MULTI-TASK GENERALIZATION USING PRACTICE FOR DISTRIBUTED DEEP REINFORCEMENT LEARNING

by

Upasana Pattnaik

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2021

Approved by:

_____
Dr. Minwoo Lee

_____
Dr. Min Shin

_____
Dr. Liyue Fan

ABSTRACT

UPASANA PATTNAIK. Multi-Task Generalization using Practice for Distributed Deep Reinforcement Learning. (Under the direction of DR. MINWOO LEE)

Feedback driven deep reinforcement learning methodologies are widely favoured approaches to solving artificial intelligence problems. The algorithms navigate complex decision-making tasks without manual state space engineering. Notable problems considered out of reach by machines, like mastering Go, StarCraft, Dota2 and Atari 2600 games were solved successfully. However, these algorithms require extensive amounts of time and data to specialize in one problem.

Transfer learning strategies approach this challenge by supplementing the reinforcement learning task with shareable knowledge from a source task. The source provides basal knowledge about the environment, thus reducing the time to learn. Additionally, these strategies can be adopted in the multi-task learning domain. This strategy favours building generalized representations capable of solving different problems simultaneously using shared representation from similar tasks as a source of inductive bias. The addition of transfer learning helps to ground the simultaneous training of different learning objectives.

This thesis employs the transfer learning paradigm *Practice* as an auxiliary task to improve the generalization capabilities of a distributed deep reinforcement learning algorithm in the multi-task problem setting. The algorithm implements distributed learners that optimize on different task objectives and contribute to training a single representation proficient in all objectives. Experimental results indicate that Practice's addition of state dynamics information effectively improves the generalization capabilities of the algorithm. Additionally, the contributions of each learner are analyzed to study their impact on the overall multi-task learning objective.

DEDICATION

To Aniruddha Pattnaik, Leena Lama Pattnaik and Upamanyu Pattnaik for their faith and love.

To the women in my life who lift me up.

ACKNOWLEDGEMENTS

I am incredibly grateful to Dr. Minwoo Lee for giving me this opportunity to learn. His guidance and expertise have been invaluable in the development of this work. I would like to extend my sincere thanks to Dr. Min Shin and Dr. Liyue Fan for their patient support and feedback at every milestone. Their counsel provided me with the tools I needed to choose the right direction for my thesis.

I would like to offer my appreciation to the University of North Carolina at Charlotte for providing the infrastructure to facilitate my love for learning. I would like to acknowledge Chris Maher and Jon Halter at University's Research Computing Team for helping me harness the power of high performance computing.

This work could not have been done without the constant encouragement and unwavering support from my family and friends. I would like to express my deepest gratitude to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A3C An acronym for Asynchronous Advantage Actor Critic.

AI An acronym for Artificial Intelligence.

CKA An acronym for Centered Kernel Alignment

CNN An acronym for Convolutional Neural Network.

DL An acronym for Deep Learning.

DQN An acronym for Deep Q Networks.

DRL An acronym for Deep Reinforcement Learning.

MDP An acronym for Markov Decision Process

MSE An acronym for Mean Squared Error

MTL An acronym for Multi-Task Learning.

NN An acronym for Neural Network.

RL An acronym for Reinforcement Learning.

TL An acronym for Transfer Learning.

CHAPTER 1: INTRODUCTION

The Artificial Intelligence (AI) field was established to bridge machine capability and qualities of human intelligence like reasoning, inference, adapting, abstract object comprehension, and prior knowledge utilization. What is innate and inherent to human understanding currently requires AI approaches to explicitly factor each element and its impact on a singular objective. Since the term emerged in 1956, research in the field has branched in several directions to achieve the combination of mechanical computing and human cognitive ability.

Deep Reinforcement Learning (DRL) has successfully solved AI problems considered out of reach for its time. DRL is a feedback-based learning methodology that combines the principles of Reinforcement Learning (RL) [1] and Deep Learning (DL) [2] to learn sequential decision making in complex environments. Celebrated achievements include DQN [3] that solved Atari games using raw pixel values, AlphaGo [4] that mastered Go to beat the world champion in 2016, and OpenAI Five [5] that achieved expert-level performance in Dota2. These algorithms present innovative strategies and surpass human-level performance in numerous challenges.

However, DRL solutions are often developed to specialize in a single problem domain. The nature of such specialization leads to poor performance in unseen circumstances. Given the large amount of resources required to train DRL, this methodology is not sustainable. This has led to research focused on increasing training efficiency and generalized frameworks that enhance the algorithm's ability to adapt to unobserved states. Prominent approaches include transfer learning [6] and multi-task learning[7].

Transfer learning uses source task information to improve target task learning. Un-

der the assumption that the tasks share underlying similarities, the addition of source task knowledge has demonstrated reduction in training time and performance boost [8][9]. Multi-task learning focuses on learning shared representation between multiple tasks simultaneously. This approach is highly effective in building representations capable of generalizing [10][11]. Distributed learning is a popular approach incorporated into multi-task learning to scale agent training and solve problems faster.

RL algorithms like Asynchronous Advantage Actor Critic (A3C) [12] parallelize RL training by launching multiple learners to scale and decorrelate experience data for a single task objective. It can also be extended to distributed multi-task learning, where separate learners contribute to different learning objectives. However, multi-task learning suffers from distraction actuated by optimizing multiple objectives, leading to oscillations in performance [11].

This thesis incorporates Practice [13][14] paradigm into A3C to help ground multi-task learning. Practice has demonstrated the ability to accelerate RL training by transferring state dynamics information. It sets up a non-RL regression task to learn environmental dynamics by predicting state change without any expert input. Introducing practice to distributed multi-task learning supplements individual task learners with common shareable knowledge about the environments. This methodology exhibits enhancement of multi-task RL's generalization capabilities.

1.1 Problem Statement

Hypothesis: Improvement of multi-task generalization capabilities of distributed Deep Reinforcement Learning (DRL) algorithm by supplementing learning with a predictive auxiliary task.

Given tasks $t_i \in T$ in a domain, we examine the hypothesis by integrating Practice methodology as an auxiliary task t_{aux} to improve distributed DRL agent performance on task set T. The following sub-goals help investigate the hypothesis:

- Implement non-reinforcement learning auxiliary task to enhance generalization of distributed DRL algorithm in the multi-task setting.
- Develop strategies to increase the effectiveness of practice auxiliary task.
- Examine practice contributions to the multi-task learning objective.

1.2 Contributions

1. Practice for Multi-Task A3C: In this thesis, I extended the A3C algorithm with practice to supplement multi-task learning. Practice serves as an auxiliary task that recurrently optimizes the global network with state dynamics information. This approach was evaluated on a multi-task version of the Pong game.
2. Practice Sampling Strategies: Producing good state dynamics knowledge on all tasks improves practice's impact on multi-task learning. To increase the effectiveness of practice on multi-task objective, the training data is must be representative of the entire domain. I implemented two sampling strategies to increase the effectiveness of practice. They present the effect of sample source on practice representation.
3. Multi-Task Gradient Contribution Analysis: Additionally, I provide an analysis of the gradients produced between all the learners of Practice for Multi-Task A3C using Centered Kernel Alignment (CKA) [15] similarity measure. This provides the overarching impact of training a single neural network to optimize multiple objectives using practice.

1.3 Thesis Outline

In Chapter 2, concepts that build the foundation of this thesis are briefly summarized. Relevant avenues of investigation related to the proposed methodology serve as the motivational basis for the line of questioning is pursued. Deep reinforcement

learning and its generalization challenges supply the rationale to study transfer learning and multi-task learning.

In Chapter 3, the research design is presented. It provides a comprehensive view of the proposed methodologies used to examine the hypothesis. This chapter describes the design of Practice for Multi-Task A3C and the sampling strategies that aim to improve multi-task generalization. Additionally, the outline to study the contributions of multi-task learners via gradient analysis is provided.

In Chapter 4, experimental results are presented. This chapter begins with the multi-task environment set designed to test proposed methods and the experiment configurations. The results report the effectiveness of the proposed methodology on a multi-task environment set. In Chapter 5, the impact of experimental results and future work is discussed. The thesis wraps up in Chapter 6, which provides concluding statements about the proposed methods.

CHAPTER 2: BACKGROUND

2.1 Reinforcement Learning Basics

Reinforcement Learning (RL) algorithms are designed to use feedback signals to learn successful navigation within its environment. Unlike machine learning algorithms, RL algorithms are not given a fixed dataset to train. They are trained on instances of the environment and learn how to maximize actions that produce the desired behaviour.

RL methodology is built around five essential components- Agent, Environment, Observation, Action, and Reward. An RL Agent embodies the algorithm that learns how to maneuver around the environment. The universe that the agent interacts with is called the Environment. An Observation is a snapshot of the environment at a given discrete time step t. The agent receives observations recurrently and selects actions to move around the environment. Actions are the legal set of motions defined for successful navigation. The environment design includes a goal or reward signal to drive the agent's optimization towards maximizing it. The Reward is a positive or negative scalar value sent to the agent by the environment based on the action selected.

2.1.1 Markov Decision Process

Mathematically, this methodology is framed as a Markov Decision Process (MDP). It formalizes sequential decision making by an agent in the environment. In MDP, all states fulfill the Markov Property (MP). MP rules that future state dynamics are conditionally dependent on the current state alone. An observation is called state s, and change of state occur according to the system's laws of dynamics. The agent

is given a state. It selects an action a and receives reward r based on goodness of state-action selection defined by the environment design. An MDP is said to be fully observable if the agent receives complete state information. If the agent is given incomplete state information, the MDP is partially observable. Top-level agent and environment interaction can be represented by Fig 2.1.

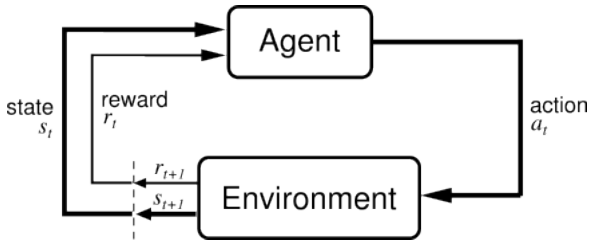


Figure 2.1: Agent-Environment Interaction [1]

MDP can be represented as a tuple in the form of (S, A, P, R), where S represents set of all possible states, A is the legal action set, P is the transition probability, R is the reward function. Transition probability P is the probability of moving from state s to next state s' given a, creating transition step. It is denoted by $P(s'|s,a)$. Reward function R produces the return reward for selection of action a in state s. The agent's understanding of the environment can be termed as model. The model of the environment defines the transition probabilities and reward function.

A trajectory is tuple (s_t, a_t, r_t, s'_t) which is state, action, reward and next state at time step t. At each t, given s, a is selected based on transition probability P to move to s'. r is the scalar value obtained that indicates goodness of action selection. State transition function $P^a_{ss'}$ can be defined as $P(s'|s,a) = P[S_{t+1} = s'|S_t = s, A_t = a]$.

An important aspect of learning is finding good Policy π. It defines the agent's behaviour in the environment. It can be stochastically represented as $\pi(a|s) = P[A_t = a|S_t = s]$. Multiple trajectories are collected over time to learn policy. These trajectories are used to calculate return or discounted future reward G_t. Which is formulated as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. Gamma γ is the multiplicative factor used to discount future

rewards and disable infinite loops in state transition.

Value function uses G_t to calculate the value of a state; represented as $V_\pi(s) = E[G_t|S_t = s]$. If the state-action value needs to be calculated, a is added to the formulation to produce the Q-value function, represented by $Q_\pi = E_\pi[G_t|S_t = s, A_t = a]$. Monte-Carlo (MC) Sampling finds the value of state $V_\pi(s)$ by fully traversing all trajectories under current policy π until terminal state s_T. This method is inefficient if the state space is large. Temporal Difference (TD) methods overcome this drawback by estimating state value using immediate reward and next state value.

2.1.2 Model-Based vs Model-Free

Model-based RL explicitly defines the model of the environment to simulate dynamics. It introduces planning to RL. Model-based RL uses planning to find optimal strategies. The algorithm creates a transition model of the environment and select actions using learned predictive control. Planning involves using simulated experience from the model of the environment to find the policy. Some variations of model-based methods use both real and simulated experiences to find optimal policy (Figure 2.2). Using a model reduces the environmental interaction required to collect samples. It leads to sample efficient RL.

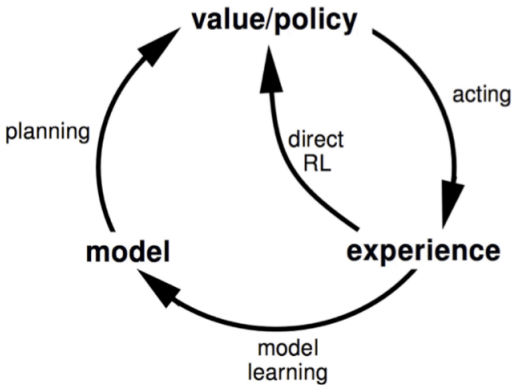


Figure 2.2: Model-Based Learning [1]

Model-based RL algorithms work well in deterministic environments. However,

large state spaces limit the effectiveness of model-based RL. These spaces exponentially increase the possible transition dynamics that the model might fail to capture. The complexity can be overcome with vast computing resources, but this is not an optimal solution. Model-free RL methods learn to map raw environment observations directly to values or action. They learn policies from incomplete information about the environment. Model-free methods do not use transition probability distribution. They directly learn from environmental interaction. Monte-Carlo (MC) and Temporal Difference (TD) methods are model-free RL approaches. Many popular RL algorithms use model-free approach to learn.

2.1.3 On-Policy vs Off-Policy

There are two strategies to enforce action selection: On-Policy and Off-Policy. On-policy methods choose actions under the current policy π. Samples from current policy result in behaviour likely to explore. On the downside, it might not utilize known patterns that could reliably yield rewards. State-Action-Reward-State-Action (SARSA) is an example of an on-policy algorithm. On-policy TD control method SARSA [1], termed after Q-value update sequence of $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, is a model-free approach which chooses actions from current policy.

Algorithm 1: SARSA

1 Initialize $t = 0$

2 Start with s_0, and choose action $a_0 = argmax_{a \in A} Q(s_0, a)$, using ϵ-Greedy method

3 At time t, we observe reward r_{t+1} and go to next state s_{t+1}

4 Select action using method in step 2: $a_{t+1} = argmax_{a \in A} Q(s_{t+1}, a)$

5 Update Q-value function $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

6 $t = t + 1$ repeat from step 3

Off-policy methods use a greedy policy to learn optimal policy and a behaviour policy to select actions. It offers more control between exploration and exploitation of

environmental knowledge. Q-Learning is an example of an off-policy algorithm. Off-policy TD control method Q-learning [1] is a model-free approach. Unlike SARSA, the greedy action in the next state is used to update the Q-value function.

Algorithm 2: Q-Learning

1 Initialize $t = 0$.

2 Starts with s_0.

3 At t, pick action according to Q values, $a_t = argmax_{a \in A} Q(s_t, a)$ and ϵ-Greedy.

4 After a_t, we observe reward r_{t+1} and go to next state s_{t+1}.

5 Update the Q-value function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r + \gamma max_{a \in A} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

6 $t = t + 1$, repeat from step 3.

Using ϵ- Greedy, random action is selected with a small probability of ϵ, every other time the highest value action is selected. Learning rate α is the hyperparameter that tunes the importance of value during training. Between 0 and 1- 0 would suggest prioritizing old values whereas, 1 suggests current information and encourages learning.

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) employ Deep Neural Networks (DNNs) to approximate functions. DNNs have been the driving force in advances and adoption of AI in several industries including, but not limited to, computer vision [16] [17] [18], audio and speech [19], natural language processing (NLP) [20] and board games [21]. The depth and structural complexity of DNNs allow rich representations to be learnt. When combined with RL algorithms, it can capture complex relationships within the environment and successfully navigate without explicit state space design. DNNs serve as function approximators for the value of state-action pairs and policy.

Games are a popular evaluation tool for DRL algorithms. They present a perceivable measure to quantify human intelligence traits. In 2015, the researchers at

DeepMind presented the Deep Q Network (DQN) [3], a deep learning extension to the Q-learning algorithm. DQN achieved superhuman or baseline scores on most of the Atari Games using raw video game input. In the following year, AlphaGo algorithm [4] beat Go World Champion Lee Sedol 4-1. It was an outcome that was considered highly unlikely. The consensus at the time placed Go as a complex AI problem, out of reach for at least a few more years. This led to a massive increase in interest in Deep Reinforcement Learning as an approach to tackle AI challenges.

In 2017, AlphaZero algorithm [22], successor to AlphaGo, reported competition-level or superior performance at Chess, Shogi, Go and the Atari games. In 2019, model-based MuZero [21] succeeded AlphaZero. In the same year, OpenAI Five [5], a 5-on-5 Dota-2 algorithm, beat the world champion team OG in a demonstration match. Robotics problems like Solving Rubik's cube with a robot hand [23], navigation optimization problems like autonomous stratospheric balloon navigation [24], or self-driving vehicles [25] are some of the interesting applications in Deep Reinforcement Learning.

2.2.1 Deep Neural Networks

Structure As seen in Figure 2.3, a Deep Neural Network(DNN) consists of an input layer, multiple hidden layers and an output layer. Outputs are determined based on how the input activates the layers of the neural network. The vector of each layer l_i calculated by multiplying its weight matrix θ_{l_w} with the previous layer's vector l_{i-1}, adding a bias θ_{l_b}, and applying an activation function. The entire network's parameter, i.e., weights and biases, can be represented by θ. The goal of learning is to improve the network's approximation capabilities by updating θ.

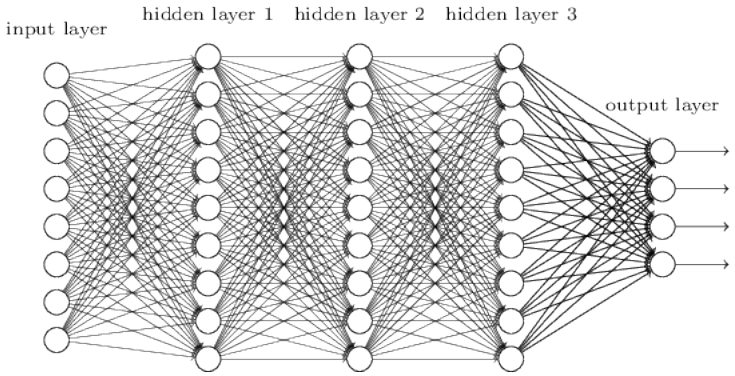


Figure 2.3: Deep Neural Network Architecture [26]

This update is performed by calculating loss and optimizing in the direction of loss reduction. The loss is the difference between the desired output and the neural network output. For example, in supervised regression problems, mean squared error (MSE) loss calculates the squared difference between provided true label and network prediction. For classification problems, cross-entropy or negative log-likelihood calculates the likelihood of network output matching target label.

The optimization process is trained on the loss function $\mathcal{L}(\theta)$ to update each layer θ_l, to improve the overall network's capability of approximating desired output. Gradient Descent optimization is used to calculate the gradient of the loss function with respect to all the parameters θ. Using this method, the parameters can be shifted to the direction that minimizes the overall loss. The Backpropagation algorithm [27] is used to calculate all the parameter gradients using the chain rule. Autograd libraries like PyTorch and TensorFlow provide tools to efficiently compute the partial derivatives needed to apply the chain rule. Finally, using a gradient-based optimizer like Adam or Stochastic Gradient Descent (SGD), the parameters are updated. Adam Optimizer is the preferred DRL gradient optimizer.

Convolutional Neural Networks (CNNs) [28] are modified DNNs that are highly successful in capturing spatial and temporal dependencies. They have been invaluable to the computer vision domain. The architecture was inspired by the brain's

neural connectivity pattern in the visual cortex. The intuition behind the CNN function is that the hidden layer shares weights over the input to produce a feature map. The neurons of the function are able to capture important characteristics/features of an image.

Popular applications include image classification [27], object detection [17] [18], reinforcement learning algorithms like DQN [3], A3C [12] and AlphaGo [22]. Figure 2.4 is a great example of CNN structure representation and how it maps pixel values from a game frame to output Q-value for each action.

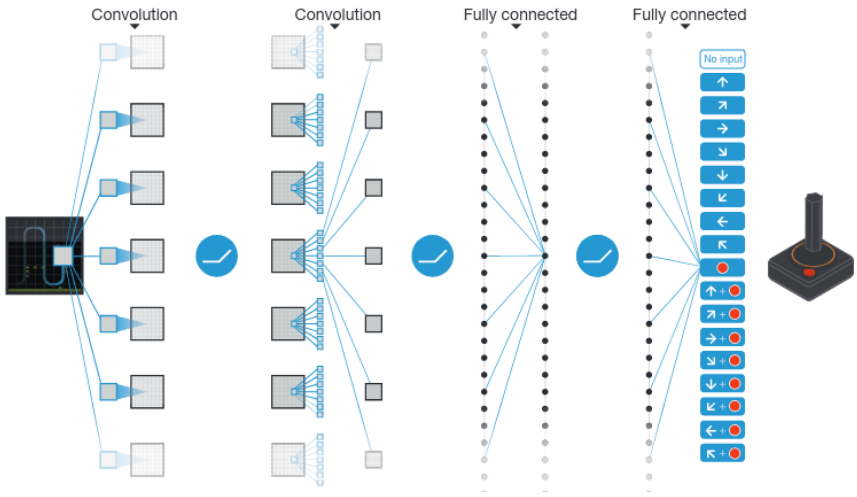


Figure 2.4: Deep Q Network Convolutional Neural Network Architecture [3]

The input is generally a 4-dimension tensor consisting of the number of input frames x input height x input width x input colour channels. The hidden layer of a CNN is called a kernel or filter and is smaller than the input. The neurons of the kernel are arranged in 3-dimensions: height x width x depth. The kernel convolution operation slides a predefined filter over the input, computing the dot product between the filter values and input region values to generate the feature map.

The ReLu layer will activate elements on the map when similar features are detected, as seen in Figure 2.5. The addition of pooling layers, like max-pooling, provide the option to down-sample feature maps and increase spatial invariance (Figure 2.6).

The fully-connected (FC) layer is the final layer, and it bridges the activations from the previous layers to the output layer. The convolutional layers are fully differentiable. Effective filters can be learnt by updating the parameters using backpropagation and optimizers.

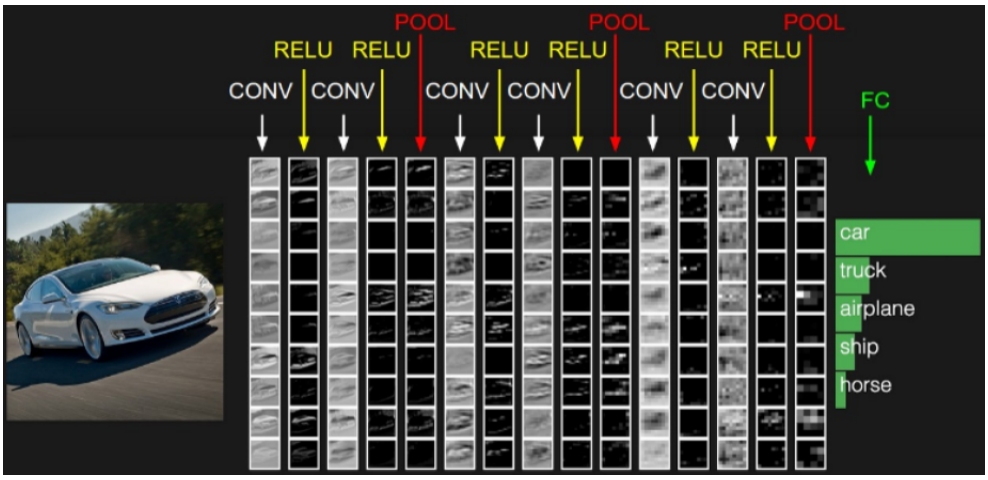


Figure 2.5: Convolutional Neural Network Layers [1]

Max-pooling is generally avoided in DRL network architectures as the algorithms require spatial information for training. Small objects need to be tracked, and excessive down-sampling can cause loss of crucial information. This design leads to an increase in the number of trainable parameters in a DRL network.

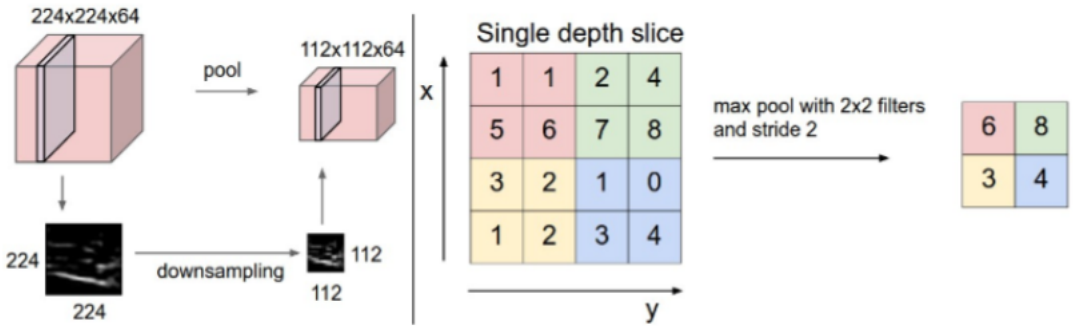


Figure 2.6: Max-Pooling Layer [2]

[1]Image is retrieved from `https://cs231n.github.io/convolutional-networks/`
[2]Image is retrieved from `https://cs231n.github.io/convolutional-networks/`

CNNs are used in value-based DRL methods use to approximate the Q-value of each possible state-action pair. For example, DQN [3] is an off-policy, value-based, model-free algorithm that uses CNNs to learn policy from raw pixels. Value-based RL assigns value to state-action selection by predicting future reward or return. The value is the calculated by estimating the total reward that can be expected. Mathematically, it can be represented by the Bellman equation: $Q(s,a) = r + \gamma max Q(s',a')$. Value estimation updates the value function $V(s_t)$ towards estimated return $r_{t+1} + \gamma V(s_{t+1})$ known as Temporal Difference(TD) target. Q function, represented by $Q(s_t, a_t) \approx E[\sum_{k=1}^{\infty} \gamma^k r_{t+k+1}]$, is approximated by the neural network. The computational impracticality of memorizing Q-value tables for all state-action pairs in Q-learning was overcome by DNN function approximation.

2.2.2 Asynchronous Advantage Actor Critic

Asynchronous Advantage Actor Critic (A3C) [12] is a representative example for a Policy Gradient based DRL algorithm. It is an on-policy, model-free algorithm that introduces efficient parallelism to RL training. The usage of state-value and advantage functions gives A3C an edge over Q-learning based methods that stabilize non-stationary Q-values.

Policy search methods directly learn policy π_θ to maximize return G of all trajectories τ, represented by $J(\theta)$. Policy gradient methods apply gradient ascent on weights θ to maximize the objective of $J(\theta)$. The gradient ascent equation can be represented as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{2.1}$$

where,

$$J(\theta) = E_{\tau \sim \rho_\theta}[G(\tau)] \tag{2.2}$$

$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta}$ is the objective function with respect to the weight. The likelihood of trajectory generated by policy π_θ is ρ. α is the learning rate and in the equation 2.1, it

is the size of step taken in the gradient ascent direction. The REINFORCE algorithm [29] provided an update to approximate policy gradient estimation (Eq. 2.3). Policies have inherently lower variance than value functions which helps performance and stability [8] [30].

$$\nabla_\theta J(\theta) = E_{\tau\sim\rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) G(\tau)] \tag{2.3}$$

Actor-Critic methods evolved from the replacement of the return of sampled trajectories with Q-value for each action, which allowed single transition or bootstrapping (Eq. 2.4). As the Q-values were unknown, an additional function approximator called the Critic was added. Φ is used to parameterize the critic (Eq. 2.4). This enables actor-critic architecture to learn parameterized policies. Most policy gradient algorithms have actor-critic architectures. The actor learns to approximate the policy, and the critic learns to estimate the goodness of policy using Q-values. Advantage Actor-Critic methods introduced a baseline called the Advantage $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ to reduce the variance of gradient methods and to multiply it to the log-likelihood of the policy.

$$\nabla_\theta J(\theta) = E_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\Phi(s,a)] \tag{2.4}$$

To keep data independent and identically distributed (IID), a large buffer of transitions is required for training. This method requires a lot of memory. A solution to this is to parallelize actor-critic methods. Asynchronous Advantage Actor Critic(A3C) achieves this by launching multiple actor-critic instances called workers or learners. As seen in Figure 2.7, using multiprocessing, the global A3C algorithm launches multiple workers to run on many environments and obtain large amounts of training data. The actors exploit policy to select good actions, and their critics learn the state-value function, which used as the baseline in policy gradient update. The workers calculate

gradients and asynchronously sends them to the global network for update. A3C removes the need to wait for all the workers to synchronize for gradient update.

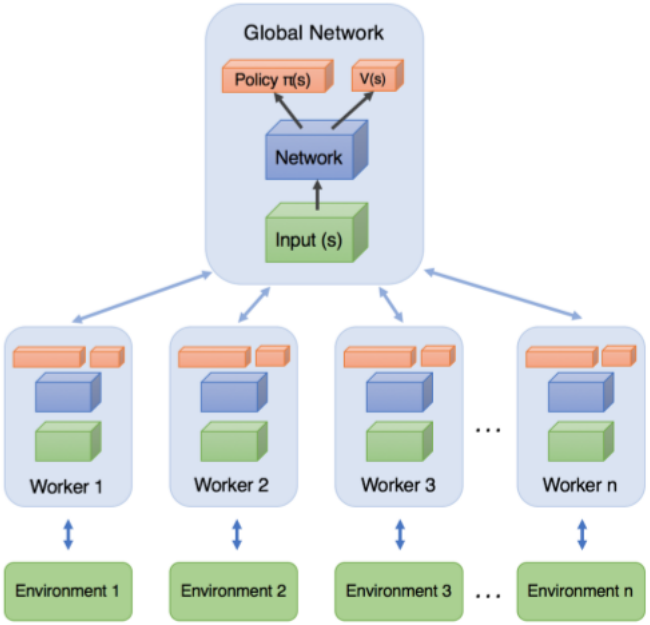


Figure 2.7: A3C Overview [3]

A3C algorithms generally use 3 CNN layers followed by two outputs heads (actor and critic). The two heads can originate from a single network or have their individual CNN networks. The actor outputs probabilities for taking each action and updates policy parameters θ for $\pi_\theta(a|s)$ as suggested by the critic. The critic outputs a single number value for the current state and updates value function parameters using $V_{\theta_v}(s)$. The policy gradient update is mathematically formalized as $\nabla_{\theta'} \log \pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v)$. Entropy regularization H is added to implement exploration by discouraging premature convergence on sub-optimal deterministic policies (Eq. 2.5). H is represented by $H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(s_t,a)\log\pi_\theta(s_t,a)$.

[3]Image is retrieved from `https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2`

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figure 2.8: A3C Algorithm [12]

$$\nabla_\theta J(\theta) = E_{s_t \sim \rho_\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)(G_t - V_{\theta_v}(s_t)) - \beta \nabla_\theta H(\pi_\theta(s_t))] \tag{2.5}$$

Entropy is maximum if the policy is random and zero if deterministic. The regularization level is controlled using β parameter. High β results in random policy, and low *beta* would be insignificant. The original A3C algorithm uses RMSProp to optimize the model. Additionally, A3C can be extended to solve continuous action space problems.

2.2.3 Distributed Deep Reinforcement Learning

Distributed DRL emerged from a demand for scalable architectures capable of increasing the number of learners and experience generators, leverage multiple strategies, and parallelizing mathematical computations. Distributed methods in DRL perform faster than a single agent and can be scaled to solve problems efficiently. A3C algorithm encapsulates distributed DRL's properties. A3C's architecture launches multiple workers that conduct learning in their subsystem and work together to achieve

the overall goal. This work narrows the scope of distributed DRL to study single agent problems that parallelize learning components.

GORILA [31] is a combination of AC methods and DQN, as seen in Figure 2.9. Multiple actors contain a Q-network that interact with the environment and generate experience. The experience samples are stored in a Replay Buffer. The learner modules sample experiences from the buffer to update the Q-network. A target network, like DQN's, is periodically updated by the parameter server that keeps track of network parameters θ. The actor's parameters are also periodically updated by the parameter server. Multiple workers and learners run within a single machine, and the samples are stored in a replay buffer. However, storing experiences in the replay buffer requires more memory and compute per interaction. Having agents run asynchronously in A3C diversifies and decorrelates the training data. A3C has parallel learning designed into its framework.

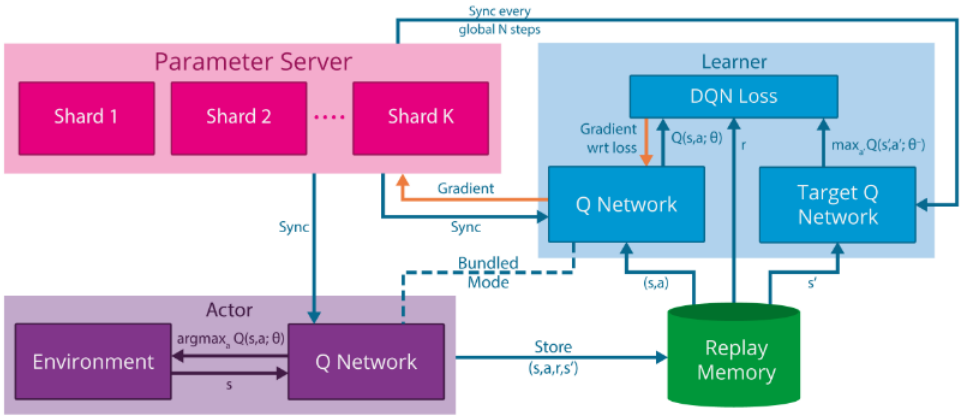


Figure 2.9: GORILA Architecture [31]

The Advantage Actor Critic (A2C) global network waits for all worker gradients to arrive before averaging and optimizing the global network. Consequently, the new parameters are sent to the worker. Distributed Proximal Policy Optimization (DPPO) [32] combines A2C's architecture with PPO's clipped objective optimization strategy. It is a continuous control problem extension to A3C. A2C and A3C have

similar performances. Asynchronous learning removes the need to wait for all the workers to synchronize for gradient update.

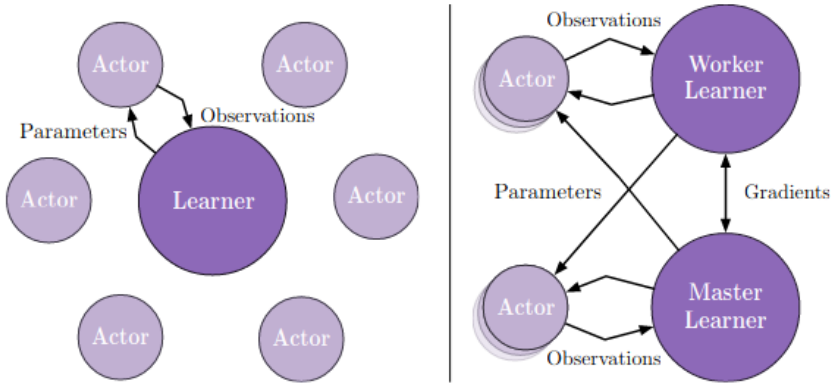


Figure 2.10: IMPALA Overview [33]

Similar to GORILA and unlike A3C, IMPALA [33] separates acting from learning. The actor generates experience, which is then sent to a central learner that computes gradients. The central learner optimizes on the policy and value function. The actors retrieve the latest parameters from the learner (Figure 2.10). The distributed actors function as a massive source of data and remove the onus of calculating gradients. IMPALA's architecture allows experimentation in the multi-task setting and reaches 59.7% median human normalized score on 57 Atari games.

Distributed methods accelerate DRL and provide a framework to allow multiple environments to interact with each other. These environments pose a challenge in the form of multiple learning objectives. A single algorithm attempting to optimize multiple tasks simultaneously requires design considerations to ensure that the different objectives are being met efficiently. Distributed DRL algorithms serve as a great tool to study the generalization capabilities of multi-task problems.

2.3 Challenges of Deep Reinforcement Learning: Generalization

Generalization is the quality that lays emphasis on an agent's ability to adapt to unobserved states and similar conditions. For example, Atari games like SpaceIn-

vaders, DemonAttack, BeamRider and Assault (Figure 2.11) are similar space shooting games, but there is variation in environmental setups and reward structures. A person proficient in any one of these games can leverage their knowledge to find commonalities and solve related tasks. The algorithmic equivalent is to create a framework that can structurally support environmental variations and learn robust policies. Most RL environments are developed in simulated environments. The overarching objective of generalization would be a smooth transition from simulation to real environment.

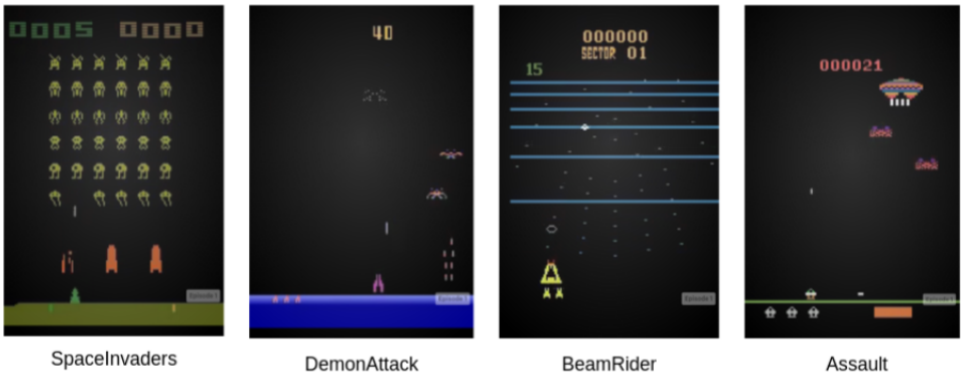


Figure 2.11: Space Shooting Games

Success in DRL began with an agent being able to solve a single problem like certain Atari games [3], Go [4] or Dota2 [5]. In the DQN paper [3], each Atari game was trained on 50 million frames. Distributed framework IMPALA trained each Atari game on 200 million frames. Atari games are rendered at the rate of 60 frames per second. Training DRL to solve these problems successfully require a lot of computing power and memory. Specialization in a single environment makes these algorithms highly sample inefficient. The curse of a well-performing DRL agent is overfitting to an environment.

Single agent algorithms are evaluated on their ability to successfully navigate an environment rather than perform well in unobserved states. With the amount of training data required for a single environment, the agent network evolves to be-

come highly specialized. The balance between specialization and generalization gets skewed. The goal is to create an agent that can generalize to unseen states. There are different strategies to approach generalization dependent on how information is leveraged. Three high level strategies are Meta-Learning [34], Multi-Task Learning [7] and Transfer Learning [6]. Meta-Learning learns an explicit representation during training that generalizes to new tasks during testing. Multi-Task Learning trains on multiple tasks in the same domain to improve performance during test time. Transfer Learning transfers information from a source task to a target task. In this thesis, Transfer Learning and Multi-Task learning are relevant to experimentation.

2.3.1 Transfer Learning

RL framework's tabula rasa learning structure is computationally expensive and tends to perform poorly on unseen states. It is sensitive to dynamic changes. DRL requires a vast number of interactions to find optimal policies and typically specializes in one task. Transfer Learning (TL) avoids the tabula rasa state by providing knowledge from a fully trained source agent or auxiliary task data. When executed under the best circumstance, similar task and domain, the target task learning speed improves. Pan and Yang's Transfer Learning survey [35] provides an excellent starting point to define generalization conventions. Although it contains formalism for supervised learning TL, it introduces the relevant concept of domains and tasks.

Domain D consists of a feature space X, and marginal probability distribution $P(X)$ over the feature space, where $X = x_1, ..., x_n \in X$. Given a domain, $D = X, P(X)$, a task T consists of label space Y and conditional probability distribution $P(Y|X)$ that is typically learned from the training data consisting of pairs $x_i \in X$ and $y_i \in Y$.

In this thesis, domain D is an environment containing sub-tasks $t_i \in T$. Source domain D_{source} contains source task T_{source} or tasks $t_i \in T_{source}$. Target domain D_{target} contains target task T_{target} or tasks $t_i \in T_{target}$. The objective of generalization

methods is to use information from the source domain effectively to adapt and improve learning for in-domain or cross-domain tasks.

TL leverages knowledge from a fully trained model in source task T_{source} and utilizes its knowledge on a target task T_{target}, assuming that tasks are similar. For example, an agent trained on T_{source} to grasp an object. This knowledge can be employed to grasp objects of different shapes, sizes and locations.

From surveys [6][36], the high-level goals of reinforcement learning TL are the reduction of training time, asymptotic performance improvement, jumpstart in performance and effective knowledge usage. Empirical observations are the norm for evaluating TL for RL. For example, observing a steeper learning slope for the target task is an indicator of successful transfer.

TL has found successful application in the supervised DL domains by fine-tuning a fully trained model in new tasks [37] [38]. However, DRL comes with its own set of challenges which make TL tricky to implement successfully. Performance rapidly deteriorates if the model does not generalize well. A dip in performance is indicative of negative transfer, which occurs when source information is orthogonal to the target task's goals.

Early TL experiments in Reinforcement Learning leveraged rule induction and classification [6] [39]. The emergence and success of DRL replaced these techniques. Neural network weights, also known as parameters, are the popular choice of source information to transfer for DRL problems. They contain abstract representations that can embody the policy function, value function, or auxiliary tasks. These representations remove the need to learn relevant features from scratch and reduce RL training time.

Cross-domain weight initialization using the source domain's neural network parameters was one of the first DRL TL approaches. In the initial TL experiments conducted on Atari games [40], fully trained models were used to initialize a new

environment's network and fully fine-tuned to solve the new environment. Performance jump in DQN agent was observed from Pong to Breakout and vice versa, but DemonAttack to SpaceInvaders experienced negative transfer. A DRL update to RL approaches like rule transfer [39], exists in the form of visual transfer. It facilitates TL by tasking an adversarial network to align the source and target task feature distribution and transfer representation to the target RL network [41]. This experiment focuses on developing a generalized representation of visually distinct domain that perform well on the target tasks.

In-domain TL occurs where the source information is transferred within the bounds of the domain. TL can be conducted by way of pre-training or auxiliary tasks. In [9], single domain A3C training is augmented by supervised learning classification pre-training task on layman human demonstration data speeds up DRL feature learning. In [42], the cosine similarity between task gradients is calculated to control auxiliary task contribution to main task learning. Auxiliary task selection does not have an exact science. Although, it is commonly chosen in relation to the main task. These tasks could be related to domain task, classification, adversarial, attention, or output prediction. Transferring from an auxiliary source has demonstrated the ability to improve RL training by providing relevant representations.

There is no established method that guarantees successful transfer, and there are no standards for measuring similarity between the source and target domains in deep learning environments. What we understand as humans is vastly different from what the agent perceives. This affects agent performance. Due to the imprecise nature of TL, the possibility of negative transfer is high, and applications need to be carefully designed to avoid performance damage.

2.3.2 Multi-Task Learning

Multi-Task Learning (MTL) is a method to improve generalization by training a single representation using similar tasks as a shared source of inductive bias [7]. Multi-

Domain Learning (MDL) is multi-task learning carried out in different domains $d_i \in D$ that contain underlying similarity. In some experiments, MDL and MTL are used interchangeably. For this thesis, MTL is conducted in a single environment domain D contain multiple tasks $t_i \in T$. The objective of MTL is to learn a generalized representation by simultaneously solving multiple tasks. MTL reduces the risk of overfitting, but it runs the risk of multiple objective distractions to overall learning.

Policy distillation [8] is an approach to leverage game-play information from fully trained DQN teacher networks and use it to train a smaller student policy network capable of solving multiple tasks. In the same vein, [10] presented the actor-mimic approach to train a single generalized representation that learns to solve multiple tasks successfully, similar to my thesis objective. It employs imitation learning by using the information from multiple DQN networks, trained to expert level on different tasks, as a supervised training signal to teach a single student multi-task policy network. In addition to DQN's employment of replay buffers to store a vast number of transition information, these methodologies rely on domain experts, which require a lot of resources to fully train.

Progressive neural networks [43] approaches multi-task learning by maintaining a pool of trained A3C models on different tasks and using lateral feature learning to add new tasks. This framework incorporates lifelong/continual learning into its objective and is designed to avoid catastrophic forgetting and negative transfer. A new neural network or column is added to the pool with every task addition. It is integrated by leveraging lateral connections to previous columns and transferring relevant information. Although this method presents successfully solves many complex multi-task and multi-domain learning problems, it becomes computationally expensive as the number of new tasks increases.

In [12], A3C was introduced with results that indicated faster performance over DQN and GORILA (Section 2.2.3). It removes the need for a replay buffer by launch-

ing multiple workers that learn policy in their assigned environments and decorrelate training experience. A3C's advantages lie in its parallelizable, asynchronous framework that can utilize the multithreading capabilities of CPUs. This makes it suitable for simultaneous training and distributed learning.

In contrast, popular actor-critic method IMPALA [33] separates acting from learning and launches multiple actors instead. When trained in the multi-domain setting, it achieved 59.7% median human normalized score on 57 Atari games trained on a total of 11.4 billion frames. It performed better than A3C on multiple benchmarks but did not achieve state-of-the-art performance in each game. It's successor PopArt-IMPALA [11] uses scale invariant normalization methodologies to address the balance between optimizing the objective for multiple domains and single learner resources. It achieved 110.7% median human normalized score, surpassing human-level performance in DRL multi-domain, thus presenting a method highly effective in solving MTL and MDL challenges.

Unlike IMPALA, A3C workers calculate the gradients and send them to the global network for update. It does not separate acting from learning. The gradients are intrinsic to the learning process of neural networks to find features that optimize task objective. This structure allows the A3C workers to be independent in their understanding of the environment assigned. Multiple workers launched by A3C distribute the learning, whereas IMPALA launches actors that generate trajectories and send them to a central learner. A3C's framework can be extended to understand the behaviour of distributed multi-task learners. While IMPALA has its advantages, studying multi-task generalization with respect to distributed learners is the primary focus of this thesis.

2.4 Practice

Practice [44][13] is a transfer learning paradigm that initializes the RL network using the representation from a pre-training task focused on state dynamics prediction.

It finds shareable knowledge between the non-RL prediction task and the RL task. It does not require expert information to train and speeds up RL training. As seen in Figure 2.12, this approach uses a single neural network with two output heads to predict state difference and produce Q-values.

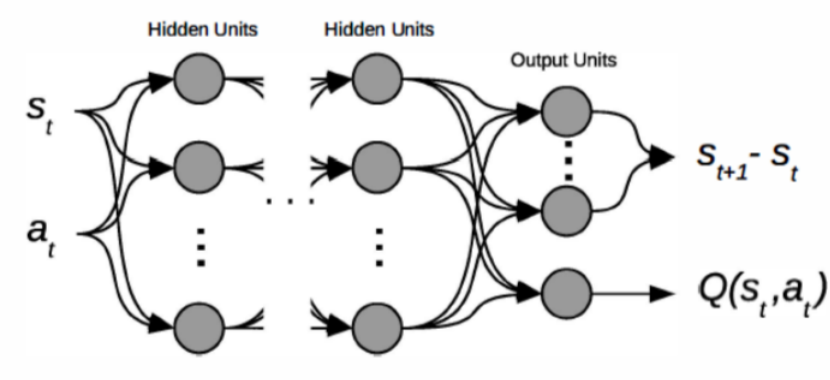


Figure 2.12: Practice Architecture [13]

This methodology was expanded into DRL in [14] by integrating it with the DQN algorithm. This was achieved by separating the practice task and RL task into two networks. The DQN network (Figure 2.13 (b)) was initialized by transferring the parameters from the pre-trained practice network (Figure 2.13 (a)). Additionally, iterative practice was introduced, where practice was conducted recurrently for a short duration to supplement RL training.

During training, samples (s_t, a_t, s_{t+1}) are collected using random action selection. Practice does not use reward to train. The ground truth labels are calculated using $s_{t+1} - s_t$. It can be represented as $\Delta s_t \approx s_{t+1} - s_t$. The practice network is used to predict Δs_t and is formalized by function $f(s_t, a_t; \theta) \approx \Delta s_t = s_{t+1} - s_t$, which predicts the approximate difference between next state and current state. The parameters of the practice task θ are optimized on the MSE loss between the difference of ground truth label Δs_t and network estimation of $f(s_t, a_t; \theta)$.

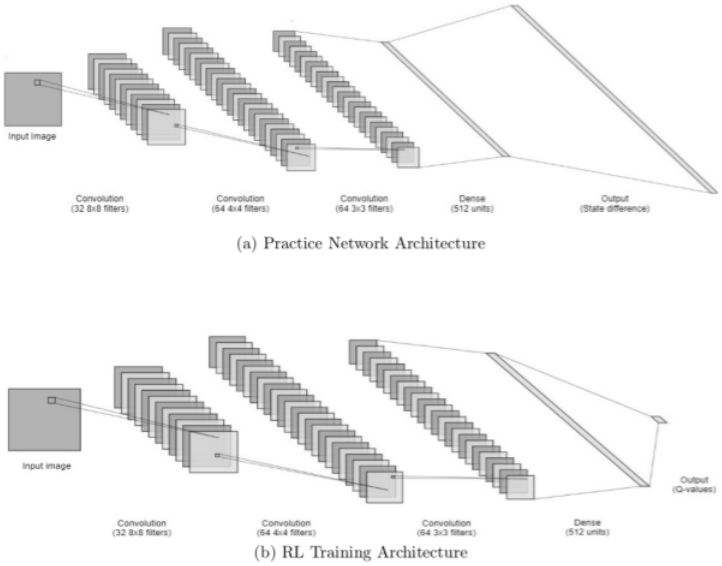


Figure 2.13: DRL Practice Architecture [14]

A similar model-based approach conducts unsupervised pre-training on video frames to learn task agnostic dynamic priors [45]. The dynamics model is initialized by the predictor, which aims to capture the physics of the environment by enabling future frames predictions. The future frames generated by the dynamics model are then used as additional input to the policy network. Although model-based methods are sample efficient, they suffer from variation in performance and generally do not achieve the state of the art results.

Sample efficient version of model-based MuZero algorithm [46], MuZero Reanalyze, achieves state-of-the-art performance on 57 Atari games using 200 million frames of experience per game. It models parts of the environment that are important to decision making and does not require knowledge of environmental dynamics. At a smaller experimentation scale, practice has demonstrated to be sample efficient in solving model-free RL problems.

Practice is similar to imitation learning approaches [8] which uses a teacher signal to guide learning but does not require multiple trained RL teacher environments

or expert supervision. Instead, it supplements RL with shareable knowledge about environmental state transition. Practice and RL combined have shown to learn features faster and help the model generalize better in single domains. Implementing regression to predict state difference and capture change creates an implicit model of the environment dynamics. My work builds upon this approach to study practice's impact on multi-task generalization in the asynchronous distributed learning setup.

CHAPTER 3: PROPOSED METHODS

Transfer learning paradigm Practice [44][13][14] uses a non-RL regression task, that predicts state dynamics, to enhance RL task performance. This method has demonstrated a reduction in RL training time and good generalizability in single agent settings. The state dynamics representation learnt by practice shows promise in guiding RL to adapt to new tasks. This thesis explores practice generalization capabilities and effectiveness in the asynchronous distributed learning setup.

The A3C algorithm is a lightweight, on-policy DRL framework that distributes gradient calculation by launching parallel actor-critic workers. These workers produce gradients that are endemic to their network's learning and asynchronously push their updates to the global network. The nature of this framework provides independence to individual workers and allows them to build their understanding of the environment. This can be extended to study distributed multi-task learning where each worker learns a different task and the goal is to find a single representation capable of solving all the tasks.

Conflicting objectives between tasks cause distraction. When a single representation is learning multiple tasks, the priority keeps shifting to serve the needs of different tasks. For example, performance increase in one task can harm the other tasks if there is orthogonality between tasks objectives at the given instance. This can cause learning to slow down. The proposed **Practice for Multi-Task A3C** incorporates model-free iterative practice [14] into the A3C ecosystem to improve multi-task generalization for distributed learners and provide common dynamics knowledge to boost multi-task RL training.

In this approach, practice is an auxiliary task that learns the underlying environ-

mental dynamics by predicting state difference. Two data sampling strategies are implemented to analyze the impact of data sample origin on the learning objective. The first strategy uses samples from the A3C workers, and the second utilizes the global network's policy head to produce samples. Additionally, the relationship between A3C's learners is examined by comparing their gradients using Centered Kernel Alignment(CKA) to study gradient contributions towards the multi-task learning objective.

3.1 Practice for Multi-Task A3C

The A3C algorithm is primarily selected to study the generalization capabilities of independent distributed learners working towards a common representation. The ecosystem allows practice's contributions to be incorporated towards the overall multi-task objective. The goal for Practice for Multi-Task A3C is to enhance generalization of the A3C algorithm in the multi-task setting using non-RL regression auxiliary task representation. For experimentation, the term environment is used interchangeably with task. These tasks are different objectives $t_i \in T$ in a single domain D. Practice is an auxiliary task t_{aux} that is does not have an RL task objective. It is a supervised learning regression task that does not involve feedback-based variables like rewards and shares its representation to help RL.

The main A3C algorithm launches multiple instances of its global actor-critic network architecture. These instances are called workers or learners. The workers train in their assigned environments and calculate the gradient of the loss function with respect to their network parameters. As seen in Figure 3.1 (left), the calculated gradients are asynchronously sent to the global network for optimization update. The main algorithm maintains a queue that can accommodate sparse or synchronous updation from different combinations of workers. It does not wait for all the workers to finish calculating their gradients. In Figure 3.1 (right), an additional practice worker is added. The practice network's gradients aim to supplement the global network's

learning with state dynamics information. There are two strategies with different assumptions implemented to train the practice worker. They are discussed further in Section 3.2.

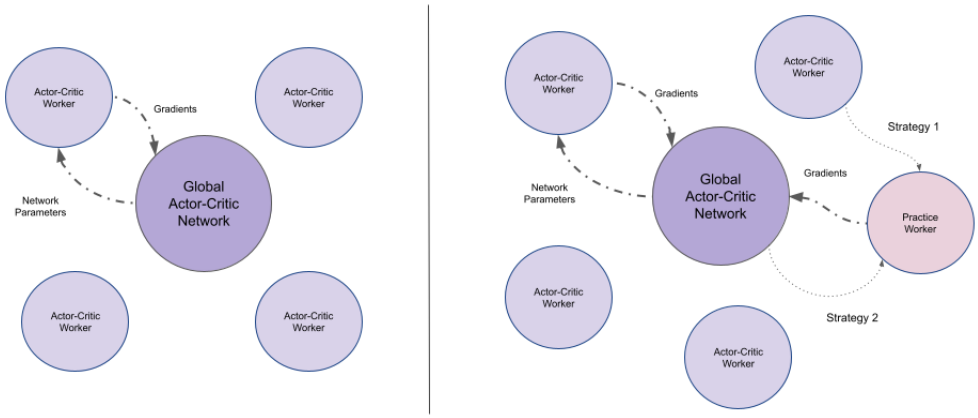


Figure 3.1: Asynchronous Components Overview: A3C (left) and Practice for Multi-Task A3C (right)

A3C Network Architecture The global actor-critic network consists of 3 convolution layers. The input layer feeds the first convolutional layer raw pixel value input of size 84x84. The first layer contains 32 8x8 filters with stride 4 and followed by ReLu activation. The second layer convolves 64 4x4 filters with stride of 2, followed by ReLu. The third convolutional layer convolves 64 3x3 filters with stride of 1 and is followed by ReLu. The output of the convolution layer stack is connected to two output heads- policy "actor" head and value "critic" head. Each head is connected to the convolution layers output by a 512 hidden unit fully-connected layer. The actor head outputs logits of n actions, where n depends on the environment's legal action set. Softmax is applied to find the probability distribution over the logits. The value head produces the value of the state. A separate forward pass is required to compute the value of each state. The main algorithm launches multiple instances of the global actor-critic network called workers with the same network configuration (Figure 3.2).

Each worker's network uses Adam optimizer to calculate its gradients. The gradients are calculated on the Policy Gradient update (Eq. 3.1) with entropy regular-

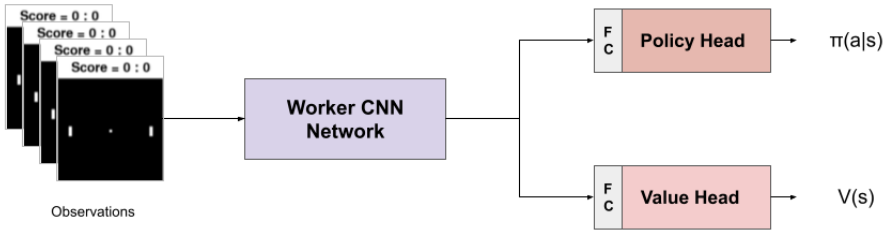


Figure 3.2: Worker Network

ization H. Where $H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(s_t, a) \log \pi_\theta(s_t, a)$. Entropy Beta β parameter controls exploration-exploitation. The calculated gradients are sent to the main algorithm's queue that updates global network. PyTorch's multiprocessing allows asynchronous parameter sharing and updates the worker networks to update without explicit coding [47].

$$\nabla_\theta J(\theta) = E_{s_t \sim \rho_\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)(G_t - V_{\theta_v}(s_t)) - \beta \nabla_\theta H(\pi_\theta(s_t))] \tag{3.1}$$

Practice Network Architecture Practice has the same CNN architecture as A3C's CNN. It is configured using 32 8x8, 64 4x4, 64 3x3 filters (Figure 3.3). The value head is replaced by state difference head which outputs $\Delta s_t = s_{t+1} - s_t$. Practice network is also trained on raw pixel input of size 84x84. The network is used to predict $f(s_t, a_t; \theta)$ (Eq. 3.2), which is the approximate difference between next state and current state. The practice parameters θ are optimized on the MSE loss between the actual state difference Δs_t and network output state difference $f(s_t, a_t; \theta)$.

$$f(s_t, a_t; \theta) \approx \Delta s_t = s_{t+1} - s_t \tag{3.2}$$

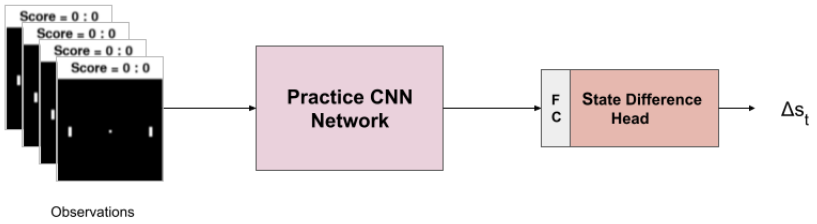


Figure 3.3: Practice Network

Practice for Multi-Task A3C Algorithm The workers of this algorithm are denoted by N. The algorithm contains one practice worker N_p with auxiliary task t_{aux} and $N_i \in N$ A3C workers paired with tasks from the task set T. These workers are launched by the main algorithm that controls the overall process. Each worker periodically sends gradients to the global network (Figure 3.4) via a queue managed by the main algorithm. The algorithm is designed to incorporate the short training structure of iterative practice [14] into A3C's asynchronous updation. Unlike [14], A3C's practice worker periodically transfers gradients instead of neural network parameters.

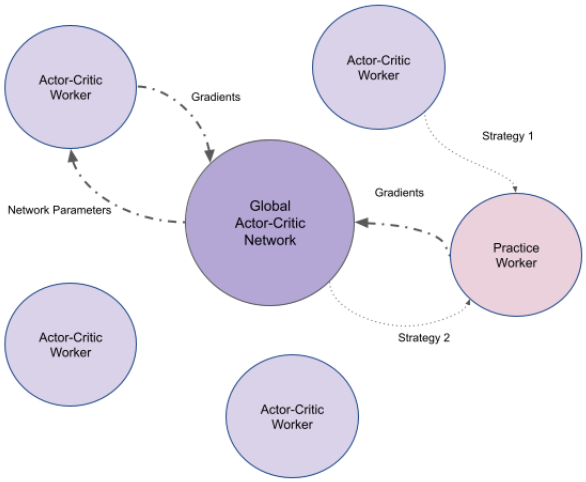


Figure 3.4: Practice for Multi-Task A3C: Asynchronous Components Overview

Gradients contain information that suggests the direction of network loss optimization. The intuition behind using practice gradients is to use the knowledge of an auxiliary task focused on predicting underlying environment dynamics to supplement

the multi-task learning objective. The aim is not to optimize practice regression but to use the shared knowledge representation to improve multi-task learning.

Training As seen in Figure 3.5, there are two steps to training. The algorithm starts at Step 1, where initial practice is conducted. Pre-training TL is done by training the practice network for a certain number of frames and using the developed representation to initialize A3C's global network. Initial practice trains on samples (s_t, a_t, s_{t+1}) generated by all task environments T using random action selection. The practice network is trained to perform state dynamics regression. Rewards are not used in practice's regression task. During training, samples are collected in batch sizes of e_{pr}, and the network is optimized using Adam optimizer on MSE loss between Δs_t and network output $f(s_t, a_t; \theta)$. After initial practice, the parameters of the first two practice convolution layers are transferred to the global A3C network. The lower layers of CNNs contain generalized features. Transfer-based initialization on networks results in better performance over random initialization [48].

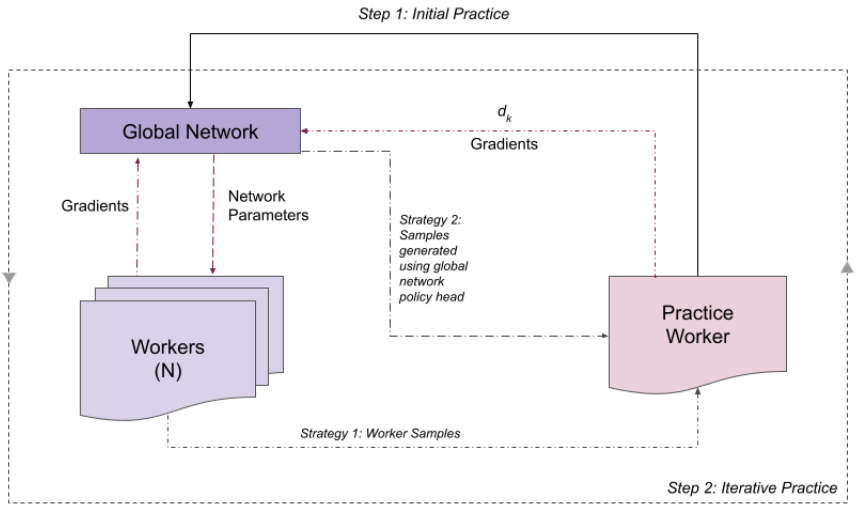


Figure 3.5: Practice for Multi-Task A3C Training Overview

Once the global network is initialized, Step 2 starts with the main algorithm launching N instances of the global network (workers). Each worker is initialized to a corresponding task in task set T. Workers interact with the environment, collect

experience in batches of e_i, train, and asynchronously push their gradient updates to the global A3C network. The worker's network trains and optimizes on the actor-critic loss using Adam optimizer. Once the gradients are calculated, they are sent to the global network for updation via a queue managed by the main algorithm. Unlike the A3C workers, the practice worker sends gradients to the global network at specified intervals d_k. During Step 2, short practice is conducted recurrently. This repetitive inclusion of an auxiliary task's gradients periodically updates the network's parameters with state dynamics information.

DRL algorithms interact with highly dynamic environments and require a substantial amount of samples to navigate successfully. When the agent interacts with the environment, the sequential trajectories run the risk of being highly correlated. Learning on a set amount of samples require considerations to decorrelate the input. For example, in model-based methods, sample efficiency is achieved by fixing the number of training samples. The samples must be updated regularly to ensure that the model operates on a good representation of the environment. Model-free methods like DQN use a large replay buffer to store experience and sample from it.

A3C is an on-policy method that typically does not use replay buffer to store trajectories. Samples are generated and decorrelated using parallel learners. Multi-task A3C shifts learner's focus from collectively working on a single task to different tasks. To ensure that the A3C worker's training data does not stagnate, multiple instances of the assigned environment can be launched by the workers to gather training data. To decorrelate the data, Practice for Multi-Task A3C uses PyTorch toolkit PyTorch AgentNet (PTAN) [49] to manage trajectory generation.

PTAN library [1] is the PyTorch implementation of popular deep reinforcement learning frameworks and tools designed to aid research. It offers various functionalities that make DRL experimentation efficient. PTAN's Agent class provides the means

[1]The library is available from `https://github.com/Shmuma/ptan` (Accessed: 09 December 2020)

for a neural network to communicate with the environment. It implements a function capable of accepting a batch of observations and producing the corresponding batch of actions dependent on the neural network it is initialized on. This is designed to reduce the computation time associated with processing each trajectory individually. The Agent class can be initialized on different action selection strategies like argmax or logits sampling.

The Agent class paired with PTAN's ExperienceSource functionality provides control over the trajectory granularity. It can provide step by step trajectory data, generate trajectories from instances of the same or different environments, and be specified to accumulate rewards from sub-trajectories. A single step generated by ExperienceSourceFirstLast class can be represented by tuple $ExperienceSourceFirstLast(state = 0, action = 1, reward = 1, last\_state = 1)$. Under the hood, ExperienceSource resets the environment, the initialized agent selects action based on reset state, executes step function to get reward and next state, provides the agent with next state, and repeats. Using this functionality, multiple instances of an environment can be launched by each worker to generate decorrelated trajectories.

3.2 Samples for Practice

The quality of samples largely contributes to an agent's overall understanding of its surrounding. The samples that practice trains on are representative of current obstacles that the environment is navigating. Shared knowledge learnt by the practice network is dependent on samples that provide the best depiction of the environment. Practice does not explore. It selects random actions independent of reward considerations. Limited movement in the environment could reduce the impact of practice's representation on the overall learning.

During Step 1, initial practice trains on samples generated the environments sets using random action selection. Step 2 bifurcates practice for multi-task A3C into two approaches. Two different sampling strategies are presented based on different

assumptions about the source. The goal is to ensure that the training data is representative of the multi-task environments. The first uses samples that the A3C worker trains on, and the second uses the global network's policy head.

3.2.1 Shared Experience Samples

In [14], iterative practice is trained on DQN's replay buffer, effectively reducing interactions between practice and the environment, and this method ensures that the training data for practice represents the current state of the environment. Extending this approach to Practice for Multi-Task A3C trains practice on samples generated by A3C workers for their training. The experience that each A3C worker trains on is task-specific. The trajectories generated are relevant to the worker's current progress in its environment. Using task-specific samples to train practice network works under the assumption that learning generalized environmental dynamics from specific environments can benefit the worker's current navigation challenges and help the overall multi-task objective.

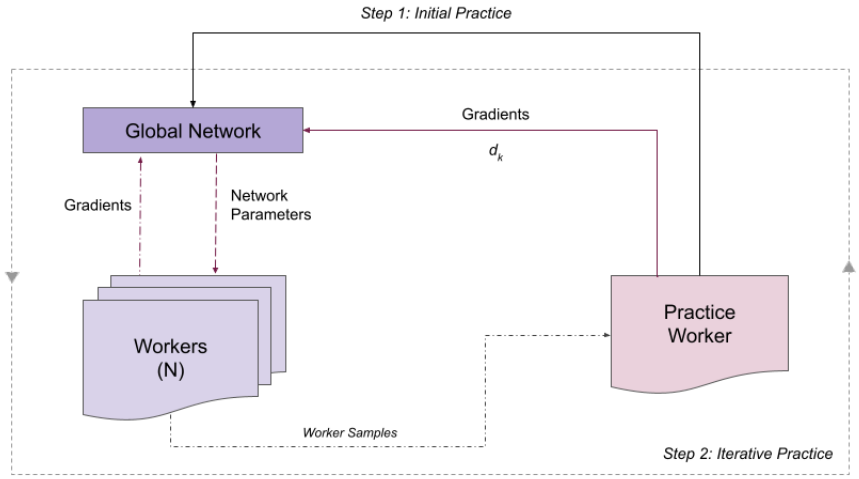


Figure 3.6: Shared Experience Sampling Strategy

As seen in Figure 3.6, in Step 2, batch training samples from each worker is sent to practice. The experience trajectories for workers take the form of tuples $(s_t, a_t, r_{t+1}, s_{t+1})$. As practice does not require rewards, that information field is not

used. Practice network optimization is a continuous process that trains on these worker samples. The intuition behind this method is to train the practice network on the same data that each worker is training to optimize its objective. Setting a high interval value for practice update d_k will ensure that the practice representation incorporates samples from all the workers uniformly. After every d_k global network update, the first two CNN gradients from practice update is used to optimize the global network.

3.2.2 Samples using Global Policy Head

After Step 1, the main algorithm launches N workers on different tasks ($t_i \in T$) along with the global network. This network becomes endemic to the worker environment and optimizes the specific task objective. The asynchronous nature of A3C entails workers operating on lagging versions of the network. The most updated network of the A3C is the global network. Generating practice samples from the global network ensures that the dynamics learnt represents the overall multi-task objective.

On-policy samples from the network's understanding of the multi-task environments can be generated using the global network policy head initialized on each task environment. Practice does not explore but this method could provide training data focused on the areas of interest. The worker's network lags behind the global network, and their independent learning is primarily focused on their optimization problem. Shifting the environmental dynamics from endemic sources to a global source favours generalized understanding over specialized understanding.

As seen in Figure 3.7, the global network's policy head is used to generate samples for Practice after every d_k network update. The samples are collected by iterating over task set T. Single step trajectories are produced, and the reward field is not used by practice training. In Step 2, practice occurs after k global network updates, and at every d_k step, short practice is conducted for a set number of frames. The global

network's policy head is given the set of multi-task environments T and generates samples by iterating over the set till the batch size e_{pr} is met. The samples are generated, MSE loss is calculated between Δs_t and network output $f(s_t, a_t; \theta)$, and the practice network is optimized using the gradients generated. The gradients are then sent to optimize the global network. Only the first two convolutional layer gradients are incorporated in global network optimization. The intuition behind this to use the general information captured by the lower levels of practice CNN. Training ends when the threshold value of tasks is achieved.

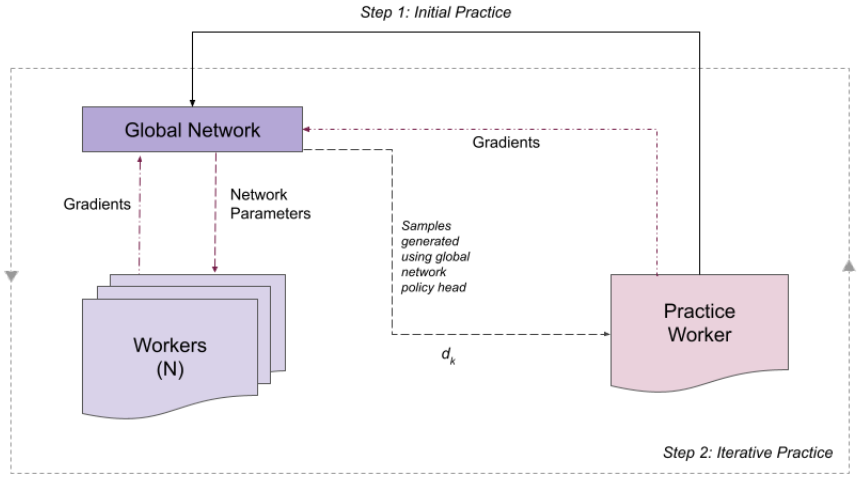


Figure 3.7: Global Network Sampling Strategy

PTAN's Experience function can be used to generate samples and maintain the environment set. Using PTAN ExperienceSourceFirstLast class initialized on the global network's policy head and the set of multi-task environments, samples can be produced for practice. Multi-environment sampling is implemented in round-robin fashion. The practice function requires the most updated version of the global network for training. The global network's policy head produces single step trajectories, and practice training does not use the rewards associated with the tuples generated.

3.3 Multi-Task Gradient Contribution Analysis

Distributed learner architectures like A3C utilize multiple learners to optimize a single objective. When A3C is extended to the multi-task setting, the number of objectives increase. This comes with certain advantages and disadvantages. Training a single network to solve multiple tasks improves the generalization capabilities of the model. The different tasks encourage the agent to balance specialization and generalization. The downside to this approach is multiple objectives vying for a single network to optimize on. Multi-task A3C's workers periodically send gradients to the global network for updation. The gradients calculated for the worker's global network copy represent the direction of its loss reduction. If the gradients are similar, multi-task learning improves. If the gradients are orthogonal to each other, they harm the multi-task learning objective. Understanding the underlying task similarity can help us determine multi-task learning success.

Centered Kernel Alignment (CKA) [15] was designed to provide a measure to understand representation similarity between deep neural networks. This method is invariant to invertible linear transform, orthogonal transformation and isotropic scaling. CKA is made invariant to isotropic scaling by normalizing Hilbert-Schimidt Independence Criterion (HSIC). HSIC (Eq. 3.5) determines the independence between $K_{ij} = k(x_i, x_j)$ and $L_{ij} = l(y_i, y_j)$, where k and l are kernels, and H is the centering matrix. The inter-example dot product-based similarity between the representations of network X and Y, XX^TYY^T (Eq. 3.3) is centered (Eq. 3.4) and generalized (Euclidean space to Hilbert space) Eq. 3.5. Radial Basis Function (RBF) CKA uses parameter σ to control weightage of smaller distances over larger distances.

$$< vec(XX^T), vec(YY^T) >= tr(XX^TYY^T) = \|Y^TX\|_F^2 \tag{3.3}$$

$$\frac{1}{(n-1)^2} tr(XX^T YY^T) = \|cov(X^T, Y^T)\|_F^2 \tag{3.4}$$

$$HSIC(K, L) = \frac{1}{(n-1)^2} tr(KHLH) \tag{3.5}$$

$$RBF\ CKA = \frac{tr(KHLH)}{\sqrt{tr(KHLH)tr(KHLH)}} \tag{3.6}$$

CKA scale lies between 0.1 and 1, where 0.1 indicates low similarity and 1 is the highest similarity value. CKA was able to capture the relationship between the same CNN with two different initialization better than pre-existing methods [15]. Considerations of transforms during neural network operations provides a reliable measure of similarity. Gradients contain information on the direction of loss minimization. Quantify similarity between gradients of different tasks using CKA can help visualize single learner contribution towards multi-task learning objective.

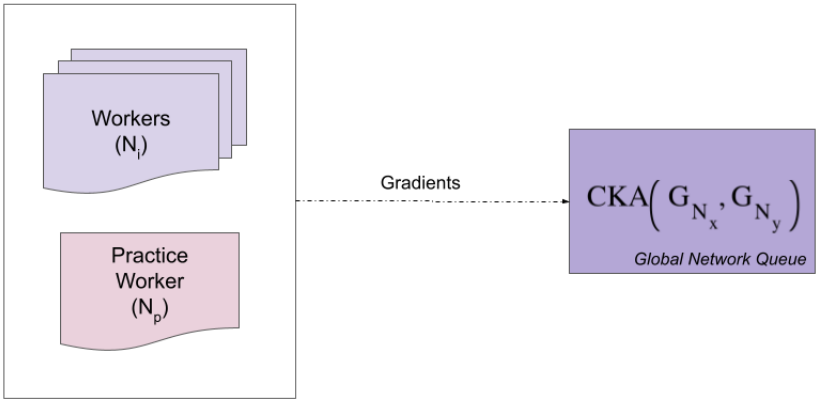


Figure 3.8: Gradient Analysis Overview

Practice for Multi-Task A3C gradient updates are batched for global network optimization, but for this experiment, they update sequentially. Workers put gradients in a queue collected by the global network updation algorithm (Figure 3.8). The gradients from A3C actor-critic worker can be represented as G_{N_i}, where $N_i \in N$

workers are assigned an environment from the environment task set ($t_i \in T$). G_{N_p} are the gradients from the practice worker. In the CKA formulation, x and y can be a combination between an A3C worker from task set T and practice worker p or between the A3C workers ($t_i \in T$). For example, if the task set is $T = 4$ environments, CKA between workers gradients can be represented by $CKA(G_{N_1}, G_{N_3})$ and between worker-practice gradients as $CKA(G_{N_2}, G_{N_p})$.

As this experiment is conducted on the Global Network Sampling strategy, the practice gradients are calculated periodically. A separate function was created to keep a copy of the latest practice gradient G_{N_p} and calculate the CKA value with the worker gradients G_{N_i} that arrive at the global network queue. The information in the queue is representative of the direction of each worker's loss at the same time. Thus, the CKA of worker gradients G_{N_i} in the queue compared to each other. The relationship between tasks can provide insight into how they could be better adapted to solve the multi-task objective.

CHAPTER 4: EXPERIMENTS AND RESULTS

4.1 Environment

The similarity between tasks resides on a spectrum, and algorithms that capture the underlying similarity are good at generalizing to unseen states. To illustrate the efficacy of the proposed methods, multiple tasks were designed into the Pong environment domain. In the micro-world of Pong, I designed a few variants to increase the difficulty and introduce different objectives to test the multi-task generalizability of the proposed methods.

Pong consists of two paddles that move vertically in competition to ensure the ball does not cross its boundary. If the ball goes out of frame, 1 point is given to the opposite paddle player that sent the ball. The computer is the right paddle in this game setup. The max score achievable by classic Pong is 21. I have set the threshold at 15 points to set a baseline performance for all variants. The Pong variants were created using PyGame[1].

Pong Variants:

- V1.0: Original Pong game.
- V1.1: Increase the ball size to make an easy, perceivable version.
- V1.2: Increase in paddle size to make easier defensive plays.
- V1.3: Decrease in paddle size to make harder defensive plays.
- V2.1: Rectangular block added perimeter to increase difficulty.

[1] Adapted from `https://github.com/xinghai-sun/deep-rl` (Accessed: 01 February 2021)

- V2.2: Floating immovable rectangular block added to increase difficulty.
- V3.0: Additional ball to increase difficulty.
- V3.1: Additional ball and floating rectangular block added to increase difficulty.
- V3.2: Additional ball and perimeter rectangular block added to increase difficulty.

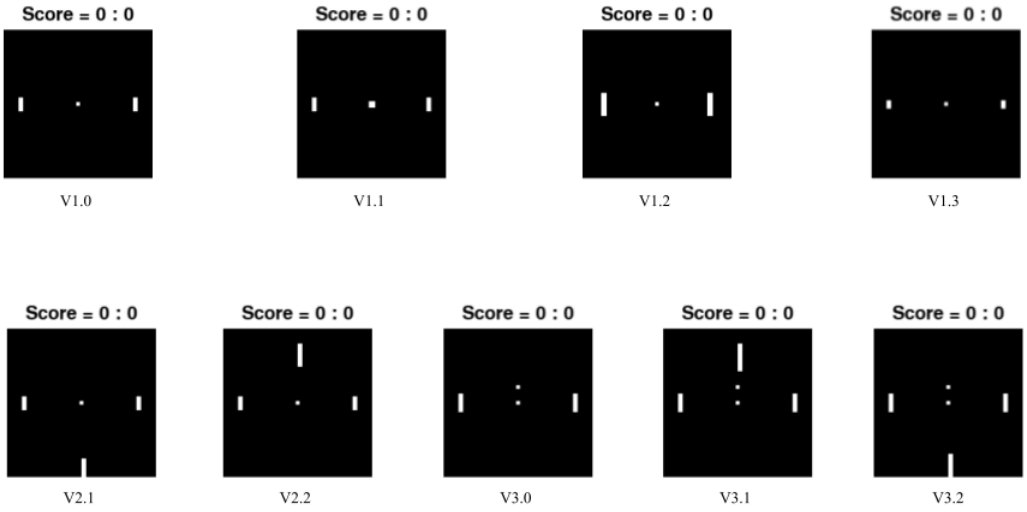


Figure 4.1: Pong Environments

These environments have varying levels of difficulty. The environments under V1 have changes to the main variables- the two bats and the ball. They were designed to introduce small changes to challenge the perception of neural networks. V1.3 is fairly harder to master as the bat size area was reduced and requires more time to cover their respective bases. The V2 environments pose a challenge in the form of stationary obstacles. The agent has to maneuver the block and successfully score points. V3 environments are much more difficult as there is an additional ball that the agent needs to balance. The additional ball does not reset the environment. The

stationary blocks added to V3.1 and V3.2 make this environment variant much more difficult to solve. The max score is raised to 25 for these variants to account for the additional ball's scoring capabilities.

4.2 Practice for A3C on Pong

Environment Training Set: The Pong variants are divided into two sets to test the proposed methods. The Easy Set containing four environments- the original Pong environment (V1.0), decreased bat size (V1.3), additional perimeter block (V2.1), and floating block (V2.2). This set contains environmental modifications that obstruct the ball's trajectory and challenge the agent's ability to work with a smaller bat surface. The Hard Set is more challenging than the Easy Set. It consists of the original Pong environment (V1.0), additional perimeter block (V2.1), and additional ball along with a perimeter block in the environments (V3.1 and V3.2). Keeping track of the second ball and environment hindrance makes this set difficult. For Easy Set, environment max score is set to 21, and for the Hard Set, the maximum score is set to 25 on account of the additional ball rewards.

Experiments Three algorithms are evaluated- Practice for Multi-Task A3C with Shared Sampling, Practice for Multi-Task A3C with Global Sampling, and baseline Multi-Task A3C. The algorithms are evaluated based on their performance in Pong Easy Set and Hard Set environments. During training, the positive rewards are clipped at +1 and negative rewards at -1. This is done to limit the scale of error derivatives and make it easier to learn multiple environments.

Training PyTorch Multiprocessing is used to launch A3C workers on different CPU or GPU processes. Every worker is initialized with the global network and assigned an environment. For clarity, the worker's copy of the global AC network is referred to as the worker network. As there are four environments in one set ($T = 4$), there are four workers ($N = 4$). 84x84 Pong game frames are generated with frameskip of 4 to reduce computation. Each worker trains its network on mini-batch samples of

size 64 ($e_w = 64$).

The actor head of the worker neural network selects actions using softmax, which rank each action according to their action-value estimate. The critic's head assigns value to the states. The actor-critic loss is calculated, and the gradients of the loss function with respect to all the network parameters are calculated. These gradients are then added to a queue which is managed by the main algorithm for global network updation. The global network uses Adam optimizer on gradient batch-size of 2 (Train Batch). The model was tested on learning rates of 0.001 and 0.0001. The former was selected due to consistent performance. The best performing hyperparameters for baseline A3C's are listed in Table 4.1.

Table 4.1: Multi-Task A3C Hyper-Parameters

Hyperparameters	Value
Mini-Batch Size	64
Learning Rate	0.001
Optimizer	Adam
Gamma	0.99
Clip Grad	0.1
Train Batch	2

Practice: Shared Experience Sampling During initial practice the practice network is trained on about 1×10^5 frames, using a batch size e_{pr} of 64. A dull agent is used to select random actions and generate samples using PTAN's ExperienceSourceFirstLast class. The difference between new state frame s_{t+1} and current state frame s_t produces the Δ for the mini-batch. The MSE loss is calculated between Δ and practice network output of $\Delta_{predicted}$ when fed the batch's current state s_t. The practice network parameters are updated using Adam Optimizer using a learning rate of 0.00001 on the MSE loss. Learning rate of 0.00001 was selected after 0.001 and

0.0001 produced inconsistent results. Additionally, e_{pr} of 32, 64, 128, 512 were tested. $e_{pr} = 64$ results provided consistent and computationally efficient results. The best performing hyperparameters are listed in Table 4.3.

Table 4.2: Practice: Shared Sampling Hyper-Parameters

Hyperparameters	Value
Worker Batch Size	64
Learning Rate	0.001
Optimizer	Adam
Gamma	0.99
Clip Grad	0.1
Train Batch	2
Practice Learning Rate	0.00001
Practice Loss	MSE
Practice Optimizer	Adam
Practice Batch Size	64
Initial Practice	100000 frames
Iterative Practice Updates	500

After initial practice, the first two network convolutional layer parameters are transferred to initialize the global network's convolution layers 1 and 2. The main algorithm subsequently launches four workers initialized on the different Pong environments. Each worker trains on a mini-batch size of 64 ($e_w = 64$). The workers calculate AC loss, followed by the gradients for each network parameter. The gradients and batch samples are then added to the main algorithm queue. The gradients are used to optimize the global network using a gradient batch size of 2.

The batch samples from the worker are sent to the practice worker for training. Every $d_k = 500$ gradient update, the first two convolution layer gradients of the

practice network are sent to the global network for updation. Training concludes when each environment achieves maximum reward or threshold reward value of 15. This method is referred to as *Shared Practice* in the results.

Practice: Global Sampling Sampling using global policy head is referred to as global sampling in this experiment. Initial practice is carried out for about 1×10^5 frames, using a batch size of 64 ($e_{pr} = 64$) to train the practice network. PTAN ExperienceSourceFirstLast class is provided with the set of environments and a dull agent to generate samples without implementing any frameskip. The MSE loss is calculated between Δ and practice network output $\Delta_{predicted}$. The practice network parameters are updated using Adam optimizer using a learning rate of 0.00001 on the MSE loss.

Once initial practice is done, the first two network convolutional layer parameters are used to initialize the global A3C network's convolutional layer 1 and 2. Subsequently, the main algorithm launches four workers initialized on the different Pong environments specified by the environment training set. Each worker trains on a mini-batch size of 64 ($e_w = 64$), calculates AC loss, followed by gradients for their network parameters. The gradients are then added to the main algorithm queue that uses the gradients to optimize the global network using an update batch size of 2.

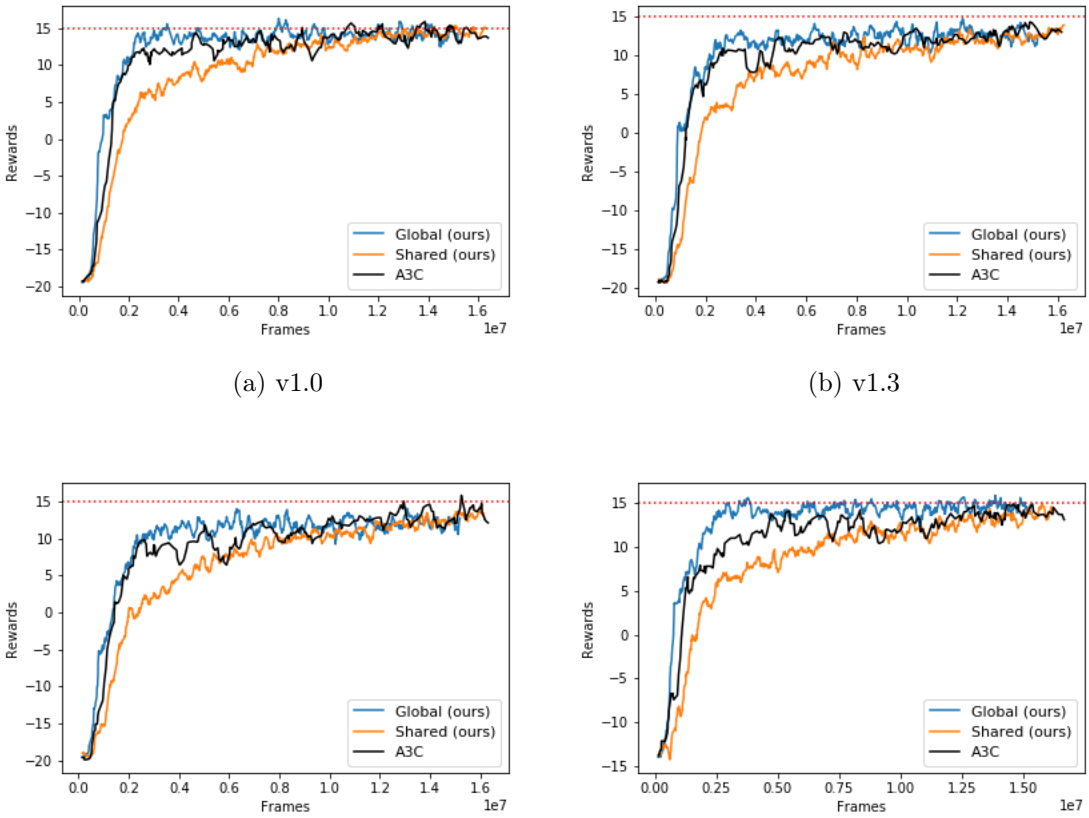
Every $d_k = 500$ gradient update, short practice is conducted. Short practice trains for about 1×10^4 frames using batch size 64 ($e_{pr} = 64$). The gradients of the first two convolution layer are sent to the global network optimizer for update. The workers train till maximum reward or threshold reward value of 15 is achieved by the environments. The best performing algorithm hyperparameters are listed in Table 4.2. This method is referred to as *Global Practice* in the results.

Table 4.3: Practice: Global Sampling Hyper-Parameters

Hyperparameters	Value
Worker Batch Size	64
Learning Rate	0.001
Optimizer	Adam
Gamma	0.99
Clip Grad	0.1
Train Batch	2
Practice Learning Rate	0.00001
Practice Loss	MSE
Practice Optimizer	Adam
Practice Batch Size	64
Initial Practice	100000 frames
Iterative Practice Updates	500

4.2.1 Results

The results in Figure 4.2 reflect the performance of Practice for Multi-Task A3C in the easy environment set. The figures show the result curves for the algorithms in the four environments during training. The experiments run for an average of 20 million frames. On the easy set, Global Practice matches or outperforms baseline multi-task A3C. Shared Practice initially lags on learning. The easy set results have a sharp learning curve, indicating that the agents can successfully optimize the multi-task objective. On average, V1.3 (smaller bat size) does not meet the threshold value during training. The harder set results provide more evidence on generalization capability and similar environment behaviour.



(a) v1.0

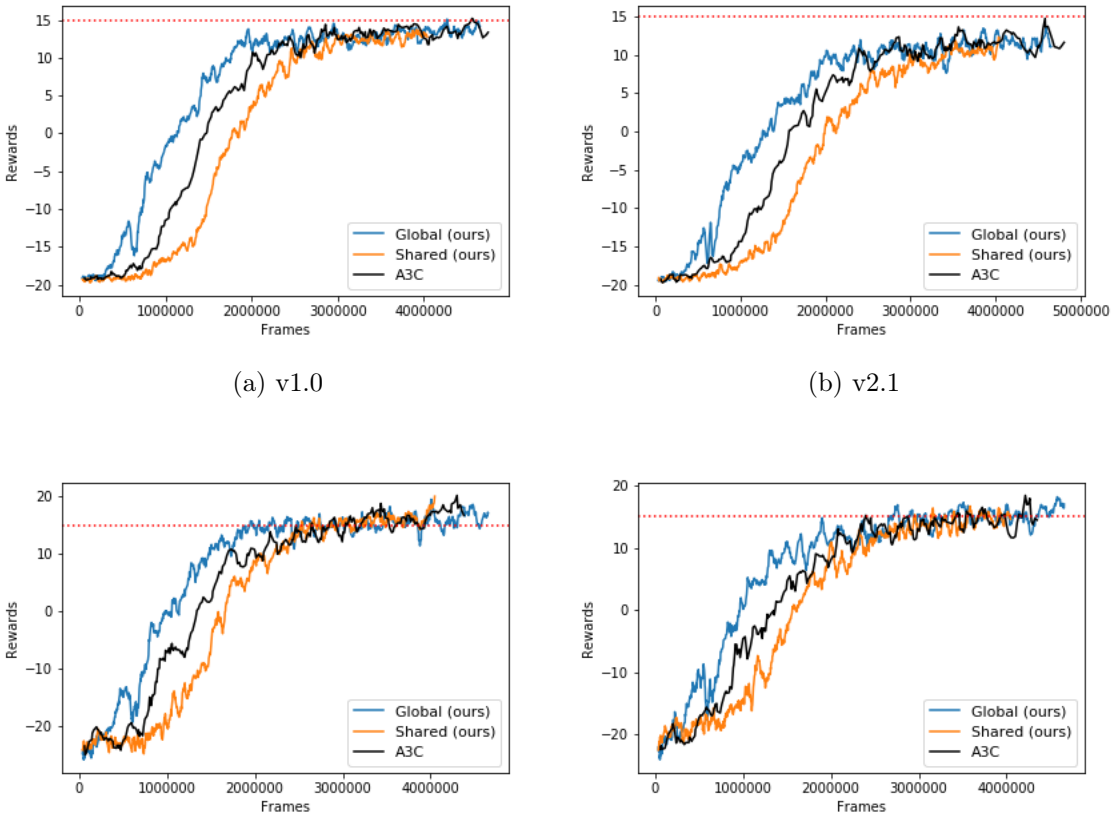
(b) v1.3

(c) v2.1

(d) v2.2

Figure 4.2: Average of reward curves of Global Practice, Shared Practice and Multi-Task A3C on the Easy Set during training. These results present the mean performance over 10 experimental runs. The dotted red line indicates the threshold value of 15.

The results in Figure 4.3 reflect the performance of Practice for Multi-Task A3C on the hard Pong environment set. The experiments run for an average of 40 million frames. Global Practice outperforms baseline A3C on the hard set and has more impact on multi-task learning on this set. This could indicate that the global network samples provide practice with the most generalized view of the domain. Similar to easy set performance, Shared Practice initially lags on training but catches up. The possible reasons are discussed in Chapter 5.



(a) v1.0

(b) v2.1

(c) v3.1

(d) v3.2

Figure 4.3: Average of reward curves of Global Practice, Shared Practice and Multi-Task A3C on the Hard Set during training. These results present the mean performance over 10 experimental runs. The dotted red line indicates the threshold value of 15.

It is interesting to see how the v2.1 results differ in the easy set and hard set. With the addition of a similar task v2.2 (Figure 4.2d), v2.1 is able to achieve the minimum threshold score. This finding corroborated by v3.1 and v3.2 present in the hard set. Having similar underlying dynamics tasks provides a boost in performance. These results are further studied from a gradient contribution perspective.

4.3 Multi-Task Gradient Contribution Analysis

A3C launches multiple workers in different environments and send gradients to the global network for update. The gradients contain information on how the network

should be updated to optimize the objective. In multi-task learning, each task has a different objective. The gradients that arrive in the main algorithm's queue are studied in Global Practice's training on the hard environment set. The purpose is to observe the relationship between different task gradient updates and their contribution to the main network's learning. CKA [2] is used to measure similarity.

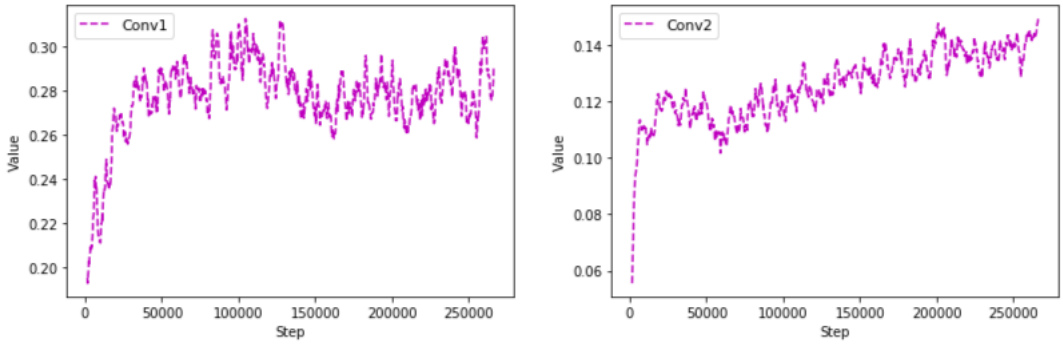
Practice vs Worker Practice's convolutional layer 1 and 2 gradients are compared to each A3C worker's. These two layers are selected by virtue of containing the most general features. Due to the recurrent nature of practice, the worker gradients are compared to the latest practice gradient version. This experiment provides insight into practice's optimization direction with respect to the other workers.

From Figures 4.4a, 4.4b, 4.4c and 4.4d, it can be observed that practice vs worker gradients similarity is consistent across all the task environments. In the first convolutional layer, CKA value starts at around 0.2 and sticks close to a score of 0.3 for the rest of training. A score of 0.3 indicates low levels of similarity. Convolutional layer 2 has a lower CKA score- hovering around 0.125. The practice-worker plots indicate that practice is not adding the same experience. There is a slight similarity to the gradients consistent throughout the training process, which adds to the learning process without harming multi-task performance.

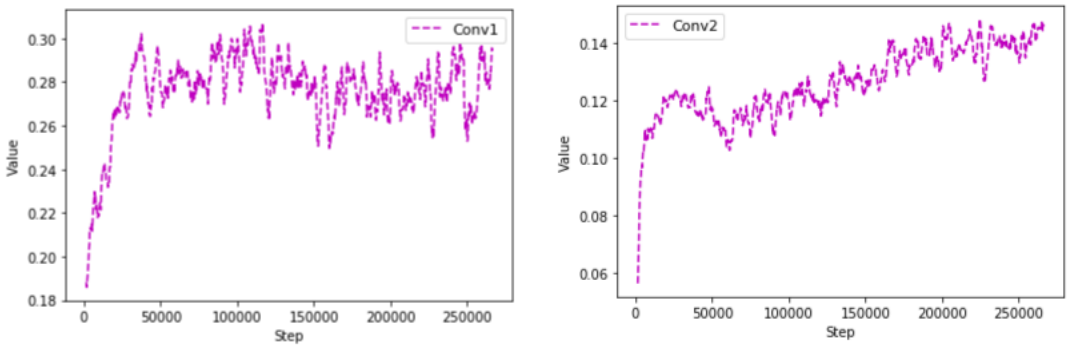
Worker vs Worker The worker gradients that arrive at the main algorithm queue are compared to each other. Figures 4.5a, 4.5b, 4.5c, 4.5d, 4.5e, and 4.5f, report CKA scores between the worker gradients. The relationship between the multiple tasks in a distributed learning setup requires to have a degree of similarity to learn successfully. The CKA results have the same pattern for all worker relationships.

On investigating the CKA values between workers, the similarity score is much higher than the practice-worker CKA score, which ranges between 0.2-0.3. For convolutional layer 1, the CKA score hovers between 0.7 and 0.5 for most of the training.

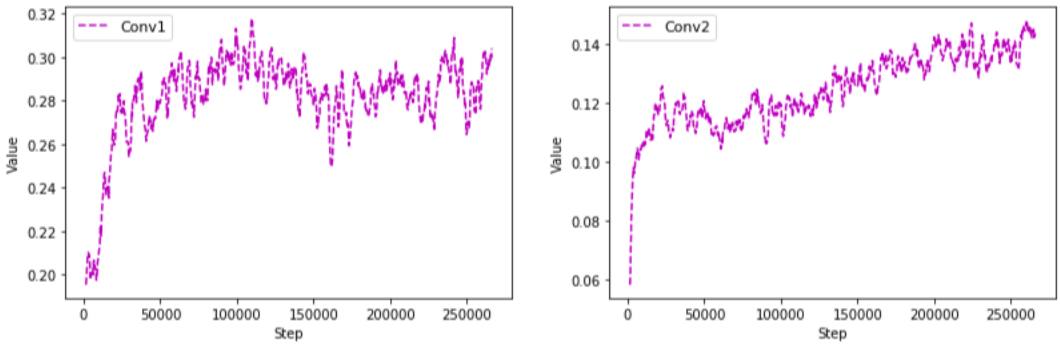
[2]Referenced from `https://github.com/google-research/google-research/tree/master/representation_similarity` (Accessed: 17 February 2021)



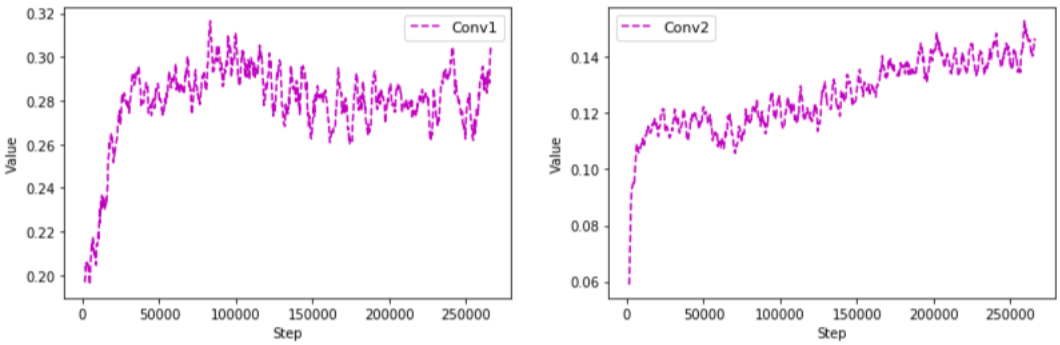
(a) Practice vs Worker v1.0



(b) Practice vs Worker v2.1

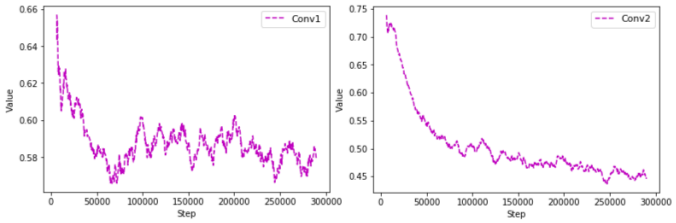


(c) Practice vs Worker v3.1

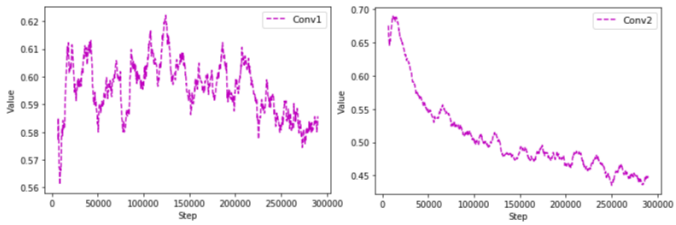


(d) Practice vs Worker v3.2

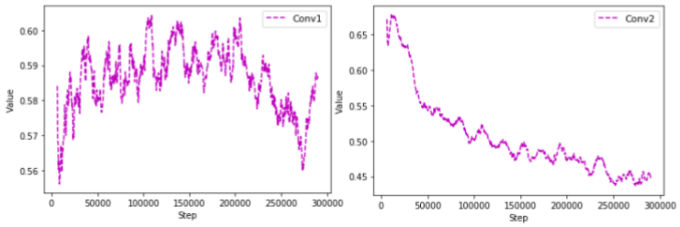
Figure 4.4: Results: Practice vs Worker Gradient Similarity



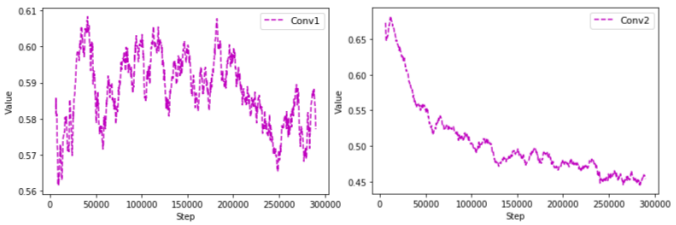
(a) Worker v1.0 vs Worker v2.1



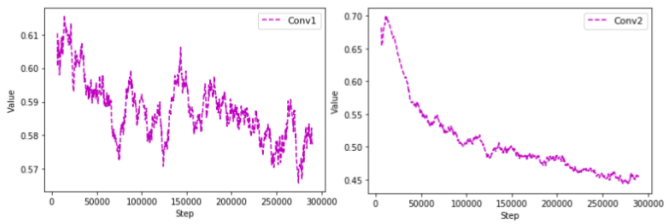
(b) Worker v1.0 vs Worker v3.1



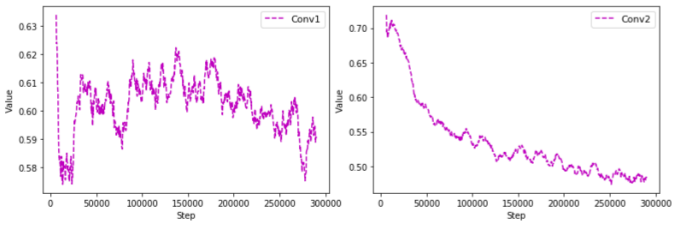
(c) Worker v1.0 vs Worker v3.2



(d) Worker v2.1 vs Worker v3.1



(e) Worker v2.1 vs Worker v3.2



(f) Worker v3.1 vs Worker v3.2

Figure 4.5: Results: Worker vs Worker Gradient Similarity

Convolutional layer 2 CKA scores have a sharper decrease in value. The CKA scores of this layer point to their individual gradient updates being more task-specific and losing generalizability. These values start from around 0.7 and drop to around 0.4. Although these values are much higher than practice's CKA score for this layer, similarity scores between the second layer reduce as training progresses. This is indicative of the upper layers focusing on task specificity. These results coincide with the findings of [48] that state that lower levels of neural networks are capture general features. It could be considered the same for gradients.

Env v3.1 and v3.2 have comparable environment dynamics. Their CKA scores do not indicate their similarity, which leads to the assumption that their individual task optimization does not focus on the same problem during training. Although the results indicate that having a similar task improves learning, the gradient CKA scores do not indicate high task similarity. The CKA values demonstrate that the worker objectives move in different directions but have an underlying similarity that affects the overall multi-task learning objective. This prompts the question about the slight orthogonal nature of practice gradients. Gradient updates using practice objective push general state dynamics information to the worker networks. As long as practice is not entirely orthogonal to the multi-task objective, it matches or succeeded multi-task A3C. This approach could be further tested on other multi-task and cross-domain environment experiments.

CHAPTER 5: DISCUSSIONS AND FUTURE WORK

5.1 What Determines Task Similarity?

An RL algorithm's understanding of similarity is difficult to gauge. Using the gradient contribution analysis, it can be inferred that the practice gradients do not harm the overall multi-task learning objective and provide representation that shares some similarities with the multiple objectives. When and where gradients are unfavourable to multi-task learning is not well understood, the human understanding of similarity and algorithmic understanding of similarity is perpetuated differently.

In [42], cosine similarity is calculated between auxiliary and main task gradients to control auxiliary task contribution to main task learning. A threshold value is used to filter auxiliary gradients dissimilar to main task gradients. When this measure was adapted to multi-task A3C worker gradient similarity, it resulted in values close to 0 for most of the training, suggesting that the tasks were orthogonal. However, the success of multi-task learning in reaching the threshold value for the environments implies differently.

The CKA results suggest a degree of similarity between tasks. The efficacy of this approach could be studied under a few different circumstances. By removing the asynchronous nature of the algorithm, the current state of the workers could be reported over time. The line of questioning followed by this thesis can be expanded to include larger task sets or different domains. This can provide an insight into the relationship between a large number of different tasks and the degree of orthogonality between domains with different environmental design. Understanding the relationship between tasks can provide insight into how they could be better adapted to solve the multi-task objective. Future work can determine the bounds to CKA's ability to

grasp multi-task learning objectives and algorithmic understanding of task similarity. By knowing its capabilities and limitations, it can be used as a hyperparameter to tune multi-task learning and potentially help regulate multi-task or transfer learning drift.

5.2 Practice and Generalization

The current experiment indicates that practice strategy helps improve multi-task generalization in distributed DRL. Practice gradients do not optimize on the same objective as these tasks, and with the help of CKA, it can be inferred that practice gradients share some similarity with the tasks, albeit on the lower end. This provides some grounds for the belief that practice develops common environmental dynamics knowledge.

Generalization in DRL has multiple strategies. This thesis approached generalization from multi-task and transfer learning perspectives. Approaches like L2 regularization, data augmentation, batch normalization and noise injection address generalization from the data diversity perspective [50][51]. These methods enhance training data by introducing variation and noise. They reduce the risk of overfitting and make networks robust to change. However, the nature of the noise added to the training data might introduce unwanted orthogonality resulting in poor performance. Unlike using random noise, practice representation contains state dynamics information that the CKA scores suggest is slightly similar to the RL gradients. It indicates that the information being added shares some common knowledge and is not completely orthogonal.

Future line of questioning could examine the limits of practice perception in distributed DRL. Practice perception depends on its training data. Global Practice's on-policy sampling does not follow a greedy or random approach. It leverages the current policy trained on all the tasks to generate samples. The representation learnt by practice using the global network can be further investigated by integrating practice

with the A2C algorithm, which is synchronous. This would ascertain if Shared Practice's performance can be attributed to representation lag. Implementing a replay buffer for practice in this setting can provide insight into the on-policy vs off-policy sampling.

The current version of practice generalization is very similar to meta-learning [34] which optimizes for representations that can quickly adapt to new tasks. It shares the same goal of learning internal features applicable to all tasks in task distribution $P(T)$. Calculating the state difference requires less computation than calculating the gradient step in multiple task directions. The representation learnt by practice helps in the generalization of perception. Adapting practice to visually complex domains could improve our understanding of the capabilities of practice perception.

Another parallel could be drawn by linking model-based learning and practice. Model-based RL introduces planning by explicitly defining the model of the environment to find optimal strategies. It creates a transitional model of the environment to simulated dynamics and select actions using predictive control. The model of the environment created by practice learns state dynamics information and supplements RL training. They serve a similar purpose of developing an understanding of the environment, but model-based RL methods weigh in on action selection, whereas practice transfers its knowledge about state dynamics.

This work uses practice as an auxiliary task. The current design of practice recurrently contributes to the multi-task learning objective. Continuous integration of practice gradients was tested but resulted in poor performance. In [42], cosine similarity was used to regulate auxiliary task contributions and improve RL performance. Future work could expand on this and explore the possibility of regulating practice contributions in the multi-task setting using CKA. This investigation could provide a stepping stone to adapt practice for multi-domain learning.

CHAPTER 6: CONCLUSIONS

Practice for Multi-Task A3C model is a promising avenue to explore generalization in distributed learning architectures. The goal of this thesis was not to beat state-of-the-art results but to provide evidence that supports practice's generalization capabilities. The two sampling strategies I have implemented lay emphasis on sample generation impact on practice representation.

To evaluate the efficacy of the approaches, I developed variants of the Pong environment as the initial test-bed to present the merits of Practice for Multi-Task Learning. Empirical results indicate that practice representation improves multi-task performance when trained on global network samples. By analyzing the gradient contributions, it can be inferred that practice provides representation that shares some similarities with the multiple objectives.

Future Work As discussed in Chapter 5, future directions of this work could examine the effectiveness of this approach in different multi-task and visually complex environments. Investigating the cause of local samples lag could be the next steps of this work. This could be examined by implementing synchronous advantage actor-critic algorithm and different sampling strategies like a shared experience buffer.

Additionally, it would be interesting to explore developing practice as a robust module that works as a bridging function between different domains by creating a state dynamics knowledge base from different environments. Practice would serve as a supplementary RL tool for adapting to new domains. Future work could extend this methodology's effectiveness to cross-domain learning.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb 2015.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct 2017.

[5] OpenAI, "Openai five." `https://blog.openai.com/openai-five/`, 2018.

[6] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Dec. 2009.

[7] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, pp. 41–75, Jul 1997.

[8] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," 2016.

[9] G. V. de la Cruz, Y. Du, and M. E. Taylor, "Pre-training with non-expert human demonstration for deep reinforcement learning," *CoRR*, vol. abs/1812.08904, 2018.

[10] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," 2016.

[11] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-task deep reinforcement learning with popart," *CoRR*, vol. abs/1809.04474, 2018.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[13] M. Lee and C. W. Anderson, "Can a reinforcement learning agent practice before it starts learning?," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 4006–4013, May 2017.

[14] V. S. S. R. Teja Kancharla and M. Lee, "Efficient practice for deep reinforcement learning," in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 77–84, Dec 2019.

[15] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 3519–3529, PMLR, 09–15 Jun 2019.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012.

[17] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[18] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[19] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR*, vol. abs/1609.03499, 2016.

[20] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[21] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, Dec 2020.

[22] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[23] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019.

[24] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous navigation of stratospheric balloons using reinforcement learning," *Nature*, vol. 588, pp. 77–82, Dec 2020.

[25] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," 2021.

[26] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, 2015.

[27] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, p. 541–551, Dec. 1989.

[28] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, *Object Recognition with Gradient-Based Learning*, pp. 319–345. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[29] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 2004.

[30] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1471–1530, Dec. 2004.

[31] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *CoRR*, vol. abs/1507.04296, 2015.

[32] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *CoRR*, vol. abs/1707.02286, 2017.

[33] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," 2018.

[34] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[35] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[36] A. Lazaric, *Transfer in Reinforcement Learning: A Framework and a Survey*, pp. 143–173. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[37] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for NLP," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799, PMLR, 09–15 Jun 2019.

[38] J. Howard and S. Ruder, "Fine-tuned language models for text classification," *CoRR*, vol. abs/1801.06146, 2018.

[39] M. E. Taylor and P. Stone, "Cross-domain transfer for reinforcement learning," in *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, (New York, NY, USA), p. 879–886, Association for Computing Machinery, 2007.

[40] G. de la Cruz, Y. Du, J. Irwin, and M. Taylor, "Initial progress in transfer for deep reinforcement learning algorithms," 07 2016.

[41] J. Roy and G. Konidaris, "Visual transfer for reinforcement learning via wasserstein domain confusion," 2020.

[42] Y. Du, W. M. Czarnecki, S. M. Jayakumar, M. Farajtabar, R. Pascanu, and B. Lakshminarayanan, "Adapting auxiliary losses using gradient similarity," 2020.

[43] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *CoRR*, vol. abs/1606.04671, 2016.

[44] C. W. Anderson, M. Lee, and D. L. Elliott, "Faster reinforcement learning after pretraining deep networks to predict state dynamics," *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2015.

[45] Y. Du and K. Narasimhan, "Task-agnostic dynamics priors for deep reinforcement learning," *CoRR*, vol. abs/1905.04819, 2019.

[46] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *CoRR*, vol. abs/1911.08265, 2019.

[47] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019.

[48] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *CoRR*, vol. abs/1411.1792, 2014.

[49] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, 2018.

[50] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *CoRR*, vol. abs/1812.02341, 2018.

[51] C. Zhao, O. Sigaud, F. Stulp, and T. M. Hospedales, "Investigating generalisation in continuous deep reinforcement learning," *CoRR*, vol. abs/1902.07015, 2019.