Playing Doom using Deep RL and Curriculum Learning

Maneesh Tipirineni
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002
stipirineni@umass.edu

Sanjay Reddy S
University of Massachusetts Amherst
140 Governors Dr, Amherst, MA 01002
ssatti@umass.edu

Abstract

Enabling an agent to understand the situation based on visual data and react to it, is central to problems from multiple domains ranging from Robotics to Packet routing. There has been lot of recent work in this regard, performing (visual-based) reinforcement learning by utilizing the power of deep neural networks. In this paper, we compare various advanced Q-Learning models and improvement techniques to these models and also present a curriculum learning approach using transfer learning. We apply and compare our models on scenarios with differing objectives using the VizDoom platform (Doom framework). In particular, we explore Prioritized Experience Replay (PER), a sampling technique for efficient learning by adding it to DQN variants like DDQN and dueling-DDQN. Furthermore, we investigate the effectiveness of curriculum training as opposed to training from scratch. The results show that appending DDQN models with PER, results in much faster convergence (in terms of number of episodes), though with a caveat of slow training time, higher memory overhead and also higher variance in the accuracies (We also proposed a theoretical way to counter this variance). The results from the curriculum learning phase, confirm the intuition that pre-training from similar (sub)-games is beneficial. The accuracies are comparable with significant reduction in total training time.

I. Introduction

Games have long been used in Artificial Intelligence as benchmarks to evaluate the progress of the field. Reinforcement learning models have been showing significant progress in the field lately, achieving a superhuman level of performance [3] playing Atari 2600 games, mastering the illusive game of Go [8] by beating the world champion [10-11] to name a few. The applications of reinforcement learning go far beyond playing games, in the words on Arulkumaran et al. [14] "One of the driving forces behind DRL is the vision of creating systems that are capable of learning how to adapt in the real world". But the field has a long way to go and a multitude of important open challenges remain before that becomes a reality [9]. Currently, the RL models take too long to converge to the optimal policy because it is mostly inferred by trial-and-error and the models do not deal effectively with long range dependencies.

Transfer Learning is seen as a way to reduce training time by pre-training RL models to play similar games so they can learn from actions on those games and have some context of the current game, which might aid training. The idea of being able to learn a task and then transfer and apply the knowledge gained to quickly learn some other task is an important step towards artificial general intelligence, which makes it all the more exciting. However, there is not much existing work that utilizes transfer learning in a curriculum learning setting, which is a natural fit to game playing agents. When humans play a game, we are not immediately introduced to a relatively tough level/boss and expected to play well and overcome it. The game builds towards this, starting with a tutorial to learning the game mechanics, throwing simple opponents/puzzles into the fold to make sure that the player learns and understands the applications of the mechanics and gradually increases difficulty to increase the reward of player satisfaction. We tried to model our training based on this phenomenon. The model learns to "walk before it can try running". We see that this kind of training on incremental subtasks of the main task reduces the training time and improves stability of agent's performance.

In this project, we built a reinforcement learning agent to play the First-person shooter game Doom using the ViZDoom [12] framework. More specifically, we have two phases where in Phase 1 we compared various advanced variants of Deep Q-Learning Networks, Double DQN and Dueling DDQN pairing them up with an effective sampling technique called Prioritized Experience Replay. In Phase 2, we explored curriculum learning to try to make the model understand the game world and game mechanics through iterative subtasks of increasing difficulty, so it can easily learn more complex tasks once the basics are nailed down.

II. Background and Related Work

Significant research is being conducted lately in the field of reinforcement learning. The focus of a substantial portion of this research has been "Deep Reinforcement Learning" i.e., reinforcement learning using deep neural networks. After the incredible success of TD-gammon [27] (back in the 90's), a RL backgammon playing agent that achieved a super-human level of play entirely using reinforcement learning and self-play, the renaissance of reinforcement learning game playing agents came through neural networks and was kicked off by Minh et al. [3] where the first deep learning model to successfully learn control policies directly from high-dimensional sensory input was presented. Minh et al. [3] proposed the Deep Q-Learning Network(DQN) which tries to learn Q-values for state, action pairs through backpropagation to arrive at an optimal policy. Rosenberg et al. [19] and Chuchro et al. [20] have used DQN with different architectures to model game-playing agents.

Many other advanced models were built around DQN network, with improvements. Hasselt et al. [4] came up with a Double Q-Learning Network (DDQN) which reduced the overestimation errors the DQN suffers in some cases. Wang et al. [15] proposed a model called a Dueling Network (Duel DDQN) which not only focuses on the Q-values for each state, action pair but also takes into account the fact that some states give the model more information/learning than others and try to incorporate them into the training process to drive efficient learning. Nair et al. [21] combined parallelism and deep learning and came forward with a distributed architecture for deep learning (Gorila). Hausknecht et al. [31] proposed a recurrent Q-learning network (DRQN) which uses recurrent nets to learn under partially observable environments where data from the previous frame provides key information because the agent does not have full view of the space at a given time. Hessel et al. [18] combines all the improvements/extensions over the DQN and empirically studies their combination comparing their performance with not only each other but also with human performance for different Atari 2600 games.

There have been advances in the way samples are drawn from memory of the network too, to further improve learning. Schaul et al. [2] and Andrychowicz et al. [26] explored different ways of sampling memories rather than uniform learning. The intuition for these "experience replay" techniques is that not all memories are equally advantageous in terms of model learning.

ViZDoom [12] and Atari have been the quintessential fighting pits for reinforcement learning agents and it is no surprise that a range of different models were proposed for modelling doom playing agents. ViZDoom also holds an annual AI competition where these models are evaluated on single-player and multi-player tasks. Clyde [34] and Arnold [13] were amongst the top performing models and they used Actor-Critic (A3C) [32] and DRQN [31] respectively. Bhatti et al [16] used a DQN to play Doom but augmented the data input using a SLAM model which provides localization information and achieved better results both in terms of training time and performance compared to DQNs, DDQNs and Dueling DQNs. Adil et al. [33] extended the same idea of augmenting the data to an Actor-Crtitc (A3C) model. Alvernaz et al. [17] proposed an autoencoder based model with neuro-evolution techniques for optimization.

Transfer Learning between games has also been a subdomain that has seen a lot of innovation lately. Transferring knowledge between different domains saves time to train agents and can improve agent performance. Asawa et al. [1] applied transfer learning to DQN and Duel DQN networks, they learnt weights for their network through backprop of the "Snake" Atari game and using these learned weights, initialized their network for "Puckworld" game. They found that pretraining the convolution layers of the DQN but not the affine layers led to better performance. They also found that transfer learning was able to improve performance and stability in performance. Yin et al. [22] tried to train a single policy network that can be used for multiple tasks at the same time using knowledge transfer and policy distribution where the aim is to transfer policies learned one or several teacher Q-networks to a single student network via supervised regression. They also proposed the idea of hierarchical prioritized experience replay which is a variant of PER adopted for multi-task environments. Spector et al. [24] introduces a variant of data augmentation through automatically generated game sets (by changing colors and patterns) for robust training and then used transfer learning between these auto-generated game sets. Joshi et al. [25] describes an interesting approach to transfer learning in RL. They propose an algorithm that auto-adapts the source policy using inter-task mapping to the target space, assuming that the source and target domains are compatible with each other.

Curriculum Learning [36] in Reinforcement Learning is a relatively unexplored topic. Graves et al. [38] explored automated curriculum learning in neural networks using progress and reward signals which closely mirrors reinforcement learning rewards. A very recent paper by Weinshall et al. [37] explores curriculum learning using

transfer learning. The relative difficulty of training examples is automatically graded based on knowledge transfer from another "teacher" classifier. Narvekar et al [39] formulated the design of an automated curriculum as a Markov Decision Process that models the accumulation of knowledge by the agent as it learns through a sequence of tasks and the cost of learning a task relative to the agent's current ability. Svetlik et al [40] follows along the same lines automatically generating a curriculum as a directed acyclic graph which adapts based on the agent's ability.

Other approaches to Deep Reinforcement Learning include policy gradients which directly optimize in the policy space and are explored in Minh et al. [32]. Actor-Critic models are examples of such models. Imitation Learning where the network is trained to imitate the behavior of an expert in the domain, be it another algorithm or a human expert, is also shown to be very effective and is explored in [29], [30] on different 3D games.

III. Approach

A. Reinforcement-Learning Topics

The general objective of reinforcement learning is to train an "intelligent agent" capable of interacting with a dynamic environment and make good decisions. The agent makes 'actions' in the environment and this results in a change in the 'state' and a 'reward' from the environment. There are primarily two approaches for this task: Q-Learning methods and Policy Gradient (PG) methods. Q-Learning tries to learn the value/measure of being in a given state and based on the value metric, take subsequent actions. PG method on the other hand, attempts to learn functions which can directly map an observation to an agent action [6]. PG will directly optimize in the policy space [5] while Q-Learning will first approximate the Q-function and then use that to infer the optimal policy. The effectiveness of both the methods are widely debated [32]. We however will be focusing our attention on Q-Learning for the rest of this report.

$Q^{\pi}(s,a) = E[R_t]$ (The expected value of sum of future rewards R_t following policy π) (1)

$R_t = r_t + \mu r_{t+1} + \mu^2 r_{t+2} + \cdots$ (μ represents the discount factor, a hyper-parameter) (2)

Q-learning tries to approximate this Q^{π} function. The input is a stream of data points $<s,a,r,s'>$ where s represents the initial state, a denotes the action taken by the agent, r is the reward given by the environment and s' represents the next state after performing action a. Here π denotes the policy which can be simply thought of as a guide followed by the agent, given the states. (The end goal of Reinforcement Learning is for the agent to learn this policy given an environment).

Once we have a good estimate of Q function, finding policy π at a state s just becomes:

$$\pi(s) = \text{argmax}\, Q(s,a) \quad (3)$$

We use Bellman equation to obtain the estimate of Q function from the input $<s,a,r,s'>$:

$$Q^{\pi}(s,a) = r + \mu * \max_{a'} Q'(s',a') \quad (4)$$

We try to minimize the difference between LHS ($Q^{\pi}(s,a)$: the prediction term) and RHS over time ($r+\mu*\max_{a'}Q'(s',a')$: the target term).

B. Deep Model Descriptions

When we use neural networks as the Q-function approximator we have entered the realm of Deep Q-Learning Network (DQN). DQN aims to reduce the following Loss (L):

$$L = \tfrac{1}{2} * [r + \mu * \max_{a'} Q(s',a') - Q(s,a)]^2$$ (MSE-error b/w Target and Prediction terms) (5)

However, just using DQN often doesn't lead to satisfactory results. Instead, a close variant called DDQN (Double DQN) is preferred. This leverages two neural networks: One for generating the prediction term and another for the target term. Basically, DQN introduces some inaccuracies at the beginning of the training and these will increase bias the system [6]. The vanilla Q-learning algorithm implementation sometimes learns unrealistically high action values under certain conditions (because of the maximization step over action values in eqn. (4)). DDQN decouples the selection of action and the evaluation into two separate steps [4]. Though DDQN is not always guaranteed to give better performance [7], it will stabilize the network and thus is better suited for complex tasks. DDQN thus changes the Bellman eqn. (4) to

$$Q^{\pi}(s,a) = r + \mu * Q'(s', \text{argmax}_{a'} Q'(s',a')) \quad (6)$$

Dueling DDQN is another improvement. Here also, two independent networks are used, one for state value functions and another for state-dependent action advantage function [15]. This is considered one of the best networks for playing games currently. We adopted the base model for Dueling DDQN from [6].

C. Experience Replay and Exploration

Using an online algorithm like neural network directly, presents us with 2 problems: the datapoints $<s,a,r,s'>$ are highly correlative and thus directly learning from them in a sequence will cause it to overfit [7]. Also updating after

every step is very inefficient in terms of training time complexity and can also lead to high variance. Another issue is that, this experience $<s,a,r,s'>$ is immediately discarded after it is used. Both these drawbacks can be overcome using experience replay [28]. The idea is to store these experiences in the network's memory and randomly sample from them during each learning step (upon which the neural network's gradient descent on the current batch will kick in). The size of the memory is a key hyper-parameter deciding the performance. The bigger the memory the robust the learning, but the higher the space required by the model.

One improvement over this vanilla experience replay, is prioritized experience replay [4]. The intuition is that not all experiences are equal, and we want to learn more from experiences that don't align with our current estimate of Q function. However, instead of just learning from the top experiences as suggested in [4], we instead opted to implement a variant described in [7]. Here we don't just pick the most rebellious experiences always, but instead assign weights to each experience (proportional to the error) and do weighted sampling. The error is defined as:

$$\text{Error} = Q(s,a) - \text{Target}(S) \quad \text{and weight} = (\text{error} + \varepsilon)^{\alpha} \tag{7}$$

This is calculated (and stored) during the forward pass. Although some extra computations are required for this Error calculation, the main trade-off is space. Using PER incurs a sizable overhead on memory being used.

Another point to be noted is the exploration policy. Instead of always greedily picking the actions we do ε–greedy policy. By acting greedily, we might never find out that some actions are innately better than others (especially in the beginning of the training process). This policy picks a random action with probability ε and rest of the time it is greedy. We also perform a linear decay on this ε as the training process continues (we can also add in an explicit epoch limit after which ε is made 0).

D. *Doom Configurations*

Doom has many game scenarios and we leverage the VizDoom framework to put our agent in different environments. The config. files provide us some flexibility in that, we can specify what actions can be taken by the player and how skillful the opponent monsters are. Although some rewards are hardwired to individual levels, we can modify parts of the rewards as we see fit before training the neural network. (explained in the next section).

Figure 1. Doom's first-person perspective

We used the following configurations:

1) Basic: The player is placed in a small rectangular room (along the longer wall) and at the other end a stationary monster (who does nothing) is placed. There is a reward of +101 for killing the monster and -5 for each missed shot. (Agent is expected to learn shooting a monster using this level).

2) Defend the center: The player is placed at the center of a big circular arena. Enemy monsters are spawned randomly near the wall. The monsters move and also attack the player when he's in range (melee attacks). There is a reward of +1 for killing a monster. (The agent is expected to learn shooting moving targets while defending himself).

3) Deadly Corridor: The player is placed at the end of a long corridor consisting of 4 rooms. At the other end of the corridor, is a vest and when it is collected the game ends. Each room has 2 monsters who try to shoot and kill the player. The reward is specified as the distance to the vest at the other end of corridor when the game episode ends. (The agent is expected to learn shooting while navigating and also while completing an objective).

4) Deathmatch: This is the most general case of the game where there is a complex map (with rooms) and agent must try to kill as many monsters as he can, while conserving his health. Agent can also pick up ammo and health points. The score for a game episode is defined as number of monsters killed before the player dies but can also be modified to include a weighting for the time the player is alive.

As one can see, the difficulty of each level increases.

IV. Experiments

Each game level is modeled as a Markov Decision Process and as mentioned above, Q-learning is used to learn the policy. The action is selected using an ε-greedy policy,

with a linear ε-decay (ε starts from 1.0 and decreases to 1e-4 linearly). The convolution network used to estimate Q function is trained with Stochastic Gradient Descent (using 'adam' optimization method). When random weights are needed, we use He_normal initialization [35]. We use 5 epochs each with 1000 episodes (A game episode ends when the agent is killed or if there is a timeout. The timeout is a configurable option [Eg- basic level was set to timeout after 350 sec]). For the 'basic' setting, the memory size was set to 10,000 (since the level is relatively easy, even with such a high capacity we do train fast) and for the other three settings, the memory capacity is set to 100 (these are complex tasks can can take upto 9 hours for all 5 epochs to complete).

To make comparisons more apparent, we used the same neural network architecture in all the tasks. The 'out_dim' in Figure 2 corresponds to total number of actions allotted to the agent in that game and input corresponds to 4 frames of the game (preprocessed and then stacked). The learning rate is kept constant throughout at 1e-4.

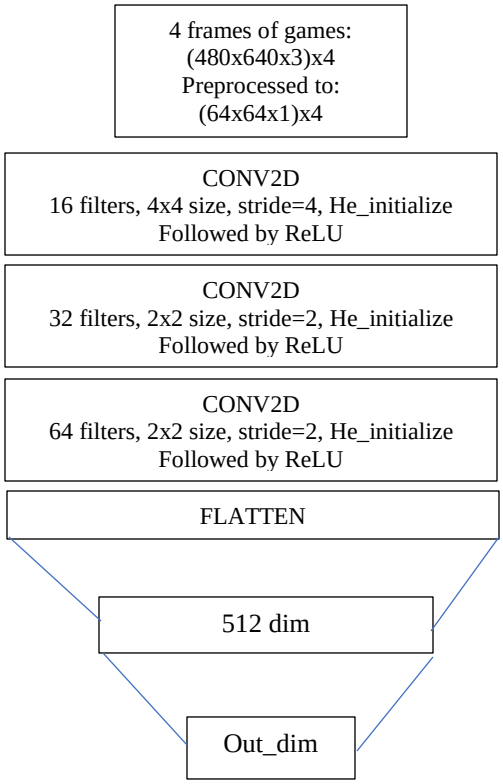

Figure 2. DQN architecture

We have also made some slight modifications to the rewards sent into the network (Note that rewards given by the game environment are hard-wired according to the level). Each bullet fired reduces the reward by 0.08 and each health point reduction reduces reward by 0.1.

A. Phase-1 results:

In phase-1 we compare networks with and without Priority Experience Replay (PER). The Y-axis is the scores (i.e., total game reward returned by the environment) per episode. The scores include factors like, number of monsters killed, time the agent survives etc. and varies from game to game. X-axis is the number of game episodes. (The same labels are used for all plots in the report).

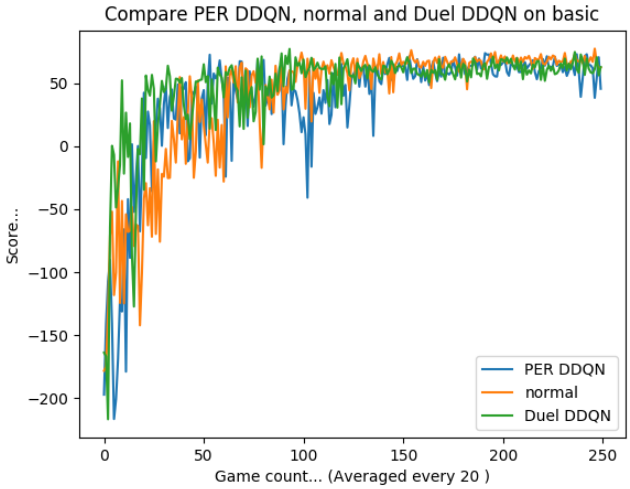

Figure 3. Compare PER DDQN, vanilla DDQN, Dueling DDQN on basic

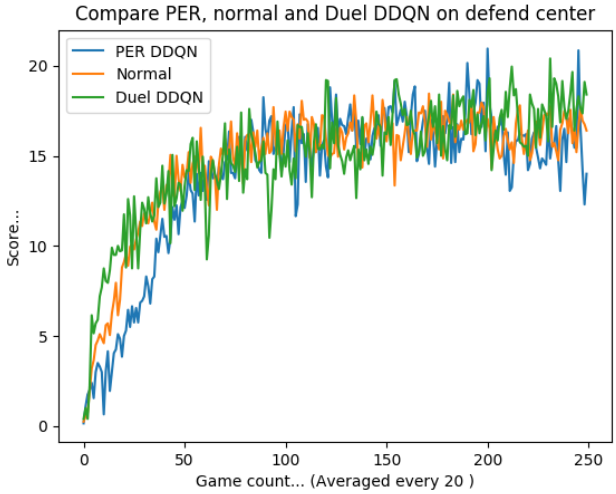

Figure 4. Compare PER DDQN, vanilla DDQN, Dueling DDQN on defend the center

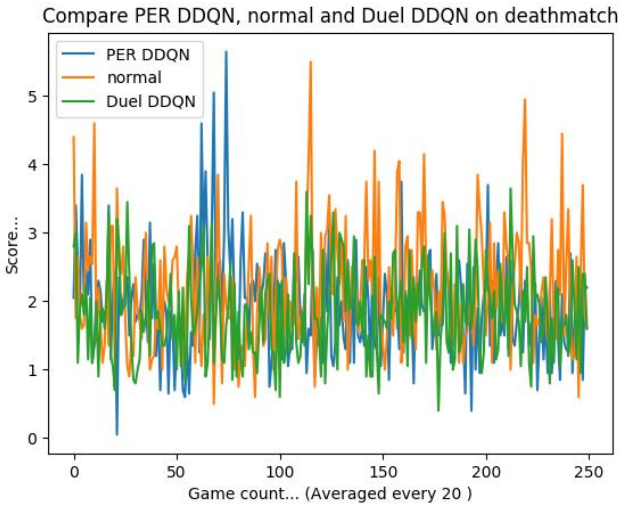

Figure 5. Compare PER DDQN, vanilla DDQN, Dueling DDQN on deathmatch

In the above graphs, Vanilla DDQN is orange, DDQN with PER added is blue and dueling DDQN with PER is green. We see the trend that models with PER achieve high scores relatively fast compared to the model without PER (hence the tall blue and green spikes). This is expected because of the sampling PER induces. The model pro-actively searches for tougher experiences and learns from them. But this is a double-edged sword. On adding PER, the variance tends to increase (more wiggly spikes) compared to vanilla DDQN (The orange curve is much smoother). We believe the reason for this is that always learning mainly from poor memories could be misleading, especially in the initial stages where the agent does not have a clue about how things work and why it is receiving the rewards it is receiving. A potentially better method would be a hybrid, where we learn from both episodes where it performed worse (learns what actions not to do) and episodes where it performed well (learns what actions to do).

Another trend we notice is that adding PER will help in getting high scores at a much lower game episode count as compared to vanilla (most evident in case of dueling networks with DDQN; the green curve). This confirms our claim. (Though we expected a much higher difference than the ones shown in the plots). One thing not evident from the graphs is the training time. Since PER needs weights to be re-calculated after each action due to updates in memory, and because we also need to store these subsequent priorities, this method is both compute and memory intensive. It takes a bit longer to train in comparison with Vanilla DDQN.

B. Phase-2 results:

Before we go to curriculum learning using transfer learning to verify that we see positive results as we expected. We trained a PER DDQN network (green) from scratch (random weights) on the "basic" scenario (This will serve as a good baseline). Another PER DDQN network was trained from scratch on the "defend_the_center" scenario. Then the model was loaded into the "basic" scenario. Using this as a starting a point, we then experimented with two approaches: Pre-train (full end-to-end training) and Fine-tune (We froze the Convolutional layers and allowed only the Fully Connected Layers to update their weights).

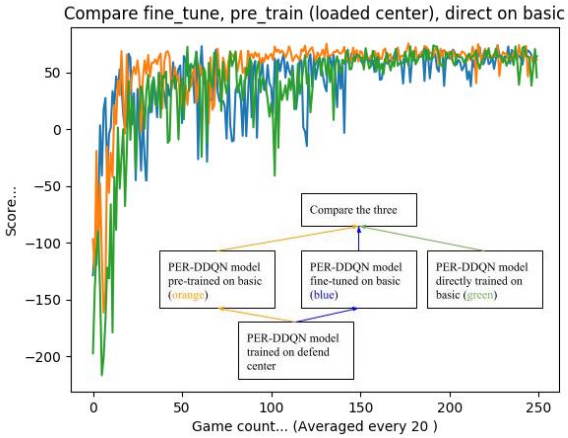

Figure 6. Compare “fine_tune” net, “pre-train” net and normally trained net for "Center" - > "Basic"

We see that, using the model from "center" as a launching pad is definitely helpful. Both "fine_tune" model (blue), "pre_train" model (orange) converge faster than the model starting from scratch (green). It is also evident that fine-tuning converges faster than pre-training, because the initial layers are frozen in the former case. In terms of training time too, "fine-tune" network trained much faster than "pre-train" network. But, "pre-train" network does better than "fine-tune" on average (there are more episodes in orange curve with high scores). Since all three models are using PER sampling, the variance remains high.

We adopted the following methodology for curriculum learning was:

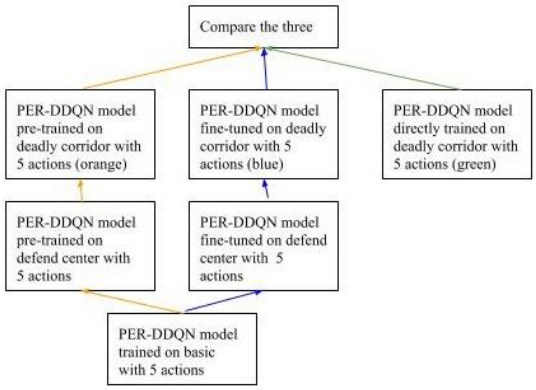

Figure 7. Roadmap for curriculum learning experiment

We trained a PER DDQN network (green) from scratch (random weights) on the "corridor" scenario (Our models from curriculum learning will be compared against this). Another PER DDQN network was trained from scratch on the "basic" scenario. Then the model was loaded into the "center" scenario and then finally loaded into "deadly_corridor" scenario. The results were as follows:

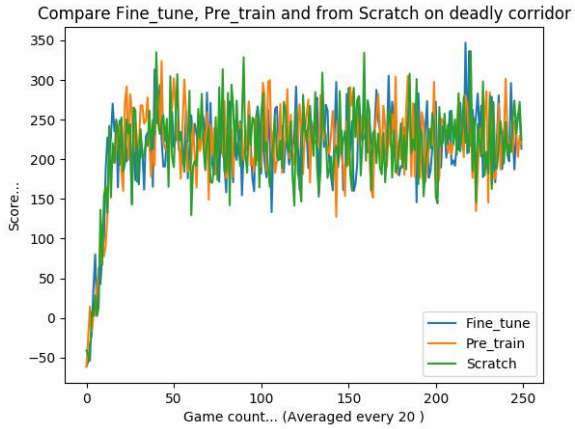

Figure 8. Compare "fine_tune" net, "pre_train" net and normally trained net for "Basic"->"Center"->"Corridor"

In Figure 8 above we see that, the models trained using curriculum training give comparable results considering game scores as the metric. We did observe that the models (blue, green) trained using curriculum learning trained faster than the model from scratch. We could not provide quantitative measures because these models were run on systems with slightly unequal configurations. However, faster training time for this particular scenario "deadly corridor" means better performance. For this scenario, nothing can be inferred about the training time from the results graph. In other modes, we can do that. For example, in the "center" scenario, a high score indicates the game lasted longer compared to a low score episode, this is because the higher the score, the more monsters the agent shot down and longer it survived, thereby increasing the episode time. Hence, the longer the training time, the better the model performance. However, for this scenario ("deadly corridor"), no correlation exists between training time and the results in Figure 8. The objective is to pass a deadly corridor and reach a vest before dying, while enemies are shooting at the player. The rewards are based on the agent's movement w.r.t the vest i.e., positive reward is obtained by moving towards the vest and negative reward by moving away from it. Obviously, reaching the reward faster is a better performance than reaching the reward later even though they both effectively have the same game scores.

V. Future Work and Conclusion

A. *Possible avenues for improvement*

RL takes a lot of time to train. If we had more time or specialized hardware, we would have loved to pursue the following avenues:

1) Try an adaptive variant of PER, where initially the agent focuses on the samples where it has performed well to learn how to get positive rewards and gradually (once it has learned enough to play the game at a satisfactory level) move towards bad experience so as to learn from its mistakes (and try to rectify them). This type of learning closely mirrors the human way of learning any specific task (especially so in game-playing). To bring down the space and time overhead, data structures like heap/prioritized-queue can be employed.

2) One limitation in our approach is that, the actions must be same across all the game levels. We did explore a variant where we loaded the weights from a model and just changed the final FC layer to accommodate different number of actions. We hoped the probabilities would distribute properly among these new set of actions but this approach suffered. Making the curriculum learning approach work when (i) action space increases and (ii) action space changes are both important paths on which future work can be done.

3) We initially also wanted to test on different games (as mentioned in the milestone, we explored lot of Open AI gym initially working on ATARI games like Cart Pole, Pong). But we faced trouble integrating the same model on both VizDoom and Open AI Gym and we ran into installation issues due to hardware limitations and proprietary libraries. We reasoned regarding this, stating that different game levels in Doom are, strictly speaking, different games and thus our methods should, more-or-less, return the same results as above on different games too (which are related and not completely different). However, we would like to test our model on different games and confirm our hypothesis.

4) We believe multi-task learning could be used to overcome the limitation mentioned in (2). Take pre-training

for example. The initial weights trained for "basic" will inevitably be changed by the next scenario "defend_the_center". But for the final scenario "deadly corridor", the intuition of find and shoot an enemy from "basic" is needed. Multi-task training is known to allow the model to retain knowledge across many tasks without suffering a performance penalty on source tasks [23]. Sharing convolutional layers across tasks and having different dense layers for each task (as shown in Figure 9) might be a promising idea. We had trouble integrating the networks from different games (given the constrained time) but it is definitely something we would like to try out in the future.

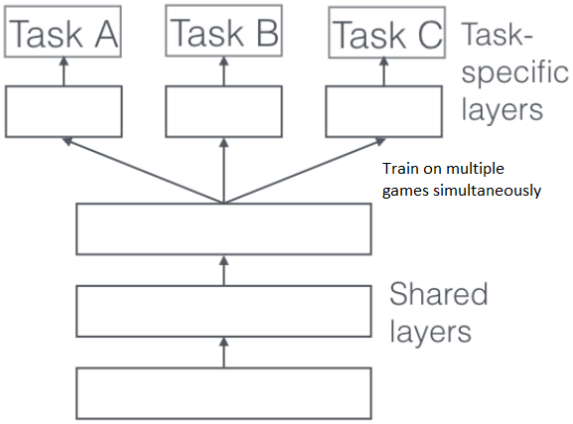

Figure 9. Simultaneous training for multi-task learning [23]

5) Since our aim was comparing networks and exploring curriculum learning and given the notoriety of RL in terms of training time, we used a simplistic neural network architecture for each task in the project. One can certainly improve the base DQN by adding more layers, using Batch Normalization (if required), incorporating dropout techniques etc. Dropout for example is known to improve the accuracy for deathmatches [13].

6) As touched upon in relevant work, there are DQNs using RNNs (the so-called DRQNs). We actually wanted to present results comparing various DRQNs by changing the inner base (vanilla RNN, or a LSTM or a GRU) and suggest improvements (if possible) in a new phase (phase-3!) of this same report. We have actually written the code and ran individual tests (found on github). As expected, using sequences gave us slightly better results (the agent can now remember where the monster is. In a scenario like "defend_the_center", this information is invaluable). However, we had to scrap off this idea because these recurrent networks were taking much too time to train.

7) As mentioned above, Policy Gradients (PG) is another alternative to Q-Learning often used in Reinforcement Learning. We did some work on it early on, (mentioned in the milestone), but then didn't have the time to pursue it. This is an important path to work on, since it is often hypothesized that PG is a much better method when it comes to general AI tasks. Both PER and our curriculum learning approach is highly adaptive and we hope both of them will prove effective on policy gradient networks too.

8) In this paper, we manually selected the curriculum incorporating incremental tasks. But automated subtask generation as in [38 - 40] which can adapt to the agent's current ability would likely lead to decrease in training time.

B. *Learnings*

We used variants of DQN to play the game of Doom. We saw that Priority Experience Replay (PER) was very effective on DQNs and boosted their performance. Since it is not directly coupled with the RL model itself, we hypothesize that it will blend in well with Policy Gradient models too. We also proposed few improvements to the way PER can be improved. In the next phase, we explored curriculum learning in the context of Reinforcement Learning. This, we believe, is a natural fit to game-playing because that is how humans learn. The total training time is reduced with almost no perceptible loss in game scores. We also proposed different ways in which curriculum learning can be implemented so that it can be further generalized. Finally, we think curriculum learning in the context of RL will open many new avenues for agents to play better.

C. *Code Repository*

Github Repo

REFERENCES

[1] C. Asawa, C. Elamri and D. Pan. Using Transfer Learning Between Games to Improve Deep Reinforcement Learning Performance and Stability. URL: https://pdfs.semanticscholar.org/4a55/2a11f7cfe38cc06b516fa4e7e48a2b489692.pdf. 2017.

[2] T. Schaul, J. Quan, I. Antonoglou and D. Silver. Proioritized Experience Replay. arXiv preprint arXiv:1511.05952v4. 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. Playing Atari with Deep Reinforcement Learning arXiv preprint arXiv:1312.5602v1. 2013.

[4] H. van Hasselt, A. Guez and D.Silver. Deep Reinforcement Learning with Double Q-learning. arXiv preprint arXiv:1509.06461v3. 2015.

[5] A. Karpathy. "Deep Reinforcement Learning: Pong from Pixels". Andrej Karpathy blog. URL: http://karpathy.github.io/2016/05/31/rl/. 2016.

[6] F. Yu. Deep "Q Network vs Policy Gradients - An Experiment on VizDoom with Keras". Felix Yu blog. URL: https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html. 2017.

[7] J. Janisch. "LET'S MAKE A DQN: DOUBLE LEARNING AND PRIORITIZED EXPERIENCE

REPLAY". Jaromir's Blog. URL: https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/#fn-444-2. 2016.
[8] C. Clark and A. Storkey. Teaching Deep Convolutional Neural Networks to Play Go. arXiv preprint arXiv:1412.3409v2. 2015.
[9] N. Justesen , P. Bontrager , J. Togelius , S. Risi. Deep Learning for Video Game Playing. arXiv preprint arXiv:1708.07902v2. 2017.
[10] Silver et al. Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.
[11] Silver et al. Mastering the game of go without human knowledge. Nature, 550:354– 359, 2017.
[12] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In IEEE Conference on Computational Intelligence and Games, 341–348, 2016.
[13] G. Lample. and D. S. Chaplot. . Playing fps games with deep reinforcement learning. In Thirty-First AAAI Conference on Artificial Intelligence. 2017.
[14] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath. A Brief Survey of Deep Reinforcement Learning. arXiv preprint arXiv:1708.05866v2. 2017.
[15] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot and N. de Freitas, Dueling Network Architectures for Deep Reinforcement Learning. arXiv preprint arXiv:1511.06581v3. 2015.
[16] Bhatti et al. Playing Doom with SLAM-Augmented Deep Reinforcement Learning. arXiv preprint arXiv:1612.00380v1. 2016.
[17] S.Alvernaz and J. Togelius. Autoencoder-augmented Neuroevolution for Visual Doom Playing. arXiv preprint arXiv:1707.03902v1. 2017.
[18] Hessel et al. Rainbow: Combining Improvements in Deep Reinforcement Learning. arXiv preprint arXiv:1710.02298v1. 2017.
[19] A. Rosenberg and G.Somappa. Pixel to Pinball: Using Deep Q Learning to Play Atari. URL: http://www.cs.virginia.edu/~gs9ed/reports/Reinforcement_Learning_Paper.pdf.
[20] R. Chuchro and D.Gupta. Game Playing with Deep Q-Learning using OpenAI Gym. URL: http://cs231n.stanford.edu/reports/2017/pdfs/616.pdf. 2017.
[21] Nair et al. Massively Parallel Methods for Deep Reinforcement Learning. arXiv preprint arXiv:1507.04296v2. 2015.
[22] H. Yin and S. J. Pan. Knowledge Transfer for Deep Reinforcement Learning with Hierarchical Experience Replay. In Thirty-First AAAI Conference on Artificial Intelligence. 2017.
[23] S. Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. ArXiv preprint arXiv:1706.05098v1. 2017.
[24] B. Spector and S. Belongie. Sample-Efficient Reinforcement Learning through Transfer and Architectural Priors. arXiv preprint arXiv:1801.02268v1. 2018.
[25] G. V. Joshi and G. Chowdhary. Cross-Domain Transfer in Reinforcement Learning using Target Apprentice. arXiv preprint arXiv:1801.06920v1. 2018.
[26] Andrychowicz et al. Hindsight Experience Replay. arXiv preprint arXiv:1707.01495v3. 2018.
[27] G. Tesauro. Temporal difference learning and TD-Gammon. Machine Learning, 8, 1992.
[28] LJ Lin. Reinforcement Learning for Robots Using Neural Networks. Ph.D dissertation in CMU. 1993.
[29] Harmer et al. Imitation Learning with Concurrent Actions in 3D Games. arXiv preprint arXiv:1803.05402v3. 2018.
[30] S. Ross, G. J. Gordon and J. A. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. arXiv preprint arXiv:1011.0686v3. 2011.
[31] M. Hausknecht and P. Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. arXiv preprint arXiv:1507.06527v4. 2015.
[32] V.Minh et al. Asynchronous Methods for Deep Reinforcement Learning. arXiv preprint arXiv:1602.01783v2. 2016.
[33] K. Adil et al. Training an Agent for FPS Doom Game using Visual Reinforcement Learning and VizDoom. In International Journal of Advanced Computer Science and Applications, Vol. 8, No. 12. 2017.
[34] D. S. Ratcliffe, S. Devlin, U. Kruschwitz and L.Citi. Clyde: A Deep Reinforcement Learning DOOM Playing Agent. In AAAI Publications, Workshops at the Thirty-First AAAI Conference on Artificial Intelligence. 2017.
[35] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385v1. 2015.
[36] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, 41–48. 2009.
[37] D. Weinshall, G. Cohen and D. Amir. Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks. arXiv preprint arXiv:1802.03796v2. 2018.
[38] A. Graves, M. G. Bellemare, J. Menick, R. Munos and K. Kavukcuoglu. Automated Curriculum Learning for Neural Networks. arXiv preprint arXiv:1704.03003v1. 2017.
[39] S. Narvekar, J. Sinapov and Peter Stone. Autonomous task sequencing for customized curriculum design in reinforcement learning. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017.
[40] M. Svetlik et al. Automatic curriculum graph generation for reinforcement learning agents. In Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI). 2017.