INCREASING POLICY NETWORK SIZE DOES NOT GUARANTEE BETTER PERFORMANCE IN DEEP REINFORCEMENT LEARNING

by

Zachery Berg

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Computer Science

West Lafayette, Indiana

May 2022

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Yexiang Xue, Chair

School of Science

Dr. Kamyar Azizzadenesheli

School of Science

Dr. Bruno Ribeiro

School of Science

Approved by:

Dr. Kihong Park

ACKNOWLEDGMENTS

To everyone who has taught me something, I appreciate all of it. Most recently, to my advisor Yexiang and my mentor Masudur. I thank the both of you for teaching me about deep reinforcement learning research essentially from square one.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The capacity of deep reinforcement learning policy networks has been found to affect the performance of trained agents. It has been observed that policy networks with more parameters have better training performance and generalization ability than smaller networks. In this work, we find cases where this does not hold true. We observe unimodal variance in the zero-shot test return of varying width policies, which accompanies a drop in both train and test return. Empirically, we demonstrate mostly monotonically increasing performance or mostly optimal performance as the width of deep policy networks increase, except near the variance mode. Finally, we find a scenario where larger networks have increasing performance up to a point, then decreasing performance. We hypothesize that these observations align with the theory of double descent in supervised learning, although with specific differences.

1. INTRODUCTION

Deep reinforcement learning (RL) has achieved tremendous success in recent times, primarily due to its ability to harness high-capacity neural networks as function approximators. Recent work in RL generalization has adapted the train/test setup from supervised learning, where conventional wisdom states that larger neural networks both train and generalize better than smaller networks. Henderson, Islam, Bachman, *et al.* [1] and Neal and Mitliagkas [2] showed that policy network architectures affect performance of various RL algorithms, while Cobbe, Hesse, Hilton, *et al.* [3] and Song, Jiang, Tu, *et al.* [4] observed that overparameterized policy networks achieve greater train and test performance than underparameterized networks.

Our work connects this observation to the studies of bias and variance in supervised learning. Classical machine learning theory identifies the bias-variance trade-off, where models with few parameters have low variance but high bias and models with many parameters have low bias but high variance [5]. The bias-variance trade-off leads to a single descent, or U-shape, of the empirical risk curve.

Yet it is known that overparameterization improves generalization in deep neural networks, contradicting the classical bias-variance trade-off. Neal, Mittal, Baratin, *et al.* [6] propose that networks with a large number of parameters actually have decreasing variance due to initialization, leading to lower test error. Yang, Yu, You, *et al.* [7] find that the variance is actually unimodal, and the variance peak occurs when the model interpolates the training data, i.e. reaches zero train error. The unimodality of the variance agrees with the historical bias-variance tradeoff, where the U-shaped risk curve is a special case of sufficiently large variance. By increasing the model complexity past the variance peak, the single descent risk curve becomes a "double descent" risk curve, named as such by Belkin, Hsu, Ma, *et al.* [8] and observed by Geiger, Jacot, Spigler, *et al.* [9], Yang, Yu, You, *et al.* [7], and Nakkiran, Kaplun, Bansal, *et al.* [10].

To estimate bias and variance, we train deep RL agents of varying width (number of neurons per hidden layer) on a distribution of contextual environments and obtain their average zero-shot return on a held-out test context set from the same distribution. This work contributes empirical evidence that a phenomenon similar to double descent can occur

in deep RL, manifesting as a "double ascent" of the return curve, with the second ascent occurring past the variance peak. However, we do not observe the variance mode at the location of interpolation, as would be expected by double descent theory, nor do we observe the mode in the same relative location across experiments.

In the first experiment, we observe the variance mode (and second ascent) in moderately performing DQN agents trained on the discrete Cartpole environment with distracting noise added to the input. The existing double descent theory does not explain this phenomenon, as it expects the variance mode at a location of zero train error (high-performing models). The theory also expects that only the test performance should suffer, as the variance is related to overfitting of the model. Yet, we find the variance mode accompanies a small drop in both train and test return. We propose that agents of this particular size are in a transition stage - slightly smaller agents are too small to ignore the noise in the observation while slightly larger agents are large enough to always ignore the noise. Meanwhile, agents that appear to have moderate performance ignore the noise only sometimes.

In the second experiment, we observe the variance mode using PPO agents trained on the Cartpole environment with static distractions that are easier to ignore. The variance peaks (and the second ascent in test return occurs) in a range of policy network sizes that have high train performance, similar to double descent occurring at low train error. Yet, these policies are considerably larger than the first interpolating (achieving maximum train return) policies. We consider that policies experiencing a performance drop may fail to ignore the noise (like the first experiment), or that they are indeed overfitting to the noise but implicit regularization allows for generalization in larger models.

In the third and final experiment, we increase the difficulty by training larger PPO agents on a confounded version of the HalfCheetah environment. We find a policy network size that begins to look like a double ascent, as the median train return increases but the median test return eventually decreases, but the second ascent does not occur and instead increasing the size of the policy networks causes a drop in train return as well as test return. This experiment highlights the need to train very-large agents successfully before they can be exploited for increased performance.

2. RELATED WORK

This chapter covers the background information and related literature required to understand our experiments. We cover reinforcement learning (RL), the RL algorithms and environments used in our experiments, and existing literature regarding overparameterization in deep RL, generalization in deep RL, and the double descent phenomenon.

2.1 Reinforcement Learning

In reinforcement learning, an agent interacts with an environment by taking actions and receives a reward from the environment after every action. The goal of the agent is to learn a policy (selecting an action given an observation of the environment) that maximizes its expected cumulative reward. Typically the agent's interaction with the environment is defined as a *Markov Decision Process (MDP)*, denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, s_0)$. The environment begins (time $t = 0$) at an initial state s_0 and at every timestep t the agent chooses an action a_t from the action space $\mathcal{A}$ and receives a reward r_t while the environment moves to a new state s_{t+1} in the environment's state space $\mathcal{S}$ based on the transition probability $\mathcal{P}(s_{t+1} \mid s_t, a_t)$. That is, the next state in the environment is assumed to only depend on the previous state and the action chosen, and not on any earlier states. The agent learns a policy $\boldsymbol{\pi}$ by interacting with the environment. To maximize expected cumulative reward, also referred to as the return, the agent estimates the value function or action-value function (Q-function) given the state of the environment. The value function estimates the value of a state by calculating the expected return starting in state s and always acting according to the same policy $\boldsymbol{\pi}$. The Q-function estimates the value of an action a by calculating the expected return if the agent starts in state s, takes action a, then acts according to the policy $\boldsymbol{\pi}$. Recently, so-called deep RL has made use of a neural network architecture to represent the policy and approximate the value function of a given environment. In Experiment 1 we use the DQN [11] algorithm, which approximates the Q-function so that the policy selects an action based on the highest expected Q-value across the possible actions. Experiments 2 and 3 use the PPO [12] algorithm, which directly represents the policy and estimates the value function. We describe these algorithms in more detail in the next section.

2.2 DQN and PPO (RL Algorithms)

2.2.1 Deep Q-Networks (DQN)

We make use of the DQN [11] algorithm in Experiment 1. DQN uses Q-learning to update its policy, and approximates the value of the Q-function of an environment using a deep neural network, called a Q-network. Every gradient update, the agent samples from a replay buffer of its past experiences and maximizes its expected return, assuming that it will act optimally after the next action. Sampling from the replay buffer reduces the impact of the large variance between sequential samples from the environment. DQN is inherently off-policy, that is, estimating the value of the Q-function using experience gained from a different policy than the current one. DQN also uses a separate network, called a target network, to estimate the optimal return. This target network is simply an older version of the current Q-network. In our experiments we vary the width (number of neurons in the hidden layer) of a single-layer fully-connected Q-network.

2.2.2 Proximal Policy Optimization (PPO)

In Experiments 2 and 3, we train agents using the popular PPO [12] algorithm. PPO is comparatively simple to implement, parallelizable, and offers state-of-the-art performance on many tasks. It directly maximizes its estimated advantage (expected cumulative reward gained by updating its policy) subject to a constraint on the size of the policy update. The constraint is upheld either by clipping the policy update to within a factor ϵ of the old policy, also referred to as PPO-Clip, or by adding a penalty on the KL divergence between the old and new policy, called PPO-Penalty. PPO is considered an on-policy algorithm because the agent takes a set of actions, then updates its policy based on its expected return using the current policy. In our experiments we use PPO-Clip, with a clipping value of $\epsilon = 0.3$. We use one neural network with one input, the agent's observation, and two activation outputs - one output determines the action of the agent (the agent's policy), while the other output estimates the value (expected return) given the current observation, which is used to estimate

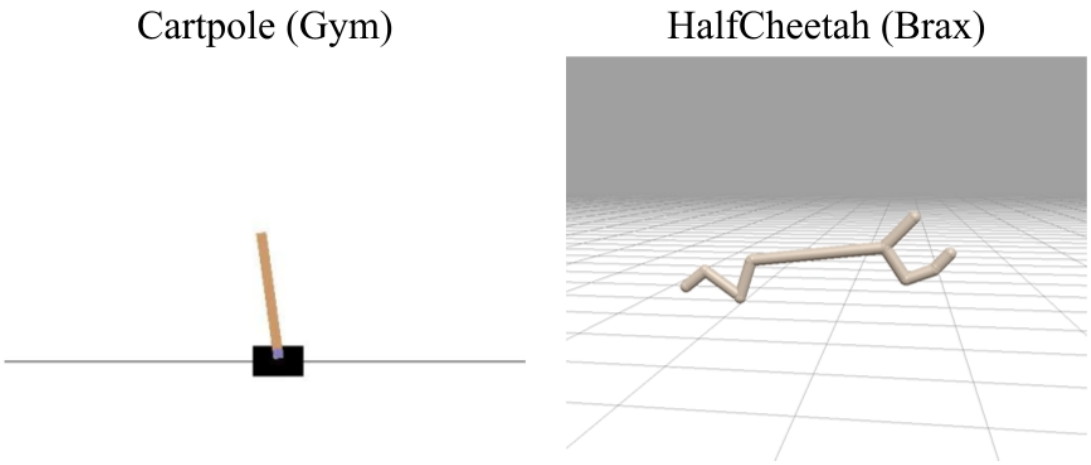

Figure 2.1. Visualizations of the Cartpole and HalfCheetah environments. We add distracting or confounded features to these environments for our experiments.

the advantage. We vary the width (number of hidden neurons) of this neural network in our experiments.

2.3 Cartpole and HalfCheetah (RL Environments)

2.3.1 Cartpole

We train and evaluate agents in the Cartpole environment in Experiments 1 and 2. Cartpole is a classic reinforcement learning environment first proposed in Barto, Sutton, and Anderson [13]. In this environment, a pole is attached upright to a cart that moves horizontally along a track. The environment gives a reward of +1 every timestep that the pole has not fallen over and the cart has not gone off the track. The episode ends when either of these has occurred or after a fixed number of timesteps, the horizon. We use the discrete formulation of this environment, where the action space consists of just two discrete actions: push the cart left or push the cart right with some constant force. A more challenging version of this environment makes the action space continuous, so the agent must also decide how much force should be exerted on the cart. In our experiments, we use a fixed horizon of 200

timesteps, so an optimal policy in this environment achieves a return of 200 every episode. We use the version of Cartpole implemented in OpenAI Gym [14], as seen in Figure 2.1.

2.3.2 HalfCheetah

We use the HalfCheetah environment in Experiment 3. This environment, first described in Wawrzynski [15], tasks the agent with moving in a forward direction by learning to run. The environment gives a positive reward based on the forward velocity of the agent as well as a small negative reward for the amount of force applied to each joint, considered a control cost. The action space is continuous and 6-dimensional, applying force to each of the 6 joints at every timestep. We use a horizon of 1000 timesteps in our experiments. Because the reward is a function of the agent's forward velocity, the cumulative reward achievable by an optimal policy is not obvious to compute. We use the Brax [16] version of HalfCheetah in our experiments, which can be seen in Figure 2.1.

2.4 Deep RL Overparameterization

Until recently, the size of deep RL policies was not given much consideration. Islam, Henderson, Gomrokchi, *et al.* [17] and Henderson, Islam, Bachman, *et al.* [1] highlight that the architectures of policy networks have a direct impact on the performance of deep RL agents. Neal and Mitliagkas [2], Cobbe, Klimov, Hesse, *et al.* [18], and Cobbe, Hesse, Hilton, *et al.* [3] present evidence supporting the theory that larger policy networks have increased performance. Specifically, Neal and Mitliagkas [2] train deep RL policies of varying widths on four environments from OpenAI Gym [14]. They find that overparameterization (extremely wide policies) improves policy performance in three of those environments (with Cartpole being one of them), despite using hyperparameters that were tuned for only one network width. They only consider training performance, and do not study generalization to a test set. Meanwhile, Cobbe, Hesse, Hilton, *et al.* [3] find that increasing the size of the IMPALA [19] architecture, commonly used in deep RL, improves sample efficiency and generalization in benchmark tasks, though with diminishing returns.

Song, Jiang, Tu, *et al.* [4] study deep RL overparameterization in the context of observational overfitting, where the agent overfits to spurious features in its observation, such as the background of an environment, that do not generalize to other settings. They define an (f, g)-scheme for studying observational overfitting, where a function f describes invariant (important) features in the MDP while a function g describes latent (unimportant) features such as the background dependent on a set of parameters θ. A combination function h combines the outputs of the two functions together to produce the agent's observation $\phi_\theta(s) = h(f(s), g_\theta(s))$. Song, Jiang, Tu, *et al.* [4] find that implicit regularization due to overparameterization potentially helps with observational overfitting in linear quadratic regulators (LQR), Gym environments that are projected to a higher-dimensional space, and the CoinRun [18] environment.

Our setup for confounding the HalfCheetah environment is similar to their method of projecting Gym environments to higher-dimensional space, while our overall findings lend support to the overparameterization hypothesis in deep RL with the caveat that increased complexity does not guarantee better performance in every case.

2.5 Deep RL Generalization

Early experiments to test the capabilities of deep RL agents reported scores on the exact same environments on which the agent was trained, colloquially "training on the test set". Zhang, Vinyals, Munos, *et al.* [20] identify the need for distinct training and testing environments in deep RL. Farebrother, Machado, and Bowling [21] use different game modes and difficulties of games in the Arcade Learning Environment as a test set, while Nichol, Pfau, Hesse, *et al.* [22] split different levels from *Sonic the Hedgehog*^{TM} into training and testing levels. Procedural generation has become a popular method for generating unique training and testing environments, both by extending existing environments like Gridworld [20] and GVG-AI [23] and developing new environment testbeds such as Procgen [3]. In a similar vein to Song, Jiang, Tu, *et al.* [4], benchmarks such as Distracting Control Suite [24] have been developed to test the abilities of agents in the context of visual distractions. Kirk, Zhang, Grefenstette, *et al.* [25] define a formalism, based on previous literature, for

the train-test setup used in deep RL, referred to as a Contextual Markov Decision Process (CMDP). We turn readers to Kirk, Zhang, Grefenstette, *et al.* [25] for an in-depth discussion of both RL generalization benchmarks and methods.

2.5.1 Train/Test Setup

We consider the Contextual Markov Decision Process [25]–[28] (CMDP) in our experiments. Using the formalism and similar notation to Kirk, Zhang, Grefenstette, *et al.* [25], in a CMDP $\mathcal{M}_c$ each state $s \in S$ can be decomposed into the underlying state s' and a context c from a context set C. The agent is trained on a set $\mathcal{M}_{C_{train}}$ of i.i.d. "train levels" and evaluated either on a held-out set $\mathcal{M}_{C_{test}}$ or the full set $\mathcal{M}_C$. C_{train} and C_{test} are disjoint subsets of the context set C, unless we choose to use the full context set C for testing. Formally, the context is sampled from a uniform probability distribution over all possible contexts; $c \sim p(C)$. During training, the context is sampled from a uniform distribution over just the training context set; $c \sim p(C_{train})$. We assume C_{train} is generated from a distribution $p(C')$ that generates sets of i.i.d. contexts. The context changes every episode.

In the general case a context may determine the reward function, transition function, initial state distribution, and emission function. For our experiments the context *only* affects the agent's observation, as the context appends either distracting or confounded features to the true state of the underlying MDP. An agent that can distinguish the true features from the context will perform equally well on any level of the CMDP. Figure 2.2 shows a visual overview of our train/test setup. In Experiment 2, the context adds distracting features that are unchanging throughout the episode. In the case of Experiments 1 and 3, the context adds dynamically changing distracting features. This is further explained in Section 3.2.

2.6 Double Descent

The term "double descent" was first coined in Belkin, Hsu, Ma, *et al.* [8], where they observed a decrease, increase, then another decrease in the test risk with increasing model complexity of various neural networks as well as random forests. The increase in empirical risk occurs near the interpolation threshold, when the model is just large enough to be

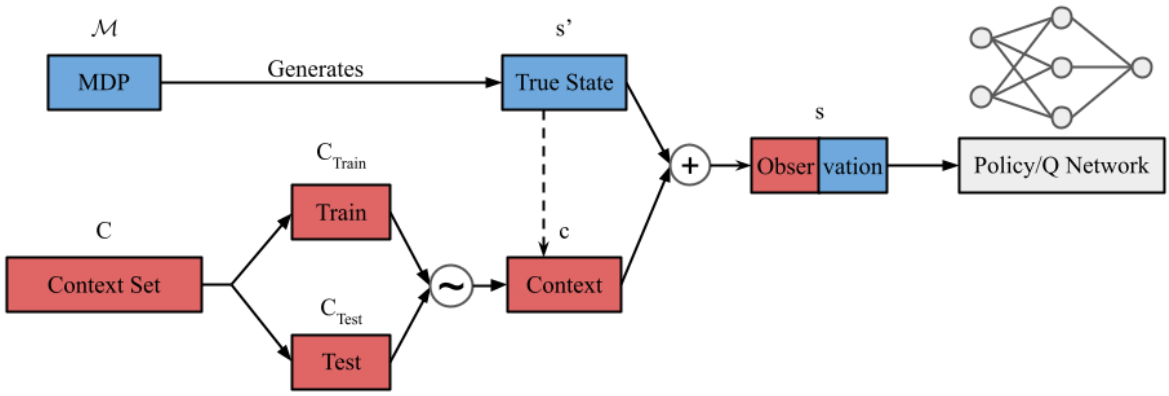

Figure 2.2. Our train/test setup using Contextual MDPs. + represents concatenation of the true state and the context, while $\sim$ represents choosing a specific context from the train or test context distribution. The dotted line indicates that the context may be confounded from the true state.

able to reach zero train error. At this point, there are a small number of solutions that can interpolate the training data. Most of these solutions overfit to noise in the training data, leading to an increase in test error. Past the interpolation threshold, more complex models find low-norm (simpler) solutions that are more generalizable to unseen samples. Double descent has been observed in other deep neural networks, including convolutional neural networks (CNNs) such as Resnet and VGG [7], [9], [10] as well as transformers [10]. Nakkiran, Kaplun, Bansal, *et al.* [10] also observe double descent across epochs and non-monotonicity of test loss across the number of training samples, adding further parameters to the notion of model complexity.

Neal, Mittal, Baratin, *et al.* [6] find that variance decreases with width and increases with depth, supporting the observation that overparameterization benefits neural networks as noted by Neyshabur, Tomioka, and Srebro [29] and Zhang, Bengio, Hardt, *et al.* [30] and corroborated by many others. Yang, Yu, You, *et al.* [7] explain the double descent phenomenon by showing that increasingly wide neural networks have monotonically decreasing bias and unimodal variance, with the variance mode occurring directly before the interpolation threshold. Since test risk is the sum of bias and variance, double descent in the overall test risk may or may not occur at the variance's mode depending on the scale of the decreasing bias. Multiple works have further decomposed the variance into variance due to

initialization of parameters, variance due to the training data samples, and variance due to label noise. In classical ML, ensembling and bagging are used to reduce the variance due to initialization and samples respectively. Increasing the label noise has been empirically found to increase the effect of double descent [10], and Lin and Dobriban [31] show that this is due to the interaction of label noise with samples and initialization. Lin and Dobriban [31] and Adlam and Pennington [32] prove in similar settings that the variance at the interpolation threshold is dominated by the interaction between the samples and initialization.

To our knowledge, we are the first to consider the theory of double descent in deep reinforcement learning agents. We find that a similar phenomenon can occur; an increase, decrease, and then increase of the average return, but we do not find that the decrease in performance occurs at the exact interpolation threshold. For this reason, as well as differences in experimental setup, we cannot fully conclude that the phenomenon of double descent is occurring.

3. METHODOLOGY

3.1 Calculating Bias and Variance

Yang, Yu, You, *et al.* [7] provide a method for estimating the random-design bias and variance of a neural network learning a classification task, which we adapt to reinforcement learning policy networks. We assume the agent is tasked with maximizing its return on a CMDP $\mathcal{M}_C$, which is a single MDP with a context set C. The agent is trained on a CMDP $\mathcal{M}_{C_{train}}$, where C_{train} is a subset of the full context set C. We assume that the set C_{train} is generated from a distribution $p(C')$, which generates sets of i.i.d. contexts while a single context c is generated from a uniform distribution denoted by $p(C)$ over the full context set C. The formalism for CMDPs and train/test sets is also described in Section 2.5.1.

We define $\mathcal{R}(\pi, \mathcal{M}_c)$ as the return of a policy π on a particular level $\mathcal{M}_c$ of the CMDP. Our definition of risk is the expected squared difference between the return achievable by an optimal policy π^* and the return achieved by a policy trained on the training context set $\pi_{C_{train}}$. The risk definition decomposes into squared bias and variance as follows:

$$\begin{aligned}
&\mathbb{E}_{c\sim p(C)}\mathbb{E}_{C_{train}\sim p(C')}[\|\mathcal{R}(\pi^*, \mathcal{M}_c) - \mathcal{R}(\pi_{C_{train}}, \mathcal{M}_c)\|_2^2] && \text{Risk} \\
&= \mathbb{E}_{c\sim p(C)}[\|\mathcal{R}(\pi^*, \mathcal{M}_c) - \mathbb{E}_{C_{train}\sim p(C')}[\mathcal{R}(\pi_{C_{train}}, \mathcal{M}_c)]\|_2^2] && \text{Bias}^2 \\
&+\mathbb{E}_{c\sim p(C)}\mathbb{E}_{C_{train}\sim p(C')}[\|\mathcal{R}(\pi_{C_{train}}, \mathcal{M}_c) - \mathbb{E}_{C_{train}\sim p(C')}[\mathcal{R}(\pi_{C_{train}}, \mathcal{M}_c)]\|_2^2] && \text{Variance}
\end{aligned} \tag{3.1}$$

We can estimate the variance using $N > 1$ disjoint sets of training contexts $C_{train} = C_1 \cup C_2 \cup ... \cup C_N$ and the unbiased estimator from Yang, Yu, You, *et al.* [7]:

$$\widehat{\text{Var}}(\mathcal{M}_c, C_{train}) = \frac{1}{N-1}\sum_{\text{j}=1}^{N}\|\mathcal{R}(\pi_{C_\text{j}}, \mathcal{M}_c) - \sum_{\text{j}=1}^{N}\frac{1}{N}\mathcal{R}(\pi_{C_\text{j}}, \mathcal{M}_c)\|_2^2 \tag{3.2}$$

In our experiments we use $N = 2$. Training agents with different initializations (seeds) increases the accuracy of $\mathcal{R}(\pi_{C_\text{j}}, \mathcal{M}_c)$, the return on level $\mathcal{M}_c$ after training on the training

Table 3.1. Summary of the setup for each experiment. We experience a different shape of the return curve depending on the setup.

Experiment	Algorithm	Environment	# Context Feats	Context Type
1	DQN	Cartpole	50	Dynamic noise
2	PPO	Cartpole	3	Static noise
3	PPO	HalfCheetah	17	Confounded state

set C_j. Finally, we calculate the bias as the difference between the risk and the estimated variance.

This calculation of bias and variance is not the same as other measures of the same name. Typically, bias and variance in RL refers to the agent's value estimate across samples, such as the reason for random sampling from a replay buffer in DQN [11]. This work studies the risk, bias, and variance of the *return* received by an agent.

3.2 Specific Experimental Setups

In section 2.5.1 we described the general train/test setup in deep RL. Combined with the method described in Section 3.1, we can estimate the bias and variance of any deep RL policy on any environment. In this section we detail the setup for our specific experiments, including how we modify existing RL environments. Table 3.1 provides a summary of each setup, including how we add context to existing RL environments.

3.2.1 Experiment 1 Setup

We train DQN [11] agents implemented in RLLib [33] (Ray version 1.3.0) for 5 million timesteps on a contextual Cartpole environment, with a single fully-connected layer for the Q-network. We use the base DQN implementation (SimpleQTrainer) that does not have any architectural optimizations, and vary the size of the network. The Q-networks of varying size all use the same hyperparameters. Relevant hyperparameters are shown in Table 3.2, and they are all default RLlib values.

We add distracting features to the Cartpole environment, and the context c consists of 50 real values sampled from the standard normal distribution $(c_1, c_2, ..., c_{50}) \sim \mathcal{N}(0, 1)$.

Table 3.2. Relevant hyperparameters for Experiment 1 (DQN agents trained on distracting Cartpole). All hyperparameters, including the ones shown, are the default settings.

Description	Variable Name	Value
Number of timesteps to train	timesteps_total	5,000,000
Gamma (reward discount factor)	gamma	0.99
Learning rate	lr	0.0005
Initial exploration rate (else greedy action)	initial_epsilon	1.0
Final exploration rate	final_epsilon	0.02
Timesteps over which to anneal epsilon	epsilon_timesteps	10000
Update target network every X timesteps	target_network_update_freq	500
Timesteps in the replay buffer	buffer_size	50000

This formulation is similar to adding a dynamically changing, distracting background to the environment. We make use of the train/test setup described in Section 2.5.1, with 56 held-out test levels and two disjoint train sets of 50 levels each. An integer for each level serves as a seed for sampling from the distribution, so when an agent revisits a level it will experience the same noise every time. We obtain the agent's average train and (zero-shot) test return by evaluating each agent for 10 episodes in each level, to account for the environment's random initial state. Figure 4.1 shows the median and quartiles of the return over 20 runs per policy width (10 unique seeds $\times$ 2 train sets), as well as the estimated risk, bias, and variance between the two disjoint train sets.

3.2.2 Experiment 2 Setup

For this experiment we train PPO [12] agents implemented in RLLib [33] (Ray version 1.3.0) for 5 million timesteps on a contextual Cartpole environment that is different from Experiment 1. The agent's policy and value networks share a single fully-connected layer, with separately learned transformation outputs. Other than varying the network width, each agent uses the same set of hyperparameters, for which the full hyperparameter settings can be seen in Table 3.3.

We generate the context $c = [R, G, B]$ by taking the level number modulo 8, converting it to a binary value, then multiplying each digit by the level number. For example, level 14 has

Table 3.3. Hyperparameters that we set for Experiment 2 (PPO agents trained on distracting Cartpole). All other hyperparameters are the default settings, except for the varying width.

Description	Variable Name	Value
Number of timesteps to train	timesteps_total	5,000,000
Gamma (reward discount factor)	gamma	0.99
Learning rate	lr	0.0003
Clipping parameter (ϵ)	clip_param	0.3
Normalize the observation	observation_filter	MeanStdFilter
Iterations of SGD per policy update	num_sgd_iter	6
Coefficient of value function loss	vf_loss_coeff	0.01
Linear activations between layers	fcnet_activation	linear
Shared policy and value network	vf_share_layers	True
Use Pytorch (instead of TensorFlow)	framework	torch

context $14 \mod 8 = 6 \rightarrow [1, 1, 0] \rightarrow [14, 14, 0]$. Because the features do not change during an episode, this formulation is similar to adding a static background to the environment. The disjoint train context sets are levels 0-99 and 100-199, and the test context set is levels 200-255. We obtain the agent's train and (zero-shot) test return by evaluating each agent for 50 episodes in each level. The median and quartiles of the return are shown in Figure 4.2 over 40 runs per policy width (20 unique seeds $\times$ 2 train sets), as well as the estimated risk, bias, and variance between the two disjoint train sets. We smooth the risk curve using exponential moving average (smoothing = 0.6).

3.2.3 Experiment 3 Setup

For our third and final experiment, we train PPO agents implemented in Brax [16], which is written in JAX [34] and makes use of TPUs to run faster. We train for 100 million timesteps, using a policy network with 4 layers and a separate value network with 5 layers. Increasing the policy width by 2x results in approximately 16x parameters. We keep the value network at a fixed width of 256 neurons. Both the policy and value networks use the SiLU activation function between layers.

Table 3.4. Hyperparameters that we set for Experiment 3 (PPO agents trained on confounded HalfCheetah). Besides changing the policy network size, all of these values match the tuned hyperparameters provided by the authors of Brax.

Description	Variable Name	Value
Number of timesteps to train	num_timesteps	100,000,000
Learning rate	learning_rate	3e-4
Clipping parameter (ϵ)	ppo_epsilon	0.3
Horizon (number of timesteps per episode)	episode_length	1000
Normalize observations	normalize_observations	True
Number of episodes between policy updates	unroll_length	20
Number of minibatches	num_minibatches	32
Epochs of optimization per policy update	num_update_epochs	8
Discount factor (gamma)	discounting	0.95
Scale of entropy loss	entropy_cost	0.001
Number of parallel environments	num_envs	2048
Batch size	batch_size	512

For this experiment we add confounding features to the HalfCheetah environment. To create the confounding features, we utilize a method similar to the method used by Song, Jiang, Tu, *et al.* [4] to create projected Gym environments, where the true state is passed through a randomly initialized single-layer network. Each level has a unique confounding network, and with a small enough number of training levels the agent overfits to the confounding networks. We use 3 training levels in this experiment. We set the number of confounding features in the context c to be equal to the number of features in the true state s', which is 17 features. The reported test return is the average return over 128 episodes on a single level from the test context set. This single test level is the same across the evaluation episodes, network widths, and seeds. All networks use the same set of hyperparameters, seen in Table 3.4, except for the varying policy network width. The hyperparameters come from the Colab Notebook 'Brax Training with TPU' created by the authors of Brax, while the policy and value networks are the default Brax networks. We report the performance of each agent at its highest train reward during training, not necessarily at 100 million timesteps. Although, in most cases the highest train reward is at 100 million timesteps. The results in

Figure 4.3 show the median and quartiles of the average train and test return over 20 seeds per policy width.

4. RESULTS AND DISCUSSION

In this section we discuss the results of our three experiments, then we discuss the implications of these results. In every experiment, we surprisingly observe a drop in performance as we increase the size of an agent's policy network. Table 4.1 summarizes the shape of the return curve across policy sizes as well as the size of the policy and/or value network when the performance drops.

4.1 (Mostly) Monotonically Increasing Performance (Experiment 1)

In Figure 4.1 we observe mostly monotonically increasing train and test return, as would be expected as policies become larger. However, at a width of 21 neurons both the train and test return drop slightly. This occurs right after the variance peaks, at a width of 20 neurons. The interquartile range across seeds also exhibits more variability in the transition between low-performing and high-performing agents, which highlights the observations by Henderson, Islam, Bachman, *et al.* [1] and Agarwal, Schwarzer, Castro, *et al.* [35] about the performance uncertainty of deep RL agents. Past the performance drop, the return increases again, leading to a "double ascent" of the return curve.

The shape of the curve draws a connection to the theory of double descent from supervised learning, yet in double descent only the test error undergoes two descents as model complexity increases. Meanwhile, the train error should be monotonically decreasing. Thus it is interesting that the drop in performance occurs in both the train and test return. Furthermore, the theory of double descent expects a performance drop in high-performing

Table 4.1. Shape of the return curve and size of the policy and/or value network for each experiment at the performance drop.

Experiment	Return Shape	Policy Width at Drop	Policy Params at Drop
1	Monotonically increasing	21	1200
2	Mostly optimal	50-100	550-1100
3	Bell-shaped	192	119,000

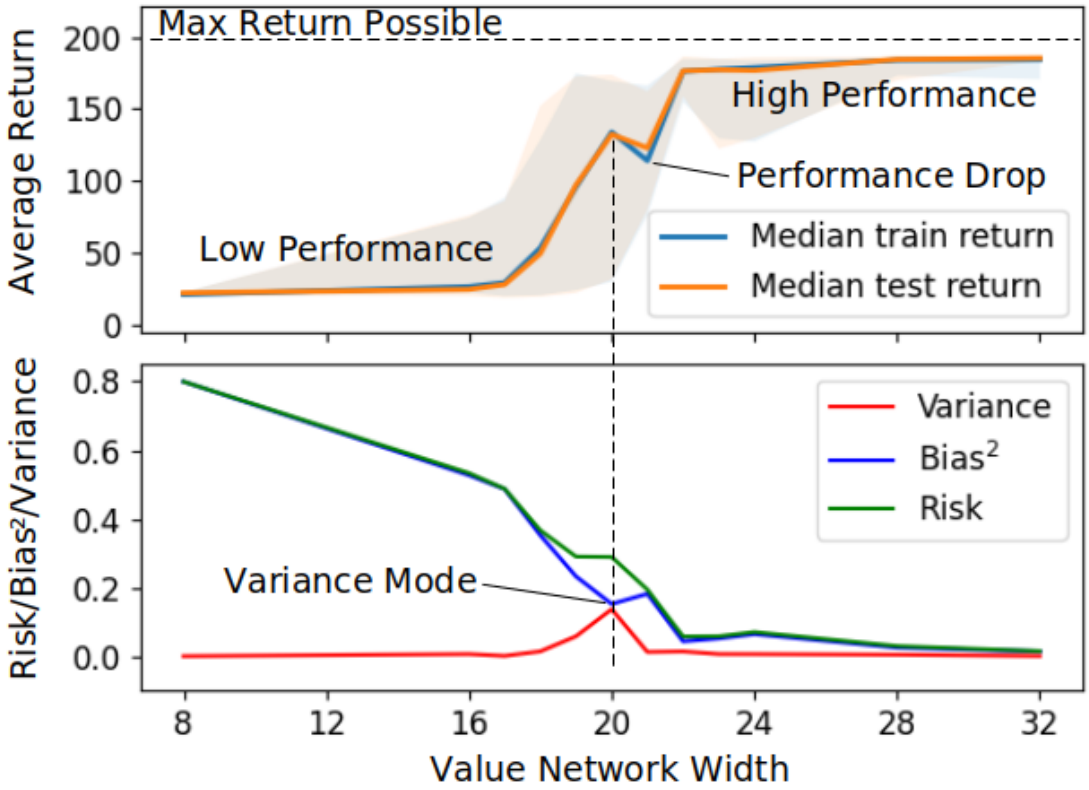

Figure 4.1. Return and risk curves for DQN agents of varying width trained on a distracting Cartpole environment. Average return is monotonically increasing except near the variance mode, where both the train and test return drop. Surprisingly, the drop occurs in moderately-performing agents. The return curve shows the median and interquartile range across 20 runs (10 unique seeds $\times$ 2 train sets).

agents that have close to zero train error. Yet, in our setting the variance peaks and the performance drops in moderately-performing agents.

What could cause this performance drop? Since the train and test return of agents closely matches throughout the varying policy sizes (there is no generalization gap), it suggests that this is an optimization problem rather than a generalization problem related to overfitting. The higher variance at a width of 20 neurons indicates that agents of this size do not find policies that achieve the same return on the same levels. The variance could be a manifestation of the fact that the context appends noise to the agent's observation, and policies of this particular size are only able to ignore the noise in some cases. Perhaps policies that are smaller (17 hidden neurons or less) are consistently unable to learn anything

meaningful because they are too low-capacity to ignore the noise. Meanwhile, policies that are larger (22 hidden neurons or more) can ignore the noise in most cases. In every situation, we would expect policies that can ignore the noise to solve the relatively simple Cartpole problem - for example, policies with just two hidden neurons in Experiment 2 are able to perform well.

The minimal generalization gap could also be a result of the similarity between different contexts in the context set. Both the train and test contexts consist of noise from the standard normal distribution $\mathcal{N}(0, 1)$. A sufficiently large network or sufficiently large amount of training samples (or both) can lead a policy to learn the distribution, resulting in no difference between train and test performance. Test contexts that are meant to be distinct from the train contexts may in fact be so similar that the agent does not notice the difference.

Although the boxes are not checked to call this double descent in the supervised learning sense, the outcome is similar - there is a select range of policies that do not perform as well as expected because they are in a transition stage between too small and large enough. Even though the performance drop appears to be an optimization problem rather than an overfitting problem, it is worth further investigation, perhaps by reproducing the result in other algorithms or environments.

4.2 (Mostly) Optimal Performance (Experiment 2)

In Figure 4.2 we observe mostly optimal policies except for the initial performance jump from 1 to 2 hidden neurons and a range of policy sizes for which the test return decreases, then increases back to optimal. The median across seeds of the average test return drops to slightly below optimal near a width of 50 neurons, accompanied by an increase in the variance of these agents. After 100 neurons, the test return becomes optimal again. For the rest of the policy sizes, performance is close to optimal (even for small policies) because the context of the environment (three static features in each level) is not difficult to ignore. This results in small values for the bias, variance, and risk. It is surprising then that further increasing the complexity of the policy networks does not maintain the same level of performance.

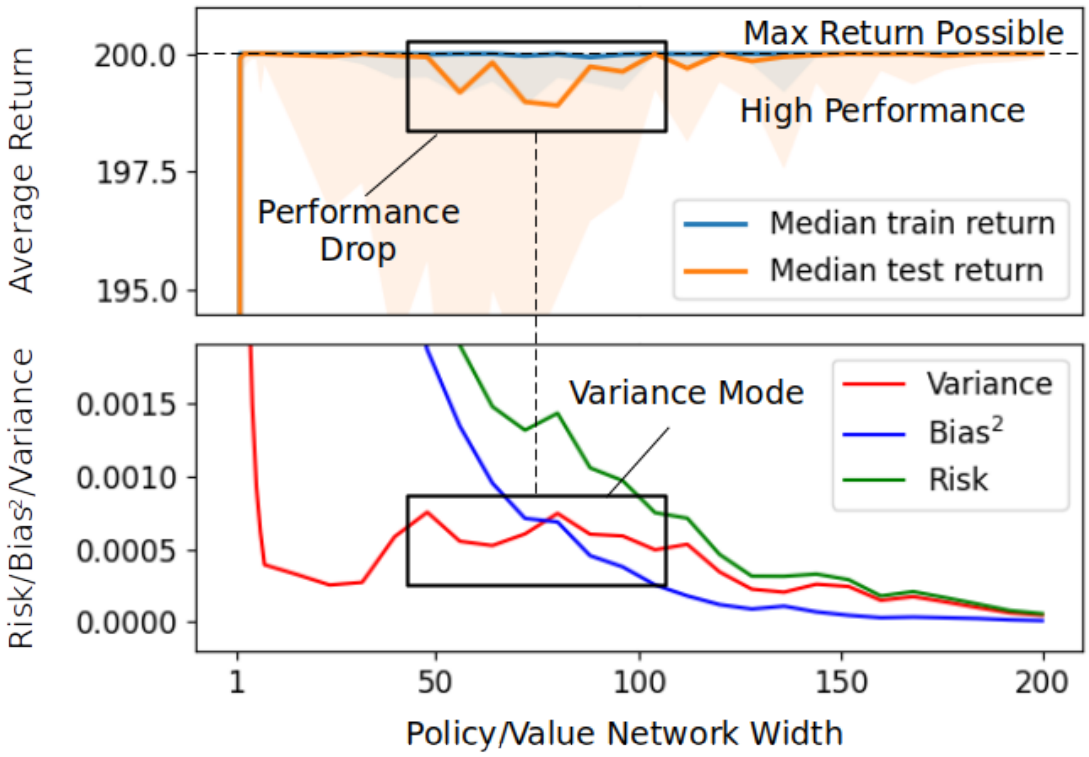

Figure 4.2. Return and risk curves for PPO agents of varying width trained on a distracting Cartpole environment. There is a sharp increase in average return from 1 to 2 hidden neurons, which is stable until the test return decreases around the variance mode, then recovers. The return curve shows the median and interquartile range across 40 runs (20 unique seeds $\times$ 2 train sets).

Unlike the previous experiment, the drop in performance is around a larger range of policy sizes. Because we see a drop in test performance but not a drop in train performance, we can draw a closer relationship to double descent in this case. However, unlike double descent, the drop in test performance does not occur near the interpolation threshold, which we could assume is at 2 hidden neurons. Furthermore, the experimental setup is a source of uncertainty for the cause behind the test performance drop. Because the context appends unimportant features to the observation, an agent that generalizes to a test context may either a) ignore the unimportant features altogether or b) be able to generalize to within-distribution contexts, similar to the previous experiment. Thus, it is not entirely clear if the test performance drop is an inability to ignore the noise or a failure to generalize to the noise.

The first explanation is similar to the explanation for the previous experiment's results. Looking more closely at Figure 4.2, the interquartile range of the seeds (the light orange shading) is larger in policies of less than 100 width. Although the median performance appears to be optimal up to widths of 50, the interquartile range indicates these policies can also fail to generalize. Meanwhile, past a width of 100 both the median seed and interquartile range of seeds is closer to optimal. Policies of width less than 100 may be small enough to occasionally be distracted by the context, despite their apparent interpolation, while wider policies consistently ignore the context.

The second explanation, inability to generalize, more closely aligns with double descent theory. Yet, the aforementioned fact that the performance drop is not at the assumed interpolation threshold hurts this theory. We may instead consider a broader explanation such as implicit regularization. Larger networks (greater than 100 width) could perfectly generalize while smaller networks do not generalize perfectly.

4.3 Bell-Shaped Return Curve (Experiment 3)

In the previous experiment, we found that the test performance of an agent was surprisingly lower in a set of policies that achieved optimal train performance. In these cases, the generalization gap was quite small due to a sufficient number of training levels and a context of unimportant, noisy features. With our confounded HalfCheetah environment, we provide the agent with fewer training levels (only 3) and a confounded version of the true state which can lead the policy to learn spurious associations between the confounded features and reward. Thus, we observe a case where the train and test return are not similar and wider policies do not serve as a replacement for an insufficient number of training contexts to successfully ignore the true context.

Figure 4.3 shows three regions of performance. In the first region (up to a width of 160), the train and test return of agents increases as the size of the policy network increases. This is what past literature has come to expect. In the second region (from widths of 160 to 192), we observe only a minor increase in the median train return, accompanied by a minor decrease in the median test return. In the final region, the train return surprisingly

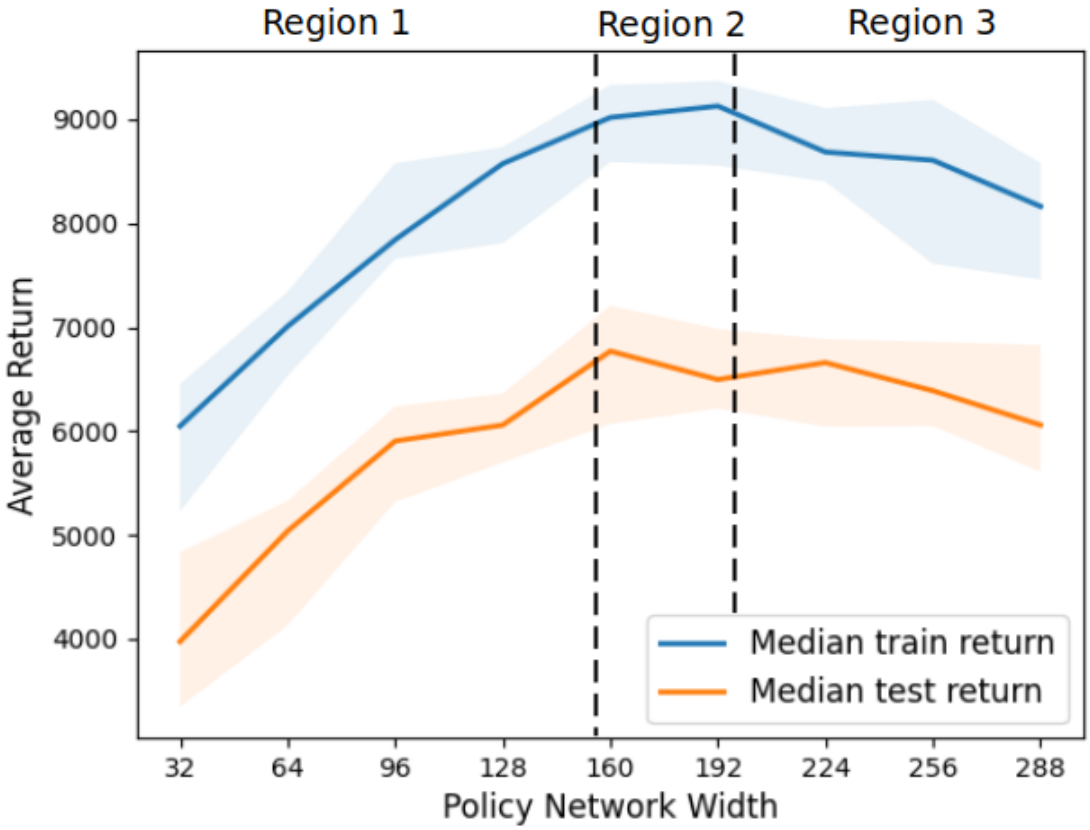

Figure 4.3. Median and interquartile range of return over 20 seeds of PPO agents of varying width trained on a confounded HalfCheetah environment. In Region 1, both the train and test return increase as expected. In Region 2, the test return decreases while the train return continues to increase. In Region 3, the train return also decreases.

begins to decrease, while the test return increases (from widths of 192 to 224) and then also decreases. More specifically, out of the 20 seeds evaluated, we observe in 6 of them that the policy network size that has the highest train performance does not also have the highest test performance. We did not observe a similar phenomenon when the agent was given enough training levels (30) to close the generalization gap.

As with the other two experiments, the three regions appear to show a first ascent, then descent, of the train and test return. However, we do not see the anticipated second ascent. Neal and Mitliagkas [2] observed training instability in very-wide PPO policies on the MountainCar environment and attributed it to a lack of hyperparameter tuning across

the large range of widths. We similarly observed stable training in Regions 1 and 2, after which point only some policies were stable during training. Cobbe, Klimov, Hesse, *et al.* [18] and Cobbe, Hesse, Hilton, *et al.* [3] find that larger agents have no issues while training, but they experience diminishing returns. We observe the same diminishing returns in Region 2 as the train return curve becomes flatter, but exceedingly large agents (Region 3) do not continue this trend. Rather, the agents with stable training in Region 3 do not reach the same maximum return as stable agents in Region 2. Henderson, Romoff, and Pineau [36] observe a similar phenomenon when varying the learning rate of deep RL agents, including PPO agents trained on HalfCheetah and optimized using Adam as done in our experiment. They find that a small range of learning rate values performs the best, with smaller learning rates performing better with Adam optimization. We use a learning rate of 0.0003 in this experiment, which compares to the smallest values tested by Henderson, Romoff, and Pineau [36], but our policy networks are larger than the network they use.

4.4 Summary and Implications of Results

In all three experiments conducted we observe a return curve with a similar shape to "double ascent", but none of them exactly match double descent in supervised learning. Nonetheless, our results indicate that policy size is an important hyperparameter to consider when training deep RL agents. In Experiment 1, we find that policies may not have monotonically increasing performance in the transition from low- to high-performing agents. In Experiment 2, we find that not all high-performing agents of a particular size will generalize well, highlighting the need to ensure train and test stability. In Experiment 3, we find that naively increasing the size of policy networks could lead to decreased performance, though this could be related to a lack of hyperparameter tuning. Current work in RL generalization tends to use the same architecture across experiments, such as IMPALA [19]. While this is beneficial for comparing solutions, we encourage deep RL researchers to confirm (for example, by training across many seeds and/or on disjoint train sets) that their solutions and results are low-variance. If they are not, changing the size of their policy networks could bring the networks to a model complexity of lower variance.

As an aside, the notion of interpolation is closely tied to double descent in supervised learning. However, this concept does not easily translate to reinforcement learning. The discussion of Experiments 1 and 2 considers the interpolation threshold to be when the median train return across seeds is maximal. Yet, the HalfCheetah environment in Experiment 3 and many real-world examples do not have an upper limit on the achievable train return. This makes it difficult or impossible to define a set of policies with zero train error. Although, it does not prevent us from estimating the variance of return of different policy sizes and recovering the variance mode. Even if the model size of highest variance (the expected interpolation location in supervised learning) is not the smallest model size that achieves high train return, the variance mode is still important to consider.

In our experiments, we only consider a select number of RL environments and algorithms. However, we observe both on-policy and off-policy RL algorithms, as well as both discrete- and continuous-action environments. The setup we describe can be used with other RL algorithms and environments, as well as other architectural modifications such as increased depth or different activations between layers. We believe the size of policy networks is an important area for future research, as it is not often considered in deep RL literature.

Future work can explore the effect of other noisy contexts like Experiments 1 and 2 as well as confounding contexts like Experiment 3 on other algorithms or environments. It may be worth considering noisy contexts that are out-of-distribution, such as a different noise distribution for the testing context set of Experiment 1. A more comprehensive set of results on CMDPs that affect more than just the observation, such as the popular Procgen [3] benchmark, should also be worthwhile. Lastly, further consideration of the definitions of interpolation and model complexity of reinforcement learning policies should prove insightful to more rigorously observe double descent in deep reinforcement learning.

5. CONCLUSION

In this work, we observe cases where more complex policy networks do not always translate to greater performance in deep reinforcement learning. We measure the bias and variance of the policy networks representing these agents and recover unimodal variance, as previously observed in supervised learning. Then, we find a case where the performance increases up to a point, then begins to decrease.

Specifically, we observe mostly monotonically increasing performance by training DQN agents on a distracting Cartpole environment, and mostly optimal performance when training PPO agents on an easier distracting Cartpole environment. On a confounded HalfCheetah environment, we observe increasing train performance followed by decreasing train performance in PPO agents. In all three cases, we do not observe strictly monotonically increasing performance as the size of policy networks increases. We present multiple hypotheses for this phenomenon and compare to the phenomenon of double descent in supervised learning. Although our results are similar, we find specific differences from double descent.

There is evidence that the recently-discovered regime of overparameterization in machine learning could apply to reinforcement learning, but it remains to be seen whether the same findings apply in every case. Either way, it is clear that policy size directly impacts the performance of deep RL agents. We hope that future work will help to confirm our preliminary results, which could be extended to more RL algorithms, different environments and contexts, and other architectures.

REFERENCES

[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," *arXiv:1709.06560 [cs, stat]*, Jan. 2019, arXiv: 1709.06560. [Online]. Available: http://arxiv.org/abs/1709.06560.

[2] B. Neal and I. Mitliagkas, "In Support of Over-Parametrization in Deep Reinforcement Learning: An Empirical Study," en, in *International Conference on Machine Learning*, 2019, p. 9.

[3] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging Procedural Generation to Benchmark Reinforcement Learning," *arXiv:1912.01588 [cs, stat]*, Jul. 2020, arXiv: 1912.01588. [Online]. Available: http://arxiv.org/abs/1912.01588.

[4] X. Song, Y. Jiang, S. Tu, Y. Du, and B. Neyshabur, *Observational overfitting in reinforcement learning*, 2019. arXiv: 1912.02975 [cs.LG].

[5] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992. DOI: 10.1162/neco.1992.4.1.1.

[6] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A Modern Take on the Bias-Variance Tradeoff in Neural Networks," *arXiv:1810.08591 [cs, stat]*, Dec. 2019, arXiv: 1810.08591. [Online]. Available: http://arxiv.org/abs/1810.08591.

[7] Z. Yang, Y. Yu, C. You, J. Steinhardt, and Y. Ma, "Rethinking Bias-Variance Trade-off for Generalization of Neural Networks," *arXiv:2002.11328 [cs, stat]*, Dec. 2020, arXiv: 2002.11328. [Online]. Available: http://arxiv.org/abs/2002.11328.

[8] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the bias-variance trade-off," *arXiv:1812.11118 [cs, stat]*, Sep. 2019, arXiv: 1812.11118. [Online]. Available: http://arxiv.org/abs/1812.11118.

[9] M. Geiger, A. Jacot, S. Spigler, F. Gabriel, L. Sagun, S. d'Ascoli, G. Biroli, C. Hongler, and M. Wyart, "Scaling description of generalization with number of parameters in deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2020, no. 2, p. 023 401, Feb. 2020, arXiv: 1901.01608, ISSN: 1742-5468. DOI: 10.1088/1742-5468/ab633c. [Online]. Available: http://arxiv.org/abs/1901.01608.

[10] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep Double Descent: Where Bigger Models and More Data Hurt," *arXiv:1912.02292 [cs, stat]*, Dec. 2019, arXiv: 1912.02292. [Online]. Available: http://arxiv.org/abs/1912.02292.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017, arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[13] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834–846, 1983. DOI: 10.1109/TSMC.1983.6313077.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540 [cs]*, Jun. 2016, arXiv: 1606.01540. [Online]. Available: http://arxiv.org/abs/1606.01540.

[15] P. Wawrzynski, "Learning to control a 6-degree-of-freedom walking robot," in *EUROCON 2007 - The International Conference on "Computer as a Tool"*, 2007, pp. 698–705. DOI: 10.1109/EURCON.2007.4400335.

[16] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem, *Brax - a differentiable physics engine for large scale rigid body simulation*, version 0.0.10, 2021. [Online]. Available: http://github.com/google/brax.

[17] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, *Reproducibility of benchmarked deep reinforcement learning tasks for continuous control*, 2017. arXiv: 1708.04133 [cs.LG].

[18] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying Generalization in Reinforcement Learning," en, in *International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, May 2019, pp. 1282–1289. [Online]. Available: http://proceedings.mlr.press/v97/cobbe19a.html.

[19] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," *arXiv:1802.01561 [cs]*, Jun. 2018, arXiv: 1802.01561. [Online]. Available: http://arxiv.org/abs/1802.01561.

[20] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A Study on Overfitting in Deep Reinforcement Learning," *arXiv:1804.06893 [cs, stat]*, Apr. 2018, arXiv: 1804.06893. [Online]. Available: http://arxiv.org/abs/1804.06893.

[21] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and Regularization in DQN," *arXiv:1810.00123 [cs, stat]*, Jan. 2020, arXiv: 1810.00123. [Online]. Available: http://arxiv.org/abs/1810.00123.

[22] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta Learn Fast: A New Benchmark for Generalization in RL," *arXiv:1804.03720 [cs, stat]*, Apr. 2018, arXiv: 1804.03720. [Online]. Available: http://arxiv.org/abs/1804.03720.

[23] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi, "Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation," *arXiv:1806.10729 [cs, stat]*, Nov. 2018, arXiv: 1806.10729. [Online]. Available: http://arxiv.org/abs/1806.10729.

[24] A. Stone, O. Ramirez, K. Konolige, and R. Jonschkowski, *The distracting control suite – a challenging benchmark for reinforcement learning from pixels*, 2021. arXiv: 2101.02722 [cs.RO].

[25] R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel, *A survey of generalisation in deep reinforcement learning*, 2022. arXiv: 2111.09794 [cs.LG].

[26] A. Hallak, D. D. Castro, and S. Mannor, *Contextual markov decision processes*, 2015. arXiv: 1502.02259 [stat.ML].

[27] F. Doshi-Velez and G. Konidaris, "Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16, New York, New York, USA: AAAI Press, 2016, pp. 1432–1440, ISBN: 9781577357704.

[28] D. Ghosh, J. Rahme, A. Kumar, A. Zhang, R. P. Adams, and S. Levine, "Why generalization in RL is difficult: Epistemic pomdps and implicit partial observability," *CoRR*, vol. abs/2107.06277, 2021. arXiv: 2107.06277. [Online]. Available: https://arxiv.org/abs/2107.06277.

[29] B. Neyshabur, R. Tomioka, and N. Srebro, *In search of the real inductive bias: On the role of implicit regularization in deep learning*, 2015. arXiv: 1412.6614 [cs.LG].

[30] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization," *CoRR*, vol. abs/1611.03530, 2016. arXiv: 1611.03530. [Online]. Available: http://arxiv.org/abs/1611.03530.

[31] L. Lin and E. Dobriban, "What causes the test error? Going beyond bias-variance via ANOVA," *arXiv:2010.05170 [cs, math, stat]*, Feb. 2021, arXiv: 2010.05170. [Online]. Available: http://arxiv.org/abs/2010.05170.

[32] B. Adlam and J. Pennington, "Understanding Double Descent Requires a Fine-Grained Bias-Variance Decomposition," *arXiv:2011.03321 [cs, stat]*, Nov. 2020, arXiv: 2011.03321. [Online]. Available: http://arxiv.org/abs/2011.03321.

[33] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for Distributed Reinforcement Learning," *arXiv:1712.09381 [cs]*, Jun. 2018, arXiv: 1712.09381. [Online]. Available: http://arxiv.org/abs/1712.09381.

[34] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, *JAX: Composable transformations of Python+NumPy programs*, version 0.2.5, 2018. [Online]. Available: http://github.com/google/jax.

[35] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *CoRR*, vol. abs/2108.13264, 2021. arXiv: 2108.13264. [Online]. Available: https://arxiv.org/abs/2108.13264.

[36] P. Henderson, J. Romoff, and J. Pineau, "Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods," *CoRR*, vol. abs/1810.02525, 2018. arXiv: 1810.02525. [Online]. Available: http://arxiv.org/abs/1810.02525.