The Knowledge Engineering Review, Vol. 34, e10, 1 of 15. © Cambridge University Press, 2019
doi:10.1017/S0269888919000055

Pre-training with non-expert human demonstration for deep reinforcement learning

GABRIEL V. DE LA CRUZ JR, YUNSHU DU and MATTHEW E. TAYLOR

School of Electrical Engineering and Computer Science, Washington State University, Pullman, Washington 99164-2752, USA
e-mails: gabriel.delacruz@wsu.edu, yunshu.du@wsu.edu, matthew.e.taylor@wsu.edu

Abstract

Deep reinforcement learning (deep RL) has achieved superior performance in complex sequential tasks by using deep neural networks as function approximators to learn directly from raw input images. However, learning directly from raw images is data inefficient. The agent must learn feature representation of complex states in addition to learning a policy. As a result, deep RL typically suffers from slow learning speeds and often requires a prohibitively large amount of training time and data to reach reasonable performance, making it inapplicable to real-world settings where data are expensive. In this work, we improve data efficiency in deep RL by addressing one of the two learning goals, feature learning. We leverage supervised learning to pre-train on a small set of non-expert human demonstrations and empirically evaluate our approach using the asynchronous advantage actor-critic algorithms in the Atari domain. Our results show significant improvements in learning speed, even when the provided demonstration is noisy and of low quality.

1 Introduction

The widespread successes of deep reinforcement learning (deep RL) have brought a resurgence in using deep neural networks for RL tasks. Using a deep neural network as its function approximator, deep RL can learn state representations directly from raw input pixels and has achieved state-of-the-art results in various domains (Mnih *et al.*, 2015; Silver *et al.*, 2016; Kempka *et al.*, 2016; Lillicrap *et al.*, 2016; Duan *et al.*, 2016; Silver *et al.*, 2018). However, despite this impressive ability, deep RL remains data inefficient and often requires a long training time to achieve reasonable performance. This drawback has made deep RL impractical in real-world applications where data are expensive to collect, such as in robotics, self-driving cars, finance, or medical applications (Bojarski *et al.*, 2017; Deng *et al.*, 2017; Miotto *et al.*, 2017).

Similar to classic RL algorithms, deep RL suffers from poor initial performance since it learns *tabula rasa* (Sutton & Barto, 2018). Inherently, deep RL takes even longer to learn because, unlike in classic RL where hand-engineered features were used, deep RL has to learn features directly from raw observations, in addition to policy learning. Therefore, one can speed up deep RL by addressing its two learning components: feature learning and policy learning. In this work, we tackle the feature learning problem and show that faster learning can be achieved by addressing one of the two problems.

There have been many techniques proposed to speed up feature learning in deep RL, from transfer learning (Taylor & Stone, 2009; Pan *et al.*, 2010), to reward shaping (Ng *et al.*, 1999; Brys *et al.*, 2015), to using auxiliary tasks (Zhang *et al.*, 2016; Jaderberg *et al.*, 2017; Mirowski *et al.*, 2017; Papoudakis *et al.*, 2018; Du *et al.*, 2018). Learning from demonstration (LfD) (Argall *et al.*, 2009) is yet another way

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

to help speed up learning in deep RL and has recently gained traction due to its ability to bootstrap the agent at the beginning of training (Kurin *et al.*, 2017; Vinyals *et al.*, 2017; Hester *et al.*, 2018; Pohlen *et al.*, 2018). In this work, we make use of human demonstration data by first using supervised learning to pre-train a neural network to learn the underlying state features and then transfer the learned features to an RL agent (Erhan *et al.*, 2009, 2010; Yosinski *et al.*, 2014). We evaluate our approach using a recently developed deep RL algorithm, the asynchronous advantage actor-critic (A3C) (Mnih *et al.*, 2016) algorithm in six Atari games (Bellemare *et al.*, 2013). Unlike previous work where a large amount of expert human data is required to achieve good initial performance boost, our approach shows significant learning speed improvements on all experiments with only a relatively small amount of noisy, non-expert demonstration data. The simplicity of our approach has made it generally adaptable to other deep RL algorithms and potentially to other domains since the collection of demonstration data becomes easy. In addition, we apply Gradient-weighted Class Activation Mapping (Grad-CAM) (Selvaraju *et al.*, 2017) on learned feature maps for both the human data and the agent data, providing a detailed analysis on why pre-training helps to speed up learning. Our work makes the following contributions:

1. We show that pre-training on a small amount of non-expert human demonstration data is sufficient to achieve significant performance improvements.
2. We are the first to apply the transformed Bellman (TB) operator (Pohlen *et al.*, 2018) in the A3C algorithm (Mnih *et al.*, 2016) and further improve A3C's performance on both baseline and pre-training methods.
3. We propose a modified version of the Grad-CAM method (Selvaraju *et al.*, 2017), which we are the first to provide empirical analysis on what features are learned from pre-training, indicating why pre-training on human demonstration data helps.
4. We release our code and all collected human demonstration data at https://github.com/gabrieledcjr/DeepRL.

This article is organized as the following. In the next section, we review some of the related work in using pre-training to improve data efficiency. Section 3 provides background on deep RL algorithms and the TB operator. In Section 4, we propose our pre-training methods for deep RL, followed by Section 5, where we describe the experimental designs. Results and analysis are presented in Section 6. We conclude this article in Section 7 with discussions and future works.

2 Related work

Our work is closely related to transfer learning (Taylor & Stone, 2009; Pan *et al.*, 2010) where learned knowledge from source task(s) is transferred to target task(s) such that the target task does not need to learn features and/or policies from scratch, thus obtaining faster learning. In supervised learning, transferring parameters from a model pre-trained on ImageNet (Russakovsky *et al.*, 2015) has shown to be an effective way of speeding up image classification in a new data set, especially when the source data set is similar to the target data set (Yosinski *et al.*, 2014). In deep RL, the performance of a target agent can be improved by making use of the knowledge learned in one or more similar source agents (Du *et al.*, 2016; Glatt *et al.*, 2016; Parisotto *et al.*, 2016; Rusu *et al.*, 2016; Teh *et al.*, 2017). All works mentioned above perform pre-training and transfer under the same problem settings, that is, pre-training in supervised learning and transfer to supervised learning, or pre-training in RL and transfer to RL. In this work, we consider a different mechanism that transfers between different problem settings. We pre-train a supervised classification model on the human demonstration data, and then transfer the learned features to an RL agent.

Existing work has shown that pre-training can effectively speed up learning in RL. For example, Abtahi and Fasel (2011) considered learning latent features with unsupervised learning through Deep Belief Networks as a pre-training method; Anderson *et al.* (2015) pre-train hidden units of a Q-network by learning to predict state dynamics. These works show learning speed up in relatively easy RL domains

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

such as Mountain Car, Puddle World, and Cart-Pole. Our approach is in spirit to these earlier works and differs in that we consider a much complex domain, Atari, and learn features directly from raw input images instead of using hand-engineered features in a simpler domain. Learning directly from raw inputs is extremely challenging to an RL agent as it has to learn both the feature representations and the policy simultaneously.

Leveraging human knowledge via LfD is another effective way to pre-train an RL agent and has shown to be successful in robotics problems (Argall *et al.*, 2009). LfD has recently been applied widely in deep RL. Christiano *et al.* (2017) use human feedback to learn a reward function; Hester *et al.* (2018) pre-train a deep Q-network (DQN) (Mnih *et al.*, 2015) with human demonstrations by combining a large margin supervised loss with temporal difference loss, such that the agent closely imitates the demonstrator's policy at the beginning and later on learn to surpass the demonstrator. Our work similarly leverages a supervised loss as pre-training but differs in that we consider only the cross-entropy-supervised loss as a feature learner, but not to imitate the policy. Pohlen *et al.* (2018) builds upon Hester *et al.* (2018) and proposes a reward-invariant update rule that uses the TB operator in the DQN algorithm which better leverages expert demonstrations. In our work, we instead apply the TB operator in the A3C algorithm. The work of Silver *et al.* (2016) also trains human demonstrations in supervised learning and then uses the supervised learner's network to initialize RL's policy network. However, their work uses a vast amount of expert demonstration data to train the supervised learner, while ours only uses a small amount of non-expert data. Our work is also the first to provide a comparative analysis on how pre-training with human data impacts learning in deep RL algorithms, as well as how our approach complements existing deep RL algorithms when (a small amount of) human demonstrations are available.

3 Background: deep reinforcement learning

An RL problem is typically modeled using a Markov Decision Process that is represented by a five-tuple $\langle S, A, P, R, \gamma \rangle$. At each time step t, an RL agent receives some *state* representation $S_t \in \mathcal{S}$ and explores an unknown environment by taking an *action* $A_t \in \mathcal{A}(s)$. A *reward* $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ is given based on the action the agent took and the next state S_{t+1} it reaches. The goal of an RL agent is to learn to maximize the expected return value $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ for each state at time t. The discount factor $\gamma \in [0, 1]$ determines the relative importance of future and immediate rewards (Sutton & Barto, 2018).

The first successful deep RL method, DQN, learns to play 49 Atari games directly from screen pixels by combining Q-learning with a deep convolutional neural network (Mnih *et al.*, 2015). In classic Q-learning, an agent learns a state-action value function $Q^\pi(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^\pi(s', a')|s, a]$, which is the expected discounted reward determined by performing action a in state s and thereafter following policy π (Watkins & Dayan, 1992). The optimal Q^* can be deduced by following actions that have the maximum Q value, $Q^*(s, a) = \text{argmax}_\pi Q^\pi(s, a)$. Directly computing the Q value is not feasible when the state space is large or continuous. The DQN algorithm uses a convolutional neural network as a function approximator to estimate $Q(s, a; \theta) \approx Q^*(s, a)$, where θ is the network's weight parameter. For each iteration i, DQN is trained to minimize

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

where $y = r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ is a *target network* parameterized as θ_i^- that was generated from previous iterations. $\{s, a, r, s'\}$ are state-action samples drawn from an *experience replay memory*, which is used to store the agent's experiences. At each time step, a batch of 32 samples (or *minibatch*) is drawn from the experience replay memory to perform an update—this off-policy method could break the correlation between data (i.e., the sampled data are *i.i.d.*), which stabilizes the learning. All rewards are clipped to $[-1, 1]$ to cope with different reward scales in games. The use of a target network, an experience replay memory, and the reward clipping are essential to stabilizing learning. The ϵ-greedy policy is used by the agent to obtain sufficient exploration of the state space; for a probability of ϵ, the agent selects a random action to explore.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

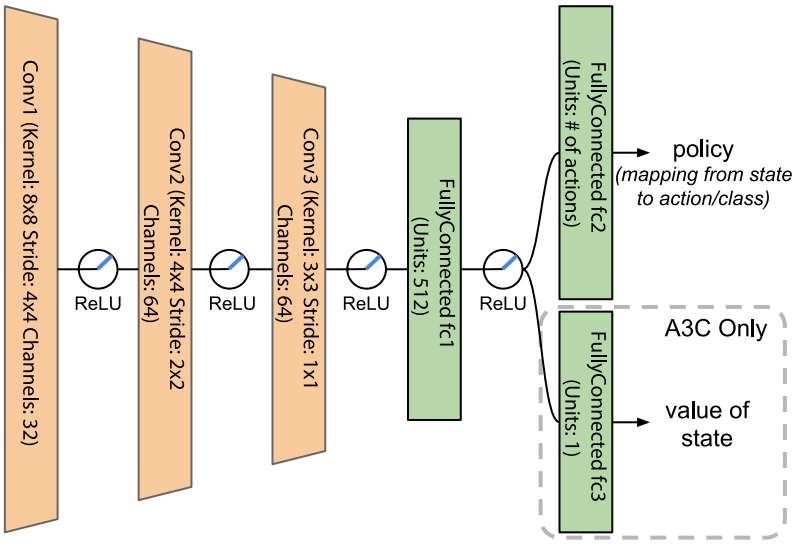


Figure 1 Network architecture for each parallel actor in the A3C agent. We follow the same architecture as in Mnih *et al.* (2015), where there are three convolutional layers (*conv1, conv2*, and *conv3*), followed with two fully connected layers (*fc1* and *fc2*), and a third fully connected layer (*fc3*) for learning the state value function as was done in Mnih *et al.* (2016)

3.1 Asynchronous advantage actor-critic

The DQN algorithm suffers from two drawbacks: long training times and high computational requirements for memory and GPU resources. The A3C algorithm, in contrast, trains faster without the need of a GPU. In this work, we choose to use the A3C algorithm for all experiments.

A3C combines the actor-critic algorithm with deep RL. It differs from value-based algorithms where only a value function is learned. An actor-critic algorithm is policy-based and maintains both a policy function $\pi(a_t|s_t;\theta)$ and a value function $V^{\pi}(s_t;\theta_v)$. The policy function is called the *actor*, which takes actions based on the current policy π. The value function is called the *critic*, which serves as a baseline to evaluate the quality of the action using the state value $V^{\pi}(s_t;\theta_v)$. The network architecture of the A3C algorithm is shown in Figure 1. There are three convolutional layers (*conv1, conv2*, and *conv3*), one fully connected layer of size 512 (*fc1*), followed by two branches of fully connected layer: *fc2* is the policy function output layer which is of the same size as the number of actions and *fc3* is the value function output layer of size 1.

In A3C, k actor-learners run in parallel with their own copies of the environment and parameters for the policy and value function, which enables exploration of different parts of the environment and therefore observations will not be correlated. Each actor-learner performs a parameter update every $t_{\max}$ actions, or when a terminal state is reached—this is similar to using minibatch update as was done in DQN. Updates are synchronized to a master learner that maintains a central policy and value function, which will be the final policy upon the completion of training.

The policy network is directly parameterized and improved via policy gradient (Sutton & Barto, 2018). To reduce the variance in policy gradient, an advantage function is used and calculated as $A(s_t, a_t;\theta,\theta_v) = Q^{(n)}(s_t, a_t;\theta,\theta_v) - V(s_t;\theta_v)$. The $Q^{(n)}$ function is defined as

$$Q^{(n)}(s_t, a_t;\theta,\theta_v) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n};\theta_v) \tag{1}$$

where n is upper-bounded by $t_{\max}$. The loss function for the policy network is then defined as

$$L(\theta) = \nabla_\theta \log \pi(a_t|s_t;\theta) A(a_t, s_t;\theta,\theta_v) + \beta \nabla_\theta H(\pi(s_t;\theta))$$

where H is the entropy of policy π that encourages exploration and therefore helps prevent premature convergence to suboptimal policies. The value network is updated using the loss function

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

$$L(\theta_v) = \nabla_{\theta_v}\left[\left(Q^{(n)}(s_t, a_t; \theta, \theta_v) - V(s_t; \theta_v)\right)^2\right]$$

3.2 Transformed Bellman operator

Reward clipping is introduced in DQN and is also used in A3C to cope with different reward scales among Atari games (Mnih *et al.*, 2015, 2016). However, this is problematic because the RL agent will not be able to distinguish between states with high rewards versus those with low rewards, resulting in learning a suboptimal policy. The poor performance in some Atari games has been attributed to reward clipping (Hester *et al.*, 2018).

In this work, we apply the TB operator (Pohlen *et al.*, 2018) to the A3C algorithm to overcome the problem of reward clipping. We use the raw rewards instead of clipping them to the scale of $[-1, 1]$. A function $h : \mathbb{R} \to \mathbb{R}$ is used to reduce the scale of $Q^{(n)}(s_t, a_t; \theta, \theta_v)$ (Equation (1)) and is transformed as

$$Q^{(n)}(s_t, a_t; \theta, \theta_v) = \sum_{k=0}^{n-1} h\left(\gamma^k r_{t+k} + \gamma^n h^{-1}\left(V\left(s_{t+n}; \theta_v\right)\right)\right) \tag{2}$$

$$h : z \mapsto \operatorname{sign}(z)\left(\sqrt{|z|+1} - 1\right) + \varepsilon z \tag{3}$$

$$h^{-1} : x \mapsto \operatorname{sign}(x)\left(\left(\frac{\sqrt{1 + 4\varepsilon(|x| + 1 + \varepsilon)} - 1}{2\varepsilon}\right)^2 - 1\right) \tag{4}$$

where εz is for regularization that ensures h^{-1} is Lipschitz continuous and a closed-form inverse.

4 Supervised pre-training for deep reinforcement learning

Deep RL can be divided into two subtasks: feature learning and policy learning. Although deep RL in itself has succeeded in learning both tasks simultaneously, it still suffers from long training time and slow learning. We believe that by addressing feature learning, we can jump-start the performance in an RL agent since it will be able to focus more on policy learning, which in turn speeds up the entire learning process.

In this article, we propose to use supervised pre-training on human demonstration data as a way to address feature learning. We train a multiclass classification network over a set of non-expert human demonstrations, where actions demonstrated by the human were used as the ground truth labels for a given input game frame. The network uses the same architecture as in A3C (shown in Figure 1) where the fc2 layer is used as the classification output layer. Note that we exclude the fc3 layer for the classification task. The network classifier minimizes a softmax cross-entropy loss using the RMSProp (Tieleman & Hinton, 2012) optimizer with a set of hyperparameters shown in Table 1. We also use gradient clipping and L2 regularization for more stable training.

However, we identify two problems when using non-expert human demonstration for pre-training. First, we assume the actions provided by the human are the correct labels—the low quality of our data shown in Table 2 indicates that we are pre-training with noisy data. In this work, we empirically study if the noise in the data would still allow us to learn important features. Second, the collected human data are highly imbalanced. For example, in the game of Breakout, after hitting the ball, the human usually do nothing until the ball bounces and starts falling back to the paddle, which results in most collected actions being the 'NOOP' action. In some games, the human demonstrator tends to use the simpler actions like 'LEFT' instead of the compound actions like 'LEFTFIRE'. The class imbalance problem plagues all six games used in our experiments. To cope with this, we use proportional sampling. During minibatch sampling, we randomly sample over all available actions based on their proportion to the entire demonstration data; doing so ensures that each minibatch includes a relatively balanced set of classes. We pre-train a classification model for each Atari game for 750 000 training iterations.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

Table 1 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

Parameter	Value
RMSProp learning rate	7×10^{-4}
RMSProp epsilon	1×10^{-5}
RMSProp decay	0.99
RMSProp momentum	0
Maximum gradient norm	0.5
Parameters unique to supervised pre-training	
Number of mini-batch updates	750 000
Batch size	32
L2 regularization weight	1×10^{-4}
Parameters unique to A3C only	
k parallel actors	16
$t_{\max}$	20
TB operator ε	10^{-2}

A3C=asynchronous advantage actor-critic; TB=transformed Bellman.

Table 2 Human demonstration size and quality. The data are of a small amount and the demonstrator is a non-expert, compared to the best human demonstration score in the state-of-the-art Ape-X DQfD algorithm (Pohlen *et al.*, 2018) (Ape-X DQfD did not collect human demonstration for SpaceInvaders)

Game	Worst score	Best score	Best score (Ape-X DQfD)	No. of states	No. of episodes
Asterix	6250	14 300	**18 100**	12 870	5
Breakout	26	59	**79**	10 190	10
MsPacman	4020	18 241	**55 021**	14 504	8
NameThisGame	2510	4840	**19 380**	17 113	4
Pong	−13	**5**	0	21 674	6
SpaceInvaders	545	1840	-	16 807	8

Best values are shown in bold.

After pre-training, the learned weights and biases from the classifier are then used to initialize the A3C's network (instead of random initialization). Note that when using all layers' parameters from the pre-trained model (including the output *fc2* layer), normalizing the output layer's weights is necessary to achieve a positive result; we empirically observe that the values of the output layer tend to explode without normalization. To normalize the output layer, we keep track of the maximum value of the output layer during training, which is then used as the divisor to all weights and biases. We refer to our pre-training method as the *pre-trained model for A3C (PMfA3C)*. We also apply pre-training to the TB operator variant of the A3C algorithm, and we refer to it as *PMfA3C-TB*.

5 Experimental design

We evaluate our approach in six Atari games: Asterix, Breakout, MsPacman, NameThisGame, Pong, and SpaceInvaders. We use the deterministic version four of the Atari 2600 environment from OpenAI Gym (Brockman *et al.*, 2016). We follow the Atari settings described in the DQN algorithm and OpenAI baselines Atari wrapper (Mnih *et al.*, 2015; Dhariwal *et al.*, 2017). Here are the Atari settings used:

- At the beginning of a game, the agent executes a random x ($0 \leq x \leq 30$) number of 'NOOP' actions.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

- Take an action for games (e.g., Breakout) that remains static unless pressing 'FIRE'.
- Apply max pooling over the game frames to remove flickering effects of the game.
- Consider loss of life as the end of an episode or as a terminal state, but only do a hard reset on the game environment (i.e., reset back to the initial game state) when losing all lives.
- Use a frame skip of four, meaning that an action is repeated for four frames before a new action is selected.

The network architecture for A3C is shown in Figure 1. The four most recent game frames are used as input to the network—each frame is converted to grayscale and resized to 84×84 without cropping. We use the same set of hyperparameters for all games (except for Pong). We summarize the hyperparameter values in Table 1. Gradient clipping is also used in A3C. For all experiments, we train a total of 50 million steps, distributed over 16 parallel A3C actors. Each step consists four game frames (since we use frame skip of four); thus, all experiments run a total of 200 million game frames.

5.1 Collection of human demonstration

We use the keyboard interface from OpenAI Gym (Brockman *et al.*, 2016) to enable interactions between the human demonstrator and the Atari environment. For each game, the demonstrator is provided with game rules and a set of valid actions with their corresponding keyboard keys. The frame skip is set to one to provide smoother game transitions during human plays (whereas we reset it to four during agent training). To simulate frame skipping during the demonstration, we collect every fourth frame of the game. At each collection step, we save (1) the game image (i.e., the state), (2) the action taken by the demonstrator, (3) the reward received, and (4) if the current state is a terminal state. For each episode, we allow a maximum of 20 minutes of playing time for the human demonstrator. The demonstration ends when the game reaches the time limit or when the game ends—whichever comes first. Table 2 provides a breakdown of the demonstration size and quality for all games.

5.2 Evaluation procedures

For all experiments, we perform policy evaluation on the RL agent at every one million training steps. We get the average testing score over 125 000 testing steps and report the average over four trials. We report the highest average reward of the RL agent and also measure the learning speed improvement using three metrics adapted from Taylor and Stone (2009):

1. *Best reward*: the highest average reward attained by the agent from over four trials.
2. *Final performance*: the final learned performance of the agent. We use the reward obtained at step 50 million as the value for the final performance.
3. *Total reward*: the total reward accumulated (i.e., the area under the learning curve (AUC)) by the agent. We approximate the AUC using the *trapezoidal rule*: $AUC \approx \sum_{t=1}^{T} \frac{f(x_{t-1})+f(x_t)}{2} \Delta x_t$, where $f(x_t)$ is the reward value at time t and $\Delta x_t = x_t - x_{t-1} = 10^6$ is the evaluation frequency. Note that for readability, we scale down all calculation results by one million and consider $\Delta x_t = 1$—this does not affect the comparison results.
4. *Reward improvement*: the ratio of the total reward improvement of the pre-trained agent compared to the baseline agent. We calculate it as $\%Ratio = \frac{AUC_{\text{pre-trained}} - AUC_{\text{baseline}}}{AUC_{\text{baseline}}} \times 100$.

6 Results

First, we present the performance of the baseline A3C and the TB operator-variant A3C (A3C-TB). Figure 2 shows that A3C-TB outperforms the baseline A3C in five out of the six games. Although Pong in A3C-TB has a low performance,[1] we still consider the results as consistent with the findings in Hester

[1] In the game of Pong, its rewards $r \in \{-1, 0, 1\}$ are already at the same reward scale as applying the reward clipping in A3C at $r \in [-1, 1]$. Thus, reward clipping has no effect in Pong's performance. However, applying the

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

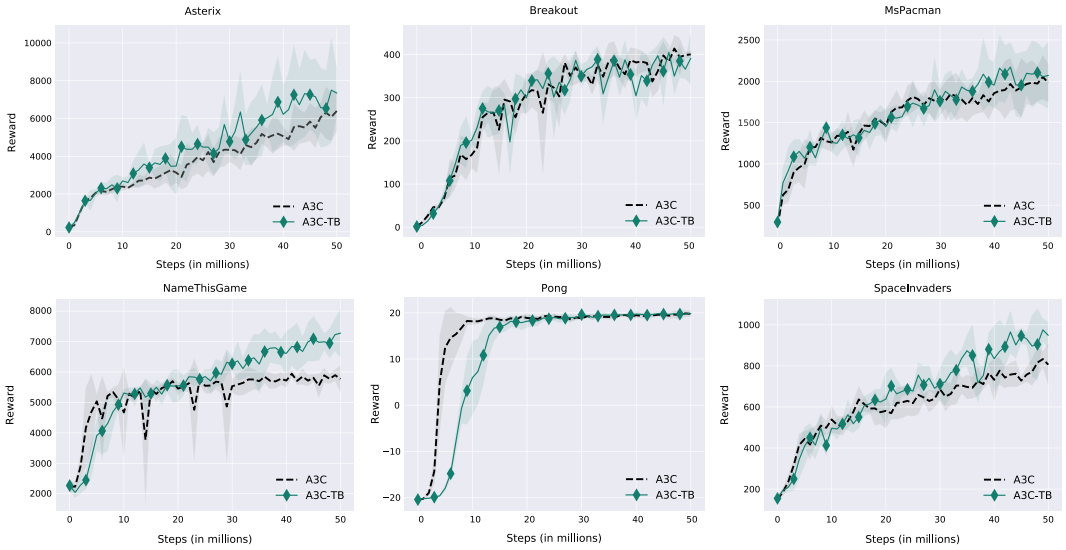


Figure 2 Performance of the baseline A3C and the TB operator variant A3C (A3C-TB). The *x*-axis is the total number of training steps (among all 16 actors), where each step consists of four game frames (we use frame skip of four). The *y*-axis is the average testing score over four trials where the shaded regions correspond to the standard deviation

et al. (2018) that using reward clipping leads to an agent learning a suboptimal policy. As we discussed in Section 3.2, A3C-TB enables using raw reward signals such that the RL agent can distinguish between low and high rewarding states, which leads to better policy learning. This is even more important to address when LfD for games that have distinct reward signals. For example, in the game of MsPacman, humans tend to take actions that move toward states with high rewards (e.g., eating an edible ghost is more rewarding than eating the dots). However, in a baseline A3C agent where the rewards are clipped, it sees all rewards as equal and might not be able to leverage the human knowledge of 'eating an edible ghost'.

Next, we present and discuss results for our pre-training approaches, *PMfA3C* and *PMfA3C-TB*. Note that we do not compare our results to recent algorithms in Hester *et al.* (2018) and Pohlen *et al.* (2018) since our training steps are much shorter and are not directly comparable to those that were trained on large scale.

6.1 Pre-training methods

We apply pre-training methods as described in Section 4 to both PMfA3C and PMfA3C-TB. A multiclass-classification network is pre-trained using the human demonstration data set. All weights and biases from the classifier are used to initialize A3C's network layers, conv1, conv2, conv3, fc1, and fc2; the fc3 layer is initialized randomly. Figure 3 shows the learning curve of both PMfA3C and PMfA3C-TB. Compared to the baseline A3C, PMfA3C outperforms in three out of six games; PMfA3C-TB shows a much stronger performance and exceeds the baseline in five out of six games.

Table 3 shows the quantitative performance improvements of our pre-training methods over the baseline. PMfA3C and PMfA3C-TB achieve the best performance in the game of MsPacman among all experiments with a remarkable total reward improvement of 67.36%. This verifies the importance of being able to learn from raw reward signals (instead of clipped rewards) in games with various reward scales. SpaceInvaders and Pong also show good reward improvement ratios in both pre-training methods compared to the baseline. While the performance increase in Breakout might not seem obvious visually from the learning curves, their total rewards indicate a 10.64% and an 8.06% improvement for PMfA3C

TB operator scales down Pong's reward values, which slows down the reward propagation. We believe this is the reason why Pong's performance is negatively affected in A3C-TB.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

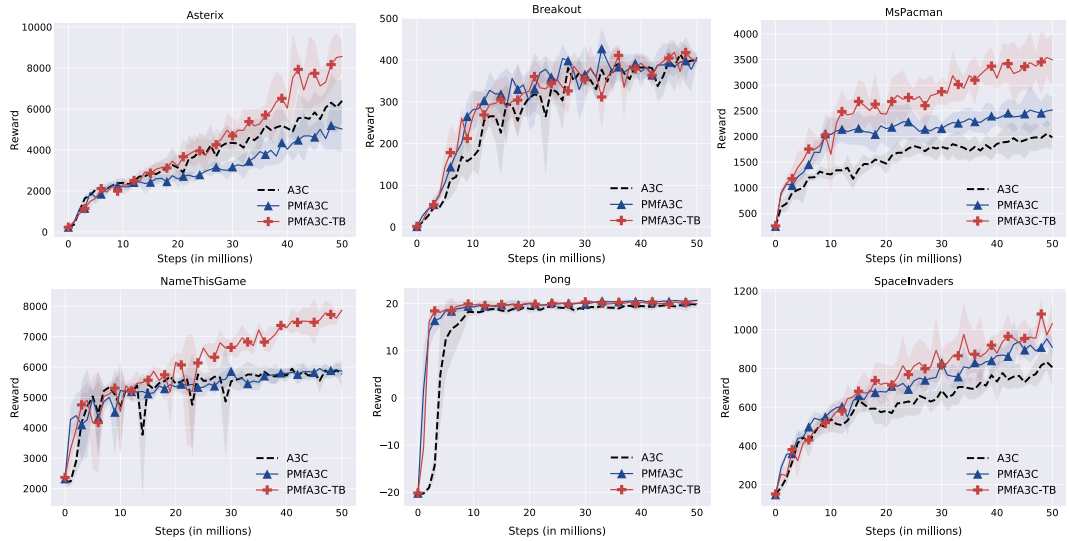


Figure 3 Performance of baseline and pre-training using A3C. The *x*-axis is the total number of training steps (among all 16 actors), where each step consists of four game frames (we use frame skip of four). The *y*-axis is the average testing score over four trials where the shaded regions correspond to the standard deviation

Table 3 Quantitative evaluation of pre-training methods using four metrics: best reward, final performance, total reward, and reward improvement

Game		Best reward	Final performance	Total reward	Reward improvement (%)
Asterix	A3C	6398.44	6398.43 ± 1072.22	187 702.21 ± 26 249.69	-
	A3C-TB	7500.58	7345.85 ± 1258.83	**223 942.78 ± 47 077.93**	**21.08**
	PMfA3C	5201.08	5022.76 ± 1134.26	153 889.07 ± 15 016.30	−18.01
	PMfA3C-TB	**8566.49**	**8566.49 ± 724.55**	216 571.05 ± 18 629.16	15.38
Breakout	A3C	413.88	400.40 ± 8.04	14 198.48 ± 606.64	-
	A3C-TB	405.03	391.23 ± 60.39	14 197.93 ± 181.07	−0.29
	PMfA3C	**427.56**	**406.03 ± 7.83**	**15 709.87** ± 171.10	**10.64**
	PMfA3C-TB	419.28	398.13 ± 16.98	15 343.39 ± 472.31	8.06
MsPacman	A3C	2052.07	1980.46 ± 231.12	78 419.18 ± 6027.52	-
	A3C-TB	2172.01	2071.19 ± 405.62	80 959.67 ± 6773.66	3.24
	PMfA3C	2514.61	2514.61 ± 247.60	103 950.65 ± 8529.05	32.56
	PMfA3C-TB	**3539.0**	**3493.90 ± 510.68**	**131 239.08 ± 9851.32**	**67.36**
NameThisGame	A3C	5942.29	5775.73 ± 435.45	264 140.63 ± 9830.70	-
	A3C-TB	7276.32	7276.32 ± 799.27	282 540.94 ± 14 059.33	6.97
	PMfA3C	5952.68	5868.82 ± 164.64	266 544.47 ± 5827.28	0.91
	PMfA3C-TB	**7869.28**	**7869.28 ± 99.12**	**306 014.77 ± 4438.13**	**15.85**
Pong	A3C	19.89	19.79 ± 0.67	791.71 ± 32.08	-
	A3C-TB	19.84	19.84 ± 0.23	604.60 ± 46.49	−23.63
	PMfA3C	**20.67**	**20.67 ± 0.20**	**948.38 ± 8.21**	**19.79**
	PMfA3C-TB	20.44	19.96 ± 0.37	937.61 ± 19.44	18.43
SpaceInvaders	A3C	832.98	805.96 ± 123.04	30 533.94 ± 1413.44	-
	A3C-TB	975.27	948.26 ± 70.67	33 293.51 ± 1506.37	9.04
	PMfA3C	952.71	908.50 ± 50.73	35 163.27 ± 697.12	15.16
	PMfA3C-TB	**1081.21**	**1032.48 ± 111.07**	**36 902.61 ± 2529.46**	**20.86**

A3C=asynchronous advantage actor-critic; A3C-TB =transformed Bellman operator variant A3C; PMfA3C: pre-trained model for A3C.
Best values are shown in bold.

and PMfA3C-TB, respectively. In contrast, we note that PMfA3C did not help in NameThisGame and Asterix. The former shows comparable results as the baseline A3C with a negligible improvement of

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

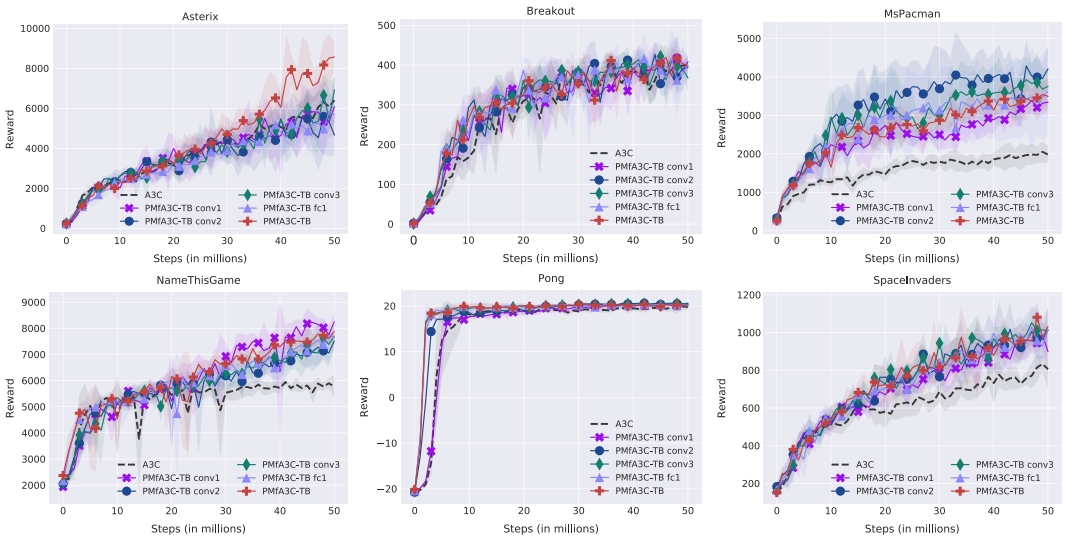


Figure 4 Performance of baseline and pre-training using A3C. The *x*-axis is the total number of training steps (among all 16 actors), where each step consists of four game frames (we use frame skip of four). The *y*-axis is the average testing score over four trials where the shaded regions correspond to the standard deviation

0.91%. When using PMfA3C for Asterix, it shows the worst performance among all experiments with an 18.01% performance drop compared to the baseline. However, we point out that when applying the TB operator, A3C-TB and PMfA3C-TB, Asterix outperforms the baseline by 21.08% and 15.38%, respectively.

In summary, our experiment results show that (1) using the TB operator is important for games with various reward scales and (2) pre-training is beneficial for training in the RL agent.

6.2 Ablation studies

In reusing networks for image classification, Yosinski *et al.* (2014) pointed out that the lower layers of a network tend to learn more general features while upper layers tend to learn more specific features toward the task. This is what inspired us to use all layers from the pre-trained network since both networks are trained on data collected from the same game.

To understand how reusing just a subset of the whole pre-trained network affects the performance speedup and how it compares to our best approach PMfA3C-TB, we conduct the following ablation studies:

- *PMfA3C-TB*: reuse all layers (as was done in Section 6.1).
- *PMfA3C-TB fc1*: reuse conv1, conv2, conv3, and fc1.
- *PMfA3C-TB conv3*: reuse conv1, conv2, and conv3.
- *PMfA3C-TB conv2*: reuse conv1 and conv2.
- *PMfA3C-TB conv1*: reuse conv1 only.

Figure 4 shows the results for our ablation study. We found the most intuitive results in Pong where the more pre-trained layers are reused, the better the results are. In Breakout and SpaceInvaders, reusing different layers shows no obvious distinctions in performance compared to reusing all layers. In Asterix, reusing all pre-trained layers still has the best performance; when reusing only a part of the pre-trained layers, the performance is similar to the baseline A3C. MsPacman and NameThisGame have the most varying results when different layers are reused. It is interesting to note that they show inverted performance for different reusing strategies—PMfA3C-TB conv1 has the worst result in MsPacman among all reusing strategies but shows the best result in NameThisGame. Despite this distinction, all pre-training methods in these two games outperform the baseline, regardless of the choice of reuse layers.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

In summary, our ablation study shows that reusing the entire pre-trained network consistently performs well—it either achieves the best results or is comparable to reusing other layers. We point out that reusing layers that are closer to the output layer (PMfA3C-TB fc1 and conv3) also shows competitive results in four out of six games. This is consistent with the findings in Yosinski *et al.* (2014) that lower layers learn general features while higher layers learn task-specific features, and transferring general features are more beneficial when the training and testing data are different. In our case, however, since the classification model and the A3C network are trained on data that are collected from the same game, leveraging task-specific features can be more helpful than reusing only general features.

6.3 What is really learned from pre-training?

In order to understand the benefit of our pre-training method, we visualize the feature maps in the last convolutional layer to assess if an RL agent can learn meaningful high-level information about the game through pre-training. We adopt the Grad-CAM as our visualization method since it is model-agnostic and can be used in any convolutional-based network without needing to change architectures (Selvaraju *et al.*, 2017). Using Grad-CAM, we analyze how different regions in feature maps are activated by corresponding actions under three settings: (1) a randomly initialized RL agent, (2) a pre-trained classification model, and (3) a final learned RL agent in PMfA3C-TB. By comparing feature map patterns among the three scenarios, we will be able to obtain an intuitive understanding of what is learned through pre-training and how an RL agent uses the pre-trained knowledge during its learning.

Grad-CAM is a visualization tool designed to increase the interpretability of prediction results of a deep neural network (Selvaraju *et al.*, 2017). Specifically, Grad-CAM computes the gradient of the target logits (output values before applying softmax) with respect to feature maps of a convolutional layer. The importance weight of each neuron can then be obtained by performing global averaging pooling (Lin *et al.*, 2013) over the gradients. For a classification task, the target logits refer to the score of a target class, while in the context of RL, the target class can be considered as the action a taken based on the current policy π. Given any game state s, the score y^a of an action a is defined as $y^a = \pi(a|s;\theta)$, where θ parameterizes the policy network in A3C. We then compute gradients for y^a with respect to the feature maps of the last convolutional layer (denoted as M^k).[2] We compute the importance weight (denoted as α_k^a) in A3C as

$$\alpha_k^a = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^a}{\partial M_{ij}^k}}_{\text{gradients via backprop}}$$

Selvaraju *et al.* (2017) also combine forward activation maps and pass through a rectifier linear units (ReLU) to show the features that has a *positive* effect on the action of interest. This gives us the action-discriminative saliency map as

$$L^a_{\text{Grad-CAM}} = ReLU\left(\sum_k \alpha_k^a M^k\right)$$

However, the output of the discriminative saliency map did not provide an interpretable visualization under A3C. According to Selvaraju *et al.* (2017), the weight α_k^a captures the *importance* of the feature map k. Since α_k^a can have both positive and negative values, we choose to emphasize only on positive weights (i.e., captures the most important features). Thus, we instead apply ReLU directly to α_k^a, transforming $L^a_{\text{Grad-CAM}}$ as

$$L^a_{\text{Grad-CAM}} = \sum_k ReLU\left(\alpha_k^a\right) M^k$$

[2] In Selvaraju *et al.* (2017), the feature maps are denoted as A^k; we change this notation to avoid confusion with the notation of the action a in the context of RL.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

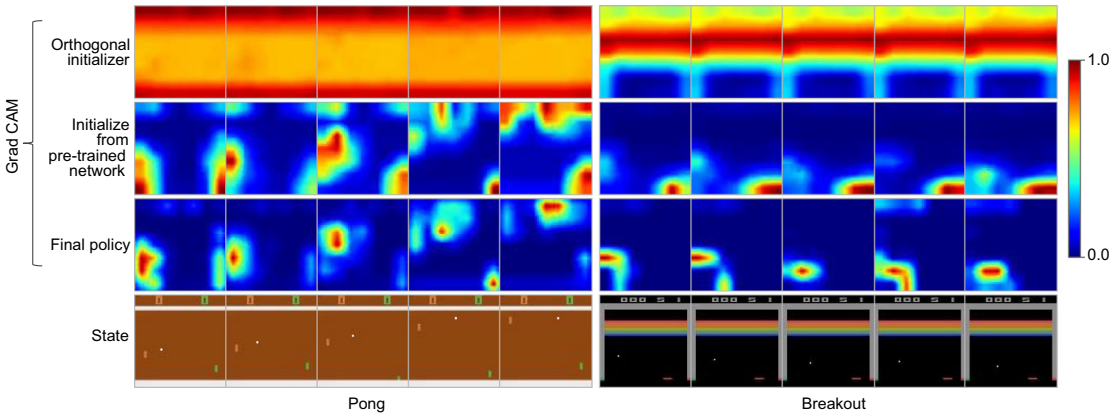


Figure 5 (Color online) Gradient-weighted Class Activation Mapping visualization for Pong and Breakout. Top row: random initialization; second row: pre-trained model; third row: final learned RL policy; bottom row: the original game image

We present Grad-CAM results for two games, Pong and Breakout, as the running example in this section for analyzing what features are learned. Video clips for all six games' Grad-CAM results are available online at https://sites.google.com/view/pretrain-deeprl. Figure 5 shows example Grad-CAM results for a sequence of five frames in Pong and Breakout and each frame is used as the input image to the network. We run a forward pass to compute the target logits y^a and output an action a based on the current policy π. Note that actions are not executed in the environment but only used to compute $L^a_{\text{Grad-CAM}}$—we do not perform gradient updates to the network. To ensure we compare the three models (random initialized, pre-trained, and final learned RL) with the same set of game inputs, we perform one episode of evaluation (until the game ends or reaches 5000 testing steps, whichever comes first) using the final learned RL agent and save all game images it encounters, and then use these images as a sequential state input to calculate Grad-CAM for the random initialized agent and the pre-trained model. Grad-CAM is visualized as a heatmap of scale [0, 1]: red (value close to 1) indicates regions that are important for a corresponding action while blue (value close to 0) indicates unimportant regions. The top row shows the feature map for a randomly initialized agent; the second row is the feature map of a pre-trained classifier; the third row shows a final learned RL agent's feature map; the bottom row shows the original game image.

At the beginning of training when the network weights are randomly initialized (top row of Figure 5), both Pong and Breakout agents act randomly and overall consider the entire game state to be important. In particular, Pong agent sees the top part (where the score is located shows more red color) and somehow the bottom part to be more important than the middle part (orange color) of the game image; Breakout agent also considers the top part to be important—with more focus on the location of bricks (red-orange color) than the location of the score (yellow-green color)—but the bottom part of the image is shown to be not important at all (blue color).

The important regions change notably when the network reuses weights from pre-trained models (second row of Figure 5). Interestingly, in both games, the agent learns to pay attention to the paddle movements. In Pong, both the opponent's paddle (left) and the agent's paddle (right) are identified to be important; in Breakout, the paddle at the bottom becomes the most important (whereas under random initialization the bottom part of the image is considered to be the least). The result is intuitive for the pre-trained model as it finds the object that follows the cardinal directions of the actions taken to be the most important feature—when the demonstrator takes an action, the object that correlates the most to this action is the paddle.

We see a further change of important regions in the final learned RL policy (third row of Figure 5). Instead of paying attention to the paddle, both games learn to track the movement of the ball. This is particularly clear in Pong: the important features evolve following roughly the same trajectory as the movement of the ball, from the bottom-left to the top-right of the game image.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

While the Grad-CAM examples in Figure 5 show that the pre-trained model picks up different important regions than the final RL policy, we also observe that some pre-training features are carried on to the final RL policy. For example, in Pong, although the RL policy has more focus on the ball, the regions around paddles are still activated with a lower importance—an inheritance from the pre-train model. In addition, in the Grad-CAM video clips where we show a complete run on all six games, it is more clear that important features in the pre-training are also identified in the final RL policy. We refer the readers to the video clips at https://sites.google.com/view/pretrain-deeprl.

7 Conclusion and discussion

The goal of this article is not to defeat the state-of-the-art results for Atari. Instead, we want to address the problem of slow learning. Other work focuses on increasing computational resources to speed up the training, while our work focuses more on improving learning speed, which is the ability to learn better policy with a smaller amount of game environment interactions. We attain the speedup in learning by supervised pre-training of the deep RL's network using non-expert human demonstrations.

We used the TB operator (Pohlen *et al.*, 2018) in A3C, which addresses reward clipping that allows the RL agent to differentiate high and low rewarding states. The TB operator helps our pre-training approach achieve improvements in all six games that we evaluated our approach. Our pre-training approach improves the reward in MsPacman with 67.36%. In addition, A3C in MsPacman reach the highest average reward of 2052 at 49 million training steps, while PMfA3C-TB in MsPacman only takes 11 million training steps to surpass A3C's highest average reward. This is quite a significant speedup in the learning speed especially when we only pre-trained PMfA3C-TB's network on 14 504 game states from a non-expert human.

We also have a much better understanding of what features are being considered during pre-training by using our modified Grad-CAM. One future direction for pre-training in deep RL is to drive the activation mapping toward the objects in the game state image. We believe it would be easier for non-expert humans to simply identify the important objects in the game relative to actually playing the game—this would be another way of using humans to improve learning.

As we investigate further ways to improve our approach, we know there is a limit to how much improvement pre-training can provide without addressing policy learning. In our approach, we have already trained a model with a policy that tries to imitate the human demonstrator, and thus we can extend this work by using the pre-trained model's policy to provide advice to the agent (Wang & Taylor, 2017).

To summarize, learning both features and policy directly from raw images through deep neural networks is a major factor why learning is slow in deep RL. This article has demonstrated the following: (1) we have shown through Grad-CAM that using supervised pre-training with non-expert human demonstration data can be used for feature learning and (2) that our method of initializing deep RL's network with a supervised pre-trained model can significantly speed up learning in deep RL.

Acknowledgments

The A3C implementation was a modification of https://github.com/miyosuda/async_deep_reinforce. The authors thank Sahil Sharma and Kory Matthewson for providing very useful insights on the actor-critic method. We also 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

Abtahi, F. & Fasel, I. 2011. Deep belief nets as function approximators for reinforcement learning, *Restricted Boltzmann Machine (RBM)* **2**, h3.

Anderson, C. W., Lee, M. & Elliott, D. L. 2015. Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–7. IEEE.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* **57**(5), 469–483.

Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. 2013. The arcade learning environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research* **47**, 253–279.

Bojarski, M., Yeres, P., Choromanska, A., Choromanski, K., Firner, B., Jackel, L. & Muller, U. 2017. Explaining how a deep neural network trained with end-to-end learning steers a car. arXiv preprint arXiv:1704.07911.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. 2016. Openai gym.

Brys, T., Harutyunyan, A., Taylor, M. E. & Nowé, A. 2015. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 181–188. International Foundation for Autonomous Agents and Multiagent Systems.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S. & Amodei, D. 2017. Deep reinforcement learning from human preferences. In *NIPS*, Curran Associates, Inc.

Deng, Y., Bao, F., Kong, Y., Ren, Z. & Dai, Q. 2017. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems* **28**(3), 653–664.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y. & Zhokhov, P. 2017. Openai baselines. https://github.com/openai/baselines.

Du, Y., Czarnecki, W. M., Jayakumar, S. M., Pascanu, R. & Lakshminarayanan, B. 2018. Adapting auxiliary losses using gradient similarity. arXiv preprint arXiv:1812.02224.

Du, Y., de la Cruz, Jr., G. V., Irwin, J. & Taylor, M. E. 2016. Initial progress in transfer for deep reinforcement learning algorithms. In *Proceedings of the Deep Reinforcement Learning: Frontiers and Challenges (DeepRL) Workshop (at IJCAI 2016)*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J. & Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338, JMLR.org.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. & Bengio, S. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* **11**, 625–660.

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S. & Vincent, P. 2009. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 153–160, PMLR.

Glatt, R., d. Silva, F. L. & Costa, A. H. R. 2016. Towards knowledge transfer in deep reinforcement learning. In *2016 5th Brazilian Conference on Intelligent Systems (BRACIS)*, 91–96, IEEE.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z. & Gruslys, A. 2018. Deep q-learning from demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, AAAI Press.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D. & Kavukcuoglu, K. 2017. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, OpenReview.net.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J. & Jaśkowski, W. 2016. Vizdoom: a Doom-based AI research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

Kurin, V., Nowozin, S., Hofmann, K., Beyer, L. & Leibe, B. 2017. The Atari grand challenge dataset. arXiv preprint arXiv:1705.10998.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *ICLR*, OpenReview.net.

Lin, M., Chen, Q. & Yan, S. 2013. Network in network. arXiv preprint arXiv:1312.4400.

Miotto, R., Wang, F., Wang, S., Jiang, X. & Dudley, J. T. 2017. Deep learning for healthcare: review, opportunities and challenges. *Briefings in Bioinformatics* **19**(6), 1236–1246.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D. & Hadsell, R. 2017. Learning to navigate in complex environments. In *ICLR*, OpenReview.net.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. & Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937, JMLR.org.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533, Nature Publishing Group.

Ng, A. Y., Harada, D. & Russell, S. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *ICML*, **99**, 278–287, Morgan Kaufmann.

Pan, S. J. & Yang, Q. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* **22**(10), 1345–1359.

Papoudakis, G., Chatzidimitriou, K. C. & Mitkas, P. A. 2018. Deep reinforcement learning for Doom using unsupervised auxiliary tasks. CoRR abs/1807.01960.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press

Parisotto, E., Ba, J. L. & Salakhutdinov, R. 2016. Actor-mimic: deep multitask and transfer reinforcement learning. In *ICLR*, OpenReview.net.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Večerík, M., Hessel, M., Munos, R. & Pietquin, O. 2018. Observe and look further: achieving consistent performance on Atari. arXiv preprint arXiv:1805.11593.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Alexander, C. B. & Fei-Fei, L. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* **115**(3), 211–252.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K. & Hadsell, R. 2016, Policy distillation. In *ICLR*, OpenReview.net.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. & Batra, D. 2017. Grad-cam: visual explanations from deep networks via gradient-based localization. In *ICCV*, 618–626, IEEE.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* **529**(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. & Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science* **362**(6419), 1140–1144.

Sutton, R. S. & Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Taylor, M. E. & Stone, P. 2009. Transfer learning for reinforcement learning domains: a survey. *Journal of Machine Learning Research* **10**(Jul), 1633–1685.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N. & Pascanu, R. 2017. Distral: robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, 4496–4506.

Tieleman, T. & Hinton, G. 2012. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* **4**(2), 26–31.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J. & Tsing, R. 2017. Starcraft II: a new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782.

Wang, Z. & Taylor, M. E. 2017. Improving reinforcement learning with confidence-based demonstrations. In *Proceedings of the 26th International Conference on Artificial Intelligence (IJCAI)*, International Joint Conference on Artificial Intelligence.

Watkins, C. J. & Dayan, P. 1992. Q-learning. *Machine Learning* **8**(3–4), 279–292.

Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. 2014. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, 3320–3328.

Zhang, Y., Lee, K. & Lee, H. 2016. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICML*, JMLR.org.

https://doi.org/10.1017/S0269888919000055 Published online by Cambridge University Press