Towards Knowledge Transfer in Deep Reinforcement Learning

Conference Paper · October 2016

DOI: 10.1109/BRACIS.2016.027

Ruben Glatt
Lawrence Livermore National Laboratory
36 PUBLICATIONS **309** CITATIONS
SEE PROFILE

Felipe Leno da Silva
Lawrence Livermore National Laboratory
62 PUBLICATIONS **1,034** CITATIONS
SEE PROFILE

Anna Helena Reali Costa
University of São Paulo
175 PUBLICATIONS **1,677** CITATIONS
SEE PROFILE

Towards Knowledge Transfer in Deep Reinforcement Learning

Ruben Glatt, Felipe Leno da Silva, and Anna Helena Reali Costa*
Escola Politécnica da Universidade de São Paulo, Brazil
{ruben.glatt,f.leno,anna.reali}@usp.br

***Abstract*—Driven by recent developments in the area of Artificial Intelligence research, a promising new technology for building intelligent agents has evolved. The technology is termed Deep Reinforcement Learning (DRL) and combines the classic field of Reinforcement Learning (RL) with the representational power of modern Deep Learning approaches. DRL enables solutions for difficult and high dimensional tasks, such as Atari game playing, for which previously proposed RL methods were inadequate. However, these new solution approaches still take a long time to learn how to actuate in such domains and so far are mainly researched for single task scenarios. The ability to generalize gathered knowledge and transfer it to another task has been researched for classical RL, but remains an open problem for the DRL domain. Consequently, in this article we evaluate under which conditions the application of Transfer Learning (TL) to the DRL domain improves the learning of a new task. Our results indicate that TL can greatly accelerate DRL when transferring knowledge from similar tasks, and that the similarity between tasks plays a key role in the success or failure of knowledge transfer.**

I. Introduction

A technique that has been gaining a lot of publicity lately is Deep Learning (DL), which has driven many of the recent Machine Learning (ML) developments by profiting from the comeback of neural networks (NN) in Artificial Intelligence (AI) research. DL allows to learn abstract representations of high dimensional input data and can be used to improve many existing ML techniques [1]. DL architectures became one of the most powerful tools in AI in short time, beating long standing records not just in one, but in many domains of ML, as for example in object recognition, hand-written digit recognition, speech recognition or recommender systems.

Another technique that can benefit from DL is the well-researched area of Reinforcement Learning (RL) [2]. In RL, an agent explores the space of possible strategies or actions in a given environment, receives a feedback (reward or cost) on the outcome of the choices made and deduces a behavior policy from his observations. As shown in Figure 1, an agent can interact with its environment by performing an action on each discrete time-step. The environment then answers by updating the current state to a follow-up state and by giving a reward to the agent, indicating the value of performing an action in a concrete state. By performing various actions in a trial-and-error manner, a sequence of states s, actions a, follow-up states s' and rewards r is generated and can be stored in episodes as tuples of $\langle s, a, s', r\rangle$. The goal of the agent is to determine a policy π that maps a state to an action, which maximizes the accumulated reward over the lifetime of the agent.

This form of decision-making can be modeled as a Markov Decision Process (MDP) [4]. The core of the MDP is the Markov Property (MP), which is given if future states of the process depend only upon the present state and the action we take in this state, but not on how this state was reached. Formally an MDP can be described as a tuple of $\langle S, A, T, R\rangle$. In this tuple, S is a finite set of all possible states s, A is a finite set of all possible actions a, T is the transition function, which provides a probability of reaching a follow-up state s' given a state s and an action a, and R is the reward function, which provides the reward r the agent receives, when reaching a follow-up state s' from state s after executing action a.

RL already achieves excellent results in a variety of domains from the board-game domain [5] to autonomous helicopter flight [6]. Recent work in combining the power of DL techniques with RL (Deep Reinforcement Learning - DRL) have led to more powerful intelligent agents, which are now able to solve problems with high-dimensional input data, like images, with reasonable computational efforts as demonstrated in the

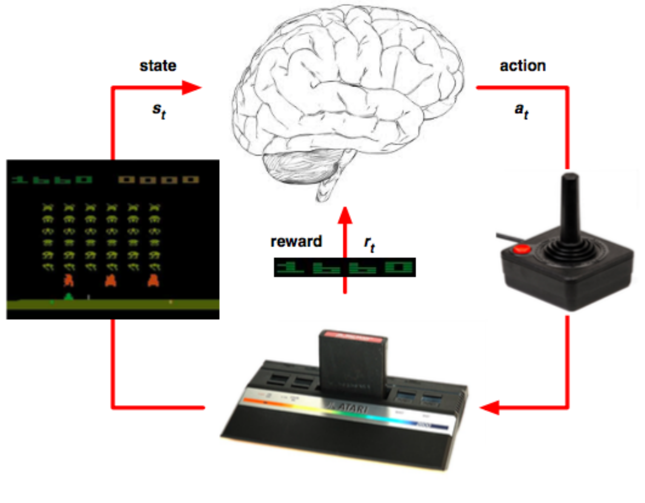

Fig. 1. Principal sketch of Reinforcement Learning in the Atari game playing domain for single task learning [3].

*We are grateful for the support from CAPES, CNPq (grant 311608/2014-0), and the São Paulo Research Foundation (FAPESP), grant 2015/16310-4. The HPC resources for the computation are provided by the Superintendency of Information at the University of São Paulo. We also thank Google for the support due to the Google Research Awards 2015 for Latin-America.

Atari game playing domain by [7].

Although this evolution led to excellent results for single task learning, it still does not provide significant improvements in multi task learning scenarios. Multi task learning can be considered a challenge for Transfer Learning (TL), which supports the ability to generalize gathered knowledge in one or several source tasks and transfer it to other tasks, offering the advantage of not having to learn every task from scratch because it relies on abstractions of past experiences [8]. Taylor and Stone [9] name three main developments for the latest interest in the topic: (1) RL techniques have achieved very notable successes and outperformed other ML techniques for a variety of difficult tasks in agent theory, (2) other classical ML techniques have matured enough to assist with TL, and (3) the initial results show that this combination can be very effective and has a positive effect on various aspects of RL. They also conclude that there is considerable room for more work in the area, a fact that is still true today.

Given those developments, our long-term ambition is to find ways which enable successful transfer of knowledge in DRL. In this article we present a novel empirical evaluation to understand the outcomes from Transfer Learning in DRL domains. In our experimental scenario, a new target task is presented to an agent that has access to the knowledge acquired in already solved source tasks, and the effects of the previous knowledge on the learning process are evaluated. Our experimental results indicate that the transfer of knowledge can be either benefitial to the agent, neutral, or even hamper the learning, depending on the similarity between the source and target tasks.

The remainder of this article is organized as follows: In Section II we outline the underlying principles of DRL and related work. In Section III we describe the TL approach, its challenges and examples from the literature. In Section IV we introduce our proposal. In Section V we present our experiments and discuss the results. The article closes with Section VI, where we specify possible next research steps.

II. Background and related work

The methods to solve RL problems can be divided into three main groups: critic-only, actor-only, and actor-critic methods. In this context we are concerned with critic-only methods, which in general use a Temporal Difference (TD) approach to solve the RL problem. TD methods are related to Monte Carlo methods [10], because they rely on sampling from the environment for the learning task, and to Dynamic Programming [11], because they approximate future results from past experiences. A popular example of TD methods is the Q-Learning algorithm [12], where the optimal action-value function $Q^*(s,a)$ is estimated by a function approximator. An advantage of this method is that it is *model-free*, which means that it can directly learn the policy without learning the transition and reward functions of the MDP explicitly. Another advantage is, that it can be trained *off-policy*, following a strategy that ensures adequate behaviour to balance exploration and exploitation while exploring the state space. The function approximator is formulated as a Bellman equation [13], which was shown to converge to an optimal solution, if the function is updated iteratively indefinitely, $Q_i \rightarrow Q^*, i \rightarrow \infty$. The optimal Q-value reflects the expected sum of the immediate reward r and the maximal discounted reward for future actions, assuming optimal behaviour in the future,

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a')|s,a]. \tag{1}$$

The policy describes a mapping from a state to an action and the optimal policy π^* is then defined by choosing the action a, which maximizes $Q^*(s,a)$ for a given state s:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} Q^*(s,a). \tag{2}$$

Although it is not a new idea to use a NN as function approximator for RL problems, as shown for example in Fitted Q-Learning [14], advances in algorithms for DL have brought upon a new wave of successful applications. The break-through work for this new approach was first published in 2013 as a workshop presentation and later refined in an extensive journal article for the Atari game playing domain [7]. The authors use a combination of convolutional and fully-connected layers to approximate the possible Q-values for a given state, nominated Deep Q-Network (DQN). The results demonstrate that a single architecture can successfully learn control policies in a variety of different tasks without using prior knowledge.

The algorithm is described in Algorithm 1, where the states are sequences of images x and actions a, $s_t = x_1, a_1, x_2, a_2, ..., a_{t-1}, x_t$, and $\phi(s)$ is a preprocessing function, which applies some steps to reduce input dimensionality and stacks the m (here 4) most recent frames to produce the input for the DQN, so that the requirement of full observability for MDPs is not violated. The Q-function is then described as $Q(\phi, a; \theta)$, where θ represents the weights of the network. The approach differs from standard Q-Learning mainly in two ways to make it suitable for training large NN without diverging. The first difference is the use of *experience replay*, a form of batch learning, where the agent stores its experience at each time-step $e_t = (s_t, a_t, r_t, s_{t+1})$ in a replay memory $D_t = \{e_1, ..., e_t\}$. During training the agent then draws off-policy random experiences from D_t to improve data efficiency, break up correlations and eliminate unwanted feedback-loops. The second difference is the introduction of a target DQN $\hat{Q}(\phi, a, \theta^-)$, which is updated with the weights θ of the training DQN $Q(\phi, a, \theta)$ only every C time-steps. This adds a delay between updating the parameters and the effect on the trained network and makes the algorithm more stable against oscillations and divergence.

The weights of the network can be trained by optimizing the loss function of the network

$$L_i(\theta_i) = \mathbb{E}\Big[\Big(\big(r + \gamma \max_{a'} \hat{Q}(s',a';\theta_i^-)\big) - Q(s,a;\theta_i)\Big)^2\Big]. \tag{3}$$

Algorithm 1 Deep Q-Learning with experience replay

```
1: Initialize replay memory $D$
2: Initialize training action-value function $Q(\phi, a, \theta)$ with random weights $\theta$
3: Initialize target action-value function $\hat{Q}(\phi, a, \theta^-)$ with weights $\theta^- = \theta$
4: repeat (for each episode )
5:     initialize $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
6:     repeat (for each time-step in episode)
7:         Choose action $a_t$ in $\phi_t$ following $\varepsilon$-greedy strategy in $Q(\phi_t, a, \theta)$
8:         Observe new image $x_{t+1}$ and reward $r_t$
9:         Set state $s_{t+1} \leftarrow s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10:        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
11:        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
12:        repeat (for each transition in minibatch)
13:            if episode terminates at step $j + 1$ then
14:                set $y_j \leftarrow r_j$
15:            else
16:                set $y_j \leftarrow r_j + \gamma \max_a \hat{Q}(\phi_j, a, \theta^-)$
17:            end if
18:            Perform gradient descent step on $(y_j - Q(\phi_j, a_j, \theta))^2$ with respect to network parameters $\theta$
19:        until no more elements in minibatch
20:        Every C steps do $\hat{Q} \leftarrow Q$
21:        Set $\phi_t \leftarrow \phi_{t+1}$ and $s_t \leftarrow s_{t+1}$
22:    until episode ends
23: until no more episodes
```

Differentiating the loss function with respect to the weights then leads to the gradient, which will be used for the gradient descent:

$$\begin{aligned}\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\Big[\Big(\big(r + \gamma \max_{a'} \hat{Q}(s', a'; \theta_i^-)\big) - \\ Q(s, a; \theta_i)\Big) \nabla_{\theta_i} Q(s, a; \theta_i\Big].\end{aligned} \tag{4}$$

In this context the trained DQN can be seen as a kind of end-to-end RL approach, where the agent can learn a policy directly from its input data without having to find a suitable representation manually first. This is especially interesting, because it offers the opportunity to efficiently work with high dimensional input data and opens up a lot of possible applications, which were too expensive to solve before.

Other researchers already picked up the idea and started to work with the DQN basic structure to improve results and learning speed. The authors of [15] introduce an extension of the DQN by adding a Long Short Term Memory (LSTM) in the form of an additional recurrent layer after the convolution layers to improve performance on partially observed states. Another extension of the DQN was proposed by [16] to reduce overestimation of action values and improve performance on a number of games by decoupling the action selection from the action evaluation. Another publication in this domain integrates an actor-critic structure with a Deep Deterministic Policy Gradient technique and shows that it can also learn competitive policies for low dimensional input data [17]. Introducing Gorila (General Reinforcement Learning Architecture), there have also been massive improvements on the computational architecture of the DQN to allow for distributed computation with parallel actors, shared experience replay, and distributed NN, leading to better results and speed-ups by an order of magnitude [18]. The role and importance of the experience replay has been researched in [19], where a framework for prioritizing experience is proposed, so as to replay important transitions more frequently, and therefore learn more efficiently.

III. Transfer Learning for Deep Q-Networks

In ML many approaches have achieved good solutions for single task learning. A remaining problem is the generalization of these approaches to allow faster and more efficient multi task learning. The idea of using accumulated knowledge for this kind of problems is taken from the human learning ability, which works quite similar. TL emerged from the need to solve this problem and many researchers have focused on expanding ML algorithms with the ability to transfer knowledge, experience or skills to allow the learning of follow up tasks with very few examples. An example for a promising approach is the introduction of an option framework to support knowledge transfer, which provides methods for RL agents to build new high-level skills [20]. Another method is proposed in [21], which shows that building stochastic abstract policies that generalize over past experiences is effective for transferring knowledge represented as a stochastic abstract policy.

Regardless of many positive results, there are still some challenges, which need further research. One of those is how to handle the problem of negative transfer, when the transferred knowledge decreases the expected result instead of improving it. Another one is the determination of when a task or a domain is suitable for transfer and what causes this limitation. Knowing about the similarity degree between two tasks could also provide information about how much training would be necessary to achieve acceptable results in a target task. A final example of the open challenges is the ability to separate data that is necessary for a transfer from data that would just provide an irrelevant bias. In regard to a TL for RL setup, it is not clear how to organize the knowledge transfer across tasks with very different reward functions.

Although there has been much work in the DRL domain lately, the focus seems to be heavily on single task learning. An earlier work, unrelated to DQNs, argues that Deep Convolutional NNs are particularly well suited for knowledge transfer and envisions creating a net that could learn new concepts throughout its lifetime [22]. One of the rare published works concerned with bridging the gap between single and multi task learning for DQNs introduces an *Actor-Mimic* method, which trains a general Single Policy Network (SPN) for a variety of distinct tasks using the guidance of several expert networks. The SPN then generalizes well for new tasks, even without expert guidance [23].

IV. Proposal

Apart from the before mentioned work, a thorough literature review has yet found little evidence on research that aims at combining the advantages of knowledge transfer with DQNs, although it represents a great opportunity to advance the state of the art towards general purpose AI agents. Since each of the techniques involved provides successful methods to deal with different aspects of learning in AI, but also has shortcomings in certain areas, a combination of these approaches may be able to solve harder problems and improve results on existing ones.

We find that not much discussion has been dedicated to analyze under which situations TL is useful for DQN and what consequences arise for the learning agent when TL is blindly applied. Our proposal in this article is to train a DQN in a source task and reuse the trained DQN for the initialization of a new (target) task. The new task is then expected to be learned faster because of the reused knowledge. However, as discussed in Section III, Transfer Learning can hamper the learning process if the tasks are not similar.

Note also that DQNs provide both state and policy abstraction, and it is not easy to separate these functionalities from a trained network. Hence, when transferring a DQN from one task to another we are in fact transferring both the state and policy abstractions.

We experimentally investigate these assumptions under different scenarios to evaluate when knowledge transfer makes sense and when the learner is hampered by its previous knowledge. We also explore the role of similarity between tasks for the transfer of knowledge in DQNs.

V. Experiments

In the Atari game playing domain the agent controls the actions while playing a selected game with the goal of maximizing the game score. For each game the agent can only use the actions that are available for the specific game, which represent a subset of $A = \{$'NOOP', 'FIRE', 'UP', 'RIGHT', 'LEFT', 'DOWN', 'UPRIGHT', 'UPLEFT', 'DOWNRIGHT', 'DOWNLEFT', 'UPFIRE', 'RIGHTFIRE', 'LEFTFIRE', 'DOWNFIRE', 'UPRIGHTFIRE', 'UPLEFTFIRE', 'DOWNRIGHTFIRE', 'DOWNLEFTFIRE'$\}$. To keep in line with the nomenclature in RL we refer to one game (finishes after loss of all lives) as one episode and to the score per game step as the reward per game step. The state space in such games is generally huge, which is the reason a DQN is used as an approximation for the state instead of saving all different states in a Q-table.

Our experiments intend to evaluate the possible effects of TL on DRL. We divided our experiments in two phases: (i) *DRL optimizer definition*; and (ii) *TL evaluation*. While in the former we evaluate different DRL training algorithms and select the best one for the second phase, in the latter we evaluate the performance of TL applied to DRL. We limited our experiments to the games shown in Fig 2, mainly to provide a controlled scenario and ease the analysis of our results.

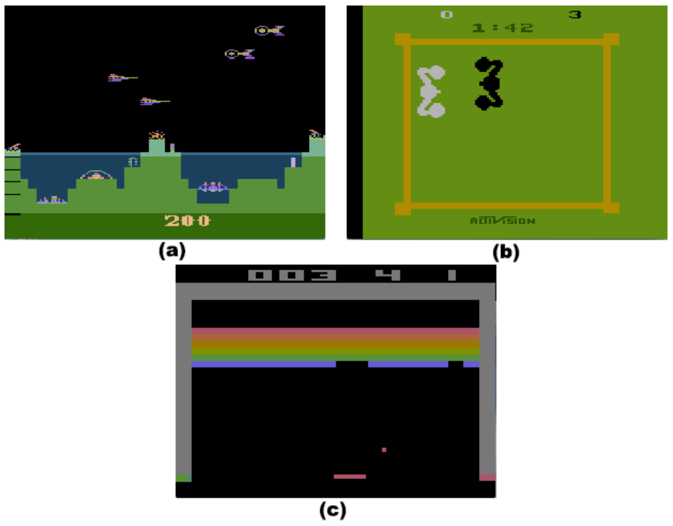

Fig. 2. The games for which DQNs were trained: (a) Atlantis, (b) Boxing, and (c) Breakout.

The first phase of our experiments intended to verify if the choice of optimizer and the number of output nodes would have an impact on the training of the DQN. Therefore we compared two state-of-art optimizers: *RMSProp* [24] and *ADAM* [25]. We selected the game Breakout as the benchmark game for this phase, because we achieved the most stable results with this game during earlier training runs. As in [7], the DQN can be trained using only the actions that matters for the specific game (6 for Breakout). However, another option is to use all the 18 possible combinations of the Atari controller. Therefore, we compared 100 epochs of learning with RMSProp both with 6 outputs and 18 outputs, and ADAM with 18 outputs.

The results are shown in Figure 3. Our graphs show the rewards per epoch as a running average over 5 epochs. It is immediately visible that all different configurations have a similar behaviour. The distinction between the number of output nodes does not have an impact on the results of the game and also did not influence the total training time significantly. ADAM optimizer learns better at the very beginning of the training, rising faster on the average game score and losing less lives earlier. Later, for Breakout between epoch 10 and 15, RMSProp catches up and remains the algorithm with higher average score and fewer loss of live for the rest of the 100 epochs. As RMSProp optimizer achieved the overall best results in average reward per episode, we chose it for the second phase of our experiment. As the difference in learning speed with different number of outputs is small, we performed the training for all possible outputs in the following experiments.

The second phase of our experiments simulates the following situation: A (source) task is given to a learning agent

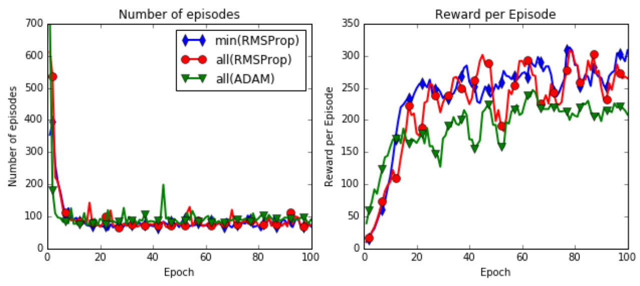

Fig. 3. Comparison between different settings for a DQN trained for the game Breakout.

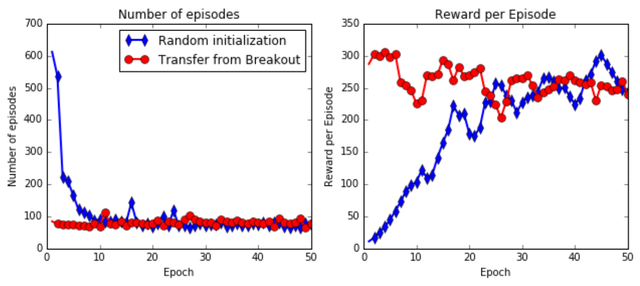

Fig. 4. Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained DQN for Breakout.

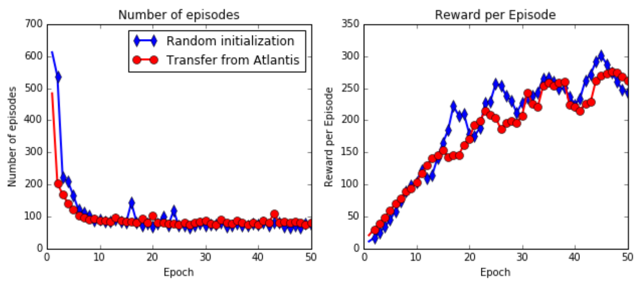

Fig. 5. Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained DQN for Atlantis.

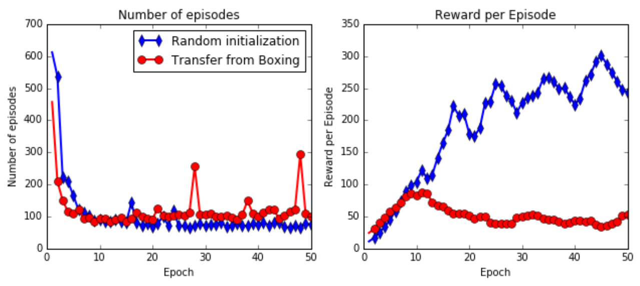

Fig. 6. Comparison between random initialization of Breakout and initialization with the best performing network for a previously trained DQN for Boxing.

that has no background knowledge. After learning an effective policy, a new (target) task is presented to the agent. Even though it has no information about the new task, the previous knowledge remains accessible to the agent, who has no reason to neglect his knowledge base.

In order to evaluate the efficiency of TL in different situations, we firstly trained a DQN for the Breakout game from scratch, then the learning results are compared under the following situations:

1) *TL from similar tasks*: TL is expected to present better results when the source and target tasks are very similar. In order to simulate this situations, we perform a new training phase for *Breakout* initializing the DQN with a previously trained DQN also on *Breakout*.
2) *TL from neutral tasks*: While in *Atlantis* an optimal policy can be achieved by only using the *fire* actions, they are mostly useless in *Breakout* (but they do not lead to terrible situations either). We here evaluate TL when the source task is different from the target task, but the optimal actuation in the source task is not expected to be much worse than a random policy. After training a DQN for the game *Atlantis*, we use it to train a new DQN for *Breakout*.
3) *TL from different tasks*: TL is reported to result in negative transfer when applied carelessly. We here evaluate the learning performance when starting with a very bad policy. The game *Boxing* offers a very different gameplay than *Breakout*, and also has very different outcomes when using the same actions. We here train a DQN in *Boxing* and also use it to train a new DQN for *Breakout*.

Figure 4 shows the results for the first situation. The achieved rewards per episode in the first epochs are much greater when starting with the transferred DQN, while the number of episodes per epoch has already decreased to a good level, meaning that fewer lives are lost during playing the game. These results show the potential of TL when transferring knowledge across similar tasks. Figure 5 depicts the results for the second situation. The number of episodes and average reward per episodes are similar in both situations, which means that the transferred knowledge did not help with the DQN training but also did not lead to worse results than the random initialization. Finally, Figure 6 presents the results for the third situation. In this case it is clear that the learned DQN suffers from negative transfer. The use of a very unfit policy when starting the learning process greatly hampered the DQN optimization, which is shown in the achieved average reward per episode, which is much worse when comparing to the random initialization. The influence was so dominant that the agent was unable to improve his performance until the end of the 50th epoch.

Our results show that the concern about negative transfer is also valid for TL in DRL algorithms. Even though TL is very beneficial to the learning process when the tasks are similar and neutral when the two domains are not very different, when dealing with very uncorrelated domains the negative transfer lets the learner take a long time to overcome the initial bad actuation. This also means that TL cannot be blindly applied to DRL, and the similarity of the source and target tasks must be evaluated before transferring knowledge.

VI. Conclusion and future work

In this article, we evaluated the applicability of TL to DRL through the evaluation of improvements in the learning process when transferring knowledge from past tasks with different degrees of similarity to the target task.

Our results show that the initialization of the DQN plays a far more important role than the choice of optimization algorithm of the gradient decent method of the network. The results also reinforce the importance of being able to start the learning of a new task with experiences from previously learned tasks. When transferring knowledge from similar tasks TL achieved a greatly accelerated learning process, realizing results closer to the optimal actuation since the beginning of the training. However, when applied to unrelated tasks, the negative transfer makes the training performance much worse and very difficult to recover than with random initialization.

These outcomes show that much of the concerns presented from researchers when applying TL in classical RL are also valid for DRL. Being able to find which of the previously learned tasks are similar to the target task (and possibly defining the degree of similarity) is directly correlated to the success or failure when applying TL.

A promising idea as proposed in [26] executes a *Policy Reuse* to leverage past knowledge. When facing a new task, the agent finds similar tasks in a policy library and uses them to accelerate learning or extends the library. In the domain of Atari game playing such a library could lead to a collection of dedicated core policies for different genres of games like for example jump-and-run, platform or racing.

However, defining a "similarity degree" between tasks is not a trivial undertaking. While we defined the similarity of the learned tasks here through subjective impressions in, this is not an appropriate procedure to use for the general case, as we do not have a complete understanding of how the neural networks generalize tasks and if they can find counter-intuitive similarities between tasks. An objective similarity determination is another challenging and unresolved research task in itself.

Another important research question that is still open is the definition of the best way to generalize and transfer learning across tasks. The concept of skill transfer [27] (also referred to as options, macro-actions or compact policies) is promising to DRL. These skills are generalizations of extended sequences of actions to achieve a sub-task and are defined by an option policy, an initiation set and a termination condition. However, extracting partial policies from DQNs is still an open problem.

Transferred to the Atari game playing domain, skills could provide a way to solve more abstract sub-tasks like destroying an enemy, walking through a door or dodging bullets, or any tasks which have repeated occurrences in a variety of games.

In conclusion, TL has shown a great potential to accelerate learning in DRL tasks, but there are still many aspects to be understood before we can formulate the definition of a comprehensive framework for knowledge reuse across DQNs.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

[3] D. Silver, "Lecture Notes in advanced topics in Machine Learning: COMPGI13 (Reinforcement Learning)," 2015, university College London, Computer Science Department.

[4] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: John Wiley & Sons, 2014.

[5] G. Tesauro, "Temporal Difference Learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[6] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via Reinforcement Learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[7] V. Mnih, D. Silver, A. A. Rusu, M. Riedmiller *et al.*, "Human-level control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[8] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[9] M. E. Taylor and P. Stone, "Transfer Learning for Reinforcement Learning domains: A Survey," *Journal of Machine Learning Research (JMLR)*, vol. 10, pp. 1633–1685, 2009.

[10] W. K. Hastings, "Monte Carlo Sampling methods using Markov Chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.

[11] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[12] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[13] R. Bellman, "On the theory of Dynamic Programming," *Proc. National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[14] M. Riedmiller, "Neural Fitted Q Iteration – first experiences with a data efficient Neural Reinforcement Learning method," in *Proc. 16th European Conference on Machine Learning (ECML)*, 2005, pp. 317–328.

[15] M. Hausknecht and P. Stone, "Deep Recurrent Q-learning for Partially Observable MDPs," in *2015 AAAI Fall Symposium Series*, 2015.

[16] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," *arXiv preprint arXiv:1509.06461*, 2015.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with Deep Reinforcement Learning," *arXiv preprint arXiv:1509.02971*, 2015.

[18] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for Deep Reinforcement Learning," *arXiv preprint arXiv:1507.04296*, 2015.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[20] G. Konidaris, I. Scheidwasser, and A. G. Barto, "Transfer in Reinforcement Learning via shared features," *Journal of Machine Learning Research (JMLR)*, vol. 13, no. 1, pp. 1333–1371, 2012.

[21] M. L. Koga, V. Freire, and A. H. Costa, "Stochastic Abstract Policies: Generalizing knowledge to improve Reinforcement Learning," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 77–88, 2015.

[22] S. Gutstein, O. Fuentes, and E. Freudenthal, "Knowledge transfer in Deep Convolutional Neural Nets," *International Journal on Artificial Intelligence Tools (IJAIT)*, vol. 17, no. 03, pp. 555–567, 2008.

[23] E. Parisotto, L. J. Ba, and R. Salakhutdinov, "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning," *Computing Research Repository (CoRR)*, vol. abs/1511.06342, 2015. [Online]. Available: http://arxiv.org/abs/1511.06342

[24] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, "RMSProp and equilibrated adaptive learning rates for non-convex optimization," *arXiv preprint arXiv:1502.04390*, 2015.

[25] D. Kingma and J. Ba, "ADAM: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] F. Fernández and M. Veloso, "Probabilistic Policy Reuse in a Reinforcement Learning agent," in *Proc. 5th Autonomous Agents and Multiagent Systems (AAMAS-06)*, 2006, pp. 720–727.

[27] G. Konidaris and A. G. Barto, "Building Portable Options: Skill Transfer in Reinforcement Learning," in *Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 7, 2007, pp. 895–900.