A Framework for Training Larger Networks for Deep Reinforcement Learning

Ota, Kei; Jha, Devesh K.; Kanezaki, Asako

TR2024-058 May 18, 2024

Abstract

The success of deep learning in computer vision and natural language processing communities can be attributed to the training of very deep neural networks with millions or billions of parameters, which can then be trained with massive amounts of data. However, a similar trend has largely eluded the training of deep reinforcement learning (RL) algorithms where larger networks do not lead to performance improvement. Previous work has shown that this is mostly due to instability during the training of deep RL agents when using larger networks. In this paper, we make an attempt to understand and address the training of larger networks for deep RL. We first show that naively increasing network capacity does not improve performance. Then, we propose a novel method that consists of 1) wider networks with DenseNet connection, 2) decoupling representation learning from the training of RL, and 3) a distributed training method to mitigate overfitting problems. Using this three-fold technique, we show that we can train very large networks that result in significant performance gains. We present several ablation studies to demonstrate the efficacy of the pro- posed method and some intuitive understanding of the reasons for performance gain. We show that our proposed method outperforms other baseline algorithms on several challenging locomotion tasks.

Machine Learning Journal 2024

A Framework for Training Larger Networks for Deep Reinforcement Learning

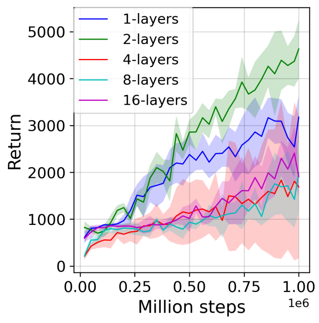
Kei Ota[1,2*], Devesh K. Jha[3] and Asako Kanezaki[1]

[1*]School of Computing, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, 152-8550, Tokyo, Japan.
[2]Information Technology R&D Center, Mitsubishi Electric Corporation, 5-1-1 Oofuna, Kamakura, 247-0056, Kanagawa, Japan.
[3]Mitsubishi Electric Research Labs, 201 Broadway, Cambridge, 02139, Massachusetts, USA.

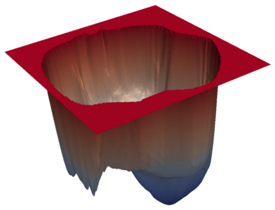
*Corresponding author(s). E-mail(s): ohta.k.ag@m.titech.ac.jp; Contributing authors: jha@merl.com; kanezaki@c.titech.ac.jp;

Abstract

The success of deep learning in computer vision and natural language processing communities can be attributed to the training of very deep neural networks with millions or billions of parameters, which can then be trained with massive amounts of data. However, a similar trend has largely eluded the training of deep reinforcement learning (RL) algorithms where larger networks do not lead to performance improvement. Previous work has shown that this is mostly due to instability during the training of deep RL agents when using larger networks. In this paper, we make an attempt to understand and address the training of larger networks for deep RL. We first show that naively increasing network capacity does not improve performance. Then, we propose a novel method that consists of 1) wider networks with DenseNet connection, 2) decoupling representation learning from the training of RL, and 3) a distributed training method to mitigate overfitting problems. Using this three-fold technique, we show that we can train very large networks that result in significant performance gains. We present several ablation studies to demonstrate the efficacy of the proposed method and some intuitive understanding of the reasons for performance gain. We show that our proposed method outperforms other baseline algorithms on several challenging locomotion tasks.



(a) Average return.



(b) Loss surface.

Fig. 1: Training curves of SAC agents with the different numbers of layers while fixing the unit size (256) on Ant-v2 environment, and the loss function surface [15] of the deepest (16-layers) Q-network. The training curves suggest that simply building a deeper MLP with a fixed number of units does not improve the performance of DRL while building a larger network is generally effective in supervised learning. The loss surface shows that deeper networks have a more complex loss surface that could be susceptible to the choice of hyperparameters [15]. Motivated by this, we conduct an extensive study on how to train larger networks that contribute to performance gain for RL agents.

Keywords: Deep Reinforcement Learning, Representation Learning, Robotics

1 Introduction

We have witnessed considerable improvements in the fields of computer vision (CV) [1–4], natural language processing (NLP) [5–8], and robotics [9, 10] in the last decade. These developments could be largely attributed to the training of very large neural networks with millions (or even billions or trillions) of parameters that can be trained using huge amounts of data and an appropriate optimization technique to stabilize training [11]. In general, the motivation for training larger networks comes from the intuition that larger networks allow better solutions as they increase the search space of possible solutions. Having said that, neural network training largely relies on finding good minimizers of highly non-convex loss functions. These loss functions are also governed by the choices of network architecture, batch size, etc. This has also driven a lot of research in these communities towards understanding the underlying reasoning for performance gains [12–15]

In striking contrast, the Deep Reinforcement Learning (DRL) community has not reported a similar trend with regard to training larger networks for RL. Some studies have reported that deep RL agents experience instability while training with larger networks [16–19]. As an example, in Fig. 1, we show the results of the Soft Actor Critic (SAC) [20] agent that uses Multi-layered Perceptron (MLP) for function approximation with the increasing number of layers while fixing its unit size to 256 (also notice the loss surface). These plots show that the use of deeper networks naively leads to poor performance for a deep RL agent. Consequently, using larger networks to train deep RL networks is not fully understood and thus is limiting in several ways. As a result, most of the reported work in the literature ends up using similar hyperparameters such as network structure, number, and size of layers.

Our work is motivated by this limitation. We explore the interplay between the size, structure, training, and performance of deep RL agents to provide some intuition and guidelines for using larger networks.

We present a large-scale study and provide empirical evidence for the use of larger networks to train DRL agents. First, we highlight the challenges that one might encounter when using larger networks to train deep RL agents. To circumvent these problems, we integrate a three-fold approach: decoupling feature representation from RL to efficiently produce high-dimensional features, employing DenseNet architecture to propagate richer information, and using distributed training methods to collect more on-policy transitions to reduce overfitting. Our method is a novel architecture that combines these three elements, and we demonstrate that our proposed method significantly improves the performance of RL agents in continuous control tasks. We also conduct an ablation study to show which component contributes to the performance gain. In this paper, we consider learning from state vectors, i.e., not from high-dimensional observations, such as images.

Our contributions can be summarized as follows:

- We conduct a large-scale study on employing larger networks for DRL agents and empirically show that, contrary to deeper networks, wider networks can improve performance.
- We propose a novel framework that synergistically combines recently proposed techniques to stabilize training: decoupling representation learning from RL, DenseNet architecture, and distributed training. Although each of these components has previously been proposed, the combination is novel and we demonstrate that it significantly improves performance.
- We analyze the performance gain of our method using metrics of effective ranks of features and visualization of the loss function landscape of RL agents.

2 Related Work

Our work is broadly motivated by Henderson et al. [16] which empirically demonstrates that DRL algorithms are vulnerable to different training choices like architectures, hyperparameters, activation functions, etc. The paper compares performance on the different numbers of units and layers and demonstrates that larger networks do not consistently improve performance.

This is contrary to our intuition considering recent progress in solving computer vision tasks such as ImageNet [21]: larger and more complex network architectures have proven to achieve better performance [1–3, 22].

Hasselt *et al.* [17] identify a *deadly triad* of *function approximation*, *bootstrapping*, and *off-policy learning*. When these three properties are combined, learning can be unstable and potentially diverge, with value estimates becoming unbounded. Several previous works have attempted to address this issue by implementing various techniques, such as target networks [23], double Q-learning [24], and n-step learning [25]. Our challenge of training larger networks is specifically related to function approximation. However, as the deadly triad is entangled in a complex manner, we also have to deal with other problems. Regarding network size, some studies investigate the effect of making the network larger for continuous control tasks using MLP [18, 26] and concluded that larger networks tend to perform better, but also become unstable and prone to diverge more. In a related investigation, Hasselt *et al.* [17] employed CNNs to approximate functions for Atari games, which also revealed the instability that arises when expanding networks based on DQNs [23]. The study also found that training stability can sometimes be achieved through the use of Double Q-Networks, although it was not consistent with the size of the networks. Similar studies on on-policy methods are performed in Andrychowi *et al.* [27] and Liu *et al.* [28], showing that too small or large networks could cause a significant drop in policy performance. Although these studies are limited to relatively small sizes (hundreds of units with several layers), we will have a more thorough study on much larger networks, specifically focusing on learning from state vectors.

To build a large network, unsupervised learning has been used to learn powerful representations for downstream tasks in natural language processing [5, 29] and computer vision [30, 31]. In the context of RL, auxiliary tasks, such as predicting the next state conditioned on the past state(s) and action(s) have been widely studied to improve the sample efficiency of RL algorithms [32–35]. Researchers have generally focused on learning a good representation of the state input setting that produces low-dimensional features [36, 37]. In contrast to this, Ota *et al.* [38] propose the use of an online feature extractor network (OFENet) that intentionally increases input dimensionality and demonstrates that a larger feature size enables improving RL performance in both sample efficiency and control performance. We leverage this idea and use larger input (or feature) for RL agents, as well as larger networks for the policy and value function networks.

One can also use the AutoRL approaches [39] that dynamically adjust hyperparameters during training to build large networks. For example, Wan *et al.* [40] employ a combination of the population-based method [41] and the Bayesian optimization framework to optimize hyperparameters, including network architecture, to maximize returns during training. Mohan *et al.* [42] empirically demonstrate that hyperparameter landscapes vary over time during training, which requires AutoRL algorithms to adjust hyperparameters dynamically. Although this work focuses on proposing a framework to train large networks while improving the performance of RL algorithms, the proposed framework can also be easily combined with these AutoRL approaches.

3 Method

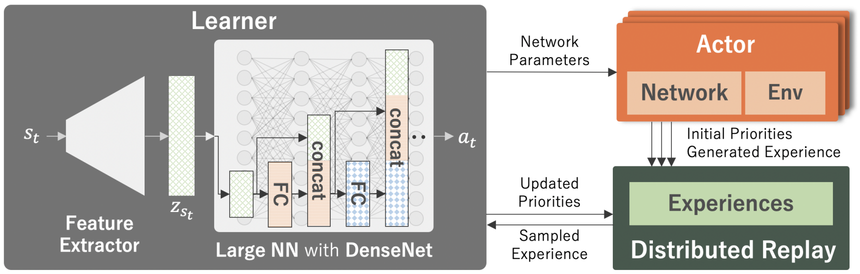


Fig. 2: Proposed framework to train larger networks for deep RL agents. We combine three elements. First, we decouple the representation learning from RL to extract an informative feature z_{s_t} from the current state s_t using a feature extractor network that is trained using an auxiliary task to predict the next state s_{t+1}. Second, we use large networks using DenseNet architecture, which allows for stronger feature propagation. Finally, we employ the Ape-X-like distributed training framework to mitigate the overfitting problems that tend to happen in larger networks and enable to collect more on-policy data that can improve performance. FC refers to a fully-connected layer.

While recent studies suggest that larger networks for DRL agents have the potential to improve performance, it is non-trivial to alleviate some potential issues that lead to instability when using larger networks to train RL agents.

Our method is based on three key ideas: (1) decoupling representation learning from RL, (2) allowing better feature propagation using good network architectures, and (3) using huge amounts of more on-policy data using distributed training to avoid overfitting in larger networks. We first obtain good features apart from RL using an auxiliary task and then propagate the features more efficiently by employing the DenseNet [3] architecture. Additionally, we use a distributed RL framework that can mitigate the potential overfitting problem. In the following, we describe in detail the three elements that we use to train larger networks for deep RL agents. Our proposed approach is shown schematically in Fig. 2.

3.1 Decoupling Representation Learning from RL

While the simplicity of learning the entire pipeline in an end-to-end fashion is appealing, updating all parameters of a large network using only a scalar reward signal can result in very inefficient training [43]. Decoupling unsupervised pretraining from downstream tasks is common in computer vision [30, 44] and has proven to be very efficient. Taking inspiration from this, we adopt the online feature extractor network (OFENet) [38] to learn meaningful features separately from RL training.

OFENet learns the representation vectors of states z_{s_t} and state-action pairs z_{s_t,a_t}, and provides them to the agent instead of the original inputs s_t and a_t, delivering significant performance improvements in continuous robot control tasks. As the representation vectors z_{s_t} and z_{s_t,a_t} are designed to have much higher dimensionality than the original input, OFENet matches our philosophy of providing a larger solution space that allows us to find a better policy. The representations can be obtained by learning the mappings $z_{s_t} = \phi_s(s_t)$ and $z_{s_t,a_t} = \phi_{s,a}(s_t, a_t)$. The ϕ_s and $\phi_{s,a}$ are neural networks with arbitrary architecture that have parameters $\theta_{\phi_s}, \theta_{\phi_{s,a}}$, and trained by minimizing an auxiliary task of predicting the next state s_{t+1} from the current state and action representation z_{s_t,a_t} as:

$$L_{\text{aux}} = \mathbb{E}_{(s_t,a_t)\sim p,\pi} \left[\| f_{\text{pred}}(z_{s_t,a_t}) - s_{t+1} \|^2 \right] , \tag{1}$$

where f_{pred} is represented as a linear combination of the representation z_{s_t,a_t}. The learning of the auxiliary task is done concurrently with the learning of the downstream RL task. Our experiments allow the input dimensionality to be much larger than previously presented in Ota *et al.* [38]. Furthermore, we also increase the network size of the RL agents. For more details, interested readers are referred to Ota *et al.* [38].

3.2 Distributed Training

In general, larger networks need more data to improve the accuracy of the function approximation [21, 45]. MLP with a large number of hidden layers is particularly known to cause an overfitting of training data, often resulting in inferior performance to shallow networks [46]. In the context of RL, while we train and evaluate in the same environment, there is still a problem of overfitting: the agent is only trained on limited trajectories it has experienced, which cannot cover the entire state-action space of the environment [28]. Fu *et al.* [26] showed that the overfitting to the experience replay does exist. To mitigate this overfitting problem, Fedus *et al.* [47] empirically showed that having more policy data in the replay buffer, i.e., collecting more than one transition while updating the policy one time, can improve the performance of the RL agent. However, it will be extremely slow.

In light of these studies, we employ the distributed RL framework, which leverages distributed training architectures that decouple learning from collecting transitions by utilizing many actors running in parallel on different environment instances [48, 49]. In particular, we use the Ape-X [48] framework, where a single learner receives experiences from distributed prioritized replay [50], and multiple actors collect transitions in parallel (see Fig. 2). This helps to increase the number of data close to the current policy, that is, more data on the policy, which can improve the performance of off-policy RL agents [47] and mitigate rank collapse problems in Q networks [51]. It is noted that one can collect more on-policy data by collecting more than one transition at each policy update iteration while being much slower, as shown in Fedus *et al.* [47]. Furthermore, while we employ a distributed training framework proposed in Horgan *et al.* [48], we do not use the RL algorithm used there, but instead use standard off-policy RL algorithms: SAC [20] and the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [52] in our experiments. Furthermore, it should be noted that our approach uses a distributed training framework as suggested by Horgan *et al.* [48]. However, the RL algorithms we evaluate differ from theirs; we employ two widely used off-policy RL algorithms, namely Soft Actor-Critic (SAC) [20] and Twin Delayed Deep Deterministic policy gradient algorithm (TD3) [52] in our experimental design.

3.3 Network Architectures

Tremendous developments have been made in the computer vision community in designing sophisticated architectures that enable training of very large networks by making the gradients more well-behaved, such as skip connections and batch normalization [2, 3, 22, 53]. We focus specifically on the use of the Dense Convolutional Network (DenseNet) architecture, which alleviates the problem of vanishing gradient, strengthens feature propagation, and reduces the number of parameters [3]. DenseNet has a skip connection that directly connects each layer to all subsequent layers as $y_i = f_i^{\mathrm{dense}}([y_0, y_1, ..., y_{i-1}])$, where y_i is the output of the i^{th} layer; thus all the inputs are concatenated into a single tensor. Here, f_i^{dense} is a composite function that consists of a sequence of convolutions, Batch Normalization (BN) [53], and an activation function. An advantage of DenseNet is its improved flow of information and gradients throughout the network, making large networks easier to train. We borrow this architecture to train large networks for RL agents.

Although there are existing examples of applying the DenseNet architecture to Deep Reinforcement Learning (DRL) agents, the full potential of this approach has yet to be fully investigated. In this regard, Sinha *et al.* [19] proposed a novel modification to the DenseNet architecture, wherein the state or the state-action pair is concatenated to each hidden layer of the multi-layer perceptron networks (MLP), except the final linear layer, in their D2RL algorithm. In contrast to the modified version proposed by Sinha *et al.* [19], Ota *et al.* [38] strictly adhered to the original DenseNet architecture in their work, utilizing the dense connection that concatenates all previous layer outputs for

OFENet training. Similarly, our approach uses the original DenseNet architecture to represent the policy and value function networks. The schematic of the DenseNet architecture is also shown in Fig. 2.

4 Experimental Settings

In this section, we summarize the settings we use for our experiments. We run each experiment independently with five random seeds. Average and ± 1 standard deviation results will be reported, which are solid lines and shaded regions when we show training curves. The horizontal axis of a training curve is the number of gradient steps, which is not identical to the number of steps an agent interacts with an environment only when we use the distributed replay.

4.1 Metrics

We evaluate the experimental results on two metrics: average return and recently proposed *effective ranks* [51] of the features matrices of Q-networks. Kumar *et al.* [51] showed that MLPs used to approximate policy and value functions that use bootstrapping lead to a reduction in the effective rank of the feature, and this rank collapse for the feature matrix results in poorer performance. The effective rank can be computed as $\text{srank}_\delta(\Phi) = \min\left\{k : \frac{\sum_{i=1}^{k}\sigma_i(\Phi)}{\sum_{i=1}^{d}\sigma_i(\Phi)} \geq 1-\delta\right\}$, where $\sigma_i(\Phi)$ are the singular values of the feature matrix Φ, which are the features of the penultimate layer of the Q-networks. We used $\text{srank}_\delta(\Phi) = 0.01$ to calculate the number of effective ranks in the experiments, as in Kumar *et al.* [51].

4.2 Implementations

RL agents

The hyperparameters of the RL algorithms are also the same as those of their original papers, except that the TD3 uses the batch size 256 instead of 100 as done in Ota *et al.* [38]. Also, for a fair comparison to Ota *et al.* [38], we used a random policy to store transitions to replay buffer before training RL agents for 10K time steps for SAC and for 100K steps for TD3. The other hyperparameters, network architectures, and optimizers are the same as those used in their original papers [20, 38, 52]. Our implementation of RL agents is based on the public codebase used by Ota *et al.* [38]. We conduct experiments on five different random seeds and report the average and standard deviation scores.

OFENet

Regarding the parameters of OFENet, we also follow the implementation of [38], that is, all OFENet networks that we used for our experiments consist of 8-layers DenseNet architectures with Swish activation [54]. To implement OFENet, we refer to the official codebase provided by Ota *et al.* [38]. We also

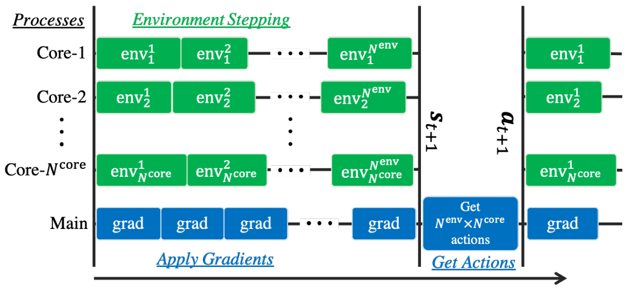

Processes
Environment Stepping
Core-1
Core-2
Core-N^{core}
Main
env_1^1 env_1^2 $\text{env}_1^{N^{\text{env}}}$
env_2^1 env_2^2 $\text{env}_2^{N^{\text{env}}}$
$\text{env}_{N^{\text{core}}}^1$ $\text{env}_{N^{\text{core}}}^2$ $\text{env}_{N^{\text{core}}}^{N^{\text{env}}}$
grad
Get $N^{\text{env}} \times N^{\text{core}}$ actions
s_{t+1}
a_{t+1}
Apply Gradients
Get Actions

Fig. 3: Schematic of asynchronous training. We use $N^{\text{core}} = 2$ cores for collecting experiences, where each core has $N^{\text{env}} = 32$ environments. Since the network parameters are shared, and the training and collecting transitions are decoupled, the collected experiences result in more on-policy data compared to the standard off-policy training, where the agent collects one transition while it applies one gradient step.

used target networks [23] to stabilize OFENet training, since the distribution of experiences stored in the shared replay buffer can change more dynamically utilizing the distributed training setting as described in Sec. 3.2. Target networks are updated at each training step by slowly tracking the learned networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, where we assume that θ are the network parameters of the current OFENet, and θ' are the target network parameters. We use the target smoothing coefficient $\tau = 0.005$, which is the same as the one used to update the target value networks in SAC [20], in other words, we do not tune this parameter.

Distributed training

The distributed training setting we used is similar to Stooke *et al.* [55], which collects experiences using N^{core} cores on which each core contains N^{env} environments. Specifically, we used $N^{\text{core}} = 2$ and $N^{\text{env}} = 32$. Figure 3 shows the schematic of the distributed training. Since the actions are computed by the latest parameters, the collected experiences result in more on-policy data.

4.3 Visualizing loss surface of Q-function networks

Li *et al.* [15] proposed a method to visualize the loss function curvature by introducing *filter normalization* method. The authors empirically demonstrated that the non-convexity of the loss functions could be problematic, and the sharpness of the loss surface correlates well with test error and generalization error. In light of this, we also visualize the loss surface of the networks to figure out why the deeper network could not lead to better performance, while the wider networks result in high-performance policies (Fig. 5).

To visualize the loss surface of our Q-networks, we use the authors' implementation[1] with the loss of:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta\left(\mathbf{s}_t,\mathbf{a}_t\right) - \hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right)^2\right], \tag{2}$$

with

$$\hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right) = r\left(\mathbf{s}_t,\mathbf{a}_t\right) + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\psi}}\left(\mathbf{s}_{t+1}\right)\right], \tag{3}$$

in which we exactly follow the notations used by SAC paper [20]. To compute this objective $J_Q(\theta)$, we collect all the transitions used in the training of deeper and wider networks and compute the target values of $\hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right)$ after the training has been completed and store the tuples of $\left(s_t, a_t, \hat{Q}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right)$ for all transitions in the training. Then, we use the authors' implementation to visualize the loss with the stored transitions and trained weights of the Q-network. Please refer to Li *et al.* [15] for more details.

5 Experimental Results

In this section, we present the results of numerical experiments in order to answer some relevant underlying questions posed in this paper. In particular, we answer the following questions.

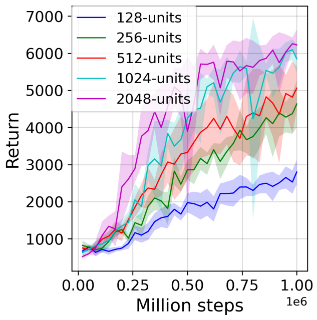
- Can RL agents benefit from the usage of larger networks during training? More concretely, can using larger networks lead to better policies for DRL agents?
- What characterizes a *good* architecture that facilitates better performance when using larger networks?
- Can our method work across different RL algorithms as well as different tasks, including sparse reward settings?
- How does the proposed framework perform in terms of wall clock time and sample efficiency?

5.1 Impact of network size on performance

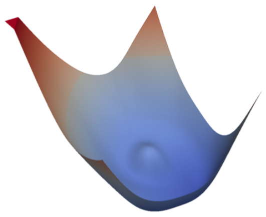
In the first set of experiments, we try to investigate whether increasing the size of the network always leads to poor performance. We quantitatively measure the effectiveness of increasing the network size by changing the number of units N^{unit} and layers N^{layer}, while the other parameters are fixed.

Figure 1a shows the training curves when increasing the number of layers while the unit size is fixed to $N^{\text{unit}} = 256$. As we described in Sec. 1, we observe that the performance becomes worse as the network becomes deeper. In Fig. 4a, we show the effect of increasing the number of units while the number of layers is fixed to $N^{\text{layer}} = 2$. Contrary to the results when making the network deeper, we can observe a consistent improvement when making the network wider. To

[1]Code used for these plots can be found at https://github.com/tomgoldstein/loss-landscape

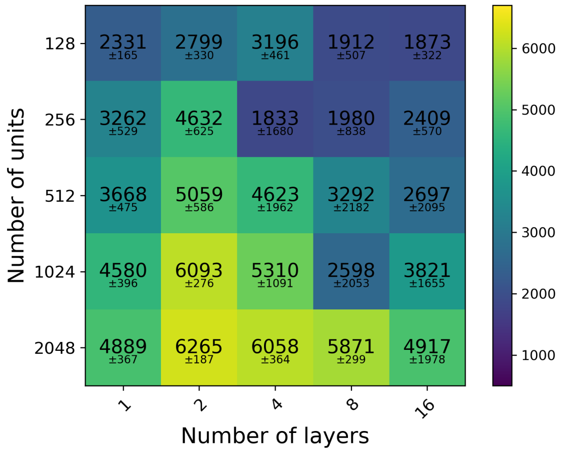


(a) Average return.



(b) Loss surface.

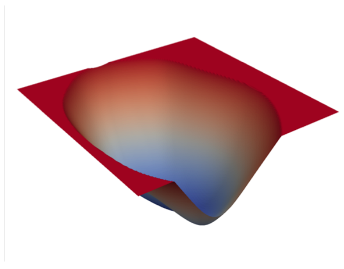
Fig. 4: Training curves of SAC agent with different number of units on Ant-v2 environment and the loss function surface of the widest (2048-units) Q-network. This shows that performance improves consistently when using wider MLPs.



Number of units \ Number of layers	1	2	4	8	16
128	2331 ±165	2799 ±330	3196 ±461	1912 ±507	1873 ±322
256	3262 ±529	4632 ±625	1833 ±1680	1980 ±838	2409 ±570
512	3668 ±475	5059 ±586	4623 ±1962	3292 ±2182	2697 ±2095
1024	4580 ±396	6093 ±276	5310 ±1091	2598 ±2053	3821 ±1655
2048	4889 ±367	6265 ±187	6058 ±364	5871 ±299	4917 ±1978

Fig. 5: Grid search results of maximum average return at one-million training steps over the different number of units and layers for SAC agent on Ant-v2 environment. This demonstrates that a deeper MLP (see horizontally) does not consistently improve performance, while a wider MLP (see vertically) generally does.

investigate more thoroughly, we also conduct a grid search, where we sample each parameter of the network from $N^{\text{unit}} \in \{128, 256, 512, 1024, 2048\}$, and $N^{\text{layer}} \in \{1, 2, 4, 8, 16\}$ and evaluate the performance in Fig. 5. We can see a monotonic improvement in performance when widening networks at almost all depths of the network.


(a) Wider network ($N^{\text{layer}} = 2, N^{\text{unit}} = 2048$)

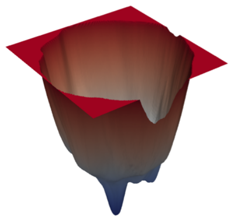

(b) Deeper network ($N^{\text{layer}} = 16, N^{\text{unit}} = 256$)

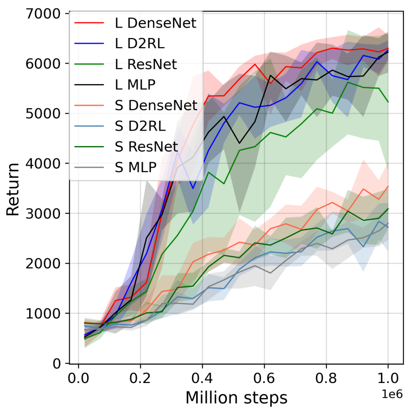
Fig. 6: Loss landscapes of models trained on HalfCheetah-v2 with one million steps, visualized using the technique in [15].

This result is in line with the general belief that training deeper networks is, in general, more difficult and more susceptible to the choice of hyperparameters [46]. This could be attributed to the initialization of the networks – it is known that the wider a layer, the closer the network is to the ideal conditions under which the initialization scheme was derived [56]. To understand why the deeper network is harder than the wider networks, we investigate the loss surface curvatures [15] of both deeper and wider networks. We show the loss surface of the deeper network ($N^{\text{layer}} = 16, N^{\text{unit}} = 256$) in Fig. 1b and the wider network ($N^{\text{layer}} = 2, N^{\text{unit}} = 2048$) of the SAC agent trained in the Ant environment in Fig. 4b as well as HalfCheetah-v2 environment in Fig. 6 using the visualization method proposed in Li *et al.* [15] with the loss of TD error of Q-functions of SAC agents. These figures show that wider networks have a nearly convex surface while deeper networks have more complex loss surface, which could be susceptible to the choice of hyperparameters [15]. Comparison of deeper and wider networks has also been done in several works [2, 14, 15, 57, 58], where wider networks are prone to have more generalization capability due to their smooth loss functions.

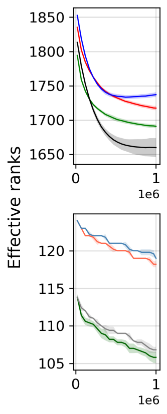
From these results, we observe and conclude that larger networks can be effective in improving deep RL performance. In particular, we achieve consistent performance gains when we widen individual layers rather than going deeper. Consequently, we fix the number of layers to $N^{\text{layer}} = 2$, and only change the number of units to learn larger networks in the following experiments.

5.2 Architecture Comparison

In the next set of experiments, we try to investigate the role of a synergistic combination of connectivity architecture, state representation, and distributed training to allow the usage of larger networks for training deep RL agents. A brief introduction to these techniques is described in Sec. 3.



(a) Average return.



(b) Effective ranks.

Fig. 7: Comparison of connectivity architecture on Ant-v2. Our proposed DenseNet architecture produces the best return on both large ($N^{\text{unit}} = 2048$, denoted by L) and small ($N^{\text{unit}} = 128$, S) networks while mitigating rank collapse as good as MLP-D2RL.

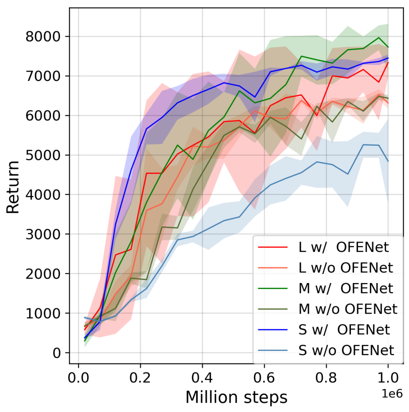
Connectivity architecture

We first compare four connectivity architectures: standard MLP, MLP-ResNet, MLP-DenseNet, and MLP-D2RL, which is a recently proposed architecture to improve RL performance. MLP-ResNet is a modified version of Residual Networks (ResNet) [2, 59], which has a skip-connection that bypasses the nonlinear transformations with an identity function: $y_i = f_i^{\text{res}}(y_{i-1}) + y_{i-1}$, where y_i is the output of the i^{th} layer, and f_i^{res} is a residual module, which consists of a fully connected layer and a nonlinear activation function. An advantage of this architecture is that the gradient can flow directly through the identity mapping from the top layers to the bottom layers. MLP-D2RL is identical to Sinha *et al.* [19], and MLP-DenseNet is our proposed architecture defined in Sec. 3.3. We compare these four architectures on both small networks ($N^{\text{unit}} = 128$, denoted by S) and large networks ($N^{\text{unit}} = 2048$, denoted by L).

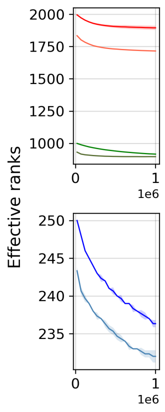
Figure 7 shows the training curves of the average return in Fig. 7a, and the effective ranks in Fig. 7b. The results show that our MLP-DenseNet achieves the highest return on small and large networks while mitigating rank collapse comparable to MLP-D2RL. This shows that MLP-DenseNet is the best architecture among these four choices, and thus we employ this architecture for both the policy network and the value function network in the following experiments.

Decoupling representation learning from RL

Next, we evaluate the effectiveness of using OFENet (see Sec. 3.1) to decouple representation learning from RL. In order to evaluate the performance on different network sizes, we sample the number of units from $N^{\text{units}} \in$



(a) Average return.



(b) Effective ranks.

Fig. 8: Training curves of w/ and w/o OFENet on Ant-v2. This shows decoupling representation learning from RL is generally effective across the different sizes of the networks in terms of both control performance and mitigating rank collapse issues.

$\{256, 1024, 2048\}$, which we respectively denote S, M, and L, and compare these against the baseline SAC agents, which do not use OFENet-like structure with the auxiliary loss and are trained only from a scalar reward signal. In other words, the baseline agents are identical to the DenseNet architecture of the previous connectivity comparison experiment.

The results in Fig. 8 show separating representation learning from RL improves control performance and mitigates rank collapse of Q-networks regardless of network size. Thus, we can conclude using bigger representations, which are learned using the auxiliary task (see Sec. 3.1), contributes to improving performance on downstream RL tasks.

To investigate more in-depth, we also conduct a grid search over the different number of units for both SAC and OFENet in Fig. 9. The baseline is SAC agent without OFENet (see leftmost column). The results suggest that the performance does improve when compared against the baseline agent (see horizontally), however, it saturates around the average return of 8000. In the following experiments, we employ distributed replay and expect we can attain higher performance.

Distributed RL

Finally, we add distributed replay [48] to further improve performance while using larger networks. We use an implementation similar to Stooke *et al.* [55], which collects experiences using N^{core} cores on which each core contains N^{env} environments, specifically we used $N^{\text{core}} = 2$ and $N^{\text{env}} = 32$.

Similarly to the previous experiments, we perform a grid search on the different number of units for SAC and OFENet with the distributed replay in

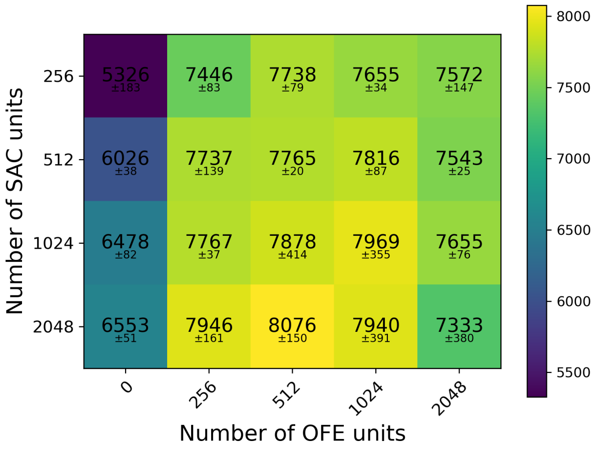


Fig. 9: Grid search results of average maximum return over the different number of units between SAC and OFENet. OFENet can improve performance on almost all settings but saturates around the return of 8000.

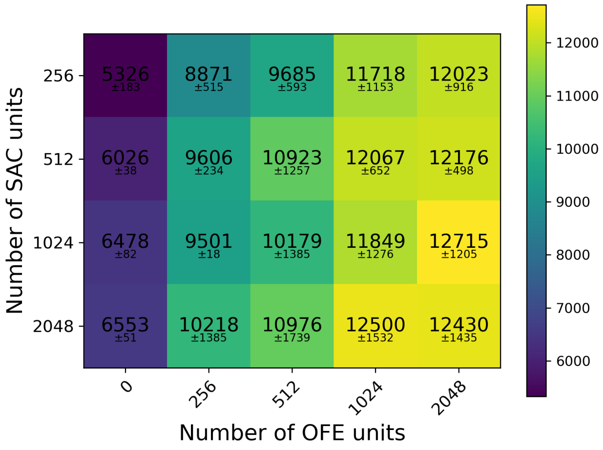
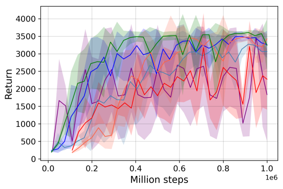
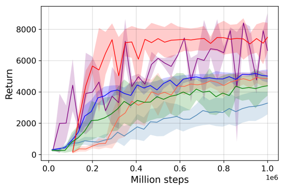


Fig. 10: Grid search results of average maximum return over the different number of units between SAC and OFENet with ApeX-like distributed training. Compared to Fig. 9, adding distributed RL enables monotonic improvement when we widen SAC or OFENet.

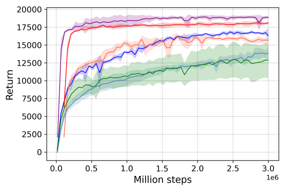
Fig. 10, and also compare the training curves of three different network size S, M, and L in Fig. 12. Note that the horizontal axis in Fig. 12 is the number of times we applied gradients to the network, not the number of interactions. Comparing Fig. 10 and Fig. 9, we can clearly see that distributed training enables further performance gain in all sizes of networks. Furthermore, we can observe a monotonic improvement when we increase the number of units for



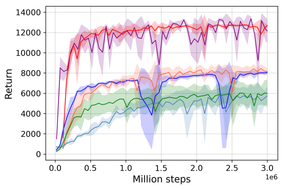
(a) Hopper-v2



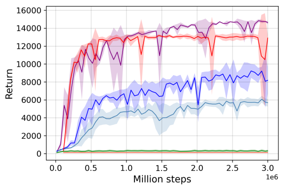
(b) Walker2d-v2



(c) HalfCheetah-v2



(d) Ant-v2



(e) Humanoid-v2

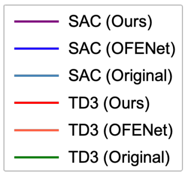


Fig. 11: Training curves on five different MuJoCo tasks with two different RL algorithms (SAC and TD3). The proposed methods denoted as *SAC (Ours)* and *TD3 (Ours)* improve performance by employing larger networks.

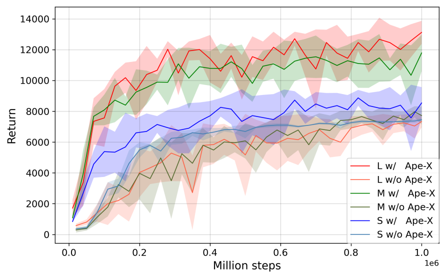


Fig. 12: Training curves on the performance of our proposed framework w/ and w/o using Ape-X architecture on Ant-v2 environment.

both SAC and OFENet. Thus, we verified that combining distributed replay contributes to further performance gain while training larger networks.

How about generalization to different RL algorithms and environments?

To quantitatively measure the effectiveness of our method across different RL algorithms and tasks, we evaluate two popular optimization algorithms, namely SAC and TD3 [20, 52], on five different locomotion tasks in MuJoCo [60]. We denote our method as *Ours*, which uses the largest network of $N^{\text{units}} = 2048$ among the previous experiments for both the OFENet and the RL algorithms. We compare the proposed method against two baselines:

Table 1: The highest average returns for each environment. The bold number indicates the best performance. Our method outperforms OFENet [38] and the original algorithm in most environments.

	SAC		
Environment	Ours	OFENet	Original
Hopper-v2	3467.3	**3511.6**	3316.6
Walker2d-v2	**8802.4**	5237.0	3401.5
HalfCheetah-v2	**19209.9**	16964.1	14116.1
Ant-v2	**14021.0**	8086.2	5953.1
Humanoid-v2	**14858.2**	9560.5	6092.6
	TD3		
Environment	Ours	OFENet	Original
Hopper-v2	3206.7	3488.3	**3613.0**
Walker2d-v2	**7645.8**	4915.1	4515.6
HalfCheetah-v2	**18147.5**	16259.5	13319.9
Ant-v2	**12811.3**	8472.4	6148.6
Humanoid-v2	**13282.0**	120.6	340.5

the original RL algorithm denoted by *Original*. Furthermore, we also compare OFENet, which can achieve current state-of-the-art performance on these tasks to the best of our knowledge.

We plot the training curves in Fig. 11 and list the highest average return in Table 1. In the figure and the table, our method *SAC (Ours)* and *TD3 (Ours)* achieves the best performance in almost all environments. Furthermore, we can see that our proposed method can work with both RL algorithms and thus is agnostic to the choice of the training algorithm. In particular, our method notably achieves much higher episode returns in Ant-v2 and Humanoid-v2, which are harder environments with larger state/action space and more training examples. Interestingly, the proposed method does not achieve reasonable solutions in Hopper-v2, which has the smallest dimensionality among five environments. We consider that the performance in smaller dimension problems saturates early and even additional methods cannot provide any significant performance gain.

5.3 Ablation study

Since our method integrates several ideas into a single agent, we conduct additional experiments to understand what components contribute to the performance gain. We highlight that our method consists of three elements: feature representation learning using OFENet, DenseNet architecture, and distributed training. Furthermore, we compare the results without increasing the network size to reinforce that a larger network improves performance.

Figure 13 shows the ablation study on SAC with Ant-v2 environment. *Full* is our method, which combines all three elements we proposed and uses large networks ($N^{\text{unit}} = 2048, N^{\text{layer}} = 2$) for the SAC agent. *sac* is the original SAC implementation.

w/o Ape-X removes Ape-X-like distributed training setting. As distributed RL enables the collection of more experiences close to the current policy, we consider that the significant performance gain can be explained by learning from more on-policy data, which was also empirically shown by Fedus *et al.* [47]. Also, we believe that receiving more novel experiences helps the agent generalize to state-action space. In other words, although the off-policy training setting obtains a new experience by interacting with the environment, it is much less compared to the on-policy setting, resulting in overfitting to the limited trajectories, which becomes more problematic in harder environments, which have larger state/action space and larger neural networks. Fu *et al.* [26] have also empirically proven this issue.

w/o OFENet removes OFENet and trains the entire architecture using only a scalar reward signal. The much lower return shows that learning the large networks from just the scalar reinforcement signal is difficult, and training the bottom networks (close to the input layer), i.e. obtaining informative features by using an auxiliary task, enables better learning of control policy.

w/o Larger NN reduces the number of units from $N^{\text{unit}} = 2048$ to 256 for both OFENet and SAC. This also significantly decreases performance; therefore, we can conclude that the use of larger networks is essential to achieve high performance.

Finally, *w/o DenseNet* replaces the MLP-DenseNet defined in Sec. 3.3 with the standard MLP architecture. The result shows that strengthening feature propagation does contribute to improving performance.

It is noted that while fully using the proposed architecture improves the performance the most, each component (decoupling representation learning from RL, distributed training, and network architecture) also contributes to the performance gain.

5.4 Sparse reward setting

So far, we have evaluated the proposed framework in dense reward settings. In this section, we apply our framework to the sparse reward settings in multi-goal environments discussed in Plappert *et al.* [61], which includes a 7 DoF robot manipulator (Fetch) to perform Reach, Slide, Push, and PickAndPlace tasks. Since our framework does not need to change the underlying RL algorithms, it can be naturally combined with any plug-and-play method. Specifically, we combine our framework with Hindsight Experience Replay (HER) [62], which is a standard method for solving RL problems with sparse reward settings.

Figure 14 shows the success rates of the four different sparse reward environments. It clearly shows that the proposed framework enables the agent to learn a better policy compared to the baseline method, which is a naive combination of HER and SAC; that is, it does not include the proposed framework

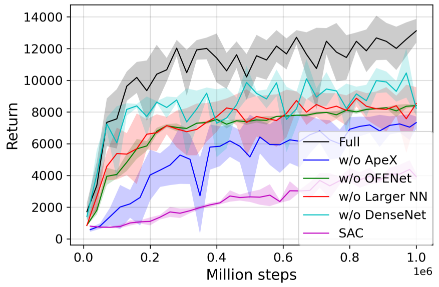


Fig. 13: Training curves of the derived methods of SAC on Ant-v2. This shows that each element does contribute to performance gain, and our combination of DenseNet architecture, distributed training, and decoupled feature representation (shown as *Full*) allows us to train larger networks that perform significantly better compared against the baseline SAC algorithm (shown as *sac*).

that consists of three components. Specifically in difficult settings (Slide, Push, and PickAndPlace), our method quickly converges to a success rate of almost 100 %, while the original algorithm does not achieve the goal in 300 thousands of steps.

5.5 Analysis on computation and sample efficiency

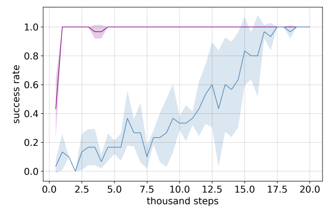
In the final experiment, we evaluate the performance of the proposed framework with respect to *wall clock time* and *sample efficiency*.

Environmental and gradient steps per second

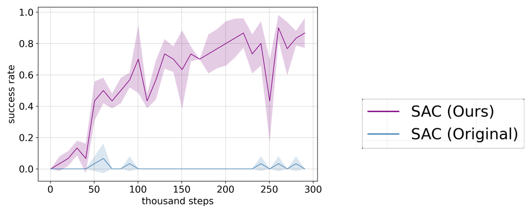
First, we analyze the number of environmental and gradient steps per second for the four different methods. Table 2 compares the gradient and environmental steps averaged over all environments for *Ours*, *Ours w/o ApeX*, *OFENet*, and *Original*. Comparing *Ours* and *Ours w/o ApeX*, which is our proposed framework without the ApeX-like distributed training, the proposed method collects 88(= 1362.0/15.4) times more transitions per second by employing the distributed training framework. Therefore, more on-policy transitions are stored in a replay buffer compared to those without a distributed training framework, which is important to train high-performance policy. Focusing on the gradient steps, *Ours* has more steps than *Ours w/o ApeX*, because our framework conducts policy updates and transition collection asynchronously, which will be effective for performance in terms of wall clock time performance.

Wall clock time performance

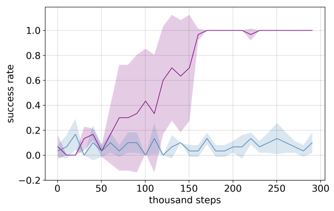
Next, we compare the performance of the models with respect to the *wall clock time*, that is, we compare models trained over a fixed period of time.



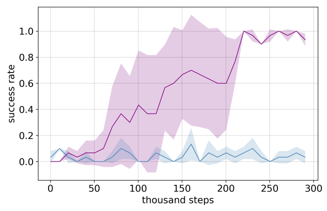
(a) FetchReach-v1



(b) FetchSlide-v1



(c) FetchPush-v1



(d) FetchPickAndPlace-v1

Fig. 14: Training curves on four different sparse reward tasks with SAC combined with Hindsight Experience Replay. The proposed framework denoted as *SAC (Ours)* converges to a success rate of almost 100 % compared to the naive baseline *SAC (Original)* that does not include our proposed framework.

Table 2: The number of environment and gradient steps per second averaged over all environments. Note that the environment and gradient steps are different only for *Ours* because of the distributed training.

	Ours	Ours w/o ApeX	OFENet	Original
Environmental steps per second	1362.0	15.4	29.8	43.9
Gradient steps per second	19.3	15.4	29.8	43.9

Specifically, we compare the performance of all methods for the amount of time taken by the naive baseline *SAC (Original)* to complete the training.

The result in Table 3 shows that our method *SAC (Ours)* achieves the best performance in almost all environments, although the number of gradient steps is 2.3(= 43.9/19.3) times less than *Original*. This is achieved by using the distributed training framework; our method collects more on-policy transitions thanks to the asynchronous data collection setup, which enables the agent to overfit its networks to the experience replay (see Section 3.2 for more details). Thus, we show that the proposed framework is effective with respect to the wall clock time.

Table 3: The highest average returns for each environment with fixed wall clock time, specifically as of the naive baseline *SAC (Original)* agent completed training, for analyzing the performance gain with respect to wall clock time. The numbers in bold indicate the best performance. Our method outperforms OFENet [38] and the original algorithm in most environments for complex environments (HalfCheetah-v2, Ant-v2, and Humanoid-v2).

	SAC		
Environment	Ours	OFENet	Original
Hopper-v2	2798.9	**3406.4**	3316.6
Walker2D-v2	4633.8	**4813.7**	3401.5
HalfCheetah-v2	**18345.3**	16214.91	14116.1
Ant-v2	**13008.5**	7748.0	5953.1
Humanoid-v2	**12678.1**	7944.0	6092.6
	TD3		
Environment	Ours	OFENet	Original
Hopper-v2	1597.0	3458.4	**3613.0**
Walker2D-v2	**7336.2**	4392.5	4515.6
HalfCheetah-v2	**17784.5**	15808.6	13319.9
Ant-v2	**12463.4**	8199.9	6148.6
Humanoid-v2	**12970.8**	119.5	340.5

Sample efficiency

Next, we analyze the sample efficiency, where we compare our method with baselines with the same number of *environmental steps* (thus we do not consider the wall clock time).

Table. 4 shows the performance of each method with a fixed number of environmental steps (1 million for Hopper-v2 and Walker2D-v2 and 3 million for the others). Note that the returns of *SAC (Original)* and *SAC (OFENet)* are identical to those in Table 1 of the manuscript because these methods do not use the distributed training framework. Comparing *Ours* with baselines (*OFENet* and *Original*), our proposed framework performs worse than the baselines. This is because it collects a huge amount of samples very quickly, thanks to the asynchronous training architecture (Fig. 3 in the manuscript); the data collection speed of the proposed framework is 88 times faster than that of the naive baseline. Therefore, effectively, our proposed method is able to apply only fewer gradient steps (approximately 100 times less) than the compared baselines. This results in poor performance, as there are simply not enough updates to train the model.

Finally, to show the effectiveness of the other two components of the proposed framework, namely the DenseNet architecture and decoupling representation learning from RL, we also added results without distributed training,

Table 4: The highest average returns for each environment with the fixed number of environmental steps to analyze the performance with respect to *sample efficiency*. The numbers in bold indicate the best performance. Our method works worse than OFENet [38] and the original algorithm in most environments due to the nature of asynchronous training. However, our framework without distributed training (*Ours w/o ApeX*) works much better than baselines, indicating one can remove distributed training when sample efficiency is most important.

	SAC			
Environment	Ours	Ours w/o ApeX	OFENet	Original
Hopper-v2	532.1	3452.7	**3511.6**	3316.6
Walker2d-v2	621.2	**6385.3**	5237.0	3401.5
HalfCheetah-v2	14548.5	**17942.6**	16964.1	14116.1
Ant-v2	8508.0	**10249.5**	8086.2	5953.1
Humanoid-v2	1177.3	**10720.3**	9560.5	6092.6
	TD3			
Environment	Ours	Ours w/o ApeX	OFENet	Original
Hopper-v2	367.5	3562.1	3488.3	**3613.0**
Walker2d-v2	635.9	**6021.3**	4915.1	4515.6
HalfCheetah-v2	14474.9	**17402.8**	16259.5	13319.9
Ant-v2	7216.5	**9820.9**	8472.4	6148.6
Humanoid-v2	4302.1	**10235.3**	120.6	340.5

denoted by *Ours w/o ApeX* in Table. 4. From the table, *Ours w/o ApeX* achieves much higher performance than the baselines, while it has the same sample efficiency as the baselines, since it does not use distributed training. Therefore, one can choose the framework with or without distributed training based on the importance of sample efficiency; While our full framework is not very sample efficient due to the nature of distributed training, the proposed method without distributed training still performs much better than the baselines. Implementing the distributed framework can really improve performance due to the distributed implementation and the large amounts of on-policy diverse data.

6 Conclusion

Deep Learning has catalyzed huge breakthroughs in the fields of computer vision and natural language processing, making use of massive neural networks that can be trained with huge amounts of data. Although these domains have greatly benefitted from the use of larger networks, the RL community has not witnessed a similar trend in the use of larger networks to train high-performance agents. This is mainly due to the instability when using larger networks to train RL agents. In this paper, we studied the problem of using a larger network to train RL agents. To achieve this, we proposed a novel framework for training larger networks for deep RL agents, while reflecting on some of the important design choices that one has to make when using such networks. In particular, the proposed framework consists of three elements. First, we decouple the representation learning from RL using an auxiliary loss to predict the next state. This allows more informative features to be obtained to learn control policies with richer information than learning entire networks from a scalar reward signal. The learned representation is then propagated to the DenseNet architecture, which consists of very wide networks. Finally, a distributed training framework provides huge amounts of on-policy data whose distribution is much closer to the current policy and thus enables the RL agent to mitigate the overfitting problem and enhance generalization to novel scenarios. Our experiments demonstrate that this novel combination achieves significantly higher performance compared to current state-of-the-art algorithms across different off-policy RL algorithms and different continuous control tasks. While this paper focused on learning from state vectors, we plan to apply the proposed framework to high-dimensional observations, such as images, in future work. Furthermore, motivated by a surge of interest in applying Transformer-based networks to RL agents [63–66], we also plan to include a self-attention-based architecture to our framework. Nevertheless, we believe that our approach could be helpful toward training larger networks for Deep RL agents.

7 Declarations

- Funding: This work is not funded by any institution.
- Conflicts of interest/Competing interests: Not applicable.
- Ethics approval: Not applicable.
- Consent to participate: Not applicable.
- Consent for publication: Yes.
- Availability of data and material: Not applicable.
- Code availability: Upon acceptance, we will upload codes on GitHub.
- Authors' contributions: Kei Ota participated in developing the paper's idea, implementing codes, doing experiments, and writing the paper. Devesh K. Jha and Asako Kanezaki participated in developing the paper's idea and writing the paper.

References

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Proceedings of Advances in Neural Information Processing Systems (NeurIPS), vol. 25, pp. 1097–1105. Curran Associates, Inc., ??? (2012)

[2] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

[3] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2261–2269 (2017). https://doi.org/10.1109/CVPR.2017.243

[4] Chen, T., Kornblith, S., Swersky, K., Norouzi, M., Hinton, G.: Big self-supervised models are strong semi-supervised learners. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS) (2020)

[5] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (2019)

[6] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Proceedings of Advances in Neural Information Processing Systems (NeurIPS) (2020)

[7] Kaplan, J., McCandlish, S., Henighan, T., Brown, T.B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., Amodei, D.: Scaling laws for neural language models. arXiv preprint arXiv:2001.08361 (2020) arXiv:2001.08361 [cs.LG]

[8] Zhou, C., Li, Q., Li, C., Yu, J., Liu, Y., Wang, G., Zhang, K., Ji, C., Yan, Q., He, L., et al.: A comprehensive survey on pretrained foundation models: A history from bert to chatgpt. arXiv preprint arXiv:2302.09419 (2023)

[9] Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R.J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., Zeng, A.: Do as i can and not as i say: Grounding language in robotic affordances. In: arXiv Preprint arXiv:2204.01691 (2022)

[10] Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jackson, T., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, K.-H., Levine, S., Lu, Y., Malla, U., Manjunath, D., Mordatch, I., Nachum, O., Parada, C., Peralta, J., Perez, E., Pertsch, K., Quiambao, J., Rao, K., Ryoo, M., Salazar, G., Sanketi, P., Sayed, K., Singh, J., Sontakke, S., Stone, A., Tan, C., Tran, H., Vanhoucke, V., Vega, S., Vuong, Q., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., Zitkovich, B.: Rt-1: Robotics transformer for real-world control at scale. In: arXiv Preprint arXiv:2212.06817 (2022)

[11] Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., Srebro, N.: The role of over-parametrization in generalization of neural networks. In: Proceedings of International Conference on Learning Representations (ICLR) (2019). https://openreview.net/forum?id=BygfghAcYX

[12] Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: a view from the width. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), pp. 6232–6240 (2017)

[13] Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: Proceedings of International Conference on Learning Representations (ICLR) (2017)

[14] Nguyen, Q., Hein, M.: The loss surface of deep and wide neural networks. In: Proceedings of International Conference on Machine Learning (ICML), pp. 2603–2612 (2017)

[15] Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), pp. 6391–6401 (2018)

[16] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of AAAI Conference on Artificial Intelligence (2018)

[17] van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., Modayil,

J.: Deep reinforcement learning and the deadly triad. CoRR (2018)

[18] Achiam, J., Knight, E., Abbeel, P.: Towards characterizing divergence in deep q-learning. arXiv preprint arXiv:1903.08894 (2019)

[19] Sinha, S., Bharadhwaj, H., Srinivas, A., Garg, A.: D2rl: Deep dense architectures in reinforcement learning. arXiv preprint arXiv:2010.09163 (2020) arXiv:2010.09163 [cs.LG]

[20] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. CoRR (2018) arXiv:1801.01290 [abs]

[21] Deng, J., Dong, W., Socher, R., Li, L., Kai Li, Li Fei-Fei: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition (2009)

[22] Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: Proceedings of International Conference on Machine Learning (ICML), pp. 6105–6114 (2019). PMLR

[23] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., *et al.*: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)

[24] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)

[25] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M.G., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Proceedings of AAAI Conference on Artificial Intelligence (2018)

[26] Fu, J., Kumar, A., Soh, M., Levine, S.: Diagnosing bottlenecks in deep q-learning algorithms. In: Proceedings of International Conference on Machine Learning (ICML) (2019)

[27] Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., Bachem, O.: What matters in on-policy reinforcement learning? a large-scale empirical study. In: Proceedings of International Conference on Learning Representations (ICLR) (2021)

[28] Liu, Z., Li, X., Kang, B., Darrell, T.: Regularization matters in policy optimization. In: Proceedings of International Conference on Learning

Representations (ICLR) (2021)

[29] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)

[30] He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)

[31] Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. In: Proceedings of International Conference on Machine Learning (ICML), pp. 1597–1607 (2020)

[32] Jaderberg, M., Mnih, V., Czarnecki, W.M., Schaul, T., Leibo, J.Z., Silver, D., Kavukcuoglu, K.: Reinforcement learning with unsupervised auxiliary tasks. In: Proceedings of International Conference on Learning Representations (ICLR) (2017)

[33] Shelhamer, E., Mahmoudieh, P., Argus, M., Darrell, T.: Loss is its own reward: Self-supervision for reinforcement learning. In: Proceedings of International Conference on Learning Representations (ICLR) (2017)

[34] Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: Proceedings of Advances in Neural Information Processing Systems (NeurIPS), (2018)

[35] Li, X., Shang, J., Das, S., Ryoo, M.: Does self-supervised learning really improve reinforcement learning from pixels? Advances in Neural Information Processing Systems **35**, 30865–30881 (2022)

[36] Munk, J., Kober, J., Babuška, R.: Learning state representation for deep actor-critic control. In: IEEE Conference on Decision and Control (CDC), pp. 4667–4673 (2016)

[37] Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., Filliat, D.: State Representation Learning for Control: An Overview. CoRR (2018) arXiv:1802.04181 [abs]

[38] Ota, K., Oiki, T., Jha, D., Mariyama, T., Nikovski, D.: Can increasing input dimensionality improve deep reinforcement learning? In: Proceedings of International Conference on Machine Learning (ICML), pp. 7424–7433 (2020)

[39] Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., *et al.*: Automated reinforcement learning (autorl): A survey and open problems. Journal of

Artificial Intelligence Research **74**, 517–568 (2022)

[40] Wan, X., Lu, C., Parker-Holder, J., Ball, P.J., Nguyen, V., Ru, B., Osborne, M.: Bayesian generational population-based training. In: International Conference on Automated Machine Learning, pp. 14–1 (2022). PMLR

[41] Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W.M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al.: Population based training of neural networks. arXiv preprint arXiv:1711.09846 (2017)

[42] Mohan, A., Benjamins, C., Wienecke, K., Dockhorn, A., Lindauer, M.: Autorl hyperparameter landscapes. arXiv preprint arXiv:2304.02396 (2023)

[43] Stooke, A., Lee, K., Abbeel, P., Laskin, M.: Decoupling representation learning from reinforcement learning. arXiv preprint arXiv:2009.08319 (2020) arXiv:2009.08319 [cs.LG]

[44] Henaff, O.: Data-efficient image recognition with contrastive predictive coding. In: Proceedings of International Conference on Machine Learning (ICML), pp. 4182–4192 (2020)

[45] Hernandez, D., Kaplan, J., Henighan, T., McCandlish, S.: Scaling laws for transfer. arXiv preprint arXiv:2102.01293 (2021) arXiv:2102.01293 [cs.LG]

[46] Ramchoun, H., Idrissi, M.A.J., Ghanou, Y., Ettaouil, M.: New modeling of multilayer perceptron architecture optimization with regularization: an application to pattern classification. IAENG International Journal of Computer Science **44**(3), 261–269 (2017)

[47] Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., Dabney, W.: Revisiting fundamentals of experience replay. In: Proceedings of the 37th International Conference on Machine Learning, pp. 3061–3071 (2020)

[48] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., Silver, D.: Distributed prioritized experience replay. arXiv preprint arXiv:1803.00933 (2018)

[49] Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., Munos, R.: Recurrent experience replay in distributed reinforcement learning. In: Proceedings of International Conference on Learning Representations (ICLR) (2019)

[50] Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. In: Proceedings of International Conference on Learning Representations (ICLR) (2016)

[51] Aviral Kumar, D.G. Rishabh Agarwal, Levine, S.: Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In: Proceedings of International Conference on Learning Representations (ICLR) (2021)

[52] Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of International Conference on Machine Learning (ICML) (2018)

[53] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of International Conference on Machine Learning (ICML) (2015)

[54] Ramachandran, P., Zoph, B., Le, Q.V.: Searching for Activation Functions. CoRR (2017) arXiv:1710.05941 [abs]

[55] Stooke, A., Abbeel, P.: Accelerated methods for deep reinforcement learning. arXiv preprint arXiv:1803.02811 (2018)

[56] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Artificial Intelligence and Statistics (AISTATS). Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, Italy (2010). https://proceedings.mlr.press/v9/glorot10a.html

[57] Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of British Machine Vision Conference (BMVC) (2016)

[58] Wu, Z., Shen, C., Van Den Hengel, A.: Wider or deeper: Revisiting the resnet model for visual recognition. Pattern Recognition **90**, 119–133 (2019)

[59] He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Proceedings of European Conference on Computer Vision (ECCV) (2016)

[60] Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2012)

[61] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al.: Multi-goal

reinforcement learning: Challenging robotics environments and request for research. arXiv preprint arXiv:1802.09464 (2018)

[62] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., Zaremba, W.: Hindsight experience replay. Proceedings of Advances in Neural Information Processing Systems (NeurIPS) **30** (2017)

[63] Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z., Ye, D.: A survey on transformers in reinforcement learning. Transactions on Machine Learning Research (2023). Survey Certification

[64] Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., Mordatch, I.: Decision transformer: Reinforcement learning via sequence modeling. Advances in neural information processing systems **34**, 15084–15097 (2021)

[65] Shang, J., Kahatapitiya, K., Li, X., Ryoo, M.S.: Starformer: Transformer with state-action-reward representations for visual reinforcement learning. In: European Conference on Computer Vision, pp. 462–479 (2022). Springer

[66] Konan, S.G., Seraj, E., Gombolay, M.: Contrastive decision transformers. In: Liu, K., Kulic, D., Ichnowski, J. (eds.) Proceedings of The 6th Conference on Robot Learning. Proceedings of Machine Learning Research, vol. 205, pp. 2159–2169. PMLR, ??? (2023). https://proceedings.mlr.press/v205/konan23a.html