The Beta Policy for Continuous Control Reinforcement Learning

Po-Wei Chou

CMU-RI-TR-17-38

June 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Masters of Computer Science.

Abstract

Recently, reinforcement learning with deep neural networks has achieved great success in challenging continuous control problems such as 3D locomotion and robotic manipulation. However, in real-world control problems, the actions one can take are bounded by physical constraints, which introduces a bias when the standard Gaussian distribution is used as the stochastic policy. In this work, we propose to use the Beta distribution as an alternative and analyze the bias and variance of the policy gradients of both policies. We show that the Beta policy is bias-free and provides significantly faster convergence and higher scores over the Gaussian policy when both are used with trust region policy optimization (TRPO) and actor critic with experience replay (ACER), the state-of-the-art on- and off-policy stochastic methods respectively, on OpenAI Gym's and MuJoCo's continuous control environments.

Acknowledgments

I would like to thank my advisor, Sebastian Scherer, for guiding and supporting me for the last two years. You give me the resources and the freedom to do independent research. I'm deeply thankful for your patience and support along the way and I really enjoy working with you. I would like to thank my thesis committee for all of their help, guidance, and feedback through this process. I would also like to thank Guan-Horng Liu, Ming Hsiao, Yen-Chi Chen, Wen Sun and Nick Rhinehart for many helpful discussions, suggestions and comments on the paper. This research was funded under award by Yamaha Motor Corporation and ONR under award N0014-14-1-0643.

Contents

List of Figures

Chapter 1

Introduction

Over the past years, reinforcement learning with deep feature representations [16, 21] has achieved unprecedented (or even super-human level) successes in many tasks, including playing Go [44] and playing Atari games [14, 28, 29, 40].

In reinforcement learning tasks, the agent's action space may be discrete, continuous, or some combination of both. Continuous action spaces are generally more challenging [25]. A naive approach to adapting deep reinforcement learning methods, such as deep Q-learning [28], to continuous domains is simply discretizing the action space. However, this method has several drawbacks. If the discretization is coarse, the resulting output will not be smooth; if it is fine, the number of discretized actions may be intractably high. This issue is compounded in scenarios with high degrees of freedom (*e.g.*, robotic manipulators and humanoid robots), due to the curse of dimensionality [3].

There has been much recent progress in model-free continuous control with reinforcement learning. Asynchronous Advantage Actor-Critic (A3C) [30] allows neural network policies to be trained and updated asynchronously with multiple CPU cores in parallel. Value Iteration Networks [50], provide a differentiable module that can learn to plan. Exciting results have been shown on highly challenging 3D locomotion and manipulation tasks [15, 40, 41], including real-world robotics problems where the inputs is raw visual data [24, 25, 58]. Derivative-free black box optimization like evolution strategies [39] have also been proven to be very successful in wide variety of tasks.

Despite recent successes, most reinforcement learning algorithms still require large amounts of training episodes and huge computational resources. This limits their applicability to richer, more complex, and higher dimensional continuous control real-world problems.

In stochastic continuous control problems, it is standard to represent their distribution with a Normal distribution $\mathcal{N}(\mu, \sigma^2)$, and predict the mean (and sometimes the variance) of it with a function approximator such as deep neural networks [9, 30, 61]. This is called a *Gaussian Policy*.

By computing the gradients of the policy with respect to μ and σ, backpropagation [38] and mini-batch stochastic gradient descent (or ascent) can be used to train the network efficiently.

However, a little-studied issue in recent approaches is that for many applications, the action spaces are bounded: action can only take on values within a bounded (finite) interval due to physical constraints. Examples include the joint torque of a robot arm manipulator and the steering angle and acceleration limits of Ackermann-steered vehicles. In these scenarios, any

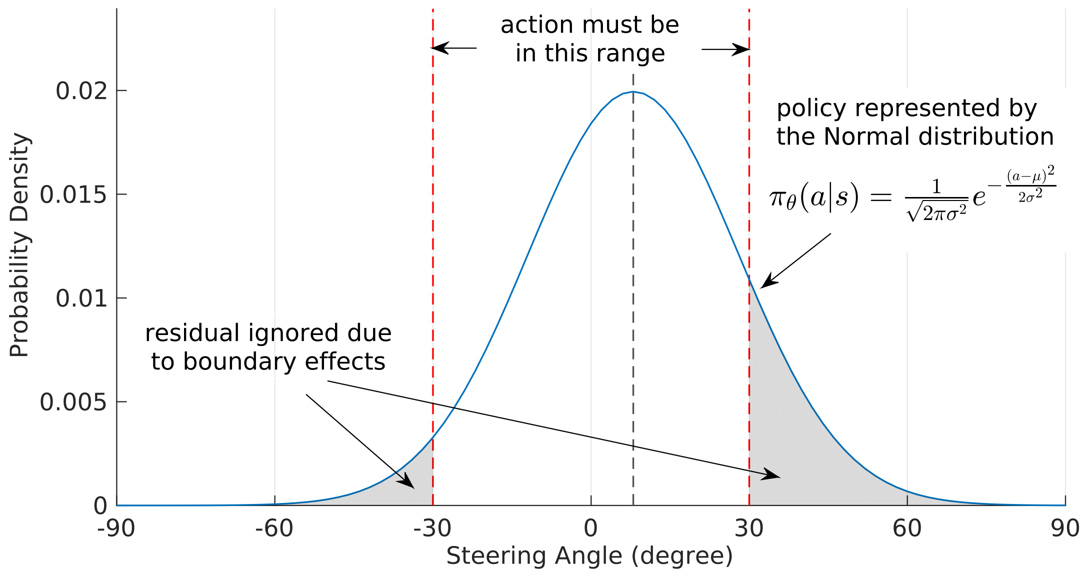


Figure 1.1: An example of continuous control with bounded action space. In most real-world continuous control problems, the actions can only take on values within some bounded interval (finite support). For example, the steering angle of most Ackermann-steered vehicles can only range from $-30°$ to $+30°$.

probability distribution with infinite support like the Gaussian will unavoidably introduce an estimation bias due to boundary effects (as in Figure 1.1), which may slow down the training progress and make these problems even harder to solve.

In this work, we focus on continuous state-action deep reinforcement learning. We address the shortcomings of the Gaussian distribution with a finite support distribution. Specifically, we use the Beta distribution with shape parameters α, β as in (3.2) and call this the *Beta policy*. It has several advantages. First, the Beta distrbution is finite-support and does not suffer from the same boundary effects as the Gaussian does. Thus it is bias-free and converges faster, which means a faster training process and a higher score. Second, since we only change the underlying distribution, it is compatible with all state-of-the-art stochastic continuous control on- and off-policy algorithms such as trust region policy optimization (TRPO) [40] and actor-critic with experience replay (ACER) [56].

We show that the Beta policy provides substantial gains in scores and training speed over the Gaussian policy on several continuous control environments, including two simple classical control problems in OpenAI Gym [6], three multi-joint dynamics and control problems in MuJoCo [52], and one all-terrain-vehicle (ATV) driving simulation in an off-road environment.

Chapter 2

Background

2.1 Preliminaries

We model our continuous control reinforcement learning as a *Markov decision process* (MDP). An MDP consists of a state space $\mathcal{S}$, an action space $\mathcal{A}$, an initial state s_0, and the corresponding state distribution $p_0(s_0)$, a stationary transition distribution describing the environment dynamics $p(s_{t+1}|s_t, a_t)$ that satisfies the Markov property, and a reward function $r(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for every state s and action a. An agent selects actions to interact with the environment based on a *policy*, which can be either deterministic or stochastic. In this paper, we focus on the latter. A stochastic policy can be described as a probability distribution of taking an action a given a state s parameterized by a n-dimensional vector $\theta \in \mathbb{R}^n$, denoted as $\pi_\theta(a|s) : \mathcal{S} \to \mathcal{A}$.

At each timestep t, a policy distribution $\pi_\theta(a|s_t)$ is constructed from the distribution parameters (*e.g.*, from $\mu_\theta(s), \sigma_\theta(s)$ if it's a Normal distribution). An action a_t is then sampled from this distribution to interact with the environment, i.e. $a_t \sim \pi_\theta(\cdot|s_t)$. Starting from an initial state, an agent follows a policy to interact with the MDP to generate a trajectory of states, actions, and rewards $\{s_0, a_0, r_0, \ldots, s_T, a_T, r_T\}$. The goal of an agent is to maximize the *return* from a state, defined as the total discounted reward $r_t^\gamma = \sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i})$, where $\gamma \in (0,1]$ is the discount factor describing how much we favor current reward over those in the future.

To describe how good it is being in state s under the policy π, a state-value function $V^\pi(s) = \mathbb{E}_\pi[r_0^\gamma|s_0 = s]$ is defined as the expected return starting from state s, following the policy π, interacting with environment dynamics, and repeating until the maximum number of episodes is reached. An action-value function $Q^\pi(s,a)$, which describes the value of taking a certain action, is defined similarly, except it is the expected return starting from state s after taking an action a under policy π.

The goal in reinforcement learning is to learn a policy maximizing the expected return from the start distribution

$$J(\pi_\theta) = \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(s,a) r(s,a) da\, ds \tag{2.1}$$

$$= \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[r(s,a)], \tag{2.2}$$

where $\rho^\pi(s) = \sum_{t=0}^{\infty} \gamma^t p(s_t = s)$ is the unnormalized discounted state visitation frequency in the limit [48].

2.2 Stochastic Policy Gradient

Policy gradient methods are featured heavily in the state-of-the-art model-free reinforcement learning algorithms [9, 25, 30, 56]. In these methods, training of the policy is performed by following the gradient of the performance with respect to the parameters, $\nabla_\theta J(\pi_\theta)$. This gradient can be computed from the *Policy Gradient Theorem* [48] by simply changing $r(s,a)$ to $Q^\pi(s,a)$ in (2.2) and moving the gradient operator inside the integral:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_\mathcal{S} \rho^\pi(s) \int_\mathcal{A} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) da\, ds \\ &= \int_\mathcal{S} \rho^\pi(s) \int_\mathcal{A} \pi_\theta(a|s) g_q da\, ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[g_q]\,, \end{aligned} \tag{2.3}$$

where $\pi_\theta(a|s)$ instead of $\pi_\theta(s,a)$ is used to represent a stochastic policy and g_q is the policy gradient estimator using $Q^\pi(s,a)$ as the target

$$g_q = \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\,. \tag{2.4}$$

However, exact computation of the double integral in (2.3) is generally intractable. Instead, we can estimate it by sampling: given enough samples of g_q, the sample mean $\bar{g_q}$, will converge to its expectation, $\nabla_\theta J(\pi_\theta)$, by the law of large numbers

$$\bar{g_q} = \frac{1}{n}\sum_{i=1}^{n} g_q \xrightarrow{P} \mathbb{E}[g_q] = \nabla_\theta J(\pi_\theta), \quad \text{as } n \to \infty\,. \tag{2.5}$$

Estimating the policy gradient is one of the most important issues in reinforcement learning. We want g_q in (2.4) to be bias-free so that it converges to the true policy gradient. As we will show in the following section, this is not always true. At the same time, we also want to reduce the sample variance, so that the gradient is less noisy and stable, as this improves the convergence rate and speeds up the training progress. The action-value function $Q^\pi(s,a)$ can be estimated by a variety of sample-based algorithms such as Monte-Carlo (MC) or temporal-difference (TD) learning. A lookup table is usually used to store $Q^\pi(s,a)$ for each state s and action a.

2.3 Stochastic Actor-Critic

For an MDP with intractably large state space, using a lookup table is no longer practical. Instead, function approximation methods are more common. Deep Q-Networks (DQN) [28] use a deep neural network parameterized by θ_v to approximate the action-value function, denoted as $Q_{\theta_v}(s,a) \approx Q^\pi(s,a)$. This is appealing since deep learning has been shown to be very powerful and successful in computer vision, speech recognition and many other domains [22].

Unfortunately, direct application of DQN to continuous action spaces is difficult. First, as mentioned earlier, if we discretize the action space, it is hampered by the curse of dimensionality. Second, in the Q-learning algorithm, one needs to find the (greedy) action that maximizes the action-value function, i.e. $a = \arg\max_a Q_{\theta_v}(s, a)$. This means an additional optimization procedure is required at every step inside the stochastic gradient descent optimization, which makes it impractical.

The solution to this is the Actor-Critic methods [8, 31, 35, 47]. In these methods an *actor* learns a policy to select actions and a *critic* estimates the value function, and criticizes decisions made by the actor. The actor with policy $\pi_\theta(a|s)$ and the critic with $Q_{\theta_v}(s, a)$ are trained simultaneously.

Replacing the true action-value function $Q^\pi(s, a)$ by a function approximator $Q_{\theta_v}(s, a)$ may introduce bias. Nonetheless, in practice, with the help of *experience replay* [26] and *target networks* [28] actor-critic methods still converge to good policies, even with deep neural networks [25, 44].

One of the best known variance reduction technique for actor-critic without introducing any bias is to substract a baseline function $B(s)$ from $Q^\pi(s, a)$ in (2.4) [12]. A natural choice for $B(s)$ is $V^\pi(s)$, since it is the expected action-value function $Q^\pi(s, a)$, i.e. $V^\pi(s) = \mathbb{E}_{a\sim\pi_\theta}[Q^\pi(s, a)]$. This gives us the definition of *advantage* function $A^\pi(s, a)$ and the following stochastic policy gradient estimates:

$$A^\pi(s, a) \triangleq Q^\pi(s, a) - V^\pi(s)\,, \tag{2.6}$$

$$g_a = \nabla_\theta \log \pi_\theta(a|s) A^\pi(s, a)\,. \tag{2.7}$$

The advantage function $A^\pi(s, a)$ measures how much better than the average it is to take an action a. With this method, the policy gradient in (2.4) is shifted in a way such that it is the relative difference, rather than the absolute value $Q^\pi(s, a)$, that determines the gradient.

Chapter 3

Infinite/Finite Support Distribution for Stochastic Policy in Continuous Control

Using the Gaussian distribution as a stochastic policy in continous control has been well-studied and commonly used in the reinforcement learning community since [61]. This is most likely because the Gaussian distribution is easy to sample and has gradients that are easy to compute, which makes it the first choice of the probability distribution.

However, we argue that this is not always a good choice. In most continuous control reinforcement learning applications, actions can only take on values within some finite interval due to physical constraints, which introduces a non-negligible bias caused by boundary effects, as we show below.

This motivates us to use a distribution that can solve this problem. Among continuous distributions with finite support, the well-known Beta distribution emerges as a natural candidate, as it is expressive yet simple, with two easily interpretable parameters.

In Bayesian statistics, the Beta distribution is often used as the conjugate prior probability distribution for the Bernoulli and binomial distributions, describing the initial belief about the probability of the success of each trial [4]. One loose inspiration behind our use of the Beta function is spike-rate coding, as seen in biological neurons [10], or pulse density modulation, as used in artificial systems; here, the Beta could be seen as modeling the probability of a neuron firing, or a pulse being emitted, over a small time interval.

In the following, we show that the Beta policy is bias-free and a better choice than the Gaussian. We compare the variance of the policy gradient of both policies and show that as with the Gaussian policy, Natural Policy Gradient is also necessary for the Beta policy to achieve a good performance.

3.1 Gaussian Policy

To employ a Gaussian policy, we can define the policy as

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(a-\mu)^2}{2\sigma^2}\right) , \tag{3.1}$$

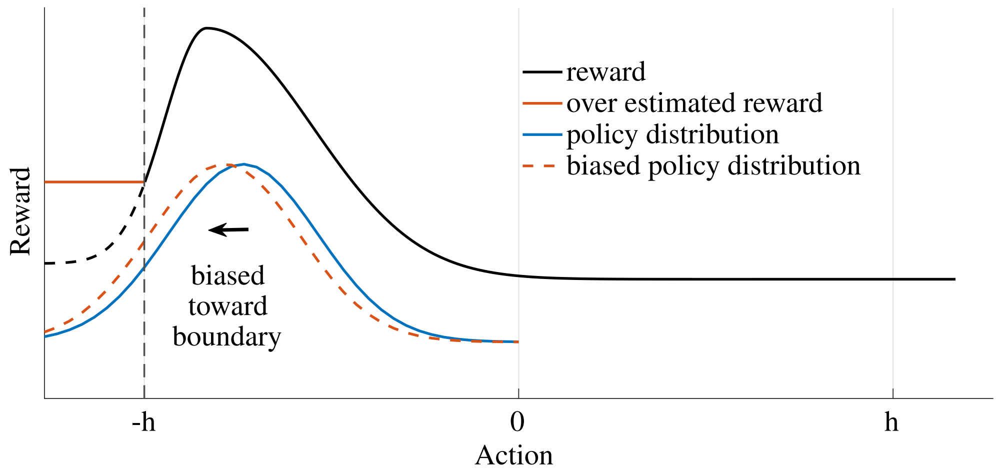


Figure 3.1: An example of over estimation of rewards outside the boundary.

where the mean $\mu = \mu_\theta(s)$ and the standard deviation $\sigma = \sigma_\theta(s)$ are given by a function approximator parameterized by θ. To enable the use of backpropagation, we can reparameterize [15] action $a \sim \pi_\theta(\cdot|s)$ as $a = \mu_\theta(s)+\sigma_\theta(s)\xi$, where $\xi \sim \mathcal{N}(0,1)$. The policy gradient with respective to μ, σ can be computed explicitly as $\nabla_\mu \log \pi_\theta(a|s) = \frac{(a-\mu)}{\sigma^2}$ and $\nabla_\sigma \log \pi_\theta(a|s) = \frac{(a-\mu)^2}{\sigma^3} - \frac{1}{\sigma}$. In general, for problem with higher degrees of freedom, all action dimensions are assumed to be mutually independent.

3.2 Bias due to Boundary Effect

Modeling a finite support stochastic policy with an infinite support probability distribution may introduce bias. By the definition of infinite support, every action a is assigned with a probability density $\pi_\theta(a|s)$ that is greater than 0. Nonetheless, in reality, all actions outside the finite support have probability exactly equal to 0 (see Figure 1.1).

To simplify the analysis, we consider the phased update framework [20]: in each phase, we are given n samples of $Q^{\pi_\theta}(s,a)$ from environments under a fixed π_θ. In other words, we focus mainly on the inner expectation of (2.2). Without loss of generality, let us consider an one-dimensional action space $\mathcal{A} = [-h, h]$, where $2h$ is the width of the closed interval. For any action space that is not symmetric around 0, we can always map it to $[-h, h]$ by scaling and shifting.

So far we have seen two main approaches to employ the Gaussian policy in this bounded action scenario in the existing RL implementations:

1. Send the action to the environment without capping (truncating) it first, let the environment cap it for us, and use the uncapped action to compute the policy gradient.
2. Cap the action to the limit, send it to the environment, and use the capped action to compute the policy gradient.

In the first approach, by letting the environment capping the actions for us, we simply pretend there are no action bounds. In other words, all actions outside the bounds just happen to have the *same effect* as the actions at the limits. The policy gradient estimator in (2.4) now becomes $g'_q = \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a')$, where a' is the truncated action. The bias of the estimator g'_q is

$$\begin{aligned}
&\mathbb{E}[g'_q] - \nabla_\theta J(\pi_\theta) \\
&= \mathbb{E}_s \left[\int_{-\infty}^{\infty} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a') da \right] - \nabla_\theta J(\pi_\theta) \\
&= \mathbb{E}_s \left[\int_{-\infty}^{-h} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \left[Q^\pi(s, -h) - Q^\pi(s, a) \right] da \right. \\
&\quad \left. + \int_{h}^{\infty} \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \left[Q^\pi(s, h) - Q^\pi(s, a) \right] da \right] .
\end{aligned}$$

We can see that as long as the action space $\mathcal{A}$ covers the support of the policy distribution (i.e. $\operatorname{supp}(\pi_\theta(a|s)) \subset \mathcal{A}$ or as $h \to \infty$) the last two integrals immediately evaluate to zero. Otherwise, there is a bias due to the boundary effect.

The boundary effect can be better illustrated by the example in Figure 3.1 where the reward function peaks (assuming a single mode) at a good action close to the boundary. This effectively extends the domain of reward (or value) function to previously undefined region by extrapolating, or more precisely, the "replicated" padding, which results in artificially higher rewards outside the bounds and therefore bias the estimated policy distribution toward the boundary. As for multimodal reward functions, one might need to consider the use of a mixture model or other density estimation methods since neither the Gaussian nor the Beta suffices under this scenario. However, this is beyond the scope of our discussion.

To make things worse, as σ grows, bias also increases. This makes sense intuitively, because as σ grows, more probability density falls outside the boundary. Note that this is not an unusual case: to encourage the actor to explore the state space in the early stage of training, larger σ is needed.

In the second approach, the policy gradient estimator is even more biased because the truncated action a' is used both in the state-value function Q^π and in the gradient of log probability $\nabla_\theta \log \pi_\theta$, i.e. $g''_q = \nabla_\theta \log \pi_\theta(a'|s) Q^\pi(s, a')$. In this case, the commonly used variance reduction techique is less useful since $\mathbb{E}_{a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a'|s) V^\pi(s)]$ no longer integrates to 0 as it should be if a instead of a' was used. Not only does it suffer from the same bias proble

3.3 Beta Policy

Let us now consider the Beta distribution

$$f(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1 - x)^{\beta-1} , \tag{3.2}$$

where α and β are the shape parameters and $\Gamma(\cdot)$ is the *Gamma* function that extends factorial to real numbers, i.e. $\Gamma(n) = (n - 1)!$ for positive integer n. The beta distribution has a support $x \in [0, 1]$ (as shown in Figure 3.2) and it is often used to describe the probability of success,

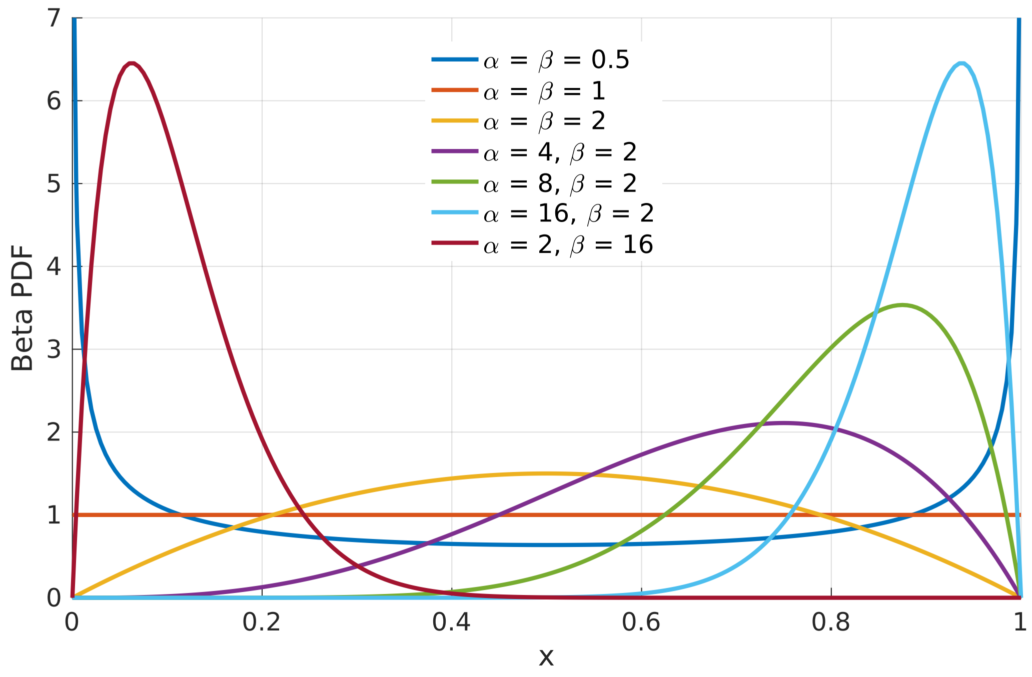


Figure 3.2: Probability density function of Beta distributions with different α and β.

where $\alpha - 1$ and $\beta - 1$ can be thought of as the counts of successes and failures from the prior knowledge respectively.

We use $\pi_\theta(a|s) = f(\frac{a+h}{2h}; \alpha, \beta)$ to represent the stochastic policy and call it the *Beta Policy*. Since the beta distribution has finite support and no probability density falls outside the boundary, the Beta policy is bias-free. The shape parameters $\alpha = \alpha_\theta(s), \beta = \beta_\theta(s)$ are also modeled by neural networks with parameter θ. In this paper, we only consider the case where $\alpha, \beta > 1$, in which the Beta distribution is concave and unimodal.

3.3.1 Variance Compared to Gaussian Policy

One unfortunate property of the Gaussian policy is that the variance of policy gradient estimator is inversely proportional to σ^2. As the policy improves and becomes more deterministic ($\sigma \to 0$), the variance of (2.4) goes to infinity [42, 43, 64].

This is mainly because the ordinary policy gradient defined in (2.4) does not always yield the steepest direction [1], but the *natural policy gradient* [19, 34] does. The natural policy gradient is given by

$$g_q^{nat} = \mathcal{I}^{-1}(\theta) g_q \,, \tag{3.3}$$

where $\mathcal{I}(\theta)$ is the Fisher information matrix defined as

$$\mathcal{I}(\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \, \nabla_\theta \log \pi_\theta(a|s)^T \right] \tag{3.4}$$

and the variance of the policy gradient is

$$\begin{aligned}\mathbb{V}_a[g_q] &= \mathbb{E}_a[g_q^2] - \mathbb{E}_a^2[g_q] \\ &= \mathbb{E}_a\left[\nabla_\theta \log \pi_\theta(a|s)\, \nabla_\theta \log \pi_\theta(a|s)^T Q^{\pi 2}(s,a)\right] - \mathbb{E}_a^2[g_q]\,.\end{aligned}$$

First note that it is often more useful (and informative) to say X *standard deviations* rather than just Y *points* above the average. In other words, one should consider the metric defined on the underlying statistical manifold instead of the Euclidean distance. The Fisher information matrix $\mathcal{I}(\theta)$ is such metric [18]. A gradient vector consists of *direction* and *length*. For a univariate Gaussian distribution, the ordinary policy gradient has the correct direction, but not the correct length. As one moves in the parameter space, the metric defined on this space also changes, which effectively changes the length of the ordinary gradient vector. The natural gradient adjusts the learning rate according to the probability distribution, slowing down the learning rate when the distance on the parameter space compresses, and speeding it up as the distance expands.

For the Gaussian distribution, the Fisher information matrix has the form of $1/\sigma^2$ (see Supplementary Section A.1). The more deterministic the policy becomes, the smaller the size of step (proportional to σ^2) is needed to take in order to increase the same amount of objective function. As a result, a constant step of the ordinary gradient descent update will overshoot, which results in higher variance of (2.4).

As for the Beta policy, the Fisher information matrix goes to zero as policy becomes deterministic, as does the variance of the policy gradient (see Supplementary Section A.2). However, this is not a desirable property. This can be better illustrated by the example in Figure 3.3, where the curvature flattens out at a rate so high that it is impossible for the ordinary policy gradient to catch up with, making the estimation of α and β increasingly hard without the use of the natural policy gradient. In this case, not just the length has to be adjusted, but also the off-diagonal terms in the information matrix.

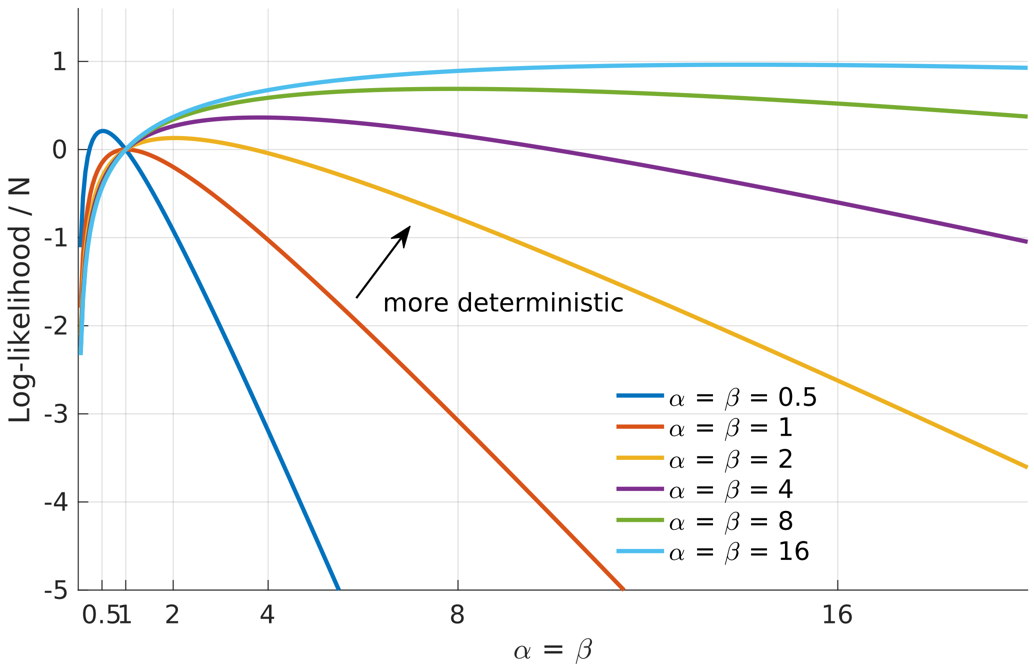


Figure 3.3: Log-likelihood of Beta distributions. For each curve, we sample N data points $x_i \sim f(x; \alpha, \beta)$ and plot the the averaged log-likelihood curve by evaluating $\sum_{i=1}^{N} \log f(x_i; \alpha, \beta)$ at $\alpha = \beta$ ranging from 0 to 20. The maximum log-likehood will peak at the same α, β where the samples were drawn from initially if N is large enough. Unlike the Normal distribution, the more deterministic the Beta distribution becomes, the flatter the log-likelihood curve (from blue, orange . . . to cyan).

Chapter 4

Experiments

We evaluate our proposed methods in a variety of environments, including the classical control problems in OpenAI Gym, the physical control and locomotion tasks in Multi-Joint dynamics with Contact (MuJoCo) physical simulator, and a setup intended to simulate an autonomous driving in an off-road environment.

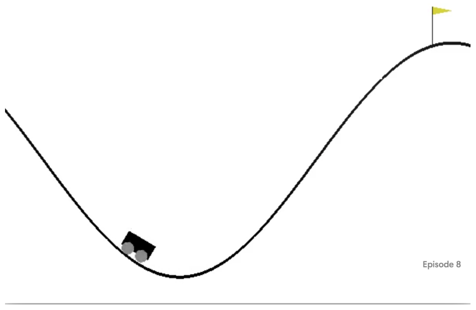
In all experiments, inputs are processed using neural networks with architectures depending on the observation and action spaces. For both distributions, we assume the action dimensions are independent and thus have zero covariance. For all architectures, the last two layers output two $||\mathcal{A}||$-dimensional real vectors: either (a) the mean $\boldsymbol{\mu}$ and the variance $\boldsymbol{\sigma}^2$ for a multivariate normal distribution with a spherical covariance, or (b) the shape vectors $\boldsymbol{\alpha}, \boldsymbol{\beta}$ for a Beta distribution. Specifically, for the Normal distribution, $\boldsymbol{\mu}$ is modeled by a linear layer and $\boldsymbol{\sigma}^2$ by a *softplus* element-wise operation, $\log(1 + \exp(x))$. For the Beta distribution, $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are also modeled by *softplus*, except a constant 1 is added to the output to make sure $\alpha, \beta \geq 1$ (see Section 3).

For both policy distributions, we add the entropy of policy $\pi_\theta(a|s)$ with a constant multiplier 0.001 encouraging exploration in order to prevent premature convergence to sub-optimal policies [30]. A discount factor γ is set to 0.995 across all tasks.

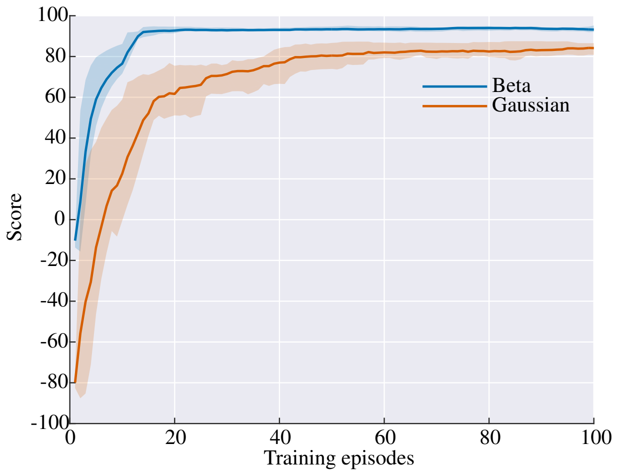
4.1 Classical Control

First, as a proof of concept, we compare the Beta distribution with Normal distribution in two classical continuous control: *MountainCarContinuous-v0* and *Pendulum-v0* (see Figure 4.1(a) and 4.1(c)) using the simplest actor-critic method: no asynchronous updates [30], experience replays, or natural policy gradient are used. For the actor, we only use low-dimensional physical state like joint velocities and vehicle speed. No visual input, such as RGB pixel values, is used. We first featurize the input state to 400-dimensional vectors using random Radial Basis Functions [36] and then pass it to a simple neural network where the only layer is the final output layer generating statistics for the policy distribution. This is effectively a linear combination of state features: $\phi(s)^T\theta$, where ϕ is the featurizing function and θ is the weight vector to be learnt. For the critic, we use 1-step TD-error[1] as an unbiased estimation of the advantage function in (2.7).

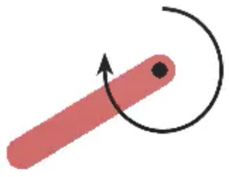
[1]k-step TD error = $\sum_{i=0}^{k-1}\left(\gamma^i r_{t+i} + \gamma^k V_\theta(s_{t+k})\right) - V_\theta(s_t)$



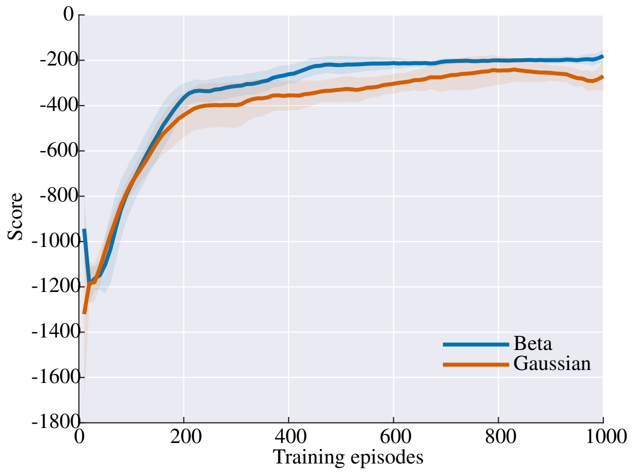
(a) Mountain Car



(b) Mountain Car

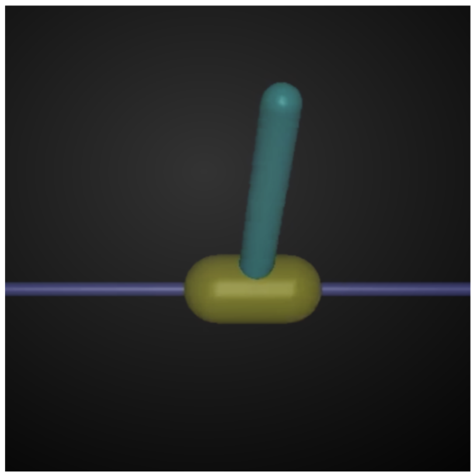


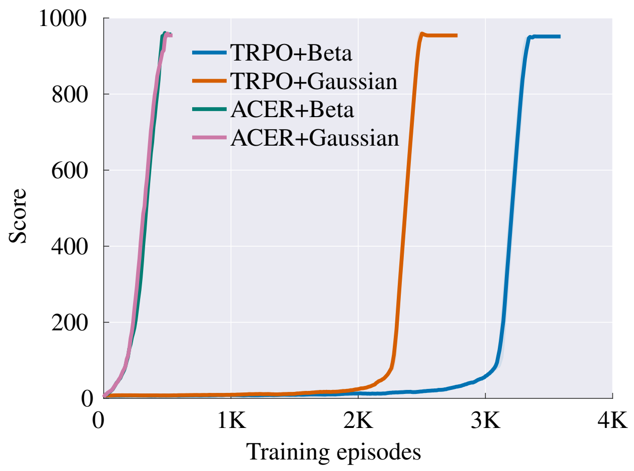
(c) Pendulum



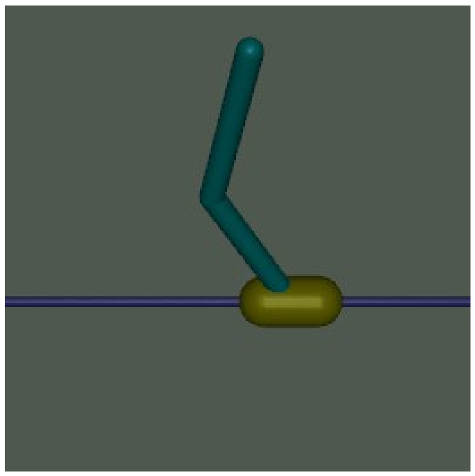
(d) Pendulum

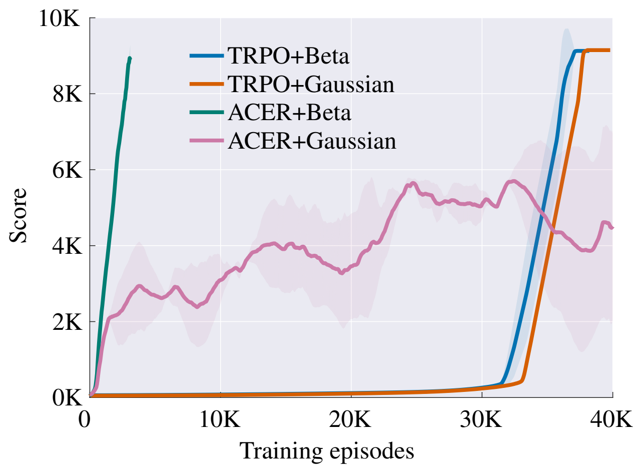
Figure 4.1: Screenshots of the continuous control tasks on OpenAI Gym classical control problems and training summary for Normal distribution and Beta distribution. The x-axis shows the total number of training epochs. The y-axis shows the average scores (also 1 standard deviation) over several trials.


(a) Inverted Pendulum



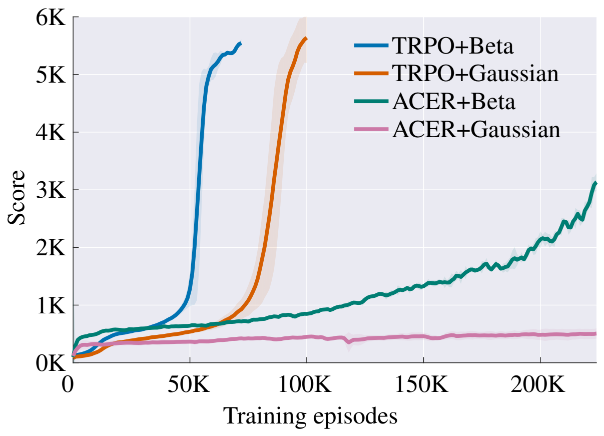
(b) Inverted Pendulum


(c) Double Pendulum



(d) Double Pendulum

(e) Humanoid



(f) Humanoid

Figure 4.2: Screenshots of the continuous control tasks on OpenAI Gym's MuJoCo continuous control problems and training summary for Normal distribution and Beta distribution. The x-axis shows the total number of training epochs. The y-axis shows the average scores (also 1 standard deviation) over several trials.

Table 4.1: List of Environments

ENVIRONMENTS	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	DOF
MOUNTAINCARCONTINUOUS-V0	2	1	1
PENDULUM-V0	3	1	1
INVERTEDPENDULUM-V1	4	1	2
INVERTEDDOUBLEPENDULUM-V1	11	1	3
HUMANOID-V1	376	17	27
OFF-ROAD DRIVING	400 + 6	2	6

In both tasks, we found that Beta policies consistently provide faster convergence than Gaussian policies (see Figure 4.1(b) and 4.1(d)).

4.2 MuJoCo

Next, we evaluate Beta policies on three OpenAI Gym's MuJoCo environments: *InvertedPendulum-v1*, *InvertedDoublePendulum-v1* and *Humanoid-v1* (see Figure 4.2(a), 4.2(c), and 4.2(e)) using both on-policy and off-policy methods. Results are shown in Figure 4.2(b), 4.2(d), and 4.2(f). The goal for the first two is to balance the inverted pendulum and stay upright as long as possible. For the humanoid robot, the goal is to walk as fast as possible without falling at the same time minimize actions to take and impacts of each joint.

In the on-policy experiments, we use the original implementation[2] provided by the authors of TRPO [40] with the same hyperparameters and configuration that were used to generate their state-of-the-art training results. TRPO is similar to natural policy gradient methods but more efficient for optimizing large function approximators such as neural networks.

By simply changing the policy distribution, we find that TRPO+Beta provides a significant performance improvement (about 2x faster) over TRPO+Gaussian on the most difficult Humanoid environment. However, only a slight improvement over the Gaussian policy is observed on the less difficult Inverted Double Pendulum. For the simplest task, Inverted Pendulum, Gaussian+TRPO has a slight advantage over TRPO+Beta; however, since both methods completely solve the Inverted Pendulum in a matter of minutes, the absolute difference is small.

For the off-policy experiments, we implement ACER in TensorFlow according to Algorithm 3 in [56]. Asynchronous updates with four CPUs and *non-prioritized* experience replays of ratio 8 are used. The learning rate is sampled log-uniformly from $[10^{-4}, 5 \times 10^{-4}]$. The soft updating parameter for the average policy network is set to 0.995 across all tasks. For the Gaussian distribution, σ is squashed by a hyperbolic tangent function to prevent a variance that is too large (too unstable to be compared) or too small (underflow). Specifically, we only allow σ ranging from 10^{-4} to h (see Section 3.2).

Substantial improvements over Gaussian policies are also observed in the off-policy experiments among all tasks. Though sometimes Gaussian can find a good policy faster than the Beta, it plummets after tens of training episodes, then repeats, which results in a lower average score and

[2]See `https://github.com/joschu/modular_rl`.

higher variance (Figure 4.2(d)). The improvement over the Gaussian policy on the Humanoid is the most prominent and that on the Inverted Pendulum is less significant. This trend suggests that the bias introduced by constrained action spaces is compounded in systems with higher degrees of freedom.

Note that these results are not directly comparable with the previous on-policy TRPO. First, a fast and efficient variant of TRPO was proposed in ACER as a trade-off. Second, we do not use the *generalized advantage estimator* (GAE) [41], though it can be done by modifying the *Retrace* [31] target update rule in ACER. Third, a smaller batch size is usually used during the alternating on-policy and off-policy updates in ACER. Similar unstable behaviors can also be observed when we try to reduce the batch size of update in on-policy TRPO experiments. We believe this is because a smaller batch size means more frequent updates, which helps the agents to explore faster in the early stage of training but starts to hamper the performance in the later stage, when a larger sample size is needed to reduce the sample variance in such unstable robot arm (or locomotion) configurations.

Similar to the findings in evolution strategies [39], humanoid robots under different stochastic policies also exhibit different gaits: those with Beta policies always walk sideways but those with Gaussian policies always walk forwards. We believe this suggests a different exploration behavior and could be an interesting research direction in the future.

4.3 Off-Road Autonomous Driving in Local Maps

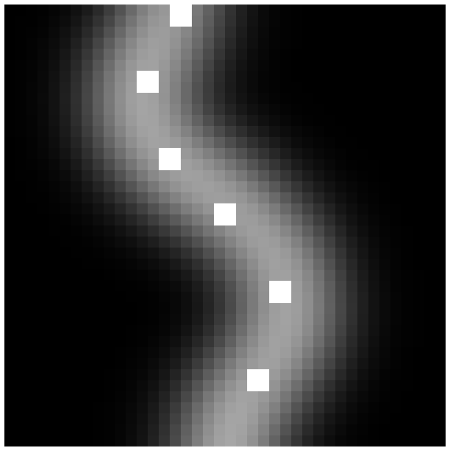
Last, we consider a simplified All Terrain Vehicle (ATV) autonomous navigation problem. In this problem, the angent (an ATV vehicle) must navigate an off-road 2D map where each position in the map has a scalar traversability value. The vehicle is rewarded for driving on smoother terrain, while maintaining a minimum speed.

The map is 20 $\times$ 20 meters, represented as a 40 $\times$ 40 grid (as shown in Figure 4.3(a)). The input of the agent is the vehicle's physical state and top-down view of 20 $\times$ 20 grid in front of the vehicle, rotated to the vehicle frame. The vehicle's action space consists of two commands updated every 5 Hz: steering angle and forward speed. Steering angle is constrained to $[-30°, 30°]$ and speed is constrained to $[6, 40]$ km/h. The vehicle's state is $(x, y, \omega, \dot{x}, \dot{y}, \dot{\omega})$, where x, y are velocity in lateral and forward direction, ω is the yaw rate, and $\dot{x}, \dot{y}, \dot{\omega}$ are the time derivatives. The vehicle commands are related to $\dot{y}$ and $\dot{\omega}$ by a second order vehicle model.

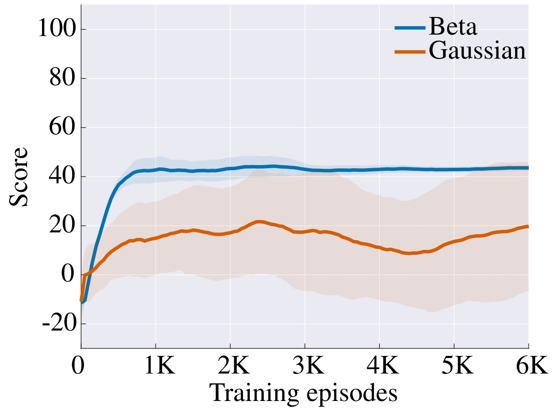
The vehicle's dynamics, which are *unknown* to the agent (thus model-free), are derived from a vehicle model obtained by system identification. The data for the identification was recorded by driving an ATV manually in an off-road environment. In all simulations, a constant timestep of 0.025 seconds is used to integrate $\dot{x}, \dot{y}, \dot{\omega}$ for generation of trajectories with a unicycle model.

We follow the (convolutional) network architectures used for 3D maze navigation in [27] and use the same setup of ACER as in Section 4.2, except we use a replay ratio sampled over the values $\{0.25, 1, 4\}$.

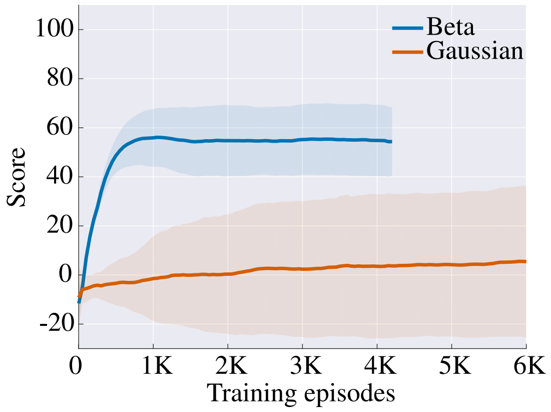
Results show the Beta policy consistently outperforms the Gaussian policy significantly under all different replay ratios. We found that higher replay ratio works better for the Beta but not for the Gaussian. We suspect that despite off-policy training being more sample efficient (a sample can be learnt several times using experience replay), it is generally noisier due to the use of importance sampling. Even with the help of Retrace, off-policy training with high experience replay ratio still destabilizes the Gaussian policy (Figure 4.3(d)).



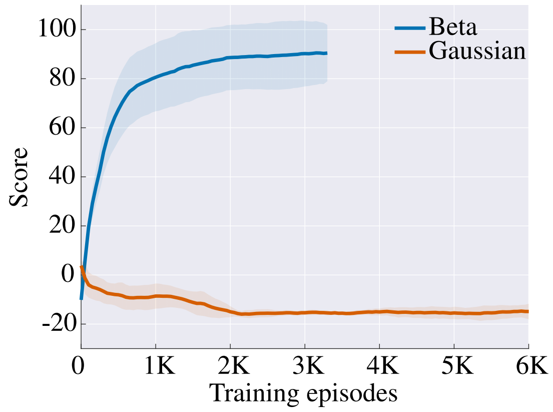
(a) Off-Road Driving



(b) replay ratio 0.25



(c) replay ratio 1



(d) replay ratio 4

Figure 4.3: Screenshots of off-road driving task and training summary for the Normal distribution and Beta distribution. The x-axis shows the total number of training epochs. The y-axis shows the average scores (also 1 standard deviation) over 10 different experiments with varying parameters (see text for details).

Chapter 5

Conclusion and Future Work

We introduce a new stochastic policy based on the Beta distribution for continuous control reinforcement learning. This method solves the bias problem due to boundary effects arising from the mismatch of infinite support of the commonly used Gaussian distribution and the bounded controls that can be found in most real-world problems. Our approach outperforms the Gaussian policy when TRPO and ACER, the state-of-the-art on- and off-policy methods, are used. It is also compatible with all other continuous control reinforcement algorithms with Gaussian policies.

For future work, we aim to apply this to more challenging real-world robotic learning tasks such as autonomous driving and humanoid robots, and extend it for more complex problems, e.g. by using mixtures of Beta distributions for multimodal stochastic policies.

Appendix A

Derivations

A.1 Fisher information matrix for the Normal Distribution

Under regularity conditions [57], the Fisher information matrix can also be obtained from the second-order partial derivatives of the log-likelihood function

$$I(\theta) = -\mathbb{E}[\frac{\partial^2 l(\theta)}{\partial \theta^2}]\,, \tag{D1}$$

where $l(\theta) = \log \pi_\theta(a|s)$. This gives us the Fisher information for the Normal distribution

$$\begin{aligned}\mathcal{I}(\mu,\sigma) &= -\mathbb{E}_{a\sim\pi_\theta}\left[\begin{array}{cc}\frac{\partial^2 l}{\partial\mu^2} & \frac{\partial^2 l}{\partial\mu\partial\sigma}\\ \frac{\partial^2 l}{\partial\sigma\partial\mu} & \frac{\partial^2 l}{\partial\sigma^2}\end{array}\right] \\ &= -\mathbb{E}_{a\sim\pi_\theta}\left[\begin{array}{cc}-\frac{1}{\sigma^2} & -2\frac{(a-\mu)}{\sigma^3}\\ -2\frac{(a-\mu)}{\sigma^3} & \frac{-3(a-\mu)^2}{\sigma^4}+\frac{1}{\sigma^2}\end{array}\right] = \left[\begin{array}{cc}\frac{1}{\sigma^2} & 0\\ 0 & \frac{2}{\sigma^2}\end{array}\right].\end{aligned} \tag{D2}$$

A.2 Fisher information matrix for the Beta Distribution

To see how variance changes as the policy converges and becomes more deterministic, let us first compute the partial derivative of $\log \pi_\theta(a|s)$ with respect to shape parameter α

$$\begin{aligned}\frac{\partial \log \pi_\theta(a|s)}{\partial\alpha} &= \frac{\partial}{\partial\alpha}\log\left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}a^{\alpha-1}(1-a)^{\beta-1}\right)\\ &= \frac{\partial}{\partial\alpha}(\log\Gamma(\alpha+\beta)-\log\Gamma(\alpha)+(\alpha-1)\log a)\\ &= \log a+\psi(\alpha+\beta)-\psi(\alpha)\,,\end{aligned}$$

where $\psi(\cdot) = \frac{d}{dz}\log\Gamma(\cdot)$ is the *digamma* function. Similar results can also be derived for β. From (D1), we have

$$\begin{aligned}
\mathcal{I}(\alpha, \beta) &= -\mathbb{E}_{a\sim\pi_\theta}\left[\begin{array}{cc} \frac{\partial^2 l}{\partial\alpha^2} & \frac{\partial^2 l}{\partial\alpha\partial\beta} \\ \frac{\partial^2 l}{\partial\beta\partial\alpha} & \frac{\partial^2 l}{\partial\beta^2} \end{array}\right] \\
&= -\mathbb{E}_{a\sim\pi_\theta}\left[\begin{array}{cc} \frac{\partial}{\partial\alpha}(\psi(\alpha+\beta) - \psi(\alpha)) & \frac{\partial}{\partial\beta}(\psi(\alpha+\beta)) \\ \frac{\partial}{\partial\alpha}(\psi(\alpha+\beta)) & \frac{\partial}{\partial\beta}(\psi(\alpha+\beta) - \psi(\beta)) \end{array}\right] \\
&= \left[\begin{array}{cc} \psi'(\alpha) - \psi'(\alpha+\beta) & -\psi'(\alpha+\beta) \\ -\psi'(\alpha+\beta) & \psi'(\beta) - \psi'(\alpha+\beta) \end{array}\right],
\end{aligned}$$

where $\psi'(z) = \psi^{(1)}(z)$ and $\psi^{(m)}(z) = \frac{d^{m+1}}{dz^{m+1}}\log\Gamma(z)$ is the *polygamma* function of order m. Figure A.1 shows how $\psi'(z)$ goes to 0 as $z \to \infty$.

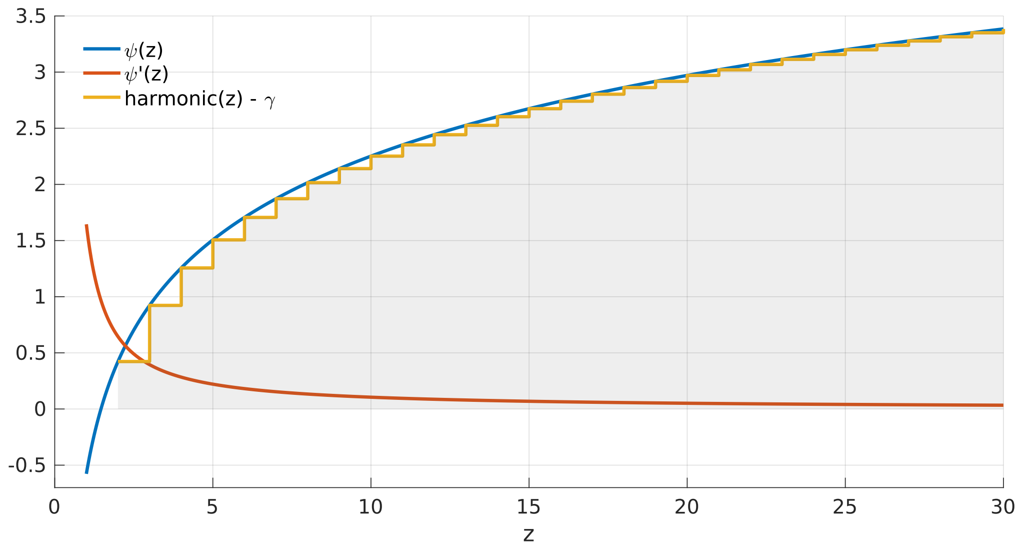


Figure A.1: Graphs of the *digamma* function, the *polygamma* function, and the harmonic series. The harmonic series and the digamma function are related by $\sum_{k=1}^{z}\frac{1}{k} = \psi(z+1)+\gamma$, where $\gamma = 0.57721\cdots$ is the Euler-Mascheroni constant.

Bibliography

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10 (2):251–276, 1998. 3.3.1

[2] J Andrew Bagnell and Jeff Schneider. Covariant policy search. IJCAI, 2003.

[3] Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956. 1

[4] J. M. Bernardo and A. F. M. Smith. *Bayesian Theory*. John Wiley & Sons, New York, 1994. 3

[5] Shalabh Bhatnagar, Richard Sutton, Mohammad Ghavamzadeh, and Mark Lee. Natural actor-critic algorithms. *Automatica*, 45(11), 2009.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. 1

[7] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*, pages 2177–2182. IEEE, 2012.

[8] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012. 2.3

[9] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1329–1338, 2016. 1, 2.2

[10] Wulfram Gerstner, Andreas K. Kreiter, Henry Markram, and Andreas V. M. Herz. Neural codes: Firing rates andbeyond. *Proceedings of the National Academy of Sciences*, 94(24): 12740–12741, 1997. 3

[11] Adolf Hermann Glattfelder and Walter Schaufelberger. *Control systems with input and output constraints*. Springer Science & Business Media, 2012.

[12] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5 (Nov):1471–1530, 2004. 2.3

[13] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *In The 5th International Conference on Learning Representations (ICLR)*, 2017.

[14] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep

learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014. 1

[15] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015. 1, 3.1

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. 1

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Harold Jeffreys. An invariant form for the prior probability in estimation problems. In *Proceedings of the Royal Society of London a: mathematical, physical and engineering sciences*, volume 186, pages 453–461. The Royal Society, 1946. 3.3.1

[19] Sham M Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, pages 1531–1538, 2002. 3.3.1

[20] Michael J Kearns and Satinder P Singh. Bias-variance error bounds for temporal difference updates. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pages 142–147. Morgan Kaufmann Publishers Inc., 2000. 3.2

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. 2.3

[23] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, pages 1–9, 2013.

[24] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 1

[25] 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. *arXiv preprint arXiv:1509.02971*, 2015. 1, 1, 2.2, 2.3

[26] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. PhD thesis, Fujitsu Laboratories Ltd, 1993. 2.3

[27] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *In The 5th International Conference on Learning Representations (ICLR)*, 2017. 4.3

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *In NIPS Deep Learning Workshop*, 2013. 1, 2.3

[29] 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):529–533, 2015. 1

[30] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016. 1, 2.2, 4, 4.1

[31] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1046–1054, 2016. 2.3, 4.2

[32] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks.

[33] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

[34] Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006. 3.3.1

[35] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008. 2.3

[36] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. 4.1

[37] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988. 1

[39] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017. 1, 4.2

[40] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1889–1897, 2015. 1, 1, 4.2

[41] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. 1, 4.2

[42] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Policy gradients with parameter-based exploration for control. In *International Conference on Artificial Neural Networks*, pages 387–396. Springer, 2008. 3.3.1

[43] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 3.3.1

[44] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 1, 2.3

[45] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[46] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38 (3):287–308, 2000.

[47] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998. 2.3

[48] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. 1999. 2.1, 2.2

[49] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000. ACM, 2009.

[50] Aviv Tamar, Sergey Levine, Pieter Abbeel, YI WU, and Garrett Thomas. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2146–2154, 2016. 1

[51] Philip S. Thomas. Bias in natural actor-critic algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–441–I–448. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3044856`.

[52] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. 1

[53] William Uther and Manuela Veloso. Adversarial reinforcement learning. Technical report.

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

[55] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1995–2003, 2016.

[56] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *In The 5th International Conference on Learning Representations (ICLR)*, 2017. 1, 2.2, 4.2

[57] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013. A.1

[58] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances*

in Neural Information Processing Systems, pages 2746–2754, 2015. 1

[59] Paweł Wawrzyński. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.

[60] Jason D Williams and Steve Young. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2):393–422, 2007.

[61] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 1, 3

[62] Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[63] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[64] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270, 2011. 3.3.1