it learns by optimizing L^{sil} using minibatch sampling from $\mathcal{D}$. The SIL-learner acts similarly to the other actor-learners as it updates the global network asynchronously. Each actor-learner contributes to $\mathcal{D}$ through a shared episode queue $Q_{\mathcal{E}}$, where $\mathcal{E}$ is an episode buffer for each actor-learner that stores observation at time t as $\{s_t, a_t, r_t\}$, until a terminal state is reached (i.e., the end of an episode). At a terminal state, the actor-learner computes the returns, G_t, with the transformation, h, for each step in the episode. Then $\mathcal{E}$ with the computed transformed returns are added to the shared episodes queue $Q_{\mathcal{E}}$. The pseudocode for the SIL-learner is shown in Algorithm 1. We denote this method as *A3CTB+SIL*.

3.2 Pre-training Methods

We now introduce our pre-training methods. The same set of non-expert human demonstration data collected from de la Cruz et al. [6] is used for all pre-training (see Table 1). This work deviates from our previous work in two aspects: 1) we integrate multiple losses for pre-training while the previous work only considered supervised pre-training, and 2) we train using *A3CTB+SIL* while the previous work train using *A3CTB*; we shall see that integrating SIL is beneficial for our pre-training approach and will discuss the reasons later in this section.

Algorithm 1 SIL-learner in A3CTB+SIL.

// Assume global shared parameter vector θ and θ_v and global shared counter $T = 0$
// Assume global shared episodes queue $Q_{\mathcal{E}}$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize replay memory $\mathcal{D} = \emptyset$
repeat
  Synchronize parameters $\theta' \leftarrow \theta$ and $\theta'_v \leftarrow \theta_v$
  for $m \leftarrow 1$ **to** M **do**
    Sample a minibatch $\{s_j, a_j, G_j\}$ from $\mathcal{D}$
    Compute gradients w.r.t.
    $\theta' : d\theta \leftarrow \nabla_{\theta'} \log\pi(a_j|s_j;\theta')(G_j - V(s_j;\theta'_v))_+$
    Compute gradients w.r.t.
    $\theta'_v : d\theta_v \leftarrow \partial((G_j - V(s_j;\theta'_v))_+)^2/\partial\theta'_v$
    Perform asynchronous update of θ using $d\theta$ and θ_v using $d\theta_v$
  end for
  while `len(`$Q_{\mathcal{E}}$`) > 0` **do**
    Dequeue first episode $\mathcal{E}$ from $Q_{\mathcal{E}}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{S_t, A_t, G_t\}$ for all t in $\mathcal{E}$
  end while
until $T > T_{max}$

From our previous work, supervised pre-training on demonstrations can accelerate learning in A3C and *A3CTB* [6]; for brevity, we denote this method as *[SL]*[1] pre-training. However, there were two potential problems with *[SL]* pre-training: 1) the value output layer (fc3) of A3C is not pre-trained, and 2) training only to minimize the loss between human and agent actions is sub-optimal since the action demonstrated by the human could be noisy.

To tackle the first problem, we pre-train an aggregated loss that consists of the supervised and the value return losses. For the supervised component, we use the SIL policy loss L_{policy}^{sil} which can be interpreted as the cross-entropy loss $-log(\pi(a_t|s_t;\theta))$ weighted by $(G_t - V(s_t;\theta_v))_+$ [21]. The $(\cdot)_+$ operator encourages the agent to imitate past decisions only when the returns are larger than the value. The value return loss is nearly identical to the value loss L_{value}^{sil} in SIL, but without the $(\cdot)_+$ operator. Note that this is also similar to A3C's value loss L_{value}^{a3c}, but instead of the n-step bootstrapped value $Q^{(n)}$, we use the discounted returns G_t, which can be easily computed from the human demonstration data since it contains the full trajectory of each episode. We denote this method as *[SL+V]* pre-training.

de la Cruz et al. [6] also revealed that supervised pre-training learns features that are geared more towards the supervised loss. For example in Pong, the area around the paddle is an important region since the paddle movements are associated with human actions. This implies the second problem mentioned above; if the features learned to focus on human actions only, they might not generalize well to new trajectories. To obtain extra information in

[1] We denote all pre-training methods with their names in brackets.

Table 1: Human demonstration size and quality, collected from de la Cruz et al. [6].

Game	Worst score	Best score	# of states	# of episodes
MsPacman	4020	18241	14504	8
Pong	-13	5	21674	6

addition to the supervised features, we take inspiration from the supervised autoencoder framework which jointly trains a classifier and an autoencoder [14]; we believe this approach will retain the important features learned through supervised pre-training and at the same time, learns additional general features from the added autoencoder loss. Finally, we blend in the value loss L_v^{saev} with the supervised and autoencoder losses as

$$L^{saev} = L_s^{saev}(W_s F x_i, y_i) + L_{ae}^{saev}(W_{ae} F x_i, x_i) + L_v^{saev}(W_v F x_i, x_i). \quad (6)$$

We denote this method as *[SL+V+AE]* pre-training. The network architecture of this pre-training method is shown in Figure 1. In this network, Tensorflow's *SAME* padding option is used to ensure that the input size is the same as the output size which is inherently necessary for reconstructing the input. This change results in a final output of 88x88 of the neural network due to the existing network architecture filters used. Thus, instead of downsizing the output from 88x88 to 84x84 (which is the original input size of A3C), we changed the input size to 88x88 in the spirit of the work of Kimura [13] which uses autoencoder for pre-training.

There are two strong motivations to use self-imitation learning in A3C when using pre-training. First, human demonstration data can be loaded into SIL's memory to jumpstart the memory and continue learning with the data. Second, the motivation of jointly pre-train with multiple losses, especially with a value loss, is not only to learn better features but also to use the pre-trained policy and value layers into the A3C network. Adding the value loss allows pre-training the entire network. In turn, since SIL self-imitates its own experience, data generated during early stages are potentially closely related to the policy and value learned from pre-training; the learning speed could be increased more at the early stage of training. By using SIL, pre-training addresses feature learning while implicitly addressing policy learning.

4 EXPERIMENTS AND RESULTS

We then present our experiments. First, we show that *A3CTB+SIL* exceeds the performance of A3C and *A3CTB*. As a comparison, we also evaluate *A3C+SIL* since Oh et al. [21] implemented SIL in the *synchronous* version of A3C (i.e., A2C) [19], which is different from our implementation that the SIL-learner is *asynchronous*. Then, we present our experiments and results for the pre-training approaches (all trained in *A3CTB+SIL* after pre-training) and show that they all outperform the baseline A3C algorithm. We use the same set of parameters across all experiments, shown in Table 2.

4.1 A3CTB+SIL

Figure 2 shows the performance of *A3CTB+SIL* when compared to the baseline A3C [19], *A3CTB* [6], and *A3C+SIL*. *A3C+SIL* helps in both games and shows better improvement in MsPacman. This is consistent with the findings in Oh et al. [21] that imitating past good experiences encourages exploration, which is beneficial for hard exploration games. Our proposed method *A3CTB+SIL* shows the best performance among all. The largely improved score in MsPacman indicates that it is important for the agent to be able to distinguish big and small rewards (the function of TB); SIL helps to imitate past experiences with large returns.

Table 2: All games use the same set of hyperparameters except for Pong, where we found setting RMSProp epsilon to 1×10^{-4} gives a much more stable learning.

Common Parameters	**Value**
Input size	88×88×4
Padding method	SAME
Parameters unique to pre-training	
Adam learning rate	5×10^{-4}
Adam epsilon	1×10^{-5}
Adam β_1	0.9
Adam β_2	0.999
L2 regularization weight	1×10^{-5}
Number of minibatch updates	50,000
Batch size	32
Parameters unique to A3C	
RMSProp learning rate	7×10^{-4}
RMSProp epsilon	1×10^{-5}
RMSProp decay	0.99
RMSProp momentum	0
Maximum gradient norm	0.5
k parallel actors	16
t_{max}	20
transformed Bellman operator ε	10^{-2}
Parameters unique to SIL	
M	4
β^{sil}	0.5
replay buffer $\mathcal{D}$ size	10^6

4.2 A3CTB+SIL with Pre-training

Since *A3CTB+SIL* has shown the largest improvement without pre-training, from now on, we investigate if using our new pre-training approaches can further accelerate *A3CTB+SIL*. That is, after pre-training, all agents are then trained in *A3CTB+SIL*. Figure 3 shows the results for pre-training methods. In the game of MsPacman, while *[SL]* already sees slight improvements over the baseline, both *[SL+V]* and *[SL+V+AE]* show superior improvements over *[SL]*, achieving a testing reward (averaged over three trials) at around 8,000. Compared with some state-of-the-art results such as Hester et al. [10] and Oh et al. [21] where the final rewards for MsPacman are roughly around 5,000 and 4,000 respectively,[2], our method largely exceeded theirs.

[2]Numbers approximately read from the figures of the mentioned papers.

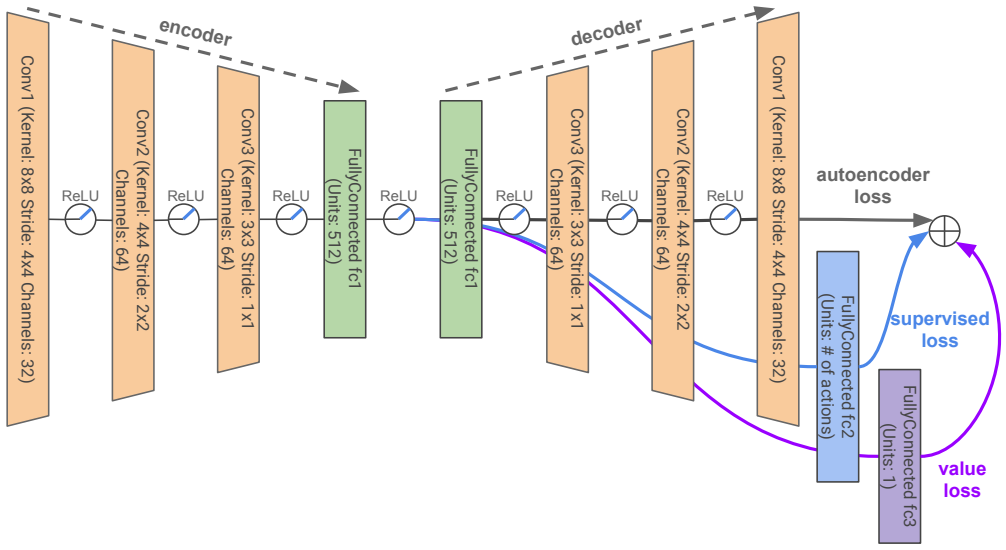

Figure 1: Network architecture on pre-training with a combined loss of supervised, value and autoencoder losses.

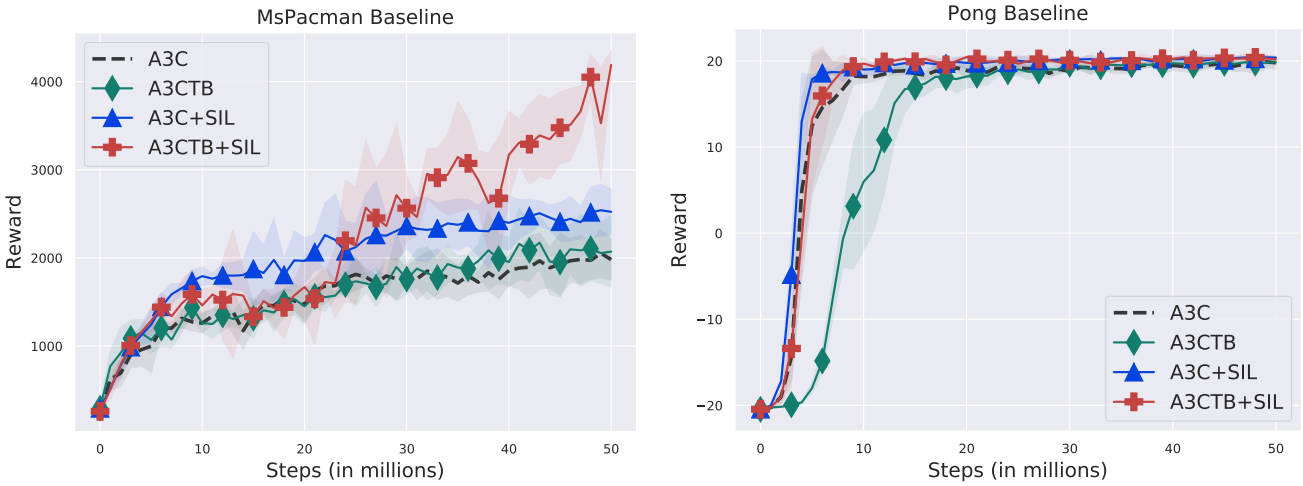

Figure 2: Baseline performance of MsPacman and Pong without pre-training. The x-axis is the total number of training steps (16 actors for methods without SIL; 16 actors plus 1 SIL-learner for methods using SIL). Each step consists of four game frames (frame skip of four). The y-axis is the average testing score over three trials; shaded regions are the standard deviation.

In the game of Pong, all pre-training methods exceed the baseline performance but the amount of improvements are not as large as in MsPacman. One reason could be that Pong is a relatively easy game to play in Atari and the agent is able to find a good policy even when learning from scratch. In addition, note that *[SL]* actually has the fastest learning speed among other pre-training methods, which is intuitively reasonable. Catching the ball is probably the most important behavior to learn in Pong and this movement is highly associated with the classification of actions; learning the value function and the feature representations did not seem to add additional benefits than learning just an action classifier.

4.2.1 Ablation Study. We want to see how useful the feature representations learned during the pre-training stage are to a DRL agent. It is known that general features are retained in the hidden layers of a neural network while the output layer is more task-specific [33]. Therefore, in this set of experiments we exclude all output fully connected layers (fc2 and fc3) and only initialize a new A3C network with pre-trained convolutional layers and the fc1 layer, then train it in *A3CTB+SIL*. This experiment will allow us to investigate how important is the general feature learning as to the task-specific policy learning for a DRL agent.

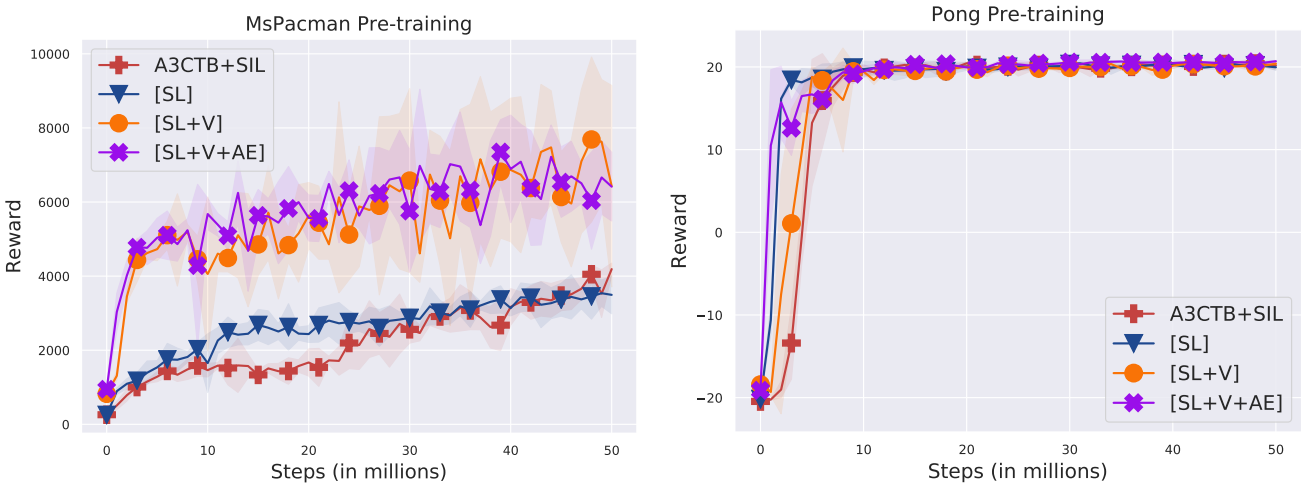

Figure 3: Pre-training performance of MsPacman and Pong. All layers are transferred. Pre-training methods are shown in brackets. After pre-training, all agents are trained in *A3CTB+SIL*. The x-axis is the total number of training steps (16 actors for methods without SIL; 16 actors plus 1 SIL-learner for methods using SIL). Each step consists of four game frames (frame skip of four). The y-axis is the average testing score over three trials; shaded regions are the standard deviation.

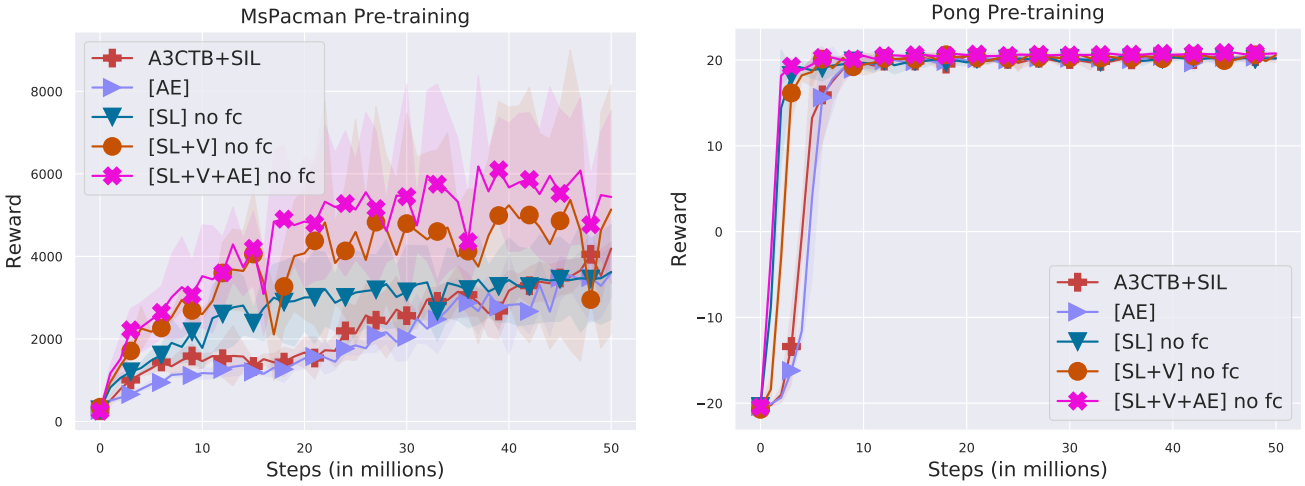

Figure 4: Pre-training performance of MsPacman and Pong. Transfer without fully connected output layers (fc2 and fc3). Pre-training methods are shown in brackets. After pre-training, all agents are trained in *A3CTB+SIL*. The x-axis is the total number of training steps (16 actors for methods without SIL; 16 actors plus 1 SIL-learner for methods using SIL). Each step consists of four game frames (frame skip of four). The y-axis is the average testing score over three trials; shaded regions are the standard deviation.

Figure 4 shows the results on transferring pre-trained parameters without the fully connected out layers ("no fc" refers to "no fc2 and no fc3"). Note that the *[AE]* pre-training refers to pre-training with only the reconstruction loss L_{ae}^{saev} (see Equation (6)); since the autoencoder model trains on fc1 and does not affect fc2 and fc3, we consider it as "transfer without fc2 and fc3" and present its results here instead of in previous experiments where "all layers are transferred." It is interesting to observe that, in MsPacman, the performance of *[SL+V] no fc* and *[SL+V+AE] no fc* dropped relatively compared to when transferring all layers. While in Pong, not transferring the output layers did not affect the performance as much. This indicates again that, due to the nature of the game MsPacman, more exploration is needed and only having a good initial feature representation is not as good as when knowing some priors about both the features and the behaviors. However, in games that require less strategy learning and exploration, having a good initial feature

of the environment can already provide a performance boost. For example in Pong, even when not transferring the output layers, the information retained in the hidden layers are still highly related to the paddle movement (since it classifies actions), which could be the most important thing to learn in Pong.

The lower performance of not transferring the output layers for MsPacman also shows the benefits of pre-training both the policy layer and the value layer. As shown in Figure 3 for MsPacman, when using all layers of the pre-trained network, it has a higher initial testing reward compared to the baseline. This indicates that the initial policy was better after pre-training both policy and value layers. We believe that this could be a way of identifying when to use the full pre-trained network and when to exclude the output layers. Future work should study if the following hypotheses hold true:

(1) If the initial performance of the pre-trained network is better than the baseline, then one should use the full pre-trained network.
(2) Otherwise, it might be better to use the pre-trained network without the output layers.

5 RELATED WORK

Our work is largely inspired by the literature of transfer for supervised learning [22], particularly in deep supervised learning where a model is rarely trained from scratch. Instead, parameters are usually initialized from a larger pre-trained model (e.g., ImageNet [25]) and then trained in the target dataset. Yosinski et al. [33] performed a thorough study on how transferable the learned features at each layer are in a deep convolutional neural network, showing that a pre-trained ImageNet classification model is able to capture general features that are transferable to a new classification task. Similarly, unsupervised pre-training a neural network can extract key variations of the input data, which in term helps the supervised fine-tuning stage that follows to converge faster [2, 8]. In this work, we combine the supervised and unsupervised methods into the supervised autoencoder framework [14] as our pre-training stage. Intuitively, the supervised component could guide the autoencoder to learn features that are more specific to the task.

It has been shown that incorporating useful prior information can benefit the policy learning of an RL agent [26]. Learning from Demonstrations (LfD) integrates such a prior by leveraging available expert demonstrations; the demonstration can either be generated by another agent or be collected from human demonstrators [1]. Some previous work seeks to learn a good initial policy from the demonstration then later on combine with the agent-generated data to train in RL [4, 12, 23]. More recent approaches have considered LfD in the context of DRL. Christiano et al. [5] proposes to learn the reward function from human feedback. Gao et al. [9] considers the scenario when demonstrations are imperfect and proposes a normalized actor-critic algorithm. Perhaps the closest work to ours is the deep Q-learning from demonstration (DQfD) algorithm [10]. In DQfD, the agent is first pre-trained over the human demonstration data using a combined supervised and temporal difference losses. However, during the DQN training stage, the agent continues to jointly minimize the temporal difference loss with a large margin supervised loss when sampling from expert demonstration data. Our work instead uses a supervised autoencoder as pre-training, which explicitly emphasizes the representation learning and our pre-training losses are not carried through in the RL training. DQfD was further improved as Ape-X DQfD by Pohlen et al. [24] where the transformed Bellman operator was applied to reduce the variance of the action-value function and is applied to a large-scale distributed DQN. We empirically observe that our pre-training approaches obtained a higher score in MsPacman than that of DQfD and are comparable to Ape-X DQfD (see Section 4.2). However, we are unable to compare directly as we do not have the computational resources to run such a large scale experiment. In addition, note that Ape-X DQfD does not have a pre-training stage; DQfD addresses both feature and policy learning while our work only addresses feature learning. Therefore, our method is not directly comparable to the above.

The use of unsupervised auxiliary losses has been explored in both deep supervised learning and DRL. For example, Zhang et al. [34] uses unsupervised reconstruction losses to aid in learning large-scale classification tasks; Jaderberg et al. [11] combines additional control tasks that predict feature changes as auxiliaries. Our methods of pre-training via supervised autoencoder can be viewed as leveraging the reconstruction loss as an auxiliary task, which guides the agent to learn desirable features for the given task.

6 DISCUSSION AND CONCLUSION

In this work, we studied several pre-training methods and showed that the A3C algorithm could be sped up by pre-training on a small set of non-expert human demonstration data. In particular, we proposed to integrate rewards in supervised policy pre-training, which helps to quantify how good a demonstrated action was. The component of the value function and autoencoder pre-training yielded the most significant performance improvements and exceeded the state-of-the-art results in the game of MsPacman. Our approach is light-weight and easy to implement in a single machine.

While pre-training works well in the two games presented in this paper, there is a need to perform this experiment in more games to show the generality of our method. Looking into what features are learned during pre-training is also interesting to study. de la Cruz et al. [6] visualized the feature learned in supervised pre-training and a final DRL model and found that they share some common patterns, indicating why pre-training is useful. In future work, we are interested in studying how the learned feature pattern differs from each pre-training method. Lastly, pre-training methods only address the problem of feature learning in DRL but do not aid policy learning. To further accelerate learning, we plan on looking into how could policy learning be improved using human demonstration data. Some existing work like DQfD integrate the supervised loss not only during pre-training but also during training the DRL agent [10]; others leverage human demonstrations as advice and constantly providing suggestions during policy learning [32]. We attribute these as our future directions.

ACKNOWLEDGMENTS

The A3C implementation was a modification of https://github.com/miyosuda/async_deep_reinforce. The authors thank NVidia for donating a graphics card used in these experiments. This research

used resources of Kamiak, Washington State University's high performance computing cluster, where we ran all our experiments.

REFERENCES

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 5 (2009), 469–483.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[3] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. 2017. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911* (2017).

[4] Jessica Chemali and Alessandro Lazaric. 2015. Direct policy iteration with demonstrations. In *IJCAI*.

[5] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *NIPS*.

[6] Gabriel V. de la Cruz, Jr., Yunshu Du, and Matthew E Taylor. 2018. Pre-training with Non-expert Human Demonstration for Deep Reinforcement Learning. *arXiv preprint arXiv:1812.08904* (2018).

[7] Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. 2018. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224* (2018).

[8] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* 11, Feb (2010), 625–660.

[9] Yang Gao, Ji Lin, Fisher Yu, Sergey Levine, Trevor Darrell, et al. 2018. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313* (2018).

[10] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep Q-learning from demonstrations. In *AAAI*.

[11] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2017. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*.

[12] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. 2013. Learning from limited demonstrations. In *Advances in Neural Information Processing Systems*. 2859–2867.

[13] Daiki Kimura. 2018. DAQN: Deep auto-encoder and Q-network. *arXiv preprint arXiv:1806.00630* (2018).

[14] Lei Le, Andrew Patterson, and Martha White. 2018. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In *Advances in Neural Information Processing Systems*. 107–117.

[15] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[16] Long-Ji Lin. 1992. *Reinforcement learning for robots using neural networks*. Ph.D. Dissertation.

[17] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2017. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* (2017).

[18] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. 2016. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673* (2016).

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*. 1928–1937.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[21] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. In *ICML*.

[22] Sinno Jialin Pan, Qiang Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[23] Bilal Piot, Matthieu Geist, and Olivier Pietquin. 2014. Boosted Bellman residual minimization handling expert demonstrations. In *ECML/PKDD*.

[24] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Večerík, et al. 2018. Observe and look further: Achieving consistent performance on Atari. *arXiv preprint arXiv:1805.11593* (2018).

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[26] Stefan Schaal. 1997. Learning from demonstration. In *Advances in neural information processing systems*. 1040–1046.

[27] Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. 2018. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835* (2018).

[28] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[29] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[30] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[31] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[32] Zhaodong Wang and Matthew E Taylor. 2017. Improving reinforcement learning with confidence-based demonstrations. In *Proceedings of the 26th International Conference on Artificial Intelligence (IJCAI)*.

[33] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.

[34] Yuting Zhang, Kibok Lee, and Honglak Lee. 2016. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICML*. 612–621.