10-703 Project Final Report

Joseph Kim
Carnegie Mellon University
Pittsburgh, PA 15213
josephkim@cmu.edu

Kevin Chon
Carnegie Mellon University
Pittsburgh, PA 15213
khchon@andrew.cmu.edu

1 Introduction

Reinforcement Learning (RL) is a fast-growing field which has accomplished much in recent years. However, many of the lauded algorithms and applications make serious assumptions about the nature of the environment. One of these assumptions is the need for a discrete and tractable action space [1,2,3]. Deep learning algorithms in general also require massive compute power and time. In this project, we investigate the challenges which accompany deep reinforcement learning in a continuous action space, as well as issues which pertain to RL in general such as speed of training. The fundamental problem that we are trying to solve is to train an agent to pick up (or, more accurately, get a stable grip on) an arbitrary object via a robotic arm.

Each state is comprised of both a feature vector of observations and a 48x48x3 image which displays the robotic arm's position. In this paper we present a variety of techniques to increase speed of training, reduce computational complexity, increase accuracy, and help deal with the continuous action space. As detailed in the **Background** section, our initial experiments revolved around using the feature vector only as input to our models. This paper explores other options for inputs including random features, as discussed in [5], raw pixels, and a pre-trained image classification network. We also investigate the effects of using a model trained on an easier environment as *input* to other more robust models.

We found success with Proximal Policy Optimization (PPO), introduced in [7], and with variants of the Deep Q-Network (DQN) [1,2,3]. Also included in our experiments were Deep Deterministic Policy Gradients (DDPG), as in [6], as well as a variety of input structures and formats, as in [5]. The specific implementation details are included in the **Methods** section.

2 Background

As a simple baseline, we implemented a Dueling Double Deep Q-Network (Dueling DDQN). The model consists of two deep neural networks with the following architecture: 2 shared hidden layers, with 16 hidden units each, all using the Rectified Linear Unit (ReLU) activation function. To complete the Dueling architecture, the shared layers are succeeded by an 8 unit, 1 hidden layer value network, with a 1 linear output, and an 8 unit, 1 hidden layer advantage network which ends with $|\mathcal{A}|$ linear units. ($|\mathcal{A}|$ is the dimension of the action space) Additionally, we use the concept of a *target network*, described in [1], to stabilize training.

We chose a relatively small network (few hidden units per layer) due to the work done in [4], which states that, empirically, a modestly deep network (2-4 hidden layers) is able to achieve impressive performance with few units. The paper discovered that performance does not scale extremely well with additional network width. Because expanding the network width significantly increases training time, we chose to implement a small network instead in order to facilitate experimentation.

In this baseline attempt we use solely the feature vector, $\vec{v} \in \mathbb{R}^{18}$, as input, and use a handicapped version of the environment in which the agent must select an action from 7 choices, rather than move in a continuous space. Additionally, we utilize the 'height hack' which forces the robot to move

Preprint. Work in progress.

vertically downwards at each timestep, limiting the episode length to 7 or 8. Under this framework, we achieve around 30% accuracy on this environment after 50,000 training episodes, with little to no hyperparameter tuning or feature engineering.

3 Related Work

Research to improve robotic control with deep reinforcement learning approaches varies with different types of robotic tasks, actions, observations, rewards, and goal state(s). Such tasks involving a robotic arm include pushing, sliding, and pick & place. In [9], there are several tasks that the robotic arm must complete in order to reach a multi-dimensional goal state and 4-dimensional, continuous action space (3-dimensions for the Cartesian coordinates of the gripper and opening/closing control of the gripper). Four different variants of DDPG are analyzed based on the inclusion/exclusion of Hindsight Experience Replay along with sparse/dense rewards. In [5], there is an emphasis upon exploration of intrinsic rewards to agent, as opposed to more commonly observed extrinsic rewards carefully set by its environment. In addition, it is noted how the utilization of different feature spaces, including that of random features, can be effective in modeling curiosity in agents driven purely by intrinsic rewards. Also seen in [5], random features can also work surprisingly well and efficiently in several different environments, and in some cases even outperforming curious agents.

Furthermore, much of current research on robotic grasping is often broken down to sensing, planning, and acting, which is non-dynamic and very specific to the environment first perceived. In contrast to these static learning behaviors, there is also work expanding upon closed-loop vision based-control to continuously update a robotic arm's grasp strategy dynamically based off recent observations [10]. This vision-based reinforcement learning framework trains a deep neural network Q-function with only RGB vision based-perception. Such an off-policy algorithm has shown promise for generalizing well in a realistic manner, and in this case grasp many other unseen objects in which static learning methods would be unable to.

4 Methods

Initial Experiments The first step in going beyond the baseline method detailed in the **Background** section is to focus on transitioning to a continuous action space. This requires a substantial paradigm shift seeing as all the models used in the baseline are strictly meant for discrete action spaces. As a second baseline for *continuous* space, we trained an agent for 50,000 episodes via DDPG using only the feature vector as input. This yielded fairly poor results, achieving only 10-15% accuracy in the environment, dependant on random initialization.

The main issue with operating within a continuous action space is that it becomes impossible to maximize the Q-values like a DQN might [6]. Because there are an infinite number of possible actions, it is obviously unfeasible to map an output value to each one like is done in discrete cases. But what if there were a way to 'guide' the model in the right direction? If it were possible to partition the action space into several subsets, this would essentially restrict the values we allow the agent to choose and theoretically make this an easier problem. We took our pretrained Dueling DDQN from the baseline, predicted on the feature vector, and took that prediction as well as the feature vector and used both (concatenated) as *input* to a DDPG model. This increased both performance and speed of training, with a resulting accuracy of between 25% to 30%, varying by random initialization and by hyperparameters. We expect that the performance of a model trained in such a way should be limited by the performance of the pretrained input model, and experimentally this hypothesis is confirmed by the fact that we struggled to beat our discrete baseline score of 30% accuracy. See the **Results** section for more information.

We also ran into issues with lack of time and computational power, partially because DDPG has relatively many hyperparameters and this makes it unfeasible to do much external tuning. The large parameter space also tends to move this project closer to a parameter engineering task rather than a space to implement novel ideas, which we sought to avoid. Thus, we proceeded to experiment with other models such as PPO, another influential model which has been adopted for many applications in recent years [7].

Training only on the feature vector $\vec{v} \in \mathbb{R}^{18}$, with little hyperparameter tuning, we are able to achieve around 60% accuracy in the environment with PPO. This is a *vast* improvement over results observed

both with DDPG and even with Dueling DDQN in the discrete environment. Our hypothesis is that PPO has been known to work well with little or no hyperparameter tuning, while other models may be more sensitive [5]. Since this produced such surprisingly good results, we proceed to investigate the merits of various types of input data.

Including the Image Representation Up until now, every model or approach which has been mentioned in this paper has completely ignored the 48x48x3 image which is also available for observation at each state. It stands to reason that there may be additional information held within said image which is not contained in the feature vector. Work has been done to use not only raw pixels as input, but also extracted features via a variational auto-encoder (VAE) and truly random features from a fixed yet randomly initialized convolutional neural network (CNN) [5]. We study more thoroughly raw pixels and random features in models trained via PPO.

As an initial experiment, we trained a PPO model using *just* the 48x48x3 RGB image as input. This, unsurprisingly, produced poor results and we were unable to observe accuracy above 10% even after many training episodes. This is understandable given that deep RL has always struggled with raw images as input [2,5,7]. We found a surprising amount of success with using random features, concatenated with the original feature vector, to train the agent. A relatively small convolutional neural network is constructed via Keras, with weights initialized via the Glorot Uniform distribution, as per Keras documentation. This is essentially a uniform distribution with parameters specified by the size of the input and output dimensions of the layers to either side of the current one. More information is available here. These weights are held *fixed* for the entirety of training, and images are fed through the network to an output vector $\vec{u} \in \mathbb{R}^{10}$. This ensures the *stability* and *compactness* requirements from [5], as well as possibly the *sufficiency* condition. These random features, combined with the feature vector, were able to achieve $\geq 70\%$ accuracy when trained with PPO and no altered hyperparameters. Not only did we observe an increase in accuracy, but the model appeared to also train quicker than when omitting random features. See the graphs in the **Results** section for more information.

Another idea we had was to use a pretrained image classification model such as MobileNet-v2, as in [8] to process the image first and then use either that output or an intermediate layer as input to a Deep RL model. MobileNet-v2 was selected due to its light weight and high accuracy. This approach has been used in transfer learning tasks pertaining to computer vision, but in our limited time to research it had not yet been implemented with respect to RL. We believed that this approach might show promise as the intermediate layers of a trained network might expose the latent structure of an image much as an Auto-Encoder would, which was mentioned in [5]. Unfortunately, due to pretrained models accepting images of certain sizes only, we found that the training proccess was prohibitively slow. First, we needed to specify to the environment to produce images with 224x224x3 resolution in accordance with MobileNet-v2 [8]. Next, we needed to pass that image though the pretrained CNN and receive a vector $\vec{w} \in \mathbb{R}^{1000}$. This was concatenated to the original feature vector and passed into PPO for training. Because of all these increases in computational requirements, the training process became far too slow to be reasonable for a project of this scale (1 episode took ~ 5 minutes on a relatively new CPU). Maybe in a faster environment would this have been helpful but the cost of training ended up being *mostly* the environment when using larger images, so we abandoned that approach from fear of having to train for weeks on end.

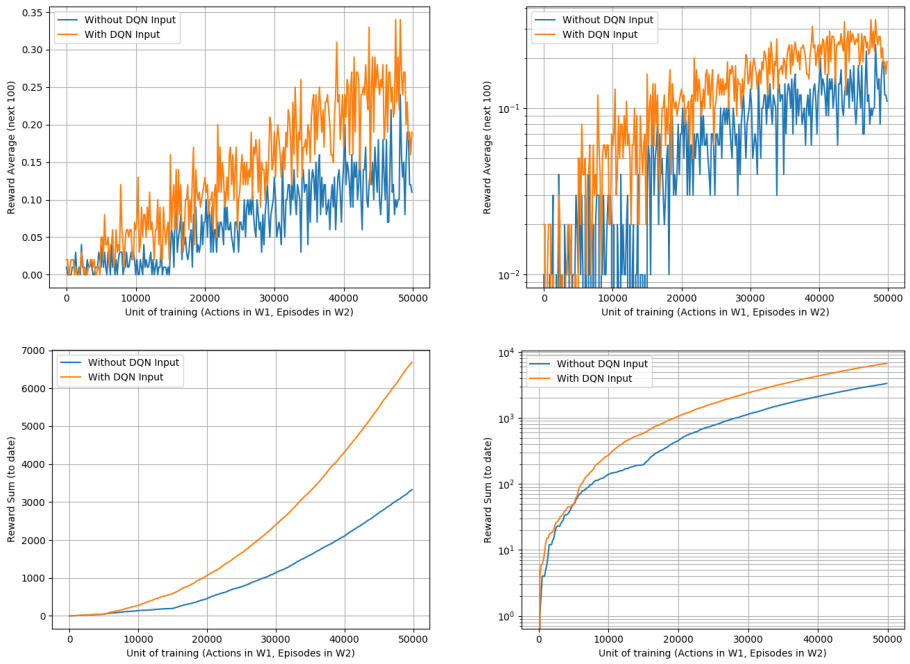
Model Implementation Details Our most successful model turned out to be PPO trained on a concatenation of the feature vector and random features from a fixed CNN. The PPO portion was implemented using OpenAI Baselines. Several environment details had to modified or specified to ensure the baselines would run with the custom environment, such as input dimension and observation space, which was taken from a grandparent class of the environment, and editing the encoding of the object (see below). The model utilized $\gamma = 0.99$, a learning rate of 3×10^{-4}, a value function coefficient of 0.5, $\lambda = 0.95$, and several other (default) parameters. It was trained with the Adam optimizer with specified learning rate and $\epsilon = 1 \times 10^{-5}$. Both the value and policy networks were multi-layer perceptrons with 2 hidden layers of 64 units each, with tanh activation functions. The CNN which provided the random input features was constructed with one convolutional layer with 20 filters, each with kernel size of 5x5, no padding, and a stride length of 1. Next is a max-pooling layer with pooling size 2x2 and equivalent stride length. This was followed by a fully connected layer with 100 hidden units which was subsequently mapped to a fully connected softmax layer with 10 output units. All activation functions were sigmoid, except the last which was softmax. The last

dimension of the feature vector describes an object index, so this was one-hot-encoded to create the feature vector of 18 dimensions. The 10 random features were concatenated to the feature vector to produce the final input vector of 28 dimensions.

The DDPG was also implemented using OpenAI Baselines. It used relatively small 3 hidden layer actor and critic networks and was trained via the Adam optimizer with learning rate 0.0001. We also utilized the Ornstein Uhlenbeck Process, a mean-reverting stochastic process which was used to perturb the action predictions in order to help deal with the explore-exploit tradeoff. This was initialized with $\mu = 0, \theta = 0.15, \sigma = 0.3$. All layers of both the actor and critic networks utilize the Rectified Linear Unit (ReLU) activation functions. Additionally, we used a sequential memory with a maximum size of 10,000 transitions. This was trained for 50000 episodes, both with and without the Dueling DDQN input. The details of the Dueling DDQN are descibed in the **Background** section.

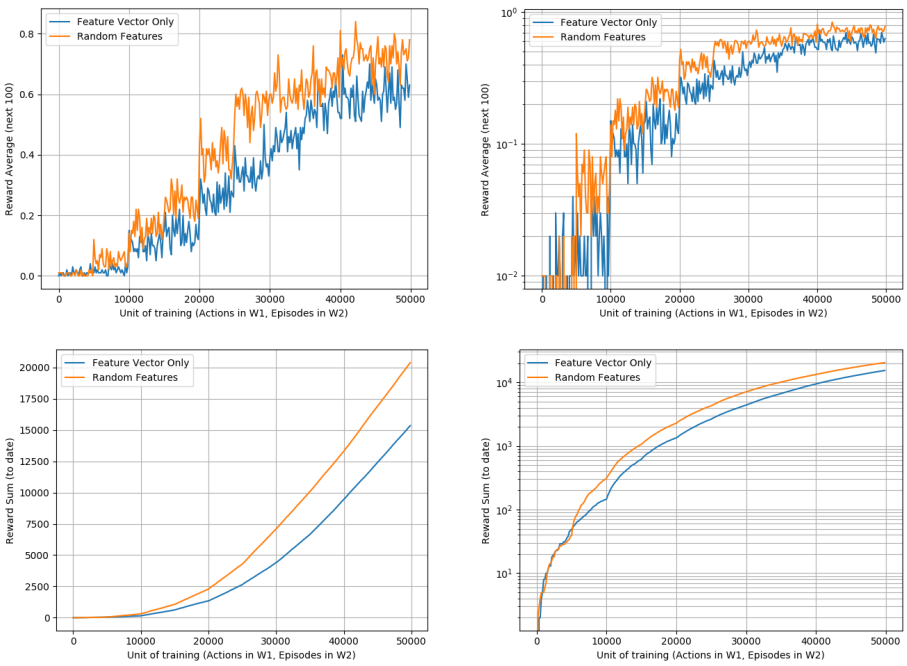
5 Results

Below are our experimental results for DDPG trained on *only* the feature vector and then again on the feature vector and predictions from a pretrained discrete DQN model:



From the above charts we can see that there exists a substantial difference between DDPG trained on only the feature vector, and DDPG trained on the feature vector with input from a trained Dueling DDQN as well. The additional input appears to help the model throughout training, and it ends up achieving $\sim 15\%$ better accuracy on the environment.

Here are our experimental results for PPO trained on just the feature vector and again on the feature vector & random features:



There is also substantial difference between PPO trained with and without random features. Not only do random features help PPO achieve $\sim 10\%$ better accuracy in the environment, they also seem to help increase training speed. PPO with random features appears to make improvements earlier on in the training process as well as learn faster. The average sum plot in the bottom left corner of each grid shows that the random features plot has a slightly steeper slope (scale adjusted) which starts increasing earlier. **We ultimately achieved around 70% accuracy in grasping and picking up an arbitrary object, with PPO trained on the feature vector and random image features.** It is unclear as to whether or not further training would have produced better results, but given our computational limitations we saw fit to halt training at 50000 episodes.

6 Discussion and Analysis

As highlighted above, the PPO model trained on the original feature vector works quite well, achieving about 60% accuracy on the environment. This is a large improvement over the previous models tried in this paper such as DDPG and DQN on the discrete environment. It highlights not only that PPO is a much more robust algorithm but also the importance of not being sensitive to hyperparameters [5]. Given our limited time to train and test models, and the cost of training said models (~ 16 hours for 50000 episodes), it is crucial to use a model which does not require as much parameter tuning. This is one factor which we think contributes to why PPO was so much more successful than other models experimented with. This is not to say that other models such as DDPG cannot outperform PPO; but empirical results from [7] as well as our lack of ability to effectively search the hyperparameter space lead us to believe that PPO is the best algorithm of the ones studied for the current environment.

That random features can have as much of a positive impact on training as they did is something which requires some thought and convincing. The interpretation which we arrived at was this: the random convolution layers can be thought of as a form of smoother over the image (albeit a bad one). Max pooling is designed to take the 'most important' element from that, and the fully connected layers which come after that are random distortions of the existing random features. If thought about in this manner, there is still quite a bit of information being held in the 'random' features, it is just now in a much lower dimension. The information contained in the original image undoubtedly gets quite muddled, but from our experimental results it is a worthy price to pay for the reduction in dimension it provides. Note that it is imperative that the random weights remain *fixed* throughout training - to

change them or even update them via gradient descent would violate the *stability* condition laid out in [5], which at a high level is important to preserving the type of information the model (PPO) expects to receive.

With respect to the DDPG and using a pretrained Dueling DDQN as input to that model, the results make it clear that the new input helps performance by a substantial amount. This makes sense, seeing as every other input is the same, it stands to reason that adding the new input could only help the model train. The fact that this new input is indicating some form of discretized action should in theory provide valuable information to the model, which is why we observe higher rewards and better accuracy in training.

It should be noted that the above results are *not* statistical analyses by any means - just because we observed higher rewards over a few trials does not imply that the different inputs we used caused said higher rewards. This highlights one of the limitations of our approach - the computational requirements of both the environment and the training were too prohibitive to promote any robust statistical analysis. Random initialization and stochasticity throughout the process could very easily over-exaggerate the effects of our experiments. Repeatedly re-training the models and collecting more data would help to validate the results included in this paper, but such training time was outside of the scope of this project.

In conclusion, our experiments supported the claim in [5] that PPO is fairly robust against hyperparameter shifts and thus does not require nearly as much tuning as other models might. We experienced this via the vast increase in performance observed after switching to an 'out of the box' implementation of PPO. Further tuning and, most critically, the use of random features, allowed us to achieve our final scores of $\geq 70\%$ accuracy on the environment. We also observe that using a model trained on an easier environment in conjunction with the available state information can also lead to improved results, as shown through our DDPG experiments. Our failures also demonstrated critical themes echoed throughout Deep RL: our lack of ability to learn on raw pixels is indicative of the dimensionality issues which plague machine learning in general. Our failure to find optimal hyperparameters for DDPG due to lack of compute time parallels computation issues which again affect machine learning in general.

Our results support that (1), PPO is a robust algorithm which is equipped to handle a variety of environments, (2), using a model pretrained on on an easier environment is a valid and effective strategy to augment training data, and (3), that random features truly do offer improvements in performance and that the stochasticity is still able to identify features from the original image and pass them on to the model.

Future Work We noticed that while our aforementioned methods do produce measurable results, there are still areas of improvement in terms of increasing accuracy and reducing training time. Future work would dive deeper into specifically targeting the reduction of training time possibly with exploration into a distributed approach involving multiple agents. As a distributed approach may come at the cost of diminishing sample efficiency, other more advanced optimization techniques within the policy gradient could be explored to combat this commonly seen issue. The reusage of weights through transfer learning from another different but similar environment could also be further explored. Finally, this work would ideally be improved to be generalized into completing not just the grasping task, but also several other common robotic tasks.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 2016.

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581, 2015.

[4] Shumeet Baluja. Empirical Explorations in Training Networks with Discrete Activations. arXiv preprint arXiv:1801.05156, 2018.

[5] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. arXiv preprint arXiv:1808.04355, 2018.

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. In ICLR, 2016.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv preprint arXiv:1801.04381

[9] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. arXiv preprint arXiv:1802.09464

[10] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, Sergey Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. arXiv preprint arXiv:1806.10293