EPiC Series in Computing
Volume 65, 2019, Pages 163–175
GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence

Multi-task Learning and Catastrophic Forgetting in Continual Reinforcement Learning

João Ribeiro, Francisco S. Melo, and João Dias

INESC-ID/Instituto Superior Técnico
University of Lisbon
Lisbon, Portugal

Abstract

In this paper we investigate two hypothesis regarding the use of deep reinforcement learning in multiple tasks. The first hypothesis is driven by the question of whether a deep reinforcement learning algorithm, trained on two similar tasks, is able to outperform two single-task, individually trained algorithms, by more efficiently learning a new, similar task, that none of the three algorithms has encountered before. The second hypothesis is driven by the question of whether the same multi-task deep RL algorithm, trained on two similar tasks and augmented with elastic weight consolidation (EWC), is able to retain similar performance on the new task, as a similar algorithm without EWC, whilst being able to overcome catastrophic forgetting in the two previous tasks. We show that a multi-task Asynchronous Advantage Actor-Critic (GA3C) algorithm, trained on Space Invaders and Demon Attack, is in fact able to outperform two single-tasks GA3C versions, trained individually for each single-task, when evaluated on a new, third task—namely, Phoenix. We also show that, when training two trained multi-task GA3C algorithms on the third task, if one is augmented with EWC, it is not only able to achieve similar performance on the new task, but also capable of overcoming a substantial amount of catastrophic forgetting on the two previous tasks.[1]

1 Introduction

Recent years have witnessed a significant number of reinforcement learning successes [2, 3, 9, 27, 29, 31], many of which can be credited to the use of deep neural networks [13, 14] in the context of reinforcement learning (RL) problems, in what has become known as *deep reinforcement learning*. Until the advent of deep RL, the success of RL agents in complex domains depended largely on a careful design of features by the agent designers [6, 19, 23, 30]. In contrast, deep RL agents directly can learn from large unprocessed inputs, overcoming the need for the agent designer to specify which features are relevant for the task at hand. For example, deep Q-networks (DQN) [16, 17] successfully learned a number of different tasks in the Atari2600 platform [4] from only raw pixel inputs and reward information, without manually designed features.

However, although the same architecture can be used to learn different tasks—as was the case with the aforementioned DQNs—each different task was learned with a different instance

[1]Our source code and obtained results are publicly available at `https://github.com/jmribeiro/UGP`.

of DQN. When faced with a new task after learning a previous one, the DQN agent would have to learn the second task from scratch, even if both tasks were very similar. Even though there have been some agents that were able to successfully learn multiple tasks [5, 8], the ability to transfer knowledge between tasks remains an important challenge in deep RL.

A second related limitation, perhaps even more severe than the inability to leverage what has been learned in one task to render learning more efficient in a second task, is the inability observed in many RL agents to retain what has been learned when learning a new task—a problem known in machine learning as *catastrophic forgetting* [11, 12, 15].

In this paper, we take a deeper look at the problems of transfer learning and catastrophic forgetting in the context of deep RL. In particular, we investigate the following research questions:

- Compared to single-task learning, can multi-task learning improve an agent's ability to learn a new, similar task? In other words, is an agent trained in multiple tasks more able to transfer learned knowledge from previous tasks to improve the learning of the new one?
- What is the impact of catastrophic forgetting in multi-task learning? Do existing strategies to address catastrophic forgetting—for example, the Elastic Weight Consolidation algorithm (EWC) [12]—allow a deep RL agent to maintain the acquired knowledge from multiple, previous tasks, after learning new ones?

To address the two questions above, we conduct two comparative studies that compare a "multi-task" RL agent against specialized "single-task" RL agents, both in terms of learning a new task and in terms of catastrophic forgetting. Our results suggest that, in general, multi-task learning provides some advantage when learning a novel (related task); moreover, although the multi-task RL agent suffers from the phenomenon of catastrophic forgetting just like the specialized "single-task" agents, our results suggest that the inclusion of a simple mechanism such as EWC can greatly alleviate the impact of such phenomenon.

In our studies we use the *asynchronous advantage actor-critic on GPU* (GA3C) algorithm, proposed by Babaeizadeh et al. [2]. Our choice of algorithm rests on two key facts. First, GA3C is a well-established deep RL algorithm that has successfully been applied to several benchmark domains from the literature. Secondly, the algorithm can easily be extended to accommodate multi-task learning through a simple modification to its core architecture (see Section 4). In our second study, we further extend GA3C to include the *elastic weight consolidation* (EWC) mechanism [12]. We perform all our experiments using domains from the Atari2600 platform, available as OpenAI Gym environments [7].

2 Related Work

In the context of deep RL, several works have explored the problem of transfer learning. For example, Rusu et al. [25] propose *policy distillation*, where the Q-values learned in the target policy are used as regression targets to train a smaller neural network to achieve expert-level performance. In another work, Parisotto et al. [22] propose *actor-mimic*, which pursues a similar approach but where the learner network seeks to "mimic" the policy one or more previously trained expert networks. Finally, Rusu et al. [26] propose *progressive networks*, in which the neural network is progressively augmented—through so-called "columns"—to handle novel tasks, using lateral connections between such columns. One inconvenience of this approach is the scalability, as each column can be seen as a neural network in its own right.

In this paper, we do not use separate networks for different tasks, but instead train a single deep RL agent to simultaneously and continuously perform multiple tasks without forgetting how to perform previously trained ones, in what is known as *continual learning* [21].

We follow the approach of Birck et al. [5], who proposed a modification of the asynchronous advantage actor-critic (A3C) algorithm [18], by having the different actor-learners interact with different environments. With this approach, they seek to test if—by learning two tasks simultaneously—the resulting agent is able to obtain a better policy than two single-task A3C instances, individually trained on each task. Even though we follow their methodology and use two source tasks, an arbitrary number of source tasks can be used.

In our approach, we instead use GA3C [2], a modification of the A3C algorithm. Even though newer asynchronous algorithms have surfaced since the release of the A3C and the GA3C [3, 9, 27], we use the GA3C as the foundation for our experiments, as it can easily be extended to accommodate multi-task learning. The GA3C holds a single global network, removing A3C's need for synchronization. Additionally, it collects the data from the actor-learners differently, in order to take advantage of GPU computation and speedup training. As in the work of Birck et al. [5], we take advantage of the actor-learners in our approach, allowing them to interact with different environments, naturally enabling multi-task learning.

Unlike the aforementioned works, we are not interested in whether the agent trained in multiple tasks achieves better performance than agents specialized in individual tasks. Instead, our goal is to *investigate whether an agent trained on multiple tasks can learn a new task more efficiently.* Our expectation is that, by simultaneously training the agent in several different (although related) tasks, it will acquire more high-level internal representations that may then be useful in the learning of new tasks.

Regarding catastrophic forgetting, several approaches have been explored. We single out three similar algorithms proposed in the literature—namely, EWC [12], LWF [15], and online-EWC [10]. According to Kemker et al. [11], catastrophic forgetting can be observed in sequential learning, whenever a trained model, upon training in a new task, moves abruptly in the space of parameters, effectively "forgetting" the original task. To avoid such abrupt changes, when training the network for the second task, these algorithms add an extra term to the loss function of the network that penalizes deviations from the parameters learned for the first task. Following Rusu et al. [26], Schwarz et al. [28] introduce the "Progress & Compress" framework, where each time a new task is learned, a new "progressive active column" is created. Afterwards, using the policy distillation method from Rusu et al. [25] combined with the EWC algorithm [12], the parameters are then "compressed" into a single "knowledge base", updated every time a new task is learnt, whilst preserving older knowledge. As argued by Kirkpatrick et al. [12], the EWC is scalable and effective, both in classification and reinforcement learning tasks. In this work, we also *investigate whether the use of an approach such as EWC can mitigate catastrophic forgetting while still leveraging the potential advantages of multi-task learning.*

3 Background

We now briefly review the main concepts of RL before moving on to describe our approach.

In reinforcement learning, an agent interacts with an environment in consecutive time steps. At each time step t, the agent observes the state S_t and selects an action A_t, receiving a reward $r(S_t, A_t) \in \mathbb{R}$ and transitioning to state S_{t+1}. We denote by $\mathcal{S}$ the set of all states, and by $\mathcal{A}$ the set of all actions. The goal of the agent is to select the actions that maximize its *expected return*, where the return at time step t is the quantity

$$R_t = \sum_{\tau=0}^{\infty} \gamma^\tau r(S_{t+\tau}, A_{t+\tau}),$$

where $\gamma \in [0, 1)$ is a discount factor. The expected return at time step t, $\mathbb{E}[R_t]$, is a function of the state S_t and of the particular process by which the actions are selected. A *policy* π is a state-dependent distribution over actions, where $\pi(a \mid s)$ denotes the probability of selecting action a in state $s \in \mathcal{S}$ according to π. We write

$$V^\pi(s) = \mathbb{E}\left[R_t \mid S_t = s, A_\tau \sim \pi(S_\tau), \tau \geq t\right],$$
$$Q^\pi(s, a) = \mathbb{E}\left[R_t \mid S_t = s, A_t = a, A_\tau \sim \pi(S_\tau), \tau > t\right].$$

A policy π^* is optimal if $V^{\pi^*}(s) \geq V^\pi(s)$ for all π and all $s \in \mathcal{S}$.

Actor-critic methods approximate π^* by considering a parameterized family of policies $\{\pi_\theta\}$ and updating the parameter θ, at each step t, along the gradient of $\mathbb{E}[R_t]$. Given a trajectory obtained with the current policy π_θ, actor-critic methods perform stochastic gradient ascent, using an update of the form

$$\theta \leftarrow \theta + \alpha \nabla \log \pi_\theta(S_t)(R_t - V^\pi(S_t)).$$

The component performing the policy updates is called the *actor*, and the estimation of V^π is usually performed independently by a *critic*. The quantity $R_t - V^\pi(S_t)$ is an estimate for the *advantage function* $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s, a)$—for which reason several actor-critic algorithms are referred as *advantage actor-critic* algorithms—such as A3C and GA3C.

4 Material and Methods

In order to test both our hypotheses, we rely on a multi-task version of the GA3C algorithm augmented with EWC. Since the GA3C algorithm has not been designed for either multi-task learning, transfer learning or continual learning, we extend the original GA3C to include:

- The hybrid architecture proposed in [5], in which the actor-learners interact with different environments, enabling multi-task learning;
- The mid-level feature transfer approach from [20], in which the input layers of a deep convolutional neural network are shared across tasks and only the output layers are task-specific;
- The EWC algorithm, by modifying the loss function used to train the actor-critic network with a term that penalizes deviations from parameters learned in previous tasks.

Additionally, we also introduce a module called *environment handler*, which provides an interface between actor-learners and the environment instances, abstracting the input data into a state s_t and a reward signal r_t. Figure 1 summarizes the overall architecture.

The algorithm is implemented in Python 3, and the model setup using TensorFlow [1], running all computational graph forward and backward propagation routines using GPU.

Environment Handlers.

An *environment handler* is an abstract module responsible for handling the interaction between an actor-learner and an instance of an environment. By abstracting this interaction, we ensure our agent's compatibility with other platforms other than the Atari2600. Even though we open-source only an implementation for the Atari2600 platform, environment handlers enable further studies for multi-task learning with tasks from multiple platforms.

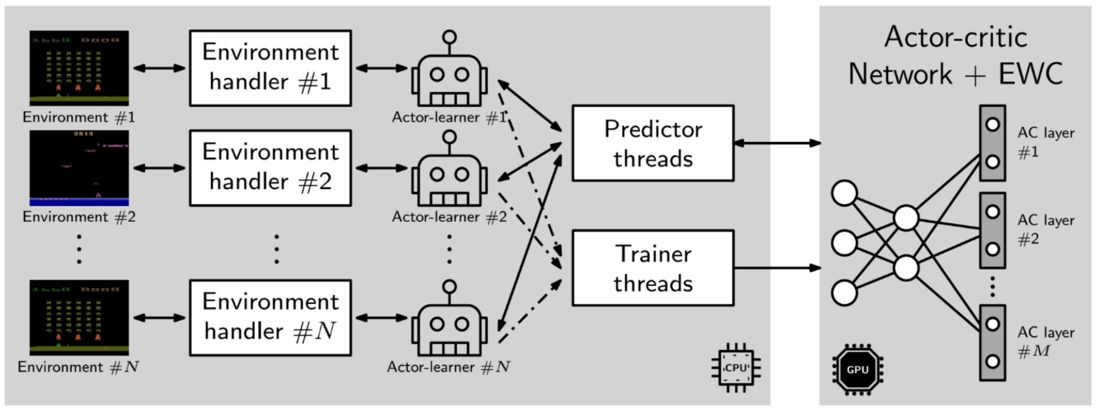


Figure 1: Summarized architecture for the multi-task, EWC-enhanced GA3C. Several actor-learners interact asynchronously with multiple (possibly different) environment instances. As with the original GA3C architecture, each actor-learner submits prediction requests to the actor-critic network using trainer threads. See main text for further network through predictor threads and feed experience batches details.

In a given time step t, an actor-learner starts by requesting the current state s_t and reward r_t from its environment handler. The environment handler builds the state s_t stacking four observations (frames)—o_t , o_{t-1}, o_{t-2} and o_{t-3}. By stacking four consecutive frames it is possible to capture the temporal context of the current state of the game. This is required since all environments used are history-dependent—i.e. the current observation may have different interpretations, depending on previous observations. The environment handler pre-processes each observation o_t (corresponding to a $210 \times 160 \times 3$ pixel image for the Atari2600 environment), by resizing it as a grey-scale 84×84 image.[2] The resulting $4 \times 84 \times 84$ tensor corresponds to the state s_t sent to the actor-learner. After obtaining the current state s_t from the environment handler, the actor-learner returns an action a_t, which the environment handler executes upon the environment. After one step, the next state, s_{t+1}, and reward, r_t, are sent to the actor-learner.

Loss and back-propagation.

The loss function used to train the actor-critic network considers the different environments that the agent-learners interact with. We setup a specific loss function, $L_{\mathcal{E}_k}(\theta)$, for each environment $\mathcal{E}_k$. When training using a multi-environment batch, the optimizer uses the corresponding data-points to compute the environment's specific loss, one environment at the time.

The parameters of the network can be split into 3 sets: $\{\theta_i\}$, $\{\theta_\pi\}$, and $\{\theta_V\}$. The first set, $\{\theta_i\}$, represents the shared parameters for the input network. These parameters are shared by all environments and are updated when back-propagating data-points from all environments. The second set, $\{\theta_\pi\}$, represents the parameters from the actor layers. The parameters associated with each environment are only updated by back-propagating data-points from that specific environment. Finally, the third set, $\{\theta_V\}$, represents the parameters from the critic layers. As with the actor parameters, these are only updated when back-propagating data-points from that specific environment. For each environment, the loss function $L_{\mathcal{E}_k}(\theta)$ combines a loss term

[2] We use the same pre-processing method of Babaeizadeh et al. [2] which, in turn, was based on the pre-processing method of the DQN [17].

Table 1: Environments from OpenAI Gym used in the experiments.

Short name	OpenAI Gym Environment Name
SpaceInvaders	`SpaceInvadersDeterministic-v4`
DemonAttack	`DemonAttackDeterministic-v4`
Phoenix	`PhoenixDeterministic-v4`

associated with the critic and loss term associated with the actor [2, 18].

Finally, we also introduce EWC to tackle catastrophic forgetting. Suppose that the network has been trained in multiple environments $\mathcal{E}_1, \ldots, \mathcal{E}_M$ and converged to the set of input parameters $\{\theta_i^*\}$. Then, given a new environment, $\mathcal{E}_{M+1}$, we add an additional term to the loss, $L_{\mathcal{E}_{M+1}}(\theta)$, given by

$$L_{\text{EWC}}(\theta_i) = \lambda(\theta_i - \theta_i^*)^\top \mathbf{F}(\theta_i - \theta_i^*),$$

where $\mathbf{F}$ is the Fisher information matrix and can be computed using samples from environments $\mathcal{E}_1, \ldots, \mathcal{E}_M$ [12]. The parameter λ allows to specify how important are the old parameters, when learning the new task. We maintain the RMSProp optimizer [24], using similar parameters to those used by Babaeizadeh et al. [2].

Domains

We resort to the OpenAI Gym toolkit [7], where available environments from the Atari2600 platform were used as tasks [4]. When interacting with an environment, the agent only has access to the game screen (pixels) and current score. We use three different environments consisting of "bottom-up" shooting games. Table 1 summarizes the used OpenAI Gym environments.[3]

We then created four agents, using our multi-task GA3C architecture:

SpaceNet: GA3C that learned how to play Space Invaders. 24 actor-learners trained on `SpaceInvaders` for a total of 150,000 episodes.

DemonNet: GA3C that learned how to play Demon Attack. 24 actor-learners trained on `DemonAttack` for a total of 150,000 episodes.

PhoenixNet: GA3C that learned how to play Phoenix. 24 actor-learners trained on `Phoenix` for a total of 150,000 episodes.

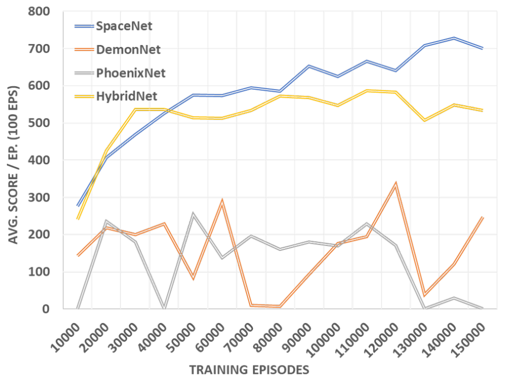
HybridNet: Multi-task GA3C that learned how to play Space Invaders and Demon Attack. 12 actor-learners asynchronously trained on SpaceInvaders and `DemonAttack`, summing up for a total of 150,000 multi-task episodes.[4].

5 Results

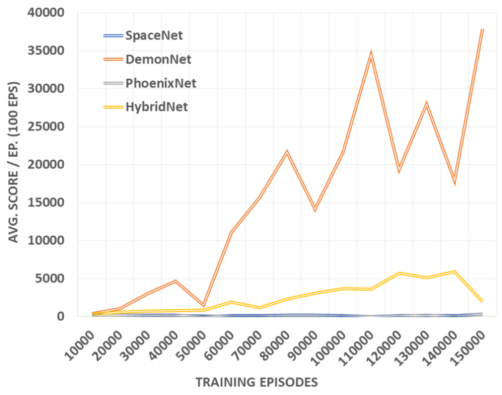
This section describes in detail the results obtained in our two studies.

[3]We resort to the deterministic versions of the environments in order to speed up the training.

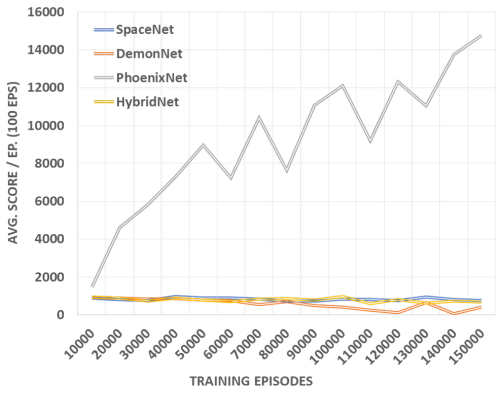
[4]We do not require each task to have the same number of episodes, since episodes from Space Invaders are quicker than episodes from DemonAttack



(a) Space invaders.



(b) Demon Attack.



(c) Phoenix.

Figure 2: Learning performance of the four agents in the different environments considered. The results were obtained by evaluating each agent during 100 episodes every $10,000$ training episodes. As expected, the best perform"single-task" agent attains the best performance.

5.1 Impact of Multi-Task Learning for New Tasks

We start by investigating whether learning multiple tasks simultaneously improves an agent's ability to when learn a new (similar) one. To investigate this hypothesis, we compare the learning performance of HybridNet in a new environment, Phoenix, against the two single-task counterparts (the SpaceNet and the DemonNet). Figure 2 depicts the comparative performance of all agents in all three environments.

As expected, each "single-task" agent outperforms all the others in its corresponding task. The HybridNet, which simultaneously trained for Space Invaders and Demon Attack, was able to obtain a relatively similar performance to the SpaceNet on Space Invaders (Fig. 2(a)), but was unable to match the performance of the DemonNet on Demon Attack (Fig. 2(b)). The difference in performance in Demon Attack may be explained by the discrepancy in the number of training episodes between the two tasks—Demon Attack episodes took longer to finish than episodes from Space Invaders, which means that, in practice, HybridNet played significantly less episodes of Demon Attack than Space Invaders. Finally, the PhoenixNet, which trained

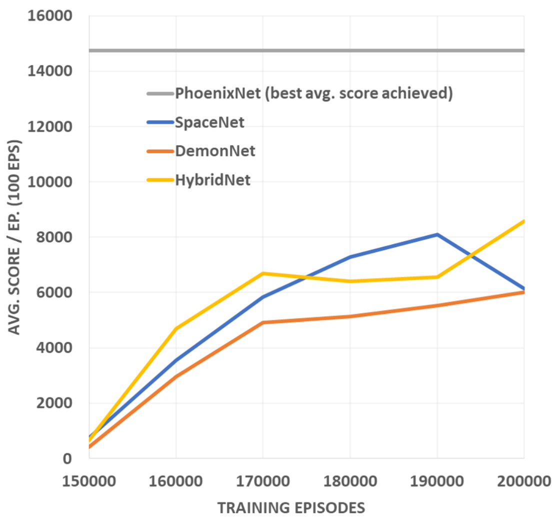


Figure 3: Evaluation of the SpaceNet, DemonNet and HybridNet agents, as they trained on Phoenix for 50, 000 additional episodes. The PhoenixNet performance after training on Phoenix for 150, 000 episodes is provided for reference.

for 150, 000 episodes of Phoenix, provides a baseline for the evaluation of the other agents on that environment. We should also reinforce the fact leading to the substantial differences in performance between the HybridNet and the SpaceNet in Space Invaders and the HybridNet and the DemonNet in Demon Attack. When training the HybridNet on both tasks, for a total of 150000 shared episodes, we do not require each task to have the same number of training episodes, since episodes from Space Invaders are quicker than episodes from DemonAttack. Otherwise, the HybridNet would have a higher number of training timesteps for Demon Attack than Space Invaders, resulting in a very task-biased training.

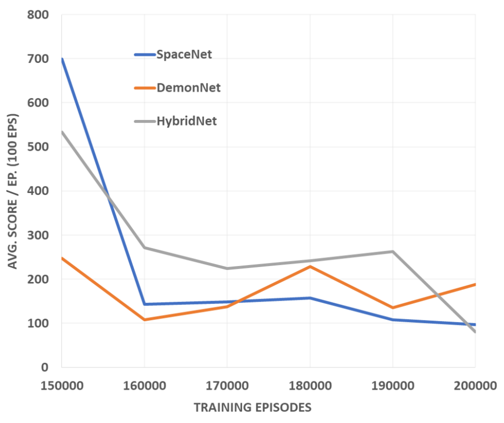
We then allowed all three agents (HybridNet, SpaceNet and DemonNet) to learn a new task—Phoenix. We trained all three agents on this new task for an additional 50, 000 episodes. The obtained results are illustrated in Fig. 3.

The worst performing agent was the DemonNet, with an average score per episode of 6, 006.3, followed by the DemonNet, with a 6, 136.6. At episode 200,000, the HybridNet outperformed the other two, with an average score of 8, 587.3. Even though these results are taken from a single instance of the agent, they suggest that learning from multiple tasks may provide an advantage when learning a new task. It should be also be noted, however, that during learning, the SpaceNet temporarily outperformed HybridNet, at episode 190,000. Since only one instance was trained per agent, this phenomenon may not be statistically significant. However, it is surely an aspect we wish to study further in the future when conducting more extensive research related to the impact of multi-task learning in continuous learning.

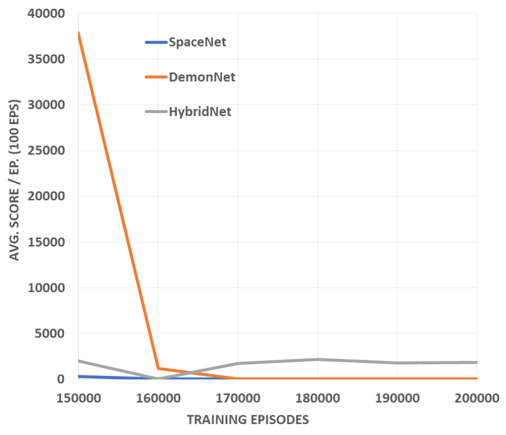
5.2 Catastrophic Forgetting

We next considered the problem of catastrophic forgetting. In particular, we took the three agents featured in Fig. 3 and evaluated their the performance in the original tasks. Figure 4 depicts the performance of all three agents.

As expected, after training on Phoenix for 50, 000 episodes, all agents catastrophically forgot

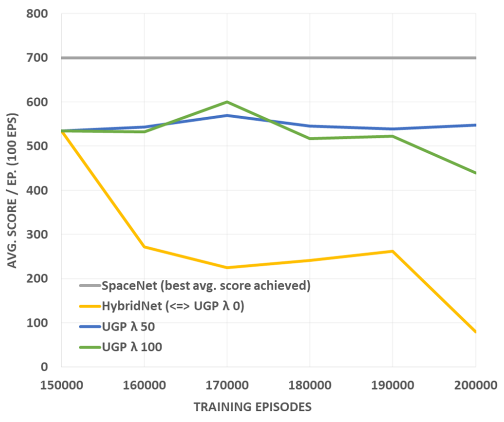


(a) Space invaders.

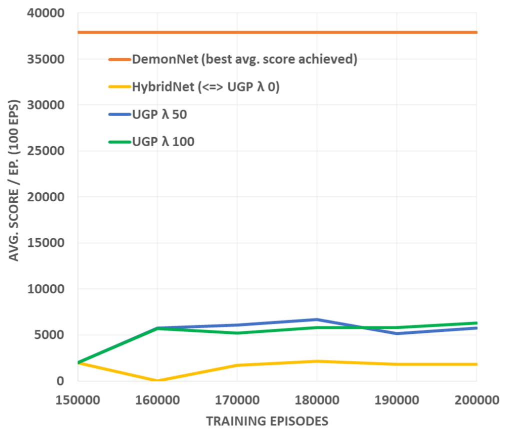


(b) Demon Attack.

Figure 4: Evaluation of SpaceNet, DemonNet and HybridNet on the two original tasks (Space Invaders and Demon Attack) after training on Phoenix for 50,000 additional episodes. All agents exhibit catastrophic forgetting, failing to achieve a performance that can be compared with that from Fig. 2.



(a) Space invaders.



(b) Demon Attack.

Figure 5: Performance of HybridNet (equivalent to UGP with $\lambda = 0.0$), UGP with $\lambda = 50.0$ and UGP with $\lambda = 100.0$, evaluated on both Space Invaders and Demon Attack, after training 50,000 additional episodes on Phoenix.

how to perform their original tasks. We thus compare the three agents with a version of the "multi-task" GA3C HybridNet augmented with EWC. We call the resulting agent the Universal Game Player (UGP), simply for symbolic purposes. We repeated the exact same experiments with three instances of UGP with different values for λ. In particular, we use $\lambda = 0.0$ (in which case UGP is just HybridNet), $\lambda = 50.0$ and $\lambda = 100.0$. Figure 5 depict the performance of UGP.

For $\lambda \in \{50, 100\}$ we can observe that, in fact, UGP was able to overcome catastrophic forgetting to a large extent. In the case of the first source task, Space Invaders, all three agents

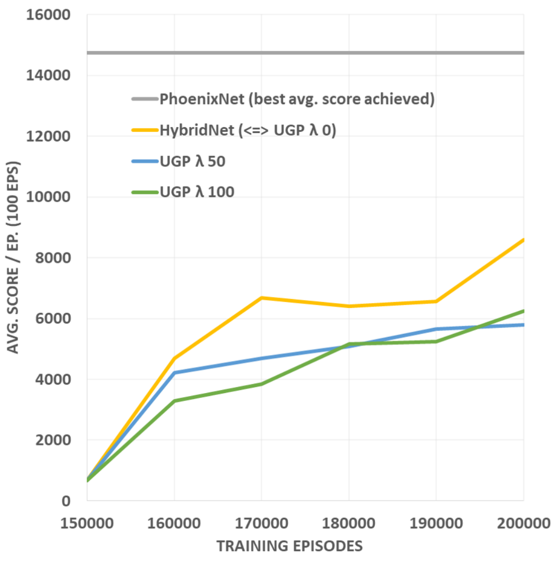


Figure 6: The HybridNet agent (equivalent to an UGP agent without EWC or with EWC λ=0), UGP agent with λ=50.0 and UGP agent with λ=100.0, evaluated on Phoenix, after training 50,000 additional episodes on the environment.

achieved an average score of 533.95. After $50,000$ additional episodes on Phoenix, the HybridNet achieved an score of 80 in Space Invaders, while the UGP with $\lambda = 50.0$ achieved a score of 547.5 and the UGP with $\lambda = 100$ achieved a score of 439.4. The differences in performance observed in Demon Attack were also significant, although the final performance of all UGP agents differed more significantly in this second task

It remains only to assess whether the inclusion of EWC impacted UGP's ability to learn a new task. Figure 6 shows the result of training UGP in Phoenix.

After training during $50,000$ episodes on Phoenix, the HybridNet was able to achieve an average score $8,587.3$. UGP with $\lambda = 50$ achieved an average score of $5,791.1$, while UGP with $\lambda = 100$ achieved a score of 6243.4. As expected, one side-effect of EWC algorithm is the decrease in performance in the new task, since the agent is "less willing" to move away from the parameters learned in previous tasks. Nevertheless, UGP with $\lambda = 100$ was still able to outperform the single-task networks in learning a new task, confirming that, in fact, multi-task learning can indeed help to learn new tasks, even with the EWC mechanism in place.

6 Conclusion

In this work we investigated whether (i) multi-task learning can help in learning new, related tasks and (ii) whether this learning comes at a cost of catastrophic forgetting. Our results show that a modified GA3C algorithm, capable of multi-task learning, can outperform other agents when learning a new, similar task. We also show that, by accommodating the EWC mechanism, the multi-task GA3C is effectively able to mitigate the effects of catastrophic forgetting.

In the future, we plan to study a similar hypothesis, by comparing the effects newer algorithms such as the online-EWC [10] and the LWF [15], in both single and multi-task learning settings, testing whether multi-task learning is useful in alleviating catastrophic forgetting.

Thanks to our novel contribution—the use of environment handlers in our architecture—we are now able to apply this methodology to multiple platforms (other than the Atari2600), enabling us to plan and conduct more extensive tests regarding continual learning [21]. The use of environment handlers in our architecture renders such extension straightforward, as it only requires that the platform provides multi-dimensional array of pixels, which can be resized and pre-processed as desired, and a reinforcement measure, which the OpenAI Gym always provides.

Acknowledgments

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2019, and this material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-19-1-0020.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement learning through asynchronous advantage actor-critic on a GPU. *arXiv preprint arXiv:1611.06256*, 2016.

[3] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[4] M. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, 47:253–279, 2013.

[5] M. Birck, U. Corrêa, P. Ballester, V. Andersson, and R. Araujo. Multi-Task reinforcement learning: An hybrid A3C domain approach. 01 2017.

[6] J. Boyan and A. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems, 7*, pages 369–376, 1994.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[8] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[9] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[10] Ferenc Huszár. On quadratic penalties in elastic weight consolidation. *arXiv preprint arXiv:1712.03847*, 2017.

[11] Ronald Kemker, Angelina Abitino, Marc McClure, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*, 2017.

[12] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[13] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256. IEEE, 2010.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

[15] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018.

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

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

[18] V. Mnih, A.P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1928–1937, 2016.

[19] T. Nguyen, Z. Li, T. Silander, and T. Leong. Online feature selection for model-based reinforcement learning. In *Proc. 30th Int. Conf. Machine Learning*, pages 498–506, 2013.

[20] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[21] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

[22] E. Parisotto, J. Ba, and R. Salakhutdinov. Actor-Mimic: Deep multi-task and transfer reinforcement learning. In *Proc. 6th Int. Conf. Learning Representations*, 2016.

[23] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proc. 25th Int. Conf. Machine Learning*, pages 752–759, 2008.

[24] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[25] A. Rusu, S. Colmenarejo, C. Gülçehre, G. Desjardins, J. Kickpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. In *Proceedings of the 6th International Conference on Learning Representations*, 2016.

[26] A. Rusu, N. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. In *arXiv preprint arXiv:1606.04671*, 2016.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[28] Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

[29] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[30] R. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems, 8*, pages 1038–1044, 1995.

[31] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.