Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Dynamic Action Repetition for Deep Reinforcement Learning

Aravind S. Lakshminarayanan,* **Sahil Sharma,*** **Balaraman Ravindran**
Indian Institute of Technology, Madras

Abstract

One of the long standing goals of Artificial Intelligence (AI) is to build cognitive agents which can perform complex tasks from raw sensory inputs without explicit supervision (Lake et al. 2016). Recent progress in combining Reinforcement Learning objective functions and Deep Learning architectures has achieved promising results for such tasks. An important aspect of such sequential decision making problems, which has largely been neglected, is for the agent to decide on the duration of time for which to commit to actions. Such *action repetition* is important for computational efficiency, which is necessary for the agent to respond in real-time to events (in applications such as self-driving cars). Action Repetition arises naturally in real life as well as simulated environments. The time scale of executing an action enables an agent (both humans and AI) to decide the granularity of control during task execution. Current state of the art Deep Reinforcement Learning models, whether they are off-policy (Mnih et al. 2015; Wang et al. 2015) or on-policy (Mnih et al. 2016), consist of a framework with a static action repetition paradigm, wherein the action decided by the agent is repeated for a fixed number of time steps regardless of the contextual state while executing the task. In this paper, we propose a new framework - Dynamic Action Repetition which changes Action Repetition Rate (the time scale of repeating an action) from a hyper-parameter of an algorithm to a dynamically *learnable* quantity. At every decision-making step, our models allow the agent to commit to an action and the time scale of executing the action. We show empirically that such a dynamic time scale mechanism improves the performance on relatively harder games in the Atari 2600 domain, independent of the underlying Deep Reinforcement Learning algorithm used.

Introduction

There has been a growing interest both in creating AI agents which can play computer games well (Mnih et al. 2015; Hausknecht and Stone 2016; Mnih et al. 2016), as well as creating Human-like AI agents against which human opponents can enjoy playing (Hingston 2010; Ortega et al. 2013; Van Hoorn et al. 2009). Indeed, both the problems are related. A solution to the former is necessary, but not sufficient for a solution to the later. This is because agents which can play the game well automatically lead to AI opponents against which humans might enjoy playing computer games. Hence, any advancement in the former domain leads to an advancement in the latter. We present one such advancement in Reinforcement Learning (RL)-based game playing.

Video game domains such as Mario (Togelius, Karakovskiy, and Baumgarten 2010), Atari 2600 (Bellemare et al. 2013) and Half Field Offensive (Hausknecht et al. 2016) have served as a test bed to measure performance of learning algorithms in AI-based game playing. Such games present unique challenges to an AI agent since performing well in them requires being able to understand the current state of the game (involves pattern recognition) as well as being able to choose actions after considering long term implications of choosing those actions (involves planning). Recent applications of Deep Reinforcement Learning (DRL) to these domains has resulted in state-of-the-art performance, when the policies have to be learnt directly from pixels in a model-free RL setting (Mnih et al. 2015; 2016; Hausknecht and Stone 2016).

Game playing AI should be real-time for it to appear plausible to humans and for it to have real world applications. This is one of the motivations behind the evolution of Deep Learning (DL)-based AI agents as a replacement for simulation based AI. However, DRL agents involve significant computation in the form of convolutional neural networks and recurrent neural networks. One way to reduce such computation is to introduce a hyper-parameter in DRL algorithms called the *action repetition rate* (ARR) which denotes the number of times an action selected by the agent is to be repeated. If the ARR is low, the decision making frequency of the agent is high. This leads to policies which evolve rapidly in space and time. On the other hand, a high ARR causes infrequent decisions, which reduces the time to train at the cost of losing fine-grained control. A higher ARR also leads to the agent learning human-like policies for game play. This is because the policies learnt with a higher ARR have fewer action sequences which exhibit super-human reflexes as compared to an agent which plays with a low ARR ((Mnih et al. 2015; 2016) use an ARR of 4). A reduction in decision making frequency gives the agent the advantage of not having to learn the control policy in the intermediate states given that a persistent action from the current state can lead to an advanta-

*Authors contributed equally, listed in alphabetical order

geous next state. Such skills would be useful for the agent in games where good policies require some level of temporal abstraction. An example of this situation in Seaquest (an Atari 2600 game) is when the agent has to continually shoot at multiple enemies arriving together and at the same depth, one behind another. In the case of Space Invaders (Atari 2600 game), the lasers are not visible in every fourth frame, as noted by (Mnih et al. 2015). Hence, a higher action repetition could help the agent avoid the confusion of having to decide an action in such peculiar intermediate states, where the lasers are not visible.

Having said all of that, there are also situations where agents have to take actions which require quick reflexes to perform well with an example from Seaquest being states in which multiple enemies attack simultaneously and are placed at varying depths. The agent is then expected to manoeuvre skillfully to avoid or kill the enemies. Similarly, in Space Invaders the agent needs to avoid multiple lasers being shot simultaneously by enemies that are progressively moving closer to the agent. Such situations merit a finer granularity of control. Therefore, a high but static ARR is probably not a solution for better game playing strategies in spite of the temporal abstractions it provides to the agent. As a small step in the direction of temporal abstractions in the policy space (in the form of temporally elongated actions), we propose a dynamic action repetition paradigm of game-playing. The paradigm is generic enough that it can be combined with any off-the-shelf discrete action space DRL algorithm. The proposed paradigm presents a more structured policy framework, wherein the policy consists of the action probabilities (or state-action values) along with the number of times the corresponding action is to be repeated, each time a decision is to be made. This is on the lines of (Hausknecht and Stone 2016), in which an Actor Critic setup is used in the Half Field Offense domain(Hausknecht et al. 2016), to learn policies that contain both the probabilities of selecting actions as well as their associated parameters. Policies learnt under this paradigm guide the agent in deciding whether it is beneficial to exert super-human-reflexes, or not. In the case of longer temporal persistence of the chosen action, the optimal level of persistence would help the agent to get to most advantageous (in terms of expected cumulative discounted future reward) temporally distant state.

One of the major drawbacks of the current DRL game playing algorithms (Mnih et al. 2015; 2016; Wang et al. 2015) is that the action repetition rate is a static hyper-parameter which has to be tuned using usual hyper-parameter tuning techniques. We demonstrate that this inability to decide the granularity of control deeply inhibits the kind of control policies that are learnt by the algorithms. We show the efficacy of the policies learnt using our paradigm over those learnt using the existing algorithms by empirically establishing the fact that dynamic action repetition is an important way in which an AI agent's capabilities can be extended. This is done by combining existing DRL algorithms with our paradigm. To demonstrate that this paradigm (Dynamic Action Repetition) can be combined with any Deep Reinforcement Learning algorithm, we perform experiments in off-policy (Mnih et al. 2015) as well as on-policy (Mnih et al. 2016) setting. Thereby we make the claim that there is a need to explore and experiment with structured parametrized policies for finding temporal abstractions using an Actor Critic setup in the Deep Reinforcement Learning based Game Playing domain.

Related work

One of the first efforts which pushed the state of the art significantly in the domain of game playing based on raw sensory inputs was (Mnih et al. 2015). Their architecture, DQN, motivated the use of convolutional neural networks for playing games in the Atari 2600 domain.

(Braylan et al. 2015) is a work that focuses on the power of the frame skip rate hyper-parameter (it is the hyper-parameter in ALE which directly controls the action repetition rate) with experiments in the Atari 2600 domain. Their learning framework is a variant of Enforced Sub-Populations (ESP) (Gomez and Miikkulainen 1997), a neuroevolution approach which has been successfully trained for complex control tasks such as controlling robots and playing games. They show empirically that the frame skip parameter is an important factor deciding the performance of their agent in the Atari domain. They demonstrate for example that Seaquest achieves best performance when an ARR of 180 frames is used. This is equivalent to the agent pausing (continuing the same action) for three seconds between decisions since the ALE emulator runs at 60 frames per second. They argue that a higher value of action repetition allows the agent to not learn action selection for the states that are skipped and hence develop associations between states that are temporally distant. However, they experiment only with a static action repetition setup. The key way in which our work differs from both the ESP and the DQN approaches is that we make action repetition a dynamic learnable parameter.

The idea of dynamic-length temporal abstractions in the policy space on Atari Domain has been explored by (Vafadost 2013). They use a Monte Carlo Tree Search (MCTS) planner with macro-actions that are composed of the same action repeated k times, for different k. The way in which our approach differs from (Vafadost 2013) is in terms of using a Deep Neural Network to build non-linear Q-function approximators instead of making use of search techniques that cannot generalize across unseen states. We also learn to pick the suitable repetition extent for a state through deep networks instead of searching through rollouts for the optimal repetition extent k.

The utility of repetitive *macro-actions* has been pointed out by (Ortega et al. 2013) in designing high level actions to imitate human-like play in Mario such as *Walk, Run, Right Small Jump, Right High Jump* and *Avoid enemy*, which are composed of varying length repetitive key presses. For instance, *Right High Jump* involves pressing the *Jump and Right* keys together for 10 frames, while *Right Small Jump* requires the same for 2 frames.

Background

The following four sub-sections introduce standard Reinforcement learning algorithms and their deep counterparts.

Q-Learning algorithm

One of the approaches for solving the sequential decision making problem is to estimate the optimal value of every state-action pair - $Q(s, a)$. The optimal value of an action a in a state s is defined as the expected cumulative sum of future rewards on taking a in s and following the optimal policy thereafter. $Q(s, a)$ is a measure of the long-term reward obtained by taking action a in state s. Computing approximations for the Q-function enables the agent to select the optimal action a in a state s. Hence, one way for the agent to learn optimal policies is to estimate $Q(s, a)$ for the given task. Q-learning (Watkins and Dayan 1992) is an off-policy Temporal Difference (TD) learning algorithm which does exactly that. The Q-values are updated iteratively using the Bellman optimality equation (Sutton and Barto 1998) with the rewards obtained from the game as below:

$$Q_{t+1}(s, a) = \mathbb{E}[r + \gamma \text{max}_{a'} Q_t(s', a')|s, a]$$

Here t denotes the time-step during the episode. During training, the behavioral policy is ϵ- greedy with respect to $Q(s, a)$ to ensure sufficient exploration.

Deep Q Network (DQN)

In high dimensional state spaces, it is infeasible to compute optimal Q-values for all possible state-action pairs explicitly. One way to address this problem is to approximate $Q(s, a)$ using a parametrized function approximator $Q(s, a; \theta)$, thereby gaining the power to generalize over unseen states by operating on low level features (Sutton and Barto 1998). Recent advances in representation learning using deep neural networks (LeCun, Bengio, and Hinton 2015) provide an efficient mechanism to learn hierarchical features across large state spaces and avoid feature engineering. DQN (Mnih et al. 2015) combines the representation learning power of deep neural networks with the Q-Learning objective to approximate the Q-value function for a given state and action. The loss function used for learning a Deep Q Network is :

$$L(\theta) = \mathbb{E}_{s,a,r,s'}[(y^{DQN} - Q(s, a; \theta))^2],$$

with

$$y^{DQN} = (r + \gamma \text{max}_{a'} Q(s', a', \theta^-))$$

Here, L represents the expected TD error corresponding to current parameter estimate θ. θ^- represents the parameters of a separate *target network*, while θ represents the parameters of the *online network*. The usage of a *target network* is to improve the stability of the learning updates. The gradient descent step is as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a,r,s'}[(y^{DQN} - Q(s, a; \theta))\nabla_\theta Q(s, a)]$$

The off-policy nature of the algorithm ensures that DQN is able to avoid correlated updates, which are likely, when learning on temporally correlated transitions that the *online network* simulates. This is facilitated through an experience replay memory (Lin 1993) D (of fixed maximum capacity) where the state transitions and rewards experienced by the agent are pooled in a First-In-First-Out (FIFO) fashion.

Advantage Actor-Critic Algorithm

The Q-learning algorithm described in the previous subsections computes only the value function associated with state-action pairs. Explicit policies are not learnt by the agent during Q-Learning. Rather, the policy of the agent is induced implicitly (by picking the action with the maximum Q-value) along with low-probability exploration based on an ϵ-greedy approach (Sutton and Barto 1998). In contrast, Actor-Critic methods (Konda and Tsitsiklis 2003) explicitly model the policy that is executed by the agent along with a value function component updated using Temporal Difference (TD) learning methods. Hence, they have two distinct components: an *actor* and a *critic*. We describe the parametric version of Actor Critic below.

The actor proposes a parametrized policy $\pi(a_t|s_t; \theta_a)$ while the critic provides the basis for improvement in the policy estimation by approximating the optimal value function $V^*(s)$ using a parametrized estimate; $V(s_t|\theta_c)$. We describe a parametric critic as a general setup applicable to high dimensional state spaces. In principle a non-parametric critic could also be learnt, using table look ups. The state value function computed by the critic is used for obtaining the Temporal Difference (TD)-error term $(r + \gamma V(s') - V(s))$ required for computing the updates to the actor parameters θ_a. The updates on the actor parameters θ_a are based on policy gradient methods with the critic term providing the sign and weight for the updates (Sutton et al. 1999). The ideal unbiased policy gradient for updating the parameters of the probability of execution of action a_t in state s_t is $\nabla_{\theta_a} \log \pi(a_t|s_t; \theta_a)\mathbb{E}[R_t]$. Here R_t is the return obtained from state s_t after executing action a_t. Naturally, based on how $Q(s_t, a_t)$ is defined, $\nabla_{\theta_a} \log \pi(a_t|s_t; \theta_a)Q(s_t, a_t)$ provides a biased estimate for the same but with lower variance when compared to using a point estimate of R_t as proxy for $\mathbb{E}[R_t]$. The variance of the former estimator can be improved even further without adding to the bias by using $\nabla_{\theta_a} \log \pi(a_t|s_t; \theta_a)(Q(s_t, a_t) - b(s_t))$ where $b(s_t)$ is a baseline term that's independent of the entity on whose *decision* the gradient is computed (a_t here). The baseline could however be state-dependent. Often, the baseline used is the value function estimate of state s_t: $V(s_t|\theta_c)$.

The proxy function for the utility R_t is hence $Q(s_t, a_t) - V(s_t)$. This is defined as the Advantage Function $A(s_t, a_t)$. Since this instantiation of Actor Critic uses the Advantage Function as the proxy utility, this method is referred to as *Advantage Actor Critic*. The efficacy of this method relies on the critic approximating the optimal value function $V(s_t)$ since the Advantage Function $A(s_t, a_t)$ is estimated using the one step TD error, as described below. The target value for $Q(s_t, a_t)$ from the Bellman optimality criterion (Sutton and Barto 1998) is $r + \gamma \text{max}_{a'} Q(s_{t+1}, a') = r + \gamma V(s_{t+1})$. Therefore, $A(s_t, a_t)$ is estimated as $(r + \gamma V(s_{t+1})) - V(s_t)$, which is just the one-step TD error in estimating $V(s_t)$.

Asynchronous Advantage Actor Critic

On-policy Actor Critic algorithms are non-trivial to model with Deep Neural Networks. This is because the on-policy

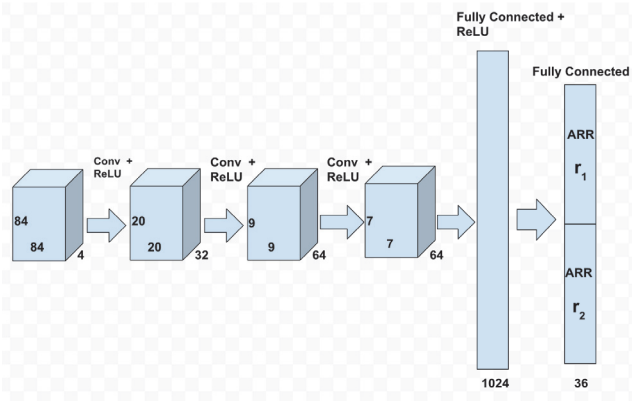


Figure 1: DQN's architecture under the proposed Paradigm. The upper part of the final layer corresponds to Q-values for actions that operate at action repetition rate r_1 and lower part corresponds to same actions which operate at action repetition rate of r_2.

nature of the learning leads to correlated updates by violating an important assumption made by popular stochastic gradient descent-based parameter update algorithms that the input data points are independently and identically distributed. Asynchronous Advantage Actor Critic (A3C) (Mnih et al. 2016) overcomes this problem by running multiple actor threads which explore different parts of the state space in parallel. The updates from the multiple threads are synchronized with respect to the global policy network parameters θ_a. This ensures that the updates made to the global policy parameters are reasonably uncorrelated. Due to multiple actor learner threads operating at different parts of the state space and pooling in gradient updates in an Advantage Actor Critic paradigm, this method is referred to as *Asynchronous Advantage Actor Critic*.

Dynamic Action Repetition For Deep Reinforcement Learning

The key motivation behind the *Dynamic Action Repetition* paradigm is the observation that when humans execute tasks (such as playing games), the actions tend to be temporally correlated and elongated, almost always. Such behavior occurs because expectancy for upcoming action requirements is a fundamental prerequisite for human action control (Gilbert and Wilson 2007; Thomaschke and Dreisbach 2013). For certain states of the game, we play long sequences of the same action whereas for some states, we switch the actions performed quickly. We base these decisions on what we expect from the temporally distant future, as a result of these actions. For example, in Seaquest, if the agent is low on oxygen and deep inside the sea, we would want the agent to resurface and fill up oxygen using a long sequence of *up* actions.

Although the framework we propose is generic enough to be combined with any discrete-action space DRL algorithm, we take the example of DQN to explain the Dynamic Action repetition scheme. To demonstrate the generality of the paradigm, experiments are performed with DQN as well as A3C. The paradigm, explained with the help of DQN is as follows:

Let $\mathcal{A} = \{a_1, \cdots, a_{|\mathcal{A}|}\}$ denote the set of all legal actions for a game (for Seaquest $\mathcal{A} = \{0, 1, \cdots, 17\}$). We introduce $|\mathcal{A}|$ new actions $\{a_{|\mathcal{A}|+1}, \cdots, a_{2|\mathcal{A}|}\}$. Figure 1 depicts this scheme using a diagram.

The semantics associated with the actions are as follows: action a_k results in the same basis action being played in the ALE as the action $a_{(k\%|\mathcal{A}|)}$. The difference is in terms of the number of times the basis action is repeated by the ALE emulator. DQN as well A3C operate with a static action repetition hyper-parameter which is equal to 4. Hence any selected action is repeated 4 times by ALE. In our model, action a_k is repeated r_1 number of times if $k < |\mathcal{A}|$ and r_2 number of times if $k \geq |\mathcal{A}|$. One can think about each a_k as representing a *temporally repeated* action or in RL terms as a macro-action. This scheme is implemented by doubling the number of neurons in the final layer of the DQN architecture. For a given state s, the augmented model (with double the number of output neurons as DQN) thus outputs the discrete set of value function approximations $\{Q(s, a_1), \cdots, Q(s, a_{2|\mathcal{A}|})\}$. $Q(s, a_k)$ representing an approximation to the return the agent would get on taking action a_k in state s and following the optimal policy thereafter. In the case of the augmented A3C model, the output would be a probability distribution over all the actions $\{a_1, \cdots, a_{2|\mathcal{A}|}\}$. These new macro-actions can lead to more-frequent and larger rewards, and thus significantly impact the value functions learnt by the augmented model. Note that r_1 and r_2 are hyper-parameters in this model and must be chosen before the learning begins. This paradigm presents the agent with 2 discrete levels of control, as far as action repetition is concerned, regardless of the underlying Reinforcement Learning algorithm. It is ideal to learn policies in a parametrized action space where the agent outputs an action a_k and a parameter r where r denotes the number of times that the agent wants to repeat action a_k consecutively. Such a framework would have the representative power to select the optimal number of times an action is to be executed. But, it would also be proportionately complex and as a result difficult to train. Our paradigm seeks to provide an approximation to this richer model and favors simplicity over optimality of game play, which would come at the cost of a more complicated learning procedure. We leave it to future work, to explore the learning of policies paramaterized by the ARR in the Game Playing domain, learnt with the help of an actor-critic algorithm.

Experimental Setup and Results

To demonstrate the generality of our framework, experiments were conducted with 2 DRL algorithms: DQN and A3C. The following sub-sections document the respective experimental setup and the results.

Augmented DQN

General game-playing evaluation was performed on 4 Atari 2600 domain games: **Seaquest, Space Invaders, Alien** and **Enduro**. A single algorithm and architecture, with a fixed

set of hyper-parameters was used to play all 4 games. The combination of DQN with our paradigm results in a model (Augmented DQN) which has the same low-level convolutional structure and input preprocessing of DQN (Mnih et al. 2015). Due to the partially observable nature of the Atari 2600 domain games, the last 4 frames are combined and given as a $84 \times 84 \times 4$ multi-channel input to the model. This is followed by 3 convolutional layers, which are in turn followed by 2 fully-connected layers.

Since the augmented model has double the number of output neurons as a static action-repetition model, we wanted to give more representational power to it, for being able to decide from a larger set of actions. Therefore, the augmented model has 1024 units in the pre-output hidden layer as compared to 512 units used in the DQN architecture (Mnih et al. 2015).

S.No.	Game	HLS	ARR	Arch	AS
1	Seaquest	1024	4	DQN	5450
2	Seaquest	1024	20	DQN	1707
3	Seaquest	1024	D	DFDQN	**10458**
4	Seaquest	512	4	DQN	5860
5	SI	1024	4	DQN	1189
6	SI	1024	D	DFDQN	**2796**
7	SI	512	4	DQN	1692
8	Alien	1024	4	DQN	2085
9	Alien	1024	D	DFDQN	**3114**
10	Alien	512	4	DQN	1620
11	Enduro	1024	4	DQN	707
12	Enduro	512	4	DQN	729
13	Enduro	1024	D	DFDQN	**1002**
14	Enduro	1024	20	DQN	124

Table 1: Experimental results for DQN architectures. AS denotes the average score in the best testing epoch. SI is the game of Space Invaders. HLS denotes the pre-final layer hidden layer size.

The values of r_1, r_2 (defined in the previous section) are kept the same for all three games and are equal to 4 and 20 respectively. To ensure sufficient exploration (given that the augmented model has double the number of actions), the exploration probability ϵ is annealed from 1 to 0.1 over 2 million steps as against 1 million steps used in DQN.

We claim that the improvement in performance is not just due to the increase in the representational power by having double the number of pre-final hidden layer neurons. to validate this hypothesis, we run DQN baselines with 1024 units in the pre-final layer for all three games. We also report the scores obtained from original DQN architecture with 512 pre-final hidden layer neurons from a recent usage of DQN in (Wang et al. 2015), where DQN is reported as baseline for the Duelling DQN model. A training epoch consists of 250000 steps (action selections). This is followed by a testing epoch which consists of 125000 steps. The score of a single episode is defined to be the sum of the rewards received from the ALE (Bellemare et al. 2013) emulator in that episode. The score of a testing epoch is defined to be the average of the scores obtained in all of the complete episodes played during that testing epoch. The scores reported in this section are for the testing epoch with the highest average episode score (known as the *best score* for a game). Table 1 presents the experimental results. Since the hyper-parameter for action repetition is known as Frame Skip in the ALE, the augmented model is referred to as DFDQN (Dynamic Frameskip DQN). HLS denotes the pre-final hidden layer size. SI stands for Space Invaders. ARR stands for the action-repetition rate used. A value of D for ARR represents that the action repetition rate is dynamic. Arch. denotes the architecture used. AS denotes the best average testing epoch score as defined above.

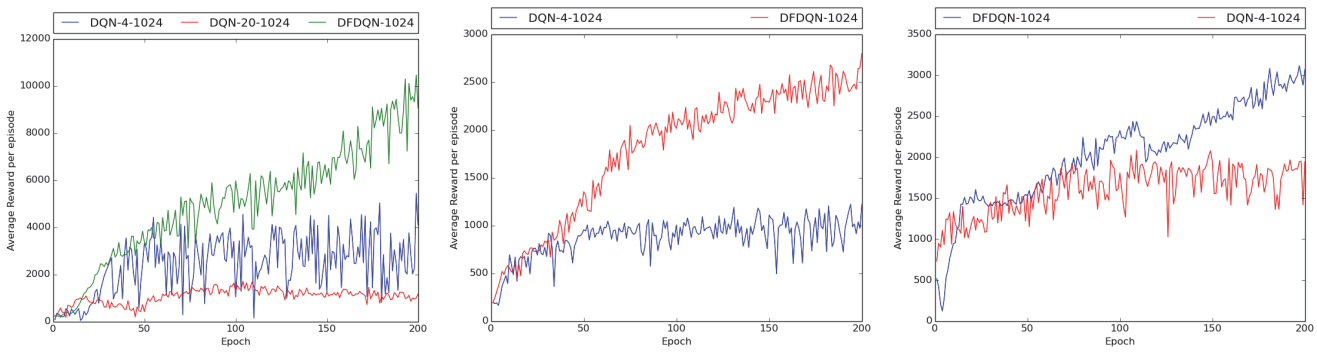
In each of plots in Fig 2, one epoch consists of 125000 steps (decisions). DQN-a-b refers to DQN architecture that has b units in the pre-output layer and operates with an ARR of a. DFDQN-b is the Dynamic Action Repetition variant of Deep Q-Network architecture with b units in pre-final layer.

An interesting characteristic of all the graphs is that Augmented Model's scores-graph has a large slope even at the end of 200 epochs. This means that there is still scope for the performance to improve on further training beyond 200 epochs. To verify our claim that temporally extended actions can lead to better policies, we did an analysis of the percentage of times the longer sequence of basis actions was selected. Using the network which yielded the *best score* (notion of best score defined in the previous section), we ran a testing epoch of 10000 decision-making steps, which resulted in 17 episodes for Seaquest, 18 episodes for Space Invaders, 18 episodes for Alien and 17 episodes for Enduro. We recorded the actions executed at each decision-step.

S.No.	Game	Percentage
1	Seaquest	69.99
2	Space Invaders	78.87
3	Alien	71.31
4	Enduro	67.84

Table 2: Longer action selection percentages for

The table above shows that the augmented model is able to make good use of the longer actions most of the times, but does not ignore the temporally shorter actions either. The agent has learnt through repeated exploration-feedback cycles to prefer the extended actions but still exercises the fast reflexes when the situation needs it to do so. We can safely claim that the addition of these higher action repetition options has been an important contributing factor to the improvement in performance. To strengthen our claim that there is a need for *dynamic* action repetition, we show results on two of the games (Seaquest and Enduro) in which the DQN operates with just the higher but *static* ARR of 20. Not surprisingly, it scores poorly during gameplay. Videos of some of the learned policies for Space Invaders, Seaquest and Alien are available at https://goo.gl/VTsen0, https://goo.gl/D7twZc and https://goo.gl/aCcLb7) respectively.



(a) Seaquest Training Comparison (b) Space Invaders Training Comparison (c) Alien Training Comparison

Figure 2: DQN vs DFDQN comparison for Seaquest, Space Invaders and Alien

Augmented A3C

To demonstrate that the proposed paradigm is not specific to the DQN algorithm but rather works across different types of DRL Algorithms for game playing, we conducted similar game-playing experiments using the Asynchronous Advantage Actor Critic (Mnih et al. 2016) model as well. Unlike in the case of Augmented DQN, **all** the hyper-parameters were kept **exactly** the same as that in the case of A3C. This represents an adversarial setting for the augmented A3C case since deciding the optimal ARR could possibly benefit from having a larger state-representation as well as changes in other hyper-parameters. We used the LSTM-controller and the best-performing open source implementation of A3C algorithm. The baseline A3C models as well as the augmented A3C models were trained for 100 million decision steps.

Evaluation was performed by sampling from the probability distribution representing the final policy and averaging the scores over 100 episodes. The results are presented in table 3. A3C denotes the implementation of A3C used by us. DFA3C denotes the A3C model Augmented with Dynamic Action Repetition. A3CP denotes the scores of the published A3C models (Mnih et al. 2016), which have been included for the sake of completeness. A note of caution is to not compare the scores obtained in this work with the published scores in (Mnih et al. 2016) since they use a very different evaluation strategy called *human starts*, which is impossible to replicate exactly in the absence of human testers. We clearly see that when compared to A3C, DFA3C leads to improved metrics on all 5 tasks with the improvement on Enduro being as high as 645 times better.

S.No.	Game	Arch	AS
1	Seaquest	A3C	1769.12
2	Seaquest	DFA3C	**2047.74**
3	Seaquest	A3CP	1326.1
4	SI	A3C	603.04
5	SI	DFA3C	**2220.18**
6	SI	A3CP	23846.0
7	Alien	A3C	1440.82
8	Alien	DFA3C	**2722.19**
9	Alien	A3CP	945.3
10	Enduro	A3C	0.77
11	Enduro	DFA3C	**497.29**
12	Enduro	A3CP	-82.5
13	Q*bert	A3C	21184.75
14	Q*bert	DFA3C	**21405.35**
15	Q*bert	A3CP	21307.5

Table 3: Experimental results for A3C architectures. AS denotes the average score in the final testing epoch. SI is the game of Space Invaders.

Conclusion and Future Work

In this paper, we introduce a novel paradigm to strengthen existing Deep Reinforcement Learning agents by providing them the ability to decide the time scale of executing an action in addition to deciding the specific action to execute. This scheme enables AI agents to dynamically decide how long a chosen action in the current state is to be repeated. The ability to decide dynamically, the extent of the repetition of an action can be seen as a skill in the direction of looking ahead (planning). Our scheme allows the agent to exert quick reflexes when required and to continue performing the same action as long as the chosen *macro-action* leads the agent to an advantageous (*valuable*) next state. Through this paradigm, we present an elegant way to introduce temporal structure in discrete-action space policies with the extent of repetition of the chosen action being decided based on the current state. We show empirically that this setup leads to significant improvement in performance, regardless of the underlying Deep Reinforcement Learning algorithm used, and regardless of the nature of the algorithm (off-policy or on-policy), with results on five relatively harder Atari 2600 domain games: Seaquest, Space Invaders, Alien, Enduro and Q*Bert. The improvement in performance has been achieved without much tuning of the DQN (Mnih et al. 2015) network parameters. In the case of Augmented A3C, *no* tuning of hyperparameters was performed. The dynamic time scale mechanism can be incorporated into any existing

deep reinforcement learning methods (both value and policy based) and an exhaustive analysis on its utility for different methods is an interesting direction for future work

Our work naturally leads to a parametrized policy setup, where each action has an associated parameter: its Action Repetition Rate (ARR). An Actor Critic setup similar to (Hausknecht and Stone 2016) to learn such structured policies with multiple time scales for both discrete and continuous action spaces is a compelling future direction. The structure in the policy naturally introduces temporal abstractions through *macro-actions* composed of the same action being repeated, with different lengths. Another direction to deal with action space explosion when consdiering more time scales is to investigate the use of action embeddings for value-based methods. In our setup, we do not consider the ability to stop executing a macro-action that the agent has committed to. However, this is a necessary skill in the event of unexpected changes in the environment while executing a chosen macro-action. Thus, *stop* and *start* actions for exiting and committing to plans can be augmented with the dynamic time scale setup for more robust planning.

Acknowledgements:

We would like to thank the anonymous AAAI reviewers for insightful feedback. We would also like to thank Karthik Narasimhan and Sherjil Ozair for useful comments.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 253–279.

Braylan, A.; Hollenbeck, M.; Meyerson, E.; and Miikkulainen, R. 2015. Frame skip is a powerful parameter for learning to play atari. *AAAI workshop*.

Gilbert, D. T., and Wilson, T. D. 2007. Prospection: Experiencing the future. *Science* 317(5843):1351–1354.

Gomez, F., and Miikkulainen, R. 1997. Incremental evolution of complex general behavior. *Adaptive Behavior* 5(3-4):317–342.

Hausknecht, M., and Stone, P. 2016. Deep reinforcement learning in parametrized action space. *4th International Conference on Learning Representations*.

Hausknecht, M.; Mupparaju, P.; Subramanian, S.; Kalyanakrishnan, S.; and Stone, P. 2016. Half field offense: An environment for multiagent learning and ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*.

Hingston, P. 2010. A new design for a turing test for bots. *IEEE Conference on Computational Intelligence and Games (CIG)*.

Konda, V. R., and Tsitsiklis, J. N. 2003. On actor-critic algorithms. *SIAM journal on Control and Optimization* 42(4):1143–1166.

Lake, B. M.; Ullman, T. D.; Tenenbaum, J. B.; and Gershman, S. J. 2016. Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Lin, L.-J. 1993. Reinforcement learning for robots using neural networks. *Technical Report, DTIC Document*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*.

Mnih, V.; enech Badia, A. P.; Mirza, M.; Graves, A.; Harley, T.; Lillicrap, T. P.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*.

Ortega, J.; Shaker, N.; Togelius, J.; and Yannakakis, G. N. 2013. Imitating human playing styles in Super Mario Bros. *Entertainment Computing, Elsevier* 4:93–104.

Sutton, R. S., and Barto, A. G. 1998. Introduction to reinforcement learning. *MIT Press*.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; Mansour, Y.; et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Conference on Neural Information Processing Systems*, volume 99, 1057–1063.

Thomaschke, R., and Dreisbach, G. 2013. Temporal predictability facilitates action, not perception. *Psychological science* 24(7):1335–1340.

Togelius, J.; Karakovskiy, S.; and Baumgarten, R. 2010. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, 1–8. IEEE.

Vafadost, M. 2013. Temporal abstraction in monte carlo tree search. *Masters thesis, Department of Computer Science, University of Alberta*.

Van Hoorn, N.; Togelius, J.; Wierstra, D.; and Schmidhuber, J. 2009. Robust player imitation using multiobjective evolution. In *2009 IEEE Congress on Evolutionary Computation*, 652–659. IEEE.

Wang, Z.; Schaul; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Watkins, C. J. C. H., and Dayan, P. 1992. Technical note: Q-learning. *Mach. Learn.* 8(3-4):279–292.