Analysis of Multi-Tasking Reinforcement Learning

Chris Nalty (cnaltypga@gmail.com), Makai Freeman (mfreeman@terpmail.umd.edu), Jill Granados (granaj91@gmail.com), Michael Stephanus (michaelstephanus82@yahoo.com)

Abstract

Training a network to complete a process will have different performance based on whether the model is using a single task for training or multiple. Mnih et al researched a similar question using Deep Q-learning to train a model to play Atari 2600 games. Hasselt et al developed a Double Deep Q-learning network based on the Deep Q-Learning Network to mitigate the overestimation of values of actions in an attempt to have the network find a more general solution. Combining these methods we used a Double Deep Q-Learning Network to train a model to play Atari 2600 games either as a single task or part of a multi task process to determine how much the multi-task model will outperform the single-task and when the performance then decreases. Our findings suggest that there is little to no performance increase when training multiple tasks versus a single task with the difference becoming greater as more tasks are added.

1. Introduction

Reinforcement learning has been shown to be very useful for learning new tasks. Each sequence of actions taken during reinforcement learning leads to either a positive or negative reward whether the agents are on track to meeting their goal or not. This is very similar to a human who is learning something new for the first time and has no related experiences they can draw upon to help them learn. However, with many new skills people learn, they can use seemingly irrelevant information from past skills to help them learn new ones. Ruder provides a perfect example of this by referencing the movie *The Karate Kid* in his overview of Multi-task learning: “In the movie, sensei Mr Miyagi teaches the karate kid seemingly unrelated tasks such as sanding the floor and waxing a car. In hindsight, these, however, turn out to equip him with invaluable skills that are relevant for learning karate”. It’s clear that it is useful to incorporate multiple different tasks during reinforcement learning to hopefully help the model master a new task.

Learning new skills can be made a little easier when there is other information that can be used to help in learning that new skill; however, if there is too much information to choose from, it can become difficult when trying to decide which pieces of information will be most useful for that particular new skill. We believe that by using a combination of multi-task learning and reinforcement learning to train a model on a few tasks, we’ll achieve a model that outperforms one that is trained on only a single task. However, as more and more tasks are added, the model’s performance will decrease.

Our goal is to determine how the performance of a model that is trained on a single task differs to one that is trained on multiple and at what point does the performance begin to degrade. In order to accomplish our goal, we will be training our model on Atari games similarly to Mnih et al where the agent observes the pixel of the current screen of the game being played to determine which action to take. Mnih et al uses his deep learning model, Deep Q-learning (DQN), to train his model which produced state of the art results. However, we will be using a Double DQN

which is introduced by Hasselt et al as an improvement to DQN. First, the model will be trained on each game separately. Then we will incorporate multi-task learning by swapping out games after a certain number of frames during training. One model will be trained on only a few similar games and the other model will be trained on all of the games.
First, we dive deeper into the background behind the models we will use. Secondly, we explain our models along with how they will be analyzed. Next, we show the performance of each of our models and, lastly, summarize our findings while also providing opportunities for future work.

2. Related Work

In 2015, Mnih et al. introduced a new deep learning model for reinforcement learning, Deep Q Network (DQN), which combines Q-learning with a deep neural network. The motivation behind this new learning model was so that it was possible to efficiently train a model to play several different Atari 2600 games without providing game specific information to the model. Their implementation makes use of a target network which is updated after every kth frame. The implementation also used an experience replay where they store the agent's experience after each timestep into a data-set pooled over many episodes into a replay memory and then samples uniformly at random from the replay memory when performing updates(Mnih et al. 2015). They trained this model on seven different Atari 2600 games (Beam Rider, Breakout, Enduro, Pong, Q*bert, Sequest, Space Invaders) resulting in the model outperforming other learning methods such as Sarsa and Contingency. It even outperformed expert human players in Breakout, Enduro, and Pong.
Although DQN has excellent performance, it tends to overestimate values of the actions which can lead to unstable learning. This is due to the max operator in the target in DQN using the same values to select and evaluate the action. Hasselt et al. addresses this overestimation with his Double DQN algorithm which decomposes the max operation in the target into action selection and action evaluation. The greedy policy is evaluated according to the online network, but uses the DQN target network to estimate its value (Hasselt et al., 2015). This small change significantly reduced overestimations and created a much more stable algorithm. The Double DQN was able to find better policies; therefore, obtaining state-of-the results on a wider array of games on Atari 2600. Another advantage of Double DQN that Hasselt et al. showed was that it is capable of finding general solutions which makes the algorithm much more robust. This will be useful for us when we want to train our model on multiple different games at once to simulate multi-tasking.

Learning multiple tasks can be useful for mastering a new task. Bits and pieces of experiences from past completed tasks become the building blocks for learning this new task. In a machine learning standpoint, this creates the opportunity for a model to generalize well since the model can use information from past unrelated tasks to perform a new one. This is the general idea of multi-task learning (MTL). Ruder explains in his overview of MTL that the most common way to perform MLT in deep neural networks, and the method that we will be using, is hard parameter sharing of hidden layers. In hard parameter sharing, the hidden layers are shared between all tasks while keeping several task-specific output layers (Ruder, 2017). In essence, the

network is spreading out it's experiences from performing other tasks so if the information is useful for performing an unrelated task. This prevents the risk of overfitting since the model is learning many tasks simultaneously so it has to try to encapsulate all of the tasks which makes it less likely to overfit.

3. Methods

For our experiments we use the Double DQN algorithm (Hasselt et al., 2015). This method is based on the classical Q-Learning algorithm, where the goal of the agent is to learn the optimal action-value function, denoted as $Q^*(s, a)$. This function takes a state, s, and an action, a, to determine its expected discounted reward, $R_t = \Sigma^T_{t'=t} \gamma^{t'-t} r_{t'}$. Where T is the time step the game terminates, γ is some discount factor, and $r_{t'}$ is the reward at time t'. So, $Q^*(s, a) = \max_\pi \mathbb{E}[R_t \mid s_t = s, a_t = a, \pi]$ where π is some policy that takes in a state, s and maps it to an action a. In Deep Q Learning the function $Q(s, a)$ is approximated by a neural network to create a DQN. In the Double DQN version this is slightly altered to have two Q networks with the same architecture, a policy network and a target network. The policy network is used to choose actions with an ε-greedy strategy, and the target network is a frozen network that is replaced by the policy network every τ steps. The loss of the policy network is smooth L1 loss, where the hypothesis $Q^p(s, a)$, and the target value is $Q^-(s, a) \cdot \gamma + r_t$, where Q^p is the policy network, and Q^- is the target network.

3.1 Algorithm and Design Decisions

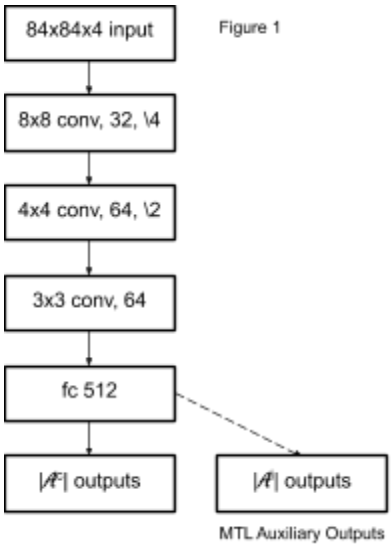

Figure 1

For our DQN we used the same architecture and image preparation as in Hasselt et al. (Figure 1). We first downsize the game images to 110x84 and crop the 84x84 region that best captures the game region. Next the images are converted to grayscale, and the last four of these game images are stacked channel-wise to create the input for the network. The network then consists of five layers. The convolutional layers consist of an 8x8 convolution with 32 filters and a stride of 4, a 4x4 convolution with 64 filters and a stride of 2, and finally a 3x3 convolution with 64 filters and a stride of 1. The output of this final layer is flattened and given to a fully connected layer with 512 hidden units and finally an output layer with 2 outputs. Each layer prior to the output uses a ReLU activation. We also use the same frameskip of four, where only every fourth frame is considered a state. For each of the three frames in between states the last action taken by the network is repeated. For multi-task learning there is an output layer unique to each game being learned. All of the previous layers are shared across all games. We chose to only have the output layer unique to each game, because the network we are using is very shallow, and we want to make sure the network is learning all games simultaneously and not individually.

learning rate (α)	1e-4*
discount factor (γ)	0.99
target update (τ)	10,000
training frequency	4
batch size	32
initial ε	1
final ε	0.1
replay memory size	150,000*

* values different from Hasselt et al.
Figure 2

In our experiments, we attempt to keep the hyperparameters as similar as possible to Hassel et al. Below (Figure 2) is a table listing the hyperparameters used, with non matching ones marked. Firstly we chose a much smaller replay memory size of 150,000 frames as opposed to 1,000,000 in the Mnih et al. This choice was not for optimization purposes, it was the largest reasonable size we could use on the available hardware without significantly increasing training time with disk read and write operations. The effects of this will be discussed in the results section. Secondly, we chose a slightly smaller learning rate, to help compensate for the smaller memory. We found larger learning rates caused bouncing gradients on some games. For fairness during multi-task learning the learning rate of 1e-4 allowed for the most smooth training on all games, avoiding bouncing gradients.

3.2 Experimental procedures

For our experiments we train on four games from the Atari 2600 environment Breakout, Pong, Beam Rider, and Seaquest. We train networks in the following ways: Each game with its own network, Breakout and Pong sharing a network, and all four games sharing a network. The shared networks are configured for multi-task learning as described in section 3.1. The reason these games are chosen is based on their similarity. We chose two games that are very similar, Breakout and Pong. In each of these games winning involves bouncing a ball off of a paddle. Beam Rider is a game that has somewhat similar features to breakout. You must control a character at the bottom on the screen and avoid incoming objects. Finally Seaquest is chosen as a stress test for multi-task learning, as it does not have many similarities to any of the other games.
For all setups we initialized the replay memory with 50,000 frames where random actions were chosen on each game being learned. For all setups each game was trained for 8 million frames, which is 2 million training steps. All agents used an ε-greedy strategy that started with $\varepsilon = 1$ and was decreased by 0.01 every 10,000 frames, until $\varepsilon = 0.1$ at frame 900,000. For multi-task learning each game had its own ε value that was decreased respective to that game's frame count. This frame count is much lower than the original paper, which trains for 20 million frames, or 5 million training steps. Our value was chosen due to time constraints, as some of these models take multiple days to train. As will be seen in the results section, our hyperparameter selection sacrifices some final maximum score when compared to Hasselt et al. The models appear able to train to their maximum potential in the training time alloted with our hyperparameter selection. During multi-task learning we used a scheme of alternating training of each game. We experimented with two different switching frequencies on the two task agent, every 10,000 frames and every 100,000 frames. We found that the larger switch interval caused extreme fluctuations in training accuracy, so the smaller value of 10,000 was used for the four task agent.

3.3 Analysis Method

Our analysis will mainly focus on the agents scores over the last 100 episodes of the game being played. Pictured in Figure 3 on the left plot are the raw scores of the agent in blue, and the

average score of the last 100 episodes in red for the game Breakout. The raw scores are extremely unstable throughout training. The right graph shows how the agents policy network evaluates the first state on each episode, which is much more stable than the agents score. This leads us to believe much of the fluctuation is a result of the ε-greedy strategy used by the agent. We'll also talk about the maximum score the agent achieved as a measure, since this is the ultimate goal of training an agent.

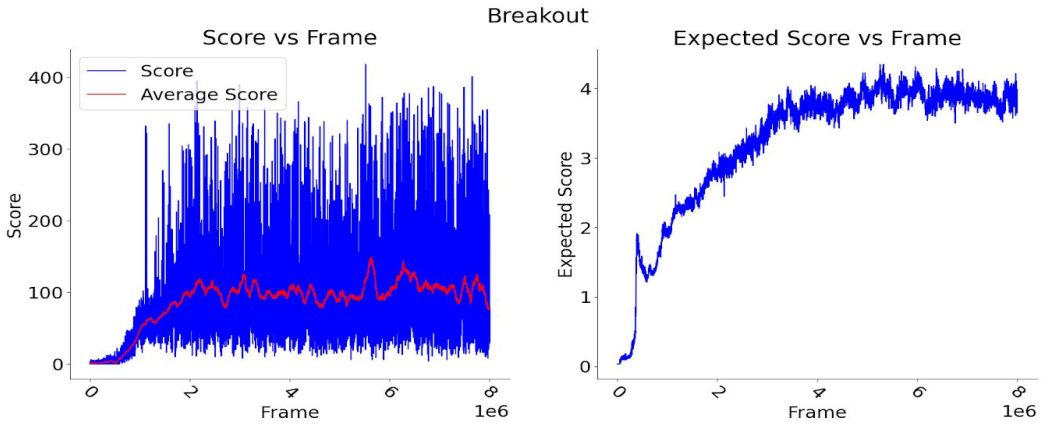

Figure 3

4. Results

During the experiments two seperate hyperparameters were tested on a two task network to determine how to train the four task network. For the first run the game being trained was alternated every 100,000 frames. This caused large instabilities in the training of the network (Figure 4). To counteract this alternating every 10,000 frames was tested before moving on to the four task network to give it the best chance to learn. It is clear that lowering the number of frames between switching games greatly improved overall performance and stability. This allowed the two task network to learn at almost an identical rate to the single task networks. Because of this we chose to alternate training every 10,000 frames for four task learning and forgo the 100,000 frame test.

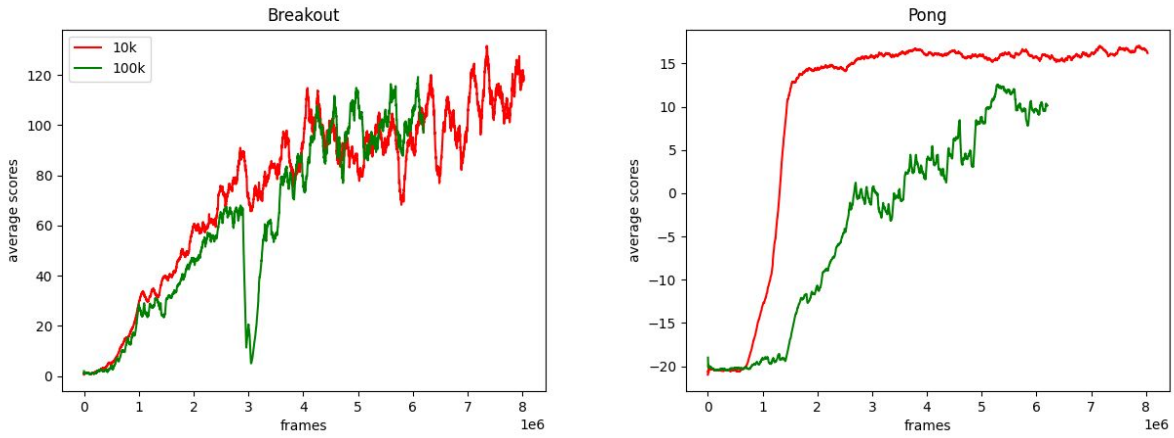

Figure 4: Graphs comparing the average scores over the last 100 frames for two MTL learning strategies. The red line shows a strategy that alternates the training game every 10,000 episodes, the green shows alternating every 100,000 episodes

Game	1 Task Max	2 Tasks Max	4 Tasks Max	1 Task Best Average	2 Tasks Best Average	4 Task Best Average
Breakout	418.0	391.0	17.0	148.4	131.6	9.6
Pong	21.0	21.0	-3.0	17.7	17.04	-14.2
Beam Rider	8520.0	-	5732.0	4700.5	-	2425.8
Seaquest	6660.0	-	1532.0	3144.7		2160.0

Figure 5: Chart of best scores and best average of last 100 frames by game and agent

The experiments show that a two task agent is able to learn nearly as well as the agents trained on individual tasks. In Breakout and Pong the two task agent reached 89% and 98% respectively of the individual agent's best average score (Figure 5). The rate at which a two task agent learned the tasks was also similar to the individual agents, learning slightly slower at breakout and at nearly the same rate at Pong. Two games was clearly the limit to what the shallow network used could handle. Figure 6 shows that the 4 task agent struggled to learn any of the games. The agent did not learn to play Breakout significantly better than an agent selecting random actions. It took over 3 million frames to do better than random on Pong, whereas the single task agent maximizes its reward by frame 1 million. The quality of the policies are also much weaker, in the best case reaching 68% of the individual agent's best average on Seaquest, and worst case only performing 6% as well on Breakout. Possibly remedies to this will be discussed in the future work section.

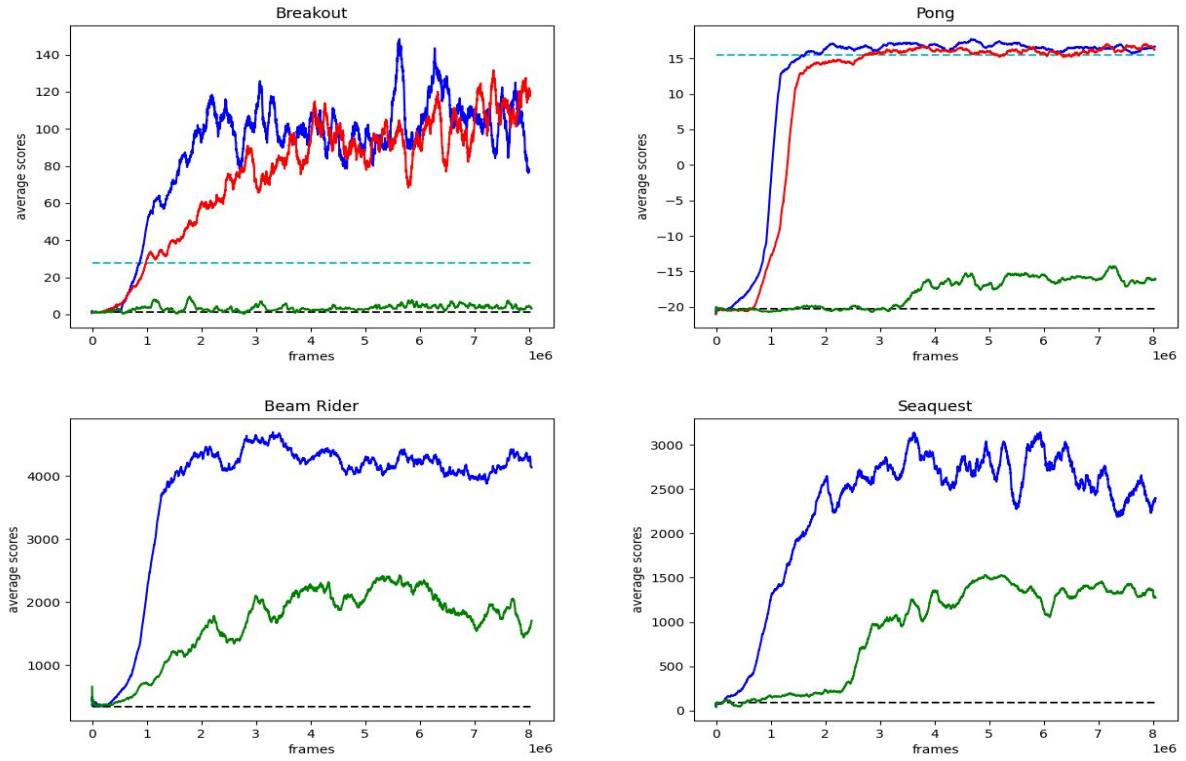

Figure 6: These graphs show the average scores vs frames. The blue line represents the training of a network with only 1 game. The red line represents the training of a network with 2 games (breakout and pong). The green line represents the training of a

network with 4 games. The scores for random agents and humans are denoted by dashed lines. Beam Rider and Sequest the human score is omitted, because it is much greater than the agent's.

5. Discussion

In summary we found that training a network on two games achieved almost identical results to training on a single game, in the case of Breakout and Pong. There were only minor differences in the amount of time taken to achieve the optimal policy given the hyperparameters. Training four games at the same time was found to significantly reduce the quality of the learned policies. The agent playing four games barely achieved a better score than random on breakout and pong, and achieved slightly less than half the scores of the single agents on Beam Rider and Seaquest.

5.1 Comparison to Past work

Our agents generally achieve scores from 33% to 85% of those trained in Hasselt et al. with single and two task networks. These results are achieved using a replay memory of only 15% the size of their work. This shows there is a significantly larger cost in memory requirement compared to the increase in agent performance, but can be game dependent. For games that require a complex strategy for large scores, such as Breakout, the score our agents obtain is significantly worse with this smaller memory. Hasselt et al.'s Breakout agent learned a strategy where it tunnels through the bricks on the edge and then traps the ball behind the bricks, allowing it to take large amounts of blocks without needing to catch the ball. Our agent is only able to achieve this a small portion of the time. With a network learning two tasks we are still able to learn this complex task at the same rate, however when we learn all four tasks at once our agent fails to learn this task at all. In fact the four task agent's scores only range from 2% to 20% of Hasselt et al.'s work.

5.2 Future Work

Given the results of our experiments we see a clear next step for future work. We would like to try alternative network architectures to increase the quality of the multi-task agent's learned policies. As stated prior, the network used was very shallow matching Hasselt et al. We propose an architecture similar to the work by Mujika, a residual network consisting of seven convolutional layers with three residual connections, and two fully connected layers. We would again use hard parameter sharing for all convolutional layers. However, we would like to allow each game to have all fully connected layers unique to each game. Mujika uses a recurrent network and Predictive Reinforcement Learning instead of DQN to train on three games; We would like to expand further on their work by continuing use of DQN and training on additional games to stress test the capacity of the network.

In addition to an improved network we would like to use a learning strategy that involves selecting frames from each game for all training batches as opposed to alternating training between games to help stabilize learning further. If better hardware is available it would be ideal to train the network using a larger replay memory bank of at least 1 million frames as in Hasselt et al. to increase the final quality of the learned networks. Finally adding more modern

techniques such as prioritized replay (Schaul et al.) would likely help agents learn a better representation. Sharing a priority queue across all games may also allow for the network to focus on learning games it's struggling with instead of continuing to learn equally on all games.

5.3 Conclusion

Looking at the results of our experiments we can see that when multi-task learning our networks were unable to consistently and definitively beat the single-task learning. This could be for a number of reasons, including the need for a different network architecture or more modern techniques, which can be explored in future experiments. Another reason could be the lack of processing power that our predecessors had access to. When comparing our attempts to our predecessors, our predecessors had much more computational power for their use which if given the opportunity could change our results handily. Any number of these reasons could be expanded upon in future work to give a better answer but as it stands now we were incorrect in thinking that the multi-task learning would provide performance improvements over single-task learning in our environment.

6. References

[1] Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* 518, 529–533 (2015). https://doi.org/10.1038/nature14236

[2] VAN HASSELT, H.; GUEZ, A.; SILVER, D.. Deep Reinforcement Learning with Double Q-Learning. **AAAI Conference on Artificial Intelligence**, North America, Mar. 2016. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389

[3] Rudder, Sebastian An Overview of Multi-Task Learning in Deep Neural Networks. CoRR, Jun. 2017 https://arxiv.org/abs/1706.05098

[4] Mujika, Asier. Multi-task learning with deep model based reinforcement learning. ICLR 2017 conference submission, 04 Nov 2016 (modified: 15 Dec 2016). https://openreview.net/pdf?id=rJe-Pr9le

[5] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver. Prioritized Experience Replay. ICLR (Poster) 2016. https://arxiv.org/pdf/1511.05952.pdf