Improving DQN Training Routines with Transfer Learning

George He [*1] **Daylen Yang** [*1] **Kelly Shen** [*1]

Abstract

In this paper, we examine the application of transfer learning in the context of deep reinforcement learning. Specifically, we apply transfer learning to different DQN and IMPALA architectures in the context of Atari games Breakout and Pong. Our goal is to determine how a pre-trained Deep Q-Network from one game can be used to improve learning outcomes in other game settings. We show that transfer learning of low level layers provides noticeable boosts to initial learning performance for DQN, but inconclusive results for IMPALA.

1. Introduction

Reinforcement learning is an area of machine learning concerned with how agents can take good actions in unknown environments. During the learning and evaluation process, the model and architecture must consider factors such as long-term/delayed rewards, and trade-offs between exploiting known actions versus exploring new or rarely tested ones. Often times, models in reinforcement learning require long training times and may diverge from an optimal agent, resulting in agents that underperform or whose actions produce high variance reward values during training.

In this context, we explore Deep Q-learning in different Atari games. Q-learning, a learning method that attempts to learn a value function for state-action pairs, has been shown to perform well in generalized settings. However, basic Q-learning approaches such as linear function approximators do not work well in more complex environments such as those in Atari games. To solve this, Deep Q-learning represents the Q-function using a deep neural network that takes as input a raw state (e.g. pixels from a game) and outputs the Q-value for each possible action. Having been shown to work well on Atari games in the past, we explore if modified training routines and architectures can lead to better learning performance in different game settings.

In this work, we focus on the Atari games Breakout and Pong. In both games, the main concept is moving a paddle towards a ball in order to maximize a scoring objective. Specifically, we investigate how transfer learning on deep networks, a technique heavily explored in image recognition tasks (He et al., 2015), applies to the aforementioned games. Transfer learning aims to take a pretrained set of weights on task A and use those weights to reduce the training time to learn task B. As an example, given known weights that are pretrained on the ImageNet dataset, it is possible to tailor a task-specific or environment-specific image recognition model with relatively limited training. The theory behind transfer learning is that since the earlier layers of a convolutional neural network pertain to recognizing local patterns in images, the same filters remain relevant even when the learning task is altered. Transfer learning in different contexts can considerably improve training times, allow for faster and more stable convergence, and reduce the need for large dataset collection.

Our second focus is to identify how transfer learning applies across different architectures, and to different transferred layers. We explore across architectures with varying depths of convolutional layers, while transferring different layer weights.

2. Related Work

2.1. Reinforcement Learning

In *Reinforcement Learning: An Introduction* (Sutton & Barto, 1998) by Sutton and Barto, learning agents adapt to take sequential actions with the goal of maximizing an often delayed reward signal. To tackle environments with larger state and action spaces, more generalized reinforcement learning architectures have been developed, such as the deep q-learning network (DQN) (Mnih et al., 2013) which learned how to play multiple Atari games and surpass human records. By utilizing convolutional layers (LeCun, 2019) commonly applied to image recognition tasks due to their ability to discern higher level features, coupled with experience replay enabling agents to learn more information from the same dataset, advancements in deep q-learning have allowed agents to learn game features without manually hand crafting them. Deep q-learning architectural improvements have been furthered (Hessel et al., 2017) by the introduction of double-q-learning, which provides the agent with a stable target network to optimize towards, and dueling networks, which help learn more informed state and action values.

[*]Equal contribution [1]Stanford University. Correspondence to: George He <georgehe@stanford.edu>.

2.2. Transfer Learning

Transfer learning allows the domains, tasks, and data distributions used in different training and testing routines to learn from each others' similarities. The fundamental motivation of transfer learning focuses on the need for machine learning methods that are able to retain and reuse previously learned information (Thrun & Pratt, 1998). In this definition, transfer learning aims to extract the knowledge from one or more source tasks and applies the knowledge to a target task. This is in contrast to multi-task learning (Caruana, 1997), which aims to learn all of the source and target tasks simultaneously.

We observe that Pong, a relatively simpler game in terms of objective and game mechanics in comparison to Breakout, is able to converge rather quickly. We are interested in determining if transferring the convolutional layers from Breakout to Pong will improve learning performance outcomes, as Breakout must handle more complex game mechanics than Pong. We are also interested in determining if the simpler representation of Pong will allow the Breakout environment to quickly learn and adapt.

3. Environment Overview

We have chosen the initial environments after evaluating the similarity between games Pong and Breakout. In particular, Breakout was initially influenced by the wild success of Pong, wherein both games involve controlling paddles to hit balls. Due to the different orientations of the games, it will be interesting to see how the learning models adapts as different layers are transferred from one agent to another.

3.0.1. Pong

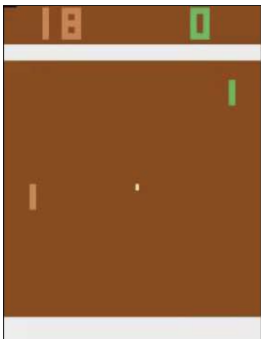

Figure 1. Atari Pong

Pong is a table tennis sports game in which the player or agent controls a paddle located across the right side of the screen that interacts with a bouncing ball. The game is played in a closed environment where the ball bounces off the bounds of the table. Players attempt to score by moving the ball past the opposing paddle. The pong scoring objective is defined as the difference between the player's score (or lives) and the opponent's score.

3.0.2. Breakout

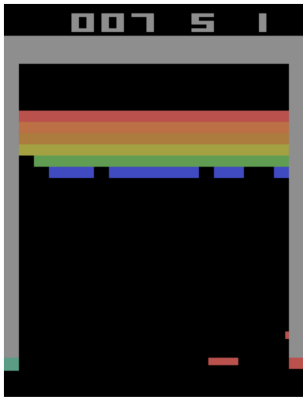

Figure 2. Atari Breakout

In Breakout, the player or agent controls a paddle located across the bottom of the screen to hit a ball, deflecting it upwards into the center of the game screen. The objective is to use the ball and paddle to interact with a series of bricks located at the top of the screen. A ball travels across the screen, bouncing off the top and side walls of the screen, just as in pong. When a brick is hit, the ball bounces away and the brick is destroyed, and points are earned. The player loses a turn when the ball touches the bottom of the screen.

4. Approach

4.1. DQN

We have set up a game environment using OpenAI Gym (Brockman et al., 2016), a toolkit for developing and comparing reinforcement learning algorithms. We use an architecture from which we can easily plug and play various games and load in saved modules, selectively resetting weights. We model our implementation to follow a double dueling deep q-network with experience replay and fixed targets (Graetz, 2018).

4.2. Q-Learning

Q-learning is the process of learning a mapping from state s to the optimal action a. In particular, we define the optimal action-value function $Q^*(s, a)$ as the maximum expected return upon visiting state s and taking action a, which equals the current reward r in addition to the discounted maximum future returns in next state s':

$$Q^*(s, a) = r + \gamma \max Q^*(s', a') \tag{1}$$

Deep Q-Learning is then a specific class of Q-learning that uses a neural network to learn θ in an approximation $Q(s, a; \theta)$ for $Q^*(s, a)$.

4.3. Experience Replay

Consecutive samples have strong correlations, as on-policy learning means that the current parameters directly influence the next data sampled, which are then used to further train the parameters. This leads to inefficient learning and unwanted feedback loops that can result in parameters stuck in local minimums. Replay memory aims to mitigate both these issues by storing the most recent sampled (a, s') in the replay buffer rather than learning immediately from it, instead drawing a random minibatch from the buffer to train on.

4.4. Fixed Targets

When performing the gradient descent step, we aim to minimize loss

$$L = 0.5(Q_{pred} - Q_{target})^2 \tag{2}$$

where $Q_{target} = r + \gamma \max Q(s', a'; \theta)$ and is calculated using the same parameters used to estimate Q_{pred}. This can lead to instability as Q_{pred} is regressing towards a moving target; the target can be "fixed" by introducing a second network, with parameters only periodically updated, to estimate target Q-values. Thus, we have main network parameters θ_{main}, and the periodically updated θ_{target} used to calculated Q_{target}.

4.5. Double Q-Learning

Double Q-Learning aims to mitigate maximization bias, which leads to unrealistically high Q-values. Specifically, maximization bias comes from the expectation of a maximum being greater than or equal to the maximum of the expectation. To prevent this, two separate sets of weights are used to estimate the next state values. We use θ_{main} to estimate which next action a' is best, and θ_{target} to estimate the Q-value of that selected action. As the two networks have different noise, the bias towards larger noisy Q-values cancels. The final Q-value update is:

$$a'_{main} = \arg\max_{a'} Q(s', a'; \theta_{main})$$
$$Q_{target}(s, a) = r + \gamma Q(s', a'_{main}; \theta_{target})$$

The main network parameters are then periodically copied over to update the target network.

4.6. Dueling Networks

The dueling network architecture has the same low-level convolutional structure as a normal DQN, but then splits the final convolutional layer into separate arms predicting the value function $V(s)$ and advantage function $A(s, a)$, rather than a single fully-connected layer directly predicting Q-values. Dueling networks have been shown to improve training stability (Wang et al., 2015) because they adds additional information about the composition of action-values during estimation. Dueling architectures learn the state-value function efficiently because every Q-value update involving state s additionally updates the value stream at $V(s)$. In contrast, a single-stream architecture updates only the value corresponding to the one action taken, leaving the remaining action values untouched.

Below, $V(s)$ estimates how good being in state s is, while $A(s, a)$ estimates how good performing action a in state s is, and $Q(s, a)$ estimates the total value of being in state s and performing action a. The value and advantage functions are them combined into the final Q estimate as:

$$Q(s, a) = V(s) + A(s, a) - 1/|A| \sum_{a'} A(s, a') \tag{3}$$

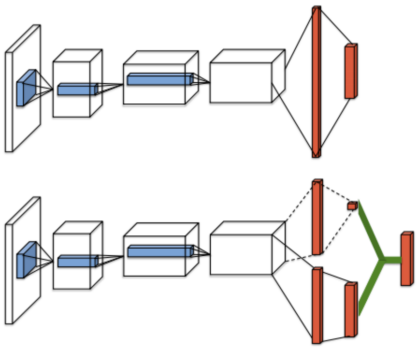

Figure 3. Standard DQN (top) vs Dueling DQN (bottom) (Wang et al., 2015)

4.7. IMPALA

One shortcoming with the approaches outlined above is that training is not parallelized across multiple machines, and thus does not take advantage of modern hardware setups. The Importance Weighted Actor-Learner Architecture (IMPALA) agent (Espeholt et al., 2018), which is an improved version of the Asynchronous Advantage Actor-Critic (A3C) method (Mnih et al., 2016), addresses this.

In the A3C training routine, there is a parameter server and multiple actors. Each actor individually performs exploration and periodically shares gradients back to the parameter server. In IMPALA, the actors do not perform gradient computations and instead solely collect experience, which is sent to a central learner. However, this makes the actors lag behind the learner, so the IMPALA agent also introduces an off-policy correction called V-trace, allowing experiences from off-policy routines to be incorporated into the current policy's updates. We observe that IMPALA has higher throughput than other actor-critic agents, so with enough workers it trains models on the Atari environment very quickly. Application of transfer learning to IMPALA will allow us to observe the effects of transfer learning in an architecture where off-policy exploration is a much larger factor than in the standard DQN architecture.

4.8. Transfer Learning

Our hypothesis is that since Pong and Breakout have similar objectives of deflecting a ball-like object using an agent-controlled paddle to score points, transfer learning should speed up convergence and improve model performance. We experiment with transferring the saved weights from the convolutional layers of an agent trained from scratch on one environment onto a different environment.

5. Results

5.1. Experiments

5.1.1. Environment

In order to allow the agent to recognize the game state at any point in time, the state space will consist of four input frames, as noted in the original Atari Deepmind paper (Mnih et al., 2013). This is because, for example, from a single frame of the game Pong, the agent cannot discern in which direction the ball moves. From a stacked sequence of frames, the agent is able to detect factors such as the direction and speed of movement.

In the OpenAI Gym environment, a frame returned by the environment has a (210,160,3) shape, representing a 210x160 pixel sized screen with 3 RGB color channels. We then transform each frame into a (84,84,1) grayscaled frame. This allows us to quickly process data and play games much faster than if we processed games in the original input space.

Our various agents are built from the Ray (Moritz et al., 2017) implementation of double dueling DQN and IMPALA (Espeholt et al., 2018), as well as an open sourced agent and environment based on Deepmind's paper (Graetz, 2018).

We apply transfer learning across the convolutional layers of three architectural configurations. We focus on transferring the convolutional layers because feature detection between games is agnostic to game control mechanics. As explored by Asawa et al.(Chaitanya Asawa, 2017), transferring all layers (including affine layers) resulted in poor training performance, and ultimately resulted in models that were not able to converge during the training period.

4-conv DQN: We use a 4 layer convolutional stack followed by a value-advantage layer prior to computing Q values, and a replay batch size of 50K. The conv layer configurations are as follows:

```
conv1: filters=32, kernel_size=[8, 8],
stride=4
conv2: filters=64, kernel_size=[4, 4],
stride=2
conv3: filters=64, kernel_size=[3, 3],
stride=1
conv4: filters=6, kernel_size=[7, 7],
stride=1
```

3-conv DQN, 3-conv IMPALA: We use a 3 layer convolutional stack with a larger kernel size for the 3rd layer, followed by a value-advantage layer prior to computing Q values. The conv layer configurations are as follows:

```
conv1: filters=16, kernel_size=[8, 8],
stride=4
conv2: filters=32, kernel_size=[4, 4],
stride=2
conv3: filters=256, kernel_size=[11, 11],
stride=1
```

We compare training time, performance, and convergence of the following experiments, using pre-trained weights on Pong as a seed for training Breakout (and vice versa):

- **Baseline:** Re-initialize and retrain all layers (no transfer learning)
- Transferring weights from certain convolutional layers and re-initializing other layers as normal
- Training on varying convolutional layer depths (3 vs. 4 layers for DQN)

5.2. 4-conv DQN

In this environment, we experimented with training from scratch, transferring all convolutional weights, and transferring the convolutional weights from just the first layer. We observe that transferring the convolutional weights from the first convolutional layer provided the best improvement to evaluation performance, compared to the other methods.

5.2.1. Pong 4-conv DQN

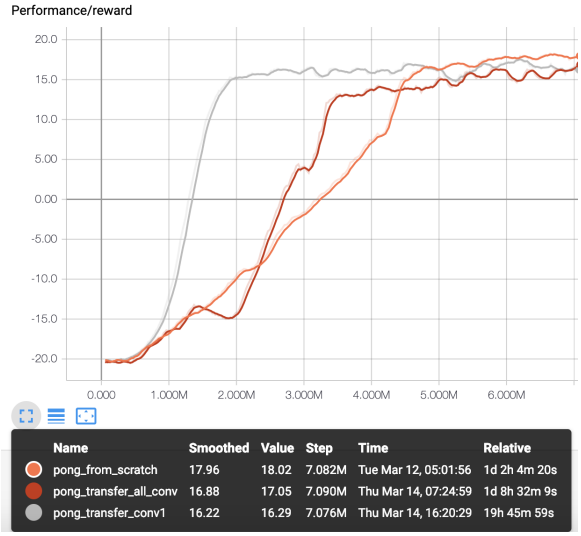

Figure 4. Pong on 4-conv DQN architecture; comparison between no transfer (orange), transferring first conv layer only (grey), and transferring all conv layers (red)

We observe that initializing the first convolutional layer with weights learned from Breakout allows training on Pong

to converge in significantly fewer iterations than when transferring all convolutional layers or when training all layers from scratch. In particular, when we transfer the first convolutional layer only, the agent is able to achieve the asymptotic mean reward within 2M frames of iterations, compared to similar convergence behavior starting 3.5M iterations when transferring on all convolutional layers, or approximately 4.5M iterations with no transfer learning at all. Transferring weights from only the first convolutional layer of Breakout to Pong resulted in an almost **2x speedup** in convergence compared to the vanilla training method. However, both models achieve the same max mean reward.

5.2.2. Breakout 4-conv DQN

We observe improvements in training behavior when transferring Pong weights to assist with Breakout training, although the advantage is less pronounced. When the training routine transfers just the first convolutional layer, Breakout achieves a higher reward earlier; at 10M iterations, the conv1 experiment reaches a reward of around 130, while transferring all convolutional layers reaches a reward of around 110 and training from scratch around 90. Although less pronounced, these experiments illustrate that transferring the convolutional layers from Breakout to Pong does provide a noticeable boost in convergence.

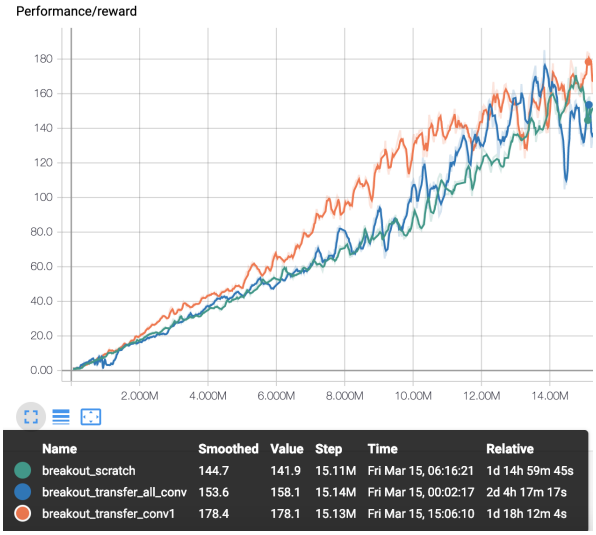

Figure 5. Breakout on 4-conv DQN architecture; comparison between no transfer (green), transferring first conv layer only (orange), and transferring all conv layers (blue)

5.3. 3-conv DQN

For 3-convolutional layer DQN experiments, we used the Ray environment (Moritz et al., 2017). Note that iterations are not comparable between 4-conv and 3-conv experiments due to differing learning behaviors caused by the different architectures. We experiment with training from scratch vs transferring all convolutional weights, and found that transfer learning led to noticably improved convergence characteristics in both environments.

5.3.1. Pong 3-conv DQN

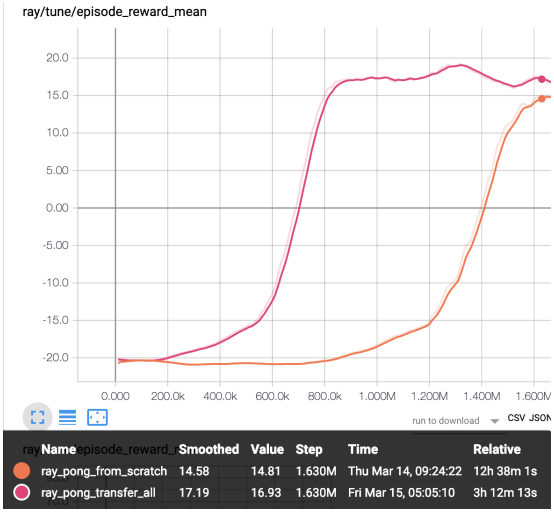

Figure 6. Pong on 3-conv DQN architecture; comparison between no transfer (orange) and transferring all conv layers (pink)

We observe an earlier convergence when transferring all convolutional layers vs training from scratch in the 3-conv Pong experiments. Specifically, learning after transferring all weights converges at roughly 800K iterations, while the no transfer counterpart is still learning at 1.6M, twice the number of iterations later. Once again we see training converge to an optimum score approximately **2x faster** when we transfer the convolutional weights. However, both models achieve the same max mean reward.

5.3.2. Breakout 3-conv DQN

When we transfer the convolutional layer weights from Pong to improve Breakout training, we see a clear improvement in training characteristics: the transfer learned model achieves a reward of 240 at 3.5M iterations while the model that had to learn from scratch only has a reward of 100 at that time. However, neither model is able to achieve higher

mean rewards on Breakout.

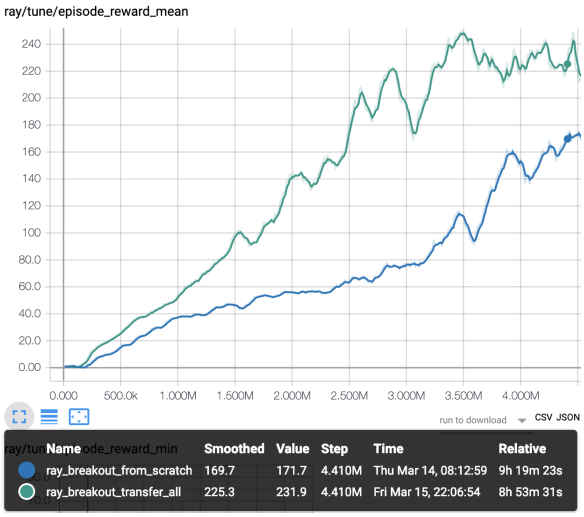

Figure 7. Breakout on 3-conv DQN architecture; comparison between no transfer (blue) and transferring all conv layers (green)

5.4. 3-conv IMPALA

As with the 3-conv DQN section, we trained IMPALA agents on Pong and Breakout from scratch, and with transferred convolutional layer weights. Vanilla IMPALA with no transfer learning trains models very quickly due to its distributed architecture, and in Breakout is able to achieve higher mean rewards than the DQN counterpart. However, we observe that transfer learning does not work as well in the distributed IMPALA environment compared to the DQN agents. For these experiments we use 4 Tesla V100 GPUs and 128 CPU cores.

5.4.1. Pong IMPALA

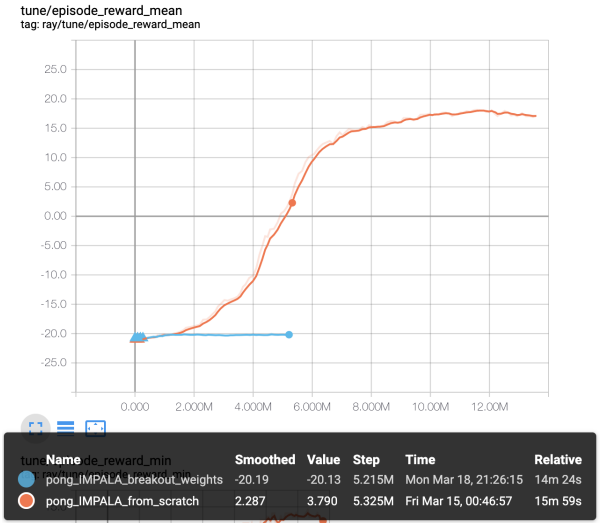

Figure 8. Pong on 3-conv IMPALA architecture; comparison between no transfer (orange) and transferring conv layers (light blue)

IMPALA reaches an asymptotic mean reward of 16-17 in about 22 minutes or 9.3M iterations (again, note that the iteration count is not comparable between different agent architectures). However, it appears that performing transfer learning from a network trained to play Breakout causes the Pong IMPALA training to stall. We observe the mean reward climbs slightly until iteration 1.5M and then stops increasing. We experimented with various checkpoints of Breakout weights (weights from the beginning, middle, and end of Breakout training) and transferring just the weights from the first convolutional layer to no further success. All experiments resulted in the same outcome: IMPALA failed to learn the Pong environment once it had already learned the Breakout task. We believe that because the IMPALA Breakout agent achieved much higher mean and max reward in the Breakout game environment compared to the DQN Breakout agents, the weights present in the convolutional layers are over-tuned to the Breakout environment. As a result, when we transfer the IMPALA Breakout weights to a Pong agent, IMPALA fails to adjust to search over the modified convolutional domain with fixed initial learning rates.

5.4.2. Breakout IMPALA

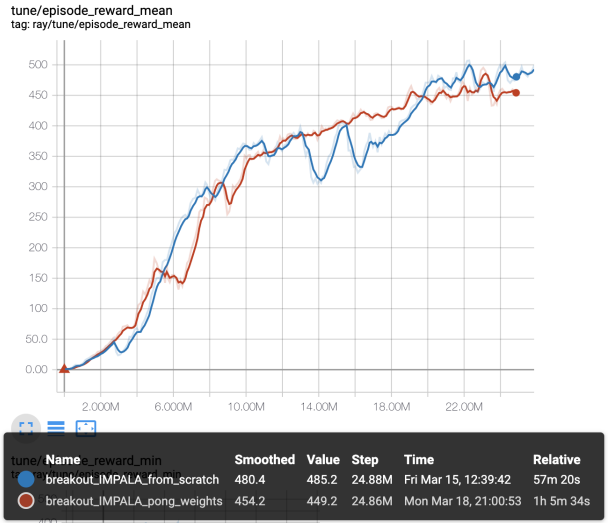

Figure 9. Breakout on 3-conv IMPALA architecture; comparison between no transfer (blue) and transferring conv layers (red)

We observe IMPALA reaches a mean reward of 450 and a max reward of 850 in just 45 minutes or 20M iterations in the Breakout environment. We then transferred the weights from the first two convolutional layers of a Pong IMPALA agent. The resulting reward curve appears to have similar performance as a Breakout agent trained from scratch,

reaching a mean reward of 450 and a max reward of 850 in 19M iterations. We theorize that IMPALA is able to rapidly learn good weights for the convolutional layers in the more complex Breakout environment with both vanilla initialization and transferred convolutional weights from the simpler Pong environment.

6. Conclusion

We have empirically shown that for DQN, transferring the weights from either the first or all convolutional layers, depending on the total number of convolutional layers in the network, helps improve training time. We see that transferring different convolutional layers can yield better convergence criteria, depending on the complexity of the architecture and similarity of the transference domains.

While we have shown better convergence behavior, asymptotic performance was not noticeably improved, which suggested that transfer learning is a good candidate to incorporate into models and routines that attempt to minimize poor decisions in the training process, as in the case of safe reinforcement learning and few-shot learning.

For IMPALA, we have shown that transfer learning does not provide any benefit over training from scratch. In fact, for Pong, transferring the convolutional weights from Breakout actually causes training to stall. Further work is required to examine this discrepancy.

7. Future Work

As we have shown success in transfer learning in the context of games Pong and Breakout, we would like to expand the pool of games and explore fine tuning approaches in transfer learning contexts to determine game mechanics and environments that make transfer learning difficult. Additionally, we believe that transfer learning combined with multi-task learning models could lead to few-shot learning models that are able to quickly adapt to new environments.

Adjusting learning rates between different game contexts is also a point of interest, as the weight domain that transfer learning agents must explore is noticeably different than a randomly initialized domain.

8. Acknowledgements

We would like to thank the CS234 course staff for providing resources through Amazon Web Services and support throughout this project.

9. Code

Our training code for the 4-conv DQN can be found at https://github.com/Georgehe4/transfer-learning. Our training code for the 3-conv DQN and IMPALA experiments can be found at https://github.com/daylen/ray/ in the pong and george_pong branches.

References

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

Caruana, R. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997. ISSN 0885-6125. doi: 10.1023/A:1007379606734. URL https://doi.org/10.1023/A:1007379606734.

Chaitanya Asawa, Christopher Elamri, D. P. Using transfer learning between games to improve deep reinforcement learning performance and stability. http://web.stanford.edu/class/cs234/past_projects/2017/2017_Asawa_Elamri_Pan_Transfer_Learning_Paper.pdf, 2017.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

Graetz, F. How to match deepmind's deep q-learning score in breakout. https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-a 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

LeCun, Y. Deep learning hardware: Past, present, and future. In *IEEE International Solid- State Circuits Conference, ISSCC 2019, San Francisco, CA, USA, February 17-21, 2019*, pp. 12–19, 2019. doi: 10.1109/ISSCC.2019.8662396. URL https://doi.org/10.1109/ISSCC.2019.8662396.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Paul, W., Jordan, M. I., and Stoica, I. Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889, 2017. URL http://arxiv.org/abs/1712.05889.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998. doi: 10.1109/TNN.1998.712192. URL https://doi.org/10.1109/TNN.1998.712192.

Thrun, S. and Pratt, L. (eds.). *Learning to Learn*. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0-7923-8047-9.

Wang, Z., de Freitas, N., and Lanctot, M. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL http://arxiv.org/abs/1511.06581.