Benefits of Transfer Learning
Final Project Report

G11: Lucia Rhode, Mahban Gholijafari, Namitha Guruprasad
due December 11, 2024

1. Abstract

In this project, we explored the application of transfer learning in reinforcement learning (RL), with a focus on its potential to improve learning efficiency and performance in two Atari games, Pong and Breakout. A Deep Q-Network (DQN) was trained on Pong and the learned knowledge (i.e. weights) was transferred to learn Breakout. With different techniques such as frame stacking, preprocessing, and using a confidence factor to balance pre-trained and new weights, we demonstrate the benefits of transfer learning. The results show that transfer learning improves training efficiency and policy performance in Breakout, compared to training a DQN on Breakout from scratch. Finally, we discuss the implications of these findings, with their limitations, and propose future work. This project highlights the potential of transfer learning in improving performance and efficiency of training RL agents on new, but related tasks. Our code is available in our GitHub repository. [1]

2. Goals & Motivation

In recent years, the integration of deep learning in reinforcement learning has enabled RL agents to take in very large inputs and train agents to make decisions in environments with less manual engineering of the state space. This has led to significant advances in various domains including video games.

In the field of deep learning, transfer learning is a fundamental technique, where knowledge gained through one task is used to improve performance on a related task. By leveraging pre-trained features, transfer learning can be used to significantly reduce the time, resources, and data required to train on a related task and address domain gaps.

This leads to our natural question, whether transfer learning can be beneficial in deep reinforcement learning? This project explores whether an agent can leverage knowledge gained from one environment to accelerate learning or improve performance in a related environment. Deep RL suffers from challenges such as sample inefficiency and scalability which makes reducing training time and resources valuable for addressing these challenges.

Building on this foundation, in this project, we want to explore transfer learning by first training an agent on Pong and then transfer the learned knowledge to Breakout. In transfer learning, we leverage the knowledge gained from one task to improve learning performance on a related but different task. We hypothesize that this approach could reduce the training time through faster convergence and improve overall performance by reusing the previously learned features and strategies. We intend to demonstrate that using transfer learning enables the agent to expand its capabilities by adding exploratory options, discovering new strategies, and understanding the game dynamics more effectively and efficiently.

[1] Github Repository Link

3. Relevant Literature

3.1. Atari Games

The Atari games Breakout and Pong are one of the foundational benchmarks in artificial intelligence and RL as their challenging dynamics make them good environments for developing and testing learning algorithms. Breakout tasks a player with controlling a paddle to bounce a ball against a wall of bricks. The objective is to eliminate all the bricks by hitting them with the ball. The player must learn to anticipate the ball's trajectory and the control, timing, and strategic planning necessary to optimize move the paddle and break the wall. Pong is one of the earliest arcade video games and simulates a simplified 2D table tennis game. Players control paddles to hit a ball back and forth across the screen and score points when the opponent fails to return the ball. Like in Breakout, the player must learn to anticipate the ball's trajectory and move the paddle optimally to score on the opponent and prevent the ball from getting past their paddle.

3.2. Q-learning

Q-learning is a traditional reinforcement learning algorithm that finds the optimal policy for finite Markov decision processes (MDP). The agent learns to maximize the total reward by estimated the value function for state action pairs through the following update rule [8].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

The Q-learning algorithm is an off-policy temporal-difference learning algorithm. As described by Sutton & Barto [7] in figure 1, at each time step, the agent will select an action A_t from the observed state S_t using the ϵ-greedy policy derived from $Q(S_t, A_t)$ and observe a reward R_{t+1} and a new state S_{t+1}. The agent will then update the Q-value for that state-action pair $Q(S_t, A_t)$ using the update rule. This is repeated for each episode until the Q-learning algorithm converges to the optimal Q-function.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 1: Q-learning Algorithm

However, the Q-learning algorithm as described above is not sufficient for complex environments with high-dimensional state spaces, such as images. For Q-learning, the Q-value for all state-action pairs needs to be stored in the Q-table. This means Q-learning is not feasible for high-dimensional state spaces due to memory and computational constraints.

3.3. Deep Q-Network

The Deep Q-Network (DQN), introduced by Mnih et al. [5], addresses the memory and computational limitations of Q-learning by combining Q-learning with a deep convolutional neural network. DQNs are able to learn control policies from high-dimensional sensory inputs using reinforcement learning. In particular, DQNs are able to learn value functions directly from raw pixels and outperform previous approaches and sometimes human experts.

DQNs use the semi-gradient form of Q-learning to update the network's weight. The update rule is describe as:

$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t)] \nabla \hat{q}(S_t, A_t, w_t)$$

where w_t is the weights at time step t, R_{t+1} is the returned reward, γ is the discount factor, and $\hat{q}(S_t, A, t, w_t)$ is the estimated Q-value for the current state-action pair, which is the output of the network.

Like Q-learning, DQN employs an ϵ-greedy policy to select the next action.

To stabilize training and allow for efficient sampling, DQN uses experience replay [4] ((cite Lin (1992))). This method requires an off-policy algorithm, which makes the semi-gradient update from the Q-learning ideal. The agent's experience, represented as the state, action, returned reward and next state, at each time step is stored in a replay memory. This replay memory collects many experiences and is then sampled for mini-batches to perform the weight updates.

Additionally, DQN stabilizes model training by using a target model. Periodically, the weights of the DQN model are copied to the target model, which uses these weights to provide the Q-learning targets in the update rule. The target model's output is used for the next state-action pair estimated Q-value, $\tilde{q}$, in the update rule, while the DQN output is used for the current state-action pair estimated Q-value, $\hat{q}$. The update rule is modified as follows:

$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma \max_a \tilde{q}(S_{t+1}, a, w_t) - \hat{q}(S_t, A_t, w_t)] \nabla \hat{q}(S_t, A_t, w_t)$$

The separate target networks helps stabilize training by reducing the risk of erratic oscillation during training.

3.4. Principle Transfer Learning in RL

Transfer learning is a machine learning technique, in which knowledge gain from a source task is applied or used to improve learning in a different but related target task. Early work in transfer learning include domain adaptation and multitask learning [3]. These methods stem from the need to adapt models to new domains and tasks. Transfer learning allows the domain and tasks used in training and testing to be different. Figure 2 shows a simple transfer learning structure.

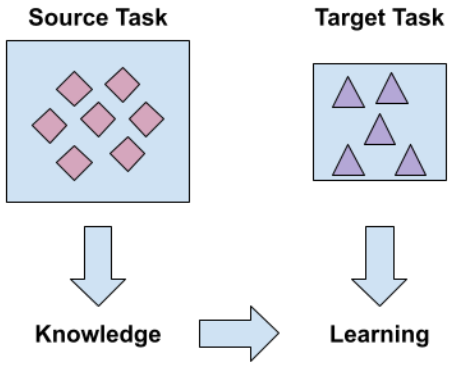


Figure 2: Transfer Learning, adapted from [6]

Two key techniques in reinforcement learning include fine-tuning and feature extraction. Fine-tuning involves initializing the model for the target task by using weights learned from a previous task. Feature extraction involves

freezing the pre-trained layers of the model train on a previous task and training the final layers on the target task. The frozen layers act as the feature extractor.

4. Problem Modeling - Description of Work

In the domain of RL, we seek to train agents to make optimal decisions through interactions with an environment. By leveraging transfer learning, we allow the agents to make use of knowledge from previously learned tasks to accelerate learning in new, related tasks. In our project, we analyze whether transfer learning can be utilized to improve an agent's performance in Breakout by leveraging its prior experience in Pong. Both games share fundamental characteristics, but they also exhibit key differences that influence the modeling of the agent and its learning process. To compare, we also trained DQN from scratch on both games separately. The full game environments can be seen in table 1. Below, we compare and contrast various aspects of the two environments:

- Environment and Agent - The environment features a rectangular playing field with a ball. In breakout, we have a movable paddle at the bottom and destructible bricks at the top. In Pong, we have two paddles on opposite sides. In both environments, the agent controls the paddle's left and right movement to hit the moving ball and aim the ball at a series of bricks (in Breakout) or the opponent's side (in Pong). In Breakout, the agent's goal is to eliminate bricks by directing a ball against them, whereas, in Pong, the agent controls the right paddle to hit a ball back to the opposing left paddle played by the computer.

- State Space - For Pong and Breakout, a single snapshot (frame) of a game is not enough to determine the state of the system. Therefore, we stack 4 frames together to represent the trajectory and velocity of the ball, the location of the paddles, and the remaining bricks (breakout) for the state space. Both games are framed as Partially-Observable Markov Decision Processes (POMDP), since the DQN is trained on a stack of 4 frames and doesn't know the explicit full state (the trajectory and velocity of the ball, the location of the paddles and bricks).

- Action Space - As shown in table 1, Pong and Breakout have different action spaces. Adjustments were made to be able to design a common policy that can be adapted between games. This adjustment will be explained in Sec 5.1.

- State Transition - In both environments, the transition function $T(s, a)$ describes how the state changes in response to an action. In Breakout, the ball's trajectory can be altered by the paddle's position upon contact, whereas, in Pong, the paddle's movement results in the ball's reflection at different angles.

- Reward structure - The reward structure in both games reinforces the agent's objectives and drive the agent to maximize its performance in both environments. In Breakout, the agent receives a positive reward for destroying a bring and a negative reward for allowing the ball to hit the groudn. In Pong, the agent receives a positive reward for scoring a point (when the ball passes the computer's paddle) and a negative reward for allowing the ball to pass its own paddle. The rewards can be summarized mathematically as follows: $R = \{1, -1, 0\}$

- Juxtaposition of Games - Both environments require the agent to utilize its paddle effectively to interact with the ball and achieve their respective objectives. In Breakout, the agent's goal is to destroy all bricks while maintaining control over the ball. In Pong, the agent seeks to prevent the ball from passing its paddle while aiming to score points against an opponent. By applying transfer learning, we hope to demonstrate that an agent trained in one environment can improve its performance or training time in the other, even if the games have slightly different dynamics.

5. Methods

In Q-learning, an action $a = Q(s, a)$ is taken at a state s based on the expected discounted sum of future rewards. With large state-space problems, Neural Networks come in handy as convenient tools for estimating/updating the Q-values and cutting down the computational costs. In our project, we utilize a DQN with ϵ-greedy strategies starting with randomly choosing an action with a probability ϵ or picking the best action after the Q-value estimate with a probability $(1 - \epsilon)$ that decays over time and cutting down the exploration. The DQN leverages experience replay by signaling the main model and the target network to stabilize the training by minimizing oscillations as mentioned in 3.

A DQN was trained to perform well on a source task, Pong, and the knowledge, weights, were transferred as a starting point to train on a target task, Breakout. The effect of transfer learning from Pong to Breakout was evaluated to confirm whether transfer learning on games of a similar nature can help speed up the training time and improve performance over a DQN trained on the target game from scratch.

5.1. State-action space adjustment for Pong

We adjust the state-action space for Pong to match that of Breakout where the actions *LeftFire* and *RightFire* are

Table 1: ATARI Environment

	Pong	Breakout
Environment	(210, 160, 3)	
State Space	Stack of 4 frames	
State Transition	T(s,a), Ball trajectory	T(s,a), Ball reflection in different angles
Action Space	Discrete(4): Noop, Left, Right, Fire	Discrete(6): Noop, Left, Right, Fire, Left Fire, Right Fire
Reward	Score points by destroying bricks	Score points by ball passing the opponent's paddle

mapped to *Left* and *Right* respectively to shrink the redundant actions. This is because *LeftFire* and *RightFire* are combinations of basic actions *Left + Fire* and *Right + Fire* and do not add new behaviors beyond what can already be achieved using simply *Left* or *Right*.

5.2. Framestack

Since Pong and Breakout are POMDPs, a single observation does not fully convey the complete information about all the constituents in the frame. From a single frame, we cannot estimate the direction or the velocity of the ball. An effective way to understand the motion and spatiotemporal dynamics of the objects in the frame is to bundle up consecutive frames as a stack. The DQN captures all the necessary information to solve the POMDPs from the stack by encoding the input representation as features from which they can learn (for example: determining the velocity of the ball as a difference between pixel positions of the ball in the consecutive frames).

5.3. Pre-processing

We crop the unnecessary information in the frames such as scoreboards and margins, and allow the network to focus only on the game-relevant features: the ball and paddles in Pong and the ball, paddle and bricks in Breakout. In addition, cropping the frame size reduces the dimensions and makes training faster without sacrificing any crucial information. Our setup does not require any color information and we can disregard it here which further reduces the complexity of our DQN by cutting down the input data channel information and only focusing on the shape, position, and motion of the objects in the frame(s).

5.4. Design and Details

We design our DQN to have three cascading layers of convolutions with a ReLU activation after each of them in the convolution block, followed by a fully connected block with two linear layers with a ReLU activation inbetween as shown in Figure 3.

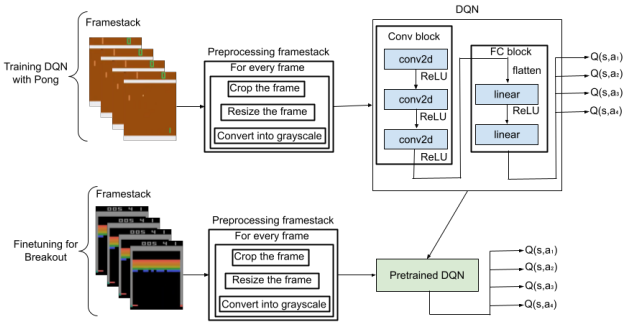


Figure 3: Transfer Learning with Deep Q Network

The DQN first trains on Pong and transfers the knowledge to Breakout by leveraging the learned weights and training for Breakout. We do not fully rely on the weights from Pong for Breakout, but instead use a confidence factor α to control how much influence these weights have during initialization. This allows us to retain and balance the learned knowledge and gives some room for DQN to refine its behavior for Breakout-specific elements and reduce overfitting.

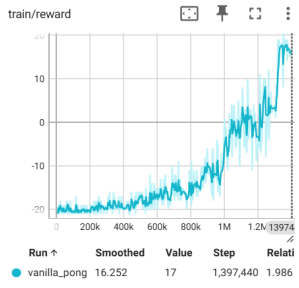
$$Breakout_{init} = \alpha * Pong_{pretrained} + (1 - alpha) * Breakout_{random}$$

We use Open AI's Gymnasium interface [2] for representing the environments of Pong and Breakout. Our network design and training are facilitated by the Pytorch framework [1] for Neural Networks.

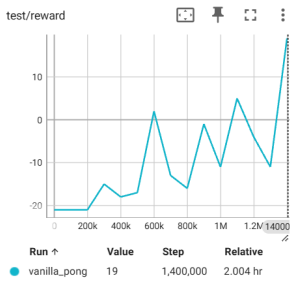
6. Results and Discussion

To analyze the impact of transfer learning, we compare the performance of a DQN trained on Breakout from scratch vs a DQN initialized with weights transferred from pretrained Pong with a confidence factor α and fine-tuned for Breakout. The vanilla training and testing reward curves are shown in Figure 4.

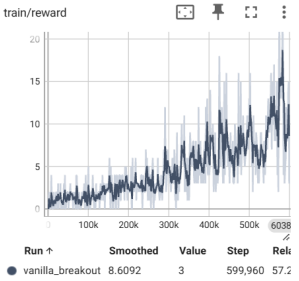
Transferring the knowledge from Pong to Breakout results in a higher cumulative reward at the end of training indicating a better overall performance and improved policy learning as shown in Figures 5 and 6.



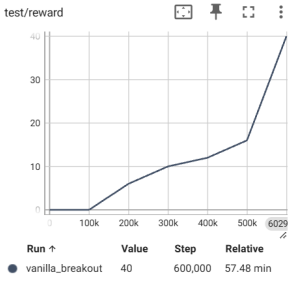
(a) Vanilla Pong train



(b) Vanilla Pong test

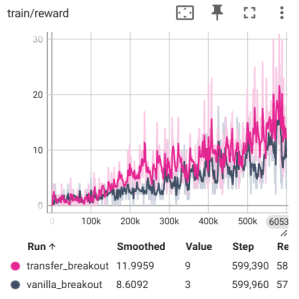


(c) Vanilla Breakout train

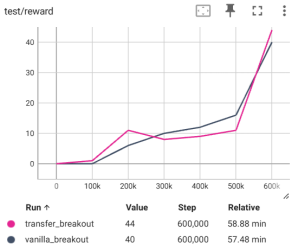


(d) Vanilla Breakout test

Figure 4: Vanilla Breakout and Pong trained separately



(a) Vanilla (black) vs Transfer (pink) Breakout train



(b) Vanilla (black) vs Transfer (pink) Breakout test

Figure 5: Transfer Learning on Breakout vs Vanilla Breakout - test and train curves

The improvement in Breakout shows that the similarities between Pong and Breakout's structures helped the DQN's training. Both games involve tracking moving objects, timing, and directions. The features learned from Pong, for example, firing the paddle to hit the ball, detecting the ball's motion, or avoiding missing the ball, are all effectively used in Breakout and reduce the need to learn the foundational features.

Unlike traditional weights initialization, we applied a confidence factor α to modulate the degree of reliance on pre-trained weights from Pong. This ensures that the transferred knowledge from Pong does not overly constrain Breakout's learning for DQN and allows it to adapt to certain Breakout-specific nuances. $\alpha \in [0.1, 0.9]$ is a hyperparameter that can be tuned,

(a) Rewards earned by Breakout before Transfer Learning

(b) Rewards earned by Breakout after Transfer Learning

Figure 6: Effect of Transfer Learning on Breakout rewards

7. Future Work and Conclusion

In the future, we would like to conduct a thorough ablation of DQN variants such as Double DQN, and Dueling DQN and compare with our existing DQN in terms of performance and stability on Pong and Breakout. Another potential improvement could be to freeze certain layers in the architecture during the transfer phase to retain any low-level features from Pong and adapt only high-level features to Breakout. We are also interested in investigating how our transfer learning approach can be generalized to different Atari games or even non-Atari games by fine-tuning. It would be helpful to incorporate Large Language Models LLMs in the training to analyze the scoreboard that was cropped during preprocessing. This could offer valuable feedback to the DQN and better align with every game's objective although there could be a trade-off between performance and computational parameters. It would be noteworthy to extend this work to multi-agent settings where we can mix up cooperative or competitive learning in a shared environment for complex game scenarios.

In our project, we explored the use of Transfer Learning in learning Atari games, focusing on Pong and Breakout. We trained a simple DQN on Pong and transferred it to Breakout using a confidence factor to guide the weights initialization. Our experiments show that leveraging the knowledge from Pong to Breakout results in an increase in rewards obtained and highlights a promising approach to accelerate the training of RL agents coupled with the effectiveness of fine-tuning for different game environments.

8. Contribution

This project was a collaborative effort. Each member contributed equally to the workload, sharing responsibilities in the proposal writing, code development, troubleshooting problems encountered along the way, and report writing. We worked together in online settings to constantly communicate and effectively collaborate throughout the project.

9. Acknowledgment

Lastly, we would like to extend our gratitude to Professor Mallada and teaching assistant, Agustin Castellano, for their insightful lectures and recitations, as well as feedback during the project proposal phase and support during the office hours.

References

[1] Pytorch.

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

[3] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[4] L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.

[5] V. Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[8] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.