Transfer Learning in DQN using weighted layers copying

Gal Shachaf, Eitan Ziv, Tom Zahavy
Electrical Engineering Department
The Technion - Israel Institute of Technology
Haifa 32000, Israel

Abstract

Transfer learning is a research problem that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. In DRL, we can refer to transfer learning as the ability to use knowledge gained while training an agent in one domain and applying it to the training of another agent, usually in a different domain. In this article we present our conclusions regarding the methods of transfer learning in DQNs. By transferring the weights of different parts of the networks, we sought to improve the learning rate and maximum reward achieved by the DQN algorithm. This method was put to trial on agents trained on domains of different affinity: from agents trained on the same domain, to agents trained on very dissimilar domains. We've seen in most of our experiments that Transfer learning by copying of weights in DQN agents, compared to random seed, tends to lead to negative transfer. We showed that the process mostly leads to negative impact in the form of an early plateau in the learning curve. When considering only short training time (few episode), though, some positive impact was achieved. These findings are important as they counter a somewhat popular belief that initializing a DNN from a source network will always be at least as good, if not better, as random seed initialization.

Introduction

Transfer Learning in Deep Networks

Studies in the area of deep reinforcement learning have shown that policies can be learned in an end-to-end manner with high-dimensional sensory inputs in many challenging task domains, such as arcade game playing, robotic manipulation, and natural language processing. As a combination of reinforcement learning with deep neural networks, deep reinforcement learning exploits the ability of deep networks to learn salient descriptions of raw state inputs, and thus bypasses the need for human experts to handcraft meaningful state features, which always requires extensive domain knowledge. One of the successful algorithms is Deep Q-Network (DQN), Mnih et al. (2015), which learns game playing policies for Atari 2600 games by receiving only image frames as inputs. Though DQN can surpass human-expert level across many Atari games, it takes a lot of resources (both time and compute power) to fully train a DQN. In addition, each DQN is trained specifically to play a single game, and usually performs poorly on other games. When considering the use of DQN in real-life applications (e.g. robotic manipulation), few questions arise: How well could an agent trained in a lower-cost simulator perform in the real world? Can we train an agent to be "general" enough, so that it can easily become an expert in any domain? Is it possible to transfer any kind of knowledge from an agent trained in domain A to improve the training of an agent in domain B? The challenge of Transfer learning holds an inherent difficulty, drawn from the basic trade off of generalization. An agent well trained on a specific domain is well fitted to that domain, and perhaps that domain only, in every aspect. Is it even possible to transfer knowledge general enough from a specialized agent?

Weighted Layers Copying

While there had been considerable improvements in our knowledge of how to create high-performing architectures and learning algorithms, our understanding of how these large neural models operate has lagged behind. Neural networks have long been known as black boxes because it is difficult to understand exactly how any particular, trained neural network functions due to the large number of interacting, non-linear parts (J. Yosinski and Lipson (2015)). Large modern neural networks are even harder to study because of their size; for example, understanding the widely-used AlexNet DNN involves making sense of the values taken by the 60 million trained network parameters. Hence, we assume it would be difficult to try and pinpoint the parts of the deep neural network (the agent's "brain") that are relevant to skills needed to perform well in any specific task. This difficulty can also be redefined as classifying the parts or attributes of the deep neural network that are general (and thus transferable) or domain specific (and thus less transferable).

Thus, it was fortunate when Yosinski et al. (2014) showed that features in deep neural networks can be transferable, specifically by using the method of copying the weights of different layers of a deep neural network. Two variations

were employed: the first was copying the layers without farther adjustments to their weights' values; the second was copying the layers as an initial seed, with the latter variation achieving better results for most experiments. This work shows that transfer learning by copying weight layers, besides proving effective for some cases, is also quantifiable: The first layers in a network are more general, and the last are more domain specific.

Negative Transfer

An important approach to transfer learning relies on the assumption that all the tasks are mutually related, in the sense that they share the same underlying representation (Pan and Yang (2016)). "Negative Transfer" happens when the source domain data and task contribute to the reduced performance of learning in the target domain. M. T. Rosenstein and Kaelbling (2005) empirically showed that if two tasks are too dissimilar, then brute-force transfer may hurt the performance of the target task. Some works have been exploited to analyze relatedness among tasks and task clustering techniques, such as A. Argyriou and Pontil (2008), considered situations in which the learning tasks can be divided into groups. Tasks within each group are related by sharing a low-dimensional representation, which differs among different groups. As a result, tasks within a group can find it easier to transfer useful knowledge. Regarding the method of weighted layers copying, Yosinski et al. (2014) showed that negative transfer is stronger as more layers are copied, or when the source and destination domains are less related. According to their work, negative transfer occurs due to two factors: optimization difficulties related to splitting networks in the middle of fragilely co-adapted layers and the specialization of higher layer features to the original task at the expense of performance on the target task. In our paper we will continue and Investigate factors for negative transfer in DQN agents.

Multi-task Transfer Learning

A technique named model compression has been introduced in the past in order to improve transfer learning. This technique utilizes distillation technique to conduct transfer learning for multi-task reinforcement learning, is also referred to as policy distillation A. A. Rusu and G. Desjardins (2016). The goal is to train a single policy network that can be used for multiple tasks at the same time. In general, it can be considered as a transfer learning process with a student-teacher architecture. The knowledge is first learned in each single problem domain as a teacher policy, then it is transferred to a multi-task policy that is known as a student policy. Such transfer learning is conducted via the distillation technique Bucilu and Niculescu-Mizil (2016). Policy distillation for deep reinforcement learning also suffers from negative transfer for some task domains. However, recent work Yin and Pan (2017) has shown that multiple-task agents based on certain architecture applying certain distillation techniques surpasses single-task teacher DQNs over all the task domains. From this we can derive that perhaps the transferable features in a network do not reside exclusively in the first layers of the network, but also in the last layers. Another assumption is that perhaps these features are better transferred using distillation techniques, in order to tackle the domain over-fitting problem suggested in Yosinski et al. (2014).

Our Work

In this paper we will examine the effectiveness of transfer learning in DQN architecture. We will focus on the effectiveness of transfer learning in the form of copying the layers of the agent network, as shown in Yosinski et al. (2014), exploring both the transfer of the first n layers and the transfer of the last n layers between agents. We will try to define inter-domain similarity by "lower dimensional" aspects of visual output, action set and game objectives, and measure the effectiveness of transfer learning according to these parameters. As optimizing steps, we will also employ policy distillation as a mean of reducing over-fitting in the source agent. Finally, we would display in this article the magnitude of negative transfer in DQN.

Background

Deep Q-Networks

A Markov Decision Process is a tuple (S, A, P, R, γ), where S is a set of states, A is a set of actions, P is a state transition probability matrix, where $P(s'|s, a)$ is the probability for transiting from state s to s' by taking action a, R is a reward function mapping each state-action pair to a reward in $\mathbb{R}$, and $\gamma \in [0, 1]$ is a discount factor. The agent behavior in an MDP is represented by a policy π, where the value $\pi(a|s)$ represents the probability of taking action a at state s. The Q-function $Q(s, a)$, also known as the action-value function, is the expected future reward starting from state s by taking action a following policy π, i.e., $Q(s, a) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a\Big]$, where T represents a finite horizon and r_t is the reward obtained at time t. Based on the Q-function, the state-value function is defined as:

$$V(s) = \max_a Q(s, a) \tag{1}$$

The optimal Q-function $Q^*(s, a)$ is the maximum Q-function over all policies, which can be decomposed using the Bellman equation as follows,

$$Q^*(s, a) = \mathbb{E}_{s'}\Big[r + \gamma \max_a Q^*(s', a'|s, a)\Big] \tag{2}$$

Once the optimal Q-function is known, the optimal policy can be derived from the learned action-values. To learn the Q-function, the DQN algorithm Mnih et al. (2015) uses a deep neural network to approximate the Q-function, which is parameterized by θ as $Q(s, a; \theta)$. The deep neural network can be trained by iteratively minimizing the following loss function,

$$L(\theta_i) = \mathbb{E}_{s,a}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2] \tag{3}$$

where θ_i are the parameters from the i-th iteration. In the Atari 2600 games domain, it has been shown that DQN is able to learn the Q-function with low-level pixel inputs in an end-to-end manner Mnih et al. (2015).

To train a DQN, a technique known as experience replay (Lin 1992) is adopted to break the strong correlations between consecutive state inputs during the learning. Specifically, at each time-step t, an experience is defined by a tuple $e_t = \{s_t, a_t, r_t, s_{t+1}\}$, where s_t is the state input at time t, a_t is the action taken at time t, rt is the received reward at t, and s_{t+1} is the next state transited from s_t after taking a_t. Recent experiences are stored to construct a replay memory $D = \{e1, ..., eN\}$, where N is the memory size. Learning is performed by sampling experiences from the replay memory to update the network parameters, instead of using online data in the original order. To balance between exploration and exploitation, given an estimated Q-function, DQN adopts the $\epsilon - greedy$ strategy to generate the experiences.

Policy Distillation

Policy distillation aims to transfer policies learned by one or several teacher Q-network(s) to a single student Q-network via supervised regression. To utilize the knowledge of teacher networks during the transfer, instead of using the DQN loss derived from Bellman error as shown in (3) to update the student Q-network, the output distribution generated by the teacher networks is used to form a more informative target for the student to learn from. Suppose there is a set of m source tasks, $S_1, ..., S_m$, each of which has trained a teacher network, denoted by Q_{T_i} , where $i = 1, ..., m$. The goal is to train a multi-task student Q-network denoted by Q_S. For training, each task domain S_i keeps its own replay memory $D^{(i)} = \{e_k^{(i)}, q_k^{(i)}\}$, where $e_k^{(i)}$ is the k-th experience in $D^{(i)}$, and $q_k^{(i)}$ is the corresponding vector of Q-values over output actions generated by Q_{T_i}. The values $q_k^{(i)}$ serve as a regression target for the student Q-network to learn from. Rather than matching the exact values, it has been shown that training the student Q-network by matching the output distributions between the student and teacher Q-networks using KL-divergence is more effective A. A. Rusu and G. Desjardins (2016). To be specific, the parameters of the multi-task student Q-network θ_S are optimized by minimizing the following loss:

$$L_{KL}\left(D_k^{(i)}, \theta_S\right) = f(\frac{q_k^{(i)}}{\tau}) \cdot \ln\left(\frac{f(\frac{q(i)}{\tau})}{f(q_k^{(S)})}\right) \qquad (4)$$

where $D_k^{(i)}$ is the k-th replay in $D^{(i)}$, $f(\cdot)$ is the softmax function, τ is the temperature to soften the distribution, and $\cdot$ is the dot product.

Method

In our work we've focused on Knowledge Transfer in the form of transferring the weights from the layers of the source agent best deep neural network. The DQN architecture we've used in our experiments consists of 11 layers, where only five layers are weighted layers (Spatial Convolution or Fully-Connected. see figure 1).

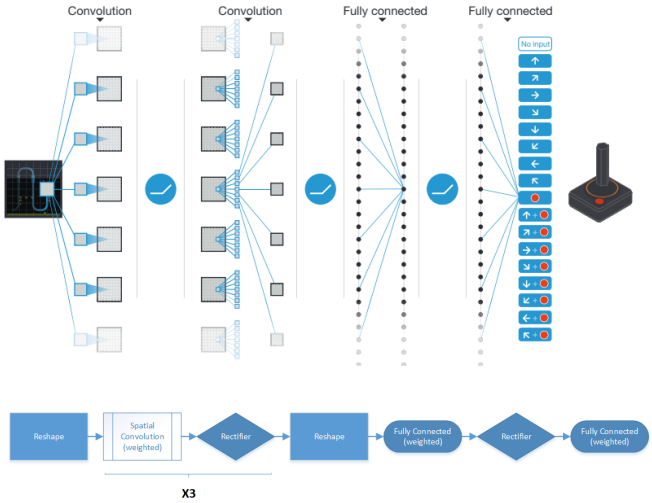

Figure 1: The experiment DQN architecture

Transferring weights

Our method consisted of copying the n first or last weighted layers of the source agent (meaning, the agent trained on the source domain) to a new, random initialized DQN "destination agent", and then training the "destination agent" on the destination domain. During the training itself, we either let the weights of the copied layers adjust according to the training process ("fine tuning"), or left unchanged throughout the entire training session (by zeroing the gradients). Though a typical DQN training session consists on 5e7 steps, we've decided, due to low computing resources, to limit each training session to 25e6, which contains exactly 100 evaluation periods.

Domain Similarity

As mentioned in the introduction chapter, transfer learning is effected greatly by the similarity of the source and destination domains. Hence, we've classified our experiments with the level of similarity of the domains involved in the experiment, to three categories: In-Domain, meaning that both the source and the master are very related (e.g. two different levels in the same Atari 2600 game); Close-Domains, meaning two domains that are considered relatively similar, like two games with similar objectives; and Far-Domains, meaning two games that seem very different from one another. Since there are no proven metrics of similarity between the Atari 2600 games, we chose the following parameters to help us asses the similarity between games:

1. Action set. for each game in the Atari 2600, a different set of controller keys are used, and for different actions. Hence, we can measure the similarity of games using this action set, when two games

who have the exact same actions are very similar, and games that use different controller keys or same keys for different actions are dissimilar.

2. Game Objectives. In the Atari 2600, there are a few families of games, e.g. the Pacman family, or the space invaders family. Each "game family" has a very similar gaming behavior the player's objectives are very similar in all of the games in the family, and the strategies the player employ in each game belonging to a family are also similar. For our experiments, we've treated two games who belong to the same family as Close-Domains.

3. Visual resemblance. The "weakest" metric we've applied relates to the visual similarity between games. In this metric we considered the colors, shapes and sizes of the objects in each game of the pair to grade their similarity.

Preprocessing by Policy Distillation

In order to reduce the over-fitting (or over specification) of the source network on the source domain, we used policy distillation to create an improved, more generalized network to use as a source network to our Knowledge Transfer process. To achieve this we've taken the already trained source agent, created an experience replay set X of 10^6 samples, and used X to train the distilled agent using Soft max to "soften" the Q values and KL divergence for the distance function between the student and the teacher Q values. We used two tactics in order to distill: The first one, which we call "fixed temperature distillation", trained the student agent using each of the samples of the experience only once, with temperature of 0.1; The second one, which we call "varying temperature distillation", had the student learning a few times from each sample, each time relating temperature rising from 0.1 to 0.5 in jumps of 0.1 (5 usages of each sample in total).

Experiments

In our experiments we used the standard DQN implementation by Ilya Kuzovkin, using the Atari 2600 games as target domains. for each target domain chosen, we trained a DQN using 25 million steps, and measured its reward history. We will address this reward history as "baseline". We then proceeded to transfer one or more layers from the source DQN to new randomized DQNs, and measured their ability to learn different games. We then compared all the networks reward history while focusing on learning rate and maximal reward achieved. These games were split into three categories: far-domain, close-domain, in-domain, as described in the "Method" chapter.

Far Domains Transfer

For the far-domain transfer experiment we chose the games Breakout and Qbert, Two games with dissimilar features and game play. We attempted to transfer knowledge from a network that had mastered Breakout to a new network attempting to learn Qbert. Comparing each networks reward history, we found that transferring knowledge between these two far

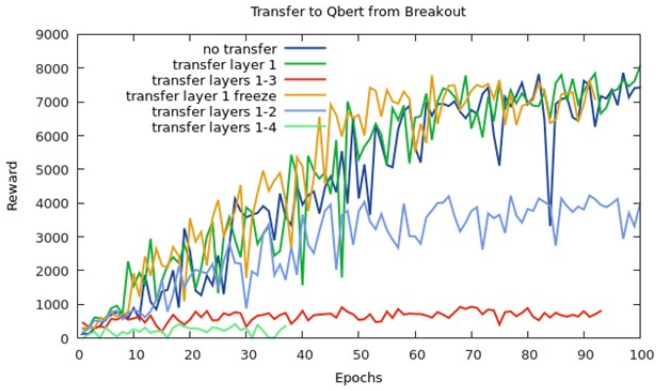

Figure 2: Far domains transfer- Qbert to Breakout

domains, resulted in a negative transfer. It seems that the transferring and even freezing the first layer had little to no effect on the learning rate and with each transferred layer the learning was negatively impacted.

Close Domains Transfer - First Layers

In this experiment, we continued our focus on the first layers. we chose the games Beam rider and Space Invaders, two games with similar features. The player character is positioned in the same place and has similar movements and abilities. The enemy characters are also similar in their abilities, but vary in terms of amount and differ in their positioning and movement. We trained two networks each to play one game. After the networks had mastered the games, we transferred their knowledge (layer 1 only) to two new networks (with and without freeze). The network that received the Beam rider knowledge was then trained on Space Invaders and vice versa. Comparing each networks reward history, we found that transferring knowledge between these two close domains, also resulted in a negative transfer.

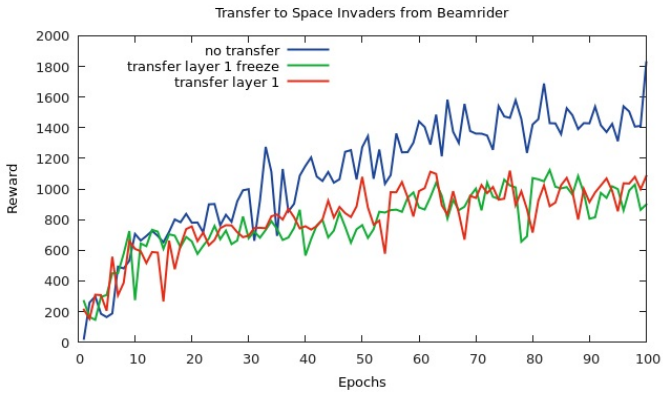

Figure 3: Close domains first layers - from Beam Rider to Space Invaders

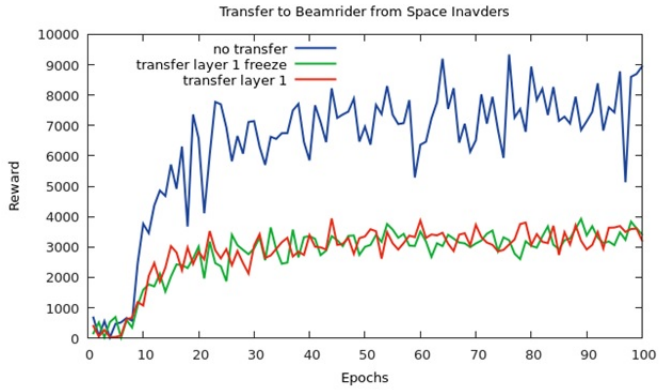

Figure 4: Close domains first layers - from Space Invaders to Beam Rider

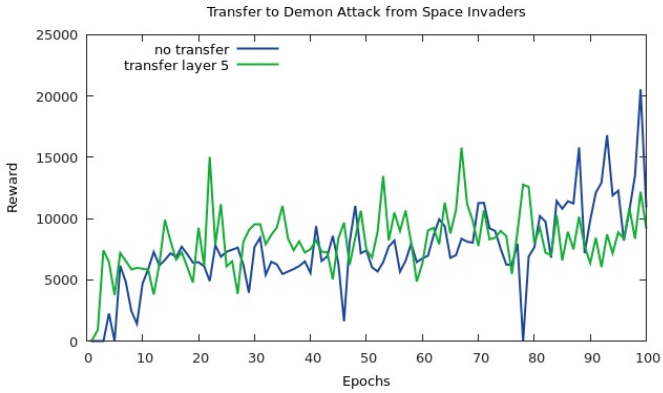

Figure 5: Close domains last layers - from Space Invaders to Demon Attack

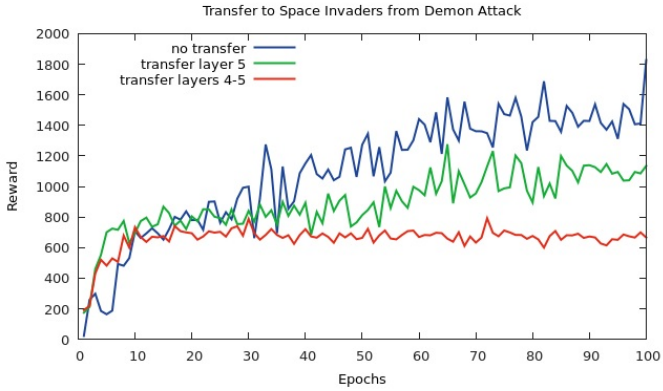

Figure 6: Close domains last layers - from Demon Attack to Space Invaders

Close Domains Transfer - Last Layers

In this experiment we picked two games that have the exact same action interface (meaning, using the same keys for same actions): Space Invaders and Demon Attack, to evaluate the effectiveness of knowledge transfer in the last layer only, which is the layer which makes the decision of which action to perform. This experiment consisted of two parts: with policy distillation and without. In the first part, we've trained a Demon Attack agent after transferring the last layer from a baseline Space Invaders agent (without freeze), and a Space Invaders agent with the two last layers from a baseline Demon Attack agent (without freeze). For the first 80 epochs it seems like the transfer of the last layer is positive: It is higher than the base line at most epochs and with peeks of around twice at some! Sadly, though, as we approached the last 20 epochs, the baseline closed the gap, and at the 100th epoch achieved almost twice than the transfered agent.

In the second part, we tried to use policy distillation in order to reduce the over fitting of the transfered agent. Therefore, after training the source agent on Space Invaders for 100 epochs, we used policy distillation with Soft Max and KL divergence Loss function to create two new distilled source agents: the first one with "fixed temperature distillation" with temperature of 0.1; The second one, with "varying temperature distillation" and temperature rising from 0.1 to 0.5. Using the weights of the last layer of these new (and perhaps less over-fitted) source agents, we trained two new agents. Unfortunately, these two agents delivered even worse results than without policy distillation.

Discussion

This work constitutes a thorough experimental process to try and accumulate the effectiveness of transfer learning in DQNs using weighted layers copying, between domains with varying levels of similarity, using different variations of said methods. Our results from the experiments discussed above show the following:

1. It was shown across all domains and experiments that plain copying of weighted layers transfer learning in DQN result in negative transfer. The effect of this negative transfer became more dominant as the number of layers copied increased. Furthermore, the effect was more dominant when copying layers and freezing their weights' values.
2. When dealing with close domains (i.e. domains with similar objectives, actions and reward states), we've seen slightly different results in accordance to the part of the network we've tried to transfered. While both experiments showed negative transfer when utilizing the training resources to 100%, transferring only the last layer showed some advantage over the baseline up until around 80% of the training time. This could suggest that for close domains (e.g. Simulator and Real World) with limited training resources for the destination agent, transfer of the last layer would have a positive impact.
3. More in Close domains, it seems that policy distillation using both methods had only a negative effect, and lead to even worse negative transfer. This is somewhat counter intuitive, since at first we related the slight failure of direct close domain transfer to the over-fitting of the source

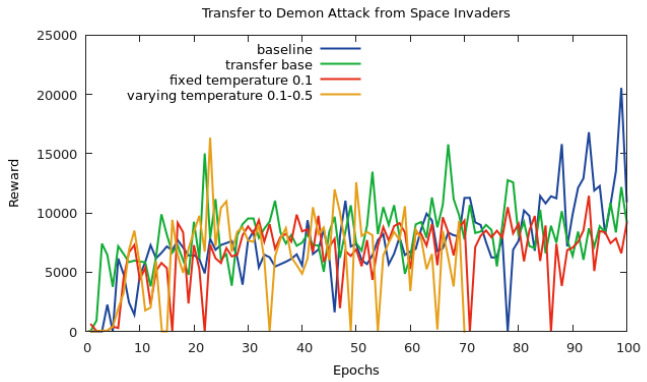

Figure 7: Close domains last layers - with policy distillation

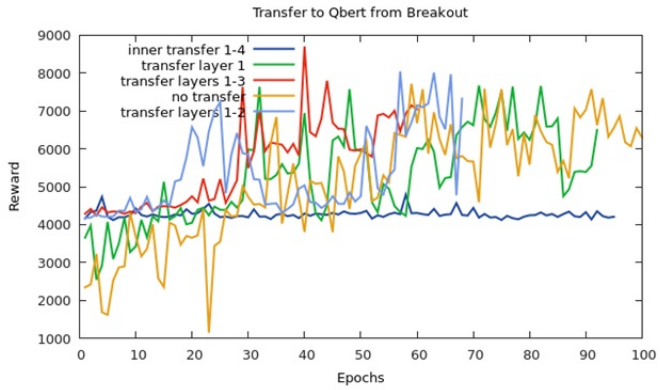

Figure 8: Inner domain - between level 1 and 2 of Qbert

agent to the source domain, a effect that is supposed to be moderated after policy distillation of the network.

4. In the context for far domains (i.e. domains with completely different objectives, actions and reward states), we've seen that while transferring the first layer (with or without freeze) had little to no effect on the learning rate, each added transferred layer lead to a stronger negative effect. This result is similar to what was seen for dissimilar domains transfer in Yosinski et al. (2014).
5. It is interesting to see that while transferring the first layer had a definitive negative effect between close domains, it had only a moderate negative (to none) effect between far domains. This means that in the close domains we've observed (Beam rider and Space Invaders), transfer of the first layer is worse than random initialization, while in far domains, perhaps due to the high dissimilarity between the games, it might have had the same aspects of a random seed.

We try to propose several explanation as for the mentioned results and conclusion:

1. The successful experiment shown in Yosinski et al. (2014) was conducted when the ImageNet database was divided into two groups randomly. Although the two group were non-overlapping, there was no uniqueness in each group, so we can regard the two domains as either very similar, or even as part of the same domain. Such experiment is difficult to conduct in the Atari 2600 Framework, when the equivalence might be perhaps two random levels in the same game.
2. Considering supervised image recognition, the DNN could be interpreted as a serial processing of the picture throughout the DNN. In contrast, in DRL the layers may have greater co-adaption, which would reduce the effectiveness of simple weighted layers transfer.
3. DQN agents are less generalized, due to quite long training time and a highly coupled task set per game. This specificity hinders transfer, as the agents are more likely to become specific and maintain a DNN with significant co-adaption of layers.

Directions For Future Research

Even though the ultimate conclusion of our work in that the proposed method leads only to negative transfer in DQNs, we have had some research leads we did not pursuit:

- It is possible that the generalization of each source domain would improve if trained using less training epochs.
- Further researching the difference in results between the negative transfer effect between different games may lead to a more empiric metric of dissimilarity between different domains.
- If we would separate the two stages of game play - reasoning and decision into two separate, connected DNNs we could create an architecture that is more receptive to transfer.by that, we would Decouple the reasoning (image processing) and decision making parts of the learning process, (E.g. for domains with similar objectives, different visual representations).

References

A. A. Rusu, S. G. Colmenarejo, C. G., and G. Desjardins, J. 2016. Policy distillation. *In ICLR*.

A. Argyriou, A. M., and Pontil, M. 2008. An algorithm for transfer learning in a heterogeneous environment. 71–85.

Bucilu, C. Caruana, R., and Niculescu-Mizil, A. 2016. Model compression. *In SIGKDD* 535–541.

J. Yosinski, J. Clune, A. N. T. F., and Lipson, H. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

M. T. Rosenstein, Z. M., and Kaelbling, L. P. 2005. To transfer or not to transfer. *in a NIPS-05 Workshop on Inductive Transfer: 10 Years Later*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Pan, S. J., and Yang, Q. 2016. A survey on transfer learning. *EEE Transactions on Knowledge and Data Engineering 22* 1345–1359.

Yin, H., and Pan, S. J. 2017. Knowledge transfer for deep reinforcement learning with hierarchical experience replay.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How transferable are features in deep neural networks? 3320–3328.