LEARNING INTRINSIC REWARD FUNCTIONS FOR ATARI

Simon Gleeson
Stanford University
sgleeson@stanford.edu

Sofía Samaniego de la Fuente
Stanford University
sofiasf@stanford.edu

Alexandros Tsaptsinos
Stanford University
alextsap@stanford.edu

ABSTRACT

Reinforcement Learning typically struggles in environments with sparse rewards. In this study we utilize an intrinsic reward function to help guide the agent in these scenarios, defined by how well the agent can predict the consequences of its own actions. We train multiple agents asynchronously using deep Q-learning via a convolutional neural network, investigating the impact of intrinsic motivation on how they learn. We test the model on a small subset of Atari games; in particular, we aim to improve performance of the agent in Montezuma's Revenge, a game which suffers from extremely sparse rewards. We have shown that the influence of intrinsic motivation can have positive benefit on the learning of the agent, however can be temperamental to hyper-parameter tuning. Whilst Montezuma's Revenge proved again too difficult, we have shown significant progress can made within Ms. Pac-Man.

1. INTRODUCTION

Reinforcement Learning (RL) is concerned with how agents ought to take actions to maximize their expected utility in the face of an uncertain environment; that is, in the setting where both the reward and the transitions are unknown. Specifically, model-free learning attempts to estimate the optimal policy and its expected utility by minimizing the distance between the current prediction and a target function. In this setting, algorithms must find a balance between exploration and exploitation when choosing a policy from which to collect data from learning, and they receive rewards or penalties along this decision making process.

In 2013, DeepMind [6] incorporated deep learning into Reinforcement Learning (RL) to solve Atari arcade games. In particular, they implemented a variant of Q-learning with the value function modeled via a deep convolution neural network (CNN) trained on the raw pixels input. Using the score of the arcade game as reward feedback, the model was able to gain above human expert-level performance on roughly half of the Atari games.

Despite this positive performance on most games, the deep Q-learning (DQL) algorithm struggled in some instances, especially in the infamously difficult Montezuma's Revenge. In this game, the agent must navigate multiple rooms filled with traps and does not receive a positive reward until they reach a key at the end of each level; in other words, they receive only sparse reward signals. As a result, this learning scheme struggled to match human-level performance, even after hundreds of millions of frames. A possible solution to this problem is the design of a reward function that provides feedback along the route, allowing for faster learning of optimal policies. However, in complex environments it is unclear how this reward function should be defined and simplistic models could lead to convergence towards the wrong goal.

Curiosity or uncertainty of the agent is commonly utilized to help construct the intrinsic reward function. If an agent is unable to predict well the next state given the current state and the action about to be taken, then the error between true and predicted next state form an incentive to encourage exploration. State prediction on pixels is difficult and it is unclear if predicting features at this low-level is helpful to the agent; thus instead an encoded state is utilized for predictions. Instead of simply training an encoding of the whole state, we can train the model so that it captures only the data most relevant to the agent's actions and success. This allows the learner to remain ignorant of items in the environment that are irrelevant to their objective.

Another hindrance to DQL is the large amount of time it takes to train the learner. In particular, in their original paper, Google trained the agents for a week over 50M frames using heavily specialized hardware. Numerous methods have been proposed to overcome these computational issues. Mnih et. al [5] proposed a framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. They showed that the use of several agents learning in parallel has a stabilizing effect on training that allows deep learning and reinforcement learning to interact successfully while reducing the computational burden significantly. Further, their methodology is applicable to both on-policy methods, such as SARSA, n-step methods, and actor-critic models, and off-policy methods, such as Q-learning. Their best performing method, Asynchronous Advantage Actor-Critic (A3C), surpassed DeepMinds's DQL while training for half the time on a single machine with a standard multi-core CPU and, hence, became the current state-of-the-art deep RL algorithm for problems with complex state and action space.

We have implemented an asynchronous variant of Google DeepMind's DQL algorithm with guided exploration through a curiosity-driven intrinsic reward function. In particular, our model is composed of two components: a 'student' that interacts directly with the environment and

a 'teacher' that guides the exploration through a novelty intrinsic reward. The goal of the agent is to maximize the expected augmented reward over all time, where the augmented reward is a linear combination of the extrinsic reward provided by the environment and intrinsic reward provided by the teacher.

We experiment with parameter configurations and across several games to attain optimal performance. We show that our algorithm can yield in faster learning in the Atari games already mastered by DQL, however fine tuning of hyper-parameters is needed to understand how the intrinsic reward should decay over time. In particular, the relationship between intrinsic and extrinsic reward is critical but if tuned correctly we show it can accelerate the initial learning of the agent.

2. PREVIOUS WORK

The problem of sparse rewards has been identified and approached from multiple angles in the last few years. In a follow-up to their original paper, DeepMind [1] tackled this problem through building surrogates of the extrinsic reward to drive curiosity in the agent. This approach, often referred to as intrinsic motivation, uses novelty signals such as prediction error, value error, or learning progress to incentivize exploration in a reinforcement learning setting.

Christiano et al. [2] utilized human feedback in the form of simple policy preference decisions to help guide the agent in solving complex tasks. This methodology is somewhat related to that of inverse reinforcement learning and apprenticeship learning, where agents observe demonstrations of the desired task and learn a reward function to train against [9], which Ng et al. [8] showed is possible for the task of autonomous helicopter piloting.

Wernsdorfer and Schmid [13] proposed a variant of deep model-based RL that allows the agent to learn arbitrarily abstract hierarchical representations of its environment. The non-Markovian policies generated were shown to be successful in situations where the relevant states of a task are not known in advance.

Liang et al. [4] suggested a generic representation of a game in the Atari learning environment using 'blobs' of pixels, performing particularly well on games with little reward signal. Kaplan et al. [3] successfully introduced defined input language instructions to Montezuma's Revenge. However, exploration in these domains was still often implemented by simple ϵ-greedy algorithms.

Stadie et al. [12] investigated more sophisticated exploration strategies such as Boltzmann exploration and Thompson sampling. Based on ideas from intrinsic motivation, they proposed a method where a concurrently learned model assigns exploration bonuses to the agent, incentivizing exploration and novelty. Whilst improving performance on numerous games, the algorithm still scored zero in Montezuma's Revenge, showing severely sparse-reward environments remain a step too far.

Pathak et. al [10] use the error in the agent's ability to predict its own actions to motivate exploration. They train the state encoder by minimizing the loss of a network which predicts the action that took the agent from one encoded state to the next. Through this, they are able to define state embeddings that are relevant to the actions that the agent takes, rather than generic state embeddings as in Stadie et al., which also encode a lot of information irrelevant to the learner's success.

While there has been a lot of research in intrinsic motivation as a guided exploration technique, we have yet to find a paper that combines asynchronous agents with intrinsic motivation to alleviate the problem of sparse rewards.

3. METHODS

This section includes a detailed description of our Asynchronous version of Deep Q-learning with Intrinsic Motivation (AIM-DQL). We start by describing the environment that the agent threads will interact with and follow by outlining our proposed algorithm.

3.1 Environment and Pre-processing

We run all experiments with Atari 2600 [1] games, a selection of classic arcade games ranging in level of difficulty. The environment emulator is provided by OpenAI gym [2]. In this environment, extrinsic rewards correspond to changes in the total point score of the game, whilst the end of an episode occurs either when the agent beats the game, or dies. The states are produced by the screen pixel output of the arcade game, with no hand-crafted features. Following Mnih et al. [7], we construct our states by preprocessing the 210×160 raw RGB output from the emulator to reduce its dimensionality. Specifically, we convert the pixels to grayscale and rescale the output to 84×84. Further, to allow capturing velocity, each of our states consists of $m = 4$ consecutive previous preprocessed frames, resulting in states of dimension $4 \times 84 \times 84$.

3.2 Asynchronous Intrinsic Motivation DQL (AIM-DQL)

We now present AIM-DQL, an asynchronous variant of deep reinforcement learning model, where independent agent workers concurrently train a 'student' model that interacts directly with the environment and a 'teacher' model that guides exploration through a novelty intrinsic reward. As suggested by its name, our algorithm is composed of

[1] The Atari 2600 is a home video game console released by Atari, Inc. in 1977 that was wildly successful for decades. The 2600 was typically bundled with two joystick controllers, a conjoined pair of paddle controllers, and a game cartridge: initially Combat, and later Pac-Man.

[2] https://github.com/openai/gym

three components: Asynchronous (A), Intrinsic Motivation (IM), and Deep Q-Learning (DQL), which we describe in detail below.

3.2.1 Deep Q-Learning

This algorithm is a variation of Q-learning where, at each state s the agent (player) selects an action $a \in \mathcal{A}$ from the set of legal actions (e.g. up, down, right, left, jump) and passes it to the environment (Atari emulator), who updates its internal state $s \rightarrow s'$ and the game score. The goal of the agent is to maximize its future expected utility. In particular, this model-free approach estimates the action value function $Q_{\text{opt}}(s,a) = \max_{a \in \mathcal{A}} Q(s,a)$ through a CNN, using stochastic gradient descent to update the weights over loss

$$L(\theta) = \mathbb{E}_{s,a,r,s'}\left[\left(Q_{\text{opt}}(s,a) - (r + \gamma Q_{\text{opt}}(s',a'))\right)^2\right].$$

Previous attempts to use a non-linear function to estimate the value function had led to instability. To help achieve stability a target network is created as a copy of the predictive network to calculate $V_{\text{opt}}(s') = Q_{\text{opt}}(s',a')$. This target model is updated infrequently by copying across the weights from the predictive network.

3.2.2 Asynchronous Agents

We execute multiple agents in parallel, on multiple instances of the environment. This parallelization has two objectives: on one hand, it decorrelates the agent's data into a more stationary process and has parallel agents employing different exploration policies, which enables it to achieve stability; and on the other hand, and most importantly for our purposes, reduces the training time to roughly linear in the number of parallel actor-learners. The decorrelation removes the need for an experience replay mechanism, which is both expensive in memory and computation.

Each agent updates the prediction network with mini-batches of experiences learned. Importantly, we can use Hogwild! [11] style updates and let each agent update the network without any thread locking. Each thread is presented with its own exploration policy to encourage distinct exploration.

3.2.3 Intrinsic Reward

We drive curiosity through guided exploration to enable the agent to explore large parts of its environment and learn skills that might be useful in the games where myopic exploration fails. Our approach takes inspiration from both the work of Stadie et al. [12] and Pathak et al. [10].

We model the intrinsic reward, $r^{(\text{int})}$, as the prediction error of a dynamic predictive model that takes an encoded version of a state s and the agent's action a and attempts to predict an encoded version of the successor state s'. The intuition behind this idea lies in the fact that when we don't understand a state-action pair well enough to make accurate predictions, the error will be high, and hence exploration of this pair will be encouraged.

Formally, let $\phi(s)$ denote the encoding of state s and $f(\phi(s),a)$ the state predictor for $\phi(s')$. At every step, the teacher model will receive a tuple $(s,a,r^{(\text{ext})},s')$ from the emulator, calculate representations $\phi(s)$ and $\phi(s')$, use the predictive model to estimate $\phi(s_{t+1})$ from $\phi(s_t)$ and a_t, calculate the intrinsic reward $r^{(\text{int})}$ as the prediction error $||\phi(s') - f(\phi(s),a)||^2$, and finally compute the modified reward $r = r^{(\text{ext})} + \beta r^{(\text{int})}$, where $r^{(\text{ext})}$ is the game score update provided by the Atari emulator. The teacher model will then send the tuple (s,a,r,s') to the student, who will use this information to update its own networks weights. The pseudo-code can be seen in Algorithm 1.

Parameters:
  β, γ, $T^{\max}$, T^p, T^t, T^m
Global Variables:
  T, θ^p, θ^t, θ^m
while $T < T^{\max}$ **do**
  Set:
    $T \leftarrow T + 1$
    $\mathcal{B}^p, \mathcal{B}^m \leftarrow \emptyset$
    $s \leftarrow s_{\text{start}}$
  while *not IsEnd(s)* **do**
    Choose action a that maximizes $\hat{Q}_{\text{opt}}(s;\theta^p)$
    Take action a and observe $r^{(\text{ext})}$, s'
    Encode s and s' to get $\phi(s)$ and $\phi(s')$
    Compute $e \leftarrow ||\phi(s') - f(\phi(s),a;\theta^m)||^2$
    Normalize e using constant decay to get $r^{(\text{int})}$
    Set $r \leftarrow r^{(\text{ext})} + \beta r^{(\text{int})}$
    Compute $y \leftarrow r + \gamma \hat{V}_{\text{opt}}(s')$
    Add (s,a,y) to batch $\mathcal{B}^p$
    Add (s,a,s') to batch $\mathcal{B}^m$
    if T *mod* $T^p == 0$ **then**
      Use $\mathcal{B}^p$ to update θ^p
      Set $\mathcal{B}^p \leftarrow \emptyset$
    end
    if T *mod* $T^t == 0$ **then**
      Update target weights $\theta^t \leftarrow \theta^p$
    end
    if T *mod* $T^m == 0$ **then**
      Use $\mathcal{B}^m$ to update θ^m
      Set $\mathcal{B}^m \leftarrow \emptyset$
      Optionally, update ϕ
    end
    $s \leftarrow s'$
  end
end

Algorithm 1: Asynchronous Intrinsic Motivation DQL: pseudo-code for each worker thread. T is the global frame counter; $T^{\max}$ is the maximum number of frames; T^p, T^t, T^m the update frequencies for the predictive model, target model, and teacher model respectively; $\theta^p, \theta^t, \theta^m$ are the weights for the predictive, target, and teacher model respectively.

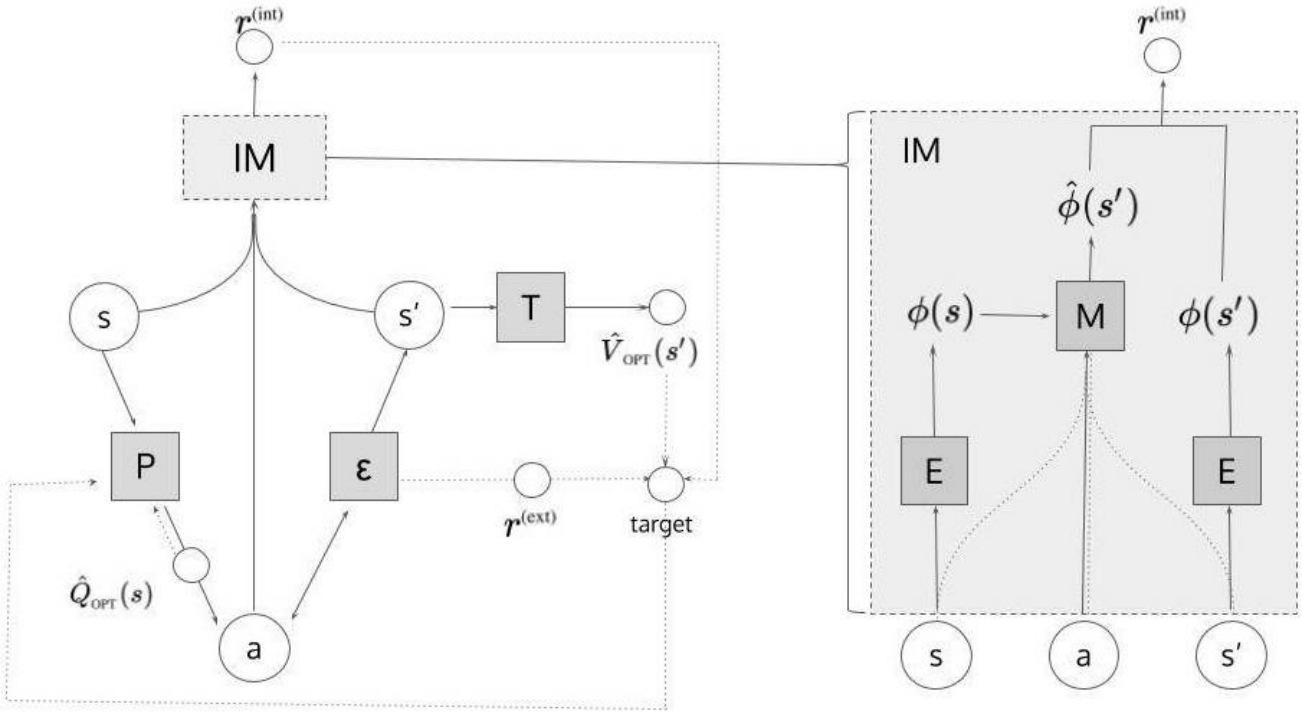


Figure 1: AIM-DQL network architecture. IM is the intrinsic motivation mechanism, seen in detail on the right.

The state encoder is trained in a manner such that only information relevant for predicting the actions of the agent is represented in the output. In this manner we hope to remove the effect of features of the environment which are out of the agent's control and which have little impact on their success.

This is achieved by including a third neural network architecture which takes as input the encoded state, the encoded next state and predicts the action which took the agent from state to the next. Since the action space is encoded into one-hot vector representations, we utilize cross-entropy loss between predicted and true action to update the weights of the action predictor and encoder. To avoid beginning the agent with a completely ineffective encoding network, the teacher is trained for a number of frames whilst the agent follows an epsilon-greedy strategy, before transitioning to teacher-guided exploration. A diagram of the architecture of the full model can be seen in Figure 1.

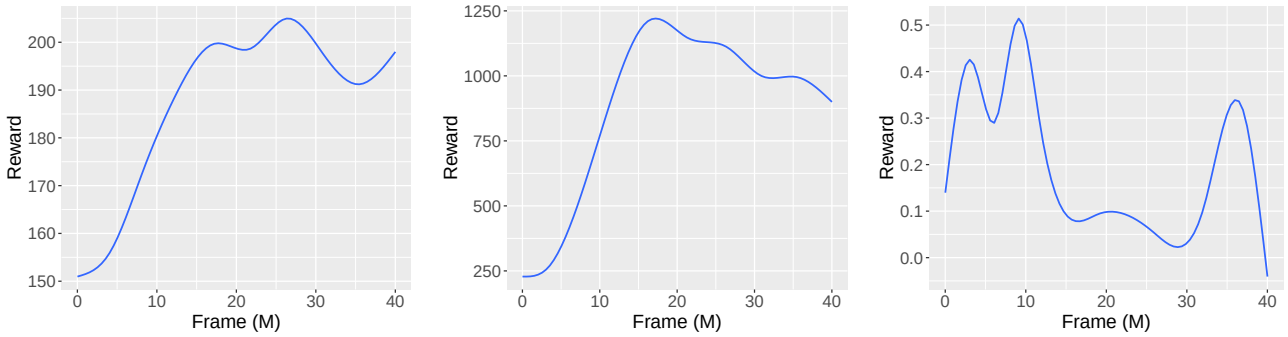
4. EXPERIMENTS

For our experiments, we trained our models on a subset of the Atari 2600 suite: Space Invaders, Ms. Pac-Man and Montezuma's Revenge. We choose this selection of games because they grant us a spread of different reward structures on which to evaluate our algorithm. Space invaders and Ms. Pac-Man more immediate reward structure allowed us to calibrate our baseline models with those implemented by DeepMind. Montezuma's Revenge, due to it's harsher, sparse reward environment, serves as the objective on which we would like our algorithm to perform well on. If the teacher model can successfully create subgoals to guide the student in this game, we expect it to perform well in other, less difficult games. We experiment using both asynchronous DQL (A-DQL), and with intrinsic motivation (AIM-DQL) included. It must be noted that since we trained A-DQL and AIM-DQL using CPUs and GPUs respectively, we were training and tweaking the models concurrently. As such, whilst written in a sequential manner below, it was not the case due to time limitations that all A-DQL optimizations were completed before we began testing AIM-DQL.

4.1 Experimental Setup

Following Mnih et al. [7] the Q-value network is a convolution network consisting of three convolutional layers followed by two fully connected layers. The first convolutional layer convolves 32 filters of 8×8 using a stride of 4, the second layer convolves 64 filters of 4×4 with stride 2, and the third layer convolves 64 filters of 3×3 with stride 1. All convolutional layers use a ReLU activation function. The first dense layer utilizes 512 units with a ReLU activation function. The final layer uses a linear activation function onto the number of possible actions.

The encoding of the state uses the same architecture without the final dense layer; thus, an encoded state corresponds to a 512 dimensional vector. The action predictor utilized to update the encodings is a two layer fully-connected network. The encoded state and encoded next state are concatenated together before being passed through a first dense layer of 256 units with ReLU activation. This is followed by a second dense layer with soft-



(a) Space Invaders (b) Ms. Pac-Man (c) Montezuma's Revenge

Figure 2: Learning curves for 40M frames using the A-DQL implementation.

max activation onto the number of possible actions. The encoder is then trained via cross-entropy loss.

The teacher model attempts to predict the next encoded state given the current encoded state and action using a two-layer fully-connected network. The first dense layer consists of 256 units with ReLU activation, whilst the second layer then outputs to 512 units, the size of the encoded representation, via a linear activation. The teacher model is then trained via Euclidean loss.

For A-DQL we use an epsilon-greedy policy to encourage exploration. Each thread is assigned a final epsilon value as stated in [5], and after an initial period of 50,000 frames random play the ϵ value is linearly annealed to its final value. In AIM-DQL the agent trains under epsilon-greedy strategy for 500,000 frames to initially train the encoder before switching to teacher-guided exploration. After initial experimentation we found that the optimal number of threads is equal to the number of cores available on the computing node. In line with resources available to us, for CPU training we thus utilized 16 threads to train models whilst for GPU training we used 12 threads.

For each game we 'clip' the extrinsic rewards, that is, we restrict all extrinsic rewards to exist within $[-1, 1]$. We implement this so that our model can be generalized to all the games in the Atari suite, as the order of magnitude of the scores varies significantly between the games. The intrinsic reward is similarly standardized to $[0, 1]$ through division by the maximum state prediction error observed in the experiment. Over time exploration should be discouraged as the agent learns more optimal games. How the intrinsic reward should decay over time was an unknown that needed to be tuned. To aid this, we experiment between allowing natural decay of intrinsic reward, formed through the natural decrease in state prediction over time, and implementing a hard-coded decay constant which linearly decays the intrinsic reward over time. That is, if the error from the intrinsic model is given by e then we normalize over the maximum error and some decay parameter δ,

$$e_n = \frac{e_n}{\max_{m \leq n} e_m} \delta.$$

Similarly to epsilon-greedy, δ begins at 1 and linearly anneals down to some final value over some period of frames.

To allow the agent to observe more frames, OpenAI gym implements a frame skipping technique where each action is repeated k times, where k is randomly sampled from $\{2, 3, 4\}$. We set the discount rate as $\gamma = 0.99$.

We use the RMSProp optimization algorithm to update all models with learning rate $\eta = 0.00025$ and momentum $\gamma_m = 0.95$. We update the prediction network in mini-batches of 32 frames unless otherwise mentioned and update the target network every 50,000 frames. In AIM-DQL the teacher and encoder are updated every 1,000 frames.

4.2 A-DQL

We began by implementing the A-DQL model proposed by Mnih et al [5] in Tensorflow [3] as described in Section 3.2.2.

4.2.1 Initial Experiments

The A-DQL model is trained for 40M frames over the chosen subset of three games. A smoothing curve of the total reward per episode attained by the workers over time can be seen in Figure 2. After this training period the agent's learnt policy was evaluated over 200 games. The scores achieved and reported scores from previous research can be seen in Table 1.

From the plots we observe that the model is certainly learning for both Space Invaders and Ms. Pac-Man, as it makes steady improvements to its reward over the episodes. As expected, Montezuma's Revenge does not learn anything over the 40M frames, with the sparse reward structure unable to guide the agent to any positive reward consistently. Both Ms. Pac-Man and Space Invaders observe a very rapid increase in the beginning; however this is followed by a decrease in performance from just

[3] http://tensorflow.org/

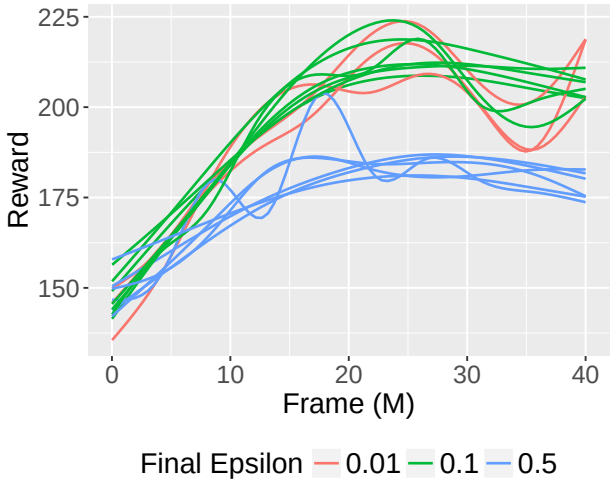


Figure 3: Learning curves for 40M frames of A-DQL Space Invaders grouped by thread and colored by final epsilon value.

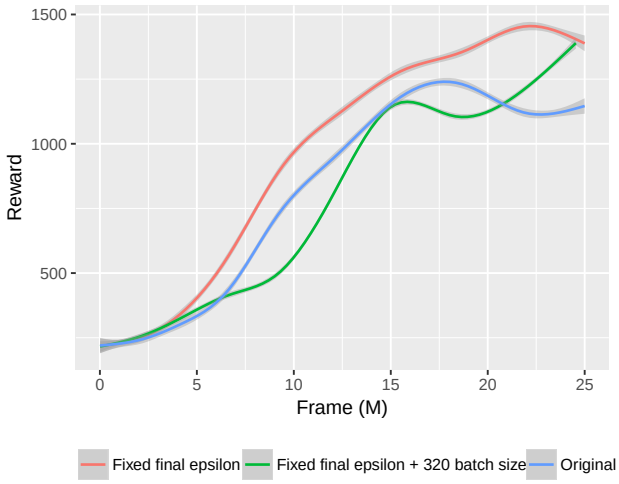


Figure 4: Learning curves for 25M frames of A-DQL Ms. Pac-Man varying whether we keep final epsilons fixed across all threads, and increasing the batch size.

before the 20M frame mark, where the agent begins 'un-learning' whatever information they learned to get him to that point.

4.2.2 Optimizations

The scores achieved in the evaluation are well-below the achieved scores of DeepMind. For this first implementation we did no hyper-parameter tuning, using the reported hyper-parameters from previous research. Comparing to the learning curves of previous research, the initial period of learning is consistent with what others have reported, but the 'un-learning' mentioned above affects the score greatly. We believe that two aspects could be causing this problem.

First, each thread is following a different learning policy with some threads final epsilon value being equal to 0.5. Thus, even when the epsilon value has finished annealing, the agent will still be performing random movements half the time. Past a certain threshold we ideally want the agent to begin exploiting its knowledge rather than continuing to explore, and by continuing to randomly explore these threads could be influencing the learning of the whole model. This can be seen in the learning curves of Space Invaders grouped by thread and colored by final epsilon value in Figure 3. Second, the 'un-learning' could be caused by too small a batch size, due to the stochastic nature of SGD. As stated in Section 4.1, we utilized a batch size of 32 to update the predictive network.

Since Ms. Pac-Man showed the most significant amount of 'un-learning' after a successful initial period we experimented with this game. We first altered the final epsilon to be set at 0.01 on all threads, rather than varying per thread. We then experimented with increasing the batch size by a scale of ten to 320. We trained the model for 25M frames to investigate whether we observe any 'un-learning' at around

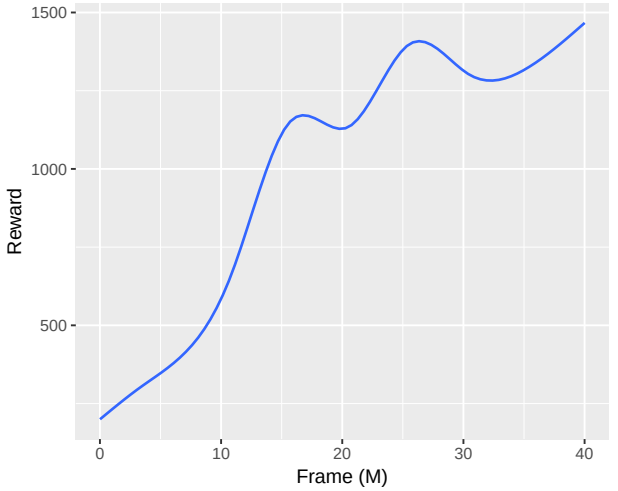
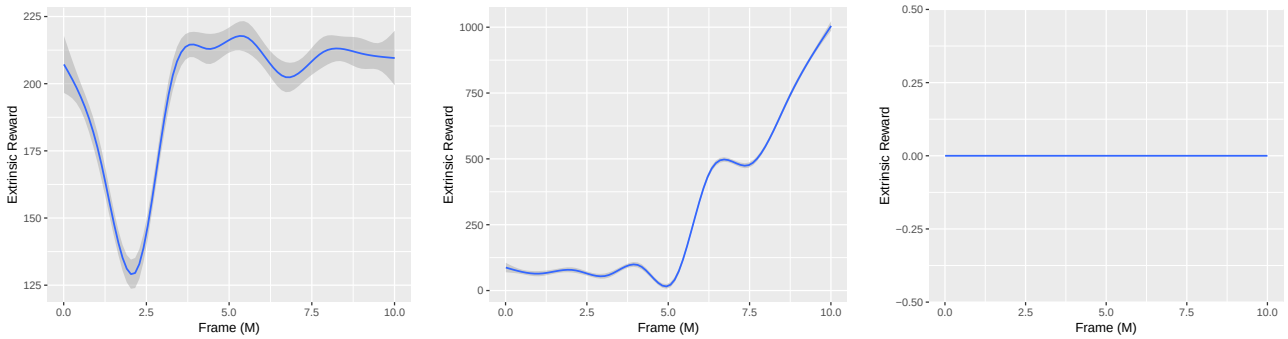


Figure 5: Learning curves for 40M frames of A-DQL Ms. Pac-Man using fixed final epsilon across threads and a batch size of 320.

the 20M frame mark, as previously. The comparison between the different configurations can be seen in Figure 4.

The agent produced higher episode rewards when the epsilons are all fixed to 0.01. This is natural, since every thread is performing less random than before and allowed to exploit rather than keeping some threads fixed at random actions half the time. However, the stagnation and beginning of 'un-learning' still exhibits itself at around the 20M mark. Despite a brief moment of uncertainty around the 15M frame mark, the larger batch size combined with fixed epsilon value finished at 25M frames the strongest performing algorithm. This is in line with our knowledge of SGD; larger batch sizes will exhibit slower learning but are more stable.

To ensure that we had not just prolonged the 'un-learning',



(a) Space Invaders (b) Ms. Pac-Man (c) Montezuma's Revenge

Figure 6: Learning curves for 10M frames using the AIM-DQL implementation.

Game	Human Expert	DQL	A-DQL	AIM-DQL
Ms. Pac-Man	15,693	2,311	841	1087
Montezuma's Revenge	4,367	0	0	0
Space Invaders	1,652	1,976	295	228

Table 1: Human and DQL scores quoted from [1], DQL is run for 50M frames, A-DQL is run for 40M, and AIM-DQL is run for 10M frames.

we proceeded to run the A-DQL model with fixed epsilons and bigger batch size for 40M frames. The learning curve can be seen in Figure 5 from where we see that learning is continued past the initial stagnation zone seen prior. After evaluating this model on 200 episodes the agent attained an average score of 1,334. Comparing this with Table 1 we see that we have improved upon our initial score, although not quite matched DeepMind's score which was subject to further training frames and extensive hyper-parameter tuning. It took approximately 14 hours for the A-DQL model to train for 10M frames and, as such, we were extremely time-limited in what experiments could be run. Pleased with the relative success of the A-DQL model, we ended our exploration of A-DQL here.

4.3 AIM-DQL

As the main focus of our work, we implemented the AIM-DQL model as described in Section 3.2.3. With three neural networks being concurrently trained, the computational expense for this model was much higher. We therefore trained all AIM-DQL models on a GPU with 10M frames taking around 13 hours to train, not including the initial epsilon-greedy stage.

4.3.1 Initial Experiments

Similarly to A-DQL, we first began by running AIM-DQL for 10M frames on our chosen subset of three games with reward tradeoff parameter of $\beta = 1$. For our first experiment we let the final decay constant δ value be 0.001, which is reached at the $1M^{th}$ frame. The learning curves can be seen in Figure 6 and the evaluated results in Table 1.

Ms. Pac-Man has showed greatly improved results from the introduction of intrinsic motivation, achieving a 25% boost in score in a quarter of the training frames. Further, the AIM-DQL model has a similar boost over the learner curve when both are compared at 10M frames. From the learning curve we can observe an initial stage where no learning seems to take place before a period of immense learning.

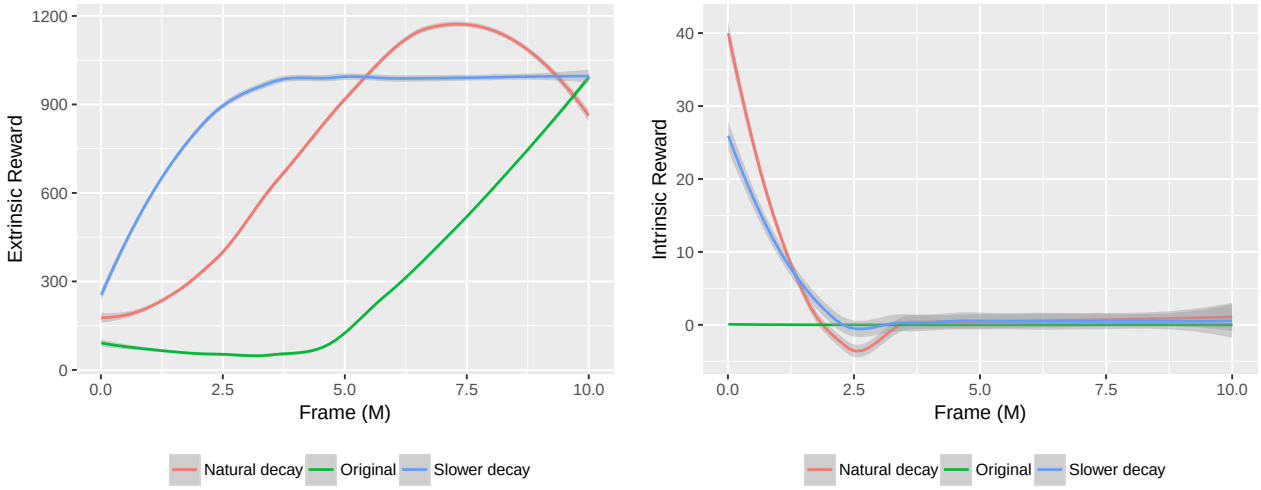
Disappointingly, Space Invaders and Montezuma's Revenge show no promise of learning anything. Whilst for Montezuma's Revenge this is no worse than before, the introduction of intrinsic motivation has worsened performance for Space Invaders.

4.3.2 Optimizations

Similarly to the previous section, with the limited time available we proceed by optimizing the AIM-DQL model over Ms. Pac-Man, due to its positive response to the introduction of intrinsic motivation.

The initial stagnation in the learning of Ms. Pac-Man seen in Figure 6b is possibly due to the decaying feature of intrinsic reward. It is possible that by the time learning has started the intrinsic reward has decayed to such a small value that it is insignificant. This could explain why the learning curve at the beginning of Figure 6b looks very similar to that seen in Figure 2b. We believe that the learning of the model is highly dependent on how the relationship between intrinsic and extrinsic motivation is defined, so we focus our attention on this matter.

There is a portion of decay automatically available to the model through normalization. Since we divide by the maximum error seen and over time the model learns better encodings, the intrinsic reward will naturally decay. How-



(a) Extrinsic Reward

(b) Intrinsic Reward

Figure 7: Learning curves for 10M frames of AIM-DQL Ms. Pac-Man varying how the intrinsic reward decays over time.

ever, it is unclear whether this decay is of a suitable rate, which is the reason behind the introduction of the decay parameter δ. We experiment by varying the decay mechanism between: the original decay which linearly annealed δ from 1 to 0.001 over 1M frames, a slower decay which linearly annealed δ from 1 to 0.01 over 10M frames, and natural decay only which kept $\delta = 1$ constant throughout the learning. The learning curves these experiments can be seen in Figure 7.

We observe some interesting trends here. Firstly, that the original decay and slower decay finish at nearly the same point despite taking very different trajectories. As discussed earlier, the original decay seems to be negatively impacted by intrinsic motivation until it has decayed to an insignificant value before learning can begin. On the other hand, the slower decay seems to gain a great initial boost through the presence of intrinsic motivation before tailing off into a period of slower learning. This implies that intrinsic motivation can certainly help to speed up initial learning of an agent by providing guided assistance, although how the interplay between intrinsic and extrinsic motivation changes over time is still unclear. Natural decay proves unpredictable and appears to suffer from 'unlearning' at around the 7.5M frame. However, this could possibly be due to the small batch size still being utilized for this experiment.

From Figure 7b we can see that the original decay gives a negligible intrinsic reward, which contributes to the similar performance with A-DQL. Natural decay allows for higher values in the beginning, however is followed by a sharp drop to negligible values when the teacher model has learned the encoding well. Slower decay attempts to elongate the period of non-negligible intrinsic reward, however all methods give negligible intrinsic reward for anything past 2.5M frames. One reason for this could be the extended period of training that the encoder is subjected to prior to beginning AIM-DQL. By shortening this pre-training period we could allow the intrinsic reward to stay higher for a longer period in AIM-DQL.

One issue we encountered was concerning whether concerning the separate worker threads continued to explore distinct trajectories. This was seen when the teacher was not given sufficient frames to initially learn a good encoding. As a consequence, the impact of intrinsic reward was to deterministically complete the same action repeatedly, disabling further learning and performing poorly.

From Figure 8 we can see that each thread does follow a similar learning curve, although similarity is expected to some degree since each thread utilizes and updates the same predictive network. While stable online deep Q-Learning is possible without experience replay, Mnih et. al [5] suggested that incorporating this technique into the asynchronous RL framework could substantially improve the data efficiency of these methods. Therefore to overcome this potential issue of threads behaving in too similar a manner an experience-replay mechanism could be implemented.

4.3.3 Optimized Final Run

As was noted previously, the optimizations for AIM-DQL were done in parallel with the optimizations over A-DQL. We finally combined the knowledge we have learnt together to train the learner. We kept epsilons fixed across threads with an enlarged batch size of 320, as this was seen to benefit A-DQL in Section 4.2.2. We utilized the slow decay mechanism for intrinsic motivation as we believed this showed the greatest promise, as can be seen in Section 4.3.2. We run the AIM-DQL model with the aforementioned configuration for Montezuma's Revenge. With a maximum wall-time of 24 hours on our GPU we train the model for as many frames as possible in that time, namely 18M frames. We would have liked to train each game over this number of frames but due to time restric-

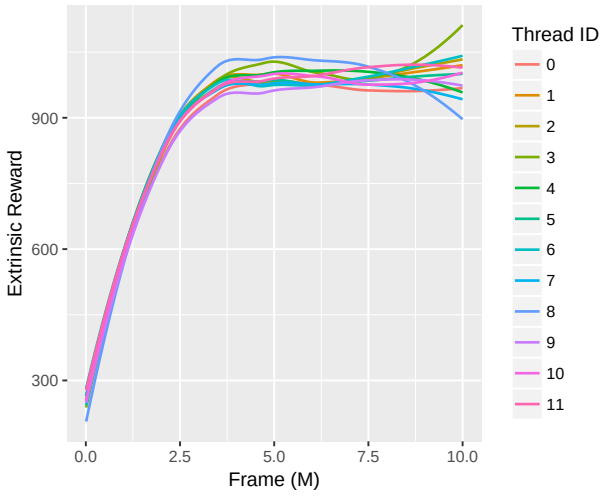


Figure 8: Learning curves per thread for 10M frames of AIM-DQL on Ms. Pac-Man.

tions we could only choose one, and with sparse-reward environments our overall target we selected Montezuma's Revenge .

Unfortunately, over 18M frames the learner was unable to achieve any non-zero extrinsic reward and scored zero on its evaluation. Whilst this is disappointing, it is not seen as a complete failure to improve performance on A-DQL. From the improvements seen on Ms. Pac-Man we still hold a strong belief that given further time for more parameter tweaking and increased computational power intrinsic motivation would be able to help guide the learner to a more successful path.

5. CONCLUSION

Sparse-reward environments remain a tough and current challenge to the AI research field. Without the researcher providing domain-specific knowledge to the model the teacher must try to influence the learner to take a successful path. In this project we utilized an exploration-based intrinsic motivation function to enable a teacher model to help guide a learner. We combined various methods from state-of-the-art models published in the last couple of years. Namely, we utilized the asynchronous aspect of training from Mnih et al. [5], combined with the role of intrinsic motivation function defined by error in a separate encoded state predictor teacher model as proposed by Stadie et al. [12]. To train the encoder we incorporated the ideas of Pathak et al. [10] by using a third model to predict actions given encoded states, enabling the teacher to learn an encoding relevant to the possible actions and outcomes of the agent and ignore irrelevant outside interference.

We began the project by implementing an asynchronous version of the DQL network originally proposed by DeepMind [6]. We showed that with little parameter tuning we could produce good results on Ms. Pac-Man from feeding in raw-pixel input through a convolutional neural-network, with multiple agents acting in parallel to allow faster training and stabilizing the learning process. We discussed optimizations to this process through increased batch sizes and keeping final epsilons fixed across threads.

We continued by implementing the intrinsic motivation model AIM-DQL and testing the model on the same subset of games. The model was shown to be extremely temperamental to how the trade-off between intrinsic and extrinsic motivation was defined, and how the intrinsic motivation decayed over time. If configured correctly, we showed that Ms. Pac-Man heavily benefited from intrinsic motivation with an extremely increased initial learning period, although this was then seen to be followed by a period of stagnation.

No previous research has been able to conquer Montezuma's Revenge without resorting to game-specific intrinsic reward-defined functions as in Bellemare et al. [1] or Kaplan et al. [3]. As such, attempting to tackle this game in a project may have proved one step too far. However, we have showed that progress can be made through the intrinsic motivation mechanism as seen in Ms. Pac-Man and we believe that with further configuration tuning progress can be made in other games.

All code is available on a Github repository, `https://github.com/sofia-samaniego/cs221project` as well as submitted in a cleaner format in conjunction with this paper. We have additionally run the A-DQL model on CodaLabs in the online worksheet found at `https://worksheets.codalab.org/worksheets/0x935af61ad7a043478ee22f0ab0a39b03/`. Due to extended training times and necessity of GPUs we have not run the AIM-DQL model on this platform.

Possible future work for this project would be to continue the experiments to find the optimal configuration. In particular, with the time and computational power a full grid search over intrinsic reward decay parameters and trade-off parameter β would be extremely beneficial. Research into whether the reintroduction of experience replay aids the learner could provide insight into whether the separate workers are behaving differently enough or not. Taking inspiration from Pathak et al. [10] we defined the encoder to have the same network architecture as the predictive model. Further research into different encoders could provide insightful, and a comparison with the autoencoder architecture introduced by Stadie et al. [12] would help confirm whether the encoder is truly learning an encoding that is more beneficial to the agents learning.

6. ACKNOWLEDGEMENTS

The authors would like to thank Bryan He for his helpful comments and remarks throughout this project and ICME for access to their CPU and GPU computing nodes.

7. REFERENCES

[1] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[2] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

[3] Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning. *arXiv preprint arXiv:1704.05539*, 2017.

[4] Yitao Liang, Marlos C Machado, Erik Talvitie, and Michael Bowling. State of the art control of atari games using shallow reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 485–493. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[8] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. Springer, 2006.

[9] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[10] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *arXiv preprint arXiv:1705.05363*, 2017.

[11] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

[12] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[13] Mark Wernsdorfer and Ute Schmid. Grounding hierarchical reinforcement learning models for knowledge transfer. *CoRR*, abs/1412.6451, 2014.