Performance of ACKTR and PPO with Transfer Learning on Atari-2600 Games

Akarsh Gopal
University of Twente
a.gopal@student.utwente.nl

Leon van der Neut
University of Twente
l.m.b.vanderneut@student.utwente.nl

January 14, 2019

Abstract

In this paper, empirical research is conducted on the effects of Transfer Learning on the performance of two state-of-the-art Reinforcement Learning algorithms in a diverse set of tasks. This is carried out using open-source high-quality implementations of the algorithms presented by OpenAI in Baselines (OpenAIa, 2018). This research intends to develop insights into the effects Transfer Learning bears on the performance of these algorithms and the methodology of research that could be followed to conduct studies which hold significance. The results show positive effects of Transfer Learning on particular source and target task combinations, while also showing negative effects on other combinations. In general, the results show that it is not straightforward to predict when transfer learning will prove beneficial for Reinforcement Learning.

Keywords

A3C: Asynchronous Advantage Actor-Critic
ACKTR: Actor-Critic using Kronecker-Factored Trust Region
ALE: Arcade Learning Environment
AI: Artificial Intelligence
AGI: Artificial General Intelligence
CNN: Convolutional Neural Network
DL: Deep Learning
DNN: Deep Neural Network
DRL: Deep Reinforcement Learning
DQN: Deep Q Network
GPU: Graphics Processing Unit
K-FAC: Kronecker-Factored Approximate Curvature
MDP: Markov Decision Process
PPO(2): Proximal Policy Optimisation
RL: Reinforcement Learning
TD: Temporal Difference
TL: Transfer Learning

I. Introduction

Artificial Intelligence (AI), a sub-field of Computer Science (Russell & Norvig, 2003), has been developing rapidly in the last few decades due to the increase in computing power, data collection and breakthroughs made in algorithms since the 1950s (Bostrom, 2014; Kurzweil, 2005). While AI currently casts a spotlight on Deep Learning methods, which are still relatively narrow in their ability to solve problems, the 'final' objective is to design an **Artificial General Intelligence** (AGI) which is defined to be capable of performing intelligently at general tasks much like humans (Baum, 2017; Muehlhauser, 2014). Philosophers and experts in the AI field argue that the advent of AGI would quickly lead to a kind of super-intelligence with an incomprehensible level of intelligence that could lead to either a utopian or dystopian scenario (Bostrom, 2014; Kurzweil, 2005; Muehlhauser and Bostrom, 2014).

Reinforcement Learning (RL), a sub-field of AI that deals with situations where an 'agent' acts in an 'environment' , is mainly aimed at creating such an AGI agent that can be implemented across a wide range of activities (Bellemare, Naddaf, Veness & Bowling, 2013; Sutton & Barto, 1998). The environment provides feedback to the agent and the agent has a particular goal to achieve within the environment. The objective of RL is to develop methods to allow the agent to learn to achieve the goal while acting within the environment.

RL literature uses a mathematical model of the environment called a **Markov Decision Process** (MDP) to develop algorithms that learn how to achieve the maximum reward. Using the physical universe as the environment for the RL agent would be computationally intensive, given the complexity of the model required. Doing so would also lead to ethical issues (Bostrom, 2014). Video games follow respective 'mechanics' or 'physics' that are much simpler than the natural laws that the physical universe seems to follow, as modelled by the scientific field of physics (Einstein, 1916; Hawking & Stone, 1992; Newton, 1934). Video games also inherently have a main goal to be achieved while playing them. Therefore, learning to play video games can be considered analogous to the learning to achieve a certain goal within the physical universe, motivating RL researchers to use video games as environments for algorithm development (Bellemare et al, 2013). Since the Atari-2600 set of video games presents a diverse and relatively simple set of core tasks, the video games serve well as a platform to research the fundamental characteristics of RL (Mnih, Kavukcuoglu, Silver, Rusu, Veness et al., 2015; Bellemare et al., 2013).

Transfer learning (TL) is an approach to use 'competencies' gained by learning a first, 'source' task in improving the learning of a second, 'target' task (Weiss, Khoshgoftaar & Wang, 2016). TL could possibly be used in RL in order to develop and test algorithms that perform well at learning multiple different tasks i.e are generalisable (Laird, Newell, & Rosenbloom, 1987; Taylor and Stone, 2009), which would be necessary in order to progress towards AGI. This is an active area of research and is the topic of this study.

The characteristics shown by an algorithm when challenged with TL can be considered 'virtual-technological phenomena'. This term is derived from 'physical-technological phenomena', introduced by Boon (2018), which is a term for the objects, properties and processes of an instance, either naturally present or created by mankind. In the virtual realm, which is created by mankind, one can think of this same concept with regard to algorithms.

This study focuses on the virtual-technological phenomena displayed by ACKTR (Wu, Mansimov, Grosse, Liao, & Ba, 2017) and PPO2 (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) when challenged with TL on specific Atari-2600 games of varying complexity in order to generate insights into the ability of state of the art algorithms to transfer their learning from one task to another. In the process of doing so, the study also intends to possibly develop insights into solutions to improve this generalisability of the algorithms. **Actor-Critic using Kronecker Factored Trust Region** (ACKTR) and **Proximal Policy Optimization** (PPO) were chosen for this research since they are considered to be the cutting edge of RL at the moment and implement advanced actor-critic architectures. The specific question addressed by the research is "What are the effects of Transfer Learning between games on the ability of ACKTR (Wu et al., 2017) and PPO2 (Schulman et al., 2017) to train agents to play Atari 2600 games of varying complexity?"

The research question is analogous to the situation where a basketball player and a non-athlete are tasked with learning to play football. Intuition would point towards the basketball player being able to pick up the game of football more easily than the non-athlete would.

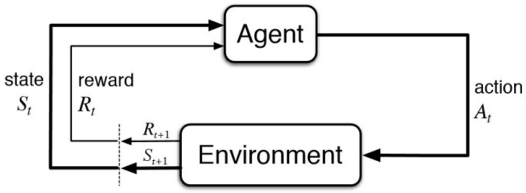


Figure 1: *Diagrammatic Model of Reinforcement Learning (Sutton & Barto, 1998).*

II. Theoretical Background

Reinforcement Learning (RL)

RL is a sub-field of AI that deals with situations where an ‘agent‘ acts in an ‘environment‘ . The agent has a particular goal to achieve and the environment provides feedback to the agent which can be used to progress towards the agent‘s goal. The objective of RL is to develop methods to allow the agent to learn to achieve the goal while acting within the environment.

The model used for an RL system (Sutton & Barto, 1998) is as follows:

An **agent** interacts with an **environment** E over a number of discrete timesteps. At each time step t, the agent receives a **state** s_t and selects an **action** a_t from some **set of possible actions** A according to its **policy** π. Based on the action the agent takes for the particular state, the agent receives the next state $s_{(t+1)}$ and a scalar **reward** signal r_t. The process continues until the agent reaches a terminal state after which the process restarts.

Formally, the RL environment is **modelled as a Markov Decision Process** (MDP) (Bellman, 1957), which is a 5-tuple:$< S, A, R, P, \rho_0 >$, where S is the set of all valid states, A is the set of all valid actions, i.e action space, R is the reward function, P is the state-transition probability function and ρ_0 is the distribution of the initial state. The MDP essentially captures all information regarding the environment and the agent in discrete time-steps such that with each step, there is information about the initial state, the current state of the environment, the action taken by the agent at the current state, the reward received by the agent for this action, the probability distribution of the next state based on the current state and the action taken. This means an MDP is a series of blocks of information with each block depending only upon the previous block

A **state** s is the set of variables which defines the **environment** at a given time. An **observation** o is a subset or equivalent of the state s which is accessible to the agent. Therefore, an observation can also be a partial description of the environment at a given time. When o is equivalent to s, the environment is said to be fully observable, and when o is a subset of s, the environment is said to be partially observable. When formally describing algorithms, in RL literature such as Sutton and Barto (1998), state is used instead of observation. Hence, state s is used for the rest of the discussion although in practice observation o is used.

N.B. While environments can be deterministic or stochastic, for the sake of simplicity and relevance to the specific environments in this research, we shall focus on stochastic environments only.

An **Action Space** is the set of all valid actions in a given environment. Action spaces can be discrete, where action spaces are discrete-valued vectors making the number of valid actions finite. Alternatively, they can be continuous which means action space is a real-valued vector. A **policy** π is the rule used by the agent to take an action given an observation, in order to achieve a goal. A stochastic policy maps an action to a state as $a_t \sim \pi(*|s_t)$

A **trajectory** or an **‘episode‘** τ is a sequence of states and actions in the environment: $\tau = (s_0, a_0, s_1, a_1, ...)$. A trajectory is defined by the policy of the algorithm, ‘choosing‘ the actions, and the state transition model of the environment, where the state transition model P defines the next state given an action-state pair : $s_{t+1} \sim P(*|s_t, a_t)$

The **reward** signal $r_t = R(s_t, a_t, s_{s+1})$ is a signal produced by the reward function of the environment. This reward signal is provided to the agent, to be used for learning.

The **return** is the cumulative reward received by the agent over a **horizon** (number of timesteps in trajectory) which may be finite or infinite depending on the environment and the trajectory. The goal of RL is to select a policy which **maximises expected return**. This can be framed as an **optimisation** problem, wherein a parameterised policy has to be optimised such that the expected return is maximised. Algorithms use approximations of the expected return when optimising their parameters.

A **value function** is such an approximation of the expected return of a particular state or state-action pair. Two main value functions (Kimura and Kobayashi, 1998) are:

On-policy value function approximates the expected return at a state if only a particular policy is followed thereafter:

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)|s_0 = s]$$

On-policy action-value function approximates the expected return at a state if an arbitrary action is taken at a particular state. This is also called the âĂŸQâĂŹ value (Watkins & Dayan, 1992).

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]$$

In order to simplify the optimisation problem, each of these value functions can be written as the sum of reward at the current state for the action taken and the value of the rest of the trajectory (Dixit & Avinash, 1990). This is done using the **Bellman Equations** (Kirk, 1970). The Bellman equations for each of the two value functions previously mentioned are as follows:

$$V^{\pi}(s) = E_{a\sim\pi,\ s'\sim P}[r(s,a) + \gamma V^{\pi}(s')]$$

$$Q^{\pi}(s,a) = E_{s'\sim P}[r(s,a) + \gamma E_{a\prime\sim\pi}[Q^{\pi}(s',a')]]$$

These are used to calculate **advantage function** $A^{\pi}(s,a)$ is used to mathematically describe how much better a particular action a is for a particular state s, compared to randomly selecting an action according to policy . The advantage function is calculated as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

The advantage function is used while optimising the policy in order to decrease variance across updates (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015).

Most state of the art algorithms since Mnih et al. (2015), including the ones being studied, use **Deep Reinforcement Learning** (DRL) (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017). DRL is based on **Deep Learning** (DL), which is an approach to developing universal approximation functions. It helps develop systems that can produce desired outputs from inputs without having to hand-engineer the transformations in between (Goodfellow et al., 2016; LeCun, Bengio, Hinton, 2015; Schmidhuber, 2015). For a more elaborate explanation of DL, please refer to Appendix A.

The ability of DL to approximate complex functions is useful in the field of RL when the observed state is complex and needs to be mapped to an action output. RL algorithms that use Deep Neural Networks (DNN) are called DRL Algorithms. Recent advances in DL applications motivated the application of DNNs in RL such as in Deep Q Network (DQN) (Mnih et al, 2015). Algorithms such as DQN have raised the standards in test-beds such as the ALE (Bellemare et al., 2013). In DRL algorithms, the policies and/or value estimators are neural networks which are optimised in order to maximise an 'objective function'. Specifically, when DRL algorithms obtain images as input, a particular type of DNN called a **Convolutional Neural Network** (CNN) is used. CNNs use a particular architecture and set of hyperparameters that enable them to excel at image-related tasks. For a more detailed explanation of CNNs, please refer to Appendix A.

Hyperparameters are the parameters that determine the architecture of the network used in the algorithm, like the number of layers of the network, and the specifics of the training process, like proportional step size and learning rate. Proper tuning of these parameters is vital to the performance and stability of the algorithm (Goodfellow, Bengio, Courville, & Bengio, 2016; Ng, 2017; Sutton & Barto, 1998).

The algorithms selected for this study perform **Policy Optimisation**. A parameterised policy $\pi_\theta(a|s)$ is to be optimised such that the objective function $J(\pi_\theta)$ is maximised. This can be done by optimising the parameters via gradient ascent on $J(\pi_\theta)$. Alternatively, this can be done by maximising local approximations of $J(\pi_\theta)$. Policy optimisation algorithms also use approximators of value functions and use them in updating the policy (elaboration in Appendix A).

Mathematical Optimisation techniques are used to perform policy optimisation. $J(\pi_\theta)$ is not necessarily a convex function and therefore can be difficult to optimise with simple gradient ascent, which is a first order optimisation method. In order to ensure global convergence for non-convex problems, **Trust Region** methods, which are second order optimisation methods, can be used (Yuan, 2015). (elaboration of 1^{st} *vs* 2^{nd} order optimisation in Appendix A) The following algorithms use Trust Region optimisation or a close approximation thereof in order to ensure a 'safe' step size and direction.

Actor-Critic Architecture

In Figure 2 (Patel, 2017) the general setup of the actor-critic model, used in ACKTR and PPO2, is presented. There are two neural networks present, the actor, using a policy-based learning algorithm, and the critic, using a value-based learning algorithm. Both the actor and critic are parameterised according to the CNN architecture proposed by Mnih, Badia, Mirza, Graves, Lillicrap et al. (2016) in OpenAI Baselines (OpenAIa, 2018) implementation of ACKTR as well as in PPO2.

Actor-critic is analogous to a novice coach and a novice player trying to maximise their combined performance in a particular game. The coach assesses the performance of the player so that the player may improve his skills, while the change in the player's next performance allows the coach to improve her own assessment of the player. Over time, the coach learns to assess more accurately and the player learns to play better. If the coach is able to assess the player rather well, the player can improve at a faster pace.

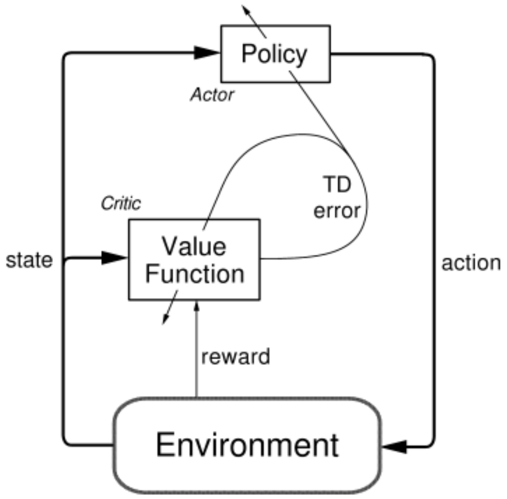


Figure 2: *Diagrammatic Model of Actor-Critic Architecture (Patel, 2017).*

The critic learns to estimate the value of being in a specific state, which is then used to approximate the TD-error and update the value estimation of the critic and the policy of the actor. The Temporal-Difference(TD)-error is the evaluation of the policy of the actor based on the estimation of the reward by the critic and the reward obtained by the policy before the update. TD-error indicates how well the critic can estimate the value of the state based on the actual reward received for the previous action taken by the actor and its estimation of the value of the previous state and its estimation of the value of the previous state. Therefore it is useful in improving both the critic and the actor (Kunz, 2000; Sutton & Barto, 1998).

The benefit of using this architecture is that the actor receives feedback from the critic on each action that it takes, whereas the environment only provides feedback on the set of actions performed by the actor during an episode in the form of win or lose, or points gained through the episode depending on the environment the agent operates in. Therefore, the critic creates a more reward-dense environment for the actor, increasing its sample efficiency (in other words, extract a lot of information from relatively little data). In the advantage actor-critic model, the advantage function is used to minimise the variance of both the value estimation performed by the critic and the policy update of the actor, thereby increasing the accuracy of the agent (Mnih et al., 2016).

Algorithms Researched

ACKTR (Wu et al., 2017)

Actor-Critic using Kronecker-Factored Trust Region (ACKTR, pronounced 'actor') is an algorithm that utilises second-order optimisation to achieve very accurate policy updates and allows the algorithm to achieve a high sample efficiency. ACKTR uses a combination of natural gradient descent (Kakade, 2002), Kronecker-Factored Approximate Curvature (Martens, & Grosse, 2015) and trust region (Schulman et al., 2015) to update its parameters.

Natural gradient descent involves computation of the Fisher information matrix. This information matrix represents how probable the estimations produced by the network are using the covariance of the log likelihood of the estimation given the policy:

$$F = E_{p(x|\theta)}[\bigtriangledown \log p(x|\theta) \bigtriangledown \log p(x|\theta)^T]$$

The inverse of the Fisher matrix, which is required in natural gradient descent, is too heavy to compute because of its sheer size of $[(number\ of\ parameters)^2, (number\ of\ parameters)^2]$, considering that the number of parameters in a convolutional neural network often go beyond million (Wu et al., 2017). To work around this, ACKTR uses Kronecker-Factored Approximate Curvature (K-FAC) (Martens, & Grosse, 2015) in both its actor and critic model, which makes this optimisation computationally feasible. K-FAC uses a number of simplifications to obtain an approximation of the Fisher matrix which works fairly well in practice. The trust region approach of ACKTR ensures that the policy will not adapt too strongly to a single experience by using the approximation of the Fisher matrix to estimate the certainty of the gradient computed to ensure the magnitude of the step-update does not pass the threshold set for uncertainty. ACKTR uses the combination of trust region and K-FAC proposed by Ba, Grosse, & Martens (2016) and brings this optimization method together with the architecture of **Asynchronous Advantage Actor Critic** (A3C) (Mnih, et al., 2016). The A3C architecture is a variant of the earlier explained advantage actor-critic model that allows multiple actors to act in an environment simultaneously while updating only one critic model. This is an adaptation that only has an impact on the efficiency of computation and has no implications for the learning of the agent (Mnih et al., 2016). Therefore, this will not be addressed further.

PPO (Schulman et al., 2017)

PPO is a policy optimisation method that empirically performs at least as well as trust region methods but is simpler to implement since it uses a first-order optimisation method. The algorithm effectively uses the same architecture as ACKTR but uses a novel method for policy optimisation: PPO performs stochastic gradient ascent updates on the parameters of the policy in order to maximise a 'surrogate' objective function (Achiam, Wu, Morales, Marciniak, Bastovanovicz, Hesse, 2018). The OpenAI Baselines (OpenAIb, 2018) implementation of PPO uses the 'clip' approach mentioned in Schulman et al. (2017). The surrogate objective function involves a clipping term that helps keep the updated policy parameters close to the previous policy parameters and therefore prevents large step-changes on the policy parameters that may make learning unstable. This allows PPO to make the largest possible update to parameters with the collected data, without compromising learning stability. PPO bypasses the complexity of trust region methods such as ACKTR using this clipping term, without compromising on performance.

In PPO, the advantage is estimated using a value function and a generalised advantage estimation method (Schulman, Moritz, Levine, Jordan, Abbeel, 2015). The algorithm uses a policy network and a value function network to estimate the surrogate objective function. In the case of the Baselines implementation of PPO for Atari games, the policy is a CNN which takes the screen of the game as input and maps this to an action space.

PPO is an on-policy method that optimises a stochastic policy. At the beginning of the training, the output actions are highly random and this randomness decreases over the training period. This is how the exploration of the state-action-reward space is inherent to the algorithm. However, the policy may get trapped in local optima due to low randomness in action selection in the later stages of training (Achiam et al, 2018).

The Baselines implementation of PPO is called PPO2 because it is a variant that improves performance by using Graphics Processing Units (GPUs) unlike the former, PPO1, which did not do so. PPO will be referred to as PPO2 for the rest of the paper since this is the variant used in the experiments (Achiam et al., 2018).

Environment Used For Study

The **Arcade Learning Environment** (ALE) (Bellemare et al., 2013) is an emulator of the Atari-2600 console for video games and is specifically designed to evaluate different algorithms independent of their domain and provides an interface with a large number of Atari-2600 games for testing RL algorithms (Brockman, Cheung, Pettersson, Schneider, Schulman et al., 2016). OpenAI Baselines (OpenAIa, 2018) is a set of high-quality, open-source implementations of the state of the art RL algorithms and supports the ALE as an environment using the OpenAI Gym toolkit (OpenAIb, 2018), which provides an interface between popular RL environments such as the ALE and programming languages such as Python. OpenAI Baselines allows saving of the parameters of the trained model and loading of these weights as initialisation for other runs. It also has a logging feature that stores the total reward per episode of the agent over the 10M timesteps of the run. OpenAI Gym and the ALE can be considered complementary platforms, designed for the objective evaluation and comparison of algorithms and, ALE specifically, for TL as well (Bellemare, et al., 2013; Brockman, et al., 2016). OpenAI Baselines allows state of the art algorithms to be tested on these platforms Therefore OpenAI Baselines and implicitly OpenAI Gym and ALE were used for the study.

The version of ALE used during experimentation has deliberately implemented stochasticity into the Atari-2600 games (Machado, Bellemare, Talvitie, Veness, Hausknecht & Bowling, 2017) but the Atari-2600 games are deterministic otherwise (Hausknecht & Stone, 2015). The specific source of stochasticity in the Baselines implementation of (game name)NoFrameskip-v0 is called ‘sticky actions‘, a probability that the action of the previous frame is repeated in the current frame instead of the inputted action. In this case, the probability that an action would be ‘sticky‘ was set 0.25 (OpenAIa, 2018), in accordance with the suggestion from Machado et al. (2017).

Transfer Learning (TL)

Transfer Learning is the process of improving learning performance in a particular‘target‘ task having already learned to perform a related, but different, ‘source‘ task (Taylor & Stone, 2009). Psychological literature studies TL in humans and develops ideas analogous to TL in Machine Learning research (Thorndike & Woodworth, 1901; Skinner, 1953).

Collection of training data for physical-world applications can be prohibitively expensive or practically impossible in many cases. TL can help reduce the need and effort to collect this training data in such cases (Pan & Yang, 2010). This has been shown to work well in DL applications such as computer vision and speech recognition (Hoo-Chang, Roth, Gao, Lu, Xu et al., 2016; Weiss et al., 2016). The insight behind TL is that generalisation may occur not only within tasks but also across tasks (Taylor & Stone, 2009). TL research could also become relevant in transferring learning from virtual simulation (source task) to the physical world (target task) applications like robotics, as demonstrated by (Andrychowicz, Baker, Chociej, Jozefowicz, McGrew et al., 2018).

It is also important to note that TL can have a detrimental effect on the performance of the learner and this phenomenon is called negative transfer (Weiss et al., 2016). This can occur in cases where the source task and target task differ significantly. Negative transfer in the context of RL has been recorded by Glatt, Silva & Costa (2016) where the source task and target task were video games with different mechanics. In order to effectively apply TL, the source and target task selection must, therefore, be compatible, i.e there should be at least a non-negative effect of TL.

Challenges in TL that require further research include the handling of negative transfer and the determination of suitability of a source and target task pair for transfer (Glatt et al., 2016). Glatt et al. (2016) defined the similarity of the learned tasks subjectively and stated that this procedure was not appropriate for a general case since a complete understanding of how neural networks generalise tasks is lacking, and human intuition can go wrong in such situations. Therefore, developing a framework for defining the similarity of the tasks is another challenge in TL.

The definition of evaluation metrics for TL is also a challenge because there are several possible metrics (Taylor & Stone, 2009). Taylor and Stone (2007) defined and used the following metrics (represented visually in Figure 3) in a cross-domain TL study:

1. **Jumpstart**: The difference in the initial performance of the agent in the target task with respect to that in the source task could indicate whether the agent has acquired useful priors.

2. **Asymptotic Performance**: The difference between the asymptotic performances of the agent in the two (target and source) tasks could indicate whether the agent reaches a higher final score.

3. **Total Reward**: The difference in the total returns gained by an agent over each of the two tasks could indicate the effect of transfer over the span of the task(s).

4. **Transfer Ratio**: The ratio of the total returns gained by the agent in the two tasks could also indicate the overall effect of TL for the source and target task pair.

5. **Time-to-Threshold**: The difference in the time needed by the agent to achieve a specified performance threshold in the two tasks could indicate whether the speed of learning has changed due to transfer.

Hypothesis

Combining the discussed theory of RL, TL, and the intuition that similar tasks would have similar underlying ‘mechanics‘ such that the algorithms can transfer competencies learned from one task to the other beneficially, we develop our hypothesis:

TL decreases training time (number of time-steps taken for the agent to achieve a threshold of return) for games that are similar when compared to directly training on the target task. Also, we hypothesize that training time for similar games is lesser than training times for dissimilar games.

The results of testing this hypothesis can be compared with the prediction made in the hypothesis to verify or disprove the hypothesis. Since the phenomenon being considered for the study is a virtual-technological phenomenon and is essentially a simulation in and of itself, our hypothesis makes a claim regarding the performance of the model (agent) in the larger system (environment(s)).

III. Methodology

The method we developed to test the effects of TL on the performance of algorithms for combinations of games is as follows:

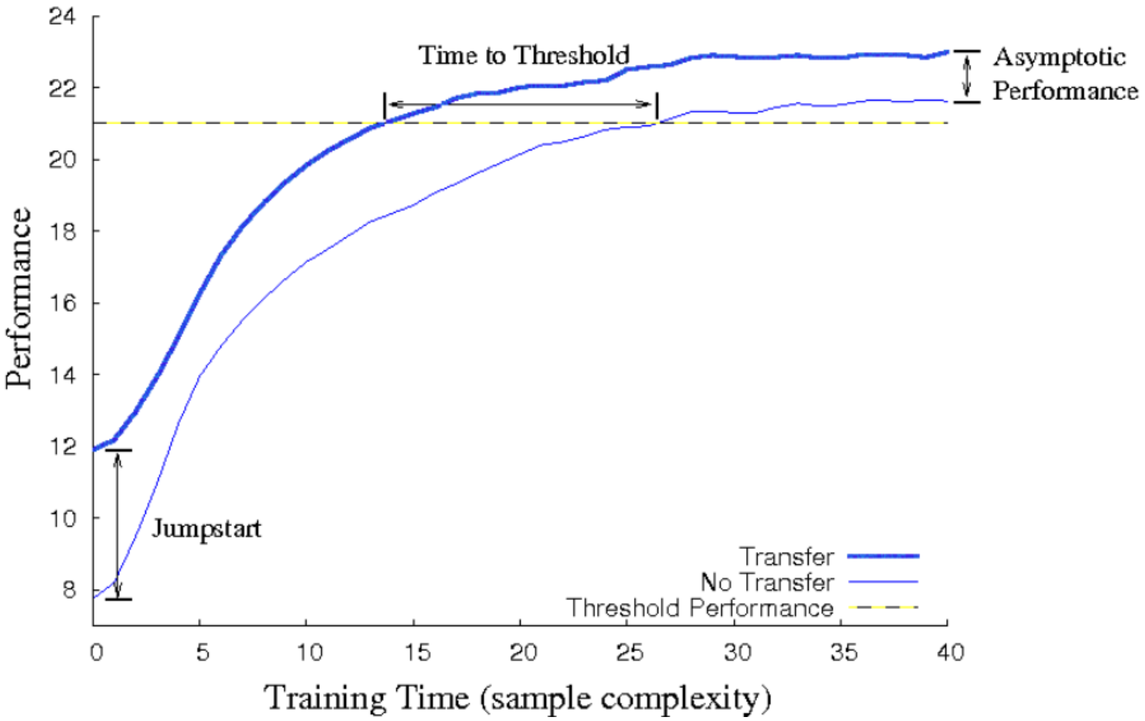


Figure 3: *Possible Metrics of TL in RL (Taylor & Stone, 2009).*

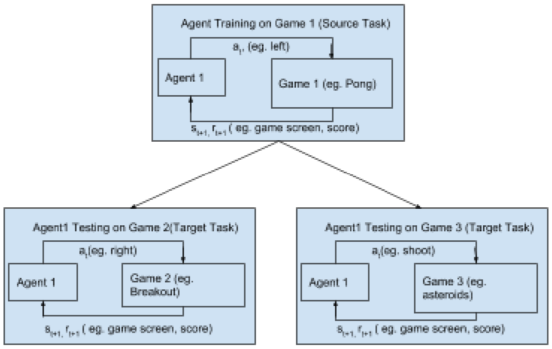

Agent Training on Game 1 (Source Task)
a_1, (eg. left)
Agent 1
Game 1 (eg. Pong)
$s_{t+1,}$ r_{t+1} (eg. game screen, score)
Agent1 Testing on Game 2(Target Task)
a_t(eg. right)
Agent 1
Game 2 (eg. Breakout)
$s_{t+1,}$ r_{t+1} (eg. game screen, score)
Agent1 Testing on Game 3 (Target Task)
a_t(eg. shoot)
Agent 1
Game 3 (eg. asteroids)
$s_{t+1,}$ r_{t+1} (eg. game screen, score)

Figure 4: *Diagrammatic representation of experiment.*

Setup

A set of 4 games were selected, the paired combinations of which were used as source tasks and target tasks. The selection of games was restricted to games with the same action space since a different action space would require to adapt the network architecture in the process of transfer and this would complicate the process. From these games, the Atari-2600 games Q*Bert, Pong, Demon Attack, and Space Invaders were chosen. Atari-2600 games were specifically chosen since they are widely used in the field of RL to compare different algorithms (Mnih et al., 2016; Mnih, Kavukcuoglu, Silver, Graves, Antonoglou et al., 2013; Wu et al., 2017; Schulman et al., 2017; Van Hasselt, Guez, & Silver, 2016) and since the selection of games researched has to allow testing of the hypothesis. This was chosen on the basis of conceptual similarity of two similar games, Space Invaders, Demon Attack, one simple game with one big contrast (not avoid the object coming at the moveable instance but bounce it instead) Pong, and one vastly different game, Q*Bert. The literature for videogame complexity analysis was deemed to be scarce and under-developed, after performing an extensive search and consulting Bellemare, Veness, & Bowling (2012); Brockman, et al. (2016); Guo, Singh, Lee, Lewis, & Wang (2014); Levine, Bates Congdon, Ebner, Kendall, Lucas, Miikkulainen, ... & Thompson (2013); Mnih, Kavukcuoglu, Silver, Rusu, Veness, ... & Petersen (2015), and Wolf (2013).

The complexities of these video games were therefore assessed using criteria selected mainly based on the intuition, in order to qualitatively define the conceptual complexities of the video games. The conceptual similarity between any two of the selected games was assessed using their respective conceptual complexities. Criteria include **Color Scheme, Background Color, Degrees of Freedom, Score Scale, Action space, Movement Orientation, Opponents, Behavior of opponent, Objective of player, Human-Level performance in the game (threshold)**

Experiments

First, one algorithm was trained on all four games with randomly initialised models, thus acquiring ‘base‘ models for each game for that algorithm. This resulted in 4 trained models and logs. These training runs will be referred to as baseline runs for simplicity. Next, Table 1 (Columns 2-5) was followed for both algorithms, generating 12 transfer models and logs. These runs will be referred to as transfer runs for simplicity. Therefore 16 logs were collected for one algorithm. The same runs were executed twice to inspect the stochasticity present in the system. All runs of the second round of TL used the same base model as the first round, in order to test the stochasticity of the results of the transfer runs. To check whether the procedure executed to transfer the model from one run to the other, ‘self-transfer‘ runs were conducted such that the source task and target task were the same. The self-transfer runs were run only once for each game since it is assumed that there is no stochasticity present in the procedure executed. This process was repeated for the other algorithm (See Appendix B).

To create a good overview of how ACKTR and PPO2 behave in TL all possible combinations of base model and the game played were considered for both algorithms. A matrix of the combination of source task and target task was developed, which is the same for both algorithms (Table 1). Each run consists of a variable number of game episodes from which the algorithm learns, since the episode can last from 0 timesteps to tmax (10 Million timesteps), depending on how well the algorithm performs. To avoid varying amounts of training data, the number of timesteps (the decisions taken by the algorithm) of training was set at 10M timesteps. This number of timesteps is commonly used to measure the performance of reinforcement learning algorithms (Mnih et al. 2015; Schulman et al., 2017; Wu, 2017)

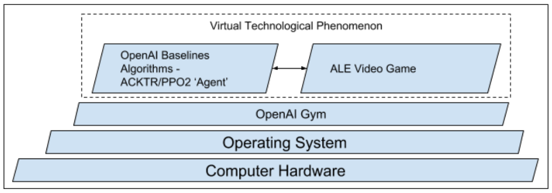

Virtual Technological Phenomenon
OpenAI Baselines Algorithms - ACKTR/PPO2 'Agent'
ALE Video Game
OpenAI Gym
Operating System
Computer Hardware

Figure 5: *Dependency hierarchy of the studied phenomena. The underlying layers had to be kept constant in order to ensure reproducibility of the phenomena*

The Baselines implementation of PPO2 and ACKTR from OpenAI (OpenAIa, 2018) were used in this experiment to ensure a stable and thoroughly tested implementation of the algorithms. The same software-environment was used for all the runs to avoid any unnecessary variables that could influence the experiment. Hyperparameters of the algorithms were not changed from the default Baselines implementation (OpenAI, 2018) and can be found in Appendix B. The computer hardware specifications such as processor clock speed, computer memory, and GPU memory affect the speed of training. The higher the specifications of this hardware, the faster is the training. Due to limitations in available equipment and time, two devices had to be used to carry out all training runs. As seen in Figure 5, the virtual technological phenomena being studied relies on a hierarchy of software and hardware.

In order to ensure reproducibility and maintain uniformity in the context of phenomena (Figure 5) between the two devices used, it was ensured that the hardware specifications were

Runs Executed					
Base model (Source Task)	Base model (Source Task)	Q*Bert	Pong	Space Invaders	Demon Attack
Game played(Target task)					
Q*Bert (QB)	x2	x1	x2	x2	x2
Pong (PO)	x2	x2	x1	x2	x2
Space Invaders (SI)	x2	x2	x2	x1	x2
Demon Attack (DA)	x2	x2	x2	x2	x1

Table 1: *Distribution of Initializations of each Game and the Game that it was Subsequently Transferred to Completed for both ACKTR and PPO2. The 'x2' within the cells says that each of the training runs was run twice. Cells with 'x1' refer to the self-transfer runs.*

similar. The relevant software specifications of the two devices such as Operating System and the particular versions of OpenAI Gym, Baselines were also the same. For the specifics on the hardware and software setup, see Appendix B.

Analysis

The following metrics were chosen based on suggestions by Taylor and Stone (2009) and measured for each run:

1. **Time-to-Threshold**. Human-level performance listed for games in the DQN paper (Mnih et al. 2015) taken as the threshold. This is calculated by finding the average of the time at which the two runs reach the threshold. In cases where both runs have not reached the threshold, this data is left black. However, in cases where just one run reaches the threshold, the time is approximated by finding the time at which the average score of the two runs reaches the threshold(if it does). This had to be done manually and is prone to error in the order of 10^5 timesteps.

2. **Score at time-to-threshold for the base model**. The score achieved by transfer runs at the time at which the baseline run achieves the threshold. This metric is used to check whether the transfer runs are lagging or leading with respect to the baseline run.

3. **Mean reward**. This is effectively the mean reward accumulated by the algorithm throughout the run, providing insight into the overall learning.

4. **Standard Deviation**. This is measured as the mean of the 500-point rolling standard deviations of the algorithm throughout the training run. This provides insight into how much the scores vary over the training.

5. **Scores at 0M, 2.5M, 5M, 7.5M and 9.9M timesteps**. M stands for Million.
   - 0M. This is measured by finding the 500-point rolling average of the first 500 episodes and helps in measuring the jumpstart (Taylor & Stone, 2009)
   - 2.5M, 5M, and 7.5M These are the 500-point rolling averages of scores at each of these times respectively, providing insight into the algorithms's training.
   - 9.9M. This is measured by finding the 500-point rolling average of the score at 9.9M timesteps. This indicates asymptotic performance (Taylor & Stone, 2009)

The 500-point rolling average is the average of the current and previous 499 episode scores achieved by the algorithm. The 500-point rolling standard deviation is the standard deviation of the current and the previous 499 episode scores achieved by the algorithm

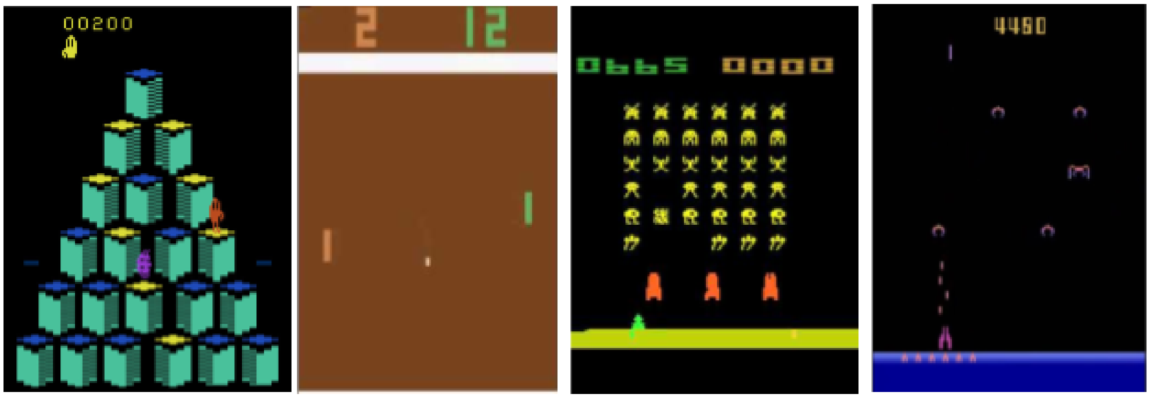


Figure 6: *Screenshots of the four selected games. From left to right: Q*Bert, Pong, Space Invaders and Demon Attack.*

The average of each of these metrics for the twin runs was used as an estimate of the mean. However, it is important to note that the mean 500-point rolling standard deviation is large and the sample size n, is just 2, therefore results are not statistically significant.

Metrics (5) were then normalised with the range (threshold, min game score) in order to be able to compare effectively, as suggested by Bellmaire et al (2013). This was done as follows: norm(metric) = (metric - min score)/(threshold - min score) This enabled comparison of the effects of transfer learning across algorithms and games.

The effects of TL were then assessed as positive if a majority of the above metrics were better than the baseline metrics for a particular target task. Effects were considered negative if a majority of the above metrics were worse than the baseline metrics for that target task. The mean reward was used as a tiebreaker.

IV. Results

Game Complexity

As seen in Figure 6 and defined in Table 2, the games differ in colour schemes and game mechanics. However they share the same (discrete) action space of dimension 6: [0: "NOOP", 1: "FIRE", 2: "UP", 3: "RIGHT", 4: "LEFT", 5: "DOWN"].

Demon Attack and Space Invaders are conceptually very similar, they have the same objectives, same degree of freedom, and similar objectives. Q*Bert is conceptually different from all other games and is conceptually more complex since it has a higher degree of freedom, a larger set of colours and an opponent with behavior adapted to the player. Pong is less complex than the other games, it has the least degrees of freedom, a simple objective, and predictable behavior of the opponent. Demon Attack seems to be the most complex game visually, considering it includes some simple animation and irregular patterns of opponents.

Experiments

From the experiments, the reward achieved by the agent at the end of each episode and the timesteps elapsed in each episode were logged for each run of 10M timesteps. The following plots visualise the reward per episode for all runs of a [game, algorithm] pair over the 10M timesteps. The various curves within the plots represent the different initialisations of the models, allowing us to visually inspect the effects of TL. The raw data is prone to noise and is unintelligible (Presented in Appendix B). In order to solve this, the 500-point rolling average of each trial is presented in Figure 7.

We observe that in the qb_ppo2, qb_acktr, and pong_acktr graphs, the respective baseline runs never reach human-level playing ability

Game Complexity				
	Q*Bert	**Pong**	**Space Invaders**	**Demon Attack**
Color Scheme	Black, Yellow, Navy Blue, Sky Blue, Red,	Brown, Red, Green, White	Black, Yellow, Green, Orange	Black, Yellow, Purple, Navy Blue
Background Color	Black	Brown	Black	Black
Degrees of Freedom	3	1	2	2
Score Scale	[0,Inf)	[-21,21]	[0,Inf)	[0,Inf)
Action space	Discrete(6)	Discrete(6)	Discrete(6)	Discrete(6)
Movement Orientation	Vertical and Horizontal /Diagonal movement	Vertical movement only	Horizontal movement only	Horizontal Movement only
Opponents	Present	Present	Present	Present
Behavior of opponent	Opponent attempts to block the player in various ways	Attempt to return to ball	Move from left to right while shooting at the player. Move one down when reaching the edge.	Move left, right, up and down in an irregular pattern while shooting at the player
Objective of player	Avoid opponent, while activating all tiles	Bounce ball past opponent	Shoot all opponents before the opponents reach the ground while avoiding fire	Shoot all opponents before opponents reach the ground while avoiding fire
Human Level performance (threshold score)	13455	9.3	1652	3401

Table 2: *Conceptual complexity of the games are described using the above features as visible on the game screens.*

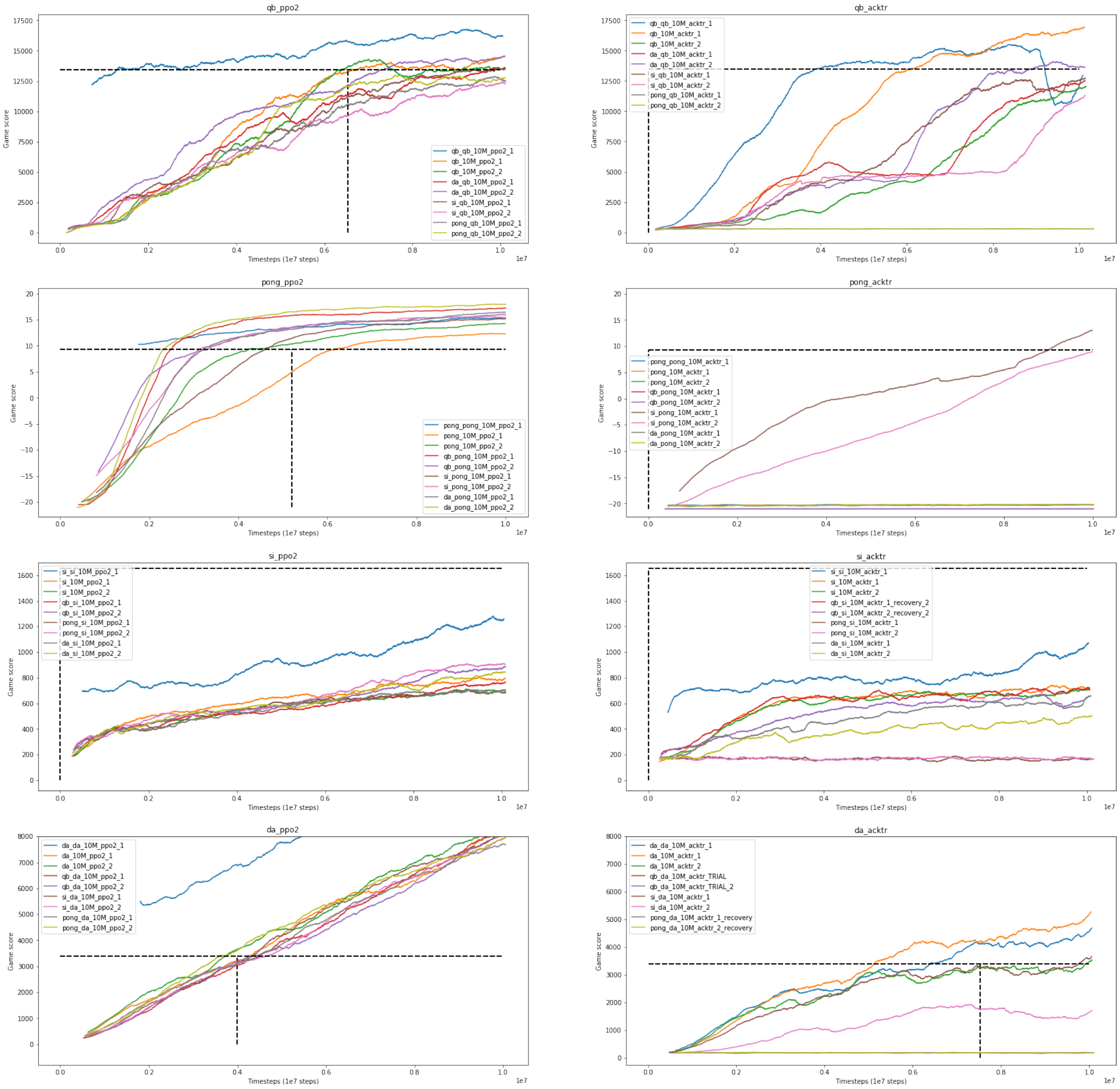


Figure 7: *500-Point rolling average of the game score plotted against the 10M timesteps. The hyphenated lines superimposed onto these graphs represents the thresholds (human level performance). Each column of graphs represents an algorithm, as PPO2 and ACKTR runs from left to right respectively. Each row of graphs represents a game as Q*Bert, Pong, Space Invaders and Demon Attack from top to bottom respectively. The titles of the graphs follow the naming convention: game_algorithm. The legend follows the naming convention: source-game_target-game_number-of-timesteps_algorithm_run-number*

Results of Experiments

Target task	Source task	Algorithm	Time-to -threshold (timesteps)	Reward at time-to -threshold	Mean Reward	Standard deviation	Normalised Score				
							0M	**2.5M**	**5M**	**7.5M**	**9.9M**
Q*Bert	-	PPO2	**6532761**	**13471.05**	6836.744	2639.221	0.024	0.273	0.737	**1.023**	**1.039**
	DA		8549481	13462.625	**7493.368**	2583.759	0.021	**0.348**	**0.746**	0.92	1.037
	SI		-	-	6321.353	2714.467	**0.03**	0.28	0.57	0.828	0.955
	Pong		-	-	6124.969	2486.774	0.001	0.265	0.669	0.899	0.942
	-	ACKTR	**9000000**	**13460***	**4805.455**	1598.237	0.017	0.147	**0.508**	**0.818**	**1.059**
	DA		-	-	4317.195	1418.888	**0.02**	**0.158**	0.346	0.735	0.965
	SI		-	-	3754.789	1491.958	0.018	0.118	0.355	0.62	0.864
	Pong		-	-	285.993	111.914	0.019	0.021	0.022	0.021	0.021
Pong	-	PPO2	5192064	7.614	1.73	5.202	0.035	0.546	0.924	1.092	1.131
	QB		2900590	**10.059**	**8.993**	3.865	0.122	**0.967**	1.171	1.21	**1.239**
	SI		3868794	8.221	7.528	4.289	**0.148**	0.693	1.096	1.173	1.199
	DA		**2760533**	8.884	8.698	4.405	0.02	0.917	**1.175**	**1.224**	1.26
	-	ACKTR	-	-	-20.303	0.837	0.022	0.021	0.023	0.025	0.025
	QB		-	-	-21	0	0	0	0	0	0
	SI		9650000	-	**-4.271**	4.435	**0.064**	**0.343**	**0.587**	**0.786**	**1.047**
	DA		-	-	-20.312	0.834	0.019	0.021	0.023	0.024	0.025
Space Invaders	-	PPO2	-	-	560.944	184.444	0.115	**0.301**	**0.369**	0.426	0.446
	QB		-	-	556.744	176.388	0.122	0.285	0.341	0.414	**0.491**
	Pong		-	-	**570.029**	186.469	**0.155**	0.288	0.361	**0.434**	0.484
	DA		-	-	552.783	188.135	0.13	0.279	0.351	0.428	0.464
	-	ACKTR	-	-	**552.3**	204.922	0.095	**0.328**	**0.386**	**0.4**	**0.435**
	QB		-	-	536.671	192.958	**0.122**	0.295	0.368	0.388	0.402
	Pong		-	-	168.104	120.654	0.101	0.103	0.101	0.101	0.102
	DA		-	-	401.872	151.412	0.109	0.208	0.258	0.297	0.328
Demon Attack	-	PPO2	**3991621**	3436.1	**4057.996**	2146.892	**0.114**	**0.665**	**1.270**	**1.868**	**2.423**
	QB		4383013	3357.715	3575.717	1861.045	0.092	0.55	1.127	1.738	2.384
	SI		4497993	**3481.975**	3707.99	1807.019	0.086	0.57	1.151	1.867	2.34
	Pong		4025153	3390.79	3864.239	1758.679	0.105	0.648	1.232	1.838	2.279
	-	ACKTR	**7532775**	**3748.935**	**2467.906**	1655.131	**0.056**	**0.538**	**0.928**	**1.094**	**1.233**
	QB		-	-	172.585	105.258	0.054	0.053	0.051	0.051	0.052
	SI		-	-	1554.872	1166.664	0.05	0.301	0.588	0.739	0.751
	Pong		-	-	173.387	107.346	0.052	0.051	0.05	0.05	0.051

Table 3: *The results of the analysis of the experimental data. Each column corresponds to one of the metrics we selected to study the effects of TL. Entries emboldened represent the maximum or the minimum of the respective metric within the (algorithm, target task) combination. Entries with '-' mean the threshold was never reached within 10M timesteps. For cells with *, the values were approximated manually by finding time at which the average of the two runs reaches the threshold because one of the runs does not reach the threshold within 10M frames.*

Effect of Transfer Learning

Source Task	Q*Bert		Pong		Space Invaders		Demon Attack	
Target task	PPO2	ACKTR	PPO2	ACKTR	PPO2	ACKTR	PPO2	ACKTR
Q*Bert	-	-	-ve	-ve	-ve	-ve	+ve	+ve
Pong	+ve	-ve	-	-	+ve	+ve	+ve	-ve
Space Invaders	-ve	-ve	+ve	-ve	-	-	-ve	-ve
Demon Attack	-ve	-ve	-ve	-ve	-ve	-ve	-	-

Table 4: *A matrix representing the instances of positive and negative transfer with respect to each (source,target) task combination. Red colour indicates negative transfer and Green colour indicates positive transfer.*

within the 10M time frames that the algorithms were given to learn to play the video games.

PPO2 implementations completely exceed human-level performance in both Pong and Demon Attack, partially reaches or exceeds human-level performance in Q*Bert, but fails to reach human-level performance in Space Invaders.

In comparison, the ACKTR baseline implementation reaches human-level performance in Demon Attack but only one implementation of ACKTR trained on Space Invaders reaches human-level performance, while all other implementations fail to get close to human-level performance.

Only one baseline implementation of ACKTR exceeds human-level performance in Q*Bert, as does one implementation of Demon Attack initialised weights.

The second baseline implementation, as well as the other initialisation of Demon Attack and both Space Invaders initialised weights all get roughly as effective at the game.

It is also worth pointing out that the Pong initialised weights implemented on both Demon Attack and Q*Bert score 0, but not for Space Invaders as no implementation gets close to reaching human-level performance in Space Invaders.

The implementation only comes close to human-level performance in Pong when transferring weights from Space Invaders, and otherwise does not improve at all. The Pong baselines do not improve within 10M timesteps either.

We also observe that, generally, the scores of the self transfer runs approximately start at the final achieved score from the base run, except for qb_acktr and da_acktr. Table 3 shows the results after analysis of the data which is used for the inferences drawn below.

V. Inferences

Jumpstart vs. learning curve vs. asymptotic performance

TL provides a sizeable jumpstart for PPO2 both with Space Invaders as target task, Pong as source task and vice versa. This might indicate a commonality between the games that allows for a jumpstart but not for a very significant asymptotic performance increase. PPO2 with Pong as a source task and Space Invaders as target task also shows the best learning curve, as it shows the highest mean reward. In general, the base runs of the games have the best learning curves, based on the mean averages.

Phenomena that stand out

The results of ACKTR playing Pong are comparable to random action selection. Normally, this behavior of neural networks is only observed when the weights are adjusted disproportionately such that it results in computational overflow and NaN is assigned to the weights, preventing further computation. In the DL field, this phenomenon is called gradient explosion (Goodfellow et al., 2016). Alternatively, the weights become uniform, which effectively reduces the neural network to a linear function. The latter is highly unlikely because the weights are randomly initialised for the base runs, which are adjusted according to the gradient of the experiences. Therefore, it is highly unlikely, if not impossible, to lead to a uniform distribution. The exploding gradient scenario is specifically avoided by the trust region approach of ACKTR and can therefore not be the cause. However, the performance similar to random action selection persists in the transfer runs. This does lead us to believe the model weights are assigned NaN, possibly generated by a lack of Random Access Memory of the computer, which resulted in memory overflow, which led to NaN weights without gradient explosion. Overall, this case provides evidence of negative transfer if source task causes the algorithm to have extreme weights.

In the case of Pong as target task, ACKTR fails to learn anything, except with space invaders as source task. This means that initialising ACKTR with the weights learnt in Space Invaders, does not lead to the extreme weights issue described earlier, and ACKTR achieves the threshold.

Demon Attack is not benefited by any of the transfer runs we executed. Since the algorithms fully depend on the frame input and the CNN employed by the algorithms does not ascribe conceptual meaning to this frame input, this could be the result of the visual complexity of Demon Attack, as compared to the other games. This is especially interesting when considering that Demon Attack and Space Invaders are conceptually similar, which is not reflected in the results.

Differences between effect of TL on the two Algorithms

TL has barely any effect on PPO2 for all target tasks except Pong, where Demon Attack as source task seems to have the most positive effect. TL seems to have significant effects on ACKTR for all tasks. However, these effects can be either positive or negative depending on the source, target task pair.

For ACKTR trained on Demon Attack as target task, all attempts of TL prove counterproductive as all metrics indicate a large negative transfer. Unlike PPO2 on Demon Attack, where the attempts at TL also lie beneath the base run, but the results lie that close to the base run results that the difference is negligible.

Comparing the effects of TL between the two algorithms, there does not seem to be significant differences except for the anomalous behaviour of ACKTR on Pong. Although the performance of these algorithms differ across games, the effects of TL on both of them seem to be the same.

Hypothesis - effect of game similarity and TL

Our hypothesis seems to have been wrong since these results are counter-intuitive in the sense that TL does not prove most effective for (source, target) tasks that are conceptually similar. In fact the most interesting results arise for (source, target) task pairs that are conceptually dissimilar.

VI. Conclusion

This paper is on an empirical research conducted on the effects of Transfer Learning on the performance of ACKTR and PPO2 on playing Pong, Demon Attack, Space Invaders and Q*Bert. This research is meant to be viewed as a pilot study, intended to develop insights into the effects TL bears on the performance of these algorithms and the methodology of research that could be followed to conduct studies into the virtual-technological phenomena exhibited by algorithms when performing TL. The results show positive effects of Transfer Learning on particular source and target task combinations, while also causing negative effects on other combinations. Generally, TL has noticeable effect on the performance of ACKTR, either positive or negative, and shows little effect on the performance of PPO2. However, the results show that it is not straightforward to predict when TL will be beneficial since they show that performance is not benefited when learning is transferred across conceptually similar games. This shows that factors like visual complexity and similarity, might weigh in more than the conceptual similarity of the games to the prediction of the outcome of TL.

VII. Discussion

Reflecting on the execution of our research and the phenomena that were observed, we have come up with several points of discussion.

The results of the transfer run si_po_acktr are very interesting. The transfer of the weights of Space Invaders clearly allows ACKTR to learn Pong, while it fails to do so in all other runs. Since there is a significant jumpstart in this case while there are none so significant in all other cases, this indicates a correlation between a significant jumpstart and positive transfer for ACKTR. Further inspection of this phenomenon might lead to new insights in Transfer Learning. It is important to note that Wu et al. (2017) do present results of ACKTR learning Pong that conflict those obtained in this study. However, Wu et al. (2017) use a different version of Pong in ALE: 'v4' , in their study, while 'v0' is used in this study.

We did not generate enough samples of training runs to use the normal distribution or the vast majority of any statistical distributions to analyse the data. A larger sample size was not feasible because of limited time. This would have allowed us to account for stochasticity and estimate the means and standard deviations relatively reliably. In hindsight we should have stuck to 3 games, since this was the minimum number required for testing our hypothesis, and run more trials at the same time. Since our study does not use enough samples to make inferences with statistical significance, the results of this paper cannot be used for any applications other than identifying point of interest for further study, effectively making this a pilot study. These points of interest would likely help make progress towards AGI and therefore achieve the major goal of RL.

The script of the failed runs was not kept, as they were adjusted to make the runs successful. This can be seen as faulty execution because these failed runs were caused by practical mistakes when setting up the experiment and are therefore not relevant to reproduction. However, the logs of these failures are maintained and can be examined at the GitHub repository (Gopal, van der Neut, 2018).

In this study, we have not attempted to study the effects of the hyperparameters used in the algorithms on the results of TL in RL. Given that hyperparameters can have great effects on the stability and rate of learning, it might be useful to study the effects of hyperparameter on TL in RL. A level of analysis that lies just outside the scope of this study is to execute was an evaluation of the changes between the initial model parameters and the final model parameters. This could be a method to compare the interpretation of abstraction created by the network, given that the early layers account for the more abstract processing of the input (Ng, 2017). The inability to access these weights also raises ethical concerns since we would not know exactly how the algorithm is achieving its goal. Since the overarching goal is to progress towards AGI, and as Bostrom and Muehlhauser (2014) argue, AGI could present an existential threat to the human species, it would be crucial to understanding how such an algorithm achieves its goal.

Development of algorithms with greater generalisability could be benefited by developing a method that ascribes conceptual meaning to the input processed by the CNN, which then inputs this conceptual meaning into a learning algorithm. This development could benefit generalisability since the counterintuitive results possibly arise from the dependency of the algorithm on the frame input and are therefore directly influenced by visual complexity, which would be alleviated by assigning conceptual meaning to the input.

VIII. Acknowledgments

The setup of this research hinged on the B&K model introduced by Mieke Boon (2018), who also provided us with feedback and direction throughout the research. We are grateful to her having done so and for the early access to her paper on B&K. Joost Voskuil was involved in the early planning stages of this study, helped to prepare the first draft of the paper and provided feedback on the work several times during the research. In multiple stages of the research, Dr. C. Brune was consulted and provided highly appreciated feedback. This research would not have been possible without the open source efforts of OpenAI which facilitated the experimental setup of this research with implementations and a platform. Lastly, we are grateful to our peers Wout Ploeg and Pim Schoolkate for their feedback on the paper and other peers and teachers with whom we discussed our ideas and who challenged our presuppositions.

References

[1] Achiam, J., Wu, A., Morales, M., Marciniak, C., Bastovanovicz, M., Hesse, C. (2018) *Part 1: Key Concepts in RL - Spinning Up documentation*. Retrieved from https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

[2] Agarwal, M. (2017, December 14). *Back Propagation in Convolutional Neural Networks - Intuition and Code*. Medium. Retrieved from https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

[3] Gopal, A., van der Neut, L. M. B., (2018) AkarshG1, ATLAS_S3_RL. GitHub repository, https://github.com/AkarshG1/ATLAS_S3_RL

[4] Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., ... & Schneider, J. (2018). *Learning dexterous in-hand manipulation*. arXiv preprint arXiv:1808.00177.

[5] Arulkumaran, K., Deisenroth, M., Brundage, M., & Bharath, A. (2017). *Deep Reinforcement Learning: A Brief Survey*. IEEE Signal Processing Magazine, 34(6), 26-38. doi: 10.1109/msp.2017.2743240

[6] Ba, J., Grosse, R., & Martens, J. (2016). Distributed second-order optimization using Kronecker-factored approximations.

[7] Baum, S. (2017). "A Survey of Artificial General Intelligence Projects for Ethics, Risk, and Policy."

[8] Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253-279.

[9] Bellemare, M. G., Veness, J., & Bowling, M. (2012, July). Investigating Contingency Awareness Using Atari 2600 Games. In AAAI.

[10] Bellman, R. (1957). "A Markovian Decision Process". Journal of Mathematics and Mechanics.

[11] Boon, M. (2018). Scientific Methodology in the Engineering Sciences. In D. Michelfelder and N. Doorn (version of May 14th 2018), Routledge Handbook of Philosophy of Engineering. Scheduled for publishing in 2019.

[12] Bostrom, N. (2014). Superintelligence (1st ed.). Oxford: Oxford University Press.

[13] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

[14] Chokmani, K., Khalil, B., Ouarda, T. B. M. J.,& Bourdages, R. (2007). Estimation of river ice thickness using artificial neural networks. In Proc. 14th Workshop Hydraulics Ice Covered Rivers. CGU HS/CRIPE (p. 12).

[15] Dixit, Avinash K. (1990). Optimization in Economic Theory (Second ed.). Oxford: Oxford University Press. p. 164. ISBN 0-19-877211-4.

[16] Einstein, A. (1916). The foundation of the general theory of relativity. Annalen Phys., 14, 769-822.

[17] Gillis, J. (2006). The gradient descent algorithm in action. (1: contour). Wikipedia. Retrieved from `https://en.wikipedia.org/wiki/Gradient_descent#/media/File:Gradient_ascent_(contour).png`

[18] Glatt, R., Silva, F., & Costa, A. (2016). Towards Knowledge Transfer in Deep Reinforcement Learning. 2016 5Th Brazilian Conference On Intelligent Systems (BRACIS). doi: 10.1109/bracis.2016.027

[19] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning (Vol. 1). Cambridge: MIT press.

[20] Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In Advances in neural information processing systems (pp. 3338-3346).

[21] Hausknecht, M. J., & Stone, P. (2015, January). The Impact of Determinism on Learning Atari 2600 Games. In AAAI Workshop: Learning for General Competency in Video Games.

[22] Hawking, S., & Stone, G. (1992). A Brief History of Time: A Readers Companion. Bantam.

[23] Kakade, S. M. (2002). A natural policy gradient. In Advances in neural information processing systems (pp. 1531-1538).

[24] Kimura, H. and Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value function. ICML, pp. 278-286.

[25] Kirk, D. E. (1970). Optimal control theory: an introduction. Springer.

[26] Kunz, F. (2000). An introduction to temporal difference learning. In Seminar on Autonomous Learning Systems.

[27] Kurzweil, R. (2005), The Singularity is Near, Viking Press, ISBN 0-14-303788-9, OCLC 71826177.

[28] Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. Artificial intelligence, 33(1), 1-64.

[29] LeCun, Y., Bengio, Y., Hinton, G. (2015). *Deep Learning*. Nature, vol. 521, no. 7553, pp. 436-444.

[30] Levine, J., Bates Congdon, C., Ebner, M., Kendall, G., Lucas, S. M., Miikkulainen, R., ... & Thompson, T. (2013). *General video game playing*.

[31] Ma, Y., Xiang, Z., Du, Q., & Fan, W. (2018). Effects of user-provided photos on hotel review helpfulness: An analytical approach with deep learning. International Journal of Hospitality Management, 71, 120-131.

[32] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., & Bowling, M. (2017). *Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents*. Journal of Artificial Intelligence Research, 61, 523-562.

[33] Markov, A. A. (1954). The theory of algorithms. Trudy Matematicheskogo Instituta Imeni VA Steklova, 42, 3-375.

[34] Martens, J., & Grosse, R. (2015, June). *Optimizing neural networks with kronecker-factored approximate curvature*. In International conference on machine learning (pp. 2408-2417).

[35] McClelland, J. L. (2015, December 16). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.*

[36] Muehlhauser, L., & Bostrom, N. (2014). *Why we need friendly AI.* Think, 13(36), 41-47.

[37] Muehlhauser, L. (2014). *What is AGI?*. Machine Intelligence Research Institute.

[38] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). *Asynchronous methods for deep reinforcement learning*. In International conference on machine learning (pp. 1928-1937).

[39] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540), 529.

[40] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602.

[41] Ng, A. (2017). *Machine Learning*, by Stanford University on Coursera.

[42] Ng, A. (2018). *Convolutional Neural Networks*, by deeplearning.ai on Coursera.

[43] Newton, I. (1934). *Principia Mathematica*. Newton's principia, 634.

[44] OpenAIa, Baselines, (2018), GitHub repository, `https://github.com/openai/baselines`

[45] OpenAIb, Gym, (2018), GitHub repository, `https://github.com/openai/gym/blob/master/gym`

[46] Pan, S., & Yang, Q. (2010). *A Survey on Transfer Learning*. IEEE Transactions On Knowledge And Data Engineering, 22(10), 1345-1359. doi: 10.1109/tkde.2009.191

[47] Patel, Y. (2017, July 30). *Reinforcement Learning w/ Keras + OpenAI: Actor-Critic Models*. Medium. Retrieved from `https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69`

[48] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2.

[49] Schmidhuber, J. (2015). *Deep learning in neural networks: An overview.* Neural networks, 61, 85-117.

[50] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347.

[51] Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P. (2015). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv:1506.02438

[52] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). *Trust region policy optimization*. In International Conference on Machine Learning (pp. 1889-1897).

[53] Hoo-Chang, S., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., ... & Summers, R. M. (2016). *Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning*. IEEE transactions on medical imaging, 35(5), 1285.

[54] Skinner, B. (1953) *Science and Human Behavior*. Colliler-Macmillian.

[55] Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (Vol. 135). Cambridge: MIT press.

[56] Taylor, M., and Stone, P (2007). *Cross-domain transfer for reinforcement learning*. In Proceedings of the Twenty-Fourth International Conference on Machine Learning, June 2007b.

[57] Taylor, M., & Stone, P. (2009). *Transfer Learning for Reinforcement Learning Domains: A Survey*. Retrieved from `http://www.jmlr.org/papers/v10/taylor09a.html`

[58] TechnoReview. (2013, June). *Artificial Neural Network : Beginning of the AI revolution.* Medium. Retrieved from `https://hackernoon.com/artificial-neural-network-a843ff870338`

[59] Thorndike, E., and Woodworth, R.,(1901). *The influence of improvement in one mental function upon the efficiency of other functions*. Psychological Review, 8:247âĂŞ261.

[60] Van Hasselt, H., Guez, A., & Silver, D. (2016, February). *Deep Reinforcement Learning with Double Q-Learning*. In AAAI (Vol. 2, p. 5).

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

[62] Weiss, K., Khoshgoftaar, T., & Wang, D. (2016). *A survey of transfer learning.* Journal Of Big Data, 3(1). doi: 10.1186/s40537-016-0043-6

[63] Wolf, M. J. (2013). *Abstraction in the video game*. In The video game theory reader (pp. 69-88). Routledge.

[64] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., & Ba, J. (2017). *Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation*. In Advances in neural information processing systems (pp. 5279-5288).

[65] Yuan, Y. (2015). *Recent advances in trust region algorithms*. Mathematical Programming, 151(1), 249-281. doi: 10.1007/s10107-015-0893-2

Appendix

A

Neural Networks

Standard neural networks consist of many, fairly simple, nodes that compute an activation. These nodes are called neurons, and their connections lead to the term 'Neural Network'.

Forward propagation In Figure 9, the inputs on the left-hand side are the outputs of all neurons of the earlier layer or constitute the vectorised input from the environment to the network (input layer). Based on the output of the last layer conclusions on the input are drawn (Schmidhuber, 2016) and is called the output layer. Every layer between the output and the input layer are called hidden layers. Each node multiplies all its inputs with a weight respectively, takes the sum of this (Ij), and this creates the input for the activation function (f). The output of the activation function (Oj) is a single number, serves as the output of the single node, and creates part of the input for the next layer of neurons. The activation function ensures that the output of the neuron lies in the desired range and creates a non-linear system. Only then is the network able to solve non-trivial problems with complex solutions, otherwise, it would only compute a linear function Wx of the input x, no matter how deep the network is (Goodfellow et al., 2016). For binary classification problems, the sigmoid function is widely used, as seen in Figure 9. (TechnoReview, 2013)

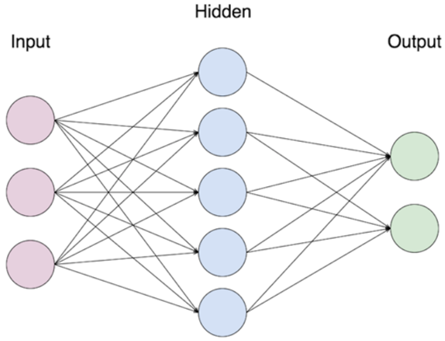


Figure 8: *Visual representation of a Neural Network.*

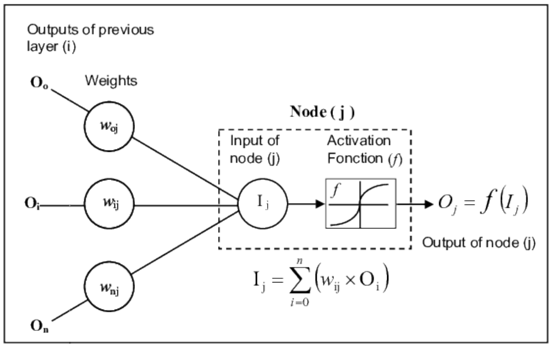


Figure 9: *(Chokmani, Khalil, Ouarda, & Bourdages, 2007) forward propagation through a single neuron is depicted.*

Back-Propagation The weights that are multiplied with input of the node are adjusted in the backpropagation through the network. In backpropagation the weights in the network are commonly adjusted according to the first order derivative of the cost function (expected return) with respect to the parameters of the model, like square meters, number of rooms, and number of bathrooms are parameters when determining the price of a house. The aim of any algorithm is to maximise the cost function, its goal is determined by how this cost function is defined. Algorithms that directly adjust their weights according to the cost function are called policy-based learning algorithms (Goodfellow et al., 2016).

Convolutional Neural Networks (CNN)

Information extracted from Convolutional Neural Networks on Coursera by Ng (2018) and Deep Learning by Goodfellow et al. (2016).

CNNs are able to process images and frames of videos with a slightly different set-up than regular neural networks. The CNN takes in a matrix representation of the picture or frame of the video. The matrix representation stores the intensity of a certain colour in every pixel for every pixel. With greyscale input, there is only one matrix and a full-colour input has a representation of three matrices stacked on top of each other, red, green, and blue respectively. These matrices are processed in two ways in one convolution layer, namely using convolution and pooling.

Forward Propagation Convolution The convolution uses a smaller matrix, a so-called 'mask'. This mask is depicted as the small square in Figure 10 of the building and it builds up the values in the second matrix as the mask moves from the top left, over the whole picture, to the bottom right. The mask always uses the same depth dimension as the input and there are often multiple different masks applied in one convolution step. The mask contains certain variable weights. When the mask is laid over a specific part of the image it multiplies the values of that point in the matrix, adds all the outcomes up and this becomes the value in the next matrix.

Pooling Pooling is done per section of the matrix and has multiple versions. This section does not slide across the matrix but separates the matrix into specific parts. Widely applied versions of pooling are average pooling and max pooling. Pooling has a similar function as to the activation function in a neural network, applying non-linearity to the system, allowing it to produce complicated networks and relations. Pooling happens from the second to the third matrix in Figure 10. It takes a defined section of the matrix and builds the third matrix by either taking the average (average pooling) or maximum (max pooling) of the values in this section.

Convolution layer and progress through the network After proceeding through the first layer, explained by the steps above, the next convolution layer then scans over the results of the pooling of the earlier layer, executes the convolution, and pools them. The matrices shrink in the convolution step under the influence of the size of the mask and the 'stride'. The stride is the step size with which the mask moves across the earlier matrix. In the pooling layer, the matrix shrinks under the influence of the size of the sections. After running the desired number of convolutional layers with the desired settings, the matrix is flattened out into one long vector. This vector is then processed by fully connected layers which are essentially the same as the neural network described before but take the output of the convolutional layers as input. This then serves to identify objects in the image, select a preferred action given the image, or recognise faces, depending on the goal of the network.

Back-Propagation In the backward propagation of the CNN, the values of the masks (middle and last matrix) are updated. The values of the mask are comparable to the weights of a

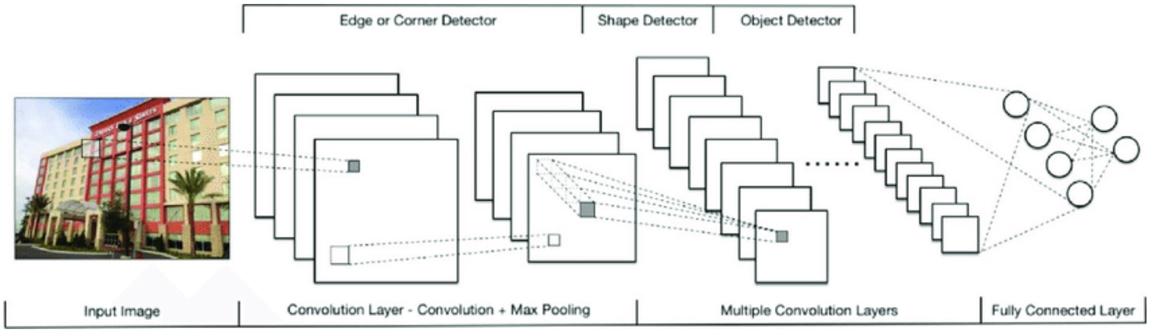
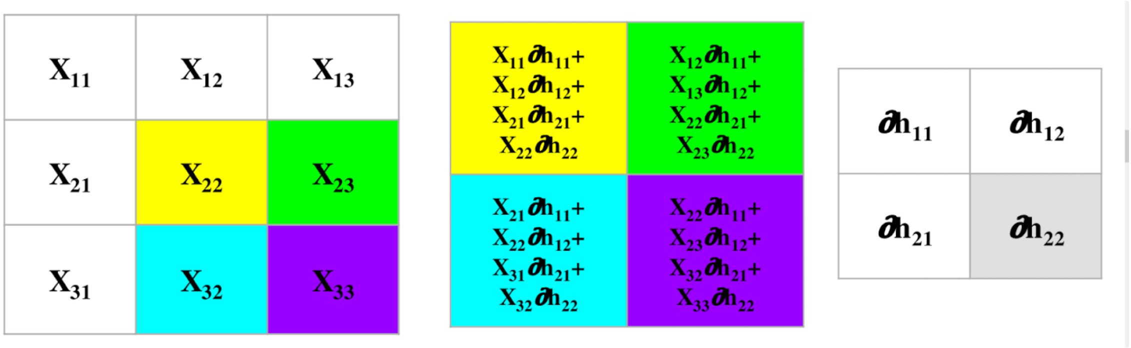


Figure 10: *Layers of a CNN (Ma, Xiang, Du, & Fan, 2018).*



X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$X_{11}\partial h_{11}+$ $X_{12}\partial h_{12}+$ $X_{21}\partial h_{21}+$ $X_{22}\partial h_{22}$	$X_{12}\partial h_{11}+$ $X_{13}\partial h_{12}+$ $X_{22}\partial h_{21}+$ $X_{23}\partial h_{22}$
$X_{21}\partial h_{11}+$ $X_{22}\partial h_{12}+$ $X_{31}\partial h_{21}+$ $X_{32}\partial h_{22}$	$X_{22}\partial h_{11}+$ $X_{23}\partial h_{12}+$ $X_{32}\partial h_{21}+$ $X_{33}\partial h_{22}$

∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

Figure 11: *(Agarwal, 2017), the derivative of the cost function just as a normal neural network but it updates the value according to the sum of the derivatives of the cost function (shown in the middle matrix) with respect to each value that is taken into account by a specific value in the mask (first matrix)*

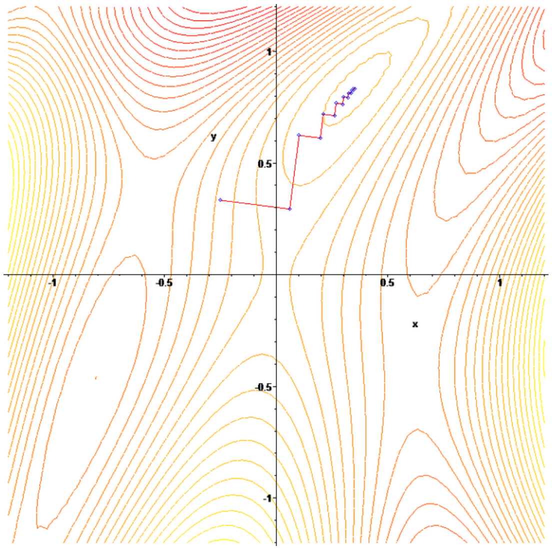


Figure 12: *(Gillis, 2006), creating an algorithm that maximises more precisely and efficiently using 2^{nd} order optimisation .*

simple neural network. The values of the mask are then updated according to the

Optimization

Optimization practically determines in which direction the algorithm is going to shift its values with the goal to minimise the loss function, eventually guiding the network to an optimal policy. 1^{st} order optimization updates the value using the first order derivative of the 'cost' function. The weight is updated by adding the first order derivative multiplied by the set learning rate to the weight. The first order derivative provides the slope at every point of the plane of the cost function

2^{nd} order optimization is a very computationally expensive process when this is done with the exact values. The 2^{nd} order statistic of the cost function, however, enables the algorithm to make an accurate estimation of how the cost can be maximised in the most efficient and certain way. The second order statistic determines how confident it is that a step in a certain direction will get it to maximise the cost function. This can then be used to determine the direction of the step and the magnitude of the step with more precision, smoothening the trajectory depicted in Figure 12 Therefore multiple different methods attempt to approximate this 2^{nd} order statistic of the cost function.

Difference between policy-based and value-based learning

Apart from the policy-based learning algorithms, there are value based learning algorithms. These algorithms are trained to estimate the value of a variable given a certain state from the environment it is evaluating. This algorithm cannot directly update its parameters since it is concerned with estimating a value that lies in the future (time + 1). It cannot calculate a cost function because it needs the next run to determine whether its estimation was accurate and only then can the algorithm update its weights (Sutton & Barto, 1998; McClelland, 2015). The remaining process of learning of value-based learning algorithms is similar to what was described earlier.

B

- Hardware Specifications: Table 5
- Software Setup: Table 6
- Training Runs per device in chronological order: Table 7
- Hyperparameters: Table 8
- Graphs of Raw Data: Figure 13

Device	1	2
Device name	Lenovo Thinkpad P51	Dell Inspiron 15 gaming 7567
Processor	Intel(R) Core^{TM} i7-7700HQ cpu @ 2.80GHz, 2808 Mhz	Intel(R) Core^{TM} i7-7700HQ cpu @ 2.80GHz, 2808 Mhz
Random Access Memory	16GB / 2400MHz DDR4, non-ECC	16GB / 2400MHz DDR4, non-ECC
Primary Graphics Card	NVIDIA Quadro M1200 (4096MB; CUDA compute capability: 5.0)	Nvidia GTX 1050-Ti (4096 MB; CUDA compute capability 6.1)
Secondary Graphics Card	Intel(R) HD Graphics 630	Intel(R) HD Graphics 630

Table 5: *Device Hardware Specifications*

Software set-up	
Operating System:	Linux Ubuntu 18.10 (same on both devices)
Graphics driver:	Nvidia driver 396.54 (same on both devices)
CUDA:	CUDA 9.0 (Same on both devices)
Python:	3.6.3 (device 1), 3.6.6 (device 2)
Dependencies:	See requirements.txt (Gopal, van der Neut, 2018)

Table 6: *Software Setup for the study*

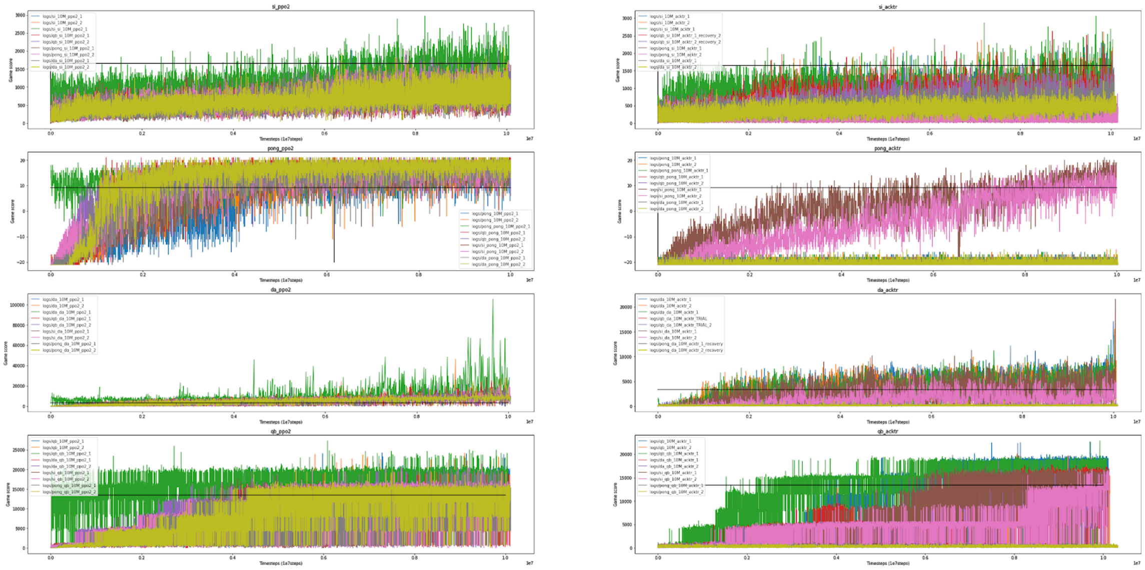


Figure 13: *Game score(Raw data) at each timestep during the 10M timesteps*

Device 1	Device 2
-	PPO2 on SpaceInvaders x2 and Pong x1
ACKTR on SpaceInvaders x2	PPO2 on Pong and DemonAttack x2
ACKTR on DemonAttack x2	PPO2 and ACKTR base SpaceInvaders on Pong
ACKTR on Pong x2	ACKTR base SpaceInvaders on DemonAttack x2
ACKTR on Q*Bert x2	PPO2 base SpaceInvaders on Q*Bert x2
ACKTR base SpaceInvaders on Q*Bert x2	PPO2 base SpaceInvaders on DemonAttack x2
ACKTR base Q*Bert on SpaceInvaders x2	PPO2 base Q*Bert on SpaceInvaders x2
ACKTR base Q*Bert on Pong x2	PPO2 base Q*Bert on Pong x2
ACKTR base Q*Bert on DemonAttack x2	PPO2 base Q*Bert on DemonAttack x2
ACKTR base Pong on SpaceInvaders x2	PPO2 base Pong on SpaceInvaders x2
ACKTR base Pong on Q*Bert x2	PPO2 base Pong on Q*Bert x2
ACKTR base Pong on DemonAttack x2	PPO2 base Pong on DemonAttack x2
ACKTR base DemonAttack on SpaceInvaders x2	PPO2 base DemonAttack on SpaceInvaders x2
ACKTR base DemonAttack on Q*Bert x2	PPO2 base DemonAttack on Q*Bert x2
ACKTR base DemonAttack on Pong x2	PPO2 base DemonAttack on Pong x2
ACKTR base SpaceInvaders on SpaceInvaders, ACKTR base DemonAttack on DemonAttack	PPO2 base SpaceInvaders on SpaceInvaders, PPO2 base DemonAttack on DemonAttack
ACKTR base Pong on Pong, ACKTR base Q*Bert on Q*Bert	PPO2 base Pong on Pong, PPO2 base Q*Bert on Q*Bert

Table 7: *Training runs per device in chronological order*

Hyperparameters	PPO2	ACKTR
Network	Nature CNN	Nature CNN
nsteps	2048	20
Discount	0.99	0.99
GAE parameter (Îż)	0.95	0.95
Learning rate	3.00E-04	0.25
VF coeff.c1(9)	0.5	0.5
Entropy coeff. c2(9)	0	0.001
Clipping parameter	0.2	(KFAC) 0.001
Noptepochs	4	-
Numprocs	-	32
VF fisher coeff	-	1
Number of actors	-	8
Lr schedule	-	linear

Table 8: *Hyperparameters as extracted from OpenAI Baselines(2018).The terms do not necessarily carry same meaning for both algorithms.*