Fast and Data Efficient Reinforcement Learning from Pixels via Non-parametric Value Approximation

Alexander Long,[1] **Alan Blair,**[1] **Herke van Hoof**[2]

[1] University of New South Wales
[2] University of Amsterdam
alex.long@unsw.edu.au, blair@cse.unsw.edu.au, h.c.vanhoof@uva.nl

Abstract

We present Nonparametric Approximation of Inter-Trace returns (NAIT), a Reinforcement Learning algorithm for discrete action, pixel-based environments that is both highly sample and computation efficient. NAIT is a lazy-learning approach with an update that is equivalent to episodic Monte-Carlo on episode completion, but that allows the stable incorporation of rewards while an episode is ongoing. We make use of a fixed domain-agnostic representation, simple distance based exploration and a proximity graph-based lookup to facilitate extremely fast execution. We empirically evaluate NAIT on both the 26 and 57 game variants of ATARI100k where, despite its simplicity, it achieves competitive performance in the online setting with greater than 100x speedup in wall-time.

1 Introduction

Deep Reinforcement Learning (DRL) has become the standard approach to tackle complex non-linear sequential decision making problems exhibiting a high number of observational variables (Li 2017), a subset of which are image, or pixel, based. Although tabular Reinforcement Learning (RL) algorithms will learn (and in fact provably converge) in such environments, they are too slow to be practically useful. Policy and value function parametrization is thus typically used to facilitate learning, which has led to several breakthrough results (Berner et al. 2019; Badia and Piot 2020).

Advances in raw performance, however, have been achieved at the cost of increased sample, and computational complexity. Agent 57 (Badia and Piot 2020) for example, requires 256 actors collecting approximately 57 years of real-time game play per game. For systems tackling more complex environments, these numbers are significantly higher. In an online, real-world setting this presents a major roadblock and, as a consequence, research attention has begun to shift towards improving sample efficiency with the goal of broadening the applicability of Deep Reinforcement Learning (DRL) in real world control and robotics. In these domains, sample efficiency is critical as the cost of each interaction is high; a physical robot must actually move. Improvements in sample efficiency, however, typically come at
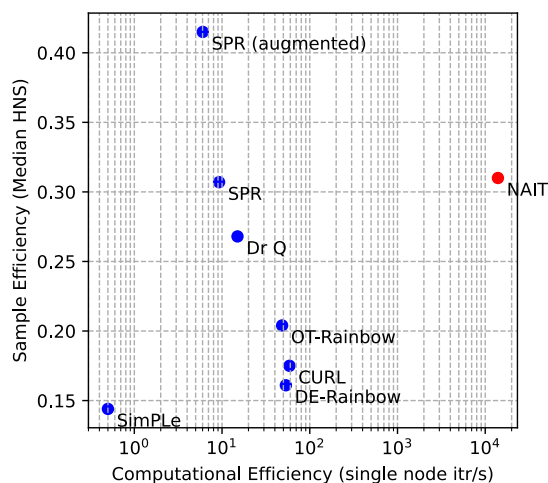
Figure 1: Comparison of our approach to other online sample efficient RL algorithms baselined on ATARI100k. NAIT achieves a >100x speedup over all other methods and is only outperformed in terms of median HNS by augmented SPR. Up and to the right is better. Algorithm labels are provided in Related Work.

the cost of increased computational complexity, where additional CPU cycles are used to "squeeze" more from the available data, through a learned model (Hafner et al. 2019), data augmentation (Laskin et al. 2020), or similar.

The dominant evaluation framework for online RL algorithms remains simulated environments. In this setting, high computational complexity is not a bottleneck on performance. It does, however, decrease the practical relevance of the associated results by increasing the cost (in dollar terms), and thus reducing accessibility (Henderson et al. 2018). Scientific progress in the area is also impacted, as is visible in the fact that the high computational cost of SimPLe (Kaiser et al. 2019) was one cause of the choice to evaluate on the 26 game "easy exploration" subset of the full 57 game suit, which made comparison to previous work difficult. It also make comparisons difficult, and expensive, for subsequent authors.

Computationally efficient approaches increase the scale

at which algorithms can operate. For this reason several works have made attempts to speed up the evaluation simulators, such as CuLE for ATARI (Dalton, Frosio, and Garland 2019). Few, however, have focused on improving the computational efficiency of the RL algorithms themselves, instead focusing on distributed implementations (Stooke and Abbeel 2018), or the use of specialized hardware (Cho et al. 2019). Increasing the wall-clock speed with which decisions can be reached also directly increases the responsiveness of the system, which greatly simplifies tasks such as tracking and control (Wagner et al. 2010).

In this paper, we present Nonparametric Approximation of Inter-Trace returns (NAIT), an algorithm that is *both* computation and sample efficient. NAIT makes use of fast approximate k-nearest neighbour algorithms to generalise locally, and approximate a value function from stored memory. We combine this with a novel inter-trace update that provides the stability of episodic returns as well as the ability to immediately learn from an experience prior to completion of the episode (an advantage that is particularly pronounced in the low sample domain). We make use of a Discrete Cosine Transform (DCT) based transformation as the state representation, with no use of a learned encoder, prior knowledge, domain specific observation transformations or data augmentation.

We focus our evaluation on the ATARI100k suite of environments (Kaiser et al. 2019) in the online setting. Our approach is able to achieve 0.31 Median Human Normalized Score (MHNS), outperforming several previous state-of-the-art approaches such as Dr. Q (Kostrikov, Yarats, and Fergus 2020) and CURL (Srinivas, Laskin, and Abbeel 2020), despite making no use of a learned representations. In addition to this high level of sample efficiency, our approach is over two orders of magnitude faster than the current state of the art, SPR (Schwarzer et al. 2020), in terms of wall-time. In a single thread, NAIT is able to complete training for one environment in 7.2 minutes, where SPR requires 3.5 hours on the same GPU-enabled node. We examine our approach with ablation and sensitivity analysis, demonstrating the ability of NAIT to trade-off performance for speed, as well as a high level of robustness across hyperparameter values. Because of NAIT's low execution time, we are able to provide baseline results under multiple evaluation settings and sample budgets, providing a strong baseline for future work.

2 Related Work

Recently significant focus has turned towards sample efficient RL in image-based environments. SimPLe (Kaiser et al. 2019) consists of a novel network architecture used to model ATARI forward dynamics in the observation space, integrated within a Dyna-style (Sutton 1990) training loop. This work introduced the ATARI100k benchmark, and achieved 0.144 median human normalised score (MHNS), significantly outperforming Rainbow (Hessel et al. 2018) and PPO (Schulman et al. 2017). Data Efficient Rainbow (DE-Rainbow) (van Hasselt, Hessel, and Aslanides 2019) was introduced to make use of more frequent value network updates per environment step, matching the update/step ratio used in SimPLe and achieved 0.161 MHNS while making

no use of a learned model. Concurrently, Kielak presented OT-Rainbow (Kielak 2020), a similarly "over-trained" but otherwise unaltered version of rainbow capable of achieving 0.26 MHNS.

Several works have identified representation learning as the bottleneck in this domain and have proposed techniques to accelerate it. CuRL (Srinivas, Laskin, and Abbeel 2020) introduced an auxiliary contrastive loss able to be utilized within existing RL algorithms. Dr. Q (Kostrikov, Yarats, and Fergus 2020), focused solely on image augmentation (random shifts and random crops of observations), and demonstrated their inclusion raised the performance of an efficient DQN implementation from 0.058 to 0.284 MHNS. SPR (Schwarzer et al. 2020) proposes a latent space forward dynamics model, combined with an auxiliary prediction loss, that achieves 0.415 MHNS when combined with an augmentation scheme similar to that in Dr. Q.

Recently, several works such as APT (Liu and Abbeel 2021) and VISR (Hansen et al. 2019) have considered an additional unsupervised pretraining phase prior to online learning. In this setting, agents are provided a 250M frame budget with no environment rewards, after which rewards are restored for the standard 100k steps. Although we do not adopt this setting to evaluate out agent, we include several results to help contextualize the performance of NAIT.

Non-parametric, or lazy learning (Sutton and Barto 1998) has been applied to ATARI previously. In the original work proposing the ALE environment (Bellemare et al. 2015), a Locality Sensitive Hashing (LSH) algorithm was proposed to convert pixel observations to binary features for SARSA(λ). Linear learners ingesting fixed features were examined in (Liang et al. 2016), and remain strong baselines. In Model Free Episodic Control (MFEC) (Blundell et al. 2016), a gaussian random projection to a 64-dimensional latent space is used on ATARI frames, however performance is poor, possibly due to the small latent space dimensionality chosen. Berariu et. al. (Berariu and Gogianu 2017) augmented MFEC with sparse projections (Achlioptas 2003) along with variance normalisation in the projected space and demonstrated a improvement in performance, along with significant computational gains on ATARI.

3 Method

3.1 Background

We consider the standard formulation and notation (Sutton and Barto 1998) of an agent interacting with some environment at discrete time steps to maximise total expected reward. The environment is modelled as a Partially Observable Markov Decision Process (POMDP) $(\mathcal{O}, \mathcal{A}, P, r, \gamma)$ with a D dimensional observation space $o \in \mathcal{O} : \mathbb{R}^D$ and discrete action space $a \in \mathcal{A}$. Traces are denoted as $\tau = \{(o_0, a_0, r_0), (o_1, a_1, r_1), \ldots\}$. $P = \text{Prob}(o'_t | o_{\leq t}, a_{\leq t})$ denotes the transition probability dynamics, $r : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function and $\gamma \in [0, 1)$ denotes a discount rate. We make use of a fixed projection $\phi : \mathbb{R}^{4 \times D} \to \mathbb{R}^F$, to convert a stack of four observations (as is standard in ATARI) to a more compact space $s \in \mathcal{S} : \mathbb{R}^F$, $s_t = \phi(o_t, o_{t-1}, o_{t-2}, o_{t-3})$ where $F \ll D$. We treat s

as the base state henceforth, with $P = \text{Prob}(s'_t|s_t, a_t)$ and $r_t = r(s_t, a_t)$. The discounted return from state s_t and action $a_t \in \mathcal{A}$ taken at time t, is defined as $G(s_t, a_t) \doteq \sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k)$ where H is the time horizon of the episode. For brevity, we often abbreviate this as $G_t = G(s_t, a_t)$ indicating the return from time t onwards. If a range is specified as in $G_{t:T}$, this denotes the return from time t until T. Our goal is to learn the optimal policy $\pi^*(s_t)$ that maximises $\mathbb{E}_P[G(s_t, a_t)]$ for all $s_t \in \mathcal{S}$ where $a_t \sim \pi^*(s_t)$. We make use of a memory M, where $M_a(s)$ refers to the value in memory for action a and state s.

At a high level, our approach acts by representing all observations via a fixed projection, calculating Q-values via local approximation, and updating these values iteratively as new rewards are encountered. The final algorithm is shown in Algorithm 1. We outline each component below.

3.2 Representation

As a fixed representation, we use a Discrete Cosine Transform (DCT) and select the first F DCT coefficients ordered by frequency. The major advantage of the DCT as representation is that it is able to capture broad detail about the image extremely compactly, while being fast to compute, as $\mathcal{O}(D \log D)$ implementations based on the FFT exist. This is faster even than a random projection at $\mathcal{O}(FD)$ for many values of F. We make use of DCT II, an orthonormal transform $\mathbb{R}^N \to \mathbb{R}^N$,

$$\text{DCT: } X_i = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)i\right] i = 0, \ldots, N-1. \quad (1)$$

We wish to apply this to the frame-stack obtained from the environment, and thus stack the four frames in a 2x2 grid to ensure equal weighting of vertical and horizontal frequencies, and apply the 2D DCT II. To compress the state, we discard high frequency coefficients and select the top F;

$$s_t = \phi\left(o_t, o_{t-1}, o_{t-2}, o_{t-3}\right) \quad (2)$$

$$= \left[\text{DCT}\left(\text{DCT}\begin{bmatrix} o_t & o_{t-1} \\ o_{t-2} & o_{t-3} \end{bmatrix}^T\right)^T\right]_{ij \; 0 \le i,j \le \sqrt{F}}.$$

As the DCT is the only component of our approach specific to images, we also present results with domain-agnostic random projections as the representation (see Table 3). A random projection transforms the observation data (of dimension D) to a lower dimensional space (of dimension F) $\phi : x \to x\mathbf{R}$, where $\mathbf{R} \in \mathbb{R}^{D \times F}$. We fix F to be the same as the number of DCT coefficients used. For each element $u_{i,j} \in \mathbf{R}$:

$$u_{i,j} = \sqrt{s} \begin{cases} +1 & \text{with prob. } \frac{1}{2s} \\ 0 & \text{with prob. } 1 - \frac{1}{s} \\ -1 & \text{with prob. } \frac{1}{2s} \end{cases} . \quad (3)$$

When $s = \sqrt{D}$, $\mathbf{R}$ is a *very sparse random projection* (Li, Hastie, and Church 2006). When $s = 3$, the projection becomes the more standard *sparse random projection* (Achlioptas 2003).

3.3 Non Parametric Value Function Approximation

We aim to learn a local approximation of the the discounted return $Q(s, a) \simeq G(s, a)$ where $a \sim \pi^*(s)$. Q-values are estimated from action-specific memory buffers M_a, of which frozen copies, M'_a are made at the start of each episode, $M'_a(s) \xleftarrow{\text{copy}} M_a(s)$ at $t = 0$. This memory is updated online with learning rate α according to the learning rule in Alg. 2. Q-values are directly estimated from memory,

$$Q(s, a|k) = \begin{cases} M_a(s) & s \in M_a \\ \frac{\sum_{s' \in L(s,a)} K(s,s') M_a(s')}{\sum_{s' \in L(s,a)} K(s,s')}, & s \notin M_a, |M_a| \ge k \\ 0 & s \notin M_a, |M_a| < k \end{cases} \quad (4)$$

where $L(s, a) = \{s_1, \ldots, s_k\}$ is the result of a nearest-neighbour (by L^2 distance) lookup of s on M_a, and $K : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a kernel function that dictates a weighting over neighbours. In all experiments, to smooth the interpolation, we use a normalized tricubic kernel. The kernel calculates the weighting for each neighbour $s' \in L(s, a)$ as,

$$K(s, s') = \left(1 - \left(\frac{||s - s'||_2}{\max_{\hat{s} \in L(s,a)} ||s - \hat{s}||_2}\right)^3\right)^3. \quad (5)$$

Normalization is required as the standard kernel has support $[0, 1]$ and the distance between two states can clearly exceed this. Consequently, we normalize by the maximum L^2 distance to all states in $L(s, a)$. This has significant consequences in that the bandwidth of the kernel expands and contracts depending on the density of the local neighbourhood at the query point due to the fixed k. This results in broad generalisation across large areas when samples are sparse, and fine-grained interpolation when a region has been heavily sampled. The particular choice of kernel is not a critical component of NAIT and a gaussian, bicubic or similar may easily be substituted. Empirically, we found the tricubic kernel to give a slight bump in performance (see Table 3).

Acting is done greedily with ties broken by an estimate of the visitation density calculated with information from the already completed k-nn lookup. This is done by a softmax distribution over the mean L^2 distance to the neighbouring states found on the lookup for that (s, a) pair (if there is an exact match, the distance is considered 0). Given a candidate set of maximum valued actions $C = \arg\max_{a \in \mathcal{A}} Q(s, a)$,

$$\pi_{\text{tiebreak}}(s) \sim \text{softmax}_{a \in C}\left(\frac{1}{k} \sum_{s' \in L(s,a)} ||s - s'||_2^2\right). \quad (6)$$

If there is no tiebreak ($|C| = 1$), the policy is standard greedy action selection, $\pi \sim \arg\max_{a \in \mathcal{A}} Q(s, a)$. This distance-based tie-breaking is the only form of exploration explicitly added to NAIT.

3.4 Inter-Trace Update

Monte-Carlo returns are desirable in the low-sample domain as they do not propagate approximation error. However, they come with the major drawback that updates cannot be carried out until the end of a trace. When the ratio of

---

Algorithm 1: NAIT: Training and Acting

Initialise: Memory $M_a \rightarrow \{\}$ for all $a \in \mathcal{A}$
while *training* **do**
    $o \leftarrow$ reset environment, empty trace history $\tau \leftarrow \{\}$, $\mathbf{G} \leftarrow \{\}$, $\mathbf{\Gamma} \leftarrow \{\}$, $M_a' \xleftarrow{\text{copy}} M_a$ for all a
    for $t = 0$ to epsiode horizon H **do**
        Get DCT representation $s \leftarrow \phi(o_t, o_{t-1}, o_{t-2}, o_{t-3})$
        for $a \in \mathcal{A}$ **do**
            Get estimate of $Q(s, a)$ using a kernel $K(s, s')$, set of neighbours $L(s, a) = \{s_1, \ldots, s_k\}$

$$Q(s, a; k) = \begin{cases} M_a(s) & s \in M_a \\ \frac{\sum_{s' \in L(s,a)} K(s,s') M_a(s')}{\sum_{s' \in L(s,a)} K(s,s')}, & s \notin M_a, \ |M_a| \geq k \\ 0 & s \notin M_a, \ |M_a| < k \end{cases}$$

        end for
        $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$ breaking ties with $\pi_{\text{tiebreak}}(s) \sim \text{softmax}_{a \in C} \left(\frac{1}{k} \sum_{s' \in L(s,a)} ||s - s'||_2^2 \right)$
        Act $o, r \leftarrow \text{env}(\pi(s))$
        Add (s, a) to τ
        Get inter-trace targets (Alg. 3) and Update$(\tau, \mathbf{Q}_{\text{IT}})$ (Alg. 2)
    end for
    Get final returns and Update$(\tau, \mathbf{G})$ (Alg. 2)
end while

Algorithm 2: Batch Update

Input τ: (s, a) tuples list, $\mathbf{Q}$: targets list, α: learning rate
for $(s, a), Q \in \text{zip}(\tau, \mathbf{Q})$ **do**
    if $s \in M_a'$ **then**
        $M_a(s) \leftarrow M_a'(s) + \alpha(Q - M_a'(s))$
    else
        $M_a(s) \leftarrow Q$
    end if
end for

---

average trace length to total interactions is high, as it is in ATARI100k, this property has a pronounced effect. To address this, we introduce a novel inter-trace update allows the incorporation of new information immediately upon receiving it, as is possible with bootstrapped updates, while retaining the stability of MC returns. We show this update can be calculated in constant time with respect the episode length.

During an ongoing trace, at time $t < T$, for an (s, a) pair first experienced at time i, the standard MC target $Q_{\text{target}}(s, a) = G_{i:T}(s, a)$, is not able to be calculated as the episode has not completed. We can, however, incorporate the rewards $\{r_i \ldots r_t\}$ already received into a new target as an approximation to the MC return, bootstrapping off a fixed copy, $Q'(s, a)$ of the current estimator. Here $Q'(s, a)$ indicates Eq. 4, calculated with the frozen memory M_a'. This new target is;

$$\begin{aligned} Q_{\text{target}}(s_i, a_i) &= r_i + \gamma r_{i+1} + \ldots + \gamma^{t-i} r_t \\ &\quad + \gamma^{t-i+1} Q'(s_{t+1}, a_{t+1}) \\ &= G_{i:t-1} + \gamma^{t-i} r_t + \gamma^{t-i+1} Q'(s_{i+1}, a_{i+1}) \end{aligned}$$
(7)

In practice we vectorize this operation so as to calculate the targets for all (s, a) pairs visited in the trace in one step,

ensuring updates can be computed in constant time. Specifically, at each time t during a trace, we have encountered n unique (s, a) pairs, with the time of their first visit given by $\mathbf{t} = [t_0, t_1, \ldots, t_n]$. We can compute the target vector,

$$\mathbf{Q}_{\text{target}} = [Q_{\text{target}}(s_{t_0}, a_{t_0}), \ldots, Q_{\text{target}}(s_{t_n}, a_{t_n})] \quad (8)$$
$$= \mathbf{G} + \mathbf{\Gamma} Q'(s_{t+1}, a_{t+1}) \quad (9)$$

where

$$\mathbf{G} = [G_{t_0:t}, G_{t_1:t}, \ldots, G_{t_n:t}], \quad (10)$$
$$\mathbf{\Gamma} = [\gamma^{t-t_0}, \gamma^{t-t_1}, \ldots, \gamma^{t-t_n}] \quad (11)$$

$\mathbf{G}$ is simply the running discounted reward, and $\mathbf{\Gamma}$ the discount that should be applied to the next reward, for each (s, a). At the beginning of a trace, all vectors are cleared, then are iteratively updated at each time-step following Alg. 3.

Algorithm 3: Vectorized Inter-trace Targets

Input r_t, $Q'(s_{t+1}, a_{t+1})$, $\mathbf{G}$, $\mathbf{\Gamma}$
Append 1 to $\mathbf{\Gamma}$
$\mathbf{G} \leftarrow \mathbf{G} + \mathbf{\Gamma} r_t$
$\mathbf{\Gamma} \leftarrow \gamma \mathbf{\Gamma}$
$\mathbf{Q}_{\text{IT}} \leftarrow \mathbf{G} + \mathbf{\Gamma} Q'(s_{t+1}, a_{t+1})$
Return $\mathbf{Q}_{\text{IT}}, \mathbf{G}, \mathbf{\Gamma}$

This update allows us to perform training once per step, as opposed to once per episode, instantly incorporating new information into our policy. The inter-trace update is similar to an n-step update, but n is always the maximum possible value for each state, and can be thought of as "doing as much MC as possible" - overwriting previous updates as new information becomes available, as opposed to bootstrapping off them. This is an important property as it does

not propagate errors present in the value function approximation whereas standard n-step updates do. While the calculation of targets can be done in constant time, each value update is $\mathcal{O}(1)$, and we do not need to update each pair at each step. Instead NAIT only utilizes new targets when $(t_0 - t)$ $(\mathrm{mod}\ r) = 0$. In all experiments we set $r = 50$.

3.5 Projected Small World Graphs

Our method relies on being able to both add high-dimensional data to memory (Alg. 2), and query this memory (eq. 4) for neighbours with low computational cost. With standard fast k-NN algorithms, such as KD-trees (Bentley 1975), this is not possible as they require re-indexing each time underlying data is changed which carries a large cost.

Instead, we make use of a Proximity Graph (PG) approach, Hierarchical Navigable Small World graphs (HNSW) (Malkov and Yashunin 2018). PG approaches are based around incremental index construction, and thus adding new points can be done cheaply. We make use of a specific standalone HNSW implementation (nsmlib 2021) that leverages this feature to allow incremental updating, along with the ability to control search time parameters `ef` and `M` which directly effect the recall/query time trade-off (for full details see (Malkov and Yashunin 2018)). In addition, querying is an order of magnitude faster than a KD-tree on densely clustered data at high (> 0.8) recalls (Aumüller, Bernhardsson, and Faithfull 2020).

4 Results

4.1 Image Random Walk

As a base environment, we first introduce a 19-state deterministic MDP with image observations and $\gamma = 0.9$. The 19 states are arranged in a flat line, in order of index. At each state two actions are available, a_0 to the left, action a_1 to the right. State 15 is special, here the actions are reversed (i.e. a_0 moves to the right, action a_1 to the left). A transition diagram is included in the supplementary material.

At all transitions the reward is zero, except for action a_0 from state 19, which yields a reward of 1. State 9 is the initial state. Observations for each state are noisy grayscale blocks, with the intensity corresponding to the states index $o_{i,j} = \mathcal{N}(n, \sigma), i, j \in \{1..D\}, n \in \{1..19\}$ where D is the height and width of the observations and n is the index of the state, normalized to a range of $[-1, 1]$. We choose $D = 5$ and $\sigma = 0.1$. The optimal Q-functions for the system are shown in the supplementary material, where it can be seen that simple policies such as "always move right" cannot succeed due to the action switching at State 15.

We also plot the reward (see supplementary material) obtained at the end of each episode for a tuned DQN agent and our approach. Both methods settle on the optimal policy, however our memory based approach learns almost immediately, whereas the DQN is still converging after 100k steps. This result highlights that in some cases it is simply much easier to store visited returns and generalise locally at inference time, as opposed to trying to fit a global value function.
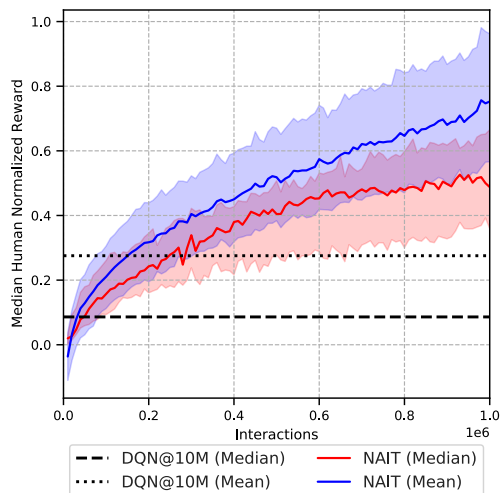

Figure 2: HNS over 1M interactions averaged over the full 57 ATARI game set. Shaded region indicates 90% confidence interval. DQN performance after 10M interactions is also shown, as comparable sample efficient methods do not evaluate on the full game set. NAIT requires only 150k interactions to outperform DQN@10M on both metrics.

4.2 ATARI100k

ATARI100k was first introduced as a testbed for SimPLe (Kaiser et al. 2019) and consists of 26 "easy exploration" ATARI2600 games. Since its introduction numerous works have focused solely on ATARI100k (van Hasselt, Hessel, and Aslanides 2019; Kielak 2020; Kostrikov, Yarats, and Fergus 2020; Srinivas, Laskin, and Abbeel 2020; Schwarzer et al. 2020). In our work, we report results for ATARI100k, but also evaluate on the 57 game set used more frequently in the wider RL literature. We do this both to provide a more robust baseline for future work, and to evaluate the generality of NAIT, as we do all hyperparameter tuning on the 26 game set alone, and make no changes when applying to the full evaluation suite. This setup was originally suggested by the ALE authors (Machado et al. 2018), and demonstrates that an approach is not overfitted to the task suit, however to date it has not been commonly adopted. We include a comparison to state-of-the-art sample efficient methods in Table 1. The unsupervised setting is included to contextualize NAIT's high level of performance on the full 57 game set, however we do not evaluate NAIT in this setting. Full settings for ATARI are listed in the supplementary material.

We run all experiments with 10 seeds, and evaluation is done on a rolling basis following the recommendations of (Machado et al. 2018), with the mean of the most recent 5 episodes logged for each seed. The DQN and SARSA(λ) baselines are also reproduced from this work. Final scores are the mean score of 50 episodes, run after the completion of training, following (Schwarzer et al. 2020). Our main metric is the Human Normalized Score, HNS $= \frac{\text{agent}-\text{random}}{\text{human}-\text{random}}$ achieved within a fixed interaction budget. We

		ATARI100K		All 57 Games	
Method	**Samples**	**Mdn**	**Mean**	**Mdn**	**Mean**
Online (low budget)					
SimPLe	100K	14.39	44.3	-	-
DE-R	100K	28.5	16.1	-	-
OT-R	100K	26.4	20.4	-	-
CURL	100K	17.5	38.1	-	-
DrQ	100K	28.42	35.7	-	-
SPR[†]	100K	**41.5**	**70.4**	-	-
NAIT	100K	31.21	36.23	**17.57**	**36.51**
Online (medium budget)					
NAIT	250K	39.46	52.86	26.54	48.81
PPO	500K	20.93	43.74	-	-
NAIT	500K	46.73	69.97	43.63	66.46
NAIT	750K	60.64	79.38	50.05	73.26
NAIT	1M	**60.60**	**90.38**	**52.04**	**82.81**
SARSA[‡]	2M	35.85	72.40	28.27	57.75
DQN	2M	27.8	52.95	8.61	27.55
Unsup. Pretraining (250M budget) + Online Finetuning					
VISR	100K	9.5	128.07	6.81	102.31
APT	100K	47.5	69.55	**33.41**	**47.73**
APS*	100K	**51.45**	**87.59**	-	

Table 1: Final performance measures. *No shared encoder. [†]Data augmented. [‡] SARSA(λ) with Blob-PROST features.

Algorithm	1 Node Itr/s	Walltime (hours)
SimPLe	0.5	3166.67
DE-R	53.4	29.65
OT-R	48.4	32.71
CURL	58.4	27.11
DrQ	15	105.56
SPR	9.25	171.17
SPR (augmented)	6.03	262.58
NAIT	**13962.24**	**0.11**

Table 2: Average single-node iterations/s and walltime required to complete the full 57 game for various methods on a single node.

Component	Replacement	MHNS	Itr/s
NAIT		**31.2**	163.4
DCT projection	Very sparse	21.5	121.3
DCT projection	Sparse	28.1	91.4
DCT projection	None	-	1.5
Tiebreaking	Uniform	12.1	125.3
Intertrace update	Monte-Carlo	13.6	141.2
Tricubic kernel	Mean	28.4	161.4
HNSW	KD tree	-	0.3

Table 3: Ablations. Median HNS and single-thread itr/s reported for the 26 game subset and 100K iterations.

primarily report the Median HNS (MHNS) across games as the mean tends to be dominated by outlier performance.

As speed is a major feature of NAIT, we log median iterations per second (itr/s) across games. Itr/s is calculated by dividing the time to complete the environment by 100k, averaged over ten seeds. Per-node speeds are the total throughput when all threads are allocated on a single AWS C5d.24xlarge (48 cores). This metric is indicative of real-world performance but is dependent on the number of cores present in the node. For this reason we also report per-thread speeds which are independent of core count but do not consider factors such as I/O blocking, although in practice we observe these effects to be minor. SPR, SimPLE and DE-Rainbow speeds in Table 2 are reproduced from (Schwarzer et al. 2020), with Dr Q and CURL and OT-Rainbow benchmarked from from open-source implementations running on P3.16xlarge (32 core, 8xP100 GPU) instances.

NAIT achieves a high level of performance, outperforming all methods with the exception of augmented SPR on the ATARI100K benchmark (see Table 1). This is achieved with a hugely reduced computational cost - over 100x faster than SPR and 1000x faster than SimPLe (see Table 2).

5 Analysis

5.1 Ongoing Learning

NAIT demonstrates the high level of performance that a nonparametric approach can learn when environment interactions are limited. However, due to the fixed representa-

tion, NAIT's performance will saturate once the importance of a good representation (i.e. non-local generalization) outweighs the importance of quickly updating the value function or policy. As stated, the 100k domain is of interest in its own right, however we are also interested in quantifying the point at which design choices such as a fixed representation begin to limit performance.

Using the same set of hyperparameters tuned for the ATARI100k benchmark (26 games, 100k interactions), we carried out a longer run over 1M interactions, and included all 57 games in the standard evaluation set (Fig. 2). We made no other alterations to NAIT or its hyperparameters. Learning is highly stable and indicates NAIT possesses a high level generality. Per-game performance during training is visualized in the supplementary material.

5.2 Ablations

When removing components, if a replacement is required, we choose what is either simplest or is already standard. For example, in place of a tricubic kernel, we use a uniform kernel, or instead of HNSW we use a KD-tree. Results are listed in Table 3.

We find that DCT components provide better quality representations with regard to performance *and* are faster to compute. Computation on the raw frames was prohibitively slow due to the resulting high dimensionality of the k-NN keys.

One unusual feature is the impact of distance based

tiebreaking, in comparison to randomly breaking ties. This is likely because early in learning this encourages exploration based on novel states, as all estimates are zero, rather than simply ensuring a roughly uniform selection of actions. This is supported by the strong level of performance on the unseen 'hard' exploration games, despite no other mechanism of exploration being built into NAIT. Of lesser importance is the use of tricubic kernel which is shown to result in an increase in performance with only a small cost in computation speed in comparison to a uniform kernel (i.e. take the mean), however this increase is minor.

5.3 Effect of Hyperparamters

Higher representation dimension F tends to be both slower, and perform worse as it is increased past $F = 289$ (see supplementary material). We hypothesise this is due to poorer recall of HNSW at higher dimensions overshadowing information preservation gained by a higher F. This could potentially be offset by increasing the recall of the lookup via the `ef` and `M` parameters, however this would come at the cost of a significant reduction in computational efficiency. Interestingly, as F is held constant and `M` increased, performance is stable up to $M = 160$, but drops sharply after this. This would indicate some level of noise in the value estimate is beneficial, and perfect recall is in fact not desirable for NAIT. Setting `M` too low however also degrades performance as expected.

Such high performance with low F indicates that the information required to learn basic play is present in a surprisingly small number of DCT coefficients. One implication for ATARI100k is that learning a good representation is significantly less important in early learning than being able to quickly assign credit.

From our sweeps, k appears to have a clear optimal value across all games (see supplementary material). This is expected as k effectively balances local generalization with the smoothness of the value estimate, and setting k too large removes information from the value estimate. In contrast, the choice of learning rate does not effect performance smoothly, however the high-level metrics are surprisingly strong even at large α, displaying a level of robustness to the choice of α that departs significantly from neural-based approaches. One explanation is due to α not being considered at new (s, a) pairs where the new value is set directly. This forces NAIT into a high-learning rate setting by default.

6 Conclusion

We have introduced a locally generalising approach to RL that is both sample and computationally efficient, achieving state of the art performance in the low sample regime on ATARI while requiring two orders of magnitude less walltime. Due to the very low walltime and computational requirements, we hope that future work adopts our algorithm as a benchmark when demonstrating the efficacy of learned model or learned representation approaches. Further work may investigate the possibility of our approach as a quick but simple initial learner that can merge into, or inform, a more complex learner that does produce useful representa-

tions, via mechanisms such as behavioural cloning or similar. This has exciting potential to significantly improve the sample efficiency of established methods while maintaining their asymptotic performance.

References

Achlioptas, D. 2003. Database-friendly random projections. In *Journal of Computer and System Sciences*, volume 66, 671–687.

Aumüller, M.; Bernhardsson, E.; and Faithfull, A. 2020. ANN-benchmarks. *Information Systems*, 87: 101374.

Badia, A. P.; and Piot, B. 2020. Agent57: Outperforming the atari human benchmark. *arXiv preprint arXiv:2003.13350*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2015. The arcade learning environment: An evaluation platform for general agents. In *IJCAI*, volume 2015-Janua, 4148–4152. ISBN 9781577357384.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9): 509–517.

Berariu, T.; and Gogianu, F. 2017. Random Projection Based Representations for Learning Policies in Deterministic Atari Games. In *CSCS*, 233–238. ISBN 9781538618394.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Blundell, C.; Uria, B.; Pritzel, A.; Li, Y.; Ruderman, A.; Leibo, J. Z.; Rae, J.; Wierstra, D.; and Hassabis, D. 2016. Model-Free Episodic Control. *arXiv:1606.04460v1*.

Cho, H.; Oh, P.; Park, J.; Jung, W.; and Lee, J. 2019. Fa3c: Fpga-accelerated deep reinforcement learning. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 499–513.

Dalton, S.; Frosio, I.; and Garland, M. 2019. Accelerating reinforcement learning through gpu atari emulation. *arXiv preprint arXiv:1907.08467*.

Hafner, D.; Lillicrap, T.; Ba, J.; and Norouzi, M. 2019. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Hansen, S.; Dabney, W.; Barreto, A.; Van de Wiele, T.; Warde-Farley, D.; and Mnih, V. 2019. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 3215–3222.

Kaiser, L.; Babaeizadeh, M.; Milos, P.; Osinski, B.; Campbell, R. H.; Czechowski, K.; Erhan, D.; Finn, C.; Kozakowski, P.; Levine, S.; et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*.

Kielak, K. 2020. Importance of using appropriate baselines for evaluation of data-efficiency in deep reinforcement learning for Atari. arXiv:2003.10181.

Kostrikov, I.; Yarats, D.; and Fergus, R. 2020. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. arXiv:2004.13649.

Laskin, M.; Lee, K.; Stooke, A.; Pinto, L.; Abbeel, P.; and Srinivas, A. 2020. Reinforcement Learning with Augmented Data. *arXiv preprint arXiv:2004.14990*.

Li, P.; Hastie, T. J.; and Church, K. W. 2006. Very sparse random projections. In *SIGKDD*, volume 2006, 287–296. ISBN 1595933395.

Li, Y. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Liang, Y.; Machado, M. C.; Talvitie, E.; and Bowling, M. 2016. State of the Art Control of Atari Games Using Shallow Reinforcement Learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 485–493.

Liu, H.; and Abbeel, P. 2021. Behavior from the void: Unsupervised active pre-training. *arXiv preprint arXiv:2103.04551*.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *JAIR*.

Malkov, Y. A.; and Yashunin, D. A. 2018. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1–1.

nsmlib. 2021. Hnswlib - fast approximate nearest neighbor search. https://github.com/nmslib/hnswlib. Accessed: 2022-2-22.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. arXiv:1707.06347.

Schwarzer, M.; Anand, A.; Goel, R.; Hjelm, R. D.; Courville, A.; and Bachman, P. 2020. Data-efficient reinforcement learning with momentum predictive representations. arXiv:2007.05929.

Srinivas, A.; Laskin, M.; and Abbeel, P. 2020. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. arXiv:2004.04136.

Stooke, A.; and Abbeel, P. 2018. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*.

Sutton, R. S. 1990. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In *Machine Learning Proceedings*. Elsevier.

Sutton, R. S.; and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edition. ISBN 0262193981.

van Hasselt, H. P.; Hessel, M.; and Aslanides, J. 2019. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32.

Wagner, D.; Reitmayr, G.; Mulloni, A.; Drummond, T.; and Schmalstieg, D. 2010. Real-Time Detection and Tracking for Augmented Reality on Mobile Phones. *IEEE Transactions on Visualization and Computer Graphics*, 16(3): 355–368.