Deep Reinforcement Learning on the Doom Platform

Shaojie Bai
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
shaojieb@andrew.cmu.edu

Chi Chen
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213
chic@andrew.cmu.edu

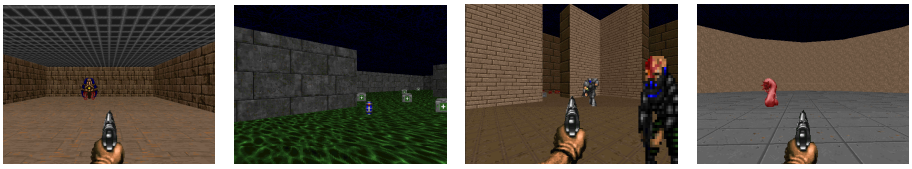
Abstract

The project looks into training AI models for the game Doom, using some of the latest deep reinforcement learning techniques. While most of the prior attempts to train complex game AIs have resorted to Q-learning based methods, in our project, we sought to improve not only on the original model but also on the underlying architecture. Therefore, besides our experiments that used classical DRQN method to train our AI, We also explored the Asynchronous Advantage Actor-Critic (A3C) algorithm [Mnih et al. 2016], which has been recently proposed as a concurrent attempt to solve reinforcement learning problems more efficiently.

1 Introduction

The usage of deep reinforcement learning in game playing has been wildly successful. One of the most important examples is the usage of Q-learning based methods in training deep reinforcement learning (DRL) players, such as the application of Deep Q-Networks (DQN) to play Atari games [Mnih et al. 2013] and the very victory of AlphaGo over human champion on Go [Silver et al. 2016]. In this project, we sought to explore the usage of DRL to train an artificial intelligence (AI) agent in the 3D game Doom, a classical First-Person Shooting (FPS) game created in 1993.

There are several critical differences between training in Doom and in other games like Atari Breakout or Pong. One major difference that makes Doom AI more difficult is its complexity. In particular, compared to other games, Doom involves more variations in actions including moving, attacking, picking items, and their combinations. A good Doom AI should form a strategy that not only picks the right action given a specific situation, but also takes into account of previous intentions and potentially also future impacts of an action. Such strategy may not be obvious to even human players. In addition, the input for training, which is the graphics interface (figure 1), does not always present the entire game state. The partial observability of game states, therefore, is another challenge that this game poses in DRL. To overcome these challenges, more sophisticated techniques and careful experiments are needed in training.



(a) Basic (b) Health Gathering (c) Deathmatch (d) Defend the Center

Figure 1: Common Doom scenarios

Recent works on training the Doom AI have proved that with correct training method and hyperparameter setting, DRL can be quite a useful tool in training an agent that outperforms built-in bots and even average human players [Lample & Chaplot 2016]. While these prior works in Doom DRL

mostly focused on improving Q-learning based methods, we also explored and trained the AI using a different architecture— the A3C model [Mnih et al 2016], which is one of the latest DRL techniques that was proposed to tackle some important drawbacks DQN had (section 3), such as reliance on replay memory and off-policy algorithms. We also tried to compare its experiment performance with that of Q-learning methods (section 5) in this project.

The training and testing task for Doom has been made more convenient by VizDoom, a Doom-based machine learning platform that offers API to control the game as well as the original Doom graphics interface if needed [Kempka et al. 2016]. Our project utilizes this platform in training and testing processes, leveraging the API provided to directly feed control actions and obtain game states.

2 Related Work

2.1 On deep reinforcement learning

Many of the successes so far in training DRL AIs were on fully observable games, such as Atari Breakout or Space Invaders [Mnih et al. 2013]. Most of the game AIs trained using Q-learning as well as its variant algorithms, such as DRQN [Pascanu et al. 2014], prioritized (weighted) replay [Schaul et al. 2016] and dueling [Wang et al. 2015] beat the benchmarks that employed other well-performing reinforcement learning models, and obtain the highest rewards among the models.

Drawing an idea parallel to the dueling architecture, Mnih et al. at Google DeepMind came up with the novel A3C model (Asynchronous Advantage Actor-Critic), which has been introduced to train AIs for Atari games and TORCS car racing [mnih et al. 2016]. Such combination of the asynchronous framework with traditional actor-critic model [Konda and Tsitsiklis 2003] was able to address some critical problems of single-threaded Q-learning. This model demonstrated great robustness and stability in training, and also gains substantial training speedup through multi-threading.

2.2 On Doom

The prior works on training Doom AI have mostly focused on extending the existing deep recurrent Q-learning network (DRQN) model with high-level game information. For example, Hafner and Kempka both experimented on DQN with replay memory (batch size 64) for stabilization [Kempka 2016, Hafner 2016]. Moreover, Lample & Chaplot managed to obtain the internal game status about enemies as extra supervision in their training phase. Their model essentially trained two separate networks, one using DRQN augmented with game features to approximate the action network, and a another simpler DQN for the navigation network. Their methods have led to further improvements on standard DRQN network, and could be easily combined with other common Q-learning techniques such as prioritized experience replay [Lample & Chaplot 2016].

These prior works on related field all have been very insightful in helping us develop our model, including on tuning hyperparamters and further optimizing the training efficiency. They serve as a baseline that we used for further exploration, such as our experiment with the A3C model.

3 Methods

For reinforcement learning, it usually suffices to learn to correctly estimate either:

- the state value function $V(s_t; \mathbf{w}_v)$, which judges the value/worth of a state (so that at a certain state we can just move to the next most valuable one); or
- the policy function $\pi(a_t|s_t; \mathbf{w}_\pi)$, which gives a probability distribution on the actions.

But neither is easy to train. With this line of thoughts, there are two methods, in general, that could be used to apply DRL on Doom AI training.

3.1 Classic Models (Q-learning) & Backgrounds

Q-learning methods essentially try to approximate a $Q = Q^\pi(s, a)$ function, which represents the expected return for selecting action a in state s given a policy π (i.e. $\mathbb{E}[R_t|s_t = s, a]$), thus a value function V associated with an action a. The optimal Q^* is then $\max_\pi Q^\pi(s, a)$, which gives the max action value.

Deep Q-network is the neural network representation of the Q function, which can be updated through standard backpropagation technique. In particular, a loss function in standard DQN can be defined as $L = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \mathbf{w}_Q^-) - Q(s, a; \mathbf{w}_Q)]$. (**Note**: other variants, like DDQN, may use the new NN for selection and old NN for evaluation). Through iteratively updating the Q function, ideally it should converge to the true action value function.

In our modeling that employed Q-learning to train the Doom AI, we chose Deep Recurrent Q-Network (DRQN) with LSTM built upon the DQN model to account for the association between game states. Meanwhile, as Mnih et al. suggested in prior works, we introduced the replay memory [Lin 1992] to de-correlate RL updates and stabilize the training. In addition, prioritized memory replay [Schaul et al. 2016] that assigns importance weights to past memories and thus replays more experiences with higher expected learning progress could potentially further enhance the efficiency of memory replay.

3.2 A3C Model: Background and Motivations

Given the prior work built using classic Q-learning methods, which have been a success in general, we have decided to explore another technique that was proposed very recently, the **Asynchronous Advantage Actor-Critic (A3C) model** [Mnih et al. 2016].

A3C has several advantages over classic Q-learning methods. It does not rely on any replay memory: using multi-threading environment (i.e. multiple agents), it directly de-correlates the DRL updates and reduces the environment bias. In addition, it is no longer limited to off-policy learning. More importantly, having concurrent agents explore the map and each with a separate actor-critic adversarial model, the training convergence rate is expected to accelerate. A3C model can run efficiently on even one multi-core machine, without dependence on large-scale frameworks such as MapReduce.

3.2.1 Advantage Actor-Critic (AAC)

While Q-learning based methods try to estimate V, the Actor-Critic model [Konda and Tsitsiklis 2003] attempts to study both V and π. The key idea is two-fold: (1) a good policy should pick a relatively more valuable next state s_{t+1} (i.e. higher V) with high probability at time t; and (2) a good estimator of state value should try its best to distinguish states resulting from good policy and those from bad decisions (i.e. learn the rewards). Two parts therefore reinforce each other interactively. This leads to a score function to be maximized, and a loss function to be minimized:

$$K_\pi^t = \log(\pi(\underbrace{a_t}_{\text{action chosen}} | s_t; \mathbf{w}_\pi)) \cdot \underbrace{(R_t - V(s_t))}_{\text{advantage}} + \beta \underbrace{H(\pi(a_t|s_t; \mathbf{w}_\pi))}_{\text{entropy score}} \quad \text{(Score)}$$

$$L_V^t = (R_t - V(s_t))^2 \quad \text{(Loss)}$$

where H is the Shannon entropy function. We want to maximize K so that we can get better advantage with higher chance; but meanwhile, we want V to be close to R_t (so minimize MSE). When doing gradient updates, we expect $\mathbf{w}_\pi \longleftarrow \mathbf{w}_\pi + \alpha \nabla_{\mathbf{w}_\pi} K_\pi^t$ and $\mathbf{w}_V \longleftarrow \mathbf{w}_V - \alpha \nabla_{\mathbf{w}_V} L_V^t$.

Moreover, especially in Doom, we centralize the extraction process so that it is not biased toward our estimation of the π or V function. One way to do this, is to share the extraction portion (see figure 2).

Figure 2 shows how an AAC model works in our architecture: (1) Stack up the frames (discussed in section 4) to form the state s_t at time t; (2) Use shared CNN+LSTM structure to understand the features of the current state; (3) Split the network into 2 NN's, one with parameters $\mathbf{w}_\pi$ and one with parameters $\mathbf{w}_V$; (4) Use the policy distribution to choose the next action a_t and act on it to generate the next state s_{t+1} and a reward r_t.

We repeat for some $t_{\max}$ times (a hyperparameter) to get $R_{t'} = r_{t'} + \gamma r_{t'+1} + \cdots + \gamma^{(t_{\max} - t')} V(s_{t_{\max}})$.

3.2.2 Asynchronous

Another key part of our architecture for training the Doom AI is the asynchronous part (figure 3). Instead of using replay memory to de-correlate the RL updates, we used parallel actor learners to update a shared model, which can achieve a similar stabilizing effect.

Essentially, for each thread, it performs the following 5 steps iteratively: (1) synchronize the neural network parameters from the global (shared) network; (2) use the parameters $\mathbf{w}_\pi$ and $\mathbf{w}_V$ fetched to

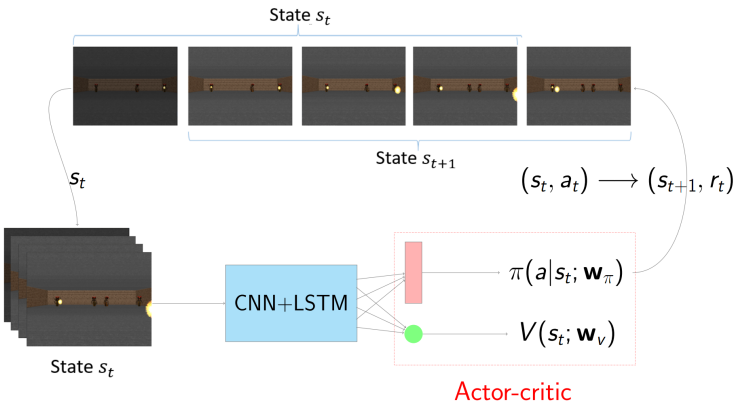


Figure 2: Advantage Actor-Critic Model Architecture. The network splits into two, representing V and π networks.

make actions $a_t, a_{t+1}, \ldots$ according to advantage actor-critic model in figure 2; (3) accumulate the gradients; (4) update the **global** network; and (5) repeat step 1.

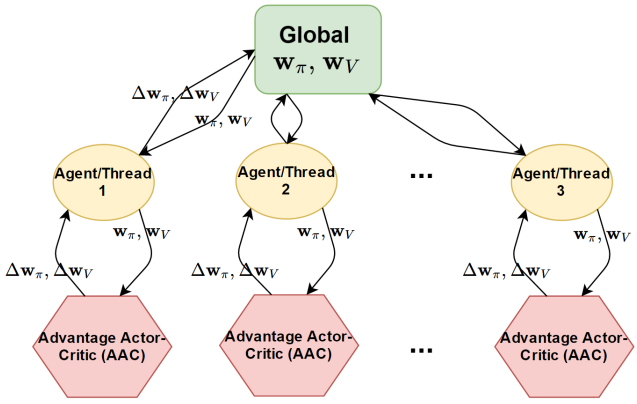


Figure 3: Asynchronous (multi-threaded) architecture. Each thread performs its own advantage actor-critic part and computes the gradients.

3.3 Features of this model

The architecture above and how it tackles the drawbacks of classical Q-learning methods thus form the primary motivation that encourage us to try and explore using the A3C model in Doom. In particular, it has the following advantages that no prior methods offer (simultaneously):

- Scalability and efficiency: Mnih et al. reports a training speed-up achieved by using increasing number of parallel actor learners. Indeed, with enough computing resources, such parallel structure of A3C is able to boost the training pace by a large factor, since more data can be consumed in the same amount of wall clock time [Mnih et al. 2016].
- More edge cases: If there is some rare/edge state not very likely to encounter frequently in single-thread training, A3C model has greater chance to encounter that.
- Stability: Threads exploring different situations all contribute to the same global network. The updates are much less likely to be biased to one certain environment.

4 Dataset

An important part of training an AI is the source of data it uses. In the case of DRL on visual-based game, the major source of data comes from the screen frame. In general, however, we classify the available data types in our Doom training into two categories: the visualization data, and the game variable data.

4.1 Some definitions

(1) **Episode**: An episode ends when either the agent dies or the game times out.

(2) **Training step**: A training step is defined as the step between two backpropagation updates. For A3C, it is the A3C loop in each thread (see figure 2), which iterates for t_{max} times.

(3) **Epoch**: An epoch is defined as a milestone number of training steps we set to complete before we can do some preliminary testing (after which a new epoch starts based on the current result).

4.2 Visualization Data

Visualization data, i.e. the screen frame, is the primary source of the data that we have employed to train the Doom agent. To our program it is a re-scaled 3D tensor of size (60, 80, 3) (i.e. size 60×80 with RGB depth 3). Ideally, we expect an input of larger size would benefit the feature extraction more. However, processing a large image can also take a lot of resources.

Then, we stack up the current screen frame and some λ previous frames to form a state (figure 2) to feed into the neural net. Note that if λ is too small or too large, we may risk losing information or imposing too much weights on history information— both can harm the training. So, in order to best balance the power of visualization data, we took $\lambda = 4$. Furthermore, the concept of skip frame τ is vital [Kempka et al. 2016]. It would surely slow the training pace if we compute and react to every frame we encounter. So, alternatively, once we decide on an action a_t from a state s_t, we repeat the same action for the next τ frames before getting the new s_{t+1}. We set $\tau = 4$ universally, agreeing with Lample & Chaplot's choice [Lample & Chaplot, 2016].

4.3 Game Variable Data

Game variable data are provided by the Doom game platform API [Kempka et al. 2016]. They are episode-based— in other words, the variables are reset whenever a new episode starts (e.g. health, ammo). Table 1 shows a list of the game variables that we most commonly used to judge the performance (and believed to be important):

Table 1: Doom game variables used in training

Doom Game Variables Used (episode based)	
`KILLCOUNT`	Number of kills, by **anyone**
`FRAGCOUNT`	Number of active kills, by **the agent**
`HEALTH`	Remaining health of the agent
`ARMOR`	Remaining armor strength of the agent
`SELECTED_WEAPON_AMMO`	Remaining ammo for the selected weapon
`POSITION_X, POSITION_Y`	The x, y-coordinate for the agent's position

Other variables, such as the `POSITION_Z`[1] (z-coordinate), and `ATTACK_READY` (true if attack can be performed by the agent), are available.

These game variables does not directly impact the agent's decisions, and they are not a part of the states. In fact, in real Doom competitions, no such external information is available. Using game variables in the training phase, however, can help us better define rewards and penalty, thus facilitating the training performance. More to be introduced in section 5.

4.4 Testing & some discussion

A testing phase starts only after a whole epoch ends. In this case, the testing is simply the game itself and the score serves as the performance metric (with the game variables data still available). In addition, since there were multiple threads running concurrently in A3C, we chose to assign the testing tasks in a round-robin fashion to each participating threads. After an epoch, the thread designated the testing task will (1) synchronize the latest network parameters from global net; (2) test for some η new episodes; and (3) report the average rewards received.

[1] A recent update to VizDoom platform in early November enabled the usage of game variables `POSITION_*`.

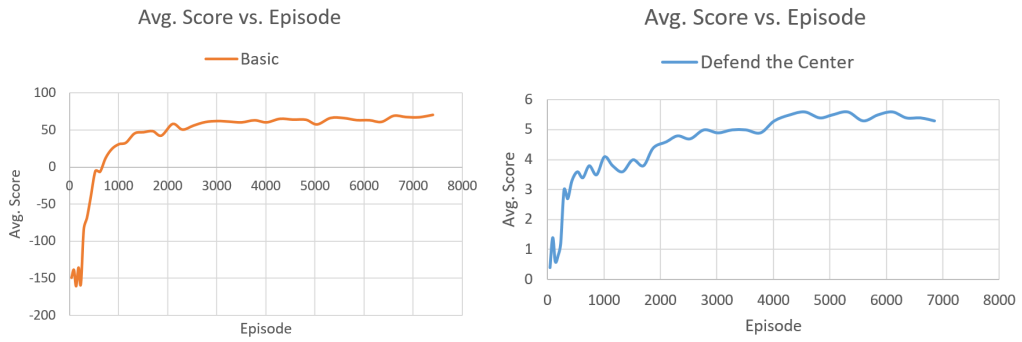
5 Experiments and Results

The most popular cases to train Doom AI are BASIC, HEALTH_GATHERING, and DEATHMATCH (see figure 1). Each focuses on a different aspect: BASIC is the simplest case where the monster is stationary and one can move left/right to attack it. In HEALTH_GATHERING, the agent's health drops continuously, and rewards is based on how long the agent survives. DEATHMATCH is the most complicated case that involves item picking, navigation, and attack (different monsters with different abilities) in a relatively large map.

5.1 Q-learning based classic methods

We first implemented DQN model (based on prior works) to train on simpler scenarios such as BASIC and DEFEND_THE_CENTER. The network consisted of 2 convolutional layers and 2 fully connected layers, together with dropout (which we found to be helpful to bring convergence) and random sampling replay memory. We also attempted to use LSTM layer (size 256) to transform DQN to a DRQN, but the convergence was very similar.

This model has shown to be effective on these simple scenarios, and the training results on BASIC and DEFEND_THE_CENTER scenarios are presented in figure 4. The training led to convergence within about 4000 episodes for BASIC and about 5800 episodes for DEFEND_THE_CENTER, both taking only a few hours. The efficiency and performance matched with those from prior works.



(a) Basic

(b) Defend the Center

Figure 4: Score vs. Episode plot. Training uses learning rate of 0.00025 and RMSProp with decay parameter 0.99. Two plots have different scales because the reward mechanism was different.

5.2 A3C model with some improvements

When using A3C model to train our agents, we found the performance of this architecture is in general similar to DRQN when used in simpler scenarios such as BASIC (horizontal movements and shoot). Therefore, most of our time were spent on training our Doom agent AI in the most challenging map DEATHMATCH, where item-picking, attacking and navigation strategies were all involved.

5.2.1 Experiment network

Our network structure is shown in the table 2 below:

Table 2: Network architecture

Layer #	1	2	3	4	5.1	5.2
	C8x8x16 s4	C4x4x32 s2	FC256	LSTM256	FC8 softmax	FC1

where "C8x8x16 s4" represents a convolution layer with 16 filters of size 8×8 stride 4. We chose RMSProp for the backpropagation, with decay parameter 0.99.

5.2.2 Experiment setting

We added the following reward/penalty settings in addition to the default DEATHMATCH setting in order to better define the desirability of different events and actions:

Table 3: Additional rewards & penalties

Event	reward(+)/penalty(-)
Health increment/decrement	0.04/-0.03 per unit health
Kill count (by anyone)	0.3 per unit
Frag count	1.5 per unit
Ammo increment/decrement	0.15/-0.04 per unit ammo
Displacement	$4 \cdot 10^{-5}$ per unit distance

From this setting, we hoped to set an objective for the agent when it explored the map. In particular, this setting puts a higher emphasis on item picking (medkits and ammo packs) and moving, than killing monsters, which is rewarded as well.

5.2.3 Experiment outcomes

We spawned 16 threads in the experiment using A3C model. Expectedly, the training in a complicated scenario like `DEATHMATCH` takes a long time, and we did not manage to fully complete the training process due to our limited computing power and resources (discussed in section 6). However, we still managed to train a agent that performed reasonably well.

After about 120,000 training steps per thread (so about 1.9M in total), we observed that the agent has already been pretty adept at picking items and meanwhile avoid health decrements (i.e. attacked by enemies). Figure 5 gives a snapshot of the agents at this time: in 6 out of 16 scenarios the agents were in the midst of item picking. Meanwhile, we witnessed progress in the agent's estimate of state value function V. In scenario (a), even though there are plenty of medkits ahead, there is one enemy attacking us— which signals possible quick health decrements subsequently. In scenario (b), we have no enemy in the medkit chamber, which is a safer situation. The difference in the V values reflects the difference in the agents' situations.

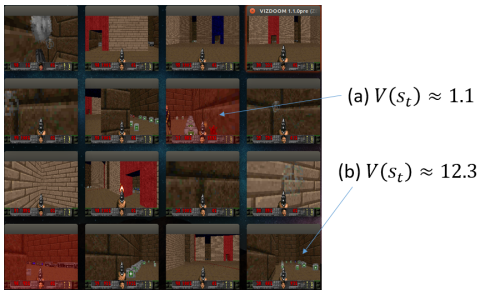


Figure 5: The agent after about 120,000 training steps, per thread (1.9 M steps in total).

After 520,000 steps per thread, we found that the agent already started to learn to engage with the enemies when its health and ammo were in good shape. However, we found that the shots were not very accurate (a little off to the right), and the agent could not easily spot the enemy if the enemy was at the edge of the screen. This sometimes led to high risk of counterattacks from the enemies.

We also roughly compared the convergence plot of A3C and DRQN (figure 6). For A3C, we eventually trained each agent for about 600,000 steps (about 10M in total, a bit more than 3.5 days). For DRQN, due to limited resources and time, we only managed to train the DRQN for about 236,000 steps (which already took a relatively long time). (**Note**: This is the same experiment as in our poster, but the x-axis scale here is "per thread" rather than "total" so that we can compare A3C and DRQN).

6 Discussions and Conclusions

6.1 Understand our results

For the DQN (& DRQN) training on `BASIC` and `DEFEND_THE_CENTER` scenarios, the overall good performance (and convergence within short time) was expected, because the scenarios are simpler and some prior works have shown the method's effectiveness. In particular, we observed the convergence

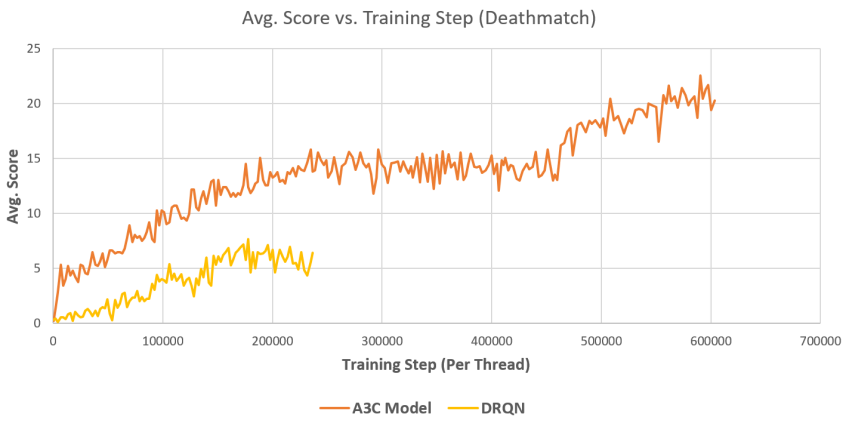


Figure 6: Score vs. Training step plot. Orange is A3C and yellow is DRQN. x-axis is per thread, not total.

of `DEFEND_THE_CENTER` is a bit slower because of the increase in complexity (enemies can move, background is similar, etc.).

The results we obtained from A3C, however, is more interesting. From figure 5 and 6 we can see that even in a scenario as complex as `DEATHMATCH`, A3C model worked quite well in training the agent. Especially for figure 6, we observed that DRQN experienced a long period of "wandering" (score around 2) before it starts to make some progress at about $t = 50,000$. However, for A3C, the multi-threading environment seems to be able to have the overall model make progress more quickly, even though we observed that between about $t = 220,000$ and $t = 460,000$ the agent's average score didn't improve. In general, our results show that using A3C leads to speedups from using normal DRQN training (no dueling [Wang et al. 2015], un-weighted replay memory [Lin 1992]) in at least the first phase of the training.

6.2 Challenges & Future work

While exploring the improvements on the baseline model and training with the A3C architecture, there have been a few challenges that we would like to highlight.

1. **Computing resources**: We are short of manpower and computing power. With only 2 Ubuntu VMs that had single CPU core and 2GB memory, 16 threads were the max that we could spawn (and they are concurrent, not parallel). Computing resources also limit the size (and so power) of our neural network, which needs lots of memory to store and do gradient updates (we had to redefine the gradient updates; see 5 below).
2. **Complexity of the game**: Very few working A3C experiments have been done on games as complex as Doom so far. So lots of hyperparameters ($\alpha, \beta, \lambda, t_{\text{max}}$) to tune. In fact, we believe that with better hyperparameter setting our results can be improved.
3. **Long training time**: Our longest run was 10 million total steps. Existing A3C works [Mnih et al. 2016, Hafner 2016] point out that to get decent performance, usually about 50 millions steps (total) are recommended (they have much larger batches), requiring weeks to train.
4. **Technical**: The unusual update method of A3C (each thread updates global instead of local network) demanded us to extend TensorFlow's implementation of the RMSProp component.
5. **Convergence instability**: Lots of local minima/maxima in A3C optimization, so we clipped the gradient and controlled the learning rate to tackle this problem ($\alpha = 10^{-4}$).

For future work, if we have plenty of computing resources, we should be able to spawn more threads for A3C to observe its benefits, and better the current neural network (by building larger convolution layers to extract features more accurately), which is still small in size compared to the task we are trying to complete here. We can also compare A3C+LSTM to other enhanced DQN model, such as DRQN with prioritized replay [Schaul 2016]. Finally, it may be interesting to further look into the possibility of training the AI with other models and compare the performance with A3C— such as the direct future prediction (DFP) technique Dosovitskiy et al. proposed in their very recent paper [Dosovitskiy et al. 2016].

Acknowledgement

We are very grateful for the invaluable guidance and suggestions from our mentor, Devendra Chaplot, as we made progress in this project.

References

[1] Guillaume Lample. & Devendra Singh Chaplot (2016) Playing FPS Game with Deep Reinforcement Learning

[2] Ba, J., Mnih, V. & Kavukcuoglu, K. (2014) Multiple Object Recognition with Visual Attention. *arXiv preprint arXiv:1412.7755*.

[3] Silver, D., Huang, A., Maddison, C.J. et al. (2016) Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature 529(7587):484-489*.

[4] Kempka, M., Wydmuch, M., Runc, G., Toczek, J. & Jaśkowski, W. (2016) Vizdoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. *arXiv preprint arXiv:1605.02097*.

[5] Hausknecht, M. & Stone, P. (2015) Deep Recurrent Q-learning for Partially Observable Maps. *arXiv preprint arXiv:1507.06527*.

[6] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In NIPS, pp. 1106–1114, 2012.

[7] Wang, Z., Schaul, T., Hessel, S., Hasselt H. et al. (2015) Dueling Network Architectures for Deep Reinforcement Learning. *arXiv preprint arXiv:1511.06581*

[8] Schaul T., Quan, J., Antonoglou, I. and Silver, D. (2016) Prioritized Experience Replay. *arXiv preprint arXiv:1511.05952v4*

[9] Dosovitskiy, A. and Koltun, V. (2016) Learning to Act by Predicting the Future. *arXiv preprint arXiv 1611.01779v1*

[10] Lin, L. (1992) Self-improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. Machine learning, 8(3-4):293–321.

[11] Konda, V.R., and Tsitsiklis, J.N. (2003) On Actor-critic Algorithms. SIAM Journal on Control and Optimization, 42(4):1143–1166.

[12] Hafner, D. (2016) Deep Reinforcement Learning from Raw Pixels in Doom. *arXiv preprint arXiv:1610.02164*

[13] Mnih, V., Kavukcuoglu, K., Silver, D. et al. (2013) Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv: arXiv:1312.5602*

[14] Pascanu, R., Gulcehre, C., Cho, K., Bengio, Y.(2014). How to Construct Deep Recurrent Neural Networks. In Proceedings of the Second International Conference on Learning Representations (ICLR 2014).

[15] Mnih, V., Badia, A.P., Mirza, M. et al. (2016) Asynchronous Methods for Deep Reinforcement Learning. *arXiv preprint arXiv:1602.01783v2*