A Nested Architecture to Improve Transfer Learning in Deep Reinforcement Learning

Sobhan Moosavi, Mohammad Hossein Samavatian, Masood Delfarah
Department of Computer Science and Engineering
The Ohio State University
Columbus, Ohio 43210
{moosavinejaddaryakenari.1, samavatian.1, delfarah.1}@osu.edu

Abstract

Using deep neural networks, reinforcement learning algorithms have been able to outperform human experts in various games. In reinforcement learning, transferring knowledge between different domains saves time to train agents and can improve agents performance. Progressive neural network (PNN) [14] and Actor-mimic network (AMN) [11] are two recent frameworks to perform transfer learning in training agents for Atari 2600 games. In this work, we propose a novel neural network architecture which combines the core ideas from PNN and AMN and develop a nested architecture to improve transfer learning. Our experiments show that the proposed architecture can achieve better performance and converge faster for some of Atari 2600 games in comparison to the state-of-the-art.

1 Introduction

Transfer Learning (or inductive learning) in knowledge engineering and machine learning refers to extracting knowledge used in solving one problem and applying to a different but similar one. For example, the two tasks of face detection and face identifications address two inherently similar tasks. In fact, the low-level features in these tasks are almost identical. Artificial intelligence (AI) can also benefit from transfer learning. It is specially desirable when the data in the target domain is scarce or expensive to collect. For example, a self-driving car program can first be trained on simulated street environments, where it is easy to generate different weather and traffic conditions. The paper by Pratt et al. [12] was the first work to integrate transfer learning into machine learning. They trained multi-layer perception (MLP) models for the task of phoneme classification. Then, they initialized new networks using the weights from previous networks and observed faster convergence in the back-propagation algorithm.

Reinforcement learning (RL) is a powerful class of machine learning approaches which the problem is modeled as an agent moving in the space of possible states through actions with the objective of maximizing a reward function [16]. In this domain the machine[1] can benefit massively by transfer learning between similar tasks. For instance, the shooting-like games of *Seaquest* and *Centipede* are both similar in that each game is about hitting a moving object as an enemy. Therefore a network trained to play one game could be very beneficial to accelerate the training of the second network to play the other game.

Transfer learning techniques for reinforcement learning have been thoroughly studied in literature [13, 18, 7, 17]. There exist many different paradigms for transfer and multi-task reinforcement learning, as these have long been recognized as a critical challenges in AI research. Many of transfer learning approaches rely on linear or other simpler models in single and multi-agent environments [3, 4, 15], which is a limitation factor for their applicability. On the other hand, Deep RL network

[1] This machine can be a Neural Network or a Hidden Markov Process.

approaches show a significant improvement of performance for different tasks [8, 2]. The Deep Q-Network (DQN) method proposed by Minh *et al.* [10] has achieved super-human performance in many of Attari 2600 games. As a result, more recent studies are focused on transfer learning in this domain. In this paper, we study two of the very successful methods for transfer learning in DRL area, that is, Actor-mimic network (AMN) [11] and Progressive Neural Network (PNN) [14]. We also propose a combined architecture which is build on top of the AMN and PNN. Similar to the previous studies, we use the Attari 2600 games in our experiments. Here, the goal is to improve the transfer learning by *speeding-up* the training process and making the process to converge "faster" and "better". By latter, one can think about having a model which can outperform baselines in terms of evaluation results, where the baselines may or may not leverage the transfer learning.

As baseline, we use DQN, AMN, and PNN to examine transfer learning and compare them with our solution. Our experiments reveal that for a subset of the tasks, our method can outperform the baselines by faster training and better learning transfer. Moreover, our algorithm shows better performance in term of evaluation rewards during train in compare to DQN and AMN. In comparison with PNN, our method show faster convergence but with different degree of performance loss in different task sets. We summarize the main contributions of this paper as follows:

- We propose a novel, nested architecture to combine the key ideas of two previously proposed architectures (i.e., AMN and PNN), to save space, speed-up the training, and converge faster by better transfer of knowledge during the train.
- We conduct some extensive experiments on a group of 9 Atari games and study the advantages and disadvantages of the baselines and our solution. As result of this study, we made several observations on properties of different approaches which may lead us to further improve our solution and get better results in terms of future work.

The rest of the paper is organized as follows. In Section 2 we provide some background. Section 3 presents the proposed architecture. Experimental results are discussed in Section 4. Finally, Section 5 concludes the paper.

2 Background

In the Q-learning technique, the problem is formulated as a Markov Decision Process (MDP). Assume S and A are the sets of possible states and actions, respectively, and $T(s'|s,a)$ is the transition probability corresponding to taking the action a in state s and ending up in state s'. Each transaction results in a reward $r \in \mathbb{R}$. An agent follows a policy $\pi(a|s)$ which is the probability of taking action a while being in state s. Based on the policy a Q-function is defined as below:

$$Q^{\pi}(s,a) = E[\Sigma_{t=0}^{H} \gamma^t r_t | s = s_0, a = a_0] \tag{1}$$

where $\gamma < 1$ is the discount factor and H is the end of the game. The optimal Q-function can be written in the form of a recursive function as follows:

$$Q^{*}(s,a) = E[r + \gamma * maxQ(s',a')] \tag{2}$$

Estimates for the optimal action values can be learned using Q-learning. We can learn a parameterized value function $Q(s,a;\theta_t)$. After taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1}, the standard Q-learning updates the parameters as follow:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q + Q(S_t,A_t;\theta_t))\nabla_{\theta_t} Q(S_t,A_t;\theta_t) \tag{3}$$

where α is a scalar step size and the target Y_t^Q is defined as

$$Y_t^Q \equiv R_{t+1} + \gamma * \max_a Q(S_{t+1},a;\theta_t) \tag{4}$$

Multi layered Q-network known as Deep Q-Network (DQN) generates vector of action values $Q(s,.;\theta)$ for an input state s, where θ is parameters of the network. In n-dimensional state space and m-dimensional action space neural net is a function from R^n to R^m. The first factor of the DQN

algorithm is keeping two separated copies of the Q network which are named as *target network* and *online network*. The *target network* approximates the correct reward values with parameters θ^- and *online network* is the network under train. Every τ steps, the parameters θ of online network will be copied into θ^-, which is the parameter set of the target network. Thus, $\theta_\tau^- = \theta$ and kept fixed on all other steps [10]. This solution helps to make divergence or oscillations more unlikely by adding delay between the time that *target network* gets updated. So the target used by DQN is:

$$Y_t^{DQN} \equiv R_{t+1} + \gamma * \max_a Q(S_{t+1}, a; \theta_t^-) \tag{5}$$

The second factor is experience replay that stores observed transition for a period of time and samples uniformly from this memory bank to update the network. Both *target network* and experience replay techniques improve the DQN algorithm significantly [10].

3 Methodology

In this section, we describe our novel, nested neural network architecture. First, we describe the main building blocks and then provide more details on the nested architecture.

3.1 Deep Q-Network (DQN)

As described in Section 1, DQN is the first deep neural network architecture to train an agent for Atari 2600 game by reinforcement learning. The DQN uses a deep convolutional neural network over image pixel inputs to parameterize a state-action value function.

3.2 Actor-Mimic Neural Network (AMN)

Actor-mimic is one of the recent approaches which is proposed on top of the DQN to make improvements by introducing transfer learning [11]. Based on this approach, a *student* network will be trained on top of k DQN networks (also called *expert* networks) which are trained for k source games separately. The student network, after being trained, can play each of the source games as good as the their corresponding DQN (expert) network. Moreover, the student network can be used to initialize another network, possibly a DQN, which will be trained for a new game. Figure 1 depicts the overall architecture of an AMN which is trained on k expert networks and is used to initialize the expert network (DQN) for game $k+1$.

3.3 Progressive Neural Network (PNN)

Progressive neural network refers to a family of neural network architectures which inherently provide the opportunity of transfer learning [14]. Based on this architecture, first we assume we have $k-1$ pre-trained and frozen networks which are trained for $k-1$ games. In order to train a network for game k, the frozen $k-1$ models will be used in terms of following equation:

$$h_i^{(k)} = f\left(w_i^{(k)} h_{i-1}^{(k)} + \sum_{j=1}^{k-1} U_i^{(j)} h_{i-1}^{(j)}\right) \tag{6}$$

In Equation 6, $h_i^{(k)}$ is the i^{th} hidden layer of model k , $w_i^{(k)}$ is the weight matrix from hidden layer $i-1$ to i in model k, $h_{i-1}^{(k)}$ is the $(i-1)^{th}$ hidden layer of model k, $U_i^{(k:j)}$ refers to the frozen weight matrix from the hidden layer $i-1$ to i in model j, and $h_{i-1}^{(j)}$ is the $(i-1)^{th}$ frozen hidden layer of model j. Also, f is a non-linearity. An example of progressive network is shown in Figure 2, where there are two frozen models (the left two ones) and one under train (the right one). The arrows show the connection from frozen networks to the one under the train.

3.4 The Nested Architecture

In this work, we propose a combined architecture based on the core ideas of AMN and PNN. To do this, similar to AMN, we train a student network on top of k DQNs trained for k source games. Then, we use the AMN student network to train the network for game $k+1$ by progressive approach.

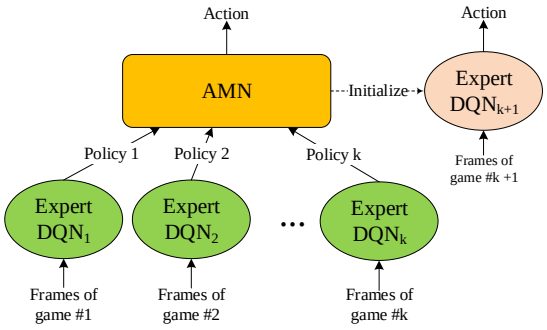

Figure 1: Actor-mimic network (AMN)

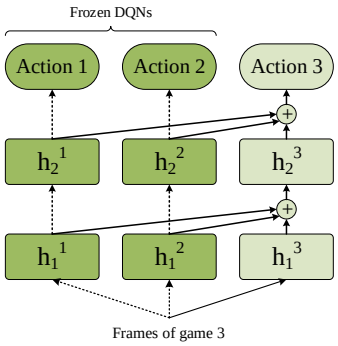

Figure 2: Progressive neural network (PNN)

Our proposed architecture is shown in Figure 3. In a similar way, we can have a more complex architecture. Assume we have k games and two DQNs are trained for the first two games. First, we train an AMN on top of those DQNs. Next, the AMN will be used to train a DQN for game 3 based on progressive architecture. Then, using the DQN trained for game 3 and AMN trained for games 1 and 2, we train another AMN which is capable of playing all three games to some extent. The latter AMN will be used to train a DQN for game 3 by progressive approach. Following this process, we will have a nested architecture as shown by Figure 4.

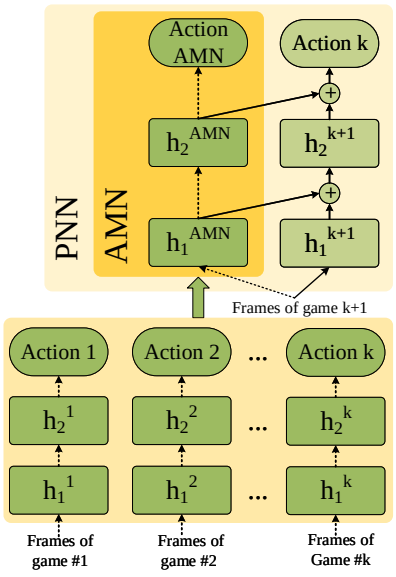

Figure 3: Proposed architecture to combine AMN and PNN on top of DQN (i.e., PNN(AMN))

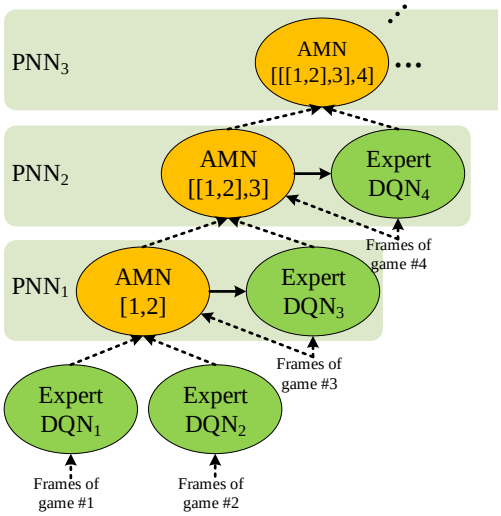

Figure 4: Nested Architecture based on PNN(AMN)

3.5 Advantages of the Proposed Architecture

The question is why our proposed architecture can be useful? Below, we try to summarize the rational behind this proposal.

- Speed-up the process: In the original Progressive architecture, the transfer is from several frozen networks to the one which is under train. However, in our proposal, we just transfer from one frozen network. This modification can help to boost the speed of training process.
- Prevent catastrophic forgetting: AMN suffers from the catastrophic forgetting by having just one model which can be used to initialize the network of a new game [14]. By our proposal, we take the advantage of Progressive architecture and train one DQN per game with no initialization.
- Save space: For the case of PNN, we have to have a set of frozen networks to be used to train a new network. By our proposal, we no longer save a set of networks and just keep the compressed model of previously trained networks.
- Extend the cascading: One of the shortcomings of the progressive architecture is that transfer cannot happen easily for more than 3 frozen networks, through the computation costs. By our proposal, we just do transfer from one network, thus, can address this shortcoming of the progressive architecture.

4 Experiments and Results

Our experiments are based on the Arcade Learning Environment (ALE) framework [1] which is a simple object-oriented framework that allows researchers to develop AI agents for Atari 2600 games. It is built on top of the Atari 2600 emulator Stella and separates the details of emulation from agent design. We used the original implementation of DQN and ActorMimic in torch [5, 6], and also developed Progressive Neural Network on top of DQN in Torch. For experiments, we used a single Tesla P100 GPU accelerator with 16GB of memory. In following subsections, we first describe the evaluation methodology and then present the results and discussion.

4.1 Evaluation methodology

For the experiments, we have selected a set of 9 Atari games and divided them into three groups of similar games. The groups are chosen based on similarities observed between them from human perspective, and this does not necessarily mean that they share the similar game strategies. Table 1 shows the games and their categorizations. In each group, we first build models for the first two games and then transfer the learning to the third game (i.e., Target game) to examine the transfer learning. We provide our evaluation process in terms of following steps:

- Train DQN for all games: first, we train a DQN for all 9 games and the resulting models are called DQN expert networks.
- Transfer by AMN: in terms of a baseline to evaluate transfer learning, we then train an AMN for the first two games of each group, based on their DQN experts, and used the resulting AMN network to initialize the DQN for the target game in that group.
- Transfer by PNN: as second baseline, we use the DQN expert networks for the first two games in each group to the train a DQN for the target game by Progressive approach.
- Train the target game by PNN(AMN): finally, we apply our proposal by first training an AMN based on DQN for the first two games in each group and then training a DQN for the target game by Progressive approach.

Note that all the training experiments are done for 10 million steps. Similar to what is done by Parisotto et al. [11], for AMN, each expert network is sampled for 4 million training steps. The rest of the settings are similar to those in the basic DQN paper [10].

Table 1: Game categorization.

Group	Source Game #1	Source Game #2	Target Game
1	Pong	Breakout	Tennis
2	Road runner	Asterix	Krull
3	Seaquest	Star gunner	Centipede

4.2 Evaluation

Transfer learning performance is measured by two measures: 1) Score in the target domain during train, and 2) Number of episodes during evaluation of the target domain. Based on the first measure, in the training phase, the network run an evaluation every 250K steps of training and make the history table of reward for each evaluation run. We use these evaluations to show the trend of training in the baselines and also our method. We also use the number of game episodes which are played by agents through evaluations run during training (i.e., the second measure). We expect to see increasing in rewards and decreasing in number of played episodes while training advances. The reason of decreasing number of played episodes would be the result of more expertise of the network while the training is resuming. Also note that the length of each evaluation run is 125K steps.

Figures 5, 7, and 9 show the average reward per episode during each evaluation run for the target games Centipede, Krull, and Tennis, respectively. For Centipede, both PNN and AMN-PNN provide better results in compare to AMN and DQN (Expert). However, our approach reach to the best possible reward (that is, about 8,000) faster than PNN, and this somehow shows the speed-up in training which is an important goal of this study. For the case of Krull (Figure 7), both Progressive and our method are outperformed by the AMN and DQN, however, there is an outlier result by PNN which cannot be confirmed as a predictable result without further experiment and analysis. Another observation for Krull is that for both AMN and DQN (expert) we can see some increasing trend in their reward as training continues, but this trend is not observable for PNN or AMN-PNN. Likewise, for the case of Tennis (Figure 9), the important observation is the oscillation pattern between negative and zero reward score during the train, which cannot be thoroughly interpreted without further analysis. In discussion section we provide more analysis for this game.

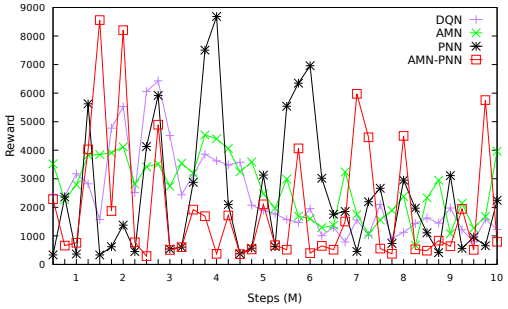

Figure 5: Average reward per episode during evaluation for Centipede

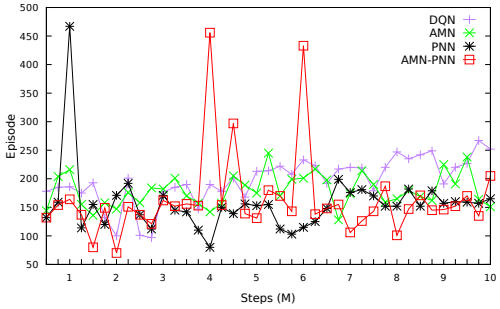

Figure 6: Number of episodes played by agent during evaluation for Centipede

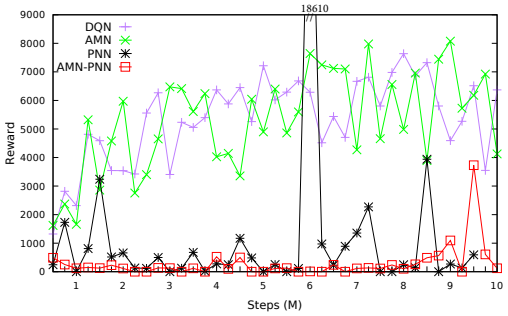

Figure 7: Average reward per episode during evaluation for Krull

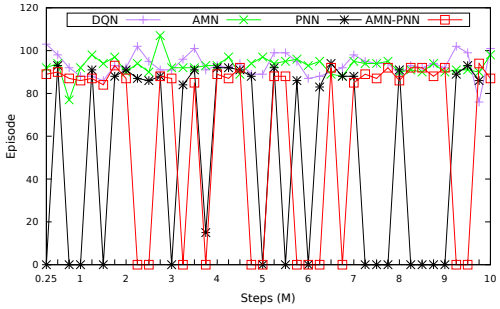

Figure 8: Number of episodes played by agent during evaluation for Krull

Besides analysis based on average reward during evaluation, we provide results on number of game episodes during 125k evaluation steps for each of Centipede, Krull, and Tennis in terms of Figures 6, 8, and 10, respectively. As mentioned earlier, we except to see an agent plays fewer episodes of a game as training advances. For the case of Centipede (Figure 5), we can see some sort of reverse pattern for DQN (Expert) and AMN, as the number of episodes for these two methods first decreases and then starts to increase. For two other approaches, however, there is not a clear pattern and we

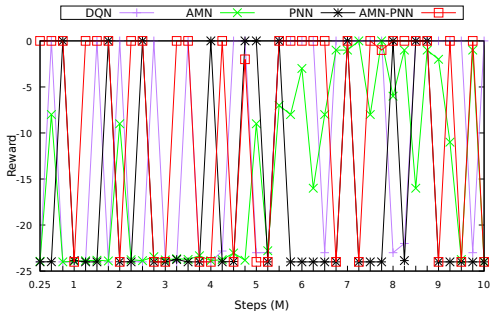

Figure 9: Average reward per episode during evaluation for Tennis

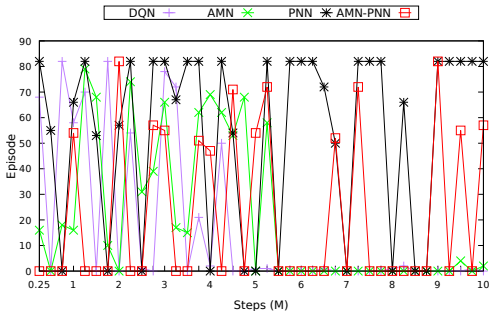

Figure 10: Number of episodes played by agent during evaluation for Tennis

observe many oscillations. For the case of Krull (Figure 8), it seems the number of episodes played by an agent reaches to some stability for DQN and AMN, but for PNN and our approach we observe oscillations without any meaningful trend. Finally, for the case of Tennis (Figure 10), both DQN and AMN converge as training advances and after some point, the agent just plays one episode of the game during 125k steps of evaluation. However, this is not the case for PNN and specially our method (i.e., AMN-PNN).

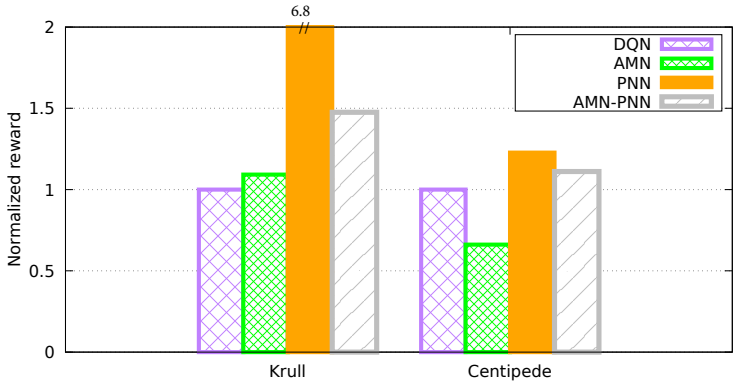

Figure 11: Test scores for games Centipede and Krull for different learning transfer methods

As the last experiment in this section, we report the result of testing different methods for target games of groups 2 and 3[2]. Here, we report the average score (reward) per 10 different episodes of a game for different approaches. Figure 11 shows the result of this evaluation. Note that here we report the normalized reward, which means, the reward of each method is divided by reward obtained for DQN in each group. Based on this analysis, PNN outperforms other approaches, however, our solution is the second best and specially for the case of Centipede, it's result is comparable to the reward obtained by PNN.

4.3 Discussion

In this section, we further discuss about the results obtained for different groups of the games in detail.

Group 1: Tennis. Almost non of the approaches show any obviously meaningful trend of obtained rewards during the evaluation (see Figure 9). However, some pattern of stability may be observed for DQN and AMN after about 5 millions training steps, where their reward is getting closer to 0. Similarly, for the number of played episodes during the evaluation, the only observable pattern (trend)

[2]Here we chose to not report the results for Tennis, as the best possible score obtained for this game after the train is 0 by all the approaches.

is that DQN and AMN agents just play one episode of the game after 5 millions training steps. After training agents by DQN and AMN, we performed some tests by visualization to understand why the behavior of agents is not as we expect. After this analysis, we realized the trained agent by either of DQN or AMN, chooses to play indefinitely by refusing to serve, similar to what is reported in [1], where they described the properties of the ALE framework.

Group 2: Krull. As described previously, both AMN and DQN provide some meaningful trend of increasing reward during train (see Figure 7). However, the number of played episodes during the evaluation is almost the same always for AMN and DQN (see Figure 8). On the other hand, non of PNN or our approach suggest any meaningful trend based on either of evaluation measures. Regardless of the evaluation metrics during the train, the testing results as presented in Figure 11 show how both PNN and our method outperform DQN and AMN. This, in some sense, can show how the transfer by progressive-based approach can help to have a batter agent. One interesting observation of Krull game by visualizations is that the PNN and our method tend to play longer with lower slope of gaining rewards which finally outperform both DQN and AMN. This can explain the lower evaluation rewards during training since evaluations is limited to 125k steps. Moreover, learning transfer from first two games in second group (Road runner and Asterix) results into very well avoidance policy in Krull games which is a main factor for playing longer in all three games.

Group 3: Centipede. Unlike two other target games, for the Centipede we can see both PNN and our approach outperform DQN and AMN based on reward during the train (see Figure 5). More specifically, we can see that AMN-PNN can reach to maximum possible reward sooner than the PNN, where it can save about 2 million training steps to reach to this point. This observation can show and support that such combined architecture helped to speed up the learning process. Moreover, the normalized reward during the test shows the superiority of PNN and our approach in compare to DQN and AMN, where our result is also comparable to one obtained by PNN (see Figure 11).

Complementary Notes. It worth mentioning that most of the baselines like DQN, AMN, or PNN performed their experiments by about 50 million training steps, however, for the lack of the time and resources, we set the training to 10 million steps. This may effect our results, as some of the pattern can be observed after sufficient training time. For instance, the original PNN reported some reward history during the train for the game of Centipede which is not much similar to our results [14]. Another point to be mentioned again is that we developed our own, simplified version of progressive network, because there was no publicly available implementation of this method. In this way, the behavior and result of the original approach may have some differences with what we present in this work. As the last note, we point to this fact that training PNN and AMN-PNN is slower than DQN and AMN, due to the high computation costs as result of their adaptation on top of the DQN. Thus, the original implementation of PNN which is based on AC3 framework [9] can be faster and also using such implementation may speed up our training process as well.

5 Conclusion and Future Work

A typical goal for transfer learning algorithms is to utilize knowledge gained in a source task to learn a target task faster. In this paper we propose AMN-PNN, a novel architecture for training a single deep policy network with transferring knowledge from a set of similar or related tasks. We have shown that a network trained using AMN-PNN is capable of outperforming other baselines (e.g., DQN and AMN), in terms of reaching to maximum reward during the train, faster than other approaches, for some of Atari games like Centipede. Generally, in our experiments we observe that progressive-based architecture outperforms other strategies in terms of evaluation results, and the fact that we leverage a nested (or combined) architecture where progressive idea is a part of that, helps us to boost up the learning performance to some extent. As future plan, we will study the proposed architecture based on more extensive experiments using more available Atari games to find out it's disadvantages and then try to improve them. Moreover, we plan to specifically work on the game of Tennis to resolve it's unexpected training behavior and then train a decent agent for that.

References

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

[2] G. Cuccu, M. Luciw, J. Schmidhuber, and F. Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–7. IEEE, 2011.

[3] F. L. Da Silva and A. H. R. Costa. Transfer learning for multiagent reinforcement learning systems. *the International Joint Conference on Artificial Intelligence*, pages 3982–3983, 2016.

[4] F. L. Da Silva, R. Glatt, and A. H. R. Costa. Simultaneously learning and advising in multiagent reinforcement learning. *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 1100–1108, 2017.

[5] Deepmind. `https://github.com/deepmind/dqn`, 2009-2016. Implementation of DQN in Torch, Accessed: 2017-12-09.

[6] Eparisotto. `https://github.com/eparisotto/ActorMimic`, 2015-2017. Imeplementation of ActorMimic in Torch, Accessed: 2017-12-09.

[7] G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496. ACM, 2006.

[8] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013.

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

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[12] L. Y. Pratt, J. Mostow, C. A. Kamm, and A. A. Kamm. Direct transfer of learned information among neural networks. In *AAAI*, volume 91, pages 584–589, 1991.

[13] J. Ramon, K. Driessens, and T. Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. *Machine Learning: ECML 2007*, pages 699–707, 2007.

[14] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[15] P. Ruvolo and E. Eaton. Ella: An efficient lifelong learning algorithm. pages 507–515, 2013.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[17] M. E. Taylor and P. Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM, 2007.

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