ACTOR-MIMIC DEEP MULTITASK AND TRANSFER REINFORCEMENT LEARNING

Emilio Parisotto, Jimmy Ba, Ruslan Salakhutdinov
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
{eparisotto,jimmy,rsalakhu}@cs.toronto.edu

ABSTRACT

The ability to act in multiple environments and transfer previous knowledge to new situations can be considered a critical aspect of any intelligent agent. Towards this goal, we define a novel method of multitask and transfer learning that enables an autonomous agent to learn how to behave in multiple tasks simultaneously, and then generalize its knowledge to new domains. This method, termed "Actor-Mimic", exploits the use of deep reinforcement learning and model compression techniques to train a single policy network that learns how to act in a set of distinct tasks by using the guidance of several expert teachers. We then show that the representations learnt by the deep policy network are capable of generalizing to new tasks with no prior expert guidance, speeding up learning in novel environments. Although our method can in general be applied to a wide range of problems, we use Atari games as a testing environment to demonstrate these methods.

1 INTRODUCTION

Deep Reinforcement Learning (DRL), the combination of reinforcement learning methods and deep neural network function approximators, has recently shown considerable success in high-dimensional challenging tasks, such as robotic manipulation (Levine et al., 2015; Lillicrap et al., 2015) and arcade games (Mnih et al., 2015). These methods exploit the ability of deep networks to learn salient descriptions of raw state input, allowing the agent designer to essentially bypass the lengthy process of feature engineering. In addition, these automatically learnt descriptions often significantly outperform hand-crafted feature representations that require extensive domain knowledge. One such DRL approach, the Deep Q-Network (DQN) (Mnih et al., 2015), has achieved state-of-the-art results on the Arcade Learning Environment (ALE) (Bellemare et al., 2013), a benchmark of Atari 2600 arcade games. The DQN uses a deep convolutional neural network over pixel inputs to parameterize a state-action value function. The DQN is trained using Q-learning combined with several tricks that stabilize the training of the network, such as a replay memory to store past transitions and target networks to define a more consistent temporal difference error.

Although the DQN maintains the same network architecture and hyperparameters for all games, the approach is limited in the fact that each network only learns how to play a single game at a time, despite the existence of similarities between games. For example, the tennis-like game of pong and the squash-like game of breakout are both similar in that each game consists of trying to hit a moving ball with a rectangular paddle. A network trained to play multiple games would be able to generalize its knowledge between the games, achieving a single compact state representation as the inter-task similarities are exploited by the network. Having been trained on enough source tasks, the multitask network can also exhibit transfer to new target tasks, which can speed up learning. Training DRL agents can be extremely computationally intensive and therefore reducing training time is a significant practical benefit.

The contribution of this paper is to develop and evaluate methods that enable multitask and transfer learning for DRL agents, using the ALE as a test environment. To first accomplish multitask learning, we design a method called "Actor-Mimic" that leverages techniques from model compression to train a single multitask network using guidance from a set of game-specific expert networks. The particular form of guidance can vary, and several different approaches are explored and tested empirically. To then achieve transfer learning, we treat a multitask network as being a DQN which was pre-trained on a set of source tasks. We show experimentally that this multitask pre-training can result in a DQN that learns a target task significantly faster than a DQN starting from a random initialization, effectively demonstrating that the source task representations generalize to the target task.

2 Background: Deep Reinforcement Learning

A Markov Decision Process (MDP) is defined as a tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, γ) where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{T}(s'|s,a)$ is the transition probability of ending up in state s' when executing action a in state s, $\mathcal{R}$ is the reward function mapping states in $\mathcal{S}$ to rewards in $\mathbb{R}$, and γ is a discount factor. An agent's behaviour in an MDP is represented as a policy $\pi(a|s)$ which defines the probability of executing action a in state s. For a given policy, we can further define the Q-value function $Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t | s_0 = s, a_0 = a]$ where H is the step when the game ends. The Q-function represents the expected future discounted reward when starting in a state s, executing a, and then following policy π until a terminating state is reached. There always exists at least one optimal state-action value function, $Q^*(s,a)$, such that $\forall s \in S, a \in A$, $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ (Sutton & Barto, 1998). The optimal Q-function can be rewritten as a Bellman equation:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim \mathcal{T}(\cdot|s,a)} \left[r + \gamma \cdot \max_{a' \in \mathcal{A}} Q^*(s',a') \right]. \tag{1}$$

An optimal policy can be constructed from the optimal Q-function by choosing, for a given state, the action with highest Q-value. Q-learning, a reinforcement learning algorithm, uses iterative backups of the Q-function to converge towards the optimal Q-function. Using a tabular representation of the Q-function, this is equivalent to setting $Q^{(n+1)}(s,a) = \mathbb{E}_{s' \sim \mathcal{T}(\cdot|s,a)}[r + \gamma \cdot \max_{a' \in \mathcal{A}} Q^{(n)}(s',a')]$ for the (n+1)th update step (Sutton & Barto, 1998). Because the state space in the ALE is too large to tractably store a tabular representation of the Q-function, the Deep Q-Network (DQN) approach uses a deep function approximator to represent the state-action value function (Mnih et al., 2015). To train a DQN on the (n+1)th step, we set the network's loss to

$$L^{(n+1)}(\theta^{(n+1)}) = \mathop{\mathbb{E}}_{s,a,r,s' \sim \mathcal{M}(\cdot)} \left[\left(r + \gamma \cdot \max_{a' \in \mathcal{A}} Q(s',a';\theta^{(n)}) - Q(s,a;\theta^{(n+1)}) \right)^2 \right], \tag{2}$$

where $\mathcal{M}(\cdot)$ is a uniform probability distribution over a replay memory, which is a set of the m previous (s,a,r,s') transition tuples seen during play, where m is the size of the memory. The replay memory is used to reduce correlations between adjacent states and is shown to have large effect on the stability of training the network in some games.

3 Actor-Mimic

3.1 Policy Regression Objective

Given a set of source games $S_1, ..., S_N$, our first goal is to obtain a single multitask policy network that can play any source game at as near an expert level as possible. To train this multitask policy network, we use guidance from a set of expert DQN networks $E_1, ..., E_N$, where E_i is an expert specialized in source task S_i. One possible definition of "guidance" would be to define a squared loss that would match Q-values between the student network and the experts. As the range of the expert value functions could vary widely between games, we found it difficult to directly distill knowledge from the expert value functions. The alternative we develop here is to instead match policies by first transforming Q-values using a softmax. Using the softmax gives us outputs which are bounded in the unit interval and so the effects of the different scales of each expert's Q-function are diminished, achieving higher stability during learning. Intuitively, we can view using the softmax from the perspective of forcing the student to focus more on mimicking the action chosen by the guiding expert at each state, where the exact values of the state are less important. We call this method "Actor-Mimic" as it is an actor, i.e. policy, that mimics the decisions of a set of experts. In particular, our technique first transforms each expert DQN into a policy network by a Boltzmann distribution defined over the Q-value outputs,

$$\pi_{E_i}(a|s) = \frac{e^{\tau^{-1} Q_{E_i}(s,a)}}{\sum\limits_{a' \in \mathcal{A}_{E_i}} e^{\tau^{-1} Q_{E_i}(s,a')}}, \tag{3}$$

where τ is a temperature parameter and $\mathcal{A}_{E_i}$ is the action space used by the expert E_i, $\mathcal{A}_{E_i} \subseteq \mathcal{A}$. Given a state s from source task S_i, we then define the policy objective over the multitask network as the cross-entropy between the expert network's policy and the current multitask policy:

$$\mathcal{L}^i_{policy}(\theta) = \sum_{a \in \mathcal{A}_{E_i}} \pi_{E_i}(a|s) \log \pi_{\text{AMN}}(a|s;\theta), \tag{4}$$

where $\pi_{\text{AMN}}(a|s;\theta)$ is the multitask Actor-Mimic Network (AMN) policy, parameterized by θ. In contrast to the Q-learning objective which recursively relies on itself as a target value, we now have a stable supervised training signal (the expert network output) to guide the multitask network.

To acquire training data, we can sample either the expert network or the AMN action outputs to generate the trajectories used in the loss. Empirically we have observed that sampling from the AMN while it is learning gives the best results. We later prove that in either case of sampling from the expert or AMN as it is learning, the AMN will converge to the expert policy using the policy regression loss, at least in the case when the AMN is a linear function approximator. We use an ϵ-greedy policy no matter which network we sample actions from, which with probability ϵ picks a random action uniformly and with probability $1 - \epsilon$ chooses an action from the network.

3.2 Feature Regression Objective

We can obtain further guidance from the expert networks in the following way. Let $h_{\text{AMN}}(s)$ and $h_{E_i}(s)$ be the hidden activations in the feature (pre-output) layer of the AMN and i'th expert network computed from the input state s, respectively. Note that the dimension of $h_{\text{AMN}}(s)$ does not necessarily need to be equal to $h_{E_i}(s)$, and this is the case in some of our experiments. We define a feature regression network $f_i(h_{\text{AMN}}(s))$ that, for a given state s, attempts to predict the features $h_{E_i}(s)$ from $h_{\text{AMN}}(s)$. The architecture of the mapping f_i can be defined arbitrarily, and f_i can be trained using the following feature regression loss:

$$\mathcal{L}^i_{FeatureRegression}(\theta, \theta_{f_i}) = \left\| f_i(h_{\text{AMN}}(s;\theta);\theta_{f_i}) - h_{E_i}(s) \right\|_2^2, \tag{5}$$

where θ and θ_{f_i} are the parameters of the AMN and i^{th} feature regression network, respectively. When training this objective, the error is fully back-propagated from the feature regression network output through the layers of the AMN. In this way, the feature regression objective provides pressure on the AMN to compute features that can predict an expert's features. A justification for this objective is that if we have a perfect regression from multitask to expert features, all the information in the expert features is contained in the multitask features. The use of the separate feature prediction network f_i for each task enables the multitask network to have a different feature dimension than the experts as well as prevent issues with identifiability. Empirically we have found that the feature regression objective's primary benefit is that it can increase the performance of transfer learning in some target tasks.

3.3 Actor-Mimic Objective

Combining both regression objectives, the Actor-Mimic objective is thus defined as

$$\mathcal{L}^i_{ActorMimic}(\theta, \theta_{f_i}) = \mathcal{L}^i_{policy}(\theta) + \beta * \mathcal{L}^i_{FeatureRegression}(\theta, \theta_{f_i}), \tag{6}$$

where β is a scaling parameter which controls the relative weighting of the two objectives. Intuitively, we can think of the policy regression objective as a teacher (expert network) telling a student (AMN) how they should act (mimic expert's actions), while the feature regression objective is analogous to a teacher telling a student why it should act that way (mimic expert's thinking process).

3.4 Transfering Knowledge: Actor-Mimic as Pretraining

Now that we have a method of training a network that is an expert at all source tasks, we can proceed to the task of transferring source task knowledge to a novel but related target task. To enable transfer to a new task, we first remove the final softmax layer of the AMN. We then use the weights of AMN as an instantiation for a DQN that will be trained on the new target task. The pretrained DQN is then trained using the same training procedure as the one used with a standard DQN. Multitask pretraining can be seen as initializing the DQN with a set of features that are effective at defining policies in related tasks. If the source and target tasks share similarities, it is probable that some of these pretrained features will also be effective at the target task (perhaps after slight fine-tuning).

4 Convergence properties of Actor-Mimic

We further study the convergence properties of the proposed Actor-Mimic under a framework similar to (Perkins & Precup, 2002). The analysis mainly focuses on L2-regularized policy regression without feature regression. Without losing generality, the following analysis focuses on learning from a single game expert softmax policy π_E. The analysis can be readily extended to consider multiple experts on multiple games by absorbing different games into the same state space. Let $D^\pi(s)$ be the stationary distribution of the Markov decision process under policy π over states $s \in \mathcal{S}$. The policy regression objective function can be rewritten using expectation under the stationary distribution of the Markov decision process:

$$\min_\theta \mathop{\mathbb{E}}_{s \sim D^{\pi_{\text{AMN},\epsilon\text{-greedy}}}(\cdot)} \left[\mathcal{H}\Big(\pi_E(a|s),\ \pi_{\text{AMN}}(a|s;\theta) \Big) \right] + \lambda \|\theta\|_2^2, \tag{7}$$

where $\mathcal{H}(\cdot)$ is the cross-entropy measure and λ is the coefficient of weight decay that is necessary in the following analysis of the policy regression. Under Actor-Mimic, the learning agent interacts with the environment by following an ϵ-greedy strategy of some Q function. The mapping from a Q function to an ϵ-greedy policy $\pi_{\epsilon\text{-greedy}}$ is denoted by an operator Γ, where $\pi_{\epsilon\text{-greedy}} = \Gamma(Q)$. To avoid confusion onwards, we use notation $p(a|s;\theta)$ for the softmax policies in the policy regression objective.

Assume each state in a Markov decision process is represented by a compact K-dimensional feature representation $\phi(s) \in \mathbb{R}^K$. Consider a linear function approximator for Q values with parameter matrix $\theta \in \mathbb{R}^{K \times |\mathcal{A}|}$, $\hat{Q}(s,a;\theta) = \phi(s)^T \theta_a$, where θ_a is the a^{th} column of θ. The corresponding softmax policy of the linear approximator is defined by $p(a|s;\theta) \propto \exp\{\hat{Q}(s,a;\theta)\}$.

4.1 Stochastic stationary policy

For any stationary policy π^*, the stationary point of the objective function Eq. (7) can be found by setting its gradient w.r.t. θ to zero. Let P_θ be a $|\mathcal{S}| \times |\mathcal{A}|$ matrix where its i^{th} row j^{th} column element is the softmax policy prediction $p(a_j|s_i;\theta)$ from the linear approximator. Similarly, let Π_E be a $|\mathcal{S}| \times |\mathcal{A}|$ matrix for the softmax policy prediction from the expert model. Additionally, let D_π be a diagonal matrix whose entries are $D_\pi(s)$. A simple gradient following algorithm on the objective function Eq. (7) has the following expected update rule using a learning rate $\alpha_t > 0$ at the t^{th} iteration:

$$\Delta\theta_t = -\alpha_t \left[\Phi^T D_\pi (P_{\theta_{t-1}} - \Pi_E) + \lambda\theta_{t-1} \right]. \tag{8}$$

Lemma 1. *Under a fixed policy π^* and a learning rate schedule that satisfies $\sum_{t=1}^\infty \alpha_t = \infty$, $\sum_{t=1}^\infty \alpha_t^2 < \infty$, the parameters θ, updated by the stochastic gradient descent learning algorithm described above, asymptotically almost surely converge to a unique solution θ^*.*

When the policy π^* is fixed, the objective function Eq. (7) is convex and is the same as a multinomial logistic regression problem with a bounded Lipschitz constant due to its compact input features. Hence there is a unique stationary point θ^* such that $\Delta\theta^* = 0$. The proof of Lemma 1 follows the stochastic approximation argument (Robbins & Monro, 1951).

4.2 Stochastic adaptive policy

Consider the following learning scheme to adapt the agent's policy. The learning agent interacts with the environment and samples states by following a fixed ϵ-greedy policy π'. Given the samples

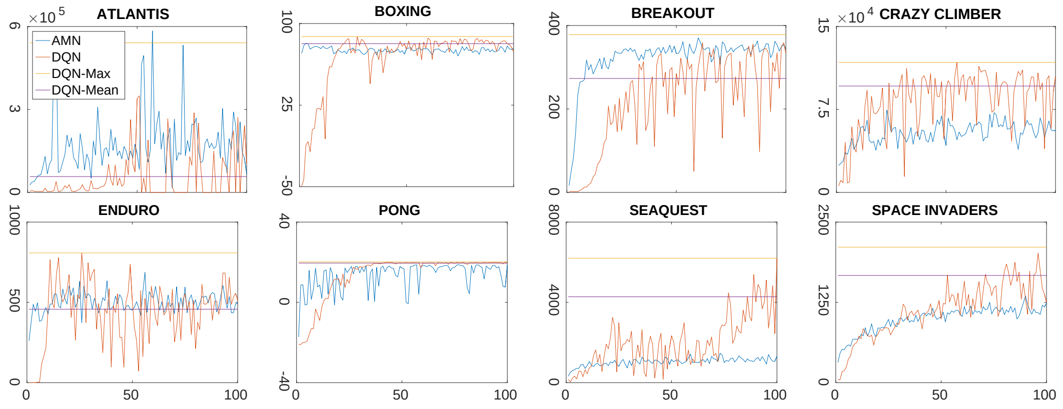

Figure 1: The Actor-Mimic and expert DQN training curves for 100 training epochs for each of the 8 games. A training epoch is 250,000 frames and for each training epoch we evaluate the networks with a testing epoch that lasts 125,000 frames. We report AMN and expert DQN test reward for each testing epoch and the mean and max of DQN performance. The max is calculated over all testing epochs that the DQN experienced until convergence while the mean is calculated over the last ten epochs before the DQN training was stopped. In the testing epoch we use $\epsilon = 0.05$ in the ϵ-greedy policy. The y-axis is the average unscaled episode reward during a testing epoch. The AMN results are averaged over 2 separately trained networks.

and the expert prediction, the linear function approximator parameters are updated using Eq. (8) to a unique stationary point θ'. The new parameters θ' are then used to establish a new ϵ-greedy policy $\pi'' = \Gamma(\hat{Q}_{\theta'})$ through the Γ operator over the linear function $\hat{Q}_{\theta'}$. The agent under the new policy π'' subsequently samples a new set of states and actions from the Markov decision process to update its parameters. The learning agent therefore generates a sequence of policies $\{\pi^1, \pi^2, \pi^3, ...\}$. The proof for the following theorem is given in Appendix A.

Theorem 1. *Assume the Markov decision process is irreducible and aperiodic for any policy π induced by the Γ operator and Γ is Lipschitz continuous with a constant c_ϵ, then the sequence of policies and model parameters generated by the iterative algorithm above converges almost surely to a unique solution π^* and θ^*.*

4.3 Performance Guarantee

The convergence theorem implies the Actor-Mimic learning algorithm also belongs to the family of no-regret algorithms in the online learning framework, see Ross et al. (2011) for more details. Their theoretical analysis can be directly applied to Actor-Mimic and results in a performance guarantee bound on how well the Actor-Mimic model performs with respect to the guiding expert.

Let $Z_t^{\pi'}(s, \pi)$ be the t-step reward of executing π in the initial state s and then following policy π'. The cost-to-go for a policy π after T-steps is defined as $J_T(\pi) = -T\,\mathbb{E}_{s\sim D(\cdot)}\left[\mathcal{R}(s,a)\right]$, where $\mathcal{R}(s, a)$ is the reward after executing action a in state s.

Proposition 1. *For the iterative algorithm described in Section (4.2), if the loss function in Eq. (7) converges to ϵ with the solution π_{AMN} and $Z_{T-t+1}^{\pi^*}(s,\pi^*) - Z_{T-t+1}^{\pi^*}(s,a) \geq u$ for all actions $a \in \mathcal{A}$ and $t \in \{1, \cdots, T\}$, then the cost-to-go of Actor-Mimic $J_T(\pi_{AMN})$ grows linearly after executing T actions: $J_T(\pi_{AMN}) \leq J_T(\pi_E) + uT\epsilon/\log 2$.*

The above linear growth rate of the cost-to-go is achieved through sampling from AMN action output π_{AMN}, while the cost grows quadratically if the algorithm only samples from the expert action output. Our empirical observations confirm this theoretical prediction.

5 Experiments

In the following experiments, we validate the Actor-Mimic method by demonstrating its effectiveness at both multitask and transfer learning in the Arcade Learning Environment (ALE). For our experiments, we use subsets of a collection of 20 Atari games. 19 games of this set were among the 29 games that the DQN method performed at a super-human level. We additionally chose 1 game, the game of Seaquest, on which the DQN had performed poorly when compared to a human expert. Details on the training procedure are described in Appendix B.

5.1 Multitask

To first evaluate the actor-mimic objective on multitask learning, we demonstrate the effectiveness of training an AMN over multiple games simultaneously. In this particular case, since our focus is

Network		Atlantis	Boxing	Breakout	Crazy Climber	Enduro	Pong	Seaquest	Space Invaders
DQN	Mean	57279	81.47	273.15	96189	457.60	19.581	4278.9	1669.2
	Max	541000	88.02	377.96	117593	808.00	20.140	6200.5	2109.7
AMN	Mean	165065	76.264	347.01	57070	499.3	15.275	1177.3	1142.4
	Max	584196	81.860	370.32	74342	686.77	18.780	1466.0	1349.0
$100\% \times \frac{\text{AMN}}{\text{DQN}}$	Mean	288.2%	93.61%	127.0%	59.33%	109.1%	78.01%	27.51%	68.44%
	Max	108.0%	93.00%	97.98%	63.22%	85.00%	93.25%	23.64%	63.94%

Table 1: Actor-Mimic results on a set of eight Atari games. We compare the AMN performance to that of the expert DQNs trained separately on each game. The expert DQNs were trained until convergence and the AMN was trained for 100 training epochs, which is equivalent to 25 million input frames per source game. For the AMN, we report maximum test reward ever achieved in epochs 1-100 and mean test reward in epochs 91-100. For the DQN, we report maximum test reward ever achieved until convergence and mean test reward in the last 10 epochs of DQN training. Additionally, at the last row of the table we report the percentage ratio of the AMN reward to the expert DQN reward for every game for both mean and max rewards. These percentage ratios are plotted in Figure 6. The AMN results are averaged over 2 separately trained networks.

on multitask learning and not transfer learning, we disregard the feature regression objective and set β to 0. Figure 1 and Table 1 show the results of an AMN trained on 8 games simultaneously with the policy regression objective, compared to an expert DQN trained separately for each game. The AMN and every individual expert DQN in this case had the exact same network architecture. We can see that the AMN quickly reaches close-to-expert performance on 7 games out of 8, only taking around 20 epochs or 5 million training frames to settle to a stable behaviour. This is in comparison to the expert networks, which were trained for up to 50 million frames.

One result that was observed during training is that the AMN often becomes more consistent in its behaviour than the expert DQN, with a noticeably lower reward variance in every game except Atlantis and Pong. Another surprising result is that the AMN achieves a significantly higher mean reward in the game of Atlantis and relatively higher mean reward in the games of Breakout and Enduro. This is despite the fact that the AMN is not being optimized to improve reward over the expert but just replicate the expert's behaviour. We also observed this increase in source task performance again when we later on increased the AMN model complexity for the transfer experiments (see Atlantis experiments in Appendix D). The AMN had the worst performance on the game of Seaquest, which was a game on which the expert DQN itself did not do very well. It is possible that a low quality expert policy has difficulty teaching the AMN to even replicate its own (poor) behaviour. We compare the performance of our AMN against a baseline of two different multitask DQN architectures in Appendix C.

5.2 Transfer

We have found that although a small AMN can learn how to behave at a close-to-expert level on multiple source tasks, a larger AMN can more easily transfer knowledge to target tasks after being trained on the source tasks. For the transfer experiments, we therefore significantly increased the AMN model complexity relative to that of an expert. Using a larger network architecture also allowed us to scale up to playing 13 source games at once (see Appendix D for source task performance using the larger AMNs). We additionally found that using an AMN trained for too long on the source tasks hurt transfer, as it is likely overfitting. Therefore for the transfer experiments, we train the AMN on only 4 million frames for each of the source games.

To evaluate the Actor-Mimic objective on transfer learning, the previously described large AMNs will be used as a weight initialization for DQNs which are each trained on a different target task. We additionally independently evaluate the benefit of the feature regression objective during transfer by having one AMN trained with only the policy regression objective (AMN-policy) and another trained using both feature and policy regression (AMN-feature). The results are then compared to the baseline of a DQN that was initialized with random weights.

The performance on a set of 7 target games is detailed in Table 2 (learning curves are plotted in Figure 7). We can see that the AMN pretraining provides a definite increase in learning speed for the 3 games of Breakout, Star Gunner and Video Pinball. The results in Breakout and Video Pinball demonstrate that the policy regression objective alone provides significant positive transfer in some target tasks. The reason for this large positive transfer might be due to the source game Pong having very similar mechanics to both Video Pinball and Breakout, where one must use a paddle to prevent a ball from falling off screen. The machinery used to detect the ball in Pong would likely be useful in detecting the ball for these two target tasks, given some fine-tuning. Additionally, the feature regression objective causes a significant speed-up in the game of Star Gunner compared to both the random initialization and the network trained solely with policy regression. Therefore even though the feature regression objective can slightly hurt transfer in some source games, it can provide large

Breakout	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	1.182	5.278	29.13	102.3	202.8	212.8	252.9	211.8	243.5	258.7
AMN-policy	**18.35**	102.1	**216.0**	**271.1**	**308.6**	**286.3**	**284.6**	**318.8**	**281.6**	**311.3**
AMN-feature	16.23	**119.0**	153.7	191.8	172.6	233.9	248.5	178.8	235.6	225.5

Gopher	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	8 mil	10 mil
Random	294.0	578.9	1360	**1540**	**1820**	1133	633.0	**1306**	**1758**	**1539**
AMN-policy	**715.0**	612.7	**1362**	924.8	1029	**1186**	**1081**	936.7	1251	1142
AMN-feature	636.2	**1110**	918.8	1073	1028	810.1	1008	868.8	1054	982.4

Krull	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	4302	6193	6576	7030	6754	5294	5949	5557	5366	6005
AMN-policy	**5827**	**7279**	6838	6971	7277	7129	7854	**8012**	7244	**7835**
AMN-feature	5033	7256	**7008**	**7582**	**7665**	**8016**	**8133**	6536	**7832**	6923

Road Runner	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	327.5	988.1	16263	**27183**	**26639**	**29488**	33197	27683	25235	**31647**
AMN-policy	**1561**	5119	**19483**	22132	23391	23813	**34673**	**33476**	**31967**	31416
AMN-feature	1349	**6659**	18074	16858	18099	22985	27023	24149	28225	23342

Robotank	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	**4.830**	**6.965**	9.825	13.22	**21.07**	**22.54**	**31.94**	**29.80**	**37.12**	**34.04**
AMN-policy	3.502	4.522	11.03	9.215	16.89	17.31	18.66	20.58	23.58	23.02
AMN-feature	3.550	6.162	**13.94**	**17.58**	17.57	20.72	20.13	21.13	26.14	23.29

Star Gunner	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	221.2	468.5	927.6	1084	1508	1626	3286	16017	36273	45322
AMN-policy	274.3	302.0	978.4	1667	4000	14655	31588	45667	38738	53642
AMN-feature	**1405**	**4570**	**18111**	**23406**	**36070**	**46811**	**50667**	**49579**	**50440**	**56839**

Video Pinball	1 mil	2 mil	3 mil	4 mil	5 mil	6 mil	7 mil	8 mil	9 mil	10 mil
Random	2323	8549	6780	5842	10383	11093	8468	5476	9964	11893
AMN-policy	**2583**	**25821**	**95949**	**143729**	**57114**	**106873**	**111074**	**73523**	**34908**	**123337**
AMN-feature	1593	3958	21341	12421	15409	18992	15920	48690	24366	26379

Table 2: Actor-Mimic transfer results for a set of 7 games. The 3 networks are trained as DQNs on the target task, with the only difference being the weight initialization. "Random" means random initial weights, "AMN-policy" means a weight initialization with an AMN trained using policy regression and "AMN-feature" means a weight initialization with an AMN trained using both policy and feature regression (see text for more details). We report the average test reward every 4 training epochs (equivalent to 1 million training frames), where the average is over 4 testing epochs that are evaluated immediately after each training epoch. For each game, we bold out the network results that have the highest average testing reward for that particular column.

benefits in others. The positive transfer in Breakout, Star Gunner and Video Pinball saves at least up to 5 million frames of training time in each game. Processing 5 million frames with the large model is equivalent to around 4 days of compute time on a NVIDIA GTX Titan.

On the other hand, for the games of Krull and Road Runner (although the multitask pretraining does help learning at the start) the effect is not very pronounced. When running Krull we observed that the policy learnt by any DQN regardless of the initialization was a sort of unexpected local maximum. In Krull, the objective is to move between a set of varied minigames and complete each one. One of the minigames, where the player must traverse a spiderweb, gives extremely high reward by simply jumping quickly in a mostly random fashion. What the DQN does is it kills itself on purpose in the initial minigame, runs to the high reward spiderweb minigame, and then simply jumps in the corner of the spiderweb until it is terminated by the spider. Because it is relatively easy to get stuck in this local maximum, and very hard to get out of it (jumping in the minigame gives unproportionally high reward compared to the other minigames), transfer does not really help learning.

For the games of Gopher and Robotank, we can see that the multitask pretraining does not have any significant positive effect. In particular, multitask pretraining for Robotank even seems to slow down learning, providing an example of negative transfer. The task in Robotank is to control a tank turret in a 3D environment to destroy other tanks, so it's possible that this game is so significantly different from any source task (being the only first-person 3D game) that the multitask pretraining does not provide any useful prior knowledge.

6 Related Work

The idea of using expert networks to guide a single mimic network has been studied in the context of supervised learning, where it is known as model compression. The goal of model compression is to reduce the computational complexity of a large model (or ensemble of large models) to a single smaller mimic network while maintaining as high an accuracy as possible. To obtain high accuracy, the mimic network is trained using rich output targets provided by the experts. These output targets are either the final layer logits (Ba & Caruana, 2014) or the high-temperature softmax outputs of the experts (Hinton et al., 2015). Our approach is most similar to the technique of (Hinton et al., 2015) which matches the high-temperature outputs of the mimic network with that of the expert network. In addition, we also tried an objective that provides expert guidance at the feature level instead of only at the output level. A similar idea was also explored in the model compression case (Romero et al., 2015), where a deep and thin mimic network used a larger expert network's intermediate features as guiding hints during training. In contrast to these model compression techniques, our method is not concerned with decreasing test time computation but instead using experts to provide otherwise unavailable supervision to a mimic network on several distinct tasks.

Actor-Mimic can also be considered as part of the larger Imitation Learning class of methods, which use expert guidance to teach an agent how to act. One such method, called DAGGER (Ross et al., 2011), is similar to our approach in that it trains a policy to directly mimic an expert's behaviour while sampling actions from the mimic agent. Actor-Mimic can be considered as an extension of this work to the multitask case. In addition, using a deep neural network to parameterize the policy provides us with several advantages over the more general Imitation Learning framework. First, we can exploit the automatic feature construction ability of deep networks to transfer knowledge to new tasks, as long as the raw data between tasks is in the same form, i.e. pixel data with the same dimensions. Second, we can define objectives which take into account intermediate representations of the state and not just the policy outputs, for example the feature regression objective which provides a richer training signal to the mimic network than just samples of the expert's action output.

Recent work has explored combining expert-guided Imitation Learning and deep neural networks in the single-task case. Guo et al. (2014) use DAGGER with expert guidance provided by Monte-Carlo Tree Search (MCTS) policies to train a deep neural network that improves on the original DQN's performance. Some disadvantages of using MCTS experts as guidance are that they require both access to the (hidden) RAM state of the emulator as well as an environment model. Another related method is that of guided policy search (Levine & Koltun, 2013), which combines a regularized importance-sampled policy gradient with guiding trajectory samples generated using differential dynamic programming. The goal in that work was to learn continuous control policies which improved upon the basic policy gradient method, which is prone to poor local minima.

A wide variety of methods have also been studied in the context of RL transfer learning (see Taylor & Stone (2009) for a more comprehensive review). One related approach is to use a dual state representation with a set of task-specific and task-independent features known as "problem-space" and "agent-space" descriptors, respectively. For each source task, a task-specific value function is learnt on the problem-space descriptors and then these learnt value functions are transferred to a single value function over the agent-space descriptors. Because the agent-space value function is defined over features which maintain constant semantics across all tasks, this value function can be directly transferred to new tasks. Banerjee & Stone (2007) constructed agent-space features by first generating a fixed-depth game tree of the current state, classifying each future state in the tree as either $\{win, lose, draw, nonterminal\}$ and then coalescing all states which have the same class or subtree. To transfer the source tasks value functions to agent-space, they use a simple weighted average of the source task value functions, where the weight is proportional to the number of times that a specific agent-space descriptor has been seen during play in that source task. In a related method, Konidaris & Barto (2006) transfer the value function to agent-space by using regression to predict every source tasks problem-space value function from the agent-space descriptors. A drawback of these methods is that the agent- and problem-space descriptors are either hand-engineered or generated from a perfect environment model, thus requiring a significant amount of domain knowledge.

7 Discussion

In this paper we defined Actor-Mimic, a novel method for training a single deep policy network over a set of related source tasks. We have shown that a network trained using Actor-Mimic is capable of reaching expert performance on many games simultaneously, while having the same model complexity as a single expert. In addition, using Actor-Mimic as a multitask pretraining phase can significantly improve learning speed in a set of target tasks. This demonstrates that the features learnt over the source tasks can generalize to new target tasks, given a sufficient level of similarity between source and target tasks. A direction of future work is to develop methods that can enable a targeted knowledge transfer from source tasks by identifying related source tasks for the given target task. Using targeted knowledge transfer can potentially help in cases of negative transfer observed in our experiments.

Acknowledgments: This work was supported by Samsung and NSERC.

REFERENCES

Ba, Jimmy and Caruana, Rich. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pp. 2654–2662, 2014.

Banerjee, Bikramjit and Stone, Peter. General game learning using knowledge transfer. In *International Joint Conferences on Artificial Intelligence*, pp. 672–677, 2007.

Bellemare, Marc G., Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bertsekas, Dimitri P. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems 27*, pp. 3338–3346, 2014.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Konidaris, George and Barto, Andrew G. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pp. 489–496, 2006.

Levine, Sergey and Koltun, Vladlen. Guided policy search. In *Proceedings of the 30th international conference on Machine Learning*, 2013.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702, 2015.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicholas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Perkins, Theodore J and Precup, Doina. A convergent form of approximate policy iteration. In *Advances in neural information processing systems*, pp. 1595–1602, 2002.

Robbins, Herbert and Monro, Sutton. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Romero, Adriana, Ballas, Nicolas, Kahou, Samira Ebrahimi, Chassang, Antoine, Gatta, Carlo, and Bengio, Yoshua. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations*, 2015.

Ross, Stephane, Gordon, Geoffrey, and Bagnell, Andrew. A reduction of imitation learning and structured prediction to no-regret online learning. *Journal of Machine Learning Research*, 15: 627–635, 2011.

Seneta, E. Sensitivity analysis, ergodicity coefficients, and rank-one updates for finite markov chains. *Numerical solution of Markov chains*, 8:121–129, 1991.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT Press Cambridge, 1998.

Taylor, Matthew E and Stone, Peter. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

Appendix A Proof of Theorem 1

Lemma 2. *For any two policies π^1,π^2, the stationary distributions over the states under the policies are bounded: $\|D_{\pi^1} - D_{\pi^2}\| \le c_D\|\pi^1 - \pi^2\|$, for some $c_D > 0$.*

Proof. Let T^1 and T^2 be the two transition matrices under the stationary distributions D_{π^1}, D_{π^2}. For any ij elements T^1_{ij}, T^2_{ij} in the transition matrices, we have,

$$\|T^1_{ij} - T^2_{ij}\| = \left\|\sum_a p(s_i|a,s_j)\left(\pi^1(a|s_j) - \pi^2(a|s_j)\right)\right\| \tag{9}$$

$$\le |\mathcal{A}|\|\pi^1(a|s_j) - \pi^2(a|s_j)\| \tag{10}$$

$$\le |\mathcal{A}|\|\pi^1 - \pi^2\|_\infty. \tag{11}$$

The above bound for any ij^{th} elements implies the Euclidean distance of the transition matrices is also upper bounded $\|T^1 - T^2\| \le |\mathcal{S}||\mathcal{A}|\|\pi^1 - \pi^2\|$. Seneta (1991) has shown that $\|D_{\pi^1} - D_{\pi^2}\| \le \frac{1}{1-\lambda^1}\|T^1 - T^2\|_\infty$, where λ^1 is the largest eigenvalue of T^1. Hence, there is a constant $c_D > 0$ such that $\|D_{\pi^1} - D_{\pi^2}\| \le c_D\|\pi^1 - \pi^2\|$. □

Lemma 3. *For any two softmax policy P_{θ^1}, P_{θ^2} matrices from the linear function approximator, $\|P_{\theta^1} - P_{\theta^2}\| \le c_J\|\Phi\theta^1 - \Phi\theta^2\|$, for some $c_J \ge 0$.*

Proof. Note that the i^{th} row j^{th} column element $p(a_j|s_i)$ in a softmax policy matrix P is computed by the softmax transformation on the Q function:

$$p_{ij} = p(a_j|s_i) = softmax\Big(Q(s_i,a_j)\Big) = \frac{e^{Q(s_i,a_j)}}{\sum_k e^{Q(s_i,a_k)}}. \tag{12}$$

Because the softmax function is a monotonically increasing element-wise function on matrices, the Euclidean distance of the softmax transformation is upper bounded by the largest Jacobian in the domain of the softmax function. Namely, for $c'_J = \max_{z\in \text{Dom}\, softmax}\|\frac{\partial softmax(z)}{\partial z}\|$,

$$\|softmax(x^1) - softmax(x^2)\| \le c'_J\|x^1 - x^2\|, \forall x^1, x^2 \in \text{Dom}\, softmax. \tag{13}$$

By bounding the elements in P matrix, it gives $\|P_{\theta^1} - P_{\theta^2}\| \le c_J\|\hat{Q}_{\theta^1} - \hat{Q}_{\theta^2}\| = c_J\|\Phi\theta^1 - \Phi\theta^2\|$. □

Theorem 1. *Assume the Markov decision process is irreducible and aperiodic for any policy π induced by the Γ operator and Γ is Lipschitz continuous with a constant c_ϵ, the sequence of policies and model parameters generated by the iterative algorithm above converges almost surely to a unique solution π^* and θ^*.*

Proof. We follow a similar contraction argument made in Perkins & Precup (2002) , and show the iterative algorithm is a contraction process. Namely, for any two policies π^1 and π^2, the learning algorithm above produces new policies $\Gamma(\hat{Q}_{\theta^1})$, $\Gamma(\hat{Q}_{\theta^2})$ after one iteration, where $\|\Gamma(\hat{Q}_{\theta^1}) - \Gamma(\hat{Q}_{\theta^2})\| \le \beta\|\pi^1 - \pi^2\|$. Here $\|\cdot\|$ is the Euclidean norm and $\beta \in [0, 1)$.

By Lipschtiz continuity,

$$\|\Gamma(\hat{Q}_{\theta^1}) - \Gamma(\hat{Q}_{\theta^2})\| \le c_\epsilon\|\hat{Q}_{\theta^1} - \hat{Q}_{\theta^2}\| = c_\epsilon\|\Phi\theta^1 - \Phi\theta^2\| \tag{14}$$

$$\le c_\epsilon\|\Phi\|\|\theta^1 - \theta^2\|. \tag{15}$$

Let θ^1 and θ^2 be the stationary points of Eq. (7) under π^1 and π^2. That is, $\Delta\theta^1 = \Delta\theta^2 = 0$ respectively. Rearranging Eq. (8) gives,

$$\|\theta^1 - \theta^2\| = \frac{1}{\lambda}\|\Phi^T D_{\pi^1}(P_{\theta^1} - \Pi_e) - \Phi^T D_{\pi^2}(P_{\theta^2} - \Pi_e)\| \tag{16}$$

$$= \frac{1}{\lambda}\|\Phi^T (D_{\pi^2} - D_{\pi^1})\Pi_e + \Phi^T D_{\pi^1} P_{\theta^1} - \Phi^T D_{\pi^1} P_{\theta^2} + \Phi^T D_{\pi^1} P_{\theta^2} - \Phi^T D_{\pi^2} P_{\theta^2}\| \tag{17}$$

$$= \frac{1}{\lambda}\|\Phi^T (D_{\pi^2} - D_{\pi^1})\Pi_e + \Phi^T D_{\pi^1}(P_{\theta^1} - P_{\theta^2}) + \Phi^T (D_{\pi^1} - D_{\pi^2}) P_{\theta^2}\| \tag{18}$$

$$\leq \frac{1}{\lambda}\left[\|\Phi^T\|\|D_{\pi^1} - D_{\pi^2}\|\|\Pi_e\| + \|\Phi^T\|\|D_{\pi^1}\|\|P_{\theta^1} - P_{\theta^2}\| + \|\Phi^T\|\|D_{\pi^1} - D_{\pi^2}\|\|P_{\theta^2}\|\right] \tag{19}$$

$$\leq c\|\pi^1 - \pi^2\|. \tag{20}$$

The last inequality is given by Lemma 2 and 3 and the compactness of Φ. For a Lipschtiz constant $c_\epsilon \geq c$, there exists a β such that $\|\Gamma(\hat{Q}_{\theta^1}) - \Gamma(\hat{Q}_{\theta^2})\| \leq \beta\|\pi^1 - \pi^2\|$. Hence, the sequence of policies generated by the algorithm converges almost surely to a unique fixed point π^* from Lemma 1 and the Contraction Mapping Theorem Bertsekas (1995). Furthermore, the model parameters converge w.p. 1 to a stationary point θ^* under the fixed point policy π^*. $\square$

Appendix B AMN training details

All of our Actor-Mimic Networks (AMNs) were trained using the Adam (Kingma & Ba, 2015) optimization algorithm. The AMNs have a single 18-unit output, with each output corresponding to one of the 18 possible Atari player actions. Having the full 18-action output simplifies the multitask case when each game has a different subset of valid actions. While playing a certain game, we mask out AMN action outputs that are not valid for that game and take the softmax over only the subset of valid actions. We use a replay memory for each game to reduce correlations between successive frames and stabilize network training. Because the memory requirements of having the standard replay memory size of 1,000,000 frames for each game are prohibitive when we are training over many source games, for AMNs we use a per-game 100,000 frame replay memory. AMN training was stable even with only a per-game equivalent of a tenth of the replay memory size of the DQN experts. For the transfer experiments with the feature regression objective, we set the scaling parameter β to 0.01 and the feature prediction network f_i was set to a linear projection from the AMN features to the i^{th} expert features. For the policy regression objective, we use a softmax temperature of 1 in all cases. Additionally, during training for all AMNs we use an ϵ-greedy policy with ϵ set to a constant 0.1. Annealing ϵ from 1 did not provide any noticeable benefit. During training, we choose actions based on the AMN and not the expert DQN. We do not use weight decay during AMN training as we empirically found that it did not provide any large benefits.

For the experiments using the DQN algorithm, we optimize the networks with RMSProp. Since the DQNs are trained on a single game their output layers only contain the player actions that are valid in the particular game that they are trained on. The experts guiding the AMNs used the same architecture, hyperparameters and training procedure as that of Mnih et al. (2015). We use the full 1,000,000 frame replay memory when training any DQN.

Appendix C Multitask DQN baseline results

As a baseline, we trained DQN networks over 8 games simultaneously to test their performance against the Actor-Mimic method. We tried two different architectures, the first is using the basic DQN procedure on all 8 games. This network has a single 18 action output shared by all games, but when we train or test in a particular game, we mask out and ignore the action values from actions that are invalid for that particular game. This architecture is denoted the Multitask DQN (MDQN). The second architecture is a DQN but where each game has a separate fully-connected feature layer and action output. In this architecture only the convolutions are shared between games, and thus the features and action values are completely separate. This was to try to mitigate the destabilizing

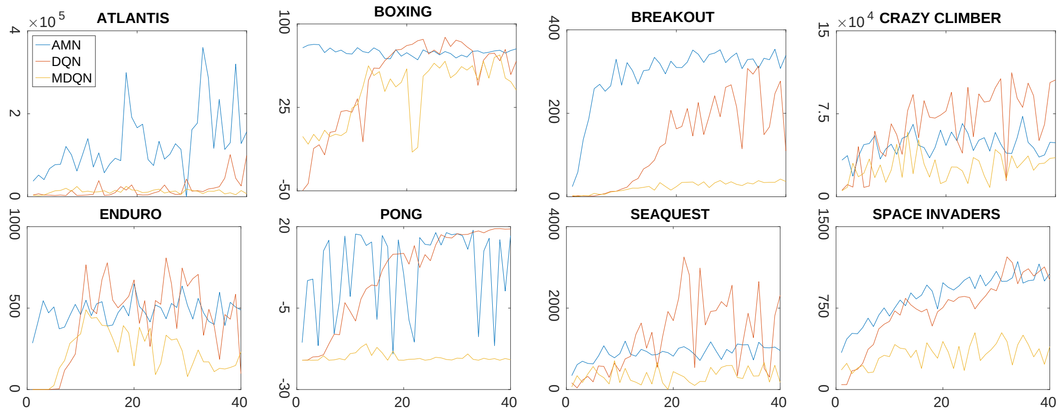

Figure 2: The Actor-Mimic, expert DQN, and Multitask DQN (MDQN) training curves for 40 training epochs for each of the 8 games. A training epoch is 250,000 frames and for each training epoch we evaluate the networks with a testing epoch that lasts 125,000 frames. We report AMN, expert DQN and MDQN test reward for each testing epoch. In the testing epoch we use $\epsilon = 0.05$ in the ϵ-greedy policy. The y-axis is the average unscaled episode reward during a testing epoch.

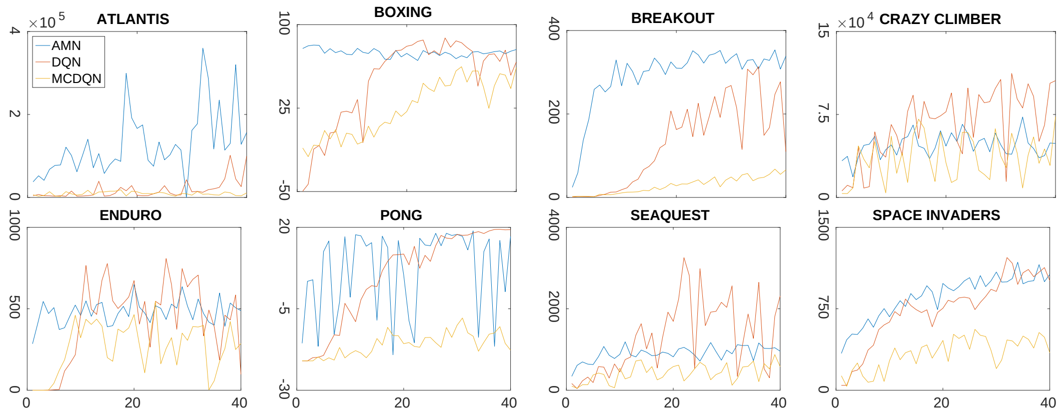

Figure 3: The Actor-Mimic, expert DQN, and Multitask Convolutions DQN (MCDQN) training curves for 40 training epochs for each of the 8 games. A training epoch is 250,000 frames and for each training epoch we evaluate the networks with a testing epoch that lasts 125,000 frames. We report AMN, expert DQN and MCDQN test reward for each testing epoch. In the testing epoch we use $\epsilon = 0.05$ in the ϵ-greedy policy. The y-axis is the average unscaled episode reward during a testing epoch.

effect that the different value scales of each game had during learning. This architecture is denoted the Multitask Convolutions DQN (MCDQN).

The results for the MDQN and MCDQN are shown in Figures 2 and 3, respectively. From the figures, we can observe that the AMN is far more stable during training as well as being consistently higher in performance than either the MDQN or MCDQN methods. In addition, it can be seen that the MDQN and MCDQN will often focus on performing reasonably well on a small subset of the source games, such as on Boxing and Enduro, while making little to no progress in others, such as Breakout or Pong. Between the MDQN and MCDQN, we can see that the MCDQN hardly improves results even though it has significantly larger computational cost that scales linearly with the number of source games.

For the specific details of the architectures we tested, for the MDQN the architecture was: 8x8x4x32-4 [1] $\rightarrow$ 4x4x32x64-2 $\rightarrow$ 3x3x64x64-1 $\rightarrow$ 512 fully-connected units $\rightarrow$ 18 actions. This is exactly the same network architecture as used for the 8 game AMN in Section 5.1. For the MCDQN, the bottom convolutional layers were the same as the MDQN, except there are 8 parallel subnetworks on top of the convolutional layers. These game-specific subnetworks had the architecture: 512 fully-connected units $\rightarrow$ 18 actions. All layers except the action outputs were followed with a rectifier non-linearity.

[1] Here we represent convolutional layers as WxWxCxN-S, where W is the width of the (square) convolution kernel, C is the number of input images, N is the number of filter maps and S is the convolution stride.

Appendix D Actor-Mimic Network Multitask Results for Transfer Pretraining

The network used for transfer consisted of the following architecture: 8x8x4x256-4 [1] $\rightarrow$ 4x4x256x512-2 $\rightarrow$ 3x3x512x512-1 $\rightarrow$ 3x3x512x512-1 $\rightarrow$ 2048 fully-connected units $\rightarrow$ 1024 fully-connected units $\rightarrow$ 18 actions. All layers except the final one were followed with a rectifier non-linearity.

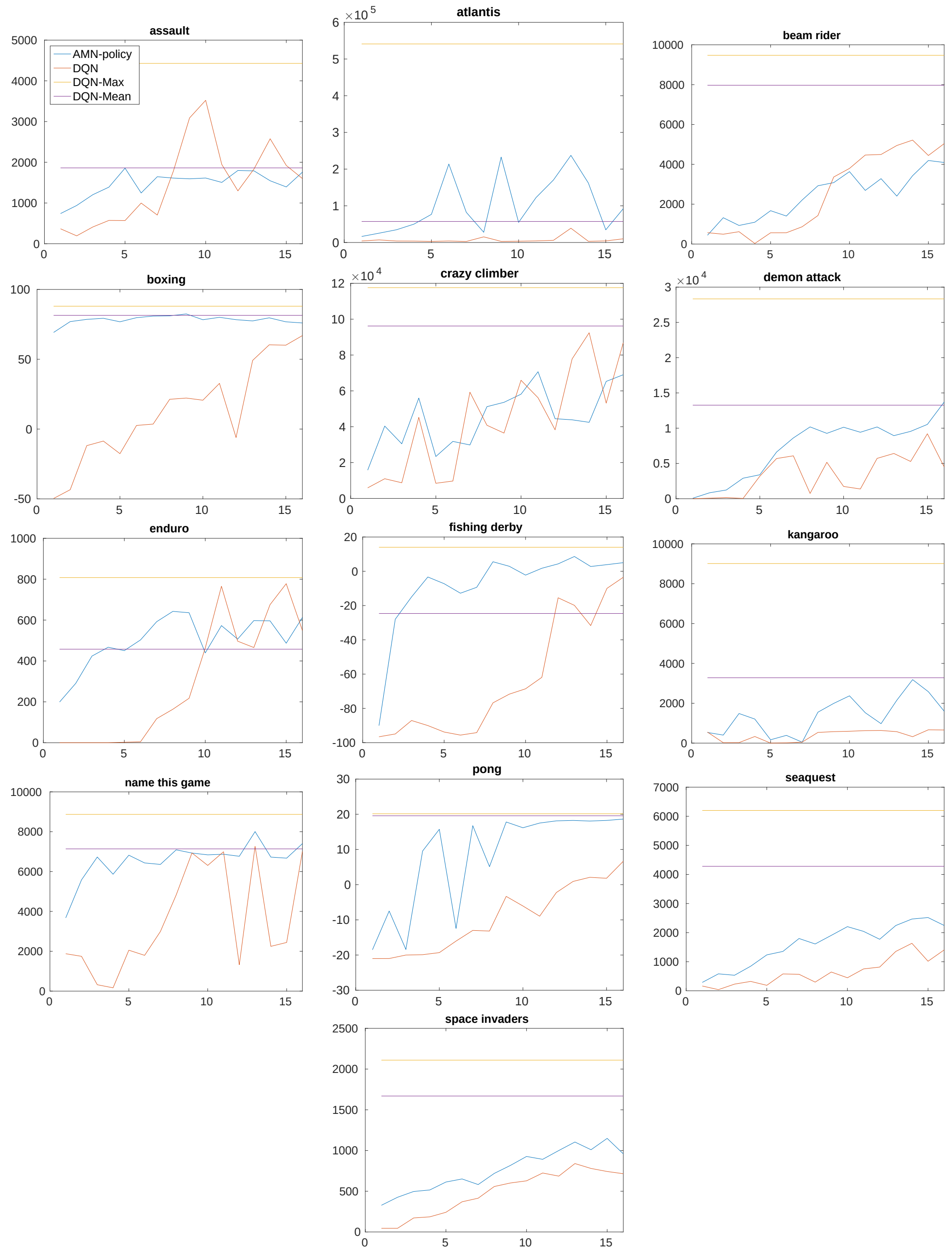

Figure 4: The Actor-Mimic training curves for the network trained solely with the policy regression objective (AMN-policy). The AMN-policy is trained for 16 epochs, or 4 million frames per game. We compare against the (smaller network) expert DQNs, which are trained until convergence. We also report the maximum test reward the expert DQN achieved over all training epochs, as well as the mean testing reward achieved over the last 10 epochs.

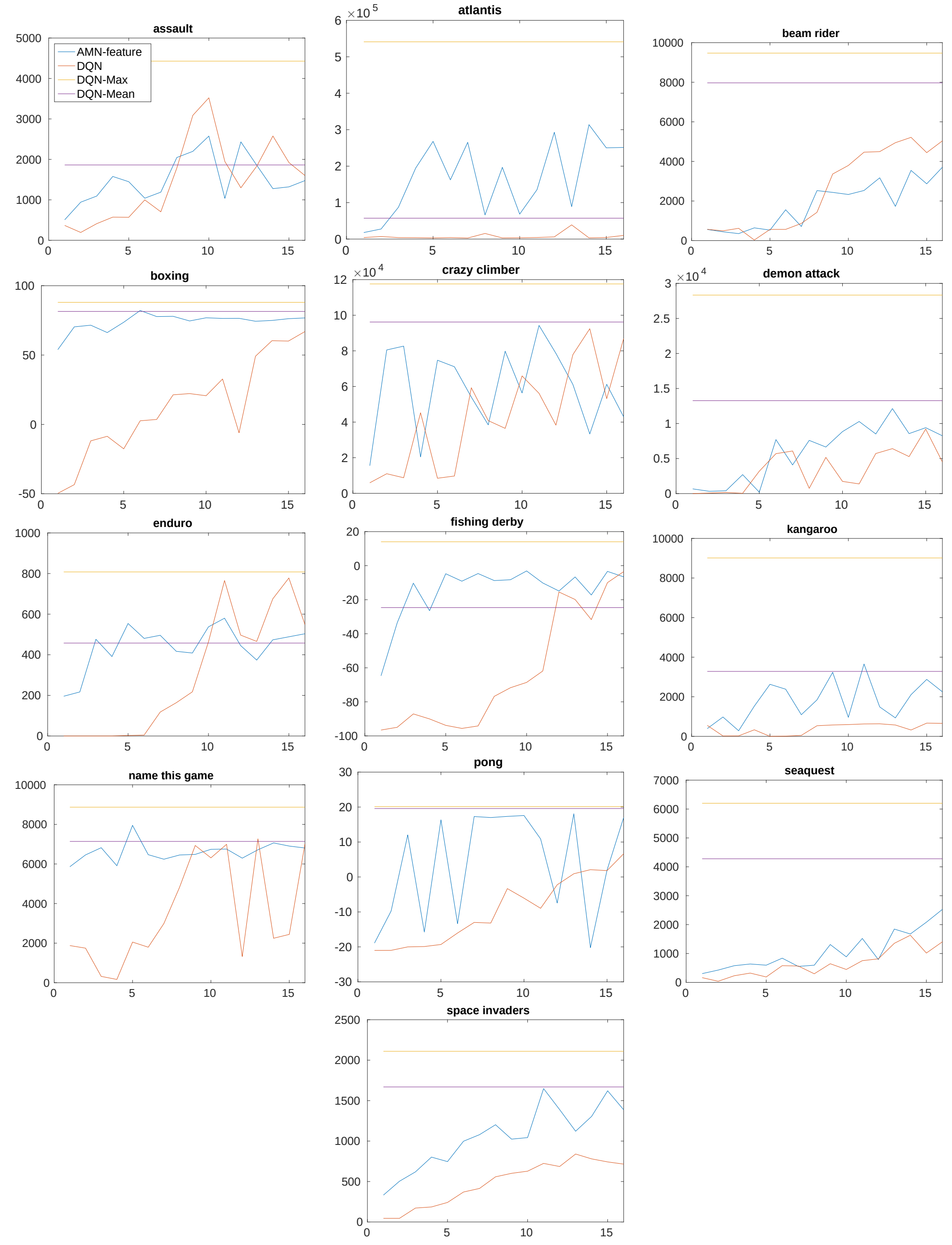

Figure 5: The Actor-Mimic training curves for the network trained with both the feature and policy regression objective (AMN-feature). The AMN-feature is trained for 16 epochs, or 4 million frames per game. We compare against the (smaller network) expert DQNs, which are trained until convergence. We also report the maximum test reward the expert DQN achieved over all training epochs, as well as the mean testing reward achieved over the last 10 epochs.

APPENDIX E TABLE 1 BARPLOT

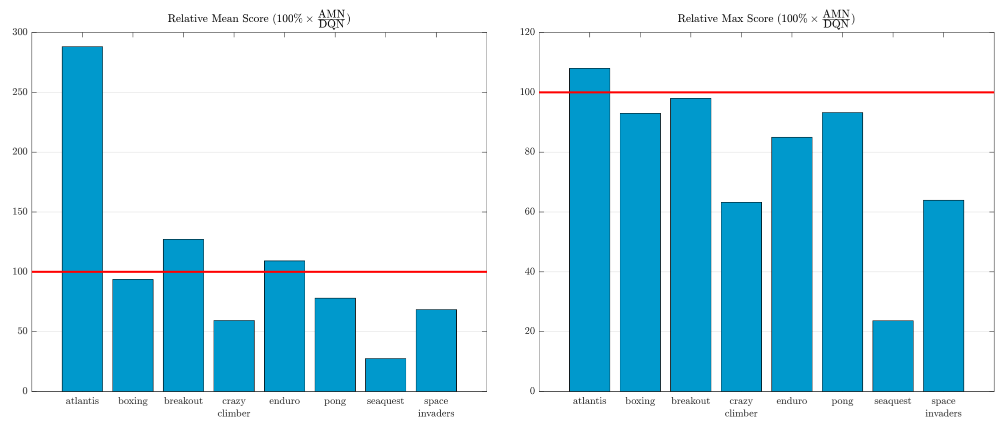

Figure 6: Plots showing relative mean reward improvement (left) and relative max reward improvement (right) of the multitask AMN over the expert DQNs. See Table 1 for details on how these values were calculated.

APPENDIX F TABLE 2 LEARNING CURVES

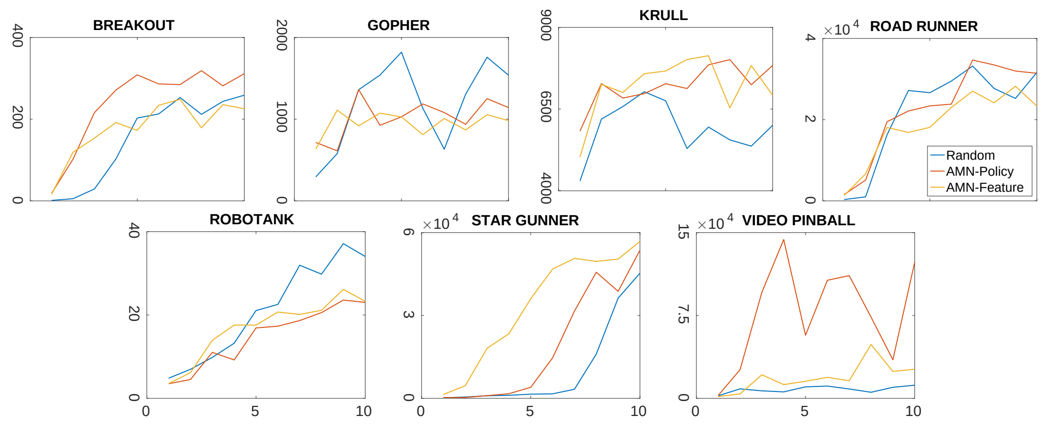

Figure 7: Learning curve plots of the results in Table2.