DISTILLATION AND GENERALIZATION IN DEEP REINFORCEMENT LEARNING

HAROLD PIJPELINK

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
AT THE SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
OF TILBURG UNIVERSITY

STUDENT NUMBER

2067939

COMMITTEE

prof. dr. Giacomo Spigler
prof. Dr. Dimitar Shterionov

LOCATION

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science &
Artificial Intelligence
Tilburg, The Netherlands

DATE

January 6, 2023

ACKNOWLEDGMENTS

I would like to thank my thesis supervisor, Dr. Giacomo Spigler for his continued support, feedback and leniency.

DISTILLATION AND GENERALIZATION IN DEEP REINFORCEMENT LEARNING

HAROLD PIJPELINK

Abstract

Deep Reinforcement Learning (DRL) is a methodology that aims to solve sequential decision making problems by training an agent that learns to solve this problem through interactions with an environment. Knowledge distillation is a method used to transfer information from a larger teacher network to a smaller student network. It has also been shown to have a regularizing effect on neural networks. A similar methodology called self-distillation distills models iteratively while maintaining the same number of network parameters.

The goal of this thesis was to examine whether knowledge distillation can improve the generalizability of DRL models. Generalizability is associated with the magnitude of the weights of the neural network, with smaller weights generalizing better. After performing knowledge distillation on the DRL model, the network weight magnitude lowered significantly, while generalization performance remained approximately the same. Distillation using self-distillation decreased the magnitude of the networks weights more; however, generalization performance decreased as well. Another comparison shows that l2-regularization affects the magnitude of the network weights stronger than knowledge distillation, but to such an extent that it can result in worse generalization performance.

1 INTRODUCTION

Deep Reinforcement Learning (DRL) is a form of artificial intelligence that trains an agent through interactions with an environment. One strength of DRL is that it is able to explore its environment, making it useful in situations where the goal is known, but programming a required behavior is complicated. In recent years, DRL has seen fast developments, going from playing Atari games in 2013 (Mnih et al., 2013a), to simulating real-life robotics (Doersch & Zisserman, 2019) in only a few years. However, DRL is still sample inefficient compared to supervised learning, with

agents requiring hundreds of thousands of environment interactions to learn something like a simple physics simulation (Brockman et al., 2016). Moreover, the training process can be unstable with high variance (Nikishin et al., 2018). As a result, there is a large body of literature dedicated to improving the efficiency of DRL.

1.1 *Motivation and Relevance*

The topic of this thesis concerns a fairly fundamental idea and, although the methods used in this thesis are limited to PPO, it may possible to extend the ideas presented here to other policy gradient algorithms.

As a result, the societal relevance of this thesis is strongly related to the relevance of DRL in general. One application of DRL is found in robotics. An example could be a robot that is required to perform successive actions, such as an assembly robot. The individual tasks may not be predefined well in this situation, so using a multi-task alternative can be more efficient. This action space is easily explored using DRL. Another example is social robotics: robot companions can use DRL through imitation learning, but that does involve several implicit behaviors to be learned by a single agent. More specific examples of DRL in practice include other robotics related topics such as (simulated) automotive control and self-driving cars (Wurman et al., 2022), industrial processes (Spielberg, Gopaluni, & Loewen, 2017), but also chip design (Mirhoseini et al., 2020), solving games and even novel ways to perform matrix multiplication efficiently (Fawzi et al., 2022).

1.2 *Knowledge Distillation*

One specific methodology used to improve DRL algorithms is knowledge distillation: this is an idea taken from the area of supervised learning. It involves training a small, student neural network while using knowledge from a larger pre-trained network to guide the training of the student (G. Hinton, Vinyals, & Dean, 2015). Model distillation has been shown to have a regularizing effect on DRL models, similar to more popular techniques such as l2-regularization (Cobbe, Klimov, Hesse, Kim, & Schulman, 2018). Usually knowledge distillation leads to smaller models, an exception being Furlanello, Lipton, Tschannen, Itti, and Anandkumar (2018), who introduce self-distillation, which is a form of iterative knowledge distillation where student networks are the same size as the teachers. Other knowledge distillation techniques related to DRL include distilling model priors into one multi-task model (Liu et al., 2021), multi-agent learning (Hong,

Nagarajan, & Maeda, 2020) and loss functions that encourage exploration by DRL agents, such as PPD (Spigler, 2023).

1.3 *Research Questions*

The aim of this thesis is to examine the use of knowledge distillation in DRL, in order to determine the effect of distillation on generalization and the magnitude of the model weights by answering the following 4 research questions:

RQ1 *To what extent does knowledge distillation improve test set performance for supervised learning and Deep Reinforcement Learning?*

According to literature it should have a regularization effect on both supervised learning and DRL, but it is not sure whether this effect is equally large in both. The general hypothesis for all models is that test performance will remain approximately the same, but that the magnitude of the weight vector will decrease.

RQ2 *How fast does knowledge distillation improve test set performance for DRL agents, and is this related to the weight magnitude of the neural network?*

At this point, it is not sure whether any gain in test set performance is due to distillation itself, or due to the fact that distillation comes with extra training steps.

RQ3 *Does additional self-distillation improve performance further for DRL agents?*

If distilling a model once lowers the magnitude of the model weights, then how often can this be repeated? The general expectation is that self-distillation will lower the weight magnitude of agents, with a possible slight drop in performance.

RQ4 *For DRL agents, how does the performance of knowledge distillation compare to l2-regularization?*

Knowledge distillation is supposed to work as regularization method, yet l2-regularization is a more popular regularization method. It may be that knowledge distillation performs worse overall than l2-regularization.

The next section is the literature review, which describes the backgrounds of DRL and knowledge distillation, followed by a discussion of different methodologies of applying knowledge distillation in DRL, and generalization. The methods section describes the four experiments that are used to answer the research questions, and how these are implemented. The results section presents the findings of these experiments, which are

evaluated and commented on in the discussion section, along with answering the research questions, the limitations of the thesis and future work. Lastly, the thesis is concluded by an overall summary that answers the research questions.

2 LITERATURE REVIEW

2.1 *Deep Reinforcement Learning*

This first section serves as a basis for the rest of the thesis, by describing the terminology and concepts used in DRL. At its core, reinforcement learning is a methodology that aims to solve sequential decision making problems. DRL involves an agent that learns to solve a pre-defined objective by interacting with an environment. A DRL problem is modeled as a Markov decision process (MDP). The key reason for modeling the DRL problem as an MDP is that the next state of an MDP depends only on its current state, and the action taken by the agent (Puterman, 1994). As a result, the agent is able to receive meaningful information about the relationship between its actions and the environment by using only the most recent state(s) of the MDP, rather than having to know the complete history of the environment. Mathematically the MDP is equal to the following tuple: $MDP = \langle S, A, T, R, \gamma \rangle$, with the variables as defined in Table 1.

Each DRL environment has a set number of states and can be low-dimensional or high-dimensional. Examples are a maze that contains discrete locations, or environments such as driving simulations that include a location, speed, acceleration etc. Depending on the environment, the number of states may be virtually infinite. By selecting an action, the agent moves on to a given state; the associated (state, action) pairs are known as transitions. In practice, DRL algorithms need to visit a state often before learning a behavior, so environments tend to be used episodically. They either end after reaching a terminal state, such as finding the exit to the maze, or after a predetermined number of actions. In order to incentivize the agent to perform actions that will solve the overall objective, the agent is given a reward for visiting a (state, action) pair, and the agent is instructed to maximize his total sum of rewards. In order to ensure that agents do not forfeit long-term reward in favor of short-term rewards, the reward is scaled by a discount factor γ.

A policy is a parameterized conditional probability distribution that maps states to actions, conditional on the present state of the environment. A DRL algorithm teaches the agent a policy that optimizes the total reward, which, in a well-defined environment, should correspond to a behavior that solves the objective. As for evaluating DRL agents, rather than having

Table 1: Definitions in Deep Reinforcement Learning

Symbol	Meaning
S	the set of states representing the environment that the agent interacts with
A	the set of actions that the agent can take
T	a transition function that describes the probability of transitioning from one state to another for a given action
r	a reward function that assigns a numerical reward to each state-action pair
γ	a discount factor that determines the importance of short-term rewards vs long-term rewards
π_θ	the parameterized policy

the trade-off between performance metrics that is common in supervised learning, the only performance metric of interest is the total discounted reward. Additionally, in DRL problems, there is often no natural test set, because DRL is used to solve a very particular task and creating an environment is time consuming. There are however, randomized test environments such as Procgen (Cobbe, Hesse, Hilton, & Schulman, 2019) that solve this problem, but the variety in between the environments in these libraries is limited.

In order to maximize the total discounted reward, the agent finds the action that maximizes the Bellman equation (Bellman, 1958). This equation gives a value for each state in the environment, based on the actions and rewards that the agent previously encountered when visiting this state.

Mathematically, the Bellman equation can be shown as follows, with T written out completely, for all $s \in S$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{1}$$

2.2 *DRL Algorithms*

One algorithm that can be used to solve a DRL problem is the tabular Q-learning algorithm. This algorithm works by learning a Q-function for each state in the environment, which corresponds to the future value of each state. The algorithm visits every state in the environment and iteratively updates the Q-function using the Bellman equation (Watkins & Dayan, 1992). As long as all (state, action) pairs are visited repeatedly, Q-learning is guaranteed to converge to a solution (Melo, 2001). However, as environments become arbitrarily large, it is no longer possible to visit every (state, action) pair, and the MDP can no longer be solved analytically.

One solution is to approximate the Q-function using a neural network, which is where the fields of reinforcement learning and deep learning meet. Policy gradient algorithms are a reliable and efficient class of DRL algorithms. They work by first defining an expected performance measure, $J(\Theta)$, which is generally related to discounted total reward. The policy gradient algorithm then takes the gradient of the policy with respect to the

parameters of the neural network. In order to maximize $J(\Theta)$, the policy is then updated using gradient ascent based on the size and direction of the gradient (Sutton, McAllester, Singh, & Mansour, 1999).

Policy gradient algorithms can generally be divided into two groups: on-policy and off-policy algorithms. The difference between the two is the origin of the transitions that are used to update the policy. On-policy algorithms update the policy using transitions generated by the current policy. On the other hand, off-policy algorithms can use any stored transitions to update future iterations of the policy. As a result, off-policy algorithms are more sample efficient, and require fewer environment interactions to achieve a high performance. The upside of on-policy algorithms is that they can be trained with parallel, asynchronous environments (Mnih et al., 2016). In practice, modern GPUs are able to simulate a high number of parallel environments, so on-policy algorithms are generally more time efficient.

2.3 *Knowledge distillation*

Knowledge distillation is a technique that aims to make neural networks smaller, while maintaining their performance. Neural networks consist of large numbers of weights that are learned during training; yet, the learned number of weights may not be optimal for model inference. By transferring knowledge from an existing model, knowledge distillation can create a new model with a lower number of weights (Buciluǎ, Caruana, & Niculescu-Mizil, 2006). Generally, this is done by training this student model to imitate the behavior of the teacher by optimizing using a loss function that takes in the softmax output layers of both the teacher and student models. The intuition behind using the softmax output rather than the predicted label is that it describes the full range of probabilities, rather than a single class. This information otherwise would not have been output by the teacher network (G. Hinton et al., 2015).

Smoothing the softmax layer over the labels can make it easier for the student model to learn from the teacher (Tang et al., 2020). For this purpose, a smoothing parameter called "temperature" is added to the softmax function. The loss function of the supervised learning problem then becomes twofold: on the one hand there is distillation loss between the 'soft targets' from the two softmax output layers, and on the other hand there is the student loss between the ground-truth label and the class predicted by the student model. G. Hinton et al. (2015) used cross-entropy loss for the student loss and KL-divergence for the distillation loss, but there are many other implementations of knowledge distillation that use different loss functions. The next section will describe the knowledge

distillation mathematically as shown in Kim, Oh, Kim, Cho, and Yun (2021).

The following equation represents the output of the softmax layer where z^f is the logit output of the layer that precedes the softmax layer, K is the number of classes and k and j are different classes. T is the temperature parameter.

$$\mathbf{p}_k^f(T) = \frac{\exp(z_k^f/T)}{\sum_{j=1}^{K}\exp(z_j^f/T)} \tag{2}$$

The cross-entropy loss L_{CE} is defined as follows:

$$L_{CE}(\mathbf{p^s}(1), \mathbf{y}) = \sum_{j=1}^{N} -\mathbf{y}_j \log(\mathbf{p}_j^s(1)) \tag{3}$$

where N is the number of samples, $\mathbf{y}_j$ is the ground-truth label for the j^{th} sample, and $\mathbf{p}_j^s$ is the predicted probability of the j_{th} class for the j_{th} sample using equation 2.

The knowledge distillation loss L_{KL} is defined as follows:

$$L_{KL}(\mathbf{p^s}(T), \mathbf{p^t}(T)) = T^2 \sum_{j=1}^{N} \mathbf{p_j^t}(T) \log(\frac{\mathbf{p_j^t}(T)}{\mathbf{p_j^s}(T)}) \tag{4}$$

Here $\mathbf{p}^s$ is the softmax output of the student network, and $\mathbf{p}^t$ is the softmax output of the teacher network.

The full loss function L can then be described as follows, with α as a hyperparameter that determines the balance between the two losses; this can be set using hyperparameter tuning.

$$L = \alpha L_{KL} + (1-\alpha)L_{CE} \tag{5}$$

2.4 *Knowledge distillation and Deep Reinforcement Learning*

There are a number of papers that apply knowledge distillation in the context of DRL. In Rusu et al. (2015), the authors introduce Policy Distillation. This is a distillation methodology that takes a pre-trained Deep Q-Network (DQN) as a teacher model (Mnih et al., 2013a) and trains a student network to predict its Q-values. This uses a replay buffer filled with saved trajectories and associated Q-values that was taken from the teacher policy, in order to use these as the target to optimize the student network. Since DQN is an off-policy algorithm, any previously gathered trajectories can be used to update the new policy. Note that saving the trajectories from more than one model makes it possible to distill several teacher models into a single student model.

The model is optimized using a loss function that takes the Q-value outputs from the student network and those from the replay buffer. The authors compare the performance of three different loss functions: negative log likelihood, Mean Square Error (MSE) and KL-divergence. After applying Policy Distillation to several Atari games, they find that the KL-divergence loss function performs best. Depending on the environment, it performs either slightly worse or slightly better than the teacher network. The sizes of the networks (in number of total parameters) decrease significantly, the largest network being 25% the size of the original, and the smallest one being only 4%.

Another paper that directly applies distillation in the context of DRL is Parisotto, Ba, and Salakhutdinov (2015), in which the authors introduce an algorithm called "Actor-Mimic". The aim of the authors is to train an agent that is able to perform well in a multi-task environment by distilling several pre-trained teacher models into a single student.

The loss function used is twofold. For the policy loss function, the authors train a DQN model for each task, and then take the softmax distribution over their output values. They note that, since this maps the output layer on the unit interval, the agent is encouraged to mimic the action of the teacher, rather than the exact Q-value. This lowers the variance, making it a stable signal for the student network across all different tasks. The second part of the loss function is a feature regression model that predicts the values of some of the hidden states of the student model, based on the sampled DRL transition. The Actor-Mimic student networks perform well on some Atari games, but on other games the performance is significantly worse than the teacher networks.

In a multi-agent setting, Hong et al. (2020) devised a methodology that uses distillation to aggregate the policies of multiple separate agents and distill them into new students. First, the agents go through a phase where they collectively gather environment transitions and store them in a shared buffer. Afterwards, this buffer is used to train all the agents. The agent with the highest total reward is selected to be used as the teacher model for a new group of student networks. The distillation loss function is the KL-divergence between the teacher and the student. Teh et al. (2017) use knowledge distillation to solve a multi-task DRL problem. The authors note that training one policy to solve multiple tasks is difficult, because during training the gradients of different policies can interfere with each other. The authors solve this by training individual task policies and then distilling them into a shared policy. The various task policies are then regularized using a KL-divergence based loss function in order to stay as close as possible to the shared policy, while still performing well on the individual tasks.

2.5 *Self-Distillation*

One particular kind of model distillation that has not yet been applied in the context of DRL are the so-called Born Again Neural Networks (BANN) described in Furlanello et al. (2018). The intuition behind this method is taken from Breiman and Shang (1996), who found that using an ensemble of decision trees, one tree can be trained to achieve similar performance to the ensemble while keeping the positive aspects of having a single tree. They refer to this model compression algorithm as *Born-Again Trees*.

Furlanello et al. (2018) extend this idea, but instead of compressing the teacher model into a smaller student, the size of the student network is kept the same as the teacher's. They train what they refer to as *Born-Again Neural Networks*, by performing knowledge distillation on a DenseNet model (Huang, Liu, van der Maaten, & Weinberger, 2016) trained on the Cifar-10 and Cifar-100 datasets (Krizhevsky, 2009). The loss function used is based on the cross-entropy between the teacher model and the student model. The authors find that the distilled models outperform their teachers on the test set data. Additionally, the authors show that this process can be performed iteratively, using student models as new teachers. Using a variety of different iterations as an ensemble classifier leads to an even better accuracy on the test set.

One thing that is not immediately clear after reviewing self-distillation is the intuition behind why it improves performance. After all, the student networks are created from an iterative process, so any mistakes present in the teacher models should also be present in its students. Mobahi, Farajtabar, and Bartlett (2020) show mathematically that self-distillation is not unique to DRL, but can be present in regression as well. By assuming a Hilbert space as the model space and applying l2-regularization in this space, the authors show that per iteration, self-distillation removes a number of basis vectors of the Hilbert space. This results in a regularization effect, not dissimilar to l2-regularization, but as more basis vectors are removed, it can also lead to underfitting. To the knowledge of the author, the use of self-distillation on a DRL model is novel, and this is part of the research niche that the thesis will explore.

2.6 *Similar Distillation Methods*

Besides self-distillation, there are a variety of distillation-based methodologies, with some of them having similar names for different processes. For clarity, this section will review a non-exhaustive list of such distillation methodologies.

The use of 'self-distillation' and 'reborn network' in literature appears to be ambiguous. Zhang et al. (2019) perform a variant of what they refer to as self-distillation on Convolutional Neural Networks. They use knowledge distillation with a triplet loss function based on intermediate hidden layers in a ResNet model (He, Zhang, Ren, & Sun, 2015). There is no form of iteration in this self-distillation, and it appears that this 'self-distillation' has no relation to the Born Again Neural Networks in Furlanello et al. (2018)

Agarwal, Schwarzer, Castro, Courville, and Bellemare (2022) introduce the reincarnating reinforcement learning algorithm. This is a methodology similar to distillation, which transfers knowledge from smaller DRL models to larger DRL models. The advantage of training a small model is that it is less resource intensive and therefore more accessible. On the other hand, it is known that using large, overparameterized networks in DRL increases the performance of the network, one example being OpenAI Five (OpenAI et al., 2019). The method shown by the authors is resource efficient, since the overparameterized network learns faster thanks to the knowledge transfer from the small network. The authors show that other distillation and transfer learning techniques such as Kickstarting (Schmitt et al., 2018) and JSRL (Uchendu et al., 2022) are inefficient when compared to the authors newly proposed reincarnating RL algorithm.

This algorithm is a mixture of Q-learning and behavioral cloning, and by first optimizing the behavioral cloning aspect, the student model learns a decent policy from its teacher. Then, the Q-learning part takes over, which allows the student to learn only from visiting transitions from a replay buffer. This slowly wanes the student off from any mistakes or suboptimal aspects of the teacher policy, while still using it to speed up learning the initial policy.

Other work that uses pre-trained models to learn a generalized behavior is Liu et al. (2021). This paper uses distillation to teach agents how to play football against each other. The authors do this by pre-training expert models for different sub-tasks, such as kicking the ball or dribbling. These expert models are then distilled into one multi-task agent using KL-divergence. It is notable that in the distillation from the priors to the final policy, the loss function used is the KL-divergence between the student and the teacher. This means that the trajectories are sampled from the student's policy. Rather than trying to emulate the teacher as closely as possible, the agent is incentivized to behave as close as possible to one of the behavioral priors, instead of the average over the priors.

2.7 *Generalization and Weights*

Generalization is defined as the ability of a trained model or agent to perform well on unseen data, and as such is also referred to as test set performance. Overall, the importance of the size of neural network with regards to generalization is somewhat unclear. In supervised learning, as the number of parameters in a neural network increases, the model is able to fit itself increasingly close to the training data. This can lead to worse generalization performance, also known as overfitting. As early as 1987, G. E. Hinton (1987) devised that adding a penalty to the loss function that is proportional to the sum of squares of the model weights will improve generalization. This method is known as weight decay; the intuition behind this idea is that forcing the network to use small weights "removes irrelevant components" and removes some effects of noise in the data (Krogh & Hertz, 1991). Later research suggests that it is not the number of weights that affects generalization, but the size of the weights (P. Bartlett, 1996; G. E. Hinton & Van Camp, 1993). This is a very intuitive idea. When fitting a curve using a polynomial function, the curve will become smoother as the size of the coefficients shrinks, which makes it less prone to overfitting. However, more recent overparameterized models have shown better generalization performance when compared to their smaller counterparts (P. L. Bartlett, Montanari, & Rakhlin, 2021).

One generalization issue that is particular to DRL is that the learning process is non-stationary. Igl, Farquhar, Luketina, Boehmer, and Whiteson (2020) note that, as an agent is updated during training, its policy changes, and this affects the distribution of actions chosen by the agent. As the distribution of actions changes, the distribution of the (state, action) pairs and their rewards changes as well. The authors find support for this hypothesis by manually adding non-stationarity to a supervised learning multi-class classifier and show that test set performance is significantly lower. The authors suggest a new algorithm called Iter, which uses knowledge distillation to learn a new policy and value function. They use a loss-function consisting of two factors that minimize the KL-divergence between the output of the student and the output of the teacher, and two factors that are off-policy DRL rewards, all of which are parameterized. Over time, these parameters are annealed and the algorithm switches from learning from the teacher model to learning from the DRL environment. Similar to the algorithm in Agarwal et al. (2022), the student network is weaned off the suboptimal teacher slowly, and the performance of the student should no longer be affected by the teachers non-stationary input data.

2.8 *PPD*

In the previous subsections of the literature review, a variety of distillation methods for DRL were discussed that included both on-policy and off-policy DRL algorithms. However, Czarnecki et al. (2019) show mathematically and empirically that distillation using cross-entropy between the student and the teacher policy while using the teacher policy to gather trajectories is not guaranteed to converge. This is because this combination does not form a gradient vector field. One method of overcoming this problem suggested by the Czarnecki et al. (2019) is to use the student policy to gather the transitions. In their Appendix C, they show that using teacher-student distillation (with transitions taken using the teacher policy) aims to replicate the teacher policy, while using student-teacher distillation (with transitions taken from the student policy) aims to find the most probable action in the teacher policy.

This does not only hold for the cross-entropy that is discussed in Czarnecki et al. (2019), but also when the KL-divergence is used in the loss function. This is the result of KL-divergence being non-symmetrical. When KL-divergence is used to optimize a parameterized function X_θ to fit a given distribution Y, $D_{KL}(Y||X_\theta)$ is known as the forward KL-divergence. The inverse, $D_{KL}(X_\theta||Y)$, is known as the reverse KL-divergence. These two KL-divergences behave differently from each other in how they fit the Y distribution. Mathematically this can be explained as follows[1]:

The equation for KL-divergence between distribution Y and X_θ:

$$D_{KL}KL(Y||X_\theta) = \sum_i Y(i) \log \frac{Y(i)}{X_\theta(i)} \tag{6}$$

In order to for the distributions to match each other, the KL-divergence is minimized w.r.t the parameters θ. For the forward KL-divergence, this leads to the following equation:

$$\underset{\theta}{argmin} KL(Y||X_\theta) = \underset{\theta}{argmax} E_Y(log X_\theta(x)) \tag{7}$$

This single term states that X should have a high probability where Y has a high probability. This corresponds to the mean of the distribution Y. In contrast, the reverse KL-divergence minimizes a slightly different equation:

$$\underset{\theta}{argmin} KL(X_\theta||Y) = \underset{\theta}{argmax} E_{X\theta}(log Y(x)) + E_{X\theta}(-log X_\theta(x))) \tag{8}$$

[1] This explanation was taken from the following blog post, as it was the most straightforward source: https://towardsdatascience.com/forward-and-reverse-kl-divergence-906625f1df06

This equation essentially states that Y always has to be close to X, which means that it now aims for the mode of the distribution, rather than the mean. This is similar to what was explained in Czarnecki et al. (2019) regarding the replication of the policy vs. choosing the most likely action of the policy.

This idea is applied in a new kind of DRL distillation methodology called Proximal Policy Distillation (PPD), described in yet-to be published work (Spigler, 2023). This is an alteration to PPO that samples trajectories from the student policy distribution using importance sampling, and adds a student-to-teacher KL-divergence term to the loss function. As a result, the value of the KL-divergence penalty depends on the action space, not on the parameter space. This means that the student policy is incentivized by the KL-divergence penalty to choose actions that are similar to the teacher, but since the trajectories are sampled from the student policy, it is able to explore new states that were not visited by the teacher. As a result, it is able to learn the same behavior as the teacher, but using a different set of parameters. As a result it should be able to learn the policy using smaller weights in the neural network which in turn may generalize better. The KL-term is scaled by a hyperparameter that can be annealed over time, allowing the student model to learn only from its DRL rewards and no longer from the teacher model, similar to the Reincarnating RL algorithm in Agarwal et al. (2022).

3 METHODOLOGY

In order to answer the four research questions in this thesis, four separate experiments were designed. This section will discuss the experiments and their implementations one by one.

3.1 *Experiment 1: Supervised Learning*

The aim of the first experiment is to determine whether knowledge distillation can improve generalization performance in supervised learning and in DRL. As such, the first experiment consists of the distillation of two supervised learning models, and the distillation of three DRL agents. The performance metric will be accuracy for the supervised learning models and total reward per episode for the DRL agents. All models will also be compared by weight magnitude W. This is calculated as follows, with θ representing the trainable parameters of the neural networks.

$$W = \sqrt{\sum_{i=1}^{n} \theta^2} \tag{9}$$

The two multi-class supervised learning classifiers were trained on the Mnist dataset (Deng, 2012) and the Cifar-10 dataset (Krizhevsky, 2009). Both datasets consist of images and associated classification labels. The Mnist images are hand-written numbers and have a size of 28x28 pixels, while the Cifar-10 images of everyday, recognizable items are 32x32 pixels. Both datasets contain 10 unique classes. The Mnist model was trained using a 2 layer Multilayer Perceptron model, similar to El Kessab, Daoui, Bouikhalene, Fakir, and Moro (2013). The model has an input layer of 784 units and 10 units in the hidden layer. Distillation was implemented using the 'Distillation' Python package[2]. The Multilayer Perceptron model was trained for 5 epochs and distilled for 5 epochs afterwards. For the test set, the pre-made Mnist test set was used.

Empirically, Cifar-10 is harder to train than Mnist, so a Resnet-18 model (He et al., 2015) was implemented. ResNet is a neural network consisting of connected convolutional layers and skip-connections. It has performed well on Cifar-10, achieving accuracies of up to 95% after data augmentation. Data augmentation was performed on the training images, in the form of rescaling the images to a 224x224 size to match the data that ResNet was originally trained on, and image normalization. The Cifar-10 dataset comes with a predetermined split between the train and test set, with 5000 training images and 1000 test set images, and these were used as such.

The Resnet-18 model with cross-entropy loss was implemented using Pytorch, along with its distillation method[3]. The distillation method adds an extra parameter to the overall loss function. This parameter is the KL-divergence between the softmax layers of the teacher and student network. The model was distilled for 20 epochs and no hyperparameter tuning was performed over the entire process. The learning rate started at 0.1 and was annealed as described in He et al. (2015).

3.2 *Experiment 1: Deep Reinforcement Learning*

For the choice of DRL environment, it is important to note that the student model may outperform the teacher. This means that the environment must not be fully solved by the teacher, so that there is still room for improvement after distillation. Moreover, evaluating the performance of the agent requires a test set of unseen environments. Both of these requirements are met by Procgen (Cobbe et al., 2019). This Python library generates randomized environments reminiscent of sidescroller video game which have well-defined rewards and are therefore solvable by DRL agents.

[2] Found here: https://github.com/Kennethborup/knowledgeDistillation

[3] The code for this model was adapted after this example: https://het-shah.github.io/blog/2020/Knowledge-Distillation/

Since Procgen generates random levels, it is able to generate an unseen test set to evaluate generalization performance.

Procgen provides both easy and hard levels. While most literature uses the hard levels as a benchmark, training agents that achieve a similar performance to the original paper on the hard environments takes approximately 250 million timesteps, which corresponds to approximately 24 hours of clock time using the available A40 GPUs. The easy levels on the other hand take approximately 30 million timeseps, so 3 hours to train. In the interest of time and given limited resources, all agents were trained and evaluated on the easy levels.

The distilled models are expected to have a regularizing effect. This could be particularly effective on environments with a large gap between training and test set performance. Based on the performance as shown in appendix I of Cobbe et al. (2019), three environments with a large difference between train and test-performance are Jumper, Miner and Ninja, so these environments were used to train the DRL models.

One advantage of using the A40 GPUs is that they allow for training with multiple parallel environments, so 128 environments were trained in parallel. In order to increase the external validity of the results, each environment was trained using four different random seeds: 201, 401, 601 and 801. This randomization was performed using the built-in random seed functionality in Procgen. The agents were then evaluated on a test set of unseen levels. Each agent was evaluated on the same test set. In the results section, the mean reward per episode will be reported for both the training set as well as the test set.

In a practical scenario, it may not be efficient to distill a model, since the time spent distilling the model can also be used to continue training. The most apt model comparison is therefore not to compare the distilled agent against its teacher, but to compare the distilled agent against the teacher model that was trained for an additional number of environment steps so that the total number of environment steps is equal for both models.

The original models were trained for 25, 30, 35 and 40 million environment steps. According to Cobbe et al. (2019), this number of training steps should lead to models that perform well enough to be a viable teacher model. With 4 different random seeds, this makes for a total of 16 models for each Procgen environment. The results section reports the mean reward and standard deviation taken over the 4 random seeds. The complete table of all original models is available in Appendix A (page 31), the complete results of all distilled models are available in Appendix C (page 33). The distilled models used the 25, 30 and 35 million environment step models as teachers, and were distilled for an additional 5 million environment steps.

The algorithm used to solve the DRL environments is Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). PPO is a so-called actor-critic model, which means that it uses a neural network with two output heads to learn the policy function and the value function. The advantage of PPO is that it uses a trust region to determine a maximum step size of the policy gradient, similar to TRPO (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015). As a result, at each timestep, the policy will be updated in the direction of the gradient, but up to a given upper bound, which limits the variance of the training process. This in turn leads to a more stable training performance when compared to other actor-critic models. An alternative would have been to use Double Q-Networks (DQN) (Mnih et al., 2013b), but this algorithm, like any Q-learning variant, requires precise hyperparameter tuning. The other advantage of using an on-policy algorithm like PPO is that it can be trained using a large number of parallel environments. This makes it well suited to be trained on the available GPU clusters, speeding up overall training time. For all models trained in this thesis, the PPO hyperparameters were kept at their default values.

The PPO implementation was taken from Stable-Baselines 3 (Raffin et al., 2021), which is compatible with Procgen. Stable-Baselines 3 uses the PPO-clip implementation. Instead of having a specific term for KL-divergence, the absolute value of the loss function is max-clipped between certain intervals to create the trust region effect. This clipped loss function is then optimized using the Adam optimizer (Kingma & Ba, 2014).

As for the feature extraction, the IMPALA architecture was used (Espeholt et al., 2018). This is similar to ResNet (He et al., 2015), consisting of a convolutional layer with a 3x3 kernel and a max-pooling layer with a 3x3 kernel. This is followed by two residual blocks, which each consist of two convolutional layers with 3x3 kernels and ReLu activation functions. These are then combined with the original input through a skip-connection. The final layer of the model is a fully connected layer with 256 units and a ReLu activation function. The feature extractor was implemented in Pytorch using the implementation available on the Impala Github page.

Model distillation was performed using the PPD implementation taken from Spigler (2023). This adapts the Stable-Baselines 3 PPO-clip implementation by adding a loss term based on the KL-divergence between the student and teacher policies. The KL loss term is then clipped using the same hyperparameter as PPO-clip, which ensures that it is at a similar order of magnitude as the other terms in the PPO loss function. The clipped KL-loss term is then added to the regular PPO loss. PPD also includes a lambda hyperparameter which determines the extent to which the model learns from the teacher rather than its DRL rewards. For all experiments

performed in this thesis, this was kept at its default value of 1, ignoring the DRL rewards.

3.3 *Experiment 2*

The aim of the second experiment is to determine whether a newly distilled model sees a quick spike in performance from distillation or a gradual increase because it undergoes more training steps. This is done by repeating the distillation part of experiment 1, but instead of distilling the models for 5 million timesteps, they are distilled for 3 million timesteps. These models are then compared to the models that were distilled for 5 million timesteps in mean reward per episode and weight magnitude. In all other aspects, experiment 2 is identical to the DRL part of experiment 1.

3.4 *Experiment 3*

Experiment 3 tests the effect of self-distillation on test set performance and weight magnitude of the network. This is done by taking the 25 million and 30 million Jumper and Miner models that were distilled for 5 million timesteps in experiment 1, and using them as teacher models to distill them for another 5 million timesteps. This will effectively give them 10 million extra timesteps of training, so their performance will be compared against the original 35 million and 40 million Jumper and Miner models. To determine the effect of self-distillation on the magnitude of the network weights, the self-distilled models will be compared against their teacher models. The distillation methodology was identical to the one for experiment 1.

3.5 *Experiment 4*

The fourth experiment is a comparison between self-distillation and l2-regularization (Cobbe et al., 2018). A Jumper agent and a Miner agent were trained for 25, 30, 35 and 40 million timesteps, but with l2-regularization applied. The l2 regularization hyperparameter was set at 0.001. This was determined by performing hyperparameter tuning over a validation set of 1000 episodes. The results of the hyperparameter tuning can be found in Appendix B (page 32). The outcomes of these models will be compared against the findings from experiments 1 and 3.

Table 2: Comparison between Distilled and Non-distilled supervised learning models for a Multilayer Perceptron on trained on Mnist and a Resnet-18 model trained on Cifar-10.

Task		Train + Accuracy	Test Accuracy	Weight Magnitude
Mnist	No Distillation	90.0	89.0	9.33
	With Distillation	93.5	90.00	6.99
Cifar-10	No Distillation	73.05	79.37	85.23
	With Distillation	67.15	64.81	79.92

4 RESULTS

4.1 *Experiment 1*

For the Mnist model, train and test accuracy increase somewhat, while the weight magnitude decreases by approximately 20%. For the Cifar-10 model, the magnitude of the models weights decreases slightly, but both train accuracy and test accuracy are significantly lower for the distilled model than they are for the original model.

As mentioned in the methods section, the DRL part of experiment 1 centers around three different Procgen environments that were each trained using four different random seeds. Table 3 shows the mean and standard deviation of the mean episode reward across the four random seeds. The full table in can be found in Appendix A (page 31).

For the Jumper models, the training performance of all distilled models is higher than that of the original models. The test performance is approximately the same for all six models, regardless of the number of training steps or distillation steps. For all three distilled Jumper models, the magnitude of the weight vector is less than half of that of the original model. Most of the standard deviations for the train and test set are fairly small, except for the 25+5 model, where they are slightly higher. The standard deviations of the weight magnitudes are all negligible.

For the Miner models, the training set performance was higher for all distilled models when compared to the originals. Contrary to the Jumper models, the distilled Miner models perform worse on the test set than the original models. The magnitude of the model weights is similar to the weights of the Jumper models, for both the original models as well as the distilled models. The standard deviations are higher than the Jumper standard deviations, but this scales approximately with the higher rewards that the Miner agents achieved.

Table 3: Comparison between Distilled and Non-distilled DRL models with an equal number of total training steps. Scores are the means across the four random seeds, the values in parentheses represent the standard deviations.

Environment	Training steps + distillation steps	Train Accuracy	Test Accuracy	Weight Magnitude
Jumper	25+5	7.12 (0.60)	5.52 (0.46)	3.78 (0.10)
	30	6.18 (0.25)	5.73 (0.05)	8.39 (0.12)
Jumper	30+5	6.85 (0.24)	5.65 (0.30	3.72 (0.15)
	35	6.00 (0.17)	5.57 (0.24)	9.06 (0.17)
Jumper	35+5	6.62 (0.22)	5.35 (0.37)	3.72 (0.20)
	40	6.07 (0.21)	5.66 (0.12)	9.64 (0.15)
Miner	25+5	10.50 (0.59)	8.51 (0.30)	3.79 (0.14)
	30	8.94 (0.46)	9.53 (0.41)	8.10 (0.13)
Miner	30+5	10.76 (0.76)	9.18 (0.87)	3.73 (0.12)
	35	9.10 (0.58)	9.52 (0.31)	8.80 (0.15)
Miner	35+5	10.13 (0.72)	8.95 (0.44)	3.87 (0.17)
	40	8.89 (0.24)	9.56 (0.27)	9.45 (0.18)
Ninja	25+5	5.80 (0.57)	5.38 (0.61)	3.41 (0.19)
	30	5.79 (0.35)	5.88 (0.30)	8.78 (0.13)
Ninja	30+5	6.02 (1.18)	5.25 (0.55)	3.33 (0.22)
	35	5.89 (0.40)	6.04 (0.49)	9.44 (0.10)
Ninja	35+5	6.50 (0.78)	5.75 (0.39)	3.54 (0.17)
	40	6.01 (0.37)	6.05 (0.38)	10.01 (0.10)

Table 4: Comparison Between DRL environments that were distilled for either 3 million or 5 million training steps. Values given are the means across the four random seeds, the values in parentheses represent the standard deviations

Environment	Training steps + distillation steps	Train Accuracy	Test Accuracy	Weight Magnitude
Jumper	25+3	7.00 (0.28)	5.25 (0.65)	3.20 (0.17)
	25+5	7.12 (0.60)	5.52 (0.46)	3.78 (0.20)
Jumper	30+3	6.80 (0.61)	5.92 (0.57)	3.19 (0.16)
	30+5	6.85 (0.24)	5.65 (0.30)	3.72 (0.15)
Jumper	35+3	6.89 (0.24)	5.50 (0.42)	3.29 (0.27)
	35+5	6.62 (0.22)	5.35 (0.37)	3.72 (0.20)
Miner	25+3	9.97 (0.83)	8.64 (0.44)	3.26 (0.13)
	25+5	10.50 (0.59)	8.51 (0.30)	3.79 (0.14)
Miner	30+3	10.31 (0.51)	9.04 (0.51)	3.37 (0.19)
	30+5	10.76 (0.76)	9.18 (0.87)	3.73 (0.12)
Miner	35+3	9.88 (0.42)	8.21 (0.25)	3.47 (0.22)
	35+5	10.13 (0.72)	8.95 (0.44)	3.87 (0.17)
Ninja	25+3	5.90 (1.19)	5.10 (0.56)	3.02 (0.13)
	25+5	5.80 (0.57)	5.38 (0.61)	3.41 (0.19)
Ninja	30+3	5.90 (0.45)	5.52 (0.56)	3.03 (0.21)
	30+5	6.02 (1.18)	5.25 (0.55)	3.33 (0.22)
Ninja	35+3	6.35 (0.17)	5.62 (0.75)	3.09 (0.14)
	35+5	6.50 (0.78)	5.75 (0.39)	3.54 (0.17)

For the Ninja models, the training set performance is approximately equal, except for the 35+5 model, which does outperform the original model trained for 40 million timesteps. The test set performance of the distilled models is slightly worse than that of the original models. The weight vector shows similar results as the Jumper and Miner models.

4.2 *Experiment 2*

The aim of experiment 2 was to determine whether the increase in generalization performance was caused by distillation or by the extra environment steps. The results of the experiment are visible in Table 4. For the Jumper models, the training set performance of the models distilled for 3 million environment steps and the models distilled for 5 million environment steps is approximately equal. The training set performance is approximately the same across the 25, 30 and 35 million environment steps models. The same is true for the test set performance. Notably, in two out of three cases, the model with fewer distillation steps performs slightly better on the test set.

Table 5: Comparison between DRL environments that were either distilled for 5 million training steps, or distilled for 5 million training steps and then self-distilled for an additional 5 million training. Values given are the means across the four random seeds, the values in parentheses represent the standard deviations

Environment	Training steps + distillation steps	Train Accuracy	Test Accuracy	Weight Magnitude
Jumper	25+5	7.12 (0.60)	5.52 (0.46)	3.78 (0.20)
	25+5+5	6.67 (0.15)	5.85(0.41)	3.54 (0.38)
	35	6.00 (0.17)	5.57 (0.24)	9.06 (0.17)
Jumper	30+5	6.85 (0.24)	5.65 (0.30	3.72 (0.15)
	30+5+5	7.05 (0.39)	5.77 (0.73)	3.66 (0.09)
	40	6.07 (0.21)	5.66 (0.12)	9.64 (0.15)
Miner	25+5	10.50 (0.59)	8.51 (0.30)	3.79 (0.14)
	25+5+5	10.42 (0.58)	8.86 (0.71)	3.51 (0.15)
	35	9.10 (0.58)	9.52 (0.31)	8.80 (0.15)
Miner	30+5	10.76 (0.76)	9.18 (0.87)	3.73 (0.12)
	30+5+5	10.67 (0.51)	9.05 (0.94)	3.58 (0.19)
	40	8.89 (0.24)	9.56 (0.27)	9.45 (0.18)

The training set performances of all Miner models are approximately equal regardless of the number of training steps or distillation steps. The models that were distilled longer perform better on the test set, but this difference is quite small. Overall, this looks very similar to the Jumper results.

For the Ninja environment, the 25+3 model and the 30+5 model have a higher standard deviation for the training set performance than is the case with the Miner and Jumper models, but this is not the case for the standard deviation of the test set performance. Both the training set and test set performances are slightly better for the models with 35 million training steps, regardless of the number of distillation steps, but this difference is quite small.

For all models in the experiment, the magnitude of the weight vectors is higher for the models were distilled for more environment steps.

4.3 *Experiment 3*

The aim of experiment 3 was to determine the effect of self-distillation on test set performance and the magnitude of the model weights. The results of the experiment are visible in Table 5.

For the Jumper models, the train performance of the distilled models is higher than that of the original models. The difference in test set

Table 6: Comparison between DRL environments that were trained with l2-regularization.

Environment	Training Steps	Train Accuracy	Test Accuracy	Weight Magnitude
Jumper - l2	30 mil	5.98	4.90	0.87
	35 mil	5.97	5.47	0.86
	40 mil	6.36	5.62	0.91
Miner - l2	30 mil	2.40	2.45	1.24
	35 mil	2.32	2.34	1.20
	40 mil	2.55	2.37	1.24

performance between the distilled, self-distilled and non-distilled models are small and stay approximately within one standard deviation.

The training set performance of the Miner models show similar results: the training performance for distilled models is higher than that of the non-distilled models. On the contrary, the test set performance of the non-distilled models is somewhat higher than that of the distilled models.

The magnitude of the model weights of the self-distilled models is slightly lower than that of the models that were only distilled once. The difference is small, but it holds for every comparison in the table.

4.4 *Experiment 4*

The aim of experiment 4 was to determine the differences in test set performance and magnitude of the model weights between distilled models and l2-regularized models. The results of the experiment are visible in Table 6 and Table 3. The results of the hyperparameter search for the l2-regularization can be found in Appendix C (page 33).

It appears that train and test set performance for both models are approximately the same for each number of training steps. A more notable finding is that the l2-regularized models perform worse on the test set than any of the original and (self-)distilled models. The second surprise is the magnitude of the weight vectors. For the Jumper and the Miner models, the magnitude is around 10% to 15% when compared to the original models, and still approximately 4 times smaller than the magnitudes of the distilled models. Moreover, where Miner had the highest mean reward in the results of the other experiments, it is now significantly lower.

This may not be an entirely fair comparison, as the results for the original and distilled models are the mean results of 4 unique random seeds, while only one l2-regularized model was trained. The low score of the Miner model can therefore be an outlier.

5 DISCUSSION

The results from experiment 1 are somewhat surprising. Mnist is an easy task, so even without distillation the model achieves a very high accuracy. After distillation, the performance on the test set goes up slightly, and the weight magnitude of the model clearly decreases. However, the opposite is true for the ResNet model. Cifar-10 is supposed to be a fairly straightforward task for Resnet-18, which explains the high training set performance before distillation. However, even though the ResNet model was distilled for 20 epochs, the magnitude of the weight vector decreased only slightly and the accuracy of the student model is significantly worse.

It is possible that the number of distillation epochs was too low, but this seems unlikely. In absolute training time, the original model was trained in approximately 5 minutes, while distilling the model took over an hour. A more likely explanation seems to be that the teacher model did not have the required knowledge to pass on to the student, making its KL-divergence loss a negative influence rather than positive for the performance of the student model. This would mean that, rather than spending that hour distilling the model, training the original model longer may lead to better results.

For the DRL side of experiment 1, the results are mostly in line with the hypothesis. Across all models, the magnitudes of the weight vectors of the distilled agents went down to approximately 40% of their original values. The test set performance was lower for the distilled models, but this was only a slight drop. It is interesting that the weights are similar across the three different environments; this may be due to the Procgen environments being designed to all be similarly complicated.

Moreover, all distilled models have a higher performance on the training set than the non-distilled models. This is somewhat surprising, as both the non-distilled as well as the distilled models have had the same number of training steps. Overall, the results for the DRL models are clear, and distillation does seem to lead to a significant drop in magnitude of the weight vector at the cost of a slight drop in test performance. For the supervised learning models there seems to be no such effect, but this side of the experiment was limited in that it only trained two models.

The aim of experiment 2 was to determine whether improvement in test set performance is due to the additional training steps, or due to the distillation. Overall, all models performed mostly the same, regardless of the number of training steps or distillation steps. The models that were trained for an additional 2 million time steps did not show any improved performance over the models that were distilled less. On one hand this could indicate that the better performance is only due to distillation and

that the extra DRL training steps do not have a significant effect. On the other hand, maybe adding 2 million extra environment steps is not enough to show that there is a difference between the two groups. The models that were trained for 5 million timesteps do have a slightly higher magnitude of the weight vector, but this is to be expected. As the model has trained for more timesteps, the weights will have had more updates and therefore have a larger upper bound of possible values. Given these results, it seems somewhat likely that knowledge distillation will improve test set performance faster rather than slower, but there is no direct relationship between test set performance and the weight magnitude of the neural network.

The aim of experiment 3 was to determine whether additional self-distillation could improve performance. Comparing the self-distilled model with the models that were only distilled once shows a small drop in weight vector magnitude for each model. Moreover, the performance of the models gets slightly worse. In a way this result is comparable to the results in (Furlanello et al., 2018), where the main improvement in performance came from using the self-distilled agents as an ensemble, rather than increasing the performance of a single agent. It is not possible to draw a hard conclusion here, given that both models were only self-distilled once, but based on these results it does not seem like self-distillation improves the performance of DRL agents.

Experiment 4 compared the test set performance and the magnitude of the weight vector of the l2-regularized models with the results from experiment 1 and 3. The weight vector magnitudes were lower than any of the magnitudes of the distilled networks. The test set performance for Jumper does see a drop in performance, but relative to the size of the weight magnitude, this is a better result than expected. The opposite is not true for the Miner results, as the model performance falls greatly. It is odd that the Miner results were lower than the Miner results, given that for all the other models it was the environment with the highest mean reward. This may be underfitting caused by the l2-regularization being too strong and removing important weights. The comparison between knowledge distillation and l2-regularization seems to be that l2-regularization is able to lower the magnitude of the network weights more than knowledge distillation, but this can easily lead to underfitting, even after performing a hyperparameter search for the l2-regularization.

5.1 *Limitations*

Overall, the main limitation of this thesis was the number of simulations. Training an agent based on 200 easy Procgen levels for 40 million timesteps

would take up to 12 hours, and running these on a shared server made it hard to do as many simulations as I would have liked. To increase external validity and limit the effect of any possible outliers in agent performance, future research should aim to perform these experiments again, but using other Procgen environments. Moreover, it is not a given that the results from this research will hold for the hard Procgen levels either.

The supervised learning parts of experiment 1 should have been more systematic, with more fine-tuning of the ResNet network and a number of random splits of the train and test sets. The same is true for experiment 4, which should have been trained using more random seeds.

Moreover, this thesis did not include several baselines that are common in DRL. One such group is Atari games. Another set that was not included in this thesis are physics simulations such as Mujoco (Todorov, Erez, & Tassa, 2012). It would be interesting to replicate this methodology on the Mujoco environments, such as Ant and Half-Cheetah, as these environments are somewhat more noisy than the Procgen ones.

Another issue was the limited number of self-distillation iterations. In Furlanello et al. (2018), the models were distilled many more times, rather than just twice. The two consecutive self-distillations in this thesis did show a drop in the magnitude of the weight vector. It may be interesting to see to what extent this can be lowered before the model starts underfitting, as described in Mobahi et al. (2020).

One other aspect that may have impacted the overall performance of all models is the limited aptitude of the teacher models. If the output of the teacher model includes any systematic mistakes, it will pass these on to its students. The performance of the teacher models was not validated in a systematic way. Especially for the models in experiment 1, an accuracy of around 80% on Cifar-10 seemed like a strong teacher model, but this may have been insufficient. Future research can avoid this problem by setting a minimum benchmark performance based on the performance of similar models in literature.

5.2 *Future Work*

This thesis used PPO as its DRL algorithm, but the methodology described here can be used with a variety of policy gradient models. It may be interesting to explore whether there are any DRL models that lend themselves better to distillation than others.

The advantage of using the PPD algorithm is that the weight vector has an upper-bound. This limits the optimizer to find a local minimum within a restricted space close to the origin. However, it is not clear if this

is useful in every situation, such as when the optimal solution is by itself close to the origin. This can also be explored more thoroughly.

One aspect of the DRL performance that was left out of the scope of this thesis is a qualitative analysis of the agents. It may be worthwhile to compare the behavior of repeatedly self-distilled agents to determine whether poor teacher models pass on any visible behaviors and how this affects future iterations of self-distillation.

6 CONCLUSION

The aim of this thesis was to examine the use of knowledge distillation in DRL with a focus on generalization performance and magnitude of the model weights. After an introduction to the mathematics of DRL and an overview of DRL and knowledge distillation techniques, four distillation experiments were performed using the PPO algorithm and PPD loss function to find the optimal policy and perform (self-)distillation.

Overall, the first research question of the thesis was answered fairly conclusively, with the magnitude of the weights of the neural network lowering significantly, at a slight cost of generalization performance. It is hard to draw a strong conclusion for the other three research questions. As for the speed of the improved test set performance, this effect is either absent, or the experiment was too limited in its number of training steps to show the effect. The results for self-distillation seemed worse than expected, but this can be due to performing too few simulations. Lastly, the l2-regularization seems to have a stronger effect on the magnitude of the network weights than knowledge distillation, but may also be prone to underfitting the model.

REFERENCES

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., & Bellemare, M. G. (2022). *Reincarnating reinforcement learning: Reusing prior computation to accelerate progress.* arXiv. Retrieved from https://arxiv.org/abs/2206.01626 doi: 10.48550/ARXIV.2206.01626

Bartlett, P. (1996). For valid generalization the size of the weights is more important than the size of the network. *Advances in neural information processing systems*, *9*.

Bartlett, P. L., Montanari, A., & Rakhlin, A. (2021). Deep learning: a statistical viewpoint. *Acta Numerica*, *30*, 87–201. doi: 10.1017/S0962492921000027

Bellman, R. (1958, Sep 01). Dynamic programming and stochastic control processes. *Information and Control*, *1*(3), 228-239.

Retrieved from https://www.sciencedirect.com/science/article/pii/S0019995858800030

Breiman, L., & Shang, N. (1996). Born again trees. *University of California, Berkeley, Berkeley, CA, Technical Report*, *1*(2), 4.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym.* arXiv. Retrieved from https://arxiv.org/abs/1606.01540 doi: 10.48550/ARXIV.1606.01540

Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* (pp. 535–541).

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2019). *Leveraging procedural generation to benchmark reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1912.01588 doi: 10.48550/ARXIV.1912.01588

Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2018). *Quantifying generalization in reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1812.02341 doi: 10.48550/ARXIV.1812.02341

Czarnecki, W. M., Pascanu, R., Osindero, S., Jayakumar, S. M., Swirszcz, G., & Jaderberg, M. (2019). *Distilling policy distillation.* arXiv. Retrieved from https://arxiv.org/abs/1902.02186 doi: 10.48550/ARXIV.1902.02186

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Doersch, C., & Zisserman, A. (2019). Sim2real transfer learning for 3d human pose estimation: motion to the rescue. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2019/file/d4a93297083a23cc099f7bd6a8621131-Paper.pdf

El Kessab, B., Daoui, C., Bouikhalene, B., Fakir, M., & Moro, K. (2013). Extraction method of handwritten digit recognition tested on the mnist database. *International Journal of Advanced Science & Technology*, *50*, 99–110.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., ... Kavukcuoglu, K. (2018). *Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures.* arXiv. Retrieved from https://arxiv.org/abs/1802.01561 doi: 10.48550/ARXIV.1802.01561

Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., ... et al. (2022). Discovering faster matrix multiplication algo-

rithms with reinforcement learning. *Nature*, *610*(7930), 47–53. doi: 10.1038/s41586-022-05172-4

Furlanello, T., Lipton, Z., Tschannen, M., Itti, L., & Anandkumar, A. (2018, 10–15 Jul). Born again neural networks. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 1607–1616). PMLR. Retrieved from https://proceedings.mlr.press/v80/furlanello18a.html

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition.* arXiv. Retrieved from https://arxiv.org/abs/1512.03385 doi: 10.48550/ARXIV.1512.03385

Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network.* arXiv. Retrieved from https://arxiv.org/abs/1503.02531 doi: 10.48550/ARXIV.1503.02531

Hinton, G. E. (1987). Learning translation invariant recognition in a massively parallel networks. In J. W. de Bakker, A. J. Nijman, & P. C. Treleaven (Eds.), *Parle parallel architectures and languages europe* (pp. 1–13). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hinton, G. E., & Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on computational learning theory* (pp. 5–13).

Hong, Z.-W., Nagarajan, P., & Maeda, G. (2020). *Collaborative inter-agent knowledge distillation for reinforcement learning.* Retrieved from https://openreview.net/forum?id=BkeYSlrYwH

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2016). *Densely connected convolutional networks.* arXiv. Retrieved from https://arxiv.org/abs/1608.06993 doi: 10.48550/ARXIV.1608.06993

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., & Whiteson, S. (2020). *Transient non-stationarity and generalisation in deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/2006.05826 doi: 10.48550/ARXIV.2006.05826

Kim, T., Oh, J., Kim, N., Cho, S., & Yun, S.-Y. (2021). *Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation.* arXiv. Retrieved from https://arxiv.org/abs/2105.08919 doi: 10.48550/ARXIV.2105.08919

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv. Retrieved from https://arxiv.org/abs/1412.6980 doi: 10.48550/ARXIV.1412.6980

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images..

Krogh, A., & Hertz, J. (1991). A simple weight decay can improve generalization. In J. Moody, S. Hanson, & R. Lippmann (Eds.), *Advances in neural information processing systems* (Vol. 4). Morgan-Kaufmann. Re-

trieved from https://proceedings.neurips.cc/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf

Liu, S., Lever, G., Wang, Z., Merel, J., Eslami, S. M. A., Hennes, D., ... Heess, N. (2021). *From motor control to team play in simulated humanoid football.* arXiv. Retrieved from https://arxiv.org/abs/2105.12196

Melo, F. S. (2001). Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, 1–4.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., ... Dean, J. (2020). *Chip placement with deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/2004.10746 doi: 10.48550/ARXIV.2004.10746

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

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013a). *Playing atari with deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1312.5602 doi: 10.48550/ARXIV.1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013b). *Playing atari with deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1312.5602 doi: 10.48550/ARXIV.1312.5602

Mobahi, H., Farajtabar, M., & Bartlett, P. (2020). Self-distillation amplifies regularization in hilbert space. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 3351–3361). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2020/file/2288f691b58edecadcc9a8691762b4fd-Paper.pdf

Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikhin, D., Garipov, T., Shvechikov, P., ... Wilson, A. G. (2018). Improving stability in deep reinforcement learning with weight averaging. In *Uncertainty in artificial intelligence workshop on uncertainty in deep learning.*

OpenAI, :, Berner, C., Brockman, G., Chan, B., Cheung, V., ... Zhang, S. (2019). *Dota 2 with large scale deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1912.06680 doi: 10.48550/ARXIV.1912.06680

Parisotto, E., Ba, J. L., & Salakhutdinov, R. (2015). *Actor-mimic: Deep multitask and transfer reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1511.06342 doi: 10.48550/ARXIV.1511.06342

Puterman, M. L. (1994). Markov decision processes: Discrete stochastic dynamic programming. In *Wiley series in probability and statistics.*

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1-8. Retrieved from http://jmlr.org/papers/v22/20-1364.html

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., ... Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

Schmitt, S., Hudson, J. J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W. M., ... Eslami, S. M. A. (2018). *Kickstarting deep reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1803.03835 doi: 10.48550/ARXIV.1803.03835

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). *Trust region policy optimization.* arXiv. Retrieved from https://arxiv.org/abs/1502.05477 doi: 10.48550/ARXIV.1502.05477

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.* arXiv. Retrieved from https://arxiv.org/abs/1707.06347 doi: 10.48550/ARXIV.1707.06347

Spielberg, S., Gopaluni, R., & Loewen, P. (2017). Deep reinforcement learning approaches for process control. In *2017 6th international symposium on advanced control of industrial processes (adconip)* (pp. 201–206).

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.

Tang, J., Shivanna, R., Zhao, Z., Lin, D., Singh, A., Chi, E. H., & Jain, S. (2020). *Understanding and improving knowledge distillation.* arXiv. Retrieved from https://arxiv.org/abs/2002.03532 doi: 10.48550/ARXIV.2002.03532

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., ... Pascanu, R. (2017). *Distral: Robust multitask reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/1707.04175 doi: 10.48550/ARXIV.1707.04175

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 ieee/rsj international conference on intelligent robots and systems* (p. 5026-5033). doi: 10.1109/IROS.2012.6386109

Uchendu, I., Xiao, T., Lu, Y., Zhu, B., Yan, M., Simon, J., ... Hausman, K. (2022). *Jump-start reinforcement learning.* arXiv. Retrieved from https://arxiv.org/abs/2204.02372 doi: 10.48550/ARXIV.2204.02372

Watkins, C. J. C. H., & Dayan, P. (1992, May 01). Q-learning. *Machine Learning*, *8*(3), 279-292. Retrieved from https://doi.org/10.1007/BF00992698 doi: 10.1007/BF00992698

Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., ... Kitano, H. (2022, Feb 01). Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, *602*(7896), 223-228. Retrieved from https://doi.org/10.1038/s41586-021-04357-7 doi: 10.1038/s41586-021-04357-7

Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., & Ma, K. (2019, October). Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the ieee/cvf international conference on computer vision (iccv)*.

APPENDIX A

All Original Models and Weights

Environment	Time Steps	Seed	Mean Train Reward	Mean Test Reward	Weight Vector
Jumper	25 million	201	5.940	5.890	7.820724
Jumper	30 million	201	6.410	5.790	8.526298
Jumper	35 million	201	5.850	5.450	9.217484
Jumper	40 million	201	6.270	5.550	9.809106
Jumper	25 million	401	6.120	6.010	7.648736
Jumper	30 million	401	5.900	5.740	8.410512
Jumper	35 million	401	5.940	5.890	9.089067
Jumper	40 million	401	5.970	5.560	9.644209
Jumper	25 million	601	6.540	5.580	7.681262
Jumper	30 million	601	6.040	5.690	8.226899
Jumper	35 million	601	5.960	5.990	8.824913
Jumper	40 million	601	5.820	5.780	9.451139
Jumper	25 million	801	6.510	5.700	7.745313
Jumper	30 million	801	6.360	5.700	8.407659
Jumper	35 million	801	6.250	5.680	9.099449
Jumper	40 million	801	6.230	5.750	9.665756
Miner	25 million	201	9.120	9.804	7.443427
Miner	30 million	201	9.413	9.754	8.182845
Miner	35 million	201	9.357	9.510	8.889602
Miner	40 million	201	9.145	9.815	9.567511
Miner	25 million	401	8.048	9.040	7.274209
Miner	30 million	401	8.395	8.951	7.981484
Miner	35 million	401	8.230	9.081	8.660154
Miner	40 million	401	8.650	9.221	9.246495
Miner	25 million	601	8.684	9.659	7.230048
Miner	30 million	601	8.721	9.560	8.003617

Miner	35 million	601	9.333	9.719	8.670640
Miner	40 million	601	8.975	9.472	9.356879
Miner	25 million	801	9.290	9.166	7.470896
Miner	30 million	801	9.219	9.870	8.243422
Miner	35 million	801	9.476	9.762	8.962582
Miner	40 million	801	9.152	9.729	9.623172
Ninja	25 million	201	4.940	4.890	8.217969
Ninja	30 million	201	5.290	5.490	8.889606
Ninja	35 million	201	5.330	5.430	9.498387
Ninja	40 million	201	5.610	5.650	10.098417
Ninja	25 million	401	5.960	5.700	8.118471
Ninja	30 million	401	6.110	6.150	8.762125
Ninja	35 million	401	6.220	6.240	9.386116
Ninja	40 million	401	6.150	6.420	9.932187
Ninja	25 million	601	5.580	5.750	8.130448
Ninja	30 million	601	5.940	6.090	8.860562
Ninja	35 million	601	5.880	5.900	9.534345
Ninja	40 million	601	5.820	5.800	10.106650
Ninja	25 million	801	5.580	5.470	7.879668
Ninja	30 million	801	5.820	5.800	8.607367
Ninja	35 million	801	6.130	6.590	9.329143
Ninja	40 million	801	6.450	6.340	9.912223

APPENDIX B

Table 8: l2-hyperparameter search for Jumper. Top row shows the values for the hyperparameter. The left number indicates training set performance, the right number indicates test set performance

Timesteps	0.1	0.01	0.001
10 mil	3.3, 3.11	3.72, 3.71	6.42, 5.09
15 mil	3.43, 2.98	3.52, 2.76	6.45, 5.33
20 mil	3.14, 2.8	3.8, 3.62	6.47, 5.45

Table 9: l2-hyperparameter search for Miner. Top row shows the values for the hyperparameter. The left number indicates training set performance, the right number indicates test set performance

Timesteps	0.1	0.01	0.001
10 mil	2.89, 2.71	4.05, 3.31	6.02, 4.89
15 mil	3.52, 2.88	3.62, 2.87	5.89, 5.42
20 mil	3.20, 3.12	3.76, 3.81	6.27, 5.02

APPENDIX C

All distilled models and weights

Environment	Seed	Time Steps	Distillation Steps	Weights
jumper	401	25	3000000	3.042056
jumper	801	25	3000000	3.140072
jumper	601	25	3000000	3.434245
jumper	201	25	3000000	3.193066
jumper	401	25	5000000	3.759687
jumper	601	25	5000000	3.911446
jumper	201	25	5000000	3.504147
jumper	801	25	5000000	3.926327
jumper	201	30	3000000	3.157309
jumper	801	30	3000000	3.175082
jumper	601	30	3000000	3.405363
jumper	401	30	3000000	3.013729
jumper	201	30	5000000	3.603258
jumper	801	30	5000000	3.711541
jumper	401	30	5000000	3.641201
jumper	601	30	5000000	3.938025
jumper	601	35	3000000	3.552505
jumper	801	35	3000000	3.381920
jumper	401	35	3000000	2.916032
jumper	201	35	3000000	3.292006
jumper	401	35	5000000	3.481034
jumper	601	35	5000000	3.961704
jumper	201	35	5000000	3.757608
jumper	801	35	5000000	3.697812
miner	201	25	3000000	3.106646
miner	801	25	3000000	3.219705
miner	401	25	3000000	3.312823
miner	601	25	3000000	3.402499

miner	201	25	5000000	3.623801
miner	801	25	5000000	3.908468
miner	401	25	5000000	3.715319
miner	601	25	5000000	3.914200
miner	401	30	3000000	3.197202
miner	801	30	3000000	3.528189
miner	201	30	3000000	3.212239
miner	601	30	3000000	3.553291
miner	201	30	5000000	3.657186
miner	401	30	5000000	3.856975
miner	601	30	5000000	3.801451
miner	801	30	5000000	3.602057
miner	401	35	3000000	3.469148
miner	601	35	3000000	3.320477
miner	801	35	3000000	3.785386
miner	201	35	3000000	3.302096
miner	401	35	5000000	3.930577
miner	201	35	5000000	3.700599
miner	601	35	5000000	4.088419
miner	801	35	5000000	3.774428
ninja	201	25	3000000	3.160333
ninja	601	25	3000000	3.098647
ninja	801	25	3000000	2.937791
ninja	401	25	3000000	2.871916
ninja	401	25	5000000	3.241063
ninja	201	25	5000000	3.641352
ninja	601	25	5000000	3.490222
ninja	801	25	5000000	3.283665
ninja	201	30	3000000	3.194531
ninja	401	30	3000000	3.091074
ninja	601	30	3000000	2.728130
ninja	801	30	3000000	3.122945
ninja	201	30	5000000	3.208316
ninja	801	30	5000000	3.640549
ninja	601	30	5000000	3.140077
ninja	401	30	5000000	3.344461
ninja	201	35	3000000	3.274333
ninja	601	35	3000000	3.072038
ninja	801	35	3000000	2.929702
ninja	401	35	3000000	3.103425
ninja	401	35	5000000	3.469126
ninja	201	35	5000000	3.772279

ninja	601	35	5000000	3.543426
ninja	801	35	5000000	3.369865