HDPG: Hyperdimensional Policy-based Reinforcement Learning for Continuous Control

Yang Ni[1], Mariam Issa[1], Danny Abraham[1], Mahdi Imani[2], Xunzhao Yin[3] and Mohsen Imani[1*]
[1]University of California Irvine, [2]Northeastern University, [3]Zhejiang University
[*]Email: m.imani@uci.com

ABSTRACT

Traditional robot control or more general continuous control tasks often rely on carefully hand-crafted classic control methods. These models often lack the self-learning adaptability and intelligence to achieve human-level control. On the other hand, recent advancements in Reinforcement Learning (RL) present algorithms that have the capability of human-like learning. The integration of Deep Neural Networks (DNN) and RL thereby enables autonomous learning in robot control tasks. However, DNN-based RL brings both high-quality learning and high computation cost, which is no longer ideal for currently fast-growing edge computing scenarios.

In this paper, we introduce HDPG, a highly-efficient policy-based RL algorithm using Hyperdimensional Computing. Hyperdimensional computing is a lightweight brain-inspired learning methodology; its holistic representation of information leads to a well-defined set of hardware-friendly high-dimensional operations. Our HDPG fully exploits the efficient HDC for high-quality state value approximation and policy gradient update. In our experiments, we use HDPG for robotics tasks with continuous action space and achieve significantly higher rewards than DNN-based RL. Our evaluation also shows that HDPG achieves 4.7× faster and 5.3× higher energy efficiency than DNN-based RL running on embedded FPGA.

ACM Reference Format:
Yang Ni[1], Mariam Issa[1], Danny Abraham[1], Mahdi Imani[2], Xunzhao Yin[3] and Mohsen Imani[1*]. 2022. HDPG: Hyperdimensional Policy-based Reinforcement Learning for Continuous Control. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530659

1 INTRODUCTION

Intelligent robotics, as one of the most important branches and early goals of Artificial Intelligence (AI), has drawn great attention along with fast developments in the field of AI. Robots need to have human-like intelligence, which is adaptive to task requirements and, more importantly, capable of learning similar to us humans [1]. Its adaptability and intelligence are therefore closely connected to recent advancements in Reinforcement Learning (RL), which has shown beyond-human performance in multiple applications [2, 3]. There are a few distinguishing RL properties that attract researchers: (1) it does not rely on labeled training datasets; (2) it learns and adapts continuously using environment observations. In other words, the outcome of RL perfectly suits the definition of intelligent robotics, which is a self-learning agent that automatically learns a task through human-like trial-and-error. Moreover, compared to supervised learning, RL methods are appealing because, in reality, robotics tasks have fast-changing environments and the examples of expected behavior are hard to collect or simply unavailable.

Current RL algorithms are being categorized into value-based and policy-based methods. The value-based method learns a value function that evaluates every possible action choice at each time step with the current observation of the environment. It then follows a greedy policy to select the action with the highest value. The well-known Q-learning [4] is a typical value-based RL algorithm, in which the Q-value function, given a particular action-state pair, approximates the future accumulated rewards. However, in policy-based RL, the agent directly learns the optimal policy that guides its selection of actions under different conditions. The target of both RL methods is to maximize the total rewards. While the value-based methods try to optimize the reward for each action separately, policy-based methods view this target as a single optimization problem conditioned on a parameterized policy. The advantages of policy-based RL over its value-based counterpart include a smoother learning curve and the capability to deal with tasks with a continuous action space. The latter one is especially important for real-world robotics control tasks, where the main focus is on policy-based RL algorithms.

Most prior works used the gradient ascent method to update the parameterized policy towards higher rewards [5, 6]. For example, the REINFORCE algorithm [7], a generalized form of policy gradient RL, is used for skill acquisition in robotics [8]. These gradient-backed methods make it easier to tackle continuous control tasks that require real-valued stochastic policies, while Q-learning methods require action discretization or function approximation for the continuous action space [9, 10]. However, the vanilla policy gradient suffers from the low learning efficiency and high reward variance. Later methods such as Actor-Critic [11] solve the high variance problem by including a value-based critic. Nowadays, most modern policy-based RL algorithms, such as TRPO [12] and PPO [13] are closely connected to Deep Neural Networks (DNN) because tasks have more complicated environments and a higher requirement for learning quality. Using DNN in the policy network of RL agents brings increased power to discover the optimal action decision, which helps shape the recent success of policy-based RL, including Atari games [12, 13], system resource management [14] and robot control [15]. While traditional RL methods struggle to handle the large action and state spaces in these applications, DNN provides approximated yet good enough solutions. However, with edge computing becoming favorable in recent RL applications, a complex DNN-based RL algorithm leads to heavy computation burdens that are not friendly to devices with limited computing power.

In this paper, we bring a more efficient machine learning method to the field of RL. Instead of using DNN, we power the policy-based

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530659

RL algorithm with the novel brain-inspired Hyperdimensional Computing (HDC) [16]. While DNN mimics neuron connections in human brains using deep hierarchical networks, HDC focuses on how information is represented and memorized in the brain. HDC is motivated by an observation that human memory, perception, and cognition rely on neural activities in large dimensions. Therefore, HDC encodes low-dimensional inputs to another space where the output encoded vectors have significantly higher dimensions, i.e., Hypervectors. Although HDC operates in high dimensions, its computation is highly efficient because of the hardware-friendly parallelizable hypervector operations, e.g., element-wise addition and multiplication. These operations are designed to realize brain-like learning and reasoning, with which multiple prior works have presented the successful application of HDC. For example, in human-activity recognition [17], bio-signal processing [18], manufacturing [19], and brain-like reasoning [20], HDC is capable of achieving high quality results comparable to Support Vector Machines (SVM) and Multi-layer Perceptron (MLP). Different HDC hardware acceleration designs are also proposed to exploit the efficient hypervector operations, using FPGA and emerging processing in-memory technologies [21–23]. Considering how current HDC works successfully contribute to supervised and unsupervised learning problems, we believe that RL tasks could also benefit from the higher efficiency of HDC.

In this paper, we propose HDPG, a novel hyperdimensional policy-based reinforcement learning that fully utilizes the efficient HDC operations and targets at RL robotics tasks with continuous action space. Our contributions in this paper are listed below:

- To the best of our knowledge, HDPG is the first hyperdimensional policy-based RL algorithm to solve general continuous control tasks such as robot control. Compared to DNN-based policy-based RL, HDPG utilizes highly-parallelizable HDC operations for decision-making and shows comparable performance in experiments. Most importantly, the high efficiency of our HDPG brings the greater potential to low-power RL at the edge.
- We also design a hyperdimensional critic that uses efficient HDC regression to provide a high-quality state value approximation. This critic then functions as the baseline in the actor training process and reduces the variance of the policy gradient. For HDC regression, we utilize a distance-preserving exponential encoder that maintains the lower-dimension distance in the hyperdimensional space.
- We verify our HDPG in two simulated robot control tasks with continuous action space. We compare the performance and runtime of HDPG with the DNN-based Proximal Policy Optimization (PPO) algorithm, a state-of-the-art policy-based RL method. Our experiments on CPU platform show that, comparing to PPO, HDPG achieves higher quality results and also provides better learning efficiency in both tasks. Our evaluation also shows that HDPG training achieves 4.7× faster and 5.3× higher energy efficiency than PPO running on embedded FPGA.

2 PRELIMINARIES

2.1 Hyperdimensional Computing

A unique property of the hyperdimensional space is that we can create a large number of near-orthogonal hypervectors by randomly sampling elements from $\{-1, 1\}$ [24]. We consider two randomly sampled hypervectors $\vec{\mathcal{H}}_1$ and $\vec{\mathcal{H}}_2$ with dimension D. With large enough D, the hamming distance $\vec{\mathcal{H}}_1 \cdot \vec{\mathcal{H}}_2 \approx 0$. The orthogonality in HDC makes it possible to have multiple highly parallelizable operations such as similarity calculation, binding and bundling. **Similarity:** it calculates the distance between the query hypervector and the model hypervector. For real-valued hypervectors, $\delta(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2) = cos(\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2)$; for bipolar hypervectors, it simplifies to hamming distance. **Bundling:** element-wise addition that creates a set of hypervectors. If we add two hypervectors as $\vec{\mathcal{H}}_{bundle} = \vec{\mathcal{H}}_1 + \vec{\mathcal{H}}_2$, unlike vector addition in lower dimension, it is easy to verify whether a query hypervector is in the set. For example, $\delta(\vec{\mathcal{H}}_{bundle}, \vec{\mathcal{H}}_1) >> 0$ while $\delta(\vec{\mathcal{H}}_{bundle}, \vec{\mathcal{H}}_3) \approx 0$ if $\vec{\mathcal{H}}_3$ is another randomly sampled hypervector. **Binding:** element-wise multiplication that associates two hypervectors and creates another near-orthogonal hypervector. Two hypervectors bind as $\vec{\mathcal{H}}_{bind} = \vec{\mathcal{H}}_1 * \vec{\mathcal{H}}_2$. The information from both hypervectors is preserved because this operation is reversible $\vec{\mathcal{H}}_{bind} * \vec{\mathcal{H}}_1 = \vec{\mathcal{H}}_2$.

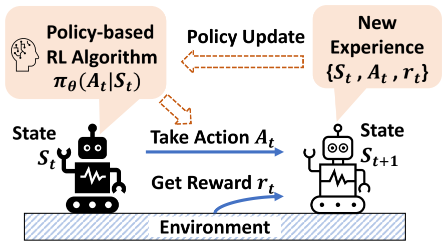

Figure 1: Reinforcement Learning overview.

2.2 Reinforcement Learning

Figure 1 illustrates a typical RL task, in which we define a self-learning agent that interacts with the environment following a certain learned policy π_θ, where θ is the policy parameter. The interaction is bidirectional, in which the agent acts based on the observed state S of the environment, and that action A influences the environment state and leads to a reward r. RL tasks usually assume that the Markov Property holds, i.e., the future is independent of the past given the present. Therefore, we can mathematically formulate the trajectories of the interaction between the agent and the environment. Consider a trajectory τ with T steps, we have $\tau = \{S_1, A_1, \ldots, S_T, A_T\}$ and the probability of having such trajectory is $p_\theta(\tau)$. We also define the decision-making process as the distribution of actions given the current state, i.e., $\pi_\theta(A_t|S_t)$. Then, the trajectory probability is expanded using the Markov Property as the following: $p_\theta(\tau) = p(S_1) \prod_{t=1}^{T} \pi_\theta(A_t|S_t) p(S_{t+1}|S_t, A_t)$, where $p(S_1)$ is the probability of starting at state S_1 and the last term stands for the transition probability from S_t to S_{t+1}. The experience of a trajectory is usually saved to the local memory as $E = \{S_t, A_t, r_t\}$, which is then used for learning.

The ultimate goal of every RL agent, irrespective of whether it is value-based or policy-based, is to maximize the total returned rewards in one trajectory. At each step t in the trajectory, the agent maximizes the future accumulated rewards $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ with a discount factor γ between 0 and 1. A factor close to zero means the agent gives more credit to returns in the near future, resulting in a short-sighted agent.

The value function is another important component in RL algorithms that is closely connected to rewards. In value-based RL methods, value functions are used to evaluate the current action choices and state observations on their ability to acquire rewards.

For example, the action-state value function, or known as the Q-value function, returns a higher value for the better action choice under the current state. This action-state value $Q^{\pi}(S, A)$, if accurately computed, is $\mathbb{E}[R_t|S_t = S, A_t = A, \theta]$. As for the state value function, it is calculated similarly as $V^{\pi}(S) = \mathbb{E}[R_t|S_t = S, \theta]$.

3 VANILLA POLICY GRADIENT METHOD

Policy gradient ascent is a natural solution for RL tasks, since the target is essentially a function optimization problem that maximizes the reward. The formal objective function is defined as the following: $J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t R_t] = \int p_\theta(\tau)R(\tau)d\tau$, where $R(\tau)$ is the accumulated rewards for the trajectory τ. The gradient of this objective is: $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau))R(\tau)] = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(A_t|S_t)R_t]$. A common transform is used here to equalize the integral gradient to another expectation, which is desirable for Monte-Carlo sampling. We directly sample the past trajectory from the starting point to its end. Within a sampled full trajectory, both discounted reward and action distribution at each step are known for calculating the gradient. Therefore, parameters of the policy π are updated using: $\theta = \theta + \alpha \nabla_\theta J(\theta)$. In DNN-based policy gradient, the output of neural networks is the distribution of action space $\pi_\theta(A_t)$ given the input state vector S_t. However, the deep network structure in DNN brings high computation costs for both forward inference and backward training; thereby, it is not ideal for RL implementation on low-power devices. In this section, we use HDC to reduce the computation cost and provide better parallelism.

HDC non-linear encoder: Considering that input state observations are real-valued vectors, we encode them using random projection with non-linearity. Assuming an n-variable state observation $S = [s_1, s_2, \ldots, s_n]$, we create the corresponding base hypervectors $\boldsymbol{\mathcal{H}} = \{\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2, \ldots, \vec{\mathcal{H}}_n\}$ for each state variable. Instead of using only bipolar hypervectors, we generate base hypervectors with elements randomly sampled from i.i.d. Gaussian distribution in order to capture subtle interconnections between input variables. Thus, for elements in $\vec{\mathcal{H}} \in \mathbb{R}^D$, $\{h_1, h_2, \ldots, h_D\} \sim \mathcal{N}(0, 1)$. We also generate a uniformly distributed bias hypervector $\vec{\mathcal{B}} \in \mathbb{R}^D$ with its elements $b \sim \mathcal{U}(0, 2\pi)$. In the end of encoding, we add a *cos* activation layer that provides extra non-linearity. The complete HDC encoding process for policy gradient is shown as the following: $\vec{\mathcal{S}} = \cos(S\boldsymbol{\mathcal{H}} + \vec{\mathcal{B}}) = \cos(S[\vec{\mathcal{H}}_1, \vec{\mathcal{H}}_2, \ldots, \vec{\mathcal{H}}_n] + \vec{\mathcal{B}})$.

HDC learning: To optimize the objective reward function, we apply gradient-based HDC learning. We first initialize m all-zero model hypervectors $\vec{C}$ for each of the variables in the continuous action vector $A = [a_1, a_2, \ldots, a_m]$. We view these hypervectors as a trainable parameter matrix $\boldsymbol{\mathcal{C}} = [\vec{C}_1, \vec{C}_2, \ldots, \vec{C}_m]$, and similarly for encoding, we have a fixed encoding matrix $\boldsymbol{\mathcal{H}}$. The first matrix maps inputs to hypervectors, and the second matrix gives the similarity between the encoded state observation and the model hypervectors. In this way, our HDC learning can be easily fitted into the common ML frameworks, such as PyTorch, for gradient calculations.

Our hyperdimensional policy gradient provides a continuous stochastic policy, in which we assume that every action variable follows a Gaussian distribution with mean μ and variance σ^2. For different observations, the agent selects actions from distributions centered at different mean values. The variance is annealing at a fixed rate during the learning process to control the exploration

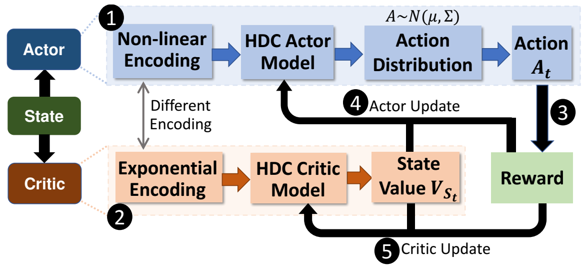

Figure 2: HDPG: hyperdimensional policy-based RL.

rate. The forward propagation is shown as the following:

$$A \sim \mathcal{N}(\vec{\mu}, \Sigma) \quad \text{where} \quad \vec{\mu} = \vec{\mathcal{S}}\boldsymbol{\mathcal{C}} = [\mu_1, \mu_2, \ldots, \mu_m] \tag{1}$$

The action is sampled from a multivariate Gaussian distribution with a co-variance matrix Σ that is non-zero only at diagonal terms. We derive the policy gradient loss function at each step t of a sampled trajectory:

$$\begin{aligned} \mathcal{L}_t &= -\log p(A_t)R_t \quad \text{where} \quad A_t \sim \mathcal{N}(\vec{\mu_t}, \Sigma_t) \\ \mathcal{L}_\tau &= \frac{1}{T}\sum_{t=1}^{T}(\mathcal{L}_t) \quad \text{and} \quad \boldsymbol{\mathcal{C}} = \boldsymbol{\mathcal{C}} + \alpha \nabla_{\boldsymbol{\mathcal{C}}} \mathcal{L}_\tau \text{ (Iterative Learning)} \end{aligned} \tag{2}$$

We back-propagate the gradient of the averaged trajectory loss $\mathcal{L}_\tau$ to update the trainable parameters in model hypervectors $\boldsymbol{\mathcal{C}}$. Notice that the encoding weight matrix and bias vector are not updated. For each sampled trajectory τ, the learning on each data point is iterated multiple times.

Comparing DNN-based policy gradient with our hyperdimensional method, although both utilize back-propagation for gradient update, our method reduces the computation cost. Usually, for complicated tasks, DNN not only has a larger number of Multiply-Accumulate (MAC) operations due to multiple hidden layers, it also suffers from the multi-level back-propagation. We only have one weight matrix to update during each step of training, while DNN has at least two. It is also well-known that back-propagation is hard to parallelize across layers. In addition, the HDC encoding process is highly-parallelizable and easy to accelerate using hardware platforms like FPGA.

4 HYPERDIMENSIONAL POLICY-BASED RL

In the previous section, we proposed a hyperdimensional policy gradient that directly optimizes the RL policy and objective with a well-defined loss. However, this vanilla policy-based RL has many drawbacks. The most crucial one is that the loss function is largely based on rewards from trajectory sampling. The sampled rewards have a significant variance due to the high randomness of the RL process. For example, the same action-state pair at the current step can lead to vastly different rewards at the end of trajectory, simply because of the uncertainty in the following steps. The second point to notice is that the model update corresponding to the gradient is not limited, potentially leading to model divergence since the gradient has a high variance. To reduce the variance, we include a value-based critic that functions as a baseline of rewards. Then we apply a clipping method for the policy gradient update [13].

As shown in Figure 2, we propose HDPG, a hyperdimensional policy-based reinforcement algorithm featuring an actor for decision-making and a value-based critic for variance reduction. Both components are constructed using efficient HDC. At each step, the observed state $\vec{\mathcal{S}}_t$ is encoded to two different hypervectors using the non-linear encoder ❶ and the exponential encoder ❷ respectively. The actor selects an action A_t and the critic generates a state value V_{S_t} ❸ using the HDC models. The feedback reward and state value are then used for actor training ❹ and critic training ❺.

4.1 Critic: State Value Approximation

The critic in our HDPG approximates the advantage of each action-state pair in the sampled trajectories. Assuming the Q-value function $Q(A_t, S_t)$ and the state value function $V(S_t)$, the advantage $\mathcal{A}$ is defined as the difference between them. Intuitively, we interpret the advantage as the extra rewards gained by taking action A_t compared to the averaged return under state S_t. The benefit of using advantage is that a high reward, possibly caused by variance, no longer has an excessive influence on the model update. Instead, only the difference between the state value and the reward is counted towards the loss function. Specifically, in our HDPG, we use an approximation for the advantage to avoid the calculation of $Q(A_t, S_t)$.

$$\mathcal{A}(A_t, S_t) = Q(A_t, S_t) - V(S_t) \approx R_t - V(S_t) \tag{3}$$

Therefore, the critic requires only the value function. In DNN-based RL, this function is parameterized with a trainable neural network similar to the policy. In our HDPG, we use HDC as a lightweight alternative for the value function approximation. We consider the value function approximation as a supervised regression problem within the policy-based RL. The target is to predict the state value as accurately as possible, in other words, as close as possible to the true value. Although RL is not typical supervised learning, the true value is available for $V(S_t)$ training. In the following, we explain how to map this task into high-dimensional space and enable efficient HDC learning.

HDC exponential encoder for regression: Compared to classification problems that are not sensitive to slight variations in the output class probabilities, regression is all about precision. The predicted output is strictly compared with the true value, and inputs close to each other should also return similar outputs. Unfortunately, the non-linear encoder designed for policy gradient, as shown in Section 3, is no longer ideal for regression, and we show that its mapping does not fully preserve the distance between inputs in the hyperdimensional space.

On the other hand, we design an HDC encoder that maps inputs to hypervectors with exponential components. Specifically in HDPG critic, an input state observation S will be encoded as the following: $\vec{\mathcal{S}} = e^{i(S\boldsymbol{\mathcal{H}}+\vec{\mathcal{B}})}$, where $\boldsymbol{\mathcal{H}}$ and $\vec{\mathcal{B}}$ are the same as those in the policy gradient HDC encoder. The major advantage of this encoder is that it creates a high-dimensional Gaussian space, in which the similarity between two encoded hypervectors closely correlates to the original low-dimensional distance. Assuming two-state observations S_1 and S_2, and $d(S_1, S_2) = ||S_1 - S_2||^2 \approx 0$. After encoding, the similarity of two hypervectors is:

$$\begin{aligned}\delta(\vec{\mathcal{S}}_1, \vec{\mathcal{S}}_2) &= real(\frac{\vec{\mathcal{S}}_1 \cdot conj(\vec{\mathcal{S}}_2)}{D}) \\ &= real(\frac{e^{i(S_1\boldsymbol{\mathcal{H}}+\vec{\mathcal{B}})} \cdot e^{-i(S_2\boldsymbol{\mathcal{H}}+\vec{\mathcal{B}})}}{D}) \approx 1\end{aligned} \tag{4}$$

Similarly, if two state observations are largely different, the similarity after encoding will be nearly zero.

HDC regression for state value approximation: HDC regression is the core of HDPG critic, in which we use a single model hypervector $\vec{\mathcal{M}}$ to provide the prediction of $\mathbb{E}[R_t|S_t, \theta]$. Assume the input is S_t, our regression model generates the prediction as

$$V_{pred}(S_t) = \delta(\vec{\mathcal{S}}_t, \vec{\mathcal{M}}) \tag{5}$$

The model training process relies on lightweight hypervector addition and subtraction. With the available true value $V_{true}(S_t) = R_t$ and learning rate β, we update the model with

$$\vec{\mathcal{M}}_{new} = \vec{\mathcal{M}}_{old} + \beta(V_{true}(S_t) - V_{pred}(S_t))\vec{\mathcal{S}}_t \tag{6}$$

4.2 Actor: Policy Gradient with Clipping

Our actor in HDPG is constructed based on the vanilla HDC policy gradient, which defines the policy of how agents act. While the state encoding and forward inference is exactly the same, the loss in Equation 2 is modified to tackle the high training variance. We present the new loss function at step t as:

$$\begin{aligned}&\mathcal{L}_t = -\min(\mathcal{L}_o, \mathcal{L}_{clip}) - c \cdot \text{Entropy} \\ &\text{where } \mathcal{L}_o = \text{Ratios} \times \text{Advantage} = \frac{\log p_{new}(A_t)}{\log p_{old}(A_t)}(R_t - V_{pred}(S_t)) \\ &\text{and } \mathcal{L}_{clip} = \text{Clip}(\text{Ratios}, 1+\epsilon, 1-\epsilon) \times \text{Advantage}\end{aligned} \tag{7}$$

The hyperdimensional critic plays an important role in the actor update. The difference between the actual return and the state value predicted by the critic approximates the advantage. This means that the model will only get positive updates when the advantage is positive, i.e., the action returns more rewards than the baseline. On the other hand, the model gets explicit negative updates if the action does not lead to expected rewards.

Instead of using the logarithmic probability directly in the loss, we use the clipped ratio between new and old probabilities to control the rate of change in iterative model updates. For example, if the advantage is positive, the model is reinforced towards the selected action. This reinforcement is limited due to the ratio capped at $1+\epsilon$. However, the ratio is not limited in the other direction because the model should not update much if the select action previously does not have a high probability. Similarly, when the advantage is negative, the model gets discouraged, and the discouragement is also limited with the ratio bottomed at $1-\epsilon$. The other direction is not capped because we want to correct the model when a previously high-probability action gets a negative advantage. This simple clipping function effectively and intelligently limits the model update in a reasonable range under all circumstances. The entropy term $\mathcal{L}_t$ encourages the agent to explore.

5 EXPERIMENTAL RESULT

5.1 Experiment Settings

In order to evaluate the performance of our HDPG, we implement it on both a traditional CPU platform and FPGA. For CPU, we use an AMD Ryzen R5-3600X, and the FPGA implementation is based on Xilinx Kintex-7 FPGA. We select two continuous control tasks from OpenAI RoboSchool [25], which provides many simulated robot control environments. The first task is 'HalfCheetah' in which a robot resembles half of a cheetah. The second task is 'Walker2D' where a two-legged robot mimics human walking. The goal of both tasks is to run as fast as possible. To compare our HDPG with DNN-based RL, we implement the state-of-the-art PPO algorithm. For both the actor and critic in the PPO method, we use DNN with

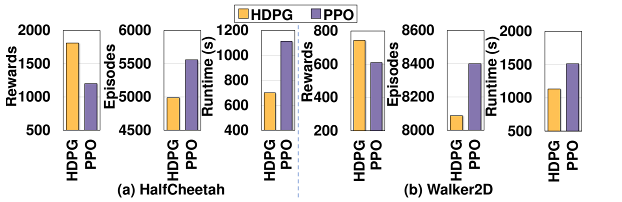

Figure 3: Rewards and runtime comparison of HDPG and DNN-based PPO: (a) the number of episodes and runtime is recorded when the 100-episode average reward reaches 1000; (b) they are recorded when the average reward reaches 600.

two 128-neuron hidden layers. For HDPG, the dimensionality of hypervectors is set to 1000 for the HalfCheetah task and 1500 for the Walker2D task. For both methods, the model update happens per 3000 steps, i.e., after the agent collects 3000 samples. The maximum number of steps within a single episode or trajectory is 1000. All the recorded rewards, shown in this section, are averaged over the past 100 episodes due to the fluctuation in both methods, and we will refer to them as average rewards.

5.2 Rewards and Runtime Comparison

In Figure 3, we present the performance comparison between two methods in terms of final average reward and episodes/runtime needed for achieving the goal of two robot control tasks. Figure 4 provides the learning curves of HDPG and PPO for both tasks. The results in this subsection are collected on CPU.

For the comparison of HalfCheetah results, Figure 3(a) shows that our HDPG achieves the 100-episode average reward of 1811, which is significantly higher than the PPO method. From the learning curves in Figure 4, it is clear that HDPG not only provides better learning quality in HalfCheetah task but also faster learning speed. When we set the average reward goal as 1000, HDPG reaches the goal about 1.6 times (more than 400 seconds) faster in terms of runtime and 500 episodes earlier in terms of the number of episodes.

In Walker2D task, the average reward goal is 600. Figure 3(b) shows that HDPG achieves an average reward of 743 at the end of 9000 episodes, which is about 130 more comparing to PPO. When comparing the efficiency, HDPG reaches the goal more than 300 episodes earlier than PPO. In terms of actual runtime, our HDPG is about 1.3 times (about 380 seconds) faster than PPO.

The learning results and CPU runtime comparison show that HDPG provides high-quality RL in continuous control tasks. We also notice that this HDC-based RL method has better learning efficiency even on a CPU platform.

5.3 HDPG Efficiency on Different Platforms

Figure 5 compares the training execution time and energy consumption of HDPG running on AMD Ryzen CPU and Xilinx Kintex-7 FPGA platform. The results are reported as for value-based and policy-based networks, separately. Both networks are designed to work in their best configurations. Our evaluation shows that HDPG is significantly more efficient than PPO for both actor and critic networks. For example, HDPG is on average 2.1× faster and provides 2.6× higher energy efficiency than PPO running on CPU. Note that HDPG efficiency mainly comes from the critic network. PPO utilizes a similar DNN network representing both actor and critic, while HDPG relies on a lightweight regression model for the critic. Our evaluation shows that actors and critics are, on average, 1.3× and 3.8× faster than PPO.

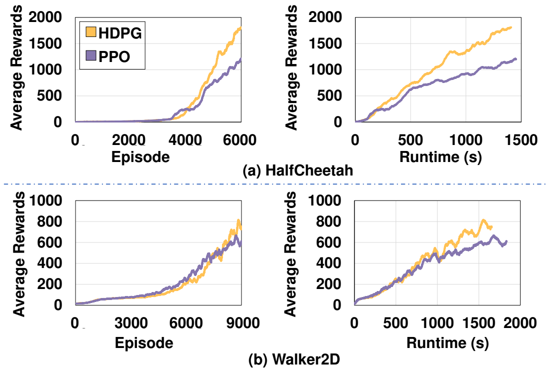

Figure 4: Larning curve comparison between HDPG and PPO for HalfCheetah and Walker2D.

We observe that CPU is not the best platform to accelerate HDPG. Unlike PPO, HDPG operates over high-dimensional vectors where CPU processors are not well-optimized to (1) access large-size hypervectors from memory and (2) provide enough parallelism required by our HDPG model. In contrast, FPGA implementation provides custom features to read and manage memory access patterns to ensure maximum parallelism and resource utilization. In addition, since HDPG is based on low-precision computation, it exploits FPGA Look-up tables (LUTs) that are significantly more efficient than Digital Signal Processors (DSPs). Our evaluation shows that HDPG achieves 4.7× faster and 5.3× higher energy efficiency than PPO on the FPGA platform.

5.4 HDPG & Dimensionality

In Figure 6, we change the dimensionality of HDPG with a range of $D = 250$ to $D = 4000$ and observe its effect on runtime and rewards using the HalfCheetah task. We collect these results at the end of 5000 episodes to show a clear trend. This bar graph shows that, with higher dimensionality, the achieved reward is also generally higher. For example, the average reward is about 1200 when the dimension is set to 4000, which is about 600 higher than the results achieved with only 250 dimensions. However, it should be noticed that higher dimensionality does not guarantee better rewards, and the improvement will finally saturate if dimensions keep increasing. In addition, larger dimensions bring higher computation cost and longer runtime. As shown in the figure, the runtime at 5000 episodes increases with dimensions. Our default dimension setting for HalfCheetah, which is 1000, tries to balance the learning quality and runtime.

6 RELATED WORK

Hyperdimensional Computing: For ML applications in low-power environments, HDC is a more efficient alternative computing paradigm compared to DNN. Its advantages include lightweight operations, less computation, more transparent models, great compatibility with multiple hardware acceleration technologies. Many prior works propose to use HDC for low-power classification algorithms, e.g., voice recognition [26] and neuromorphic computing [27], online classification [28, 29]. In the biomedical field, work in [18] proposes an HDC-based seizure detection algorithm and work in [30–32] utilize HDC for DNA pattern matching. For robotics, in work [33], researchers use hypervectors for the integration of sensory perception and action, i.e., sensorimotor control. In contrast to the supervised learning applications mentioned above, for the

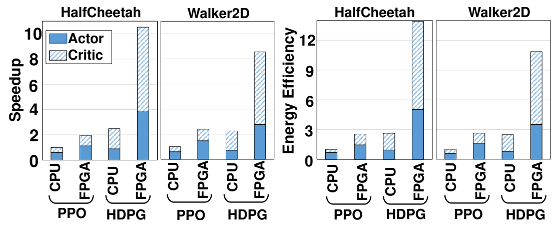

Figure 5: HDPG and PPO performance and energy efficiency on CPU and FPGA platforms (normalized to PPO).

first time, we exploit efficient HDC in policy-based RL algorithms for continuous control.

Deep Reinforcement Learning: The abilities to handle large inputs and capture complicated patterns make Deep Reinforcement Learning (DRL) a good candidate for self-learning agents in more and more demanding tasks. We mainly focus on the policy-based DRL since it supports not only discrete action spaces but also continuous ones. In recent years, many policy-based RL algorithms have been proposed. For example, TRPO [12] introduces a constraint for updating the network, which improves the stability of the policy gradient update. The PPO [13] method provides a simpler constraint that is built into the loss function. There are many recent RL applications that are implemented using these methods, including Unmanned Aerial Vehicles (UAV) [34], active object detection [35], humanoid robot [36], traffic control [37]. Work in [37] uses a model-accelerated PPO to control automated vehicles in intersections without traffic signals. However, in these applications, the computation cost of learning a deep network is sub-optimal for low-power devices. In our HDPG, we utilize the novel HDC method for both actor and critic in the policy-based RL to provide more efficiency, such that the learning runtime is significantly reduced while maintaining the learning quality.

7 CONCLUSION

In this paper, we introduce HDPG, a highly-efficient policy-based RL algorithm using Hyperdimensional Computing. Hyperdimensional computing is a lightweight brain-inspired learning methodology; its holistic representation of information leads to a well-defined set of hardware-friendly high-dimensional operations. Our HDPG fully exploits the efficient HDC for high-quality state value approximation and policy gradient update. In our experiments, we use HDPG for robotics tasks with continuous action space and achieve significantly higher rewards than DNN-based RL.

ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation #2127780, Semiconductor Research Corporation (SRC) Task #2988.001, Department of the Navy, Office of Naval Research, grant #N00014-21-1-2225 and #N00014-22-1-2067, Air Force Office of Scientific Research, and a generous gift from Cisco. Mohsen Imani and Xunzhao Yin are co-corresponding authors of this paper.

REFERENCES

[1] R. R. Murphy, *Introduction to AI robotics.* MIT press, 2019.
[2] D. Silver *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[3] V. Mnih *et al.*, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
[4] C. J. Watkins *et al.*, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[5] K. S. Narendra *et al.*, “Learning automata: an introduction,” 1989.

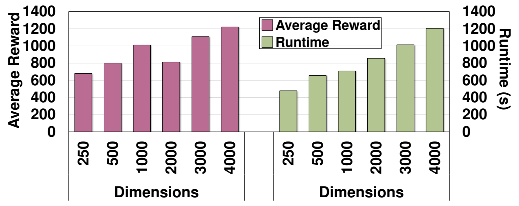

Figure 6: Explore the effect of HDPG dimension size on runtime and average reward.

[6] R. J. Williams *et al.*, “Function optimization using connectionist reinforcement learning algorithms,” *Connection Science*, vol. 3, no. 3, pp. 241–268, 1991.
[7] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.
[8] V. Gullapalli *et al.*, “Acquiring robot skills via reinforcement learning,” *IEEE Control Systems Magazine*, vol. 14, no. 1, pp. 13–24, 1994.
[9] W. T. Uther *et al.*, “Tree based discretization for continuous state space reinforcement learning,” *Aaai/iaai*, vol. 98, pp. 769–774, 1998.
[10] W. D. Smart *et al.*, “Practical reinforcement learning in continuous spaces,” in *ICML*, pp. 903–910, Citeseer, 2000.
[11] V. R. Konda *et al.*, “Actor-critic algorithms,” in *NIPS*, pp. 1008–1014, 2000.
[12] J. Schulman *et al.*, “Trust region policy optimization,” in *ICML*, pp. 1889–1897, PMLR, 2015.
[13] J. Schulman *et al.*, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
[14] H. Mao *et al.*, “Resource management with deep reinforcement learning,” in *HotNets*, pp. 50–56, 2016.
[15] S. Zhang *et al.*, “Trajectory-tracking control of robotic system via proximal policy optimization,” in *CIS*, pp. 380–385, IEEE, 2019.
[16] P. Kanerva, “Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors,” *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.
[17] Y. Kim *et al.*, “Efficient human activity recognition using hyperdimensional computing,” in *IOT*, pp. 1–6, 2018.
[18] A. Burrello *et al.*, “Laelaps: An energy-efficient seizure detection algorithm from long-term human ieeg recordings without false alarms,” in *DATE*, pp. 752–757, IEEE, 2019.
[19] R. Chen *et al.*, “Joint active search and neuromorphic computing for efficient data exploitation and monitoring in additive manufacturing,” *Journal of Manufacturing Processes*, vol. 71, pp. 743–752, 2021.
[20] P. Poduval *et al.*, “Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning,” *Frontiers in Neuroscience*, 2022.
[21] B. Khaleghi *et al.*, “tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications,” in *DATE*, pp. 408–413, IEEE, 2021.
[22] H. Li *et al.*, “Device-architecture co-design for hyperdimensional computing with 3d vertical resistive switching random access memory (3d vrram),” in *VLSI-TSA*, pp. 1–2, IEEE, 2017.
[23] A. Hérnandez-Cano *et al.*, “Prid: Model inversion privacy attacks in hyperdimensional learning systems,” in *DAC*, pp. 553–558, IEEE, 2021.
[24] P. Poduval *et al.*, “Stochd: Stochastic hyperdimensional system for efficient and robust learning from raw data,” in *DAC*, pp. 1195–1200, IEEE, 2021.
[25] G. Brockman *et al.*, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
[26] M. Imani *et al.*, “Voicehd: Hyperdimensional computing for efficient speech recognition,” in *ICRC*, pp. 1–8, IEEE, 2017.
[27] Z. Zou *et al.*, “Spiking hyperdimensional network: Neuromorphic models integrated with memory-inspired framework,” *arXiv preprint arXiv:2110.00214*, 2021.
[28] Z. Zou *et al.*, “Scalable edge-based hyperdimensional learning system with brain-like neural adaptation,” in *SC*, pp. 1–15, 2021.
[29] A. Hernandez-Cane *et al.*, “Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system,” in *DATE*, pp. 56–61, IEEE, 2021.
[30] Y. Kim *et al.*, “Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing,” in *DATE*, pp. 115–120, IEEE, 2020.
[31] P. Poduval *et al.*, “Cognitive correlative encoding for genome sequence matching in hyperdimensional system,” in *DAC*, pp. 781–786, IEEE, 2021.
[32] Z. Zou *et al.*, “Biohd: An efficient genome sequence search platform using hyperdimensional memorization,” in *ISCA*, IEEE, 2022.
[33] A. Mitrokhin *et al.*, “Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception,” *Science Robotics*, vol. 4, no. 30, 2019.
[34] E. Bøhn *et al.*, “Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization,” in *ICUAS*, pp. 523–533, IEEE, 2019.
[35] H. Liu *et al.*, “Extreme trust region policy optimization for active object recognition,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2253–2258, 2018.
[36] L. C. Melo *et al.*, “Learning humanoid robot running skills through proximal policy optimization,” in *LARS*, pp. 37–42, IEEE, 2019.
[37] Y. Guan *et al.*, “Centralized cooperation for connected and automated vehicles at intersections by proximal policy optimization,” *IEEE Trans. Veh. Technol.*, 2020.