Solution and Experimental Analysis of Open AI "Lunar Lander" using Deep Q Networks with Experience replay

Ashish Panchal *OMSCS Student - Dept. of Computer Sciences*
Georgia Institute of Technology, Atlanta, USA
apanchal33@gatech.edu and ashupanchal.007@gmail.com
Git hash:799e29b4cf63d999b182382a580af9c4d999153c

***Abstract*—Open AI Gym Lunar Lander environment was solved using of Deep Q Network non linear function approximation. Process of convergence along with "Skip Step learning","Target network" and "experience replay" was studied in isolation to reduce the instabilities in the DQN caused by Bootstrapping, off-policy function approximation and model less nature of the algorithm. It was Deduced that when selectively used, these tools can reduce the effect of Deadly triad.**

***Index Terms*—Lunar Lander, DQN, Reinforcement Learning, Generalization.**

I. Introduction

In our everyday life we try to plan our actions to achieve certain goals however big or small. Defining a "policy" for the same however is a daunting task as situations arising every day are unique and may not repeat. For example, in a soccer game, both teams try to score the highest goals. Defining exact position of each player, intensity of each kick and planning every dodge and dribble is not a feasible approach. Therefore, generalization is employed and, rules and policies are approximated based on observations. As time passes and experience grows a more holistic set of rules is developed, which can prepare players for new scenarios and enable them to take an optimal action.

In the following sections theory of experience based reinforcement learning methods are covered along with generalization of experience, dilemma of exploration and exploitation to balance learning from experience and applying the knowledge gathered to take best action. Generalization, experience and exploitation with the example of Lunar lander Deep Q algorithm is explored along with experimental and analytical overview of solution procedure and results. Since DQN is a non-linear approximator, it does not guarantee an optimal policy. The observed pitfalls and extended experimental results are discussed to understand the behavior of approximators and subsequently agents.

This paper is limited to understanding the concept through the example of Lunar Lander problem and does not exhaustively cover optimization methods or alternative approximators in much detail.

II. Background

A. Reinforcement learning method

An action can be classified as correct or incorrect based on associated positive or negative reinforcement. With experience better approximation can be made to identify the action with the highest rewards. Subsequently actions can be planned to maximize collective return. A learner can limit learning from experience to avoid incorrect action and ensure steady movement toward the goal. This cognitive method is used by reinforcement learning agents to identify optimal actions, given a situation of state of environment. However as mentioned in introductory example, real-world situations are a challenge for reinforcement learning methods like MDP and Temporal Difference learning as exhaustive definition of possible states need to be defined to determine optimal policy. Due to continuous feature environment such a lookup is not possible and model fails to produce a feasible policy.

B. Computational overview of Reinforcement Learning

Every state s is associated with a value $V_\pi(s)$, in terms of total discounted return, achievable following a policy π till the target is reached, in case of episodic task and asymptotically when the task is continuous.

$$V_\pi(s) = \oplus_{(a|s)}(R(s,a) + \gamma \otimes_{(s'|s,a)} (V_\pi(s'))) where\ a \in A(s) \quad (1)$$

Generally the policy as well as associated value function are not known and therefore estimated with experience and observations, $\hat{V}_\pi(s_t)$ with some is initial assumptions about value functions for all the states, where V at terminal = R at terminal. Where we try to reach $V^*_{\pi^*}(s)$ as the optimal value function at optimal policy. The definition of optimal may vary with respect to expectation from the an acting agent, such as pessimistic, exploratory etc, where for a given policy $V^*(s) \geq V(s)$ for all s. As the Bellman's equation, (1), is a contraction mapping [**?**], i.e , with recursive update it brings the values closer to the optimal,given function $\oplus$ and $\otimes$ are contraction mapping such as $max()$ or $mean()$. Value can also be represented for a state action pair as $Q(s,a)$,(2)

$$Q_\pi(s,a) = R(s,a) + \gamma \otimes_{(s'|s,a)} (\oplus_{(a'|s')}(Q_\pi(s',a'))) \quad where\ a \in A(s) \quad (2)$$

one way of choosing $\otimes$ and $\oplus$ is to take a stochastic average over all the next states resulted from action a from the s and choose the maximum value yielding action a' in the next possible states s'. This method is called Q learning, (3).

$$Q_\pi(s_t,a_t) = R(s,a) + \gamma \sum_{(s'|s,a)} p(s'|s,a). \max_{(a'|s')} (Q_{\pi'}(s',a'))) \quad where\ a \in A(s) \quad (3)$$

Q learning is a off policy methods, i.e $Q_\pi(s_t,a_t)$ is not dependent on immediately next observed state, s_{t+1}, rather is dependent on s' which may not follow the current policy. However, since the defining equation is a contraction mapping, it converges to Q^* and π^* on update, defined as follows

$$Q_\pi(s_{t+1},a_t) = Q_\pi(s_t,a_t) + \alpha\big(R(s,a) + \gamma \sum_{(s'|s,a)} p(s'|s,a). \max_{a'|s'}(Q_{\pi'}(s',a')) - Q_\pi(s_t,a_t)\big) \quad where\ a \in A(s) \quad (4)$$

Additionally, if the Q^* is know for every s' achievable from all the a at s, therefore Q^* at s can be estimated, from (2)

C. Exploration, Exploitation and Generalization

In Q-learning update is dependant on historic best state and action (bootstrapped method). It is also dependent on value function of the next states and action not following the current π , it requires every state visited, which asymptotically converges to Q^* and π^*. For a finite deterministic state and action space, an active agent ensures balance between exploration and exploitation, by following policies like ϵ-greedy [1] where the agent takes stochastic actions with a probability 1-ϵ or deterministic greedy actions to maximize returns. With experience agent's knowledge of environment increases and it attains better values. There are alternative methods like Rmax,Thompson sampling, Boltzmann Exploration, etc which are not explored in this report.

Q learning agent needs to visit infinite states for converging to Q^* for continuous state space. However, the value of a states un-visited can be approximated based on visited states,prior experience, to select actions. A states can be represented as a collection of aspect defined as numerical features, e.g. state s = $[1, 0, 0, 0, 2, 2]$, for a 5 feature environment. An i^{th} feature may have different contribution w_i in defining $\hat{V}(s)$, representing the approximated Value as $\hat{v}(s, w)$ at s , i.e. $\hat{V}(s) \approx \hat{v}(s, w)$ or $\hat{Q}(s, a) \approx \hat{q}(s, a, w)$. Number of weights are generally less than the states in continuous space, therefore defining variable of environment, is shared with the other states. Changing one weight affects estimated value of many states, High accuracy in prediction for one state might lower accuracy for other and therefore, the accuracy should be favoured for more important states , defined in term of distribution $\mu(s) \geq 0, \sum_s \mu(s) = 1$. Accuracy can be understood as mean squared value error, calculated as per ((5)) at time t.

$$\overline{Error}(w_t) = \sum_s \mu(s)\big[Q_\pi(s, a) - \hat{q}(s, a, w_t)\big]^2 \tag{5}$$

where $Q_\pi(s, a)$ is the target value for the s, a, which is not available , therefore, assuming a deterministic underlying policy, by ((1)) and ((5)), the error can be estimated as follows:

$$\begin{aligned}\overline{Error}(w) &= \sum_s \mu(s)\big[R(s, a) + \gamma. \max_{(a'|s',a,s)} (Q_{\pi'}(s', a')) - \hat{q}(s, a, w)\big]^2 \\ &\approx \sum_s \mu(s)\big[R(s, a) + \gamma. \max_{(a'|s',a,s)} (\hat{q}(s', a', w_{t-1})) - \hat{q}(s, a, w_t)\big]^2\end{aligned} \tag{6}$$

with a goal to reach global optimal , w^* for which $\overline{Error}(w^*) \leq \overline{Error}(w), \forall w$, however with non-linear approximators like ANN where w define weights between neuron layers, its more common to reach local optima, which in most cases it enough. [1] Since w defines the nature of the Q , it can be adjusted to minimized the error, by a small amount, α, in the direction that would most reduce the error on on an observation. as per (II-C).

$$\begin{aligned}w_{t+1} &= w_t - 1/2\alpha\nabla\overline{Error} \\ &= w_t - 0.5\alpha\nabla \sum_s \mu(s)[Q_\pi(s, a) - \hat{q}(s, a, w_t)]^2 \\ &= w_t - \alpha \sum_s \mu(s)[R(s, a) + \gamma. \max_{(a'|s',a,s)} (\hat{q}(s', a', w_{t-1})) \\ &\qquad - \hat{q}(s, a, w_t)]\nabla\hat{q}(s, a, w_t)\end{aligned} \tag{7}$$

Approximating $\hat{Q}$ with least error does not guaranty π^*. [2] however, following to Q-learning convergence , [3], w aspires to converge towards w^* , and to procure near optimal policy. Approximators like Linear Approximators, provide convergence guaranty, where $V(s_t, w) = x_t.w$, x is the feature vector of s , but fail to learn conditional dependence among features [1]. Course and Tiling coding methods utilize concept of state vicinity and can work in continuous space and solving conditional dependence [5]. However they require precise tiles' size and shapes definition. Also degree of generalization is fixed during training. Best performance is possible only if generalization is gradually reduced over time [4]. Neural Network, addressing these issues, can generalize throughout the state space, defining different degree of specification for different group of states [6], which we have used with Deep Q-learning algorithm [12] to solve the Lunar lander problem.

III. Experiment Description, Analysis and Results

A. Lunar Lander (LL): Problem statement and Environmental Specifications

Lunar Lander environment is a low gravity, low friction 2D continuous simulation space. Lander with arbitrary start position at the top of the space, and by taking no action, firing its left, main or right engines, it controls its position, its speed in 2D, tilt and the angular velocity. Accumulating -0.3 and -0.03 for using main and side engine every time, respectively, with main generates 21x static power than the latter, along with current positions and orientation defining impulse amount for movement in the chosen direction, with small random dispersion, resulting in a new state.

Lander legs have small spring action to absorb the fall, prevent jump backs, however, large momentum, generating high torque in legs may results in crash. Each leg contact gets +10 points, equal negative points for contact loss. Safe landing gives +100 points, -100 for crashing with main body touching the ground, apart from agent crossing the screen boundaries, ending the game The lander aspires to reach the flat landing pad at 0,0 coordinates,extended to irregular terrain, It is penalized for moving away from the it as much as it is rewarded, between +100 to +140 points from top to the pad at zero speed, which is accumulated apart from the other described, at every transitional state as Reward(s_t) = $-100(d(s_t)-d(s_{t-1}))-100(v(s_t)-v(s_{t-1}))-100(\hat{\theta}(s_t)-\hat{\theta}(s_{t-1}))$ where $d(s_t)$,$v(s_t)$ and $\hat{\theta}(s_t)$ are, distance from goal, lander's velocity , angular velocity at time t, encouraging to lower the distance from goal,smoothly land by decreasing the speed at the same time reach the goal quickly, prevent rolling by minimizing angular speed, and avoiding take off post landing.

And therefore 8 features, $(x, y, \hat{v_x}, \hat{v_y}, \theta, \hat{\theta}, legL, legR)$, i.e. horizontal and vertical position, horizontal and vertical speed, angle, angular speed, indicator of left or right leg touching ground, respectively define the state space and guide the action policy, to attain at-least 200 points to solve the game on average for 100 runs.

B. Deep Q Network and Learning

1) DQN : Non - Linear Approximator: Neural Networks (NN) are among the most widely used approximators, having a collection of of finite units activated non-linearly, in hidden layers, which can approximate any continuous function to any degree of accuracy [CG9]. Hierarchical compositions of many lower-level abstractions layers, could auto-engineer appropriate features for a given problem without relying exclusively on hand-crafted features.

Weights or importance of each internal feature abstraction, w are adjusted based on prediction errors. TD error can be used to solve Reinforcement learning problem . To optimize network performance, each weight is updated based on the contribution in action value estimated as partial derivative of an objective function/ Error with respect to each weight, based on the current w. Back-propagation method could translate the over all error to each layer after the experience with the help of optimizer algorithms, controlling the rateα if change, such as SGD , provided the units have differential activation. Only 2 hidden layers, with 128 units each (with units activated with rectifier nonlinearity, LeakyRELU, for granular transformation and approximation) were used to avoid slow learning from rapid gradient decay and over-fitting. 8 state features were ingested at input layer, producing 4 action values as output, one to estimate the optimal action value of each possible action. Stochastic Gradient

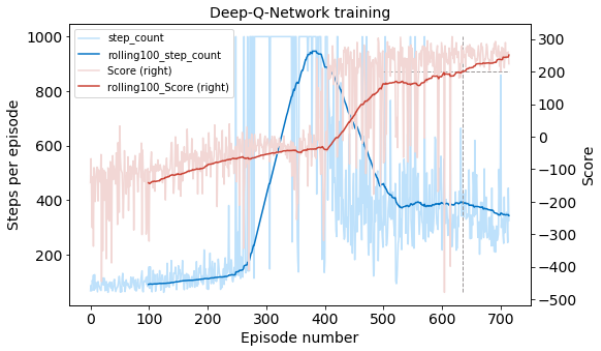

Fig. 1: *[DQN model training, depicting different stages of learning Lunar Landar agent, with scores and along with step count per episode, and 100 eps. rolling avgs]*

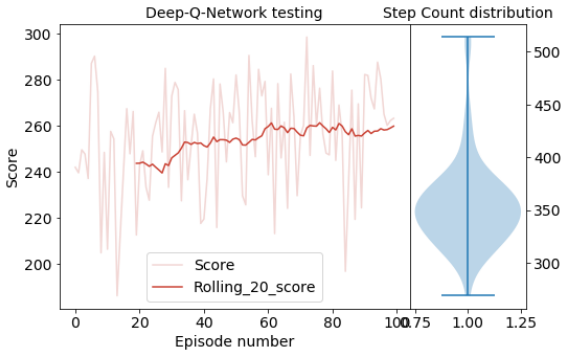

Fig. 5: *[Test run of trained DQN model over 100 randomized LL runs,depicting per episode and rolling 20 score, along with step count distribution per episode]*

descent (SGD), updates on randomly batches of data, generalizes well, but may get stuck in local minima as the network increases. Adaptive learning rate methods, such as Adam, better escape sub-optimal states and hence converge faster. Improving generalization performance by switching from Adam to SGD[IGP] demonstrates that adaptive optimization techniques generalize poorly compared to SGD. Large gradient weights (or fluctuating), will have larger accumulation. These w are regularized less than w with small and slowly changing gradients, which can be overcome with decoupling the weight decay from the gradient update, as in AdamW [10] [13]. α = 0.00005, with AdamW were used for the base model.

2) The Deadly triad (DT): DQN is a *model free, bootstrapped , function approximation, off-policy* [11]learning method, i.e. the model of the environment is not known and its estimated based experiences generalizes to represent all the states based on associated environment defining features and their contribution, not strictly depending on the sequence experience, rather best historical visited next states.Bootstrapping enables faster learning with recognition of encountered state, Function approximation can estimate any complex relationship and un-visited states, off-policy enables learning from multiple possible optimal policies simultaneously. Q learning and function approximation updating value at one state however creates a risk of incorrectly changing values of other states, if the agent is learning off-policy, These bootstrap values may not be updated often, biasing updates of other states based on less visited Bootstrapping states, leading to divergence of the w [8], hence been called "the deadly triad".

3) Stabilizing the network: **Experience replay:** To avoid bootstrapping based on less visited states, a dynamic replay memory of limited size d, (d=10000, used in base model), storing agents experience, $(s_t, a_t, r_{t+1}, s_{t+1})$ along with completion flag D,used to randomly choose a batch of experience to update w, ensuring learning from different state experiences.

Supported by off-policy nature of the Q learning, instability is removed due to the deadly triad, **Target Network:** An off-line bootstrapping process, along with exploration, , may yield sporadic updates, resulting in fast divergence. This effect can be reduced with the help of external reference for bootstrapping, therefore a target network updated once for N base/active network update, with a copy of base w. therefore using $\overline{q}(s', a', w_{t-1})$ from target network instead of $\hat{Q}(s', a', w_{t-1})$ in (II-C). For the LL problem base model, N is kept at 1, however further explored in IV .

4) Learning Procedure: Initialized to random weights w and state s_t, the DQN selects an action a_t based on decaying ϵ-greedy policy, as per II-C, staring ϵ at 1 it decreased ϵ till 0.01, executed by the game emulator, returning a reward r_t and the next state features s_{t+1} and completion boolean flag D, appended to replay memory and forgetting the oldest memory if records exceed d. After C= 3 transitions,IV-A, batch of random 64 historical records is selected to generate state action values, and compared with target network's greedy action value prediction , discounted at γ = 0.99 for low-biased Q_{π_t} estimate, to calculate Error, (6). Gradient is computed for each weight with backpropagation and updated with ADAMW optimizer instead of SGD, accelerating learning by adjusting the step-size parameter for each w based on a running average of the magnitudes of recent gradients for that weight. For stability the weight of active network were copied to the target network after N updates to the former. Sum of all rewards, for each episode is stored. Above procedure is repeated, until the average score $\psi \geq 250$, for last 100 episodes, for extended analysis, where the goal score was 200.

C. Results

DQN solved the LL environment, in 714 episode and ψ= 200 in 635.

1) Agent training behaviour:: With model initialized at ϵ =1, enabling exploration ,-ve scores were accumulated with more random actions leading to frequent crashes. As ϵ decayed and stabilized to 0.01 near 300 episode, more greedy action were taken by the

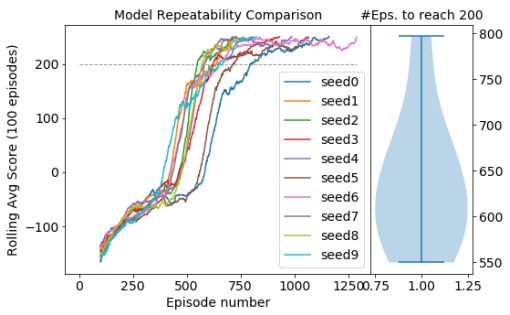

Fig. 2: *[Validation of DQN base model training performance consistency, with 10 randomized training instances]*

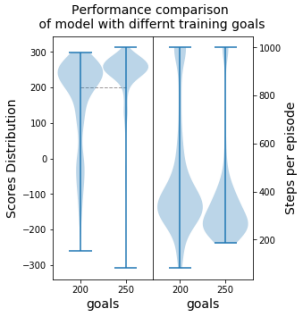

Fig. 3: *[Performance comparison of ψ = 200 & 250]*

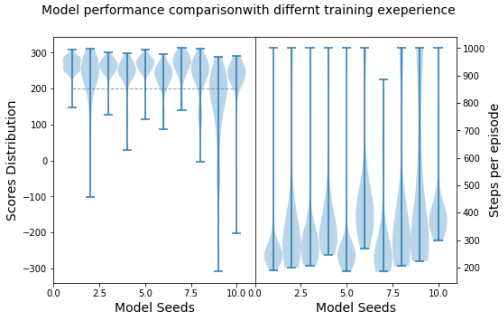

Fig. 4: *[comparison of consistency and speed of convergence among10 randomized instance of ψ = 250 models]*

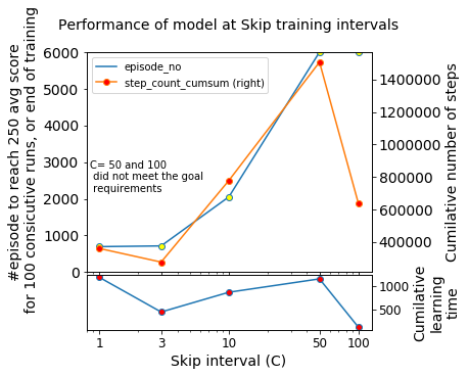

Fig. 6: *[Skip-Step training, comparing experience and time for convergence(ψ250) of different C intervals]*

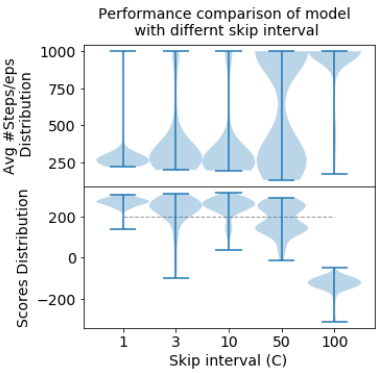

Fig. 7: *[Skip Step testing, score and prediction time consistency comparison]*

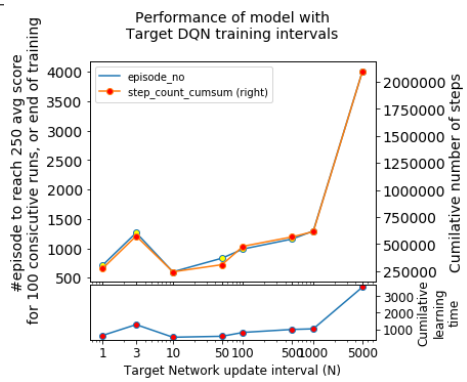

Fig. 8: *[Target network Training, comparing experience and time for convergence(ψ250) of different N intervals]*

model, to avoid crash or descending and therefore hovered at the top until 1000 steps per episode limit. With greedy actions along with slight randomness, LL descended towards the pad, accumulating positive proximity based rewards, which increased rapidly with every experience, thereafter explored optimal engine usage, prioritizing future rewards,as $Scale[0.8]\gamma = 0.99$, above -ve immediate action pts, III-A, to stabilize orientation and land softly. following increased descending speed, reducing number actions steps per run, to increase score further, till ψ =250, fig:1 . Agent with $\psi = 200$ was saved for analysis, and discussed in the next section.

2) Agents Testing results: ψ=250 trained Agent could reach ψ =200, 98 out of 100 random new environments, Fig.5. However, Score fluctuation between 298 and 186 and ending the episode in 250 to 450 steps with 99% confidence level, depicts non uniformity in policy prediction, supporting that, minimizing $\overline{Error}$ does not ensure optimal action values at all the states. Same model under different experiences may produce different policies, fig 2, which may not be ideal. However training for higher Score $\psi = 250$ ensure compliance in achieving 200 score, fig 3. and consistency in random environment. fig 4.

IV. Parameter comparison Analysis and results

As mentioned in the **??**, the deadly triad may effect DQN's performance, resulting in highly unstable results. This Section discussed the extended analysis of different parameters to reduce DT's effects and stabilize the model, with $\psi = 250$ trained model, focusing on *skip step training*, *Target DQN training* and *Experience Replay*. Along with natural termination, learning was ended, if ψ ¡ -800 or episode count exceeded 6000.

A. Skip Step Training

In this process, the active model was trained only after prediction and storage of every C number of actions. With bootstrapping, learning is highly dependent on past experiences. Consecutive per action-step, training and prediction may not enable clear differentiation between "generally" high and low "return" actions, due to frequent ∇w changes and therefore low representation of current believes, yielding high fluctuation policies, which can be reduced by lowering α, at a cost of slow learning. However, updates in controlled intervals, the agent could achieved $\psi = 250$ for C=3 and 10, in **455**, **868** training seconds, fig 6, along with **79, 88**,fig 7, test success respectively, with C= 3 achieving 54% more successes in less than half training times of C=1 trained for $\psi = 200$, with 51 test success and **931** training seconds. It is hypothesised, model with smaller skip intervals (C= 3,10) with extended training could generalize faster and better than no-skip models, like C =1 with $\psi = 250$, 98 success in 1179 seconds. However, with high C (50,100), as the updates are less, historic sub-optimal action are utilized in the model changing current belief toward older ones, resulting in catastrophic interference. *Therefore, Selective Skip Step interval model updates, may generalize faster and with high stability than no-skip models, and possibly with low α as well.*

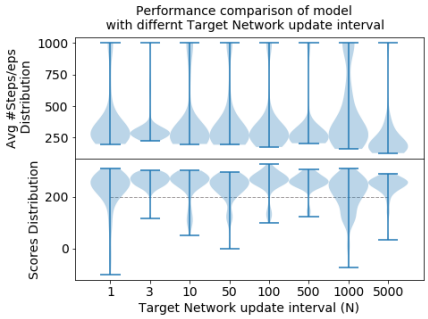

Fig. 9: *[Target Network testing, score and prediction time consistency comparison.Intermediate values show consistent and faster convergence]*

B. Target DQN training

Target DQN, stabilizes the active DQN, by bootstrapping old believes and controlling the impact of high fluctuations experiences, as per III-B3. With off-policy learning and $\overline{Error}$ objective function, $\overline{q}$ is may not be an unbiased estimator of Q, therefore, with Larger N, biases are aggravated, in w, resulting in sub-optimal policy convergence and prediction fluctuations, (N= 1000, 50000, fig: 8 ,9). Additionally, at small N(= 1), the model may perform sporadic updates, due to inherent stochasticity. Even if model reaches ψ while training, localized generalizations may occur, and it may fail to perform in new environment. fig 9. Therefore, selecting an intermediate N, may reduce these instabilities, and increase generalization, enabling consistent and faster convergence, eg. N=10,50. produced 6% more test successes,(84), in 20% less prediction time steps per episode, (0.275 sec.), than N=1.(79 , 0.323 sec.). Additionally, N=3, with 2x experience than N= 10, produced 99% test success, avoiding local generalizations resulting in prolonged corrective updates.

Hence, it can be concluded that Target DQN could enables better generalization and reduce the impact of biased bootstrapping.

C. Experience Replay

As defined in, III-B3, Experience replay reduces the bootstrap biases, to ensure network stability. The size of the Replay memory could define the reduction in instability. With small d, recent experiences drive updates, resulting in NN catastrophic interference, as the model forgets past experiences, leading to loss or local generalization during learning, and slower or

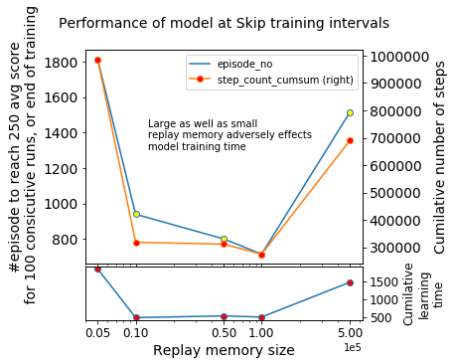

Fig. 10: *[Repay memory training, comparing experience and time for convergence(ψ250) of different d]*

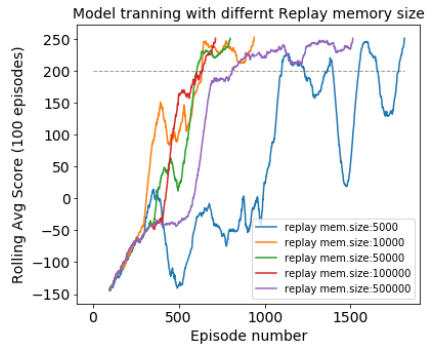

Fig. 11: *[Repay memory training, progression performance comparison of time and model stability (ψ250)]*

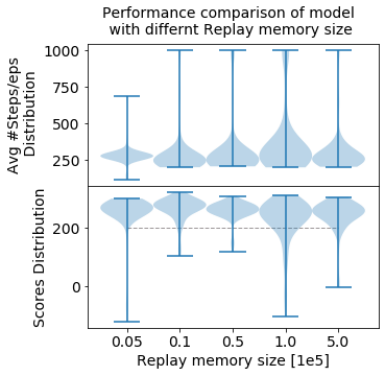

Fig. 12: *Replay memory testing, score and prediction time consistency comparison*

no convergence, eg. d =5000 took 1843 seconds, i.e 3.6x of d= $1e^4, 5e^4, 1e^5$,fig 10, resulting in high coefficient of prediction variance,(25%) . Even though training for $\psi = 250$ enables for 96 successes, it produced only 56 successes with $\psi = 200$. Larger d , fail to successfully remove less visited states biases, the impact persists due to longer storage of less visited bootstrapped states. As the model learns, the $\overline{Error}$ decreases, and better policy experiences are stored, however, the stored sub-optimal historical experiences may cause catastrophic interference, [16] [15] and instability, resulting in slower or adverse learning effects, fig 11, ex: d=$5e^5$ with 1472 training seconds.

An Intermediate storage level, could avoid the above adverse effects, while reducing bootstrap biases. ex: d = $1e^4, 5e^4$, with lesser training, succeeded in 93,95 tests with avg score of 263, 257 and only 12%-15% coefficient of deviation. respectively,fig.12

D. Other Parameters

Stabilization of DQN as discussed above is a subject to the problem being solved. It may require repeated stress testing of different parameters to understanding stabilization effects.

Multiple other hyper-parameters govern ability to generalize, speed and consistency of DQN, fig 13. such as,

Batch Size of update: Lower as well as higher batch size could introduce stochasticity and biases, due to hyper localisation and historical errors accumulation during training, respectively. Resulting in slow learning and low performance, ex: size = 8, 256. Intermediate batch sizes enable fast learning, and higher accuracy. However, slight stochasticity or biases, may enable DQN to explore more, resulting in more exposure and better informed action policies, which could be intuitively induced. ex: size = 23,128. **Epsilon Decay:** Small decay rate, promotes exploration and therefore possibly better policy estimates, with slow learning. Additionally it was observed, with fast ϵ decay, only a limited amount of dedicated exploration experience is gained , which may not cause localized generalization, and therefore, with very small remaining ϵ, agent is forced to take known optimal paths with slow and longer exploration, and better estimates. **State-Space - number of neurons/layer:** layers with more number of neurons can represent the state action space better, and therefore can learn faster, however, model may over-fit, by identifying the states action pairs rather than generalizing them.ex: 32 neurons network took 3x time of 256, but could generalize equivalently well.Therefore, a model with average neurons per layer, and adequate exposure, may learns to generalize quickly.

E. Future Motivations

Behaviour defining model parameters may relate and perform differently with different configurations and environments. In this report we have explored few parameters in isolation. however a better configuration for LL could be achieved with parameter relationship study. Additionally, tools like replay memory could be further modified using a Skew normal distribution instead of uniform, to select experiences, prioritize recent experience without neglecting historical estimates, and control any historical biases. N-Step DQN or DQN(λ) could also be used decrease the impact of one step stochastic errors.

V. Conclusion

With the use of DQN we were successfully able to learn and solve LL problem. We found that setting higher goals could yield better results, enabling longer learning exposure therefore better generalization. Tools such as Skipstep learning, Target network learning and Experience repay could potentially reduce the impact of Deadly Triads by ensuring generalized learning. Additionally, their parameter variants could result in faster convergence, with a use for time sensitive applications. However, selectively identified parameters could generalize as well as converge adequately fast.

References

[1] Sutton, R.S. Learning to predict by the methods of temporal differences. Mach Learn 3, 9–44 (1988).
[2] L Baird , Residual algorithms: Reinforcement learning with function approximation - Machine Learning Proceedings 1995, 1995
[3] Jaakkola, Jordan, Singh.(1994) On the Convergence of Stochastic Iterative Dynamic Programming Algorithms Neural Computation 6(6):1185-1201
[4] Sherstov and Stone 2005, Improving action selection in MDP's via knowledge transfer - AAAI, 2005
[5] Shimon Whiteson, Matthew E. Taylor, and Peter Stone, Adaptive Tile Coding for Value Function Approximation, AI Technical Report AI-TR-07-339, 2007.
[6] Kenji Doya,Reinforcement Learning In Continuous Time and Space, Neural Computation, 12(1), 219-245 (2000).
[7] M. Littman, Csaba Szepesvari, A Generalized Reinforcement-Learning Model: Convergence and Applications, Published in ICML 1996
[8] (Sutton, 1995; Baird, 1995; Tsitsiklis and Van Roy, 1997)
[9] Cybenko, G. Approximation by superpositions of a sigmoidal function. Math. Control Signal Systems 2, 303–314 (1989).
[10] ,Ilya Loshchilov, Frank Hutter, Decoupled Weight Decay Regularization (ICLR 2019)
[11] Hasselt, Doron, Strub, Sonnerat, and Modayil, Deep Reinforcement Learning and the Deadly Triad. CoRR (2018)
[12] V. Mnih, Human-level control through deep reinforcement learning, Nature 518 (7540): 529–533 (February 2015)
[13] Matthew D. Zeiler 2012,ADADELTA: An Adaptive Learning Rate Method, Computer Science- ArXiv
[14] Improving generalization performance by switching from Adam to SGD
[15] Ratcliff, Roger (1990). "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions". Psychological Review. 97 (2): 285–308.
[16] McCloskey, Michael; Cohen, Neal J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. Psychology of Learning and Motivation. 24. pp. 109–165.

	Batch size					epsilon decay				Neurons / hidden layer			
Values	8	32	64	128	256	0.000005	0.00001	0.00005	0.0001	32	64	128	256
Training episode till termination	6000	1450	714	935	915	3109	1966	714	904	5071	1824	714	658
Total training time (s)	5422	880	496	820	818	928	779	609	933	4439	1514	460	496
Total action steps till 250 or term	3555033	594016	275439	390252	332230	578060	450121	275439	426294	2887913	953642	275439	252802
Total Test success (of 100)	90	95	78	93	70	97	86	78	98	93	88	78	92
Avg Test Score	229	256	221	258	227	256	249	221	271	241	232	221	242
Test Score STD	61	39	93	56	64	31	47	93	28	65	35	93	41
Avg Steps per Test episode	412	267	404	256	404	309	351	404	256	347	453	404	357
Steps per Test episode STD	197	171	266	156	232	128	250	266	112	182	117	266	182

Fig. 13: *[Training and testing comparison for Batch size, exploration decay rate and count of hidden layer neurons]*