TASK ADAPTATION FOR CONTINUOUS CONTROL USING DEEP REINFORCEMENT LEARNING

Shreyas Kulkarni (S212031)

Denmark Technical University

ABSTRACT

In this work, we compare how a model free Reinforcement Learning agent learns in a different manner than an actual human by performing experiments over DeepMind Control Suite's continuous control environments and tasks. We utilize latest research in Deep Reinforcement Learning, Representation learning and Generalization via Augmentations. A comparison has also been done between 2D and 3D environments. Limitations in complex 3D environments are observed and possible improvements are explored in this process. The code can be found at: github link

***Index Terms*—** Reinforcement Learning, Deep Learning, Computer Vision, Task Adaptation, Cognitive modelling

1. INTRODUCTION

Unlike board games, language and other symbolic domains, physical tasks are fundamentally continuous in state, time and action. Their dynamics are subject to second-order equations of motion,implying that the underlying state is composed of position-like and velocity-like variables, while state derivatives are acceleration-like. In this work, we use DMC: The DeepMind Control Suite [1] which is a starting place for the design and performance comparison of reinforcement learning algorithms for physics-based control. It offers a wide range of tasks, from near-trivial to quite difficult. The uniform reward structure allows for robust suite-wide performance measures.

Model-free deep reinforcement learning (RL) algorithms have been successfully applied to a range of challenging sequential decision making and control tasks. However, these methods typically suffer from two major challenges: model-free deep RL methods are notoriously expensive in terms of their sample complexity. Even relatively simple tasks can require millions of steps of data collection, and complex behaviors with high-dimensional observations might need substantially more. One cause for the poor sample efficiency of deep RL methods is on-policy learning: some of the most commonly used deep RL algorithms, such as TRPO [2], PPO [3] or A3C [4], require new samples to be collected for (nearly) every update to the policy. This quickly becomes extravagantly expensive, as the number of gradient steps and samples per step needed to learn an effective policy increases with task complexity. Off-policy algorithms aim to reuse past experience. This is not directly feasible with conventional policy gradient formulations, but is relatively straightforward for Q-learning based methods. Unfortunately, the combination of off-policy learning and high-dimensional, nonlinear function approximation with neural networks presents a major challenge for stability and convergence [5]. This challenge is further exacerbated in continuous state and action spaces, where a separate actor network is often used to perform the maximization in Q-learning.

Secondly, these methods are often brittle with respect to their hyperparameters: learning rates, exploration constants, and other settings must be set carefully for different problem settings to achieve good results. Both of these challenges limit the applicability of such methods to real-world domains. To tackle these problems, Soft Actor-Critic (SAC) [6], an off-policy actor-critic algorithm based on the maximum entropy RL framework was developed. In this framework, the actor aims to simultaneously maximize expected return and entropy; that is, to succeed at the task while acting as randomly as possible.

Training an agent to solve control tasks directly from high dimensional images with model-free reinforcement learning (RL) has proven difficult. A promising approach is to learn a latent representation together with the control policy. However, fitting a high-capacity encoder using a scarce reward signal is sample inefficient and leads to poor performance. Prior work has shown that auxiliary losses, such as image reconstruction, can aid efficient representation learning. However, incorporating reconstruction loss into an off-policy learning algorithm often leads to training instability. A glaring issue is that the RL signal is much sparser than in supervised learning, which leads to sample inefficiency, and higher dimensional observation spaces such as pixels worsens this problem.

Directly learning from raw images posses an additional problem of partial observability, which is formalized by a partially observable MDP (POMDP). This complicates applying RL as the agent now needs to also learn a compact latent representation to infer the state. Fitting a high-capacity encoder using only a scarce reward signal is sample inefficient

and prone to suboptimal convergence. Following prior work [7, 8], SAC+AE [9] auguments SAC with a regularized autoencoder to achieve stable training from images in the off-policy regime.

Increasing the variability in training data via domain randomization and data augmentation has demonstrated encouraging results for learning policies invariant to changes in environment observations. Specifically, recent works on data augmentation [10, 11] both show improvements in sample efficiency from simple cropping and translation augmentations, but the studies also conclude that additional data augmentation in fact decrease sample efficiency and even cause divergence. While these augmentations have the potential to improve generalization, the increasingly varied data makes the optimization more challenging and risks instability.

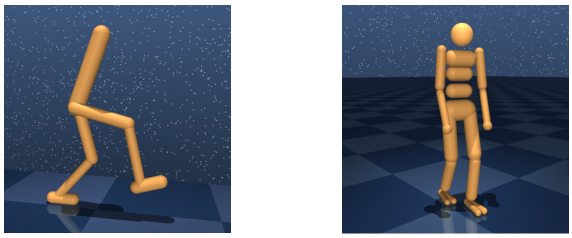
The main causes of instability in previous work's application of data augmentation are: (i) indiscriminate application of data augmentation resulting in high-variance Q-targets; and (ii) that Q-value estimation strictly from augmented data results in over-regularization. Firstly, by only applying augmentation in Q-value estimation of the current state, without augmenting Q-targets used for bootstrapping, SVEA [12] circumvents erroneous bootstrapping caused by data augmentation; Secondly, they formulate a modified Q-objective that optimizes Q-value estimation jointly over both augmented and unaugmented copies of the observations; Lastly, for SVEA implemented with an actor-critic algorithm, they optimize the actor strictly on unaugmented data, and instead learn a generalizable policy indirectly through parameter-sharing.

We study both the case of learning with state derived features as observations and learning from raw-pixel inputs for all the tasks in the Control Suite. It is of course possible to look at control via combined state features and pixel features, but we do not study this case here. We present results for both final performance and learning curves that demonstrate aspects of data-efficiency and stability of training.

2. ENVIRONMENT: DEEP MIND CONTROL SUITE

DMC is built upon MuJoCo which is a fast, minimal-coordinate, continuous-time physics engine. The convenient MJCF definition format and reconfigurable computation pipeline have made MuJoCo popular for robotics and reinforcement learning research [2]. We consider experiments in two domains: Walker and Humanoid and 3 tasks of Standing, Walking and Running for each of the domains. Both the environments have dense rewards. In the domain descriptions below, S = State Space, A = Control Space and O = Observation Space.

Walker is a planar environment and the Camera 1 takes images as seen in Figure 1.(a). Its properties are as follows: $dim(S) = 18$, $dim(A) = 6$, $dim(O) = 24$. In the stand task reward is a combination of terms encouraging an upright torso and some minimal torso height. The walk and run tasks include a component encouraging forward velocity.



(a) Walker (b) Humanoid

Fig. 1. DMC Domain Images from Camera 1

Humanoid is 3 dimensional and the camera is egocentric, i.e, the camera moves according to how the Humanoid moves in the 3D environment. Its properties are as follows: $dim(S) = 54$, $dim(A) = 21$, $dim(O) = 67$. Three tasks: stand, walk and run are differentiated by the desired horizontal speed of 0, 1 and 10m/s, respectively.

Control Suite tasks have no terminal states or time limit and are therefore of the infinite-horizon variety. Since all reward functions are designed so that r = 1 at or near a goal state, learning curves measuring total returns all have the same y-axis limits of [0, 1000] (considering a fixed time step of episode=1000), making them easier to interpret. Thus, the maximum possible score for any task is 1000. For many tasks, the practical maximum is significantly less than 1000 since it may take many steps until it's possible to drive the system into a state that gives a full reward of 1.0 each time. step.

When using the default observations (rather than pixels), all tasks are strongly observable, i.e. the state can be recovered from a single observation. Observation features which depend only on the state (position and velocity) are functions of the current state. Features which are also dependent on controls (e.g. touch sensor readings) are functions of the previous transition. [13]

3. PREVIOUS WORKS

3.1. SAC: Soft Actor Critic

Instead of only seeking to maximize the lifetime rewards, SAC seeks to also maximize the entropy of the policy. We want a high entropy in our policy to explicitly encourage exploration, to encourage the policy to assign equal probabilities to actions that have same or nearly equal Q-values, and also to ensure that it does not collapse into repeatedly selecting a particular action that could exploit some inconsistency in the approximated Q function. Therefore, SAC overcomes the brittleness problem by encouraging the policy network to explore and not assign a very high probability to any one part of the range of actions.

SAC makes use of three networks: a state value function

V parameterized by ψ, a soft Q-function Q parameterized by θ, and a policy function π parameterized by ϕ. In practice having separate function approximators help in convergence. The Value network is trained by minimizing the following error:

$$\mathbf{J}_V(\psi) = \mathbb{E}_{(\mathbf{s_t}\sim\mathcal{D})}\left[\frac{1}{2}(\mathbf{V}_\psi(\mathbf{s_t}) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}\left[Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right])^2\right] \tag{1}$$

Hence, when sampling takes place from our experience replay buffer, there should be decrease in the squared difference between the prediction of the value network and the expected prediction of the Q function plus the entropy of the policy function π (measured here by the negative log of the policy function). The below approximation of the derivative of the above objective is used to update the parameters of the V function:

$$\hat{\nabla}_\psi \mathbf{J}_V(\psi) = \nabla_\psi \mathbf{V}_\psi(\mathbf{s}_t)(\mathbf{V}_\psi(\mathbf{s}_t) - \mathbf{Q}_\theta(\mathbf{s}_t,\mathbf{a}_t) + \log\pi_\phi(\mathbf{a_t}|\mathbf{s}_t)) \tag{2}$$

In the Q-network, the following error is minimized:

$$\mathbf{J}_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim D}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \hat{Q}(\mathbf{s}_t,\mathbf{a}_t)\right)^2\right] \tag{3}$$

$$\hat{Q}(\mathbf{s}_t,\mathbf{a}_t) = r(\mathbf{s}_t,\mathbf{a}_t) + \gamma\,\mathbb{E}_{\mathbf{s_{t+1}}\sim p}\left[\mathbf{V}_{\bar{\psi}}(\mathbf{s}_{t+1})\right] \tag{4}$$

Note that the Value comes from a Value function parameterized by ψ with a bar on top of it. This is an additional Value function called the target value function. The use of target networks is motivated by a problem in training V network. The below approximation of the derivative of the above objective is used to update the parameters of the Q function:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t,\mathbf{s}_t)(Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - r(\mathbf{s}_t,\mathbf{a}_t) - \gamma\, V_{\bar{\psi}}(\mathbf{s}_{t+1})) \tag{5}$$

The Policy network π minimizes the following error:

$$J_\pi(\phi) = \mathbb{E}_{(\mathbf{s}_t\sim D)}\left[D_{KL}\left(\pi_\phi(.|\mathbf{s}_t)\middle\|\frac{exp(Q_\theta(\mathbf{s}_t,.))}{Z_\theta(\mathbf{s}_t)}\right)\right] \tag{6}$$

This objective function is trying to make the distribution of our Policy function look more like the distribution of the exponentiation of our Q Function normalized by another function Z. In order to minimize this objective, the reparameterization trick is used to make sure that sampling from the policy is a differentiable process so that there are no problems in backpropagating the errors. The policy is now parameterized as follows:

$$\mathbf{a}_t = f_\phi(\epsilon_t;\mathbf{s}_t) \tag{7}$$

The epsilon term is a noise vector sampled from a Gaussian distribution. The objective function is represented as follows:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D},\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi(f_\phi(\epsilon_t;\mathbf{s}_t)|\mathbf{s}_t) - Q_\theta(\mathbf{s}_t,f_\phi(\epsilon_t;\mathbf{s}_t))\right] \tag{8}$$

An unbiased estimator for the gradient of the above objective is given as follows:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) + (\nabla_{\mathbf{a}_t}\log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) - \nabla_{\mathbf{a}_t}Q(\mathbf{s}_t,\mathbf{a}_t))\nabla_\phi f_\phi(\epsilon_t;\mathbf{s}_t)$$

Note: The V network has a target that's indirectly dependent on itself which means that the V network's target depends on the same parameters we are trying to train. This makes training very unstable. The solution is to use a set of parameters which comes close to the parameters of the main V network, but with a time delay. Thus a second network which lags the main network called the target network is created. Either the target network is copied over from the main network regularly after a set number of steps or the target network is updated by Polyak averaging (a kind of moving averaging) itself and the main network.

There is a new version of the algorithm that uses only a Q function and disposes of the V function. It also adds automatic discovery of the weight of the entropy term called the 'temperature' [14].

3.2. SAC with Autoencoders in Model-Free Reinforcement Learning from Images

This work had noted a dramatic gap in an agent's performance when it learns from image-based observations rather than low-dimensional proprioceptive states. This result suggests that attaining a compact state representation is key in enabling efficient RL from images. Prior work has demonstrated that auxiliary supervision can improve representation learning and this work tries to utilize the same in our task at hand.

In this setting, instead of getting a low-dimensional state $s_t \in S$ at time t, the agent receives a high-dimensional observation $o_t \in O$, which is a rendering of potentially incomplete view of the corresponding state s_t of the environment. In practice, the Autoencoder is represented as a convolutional encoder g_ϕ that maps an image observation o_t to a low-dimensional latent vector z_t, and a deconvolutional decoder f_θ that reconstructs z_t back to the original image o_t. Here, they only reconstruct the current frame, instead of reconstructing a temporal sequence of frames.

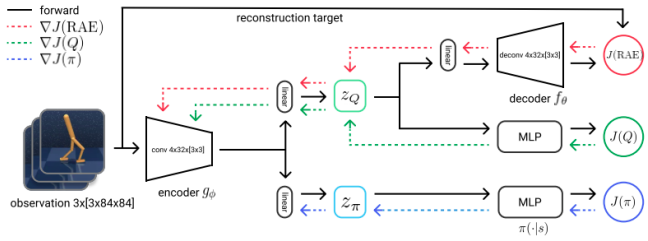


Fig. 2. Pipeline of SAC+AE

The stability in the pipeline comes from switching to a deterministic encoder that is carefully updated with gradients from the reconstruction J(RAE) and soft Q-learning objectives. J(RAE) is given by: [9]

$$J(RAE) = \mathbb{E}_{\mathbf{o}_t \sim \mathcal{D}} \left[\log p_\theta(\mathbf{o}_t|\mathbf{z}_t) + \lambda_\mathbf{z}||\mathbf{z}_t||^2 + \lambda_\theta||\theta||^2 \right] \tag{9}$$

3.3. SVEA: Stabilized Q-Value Estimation under Augmentation

An observation of s_t is transformed by data augmentation $\tau(.,v)$, $v \sim \mathcal{V}$ to produce a view s_t^{aug}. The Q-function Q_θ is then jointly optimized on both augmented and unaugmented data with respect to the objective in Eq. 10, with the Q-target of the Bellman equation computed from an unaugmented observation s_{t+1}. The Algorithm can be illustrated as follows:

Algorithm 1 Generic **SVEA** off-policy algorithm

θ, θ_π, ψ:randomly initialized network parameters ▷ Initialize ψ to be equal to θ
η, ζ:learning rate and momentum coefficient
α, β:loss coefficients, default: $\alpha = 0.5, \beta = 0.5$)
for each timestep $t = 1...T$ **do**
  act:
  $\mathbf{a}_t \sim \pi_\theta(.|f_\theta(\mathbf{s}_t))$ ▷ Sample action from policy
  $\mathbf{s}_t^{'} \sim \mathcal{P}(.|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from environment
  $\mathcal{B} \leftarrow \mathcal{B} \cup (\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_t^{'})$ ▷ Store transition in replay pool
  update:
  $\{\mathbf{s}_i, \mathbf{a}_i, r(\mathbf{s}_i, \mathbf{a}_i), \mathbf{s}_i^{'} | i = 1...N\}$ ▷ Sample batch of transitions
  $\mathbf{s}_i = \tau(\mathbf{s}_i, \mathbf{a}_i, r(\mathbf{s}_i, \mathbf{a}_i), \mathbf{s}_i^{'} = \tau(\mathbf{s}_i^{'}, \mathbf{v}_i^{'}), v_i, v_i^{'} \sim \mathcal{V}$
  ▷ Naive Application of data augmentation
  for transiton $i = 1..N$ **do**
    $\theta_\pi \leftarrow \theta_\pi - \eta \nabla_{\theta_\pi} \mathcal{L}_\pi(\mathbf{s}_i; \theta\pi)$ ▷ Optimize π_θ with SGD
    $q_i^{\text{tgt}} = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma max_{\mathbf{a}_i^{'}} Q_p si^{\text{tgt}}(f_\psi^{\text{tgt}}(\mathbf{s}_i^{'}), \mathbf{a}_i^{'})$
    ▷ Compute Q-target
    $\mathbf{s}_i^{\text{aug}} = \tau(\mathbf{s}_i, \mathbf{v}_i), v_i \sim \mathcal{V}$ ▷ Apply Stochastic data augmentation
    $\mathbf{g}_i = \left[\mathbf{s}_i, \mathbf{s}_i^{\text{aug}}\right]_N, h_i = \left[\mathbf{q}_i^{\text{tgt}}, \mathbf{q}_i^{\text{tgt}}\right]_N$ ▷ Pack Data Streams
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_Q^{\text{SVEA}}(\mathbf{g}, h_i; \theta, \psi)$ ▷ Optimize f_θ and Q_θ with SGD
    $\psi \leftarrow (1-\zeta)\psi + \zeta\theta$ ▷ Update ψ using EMA of θ
  end for
end for

► naive augmentation ► SVEA modification

$$\mathcal{L}_Q^{\text{SVEA}}(\theta, \psi) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1} \sim \mathcal{B}, v \sim \mathcal{V}} \left[(\alpha + \beta)||Q_\theta(f_\theta(\mathbf{g}_t), \mathbf{a}_t) - h_t||_2^2 \right] \tag{10}$$

$$\mathbf{g}_t = \left[\mathbf{s}_t, \tau(\mathbf{s}_t, v)\right]_N$$
$$h_t = \left[q_t^{\text{tgt}}, q_t^{\text{tgt}})\right]_N$$

Their method outperformed all state of the art methodologies, majorly: CURL [15], a contrastive learning method for RL, RAD [16] that applies a random crop, DrQ [17] that applies a random shift, PAD [18] that adapts to test environments using self-supervision and SODA [19] that applies data augmentation in auxiliary learning.

They also study augmentations in detail and observe that weak augmentations such as small random translations improve sample efficiency due to their regularization, and strong augmentations such as random convolution improve generalization at the expense of sample efficiency. They focus on stabilizing deep Q-learning under strong data augmentation with the goal of improving generalization. They experiment with a diverse set of data augmentations proposed in previous work on RL and conclude with usage of random convolution [20] and random overlay.

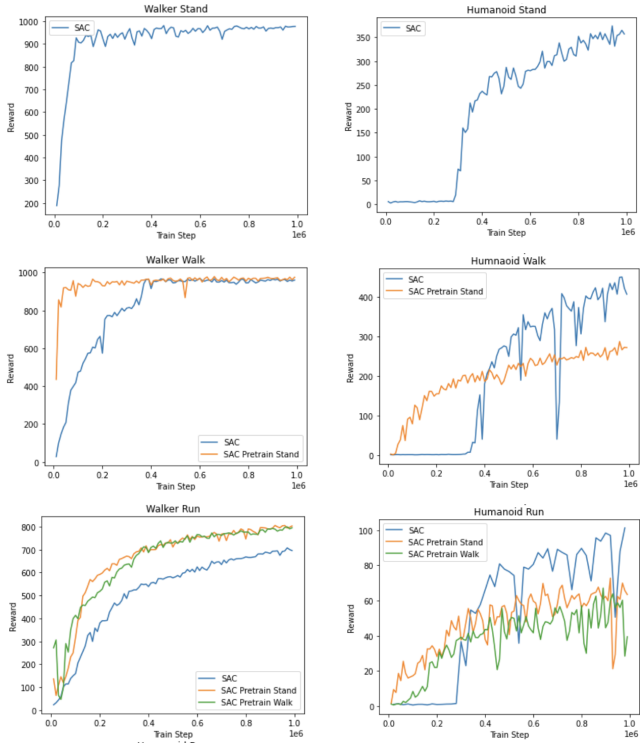


Table 1. Learning Curves (Rewards gained vs Number of Training Steps) for State derived features

4. EXPERIMENTS

To relate RL with how humans think, we perform pretraining experiments. We know that standing, walking and running are tasks with increasing order of difficulty. Hence, if we want our agent to learn a more difficult task, our intuition says that an agent which can do an easier task well shall be able to learn the difficult task faster. We perform these experiments separately for both state derived features as well as raw pixels as observations.

4.1. State derived features

As the humanoid domain has larger action and state space, we were expecting it to be more difficult to train than the walker domain. This turns to be true from our experiments. The humanoid tasks take longer to train and have slower convergence. For each of the tasks we see a steady increase in learning but in one million steps, the humanoid SAC agent is barely able to achieve half of the maximum possible rewards.

We could have trained for longer episodes in the case of humanoid but we wanted to use consistent number of episodes and were bound by the amount of time it takes to train each experiment (We trained each experiment for 3 days). For each humanoid basic tasks (the ones without pretraining), we observe no improvement in rewards for about the first 0.3 million steps. Our justification is that figuring out how to get up (even slightly) from the fallen state is one of the very few possibilities with all the high action spaces. Hence, the agent takes some time to explore the possibility. Once, it finds out how to increase its (center of gravity) distance from the ground, the rate of learning of the agent increases.

In case of walker walk pretrained with stand, we can see that the agent starts initially with half of the maximum rewards. This is majorly possible due to how related the tasks walk and stand are. Furthermore, we clearly see that the pretrained stand version outperforms the standard SAC because of this advantage. However, both the pretrained walk and pretrained stand have almost equal rate of learning and convergence. This is in contradictory to our usual thinking that a bot which knows how to walk should be able to learn to run much faster than the bot which knows how to stand. But, Reinforcement Learning agents learn in a different way. The first few layers of the network learn to extract the features from the state representation while the next few layers adapt the model to any other task. Hence, it would have not mattered if what our pretraining task is, even if the pretraining task was to hop, the agent will show similar learning performance. Do note that the pretrain walk got an initial headstart of around 300 points as compared to 100 points of pretrain stand. But, after some steps of training, they both showed similar learning curves.

Even in case of humanoid pretrainings, we observe the similarity in pretraining stand and walk for task of run. But none of the pretrainings outperform the standard trainings. The major reason behind this is that, neither humanoid walk nor humanoid stand have been able to learn the task near perfectly (they never acheive points more than 500/1000). Hence, we can assume that the encodings learnt in the initial layers are not perfect and that would have impacted the overall pretraining performance. It is however curious to note that the pretrained agents learn faster in the first few steps but it looks like the approach they are learning is not helping them towards the task in the future training steps.

4.2. Raw-Pixel Input

In case of walker tasks, we notice that SAC+AE has much poor performance than that of SAC on state derived features. However, SVEA performs sometimes even better than SAC on state derived features above. We notice that no learning takes place in the case of humanoid tasks. Hence, we can say that the pixel level images have very insufficient information for the humanoid tasks. As it is 3D in nature, the humanoid would have much more options to move. We did try to use depth maps instead of images too but that did not help the agents much. We have trained all the pixel level input tasks for half the number of steps as compared to above to keep the resources and time exhausted by each models equivalent.

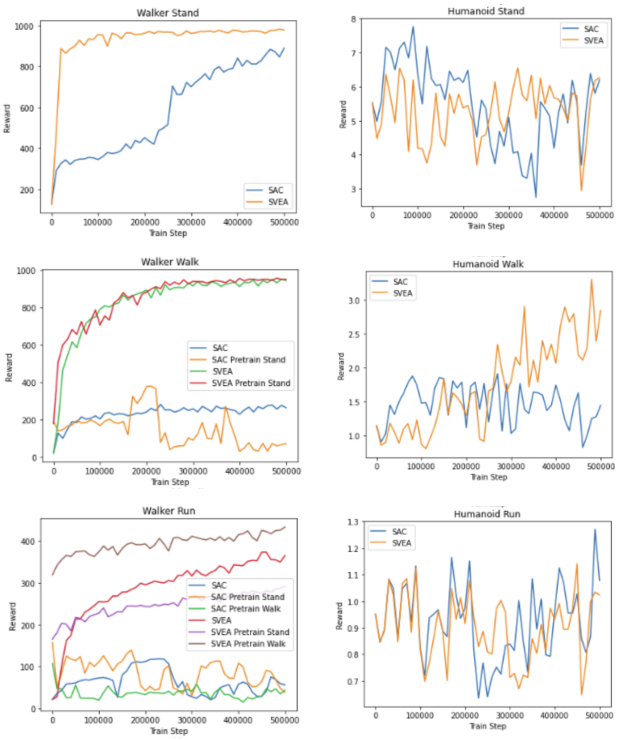


Table 2. Learning Curves (Rewards gained vs Number of Training Steps) for Raw-Pixel Input

We notice that SVEA walker walk performs equivalent to its pretrained stand version. Hence, an attempt to generalization using augmentations does not result in that helpful pretrained weights. Again, even in case of SVEA walker run pretrained with stand and walk, it just gives the model an initial advantage but not much learning takes place in over 0.5 million steps. With similar logic as in state derived features, we observe that the SAC+AE pretraining doesn't help as the representations in initial layers have not been learnt near perfectly.

4.3. Video Snapshots

This subsection is an illustration over time of how the agents have learnt. For this, we have taken a successful and an unsuccessful example. The more green the agent walker or humanoid looks, the higher the rewards it receives that instance.

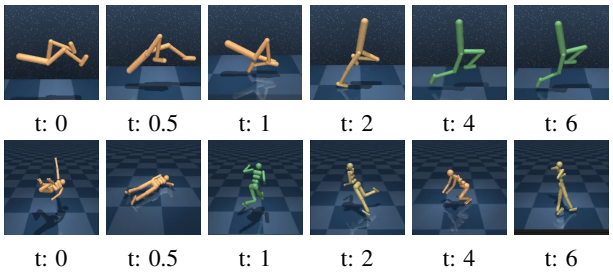


Table 3. First Row illustrates Walker Run on SAC State Derived Features Pretrained on Stand. Second Row illustrates Humanoid Stand Trained on SAC State Derived Features

In the first row, we can see how fast the walker run achieves the near perfect reward state in less than 4 seconds of the episode. We notice how at time 0 when the walker is falling, the walker knows it has to run and it moves its posture at $t = 0.5$ in a way that it can achieve the near perfect state faster.

In the second row, we notice how unstable the humanoid is. The agent finds it very difficult to keep its balance. Its interesting to see that at t=4, the humanoid uses its hand to get up just like humans do.

5. CONCLUSION AND POSSIBLE EXTENSIONS

In this project, we make a comparison of how different task adaptability is for RL agents as compared to that of humans. As mentioned earlier, we observe that network stores all the input based encoded features at the starting layers and adapts to the task by tuning the final layers. Hence, its the quality of training and not the task relatability which is important. An imperfectly trained model results in defects for pretraining experiments. Furthermore, we were able to compare the performances over state based features as well as raw pixel inputs.

When we have brought in the question of pretraining, it will be curious to see how self supervised pretraining like in Unsupervised Reinforcement Learning benchmark [21] shall work in comparison and in combination with our current work.

One major drawback we have noticed in the work is the incapability of RL to work well with images for humanoids. Hence, it is worth noting that the camera view for some of the task domains like humanoid are not well suited to a pixel-only solution for the task. Thus, some of the failure cases are likely due to the difficulty of positioning a camera that simultaneously captures both the navigation targets as well as the details of the agents body. A possible improvement to this can be made by using 3D Reconstruction in the form of Occupancy Networks [22] which are the latest and the most efficient way of representing 3D scenes. On similar lines, DexNerf [23] used NERF, an extension of representing 3D Scenes in function space to help robots grasp transparent objects.

Majority of the works rely on domain adaptation with use cases in simulation to real environments [24]. However, very few works focus on task adaptation and when one works with improving task adaptation, Meta Reinforcement Learning [25] comes into picture. This can be used to improve task adaptability or even create agents which are able to do multiple tasks.

6. REFERENCES

[1] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller, "Deepmind control suite," 2018.

[2] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel, "Trust region policy optimization," 2017.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," 2017.

[4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016.

[5] Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2009, NIPS'09, p. 1204–1212, Curran Associates Inc.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[7] Sascha Lange and Martin A. Riedmiller, "Deep auto-encoder neural networks in reinforcement learning," *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2010.

[8] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel, "Deep spatial autoencoders for visuomotor learning," 2016.

[9] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus, "Improving sample efficiency in model-free reinforcement learning from images," 2020.

[10] Ilya Kostrikov, Denis Yarats, and Rob Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," 2021.

[11] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas, "Reinforcement learning with augmented data," 2020.

[12] Nicklas Hansen, Hao Su, and Xiaolong Wang, "Stabilizing deep q-learning with convnets and vision transformers under data augmentation," 2021.

[13] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa, "dm control: Software and tasks for continuous control," *Software Impacts*, vol. 6, pp. 100022, Nov 2020.

[14] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine, "Soft actor-critic algorithms and applications," 2019.

[15] Aravind Srinivas, Michael Laskin, and Pieter Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," 2020.

[16] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas, "Reinforcement learning with augmented data," 2020.

[17] Ilya Kostrikov, Denis Yarats, and Rob Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," 2021.

[18] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang, "Self-supervised policy adaptation during deployment," 2021.

[19] Nicklas Hansen and Xiaolong Wang, "Generalization in reinforcement learning by soft data augmentation," 2021.

[20] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee, "Network randomization: A simple technique for generalization in deep reinforcement learning," 2020.

[21] Denis Yarats, "The unsupervised reinforcement learning benchmark (urlb)," `https://github.com/rll-research/url_benchmark`, 2021.

[22] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger, "Occupancy networks: Learning 3d reconstruction in function space," 2019.

[23] Jeffrey Ichnowski, Yahav Avigal, Justin Kerr, and Ken Goldberg, "Dex-nerf: Using a neural radiance field to grasp transparent objects," 2021.

[24] Andras Kalapos, Csaba Gor, Robert Moni, and Istvan Harmati, "Sim-to-real reinforcement learning applied to end-to-end vehicle control," *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, Oct 2020.

[25] Riccardo Poiani, Andrea Tirinzoni, and Marcello Restelli, "Meta-reinforcement learning by tracking task non-stationarity," 2021.

[26] Hado van Hasselt, Arthur Guez, and David Silver, "Deep reinforcement learning with double q-learning," 2015.

7. APPENDIX

7.1. Network details and Code Parameters

An image observation is represented as a stack of three consecutive 84×84 RGB renderings to infer temporal statistics, such as velocity and acceleration. For simplicity, we keep the hyperparameters fixed across all the tasks. We evaluate an agent after every 10K training observations, by computing an average return over 10 episodes. We employ double Q-learning [26] for the critic, where each Q-function is parametrized as a 3-layer MLP with ReLU activations after each layer except of the last. The actor is also a 3-layer MLP with ReLUs that outputs mean and covariance for the diagonal Gaussian that represents the policy. The hidden dimension is set to 1024 for both the critic and actor. We employ an almost identical encoder architecture as in [1], with two minor differences. Firstly, we add two more convolutional layers to the convnet trunk. Secondly, we use ReLU activations after each conv layer, instead of ELU. We employ kernels of size 3 × 3 with 32 channels for all the conv layers and set stride to 1 everywhere, except of the first conv layer, which has stride 2. We then take the output of the convnet and feed it into a single fully-connected layer normalized by LayerNorm. Finally, we add tanh nonlinearity to the 50 dimensional output of the fully-connected layer. The actor and critic networks both have separate encoders, although we share the weights of the conv layers between them. Furthermore, only the critic optimizer is allowed to update these weights (e.g. we truncate the gradients from the actor before they propagate to the shared conv layers). The decoder consists of one fully-connected layer that is then followed by four deconv layers. We use ReLU activations after each layer, except the final deconv layer that produces pixels representation. Each deconv layer has kernels of size 3 × 3 with 32 channels and stride 1, except of the last layer, where stride is 2. We then combine the critic's encoder together with the decoder specified above into an autoencoder. Note, because we share conv weights between the critic's and actor's encoders, the conv layers of the actor's encoder will be also affected by reconstruction signal from the autoencoder. We first collect 1000 seed observations using a random policy. We then collect training observations by sampling actions from the current policy. We perform one training update every time we receive a new observation. We evaluate our agent after every 10000 environment steps by computing an average episode return over 10 evaluation episodes. Instead of sampling from the Gaussian policy we take its mean during evaluation. We preserve this setup throughout all the experiments in the paper. We initialize the weight matrix of fully-connected layers with the orthogonal initialization and set the bias to be zero. For convolutional and deconvolutional layers we use delta-orthogonal initialization. We construct an observational input as an 3-stack of consecutive frames, where each frame is a RGB rendering of size 84 × 84 from the 1st camera. We then divide each pixel by 255 to scale it

down to [0, 1) range. Other parameters are summarized in this table:

Parameter name	Value
Replay buffer capacity	1000000
Batch size	128
Discount γ	0.99
Optimizer	Adam
Critic learning rate	10^{-3}
Critic target update frequency	2
Critic Q-function soft-update rate τ_Q	0.01
Critic encoder soft-update rate τ_{enc}	0.05
Actor learning rate	10^{-3}
Actor update frequency	2
Actor log stddev bounds	[-10,2]
Autoencoder learning rate	10^{-3}
Temperature learning rate	10^{-4}
Temperature Adam's β_1	0.5
Init temperature	0.1

Table 4. Parameter Table