Université de Montréal

Stabilizing Q-Learning for Continuous Control

par

David Yu-Tung Hui

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

December 31, 2022

Résumé

L’apprentissage profond par renforcement a produit des décideurs qui jouent aux échecs, au Go, au Shogi, à Atari et à Starcraft avec une capacité surhumaine. Cependant, ces algorithmes ont du mal à naviguer et à contrôler des environnements physiques, contrairement aux animaux et aux humains. Manipuler le monde physique nécessite la maîtrise de domaines d’actions continues tels que la position, la vitesse et l’accélération, contrairement aux domaines d’actions discretes dans des jeux de société et de vidéo. L’entraînement de réseaux neuronaux profonds pour le contrôle continu est instable : les agents ont du mal à apprendre et à conserver de bonnes habitudes, le succès est à haute variance sur hyper-paramètres, graines aléatoires, même pour la même tâche, et les algorithmes ont du mal à bien se comporter en dehors des domaines dans lesquels ils ont été développés. Cette thèse examine et améliore l’utilisation de réseaux de neurones profonds dans l’apprentissage par renforcement. Le chapitre 1 explique comment le principe d’entropie maximale produit des fonctions d’objectifs pour l’apprentissage supervisé et non supervisé et déduit, à partir de la dynamique d’apprentissage des réseaux neuronaux profonds, certains termes régulisants pour stabiliser les réseaux neuronaux profonds. Le chapitre 2 fournit une justification de l’entropie maximale pour la forme des algorithmes acteur-critique et trouve une configuration d’un algorithme «acteur-critique» qui s’entraîne le plus stablement. Enfin, le chapitre 3 examine la dynamique d’apprentissage de l’apprentissage par renforcement profond afin de proposer deux améliorations aux réseaux «cibles» et «jumeaux» qui améliorent la stabilité et la convergence. Des expériences sont réalisées dans les simulateurs de physique idéale DeepMind Control, MuJoCo et Box2D.

Keywords: informatique; intelligence artificielle; apprentissage profond; apprentissage par renforcement; apprentissage par renforcement profond; contrôle; contrôle continu; Q-Learning; MuJoCo

Abstract

Deep Reinforcement Learning has produced decision makers that play Chess, Go, Shogi, Atari, and Starcraft with superhuman ability. However, unlike animals and humans, these algorithms struggle to navigate and control physical environments. Manipulating the physical world requires controlling continuous action spaces such as position, velocity, and acceleration, unlike the discrete action spaces of board and video games. Training deep neural networks for continuous control is unstable: agents struggle to learn and retain good behaviors, performance is high variance across hyperparameters, random seed, and even multiple runs of the same task, and algorithms struggle to perform well outside the domains they have been developed in. This thesis finds principles behind the success of deep neural networks in other learning paradigms and examines their impact on reinforcement learning for continuous control. Chapter 1 explains how the maximum-entropy principle produces supervised and unsupervised learning loss functions and derives some regularizers used to stabilize deep networks from the training dynamics of deep learning. Chapter 2 provides a maximum-entropy justification for the form of actor-critic algorithms and finds a configuration of an actor-critic algorithm that trains most stably. Finally, Chapter 3 considers the training dynamics of deep reinforcement learning to propose two improvements to target and twin networks that improve stability and convergence. Experiments are performed within the DeepMind Control, MuJoCo, and Box2D ideal-physics simulators.

Keywords: computer science; artificial intelligence; deep learning; reinforcement learning; deep reinforcement learning; control; continuous control; Q-Learning; MuJoCo

Contents

List of tables

List of figures

List of acronyms and abbreviations

CDF	Cumulative Distribution Function
CNF	Conditional Normalizing Flow
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Networks
DL	Deep Learning
DMC	DeepMind Control
DQN	Deep Q-Networks
DRL	Deep RL
DrQ	Data-Regularized Q-Learning
ELBO	Evidence-Based Lower Bound
GN	GroupNorm
IID	Independent and Identically Distributed
IQM	InterQuartile Mean
KL Divergence	Kullback-Leibler Divergence
LINEX	Linear-Exponential
LN	LayerNorm
MaxEnt	Maximum Entropy
MaxEnt RL	Maximum Entropy RL
MDP	Markov(ian) Decision Process

ML	Machine Learning
MuJoCo	Multi-Joint dynamics with Contact
NLL	Negative-Log-Likelihood
NTK	Neural Tangent Kernel
PDF	Probability Density Function
PPO	Proximal Policy Gradient
RealNVP	Real-valued Non-Volume Preserving transformations
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SL	Supervised Learning
TD2	Twin DDPG
TD3	Twin Delayed DDPG
TD	Temporal Difference
TN	Target Network
TRPO	Trust-Region Policy Gradient
UL	Unsupervised Learning
VAE	Variational Autoencoder
VQL	Variational Q-Learning

Acknowledgements

There are many I would like to thank for making the writing of this dissertation so enjoyable.

Glen Berseth, Pierre-Luc Bacon, Aaron Courville, and Irina Rish took the time to suggest minor corrections to this monograph through their role as my dissertation jury.

My co-supervisors Pierre-Luc Bacon and Aaron Courville helped and guided me to find and develop my taste in research projects, and provided a productive work environment through their respective leadership of friendly and inclusive research groups.

Yoshua Bengio gave me my first opportunity to research machine learning as a 2019 Mila intern, significantly broadening my appreciation of what might be possible with computation, while Dzmitry Bahdanau was also a formative influence on my perspective on machine learning during that period.

Many ideas in this dissertation have been shaped through numerous interactions with colleagues, especially but not exclusively Alan Chan, Anushree Rankawat, Breandan Considine, Max Schwarzer, Tianwei Ni, Paul Crouther, Sobhan Mohammadpour, David Kanaa, Niki Howe, Muawiz Chaudhary, Pierluca D'Oro, Padideh Nouri, Maxime Chevalier-Boisvert, Aneri Muni, Evgenii Nikishin, Olexa Bilaniuk, Valentin Thomas, Shawn Tan, Ionelia Buzatu, Laetitia Teodorescu, Alexandre Piché, Joseph Viviano, Salem Lahlou, Manuel Del Verme, Harry Mingde Zhao, Myriam Lizotte, Mohammad Reza Samsami, Hugo Sonnery, Nathan Schucher, Jonathan Pilault, Chin-Wei Huang, Disha Shrivastava, Julien Roy and Jean Harb. Alan Chan also helped with the ChatGPT French translations used in this dissertation.

I would like to express my immense gratitude towards my parents, Diana and Victor, who have always been a source of inspiration and encouragement to me. Their curiosity about my research was especially invaluable during my time away from Mila following the disruption caused by the COVID-19 pandemic.

I have wanted to research artificial intelligence since a young age and will spend the rest of my life happy I had the opportunity to do so alongside great colleagues, friends, and family.

Chapter 1

Background

This thesis improves the effectiveness of deep neural networks in reinforcement learning. Deep Neural Networks, called deep networks for short, were developed for supervised and unsupervised learning and have driven many empirical successes in image recognition [LB$^+$95, HZRS16, KSH17], image generation [KW13, GPAM$^+$14, DSDB16, SE19, HJA20], and language translation [BCB14, CVMG$^+$14]. However, the mathematics used in deep (supervised and unsupervised) learning is derived from different mathematical principles from that of reinforcement learning. Understanding the difficulties of using deep networks in reinforcement learning first requires understanding its effectiveness in deep learning. This chapter therefore first prevents an overview of deep learning with a particular emphasis on uncovering the first principles that make supervised and unsupervised learning so effective. Then, the mathematics of deep networks are introduced, alongside theoretical mathematical derivations for some commonly-used methods to regularize deep networks during training. Finally, reinforcement learning is introduced. This chapter provides theory that is later used in subsequent chapters and is not intended to be a comprehensive overview of reinforcement learning, deep learning, or deep networks. A more in-depth discussion of those topics may be found in [SB18a] and [GBC16].

1.1. Deep Learning

Machine Learning (ML) algorithms uncover implicit patterns in data. They are best targeted at problems where statistically-observed relationships are difficult to mathematically specify [LBH15]. There are three broad types of machine learning. **Unsupervised Learning (UL)** algorithms learn **summarize** datasets by compressing each datapoint into a short numerical description. They have most notably generated semantically-meaningful text [DCLT18, BMR$^+$20], images, [KW13, GPAM$^+$14, DSDB16, SE19, HJA20] and recently videos [HCS+22, HSG+22]. **Supervised Learning (SL)** algorithms *match* a datapoint to an underlying concept. They have been successfully applied to recognize an item in an image [LB+95, HZRS16, KSH17] and recognize a speaker from an audio file [RB18]. **Reinforcement Learning (RL)** algorithms make decisions that would be highly rewarded (scored). A major difference between (online) RL and supervised and unsupervised learning is the stationarity of the data it is learned from. Supervised and unsupervised learning (**Deep Learning (DL)**) problems are usually posed with a fixed dataset, while an RL agent and the environment would mutually affect each other, causing RL datasets to change constantly. This is the primary source of *instability* in RL[SB18b, SB18a].

Good learning algorithms uncover patterns from a dataset that **generalize** to arbitrary unseen samples of the population the dataset is sampled from. Such patterns must be resistant to noise and thus use *high-level* **representations** of concepts such as what object is the subject of a photo, what word is being spoken in an audio file, or what action is being performed by a robot [BCV13, GBC16]. A learning algorithm abstracts these representations from *low-level information* in each datapoint such as the value of a pixel in a photograph, the frequency spectrum of a certain timeframe within an audio file, or the voltage of a motor.

Consider representing a M-datapoint dataset with N discrete states. The number of datapoints M is typically much larger than the number of representing states N. If n_i datapoints have been assigned to state i, the total number of possible datapoint-state assignations (not caring about orderings as datapoints are sampled from a set) is:

$$\Omega = \frac{M!}{\prod_i n_i!}$$

A representation gives a more *distinguishable* description of a dataset if its states partition datapoints as evenly as possible. Distinguishability may be posed as an optimization problem to maximize the number of datapoints n_i associated with each i:

$$\arg\max_{\{n_i\}} \Omega = \arg\max_{\{n_i\}} \frac{M!}{\prod_i n_i!}$$

This optimization problem is hard because of the factorials. Stirling's Approximation approximates a log-factorial $\log k!$ as $k \log k - k$ and may be used if $\log \Omega$ is instead optimized. As the log function monotonically increases, it does not affect the optimizing variable n_i.

$$\begin{aligned} &\max \log \frac{M!}{\prod_i n_i!} \\ &= \max_{\{n_i\}} \left[\log M! - \sum_{i=1}^{N} \log n_i! \right] \end{aligned}$$

$$= \max_{\{n_i\}} \left[M \log M - M - \sum_{i=1}^{N} (n_i \log n_i - n_i) \right] \quad \text{Sterling's Approximation (twice)}$$

$$= \max_{\{n_i\}} \left[M \log M - 2M - \sum_{i=1}^{N} n_i \log n_i \right] \quad \text{where } \sum_{i}^{n} n_i = M$$

$$= \max_{\{n_i\}} \left[-\sum_i \frac{n_i}{M} \log \frac{n_i}{M} \right] \quad \text{removing non-dependent terms on } n \text{ and dividing by } M$$

$$= \max_{\{p_i\}} \left[-\sum_i p_i \log p_i \right] \quad \text{where } p_i = \lim_{M \to \infty} \frac{n_i}{M} \text{ is the probability of being in state } i$$

The resulting term inside the maximization $S = -\sum_i p_i \log p_i$ is **entropy**, which measures how well a representation distinguishes between datapoints, in other words how **informative** a representation would be of a datapoint [BN06], Section 1.6. **Differential Entropy** is the entropy of continuous representations, given by:

$$S = -\int p_i \log p_i \, \mathrm{d}i$$

Maximising the information of a representation is approximated by maximising S with respect to p: the **variational principle of maximum entropy (Maximum Entropy)** [Jay57a, Jay57b], an optimization problem given by:

$$\max_p \left[-\int p_i \log p_i \, \mathrm{d}i \right], \quad \text{subject to} \int p_i \, \mathrm{d}i = 1$$

where the constraint exists to sum p_i to a probability. Solving with Lagrange multipliers yields a uniform distribution.

This distribution partitions datapoints as evenly as possible, but each datapoint occurs with equal likelihood. Another constraint is needed for maximum entropy to produce a non-uniform distribution over datapoints. A constraint may be formed by assuming that each datapoint i is associated with a real-valued quantity E_i named **energy**. E is called an energy function because entropy was first developed to study the behavior of gases used in steam engines. Instead of considering datapoints and the probability they would appear, physicists studied gas molecules whose energy (measured by its temperature, proportional to the speed at which it traveled) increased with lower probability [Car90]. For historical purposes, this section takes mathematical notation from statistical physics.

Adapting the physics of thermodynamics to machine learning implies that samples that occur with lower probability would have higher energy [LCH$^+$06]. Specifying E is difficult and the success of a machine learning depends on whether Es may be well-specified. Maximizing entropy for systems with an energy constraint produces:

$$\begin{aligned} \max_p \quad & S = -\int p_i \log p_i \, \mathrm{d}i \\ \text{subject to} \quad & \int p_i E_i \, \mathrm{d}i = \bar{E} \\ \text{and} \quad & \int p_i \, \mathrm{d}i = 1 \end{aligned}$$

where $\bar{E}$ is a numerical value that determines the mean energy of the overall system. In physics, this would be the average temperature of the gas. Solving using the method of Lagrange multipliers yields a Lagrangian of:

$$L = S - \alpha \left(\int p_i \, \mathrm{d}p_i - 1 \right) - \beta \left(\int p_i E_i \, \mathrm{d}p_i - \bar{E} \right)$$

and the **Boltzmann Distribution**:

$$p_i = \frac{e^{-\beta E_i}}{\int e^{-\beta E_j} \, \mathrm{d}j} \tag{1.1.1}$$

In thermodynamics, the distribution over gas-particle velocities was empirically verified to follow a variant of the Boltzmann Distribution named the Maxwell-Boltzmann Distribution [Max60b, Max60a].

Different constraints with different energy functions yield different probability distributions.

$$\begin{aligned} \max_p \quad & S = -\int p_i \log p_i \, \mathrm{d}i \\ \text{subject to} \quad & \int p_i \, \mathrm{d}i = 1 \\ \text{and} \quad & \int p_i i \, \mathrm{d}i = \bar{\mu} \\ \text{and} \quad & \int (i - \mu)^2 \, \mathrm{d}i = \sigma \end{aligned}$$

yields the popularly used Gaussian distribution. All distributions whose functional forms are derived from maximum entropy form the **exponential family**, so named because the logarithm in entropy produces an exponential term in their probability functions.

1.1.1. Unsupervised Learning

Unsupervised Learning (UL) algorithms represent a dataset without needing human feedback, labels, or annotations. Maximum-Entropy Unsupervised Learning is posed as:

$$\min_p \quad \mathrm{H}[p(x)]$$

$$\begin{aligned} \text{subject to} \quad & \int p(x)\,\mathrm{d}x = 1 \text{ for all } x \\ \text{and} \quad & \int E_\theta(x)\,p(x)\,\mathrm{d}x = \bar{E} \\ \text{given} \quad & \theta = \arg\min_\theta \left[\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E} \right] \end{aligned} \tag{1.1.2}$$

The above constrained optimization problem finds a probability distribution given arbitrary energy function E. Hand-crafting a good energy function requires prior knowledge of the dataset and is of equivalent difficulty to hand-crafting an unsupervised learning algorithm. A good alternative is to instead learn an energy function that is most appropriate for the dataset [LCH+06]. A learnable energy function is specified by a deep network E_θ that inputs x and outputs a scalar parameter. θ is trained to minimize the discrepancy between learned distribution $p(x)$ and empirical distribution calculated over the dataset with Dirac functions δ:

$$d(x) = \frac{1}{n}\sum_{i=0}^{n} \delta(x - x_i)$$

using the empirical expected energy $\bar{E}$ over the dataset:

$$\bar{E} = \int E_\theta(x)\,d(x)\,\mathrm{d}x = \int E_\theta(x)\,\frac{1}{n}\sum_{i=0}^{n}\delta(x - x_i)\,\mathrm{d}x = \frac{1}{n}\sum_{i=0}^{n} E_\theta(x_i)$$

and $d(x)$ is an empirical distribution over the dataset given by Dirac functions, following its use in [VI19].

p and θ in Problem 1.1.2 may be solved by iteratively optimizing between θ and p. p may be found by the method of Lagrangian Multipliers, producing Lagrangian:

$$\begin{aligned} L(p,\alpha,\beta) = \mathrm{H}[p(x)] - \alpha\left(\int p(x)\,\mathrm{d}x - 1\right) - \beta\left(\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right) \\ \text{given} \quad \theta = \arg\min_\theta \left[\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right] \end{aligned}$$

and Boltzmann Distribution:

$$p(x) = \frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}$$

Analytically solving for β might be hard, depending on E_θ, so numerical optimization is preferred. The original constrained optimization problem (Problem 1.1.2) may be rewritten with (strong) Lagrangian duality as:

$$\min_{\alpha,\beta}\max_{p} L(p,\alpha,\beta)$$

Since $\min_p$ and $\max_\alpha$ have been found, the problem simplifies to just an optimization over β:

$$\begin{aligned}\min_{\alpha,\beta}\max_p L(p,\alpha,\beta) &= \min_\beta \min_\alpha \max_p L(p,\alpha,\beta)\\ &= \min_\beta J(\beta)\end{aligned}$$

where

$$J(\beta) = \min_\alpha \max_p L(p,\alpha,\beta)$$

Substituting the expression of p into L gives:

$$\begin{aligned}J(\beta) &= -\int p(x)\log\frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}\,\mathrm{d}x - \beta\left(\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right)\\
&= -\int p(x)\left(-\beta E_\theta(x) - \log\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'\right)\mathrm{d}x - \beta\left(\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right)\\
&= -\int p(x)\left(-\log\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'\right)\mathrm{d}x - \beta\bar{E} \quad \text{(cancel expectation of } \beta E_\theta\text{)}\\
&= \log\int e^{-\beta E_\theta(x')}\,\mathrm{d}x' - \beta\int E_\theta(x)\,d(x)\,\mathrm{d}x\\
&= \int\left(\log\int e^{-\beta E_\theta(x')}\,\mathrm{d}x' - \beta E_\theta(x)\right)d(x)\,\mathrm{d}x\\
&= -\int d(x)\log\frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}\,\mathrm{d}x\end{aligned}$$

simplifying Problem 1.1.2 to:

$$\begin{aligned}\min_\beta \quad J(\beta) &= \min_\beta\left[-\int d(x)\log\frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}\,\mathrm{d}x\right]\\
\text{given} \quad \theta &= \arg\min_\theta\left[\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right]\end{aligned}$$

This simplified problem may be optimized by block-coordinate descent that iterates between:

$$\begin{aligned}\beta &\leftarrow \beta - \eta\frac{\partial}{\partial\beta}\int d(x)\log\frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}\,\mathrm{d}x\\
\theta &\leftarrow \theta - \eta\left(\int E_\theta(x)\,p(x)\,\mathrm{d}x - \bar{E}\right)\end{aligned}$$

The gradient of the first step is:

$$\begin{aligned}&\frac{\partial}{\partial\beta}\left(-\int\log\frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'}\,d(x)\,\mathrm{d}x\right)\\
&= \frac{\partial}{\partial\beta}\int \beta E_\theta(x) + \log\int e^{-\beta E_\theta(x')}\,\mathrm{d}x'\,d(x)\,\mathrm{d}x\end{aligned}$$

$$= \int E_\theta(x)\, d(x)\, \mathrm{d}x + \frac{\partial}{\partial \beta} \log \int e^{-\beta E_\theta(x)}\, \mathrm{d}x$$

The second term in the above expression evaluates to:

$$\begin{aligned}
\frac{\partial}{\partial \beta} \log \int e^{-\beta E_\theta(x)}\, \mathrm{d}x &= \frac{\partial}{\partial \beta} \log \int e^{-\beta E_\theta(x)}\, \mathrm{d}x \\
&= \frac{1}{\int e^{-\beta E_\theta(x')}\, \mathrm{d}x'} \int \frac{\partial}{\partial \beta} e^{-\beta E_\theta(x)}\, \mathrm{d}x \\
&= -\frac{1}{\int e^{-\beta E_\theta(x')}\, \mathrm{d}x'} \int E_\theta(x) e^{-\beta E_\theta(x)}\, \mathrm{d}x \\
&= -\int E_\theta(x) \frac{e^{-\beta E_\theta(x)}}{\int e^{-\beta E_\theta(x')}\, \mathrm{d}x'}\, \mathrm{d}x \\
&= -\int E_\theta(x)\, p(x)\, \mathrm{d}x
\end{aligned}$$

thus:

$$\begin{aligned}
\frac{\partial}{\partial \beta} J(\beta) &= \int E_\theta(x)\, d(x)\, \mathrm{d}x - \int E_\theta(x)\, p(x)\, \mathrm{d}x \\
&= \int \frac{\partial}{\partial \beta} \beta\, E_\theta(x)\, d(x)\, \mathrm{d}x - \int \frac{\partial}{\partial \beta} \beta\, E_\theta(x)\, p(x)\, \mathrm{d}x
\end{aligned}$$

Note that by a similar argument the derivative of the second constraint is

$$\frac{\partial}{\partial \theta} J(\beta) = \int \frac{\partial}{\partial \theta} \beta\, E_\theta(x)\, d(x)\, \mathrm{d}x - \int \frac{\partial}{\partial \theta} \beta\, E_\theta(x)\, p(x)\, \mathrm{d}x$$

The derivatives with respect to β and θ are the same, with just a change of a parameter. Given that θ in E would be found by the chain rule and E is a scalar across real numbers, β is a redundant scale parameter. This eliminates the need for block-coordinate descent, simplifying the entire optimization problem into minimizing:

$$\begin{aligned}
J(\theta) &= -\int d(x) \log \frac{e^{E_\theta(x)}}{\int e^{E_\theta(x)}\, \mathrm{d}x'}\, \mathrm{d}x \\
&= -\int d(x) \log p_\theta(x)\, \mathrm{d}x \qquad (1.1.3) \\
&= -\int \frac{1}{n} \sum_{i=0}^{n} \delta(y - y_i) \log p_\theta(x)\, \mathrm{d}x \\
&= -\frac{1}{n} \sum_{i=0}^{n} \log p_\theta(x_i) \qquad (1.1.4)
\end{aligned}$$

The energy function is learned simultaneously as the probability distribution in **end-to-end learning**, a concept that has made deep learning so successful [LBH15]. This formula gives us some rules of thumb for designing maximum-entropy deep learning algorithms:

(1) The model should be probabilistic and a member of the exponential family.
(2) The model should optimize for a **cross-entropy** with the empirical distribution (Equation 1.1.3), resulting in *minimizing the* **negative log-likelihood (NLL)** of the model calculated over the dataset (Equation 1.1.4).

The duality between maximum likelihood and maximum entropy has previously been studied in [BDPDP96] but not extended to cover the case when the energy function is learned alongside the probability distribution end-to-end.

1.1.2. Normalizing Flows

Normalizing Flows are the most mathematically straightforward class of unsupervised learning method. More comprehensive introductions to them may be found in [KPB20, BTLLW21] They assume that each datapoint x is a complicated non-linear reparameterization of random variable z drawn from a multivariate unit Gaussian distribution.

$$z \sim N(\mathbf{0}, \mathbf{I})$$
$$x = g_\theta(z)$$

The **cumulative distribution function (cdf)** of x written in terms of z is:

$$P_\theta(x) = P(g_\theta(z))$$

However, Equation 1.1.4 uses a **probability distribution functions (pdf)**, defined as:

$$p_\theta(x) = p(g_\theta(z)) \left| \frac{\mathrm{d}}{\mathrm{d}z} g_\theta(z) \right|$$

If g_θ is not a bijection, then its derivative wrt z will be non-square and its determinant used in calculating the derivative may not exist. Bijectivity is ensured if g_θ is invertible. Invertibility enables *exact inference* of the probability of a sample through recovering $z = g_\theta^{-1}(x)$ and evaluating its pdf with respect to a multivariate unit Gaussian. θ may be found by

$$-\log p_\theta(x) = -\log p(g_\theta^{-1}(x)) - \log \left| \frac{\mathrm{d}}{\mathrm{d}x} g_\theta^{-1}(x) \right|$$

where $\frac{\mathrm{d}}{\mathrm{d}x} g_\theta^{-1}$ is the **log-volume fraction**.

1.1.3. Autoencoders

Autoencoders [Kra91, BYAV13, AB14] model $p_\theta(x)$ as a Gaussian distribution. Ignoring the normalization constant and standard deviation, the distribution is given by

$$p_\theta(x) \propto \exp - \sum_j (x^{(j)} - f_\theta(x)^{(j)})^2$$

When substituted into Equation 1.1.4, the log-likelihood simplifies into a per-input-element squared error:

$$\min_\theta \sum_j (x^{(j)} - f_\theta(x)^{(j)})^2$$

The deep network f_θ takes an x as input and *reconstructs* it, thus giving the autoencoder its name. Inputs that occur more in-distribution would be reconstructed better than images that are more out-of-distribution. The energy function in this distribution is the reconstruction error, which may be used to detect out-of-distribution samples, which have high reconstruction and are unlikely to occur for trained samples [Bis94].

It is difficult to estimate the probability of a sample with an autoencoder as it requires normalizing by the reconstruction error over all samples. **Variational Autoencoders (VAEs)** [KW13] abstract a datapoint x into a latent variable z whose probability may be estimated. These methods are **variational** because of the introduced latent variable z which needs to be integrated over in order to calculate $\log p(x)$. This integration is intractable, and thus the model cannot be trained with exact inference, thus requiring the **Evidence-Based Lower Bound (ELBO)** to approximate the probability with a lower bound:

$$\begin{aligned}
\log p(x) &= \log \sum_z p(x, z) \\
&= \log \sum_z \frac{q(z \mid x)}{q(z)} p(x, z) \\
&\geqslant \sum_z q(z \mid x) \log \frac{p(x, z)}{q(z)} \\
&= \sum_z q(z \mid x) \left(\log p(x \mid z) + \log \frac{p(z)}{q(z)} \right) \\
&= \mathop{\mathbb{E}}_{z \sim q(z|x)} [\log p(x \mid z)] + \mathbb{D}_{\mathrm{KL}}[q(z \mid x) \parallel p(z)]
\end{aligned}$$

The whole model is trained by first passing a datapoint x into the encoder q to produce a latent variable z, which is then passed to decoder p to reconstruct x. The KL-divergence between q is minimized with respect to a uniform Gaussian distribution $p(z)$, regularizing the learned distribution of q.

1.1.4. Supervised Learning

Supervised Learning produces functions that match **inputs** x with **labels** y. Typically, an input x is a vector, matrix, or tensor, while a label y is a scalar. We provide a detailed derivation for the loss function used in supervised learning from the maximum-entropy perspective, such that we can subsequently reuse this framework for reinforcement learning in Chapter 2. Maximum-Entropy Supervised Learning has a loss function of:

$$\max_\theta -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i \mid x_i) \tag{1.1.5}$$

For discrete y, the model has the form:

$$p_\theta(y_i \mid x) = \frac{e^{E_\theta(x,y_i)}}{\sum_j e^{E_\theta(x,y_j)}}$$

The tranformation of E to p shown above is done by the **Softmax** function [Bri89, Bri90]

The loss function of MaxEnt SL (Equation 1.1.5) is derived from:

$$\begin{aligned}
\max_p \quad & -\iint p(y \mid x)\, d(x) \log p(y \mid x)\, \mathrm{d}x\, \mathrm{d}y \\
\text{subject to} \quad & \int p(y \mid x)\, \mathrm{d}y = 1 \text{ for all } x \\
\text{and} \quad & \iint E_\theta(x,y)\, p(y \mid x)\, d(x)\, \mathrm{d}x\, \mathrm{d}y = \bar{E} \\
\text{given} \quad & \theta = \arg\min_\theta \left[\iint E_\theta(x,y)\, p(y \mid x)\, d(x)\, \mathrm{d}x\, \mathrm{d}y - \bar{E}\right]
\end{aligned} \tag{1.1.6}$$

with moment target constraint:

$$\begin{aligned}
\iint E_\theta(x,y)\, d(y \mid x)\, d(x)\, \mathrm{d}x\, \mathrm{d}y &= \iint E_\theta(x,y)\, d(x,y)\, \mathrm{d}y \\
&= \iint E_\theta(x,y)\, \frac{1}{n}\sum_{i=0}^{n} \delta(x-x_i)\, \delta(y-y_i)\, \mathrm{d}x\, \mathrm{d}y \\
&= \frac{1}{n}\sum_{i=0}^{n} E_\theta(x_i, y_i) \\
&= \bar{E}
\end{aligned}$$

and the empirical distributions:

$$d(x) = \frac{1}{n}\sum_{i=0}^{n} \delta(x-x_i)$$

$$d(x, y) = \frac{1}{n} \sum_{i=0}^{n} \delta(x - x_i)\delta(y - y_i)$$

The objective of Problem 1.1.6 is **conditional entropy**. It may be derived from noting that entropy is a special case of forward-Kullback-Leibler Divergence with respect to a uniform distribution. In

$$\min_p \left[- \int p(x) \log \frac{u(x)}{p(x)} \, \mathrm{d}x \right] = \max_p \left[- \int p(x) \log p(x) \, \mathrm{d}x \right]$$

the uniform distribution u does not contribute towards the maximization and is the same for all elements. The entropy of a conditional distribution follows the same forward-KL with respect to a uniform distribution:

$$\begin{aligned} \min_p \mathrm{D}_{\mathrm{KL}}[p(y \mid x)\, d(x) \mid\mid u(y \mid x)\, d(x)] &= \min_p \left[- \iint p(y \mid x)\, d(x) \log \frac{u(y \mid x)\, d(x)}{p(y \mid x)\, d(x)} \, \mathrm{d}x \, \mathrm{d}y \right] \\ &= \max_p \left[- \iint p(y \mid x)\, d(x) \log p(y \mid x) \, \mathrm{d}x \, \mathrm{d}y \right] \end{aligned}$$

1.2. Deep Neural Networks

Deep neural networks are used in supervised and unsupervised learning to parameterize energy functions that take a datapoint as input and outputs a scalar. This is challenging as datapoints are often in the format of a vector, matrix or tensor and may be very high dimensional. **Deep Neural Networks**, often called **Deep Networks** or **Neural Networks** are a parametric class of functions that have been mathematically proven to *approximate any function*, making them ideal for discovering an arbitrary energy function [HSW89, C+01]. As its name suggests, a neural network is composed of many **neurons**. This mathematical function performs an *affine* transformation of its input x that is then fed into a nonlinear **activation function** ρ.

$$\rho(w^\top x + b)$$

Neurons are composed in parallel and in sequence to form a network. A **layer** of a neural network are neurons in parallel, the number of which denotes its **width**. The **depth** of a neural network are sequentially composed layers.

The set of all ws and bs from all neurons are written as θ, the **parameters** of a neural network. θ is initialized to random values, and their numerical values are adjusted during **training**. During training, each parameter is adjusted to minimize a loss function L such as Equation 1.1.4 and Equation 1.1.5. An arbitrary parameter $w \in \theta$ is adjusted by **gradient**

descent by:

$$w_{t+1} \leftarrow w_t - \eta \, \frac{\partial}{\partial w} L(w) \Big|_{w_t}$$

The **learning rate** η is chosen to be small enough such that the change in w is small because the gradient is only accurate to a small region of w: the *local change* of the loss function. Parameters are updated until the loss **converges**. The number of gradient updates, or **timesteps**, is typically on the order of magnitude of the number of datapoints.

1.2.1. The Neural Tangent Kernel

Machine learning may uncover more patterns if more data is analyzed. The greater complexity of patterns resulting from an increased volume of data requires more powerful and more complex **function approximation** to uncover them, which may be achieved by *wider* and *deeper* neural networks with **more parameters** [KMH+20, HBM+22]. However, more powerful function approximation is prone to uncovering more patterns. It may **overfit** by learning patterns specific to only the dataset it has been trained on and not the wider population. Much of deep learning research has focused on **regularizing** the numerical value of parameters during training such that larger ones may be used. Popular regularization methods may be found by finding how they affect the convergence of training a neural network. This subsection investigates the convergence of supervised and unsupervised learning by introducing a strategy that will later be used in reinforcement learning.

Suppose a neural network $f_\theta(x)$ with parameters θ minimizes $L(f_\theta(x))$ with parameters θ:

$$\begin{aligned} \min_\theta L_\theta &= \min_\theta \frac{1}{n} \sum_{i=1}^{n} l_{\theta,i} \\ &= \min_\theta \frac{1}{n} \sum_{i=1}^{n} l_\theta(x_i) \end{aligned}$$

with gradient descent update rule

$$\delta\theta = -\frac{1}{n} \sum_{i=1}^{n} \nabla_\theta l_\theta(x_i)$$

Note that the change in weights is negative due to gradient descent. For convergence, the loss must decrease to a minimum, and L must therefore be constructed to be bounded from below. We further assume that L is a convex function. One method for the loss to converge is if, for every timestep, the change in loss $\delta L \leqslant 0$. The change in loss may be analytically quantified by a first-order Taylor expansion around θ with a small change of $\delta\theta$:

$$L_{\theta+\delta\theta} = L_\theta + \delta\theta^\top \nabla L_\theta + \cdots$$

which implies that:

$$\begin{aligned}
\delta L &\approx \delta\theta^\top \nabla L_\theta \\
&= \delta\theta^\top \frac{1}{m}\sum_{j=1}^{m} \nabla_\theta l_\theta(x_j) \\
&= -\left(\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta l_\theta(x_i)\right)^\top \frac{1}{m}\sum_{j=1}^{m} \nabla_\theta l_\theta(x_j) \\
&= -\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} \nabla_\theta l_\theta(x_i)^\top \nabla_\theta l_\theta(x_j)
\end{aligned}$$

For convergence,

$$-\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} \nabla_\theta l_\theta(x_i)^\top \nabla_\theta l_\theta(x_j) \leqslant 0$$

The strictest way to enforce this loss is if each component in the sum obeys:

$$\begin{aligned}
0 &< \nabla_\theta l_\theta(x_i)^\top \nabla_\theta l_\theta(x_j) \\
&= \frac{\mathrm{d}l_i}{\mathrm{d}f_\theta}\frac{\mathrm{d}l_j}{\mathrm{d}f_\theta} \nabla_\theta f_\theta(x_i)^\top \nabla_\theta f_\theta(x_j)
\end{aligned} \tag{1.2.1}$$

Now, due to the convexity of L, we have:

$$\begin{aligned}
l_i - l_j &\geqslant \frac{\mathrm{d}l_j}{\mathrm{d}f_\theta}(f_\theta(x_i) - f_\theta(x_j)) \\
l_j - l_i &\geqslant \frac{\mathrm{d}l_i}{\mathrm{d}f_\theta}(f_\theta(x_j) - f_\theta(x_i))
\end{aligned}$$

whose product yields

$$\begin{aligned}
& & -(l_i - l_j)^2 &\geqslant -\frac{\mathrm{d}l_i}{\mathrm{d}f_\theta}\frac{\mathrm{d}l_j}{\mathrm{d}f_\theta}(f_\theta(x_i) - f_\theta(x_j))^2 \\
\Longrightarrow & & \left(\frac{l_i - l_j}{f_\theta(x_i) - f_\theta(x_j)}\right)^2 &< \frac{\mathrm{d}l_i}{\mathrm{d}f_\theta}\frac{\mathrm{d}l_j}{\mathrm{d}f_\theta} \\
\Longrightarrow & & \frac{\mathrm{d}l_i}{\mathrm{d}f_\theta}\frac{\mathrm{d}l_j}{\mathrm{d}f_\theta} &> 0
\end{aligned}$$

Substituting this inequality in 1.2.1 yields the **Neural Tangent Kernel (NTK)** [JGH18]:

$$\nabla_\theta f_\theta(x_i)^\top \nabla_\theta f_\theta(x_j) > 0$$

1.2.2. Stabilizing Deep Learning

If $\rho(x)$ are the activations of the penultimate layer of a neural network with output $f_w(x) = w^\top \rho(x) + b$, the NTK simplifies into:

$$\rho(x_i)^\top \rho(x_j) > 0$$

This term is positive when the activation function on ρ is the **Rectified Linear Unit (ReLU)** [GBB11]:

$$\text{ReLU}(x) = \max(x, 0)$$

Activation functions such as the ReLU are typically applied throughout the network and not only on the penultimate layer.

The inequality involving the NTK also controls the rate of convergence – the greater the inequality, the faster the network should converge.

$$\rho(x_i)^\top \rho(x_j) > 0$$

However, faster convergence is not necessarily desirable. Fast (and premature) convergence to a suboptimal minimum may result in **underfitting** to a pattern that is not detailed enough to capture either the dataset or the population. To prevent premature convergence, the magnitude of ρ is standardized to a fixed value following:

$$\phi(x)_i = \frac{\rho_i - \mu(\rho)}{\sigma(\rho)} \odot w_i + b_i$$

for an arbitrary element ϕ_i of the activation vector ϕ. w and b form an affine transformation on the standardized vector that was intended to add more capacity to the normalization layer to distinguish between different elements, but this was later shown to cause overfitting [XSZ+19]. The standardized vector (without the affine portion) has a constant L2 norm with respect to the input:

$$\begin{aligned}
\|\phi(x)\|_2^2 &= \sqrt{\sum_{i=1}^{n} \phi(x)_i^2} \\
&= \sqrt{\sum_{i=1}^{n} \left(\frac{\rho_i - \mu(\rho)}{\sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\rho_i - \mu(\rho))^2}} \right)^2} \\
&= \sqrt{\sum_{i=1}^{n} (n-1) \frac{(\rho_i - \mu(\rho))^2}{\sum_{j=1}^{n} (\rho_j - \mu(\rho))^2}}
\end{aligned}$$

$$
\begin{aligned}
&= \sqrt{(n-1)\frac{\sum_{i=1}^{n}(\rho_i - \mu(\rho))^2}{\sum_{j=1}^{n}(\rho_j - \mu(\rho))^2}} \\
&= \sqrt{n-1}
\end{aligned}
$$

If normalization techniques are used, the magnitude of the NTK of a layer is bounded by n, the number of terms in normalization. This is crucially greater than 0, thus ensuring convergence.

$$
\begin{aligned}
\phi(x_i)^\top \phi(x_j) &\leqslant |\phi(x_i)^\top \phi(x_j)| \\
&= \|\phi(x_i)\|_2 \|\phi(x_j)\|_2 \\
&= n-1 \\
&> 0
\end{aligned}
$$

Different normalization layers select different quantities to standardize over. **LayerNorm** [BKH16] standardizes an entire layer, while **GroupNorm** [WH18] divides a layer into a fixed number of groups that are then standardized. The first normalization layer to be introduced was **BatchNorm** [IS15], which normalizes all elements of a group pooled across a minibatch used in gradient descent.

1.3. Reinforcement Learning

Reinforcement Learning (RL) agents make decisions that maximize a feedback signal. A human trains a reinforcement learning agent just as they might condition an animal in behavioral psychology. Good decisions are *rewarded* with a high numerical score and are reinforced. The learner, or agent, does not require any prior knowledge of the environment it is situated in, making RL a generally-applicable tool that may be used where it is unknown how the agent's decisions and the environment might mutually affect each other. RL is generally-applicable because it abstracts the decision-making process down to three mathematical concepts: an abstract description of the environment, the decision maker, and the feedback signal to be maximized. A good overview of RL may be found at [SB18b, SB18a].

Environments are described as a **Markov Decision Process (MDP)** [Bel57, How60], a structure that consists of *four quantities*. A decision process transitions between **states** s according to a sequence of **actions** a. The process is Markovian when **transition** probabilities between states $p(s_{t+1} \mid s_t, a_t)$ only depend on the state and action at the previous (discrete) timestep t. Starting states are sampled from starting distribution $p(s_0)$, while transitions to a new state returning a scalar **reward** $r(s_t, a_t, s_{t+1})$. For brevity, let $\tau_i = (s_i, a_i, s_{i+1})$ and $r_i = r(\tau_i)$.

As the environment is Markovian the **policy** may be specified by $\pi(a \mid s)$ as only the current state needs to be considered. If acting for n timesteps, π produces a **trajectory** $\tau_{i:n}$ in an arbitrary MDP with probability:

$$p_\pi(\tau_{0:n}) = p(s_0) \prod_{i=0}^{n} \pi(a_i \mid s_i)\, p(s_{i+1} \mid s_i, a_i)$$

This trajectory is associated with a discounted **return**

$$R(\tau_{i:n}) = \sum_{j=i}^{n} \gamma^j r_j$$

that scores the quality of a trajectory. A **discount factor** $0 < \gamma < 1$ is used in the return to ensure that the sum is finite when the number of timesteps considered, or **time horizon**, is infinite [Sam37].

The objective of RL is to find good policies that maximize the **expected return** over trajectories:

$$\mathop{\mathbb{E}}_{\tau_{0:n} \sim p_\pi(\tau_{0:n})}[R(\tau_{0:n})] = \int p_\pi(\tau_{0:n})\, R(\tau_{0:n})\, \mathrm{d}\tau_{i:n}$$

Optimization may be performed **offline** [LGR12, LKTF20] using analytic properties or empirical statistics of the MDP or **online** through trial and error and continual interaction with the MDP. This thesis focuses on continuous control environments where analytic properties and empirical statistics are difficult to collect, making online RL appropriate.

1.3.1. Deep Reinforcement Learning

Sample efficiency is the metric of how quickly the expected return is maximized with as few samples of interaction with the environment (samples) as possible. The efficiency of learning a good policy decreases for MDPs with increasingly large state spaces because more trial and error exploration in the environment is needed to find a good solution. This is especially true for continuous control environments where the state space is functionally infinite because it is a set of continuous quantities: positions, velocities, and accelerations. Learning becomes more efficient if the policy generalizes previously made good decisions to similar states. As deep networks have shown powerful generalization in supervised and unsupervised learning, **Deep Reinforcement Learning** (DRL) specifies a policy π_θ as a neural network with parameters θ. A naïve way to find good θ is to maximize the expected return with gradient ascent:

$$\theta \leftarrow \theta + \eta\, \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}[R(\tau_{0:n})]$$

The gradient of the path integral cannot be computed analytically because in the online setting $p(s_{t+1} \mid s_t, a_t)$, $r(a_t, s_t)$ and $p(s_0)$ and thus their derivatives are unknown. The gradient is instead estimated by the **Policy Gradient Theorem** [SMSM99]:

$$\begin{aligned}
&\nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}[R(\tau_{0:n})] \\
&= \nabla_\theta \int p_{\pi_\theta}(\tau_{0:n})\, R(\tau_{0:n})\, \mathrm{d}\tau_{i:n} \\
&= \int p_{\pi_\theta}(\tau_{0:n})\, \nabla_\theta \log p_{\pi_\theta}(\tau_{0:n})\, R(\tau_{0:n})\, \mathrm{d}\tau_{i:n} \quad \text{(log-derivative)} \\
&= \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}[R(\tau_{0:n})\, \nabla_\theta \log p_{\pi_\theta}(\tau_{0:n})] \\
&= \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}\left[R(\tau_{0:n})\, \nabla_\theta \log \left(p(s_0) \prod_{i=0}^{n} \pi_\theta(a_i \mid s_i)\, p(s_{i+1} \mid s_i, a_i)\right)\right] \\
&= \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}\left[R(\tau_{0:n}) \sum_{i=0}^{n} \nabla_\theta \log \pi_\theta(a_i \mid s_i)\right] \\
&= \nabla_\theta \mathop{\mathbb{E}}_{\tau_{0:n} \sim p_{\pi_\theta}(\tau_{0:n})}\left[\sum_{i=0}^{n} R(\tau_{i:n})\, \nabla_\theta \log \pi_\theta(a_i \mid s_i)\right] \quad \text{(Markov property of return)}
\end{aligned}$$

Training a policy using Policy Gradients is unstable because the expectation requires many Monte-Carlo samples of trajectories from the environment. Insufficient samples result in inaccurate expected return estimates and unstable policy training. Popular policy-gradient algorithms such as Trust-Region Policy Gradient (TRPO) [SLA+15] and Proximal Policy Gradient (PPO) [SWD+17] stabilize training by preventing the policy parameters from changing too quickly. However, these algorithms are still inefficient because previous trajectories from previous policy rollouts cannot be re-used to evaluate the current policy. If a new policy produces a previous action with low probability, its log-probability will approach infinity, producing large gradients and introducing instability in gradient ascent. Instead, samples have to be **on-policy**: they are always sampled from the current policy. Discarding samples every gradient update is inefficient.

Model-Based Methods are an **off-policy** method that stores samples from all (previous) policies to train a policy. Samples are stored in a **replay buffer** [Lin92] and re-sampled to train a model that estimates components of the underlying MDP:

$$\begin{aligned}
&\max_{\phi_2} \log p_{\phi_2}(s_0) \\
&\max_{\phi_1} \log p_{\phi_1}(s_{t+1} \mid s_t, a_t) \\
&\min_{\phi_3} \|r_t - r_{\phi_3}(a_t, s_t)\|
\end{aligned}$$

Reusing samples give model-based methods superior sample efficiency over policy-gradient methods. The policy's expected discounted return may be estimated by the model without needing evaluation in the environment:

$$\nabla_\theta \int \left(p_{\phi_1}(s_0) \prod_{i=0}^{n} \pi_\theta(a_i \mid s_i)\, p_{\phi_2}(s_{i+1} \mid s_i, a_i) \right) R(\tau_{0:n})\, \mathrm{d}\tau_{i:n}$$

Model-based methods such as are computationally inefficient because their transition model $p_{\phi_1}(s_{t+1} \mid s_t, a_t)$ requires modeling the probability of the entire next state, which might contain noise or irrelevant distractors the policy would not consider when making decisions. Popular methods such as World Models [HS18], PlaNet[HLF+19], and Dreamer [HLBN19] model the evolution of a latent representation of the state rather than the entire state. Recent research has focused on improving latent representations to remove all components that are not **decision-aware** [Aba20, NAAB22]. Decision-aware information should only be predictive of the return, as this is the *only* information the policy should process. Decision-aware model-based approaches include DeepMDP [GKB+19] and MuZero [SAH+20] but have a more complex implementation.

1.3.2. Model-Free Reinforcement Learning

Model-free methods are sample efficient and decision aware because they use a replay buffer to directly estimate the expected return for policy π from a state input with a **value function** defined by:

$$V_\pi(s_i) = \int p(\tau_{i:n} \mid s_i)\, R(\tau_{i:n})\, \mathrm{d}\tau_{i:n}$$

A recursive definition for the value function is provided by the **Bellman-Consistency Equation**:

$$\begin{aligned}
V_\pi(s_i) &= \int p(\tau_{i:n} \mid s_i)\, R(\tau_{i:n})\, \mathrm{d}\tau_{i:n} \\
&= \int \prod_{j=i}^{n} \pi(a_j \mid s_j)\, p(s_{j+1} \mid s_j, a_j) \left(r_i + \sum_{j=i+1}^{n} \gamma^{j-i} r_j \right) \mathrm{d}\tau_{i:n} \\
&= \iint \prod_{j=i}^{n} \pi(a_j \mid s_j)\, p(s_{j+1} \mid s_j, a_j) \left(r_i + \sum_{j=i+1}^{n} \gamma^{j-i} r_j \right) \mathrm{d}\tau_i\, \mathrm{d}\tau_{i+1:n} \\
&= \iint \pi(a_i \mid s_i)\, p(s_{i+1} \mid s_i, a_i) \Big(r_i + \gamma V_\pi(s_{i+1}) \Big) \mathrm{d}\tau_i \\
&= \mathop{\mathbb{E}}_{\substack{a_i \sim \pi(\cdot \mid s_i) \\ s_{i+1} \sim p(\cdot \mid s_i, a_i)}} r_i + \gamma V_\pi(s_{i+1})
\end{aligned}$$

The state-action-value function, or **Q-function** is defined by

$$\begin{aligned} Q_\pi(s_i, a_i) &= \int p(\tau_{i:n} \mid s_i, a_i) \sum_{j=i}^{n} \gamma^{j-i} r_j \, \mathrm{d}\tau_{i:n} \\ &= \mathop{\mathbb{E}}_{s_{i+1} \sim p(\cdot|s_i, a_i)} [r_i + \gamma V_\pi(s_{i+1})] \end{aligned}$$

and has Bellman-Consistency Equation

$$Q_\pi(s_i, a_i) = \mathop{\mathbb{E}}_{s_{i+1} \sim p(\cdot|s_i, a_i)} \left[r_i + \gamma \mathop{\mathbb{E}}_{a_{i+1} \sim \pi(\cdot|s_i)} Q_\pi(s_{i+1}, a_{i+1}) \right]$$

Let $V^\star$ and $Q^\star$ be the value and Q-function for the optimal policy $\pi^\star$. An optimal policy would *always* act to maximize expected return and follow $\pi^\star(s) = \max_a Q^\star(s, a)$. The optimal policy has value and Q-functions that follow the **Bellman (Optimality) Equation**:

$$\begin{aligned} Q^\star(s_t, a_t) &= \mathop{\mathbb{E}}_{s_{i+1} \sim p(\cdot|s_i, a_i)} \left[r(s_t, a_t) + \max_a Q^\star(s_{t+1}, a) \right] \\ V^\star(s_t) &= \max_a \mathop{\mathbb{E}}_{s_{i+1} \sim p(\cdot|s_i, a_i)} [r(s_t, a) + V^\star(s_{t+1})] \end{aligned}$$

where $V^\star$ is the optimal value function. The nomenclature of Bellman Consistency and Bellman Optimality is taken from [AJKS19], which goes into more mathematical detail. A value and Q-function may be updated by **bootstrapping**: using the **target value** found on the left-hand side to calculate the new **online value** on the right-hand side. Empirical estimates for the expectation may be calculated with a replay buffer. A related but useful concept is the **Temporal-Difference (TD) error** calculated by subtracting the RHS of a Bellman equation from the LHS.

Algorithms that *evaluate* a policy with a value function are known as **actor-critic** systems where the policy is the **actor** and the **critic** is the value function [SMSM99, MBM$^+$16]. As a policy may be more easily derived from a Q-function than a value function, Q-learning is often the choice of critic.

1.3.3. Deep Q-Learning

Deep Q-learning approximates the state-action-value function with a neural network Q_θ. Parameters θ are trained to minimize the discrepancy between itself and a predicted value given by the Bellman Equation of the optimal Q-function. Denoting $y(s_i, a_i, s_{i+1}) = r_i + \gamma \max_a Q^\star(s_{i+1}, a)$ for brevity,

$$\min_\theta \mathop{\mathbb{E}}_{s_i, a_i} \left[\left(Q_\theta(s_i, a_i) - \mathop{\mathbb{E}}_{s_{i+1} \sim p(\cdot|s_i, a_i)} [y(s_i, a_i, s_{i+1})] \right)^2 \right]$$

$$
\begin{aligned}
&= \min_\theta \left[\mathop{\mathbb{E}}_{s_i,a_i} \left[\mathop{\mathbb{E}}_{s_{i+1}\sim p(\cdot|s_i,a_i)} \left[(Q_\theta(s_i,a_i) - y(s_i,a_i,s_{i+1}))^2 \right] + \mathop{\mathbb{V}}_{s_{i+1}\sim p(\cdot|s_i,a_i)} [y(s_i,a_i,s_{i+1})] \right] \right] \\
&= \min_\theta \mathop{\mathbb{E}}_{s_i,a_i,s_{i+1}} \left[(Q_\theta(s_i,a_i) - y(s_i,a_i,s_{i+1}))^2 \right] \\
&= \min_\theta \mathop{\mathbb{E}}_{s_i,a_i,s_{i+1}} \left[\left(Q_\theta(s_i,a_i) - r_i - \gamma \max_a Q^\star(s_{i+1},a) \right)^2 \right]
\end{aligned}
$$

This loss is not computable for two reasons. First, the outer expectation over s_i, a_i should be computed with respect to the stationary distribution induced by the optimal policy. As this would require many samples of interaction, (s, a, r, s') samples, where s and s' are stored in a replay buffer and used to approximate the stationary distribution:

$$\min_\theta \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(Q_\theta(s,a) - r - \gamma \max_{a'} Q^\star(s',a') \right)^2 \right]$$

Secondly, the target $Q^\star$ is unknown, so is replaced by current estimate Q_θ. When using gradient descent methods, gradients should not be taken through the target θ just as no gradients would pass through $Q^\star$. This method of passing gradients through an expression evaluated with two neural networks is called **semi-gradient descent**. The stop-gradient operator is applied to target parameters θ and denoted by $\bar{\theta}$, yielding the **Deep Q-Networks (DQN)** loss [MKS+15], where this derivation was introduced in the appendix:

$$\min_\theta \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(Q_\theta(s,a) - r - \gamma \max_{a'} Q_{\bar{\theta}}(s',a') \right)^2 \right]$$

In continuous action spaces, $\max Q_{\bar{\theta}}(s', a')$ may be found by gradient ascent on a':

$$a' \leftarrow a' + \eta \frac{\partial}{\partial a'} Q_{\bar{\theta}}(s',a')$$

Gradient ascent typically takes multiple iterations (around 20 for continuous control) until convergence, so another neural network π_ϕ trained by:

$$\phi = \arg\max_\phi Q_\theta(s, \pi_\phi(s))$$

approximates $\arg\max_a Q_\theta(s,a)$. Multiple steps of gradient ascent are amortized by interleaving gradient steps of ϕ and θ and training them together, yielding the **Deep Deterministic Policy Gradients (DDPG)**, Algorithm 1 [LHP+15]:

1.3.4. Stabilizing Deep Q-Learning

Two empirically-found regularization methods are popular in continuous control. The first deals with instability in the Q-function's learning target as it uses a target estimate of itself to compute a bootstrapped learning target. The objective for training the Q-function is constantly changing and *unstable*. Large changes to θ, the parameters of the Q-function, impact

Algorithm 1 DDPG

```
while training do
    Obtain action $a = \pi_\phi(s)$ from current state $s$
    Get reward $r$ and next state $s'$ from environment
    Store $(s, a, r, s')$ in replay buffer
    $s \leftarrow s'$
    Sample a batch $(s_i, a_i, r_i, s'_i)_{1:n}$ from replay buffer
    Update critic parameters $\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \frac{1}{n} \sum_{i=1}^{n} (Q_\theta(s_i, a_i) - (r_i + \gamma Q_{\bar{\theta}}(s'_i, \pi_\phi(s'_i))))^2$
    Update actor parameters $\phi \leftarrow \phi + \eta \frac{\partial}{\partial \phi} \frac{1}{n} \sum_{i=1}^{n} Q_\theta(s_i, \pi_\phi(s_i))$
while testing do
    Obtain action $a = \pi_\phi(s)$ from current state $s$
    Get next state $s'$ from environment
    $s \leftarrow s'$
```

the targets, sometimes occasionally leading to divergence when the Q-function becomes increasingly larger, tending to infinity and producing an inaccurate assessment of the actor. To stabilize target Q-values, DQN froze the parameters of target networks $\bar{\theta}$ for a fixed number of timesteps and periodically updated them [MKS+15, HMVH+18]. As the periodic update might still produce a large change to the target values, DDPG updated target networks by instead taking a moving average over parameters, whose update speed is determined by the size of $0 < \alpha \geqslant 1$:

$$\hat{\theta} \leftarrow (1 - \alpha)\hat{\theta}_{\text{old}} + \alpha\theta$$

This yields a critic learning loss of:

$$\min_\theta \mathop{\mathbb{E}}_{s,a,r,s'} \left[\Big(Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\bar{\hat{\theta}}}(s', a') \Big)^2 \right]$$

Stabilizing the target parameters with an exponential moving average is synechdocally referred to as **target networks** [LHP+15].

The second regularizer deals with inaccuracies in target Qs due to noise Y. Noise may come from a stochastic environment, function approximation inaccuracies, or even incomplete data in the replay buffer that may not be representative of the environment [TS93, VHGS16]. Noisy target Q-values given by:

$$\hat{Q}(s, a) = Q(s, a) + Y(s, a)$$

produce the following expected error in bootstrapped targets:

$$\mathbb{E}\left[\Big(r(s, a) + \gamma \max_a \hat{Q}(s, a) \Big) - \Big(r(s, a) + \gamma \max_a Q(s, a) \Big) \right]$$

$$= \mathbb{E}\left[\gamma\left(\max_a \hat{Q}(s,a) + \max_a Q(s,a)\right)\right]$$
$$= \gamma\mathbb{E}\left[\max_a Y(s,a)\right]$$

Even if Y has zero mean, positive components will be selected by $\max_a$, leading to *systematic overestimation* of the bootstrapped targets.

Many methods have been developed to tackle overestimation. **Double Q-Learning** [Has10] recognized that the source of overestimation originated from selecting the maximal Q-value: $\max_a Q(s,a)$. Two Q-functions are instead used, and each Q function evaluates an action that would maximize the other:

$$Q_1(s,a) \leftarrow r + \gamma Q_2\left(s', \arg\max_{a'} Q_1(s',a')\right)$$
$$Q_2(s,a) \leftarrow r + \gamma Q_1\left(s', \arg\max_{a'} Q_2(s',a')\right)$$

Noise in one Q-function is unlikely to be correlated with another, and learning would not exploit bad beliefs in either Q-function, preventing the Q-function from learning inaccuracies about itself.

Double DQN (DDQN) [VHGS16] updates Double Q-Learning to the deep learning setting, where the online and target Q-networks function as the double Qs. However, only the online Q-network is updated:

$$\left(Q_\theta(s,a) - r - \gamma Q_{\bar{\theta}}(s', \arg\max_{a'} Q_\theta(s',a'))\right)^2$$

as the target network parameters are updated by an exponential moving average.

Continuous control is more prone to overestimated targets because bootstrapped targets are evaluated by a policy by $Q_\theta(s, \pi_\phi(s))$, and ϕ has been trained to maximize Q. The continuous space of actions increases the severity of which Q might be exploited by maximization, and DDQN is shown to still produce overestimated targets. **Twin Networks** [FHM18] train an ensemble of two Q-functions and selects the minimum target of the two as the bootstrap target:

$$\sum_{j\in\{1,2\}}\left(Q_{\theta_j}(s,a) - r - \gamma \min_{i\in\{1,2\}} Q_{\overline{\theta_i}}(s',a')\right)^2$$

1.4. Summary

Unsupervised, supervised and reinforcement learning are learning algorithms but are motivated by very different principles. Unsupervised and supervised learning use loss functions derived from maximum entropy on a fixed dataset. In contrast, reinforcement learning uses loss functions that maximize the expected return from a continually-changing dataset, causing instability. Chapter 2 investigates the intersection of reinforcement learning and the maximum-entropy principle, given its success in unsupervised and supervised learning.

Function approximation introduces overestimation in reinforcement learning. Deep networks, a particularly powerful class of function approximation, might be more prone to overestimation when used in deep reinforcement learning. Theoretical motivations behind regularizers used in deep networks were introduced after considering the training dynamics of deep learning. Their application to the training dynamics of deep reinforcement learning is discussed in Chapter 3.

Chapter 2

Scaling Continuous Control

In the last decade, deep neural networks trained with supervised and unsupervised learning have produced many advances such as identifying 1000 objects[HZRS16, KSH17], translating text between languages [BCB14, CVMG^{+}14], recognizing speech [RB18] and generating coherent prose, images [KW13, GPAM^{+}14, DSDB16, SE19, HJA20] and lately videos [HCS^{+}22, HSG^{+}22] given textual inputs. The success of deep supervised and unsupervised learning has been largely attributed to *scaling* the amount of data provided to a learning algorithm [RWC^{+}19, KMH^{+}20, BMR^{+}20, HBM^{+}22]. More data contains more statistical relationships to be uncovered, provided the learning algorithm – the deep neural network – is strong enough. A limiting factor to the strength of the deep neural network is its **number of parameters**. Increasing the parameters of a deep neural network increases its complexity and ability to approximate complex functions more accurately by using more parameters to partition the dataset more evenly [SK22, BDK^{+}21]. In certain natural language processing tasks, scaling up the parameters of a deep neural network yields *empirically predictable* improvements in performance. The effectiveness of scaling parameters is limited by the size of the dataset the model is being trained on when the number of parameters becomes sufficiently large and redundant [KMH^{+}20, HBM^{+}22]. Although conventional wisdom suggests that models with many parameters might overfit, these studies instead provide a counterexample that against overfitting in large models.

Data ought to be the limiting factor for online reinforcement learning (hereafter RL). Unlike supervised and unsupervised learning, RL does not assume access to a fixed dataset. Instead, the agent gathers data through *interacting* with an environment, which may be expensive and time-consuming. The quality of an RL algorithm is often judged by **sample efficiency**: whether good behaviors may be learned from as few samples of interactions with the environment as possible. Sample efficiency may be improved by scaling. A scaled-up deep neural network learns faster, producing more informative interactions that collect better data, ultimately perpetuating a *virtuous cycle* of learning faster.

Within continuous control, model-free RL uses comparatively small neural networks to supervised or unsupervised learning. These networks consist of two hidden layers of width 256 and an output layer, totaling fewer than 10^6 parameters [LHP+15, FHM18, HZAL18]. This is *several orders of magnitudes* smaller than neural networks used in image processing and natural language processing. We investigate performance gains in reinforcement learning with scaled-up deep neural networks. Experiments are performed from state observations rather than pixel observations for speed and ease of experimentation within the DeepMind Control (DMC) [TDM+18, TMD+20] and MuJoCo [TET12, BCP+16] environments.

2.1. Scaling A Maximum-Entropy Model

Unlike supervised and unsupervised learning, loss functions in RL are not derived from the variational principle of maximum entropy (Chapter 1). We hypothesize that the principle of maximum entropy leads to beneficial scaling behaviors due to the beneficial numerical properties of the maximum-entropy framework. Entropy without any constraints, given by $-\sum_i p_i \log p_i$, is maximized by a uniform probability distribution. A loss function derived from the maximum entropy principle would favor a uniform distribution whose values would be all of similar magnitude. Larger models are more expressive and may be highly nonlinear or discontinuous. More uniform distributions increasingly avoid singularities or zeros, increasing their numerical stability during training. In the absence of information to distinguish between samples – which might be increasingly the case for large datasets – a uniform distribution would not favor one sample over another. Moreover, the *Pitman–Koopman–Darmois theorem* [Fis34, Pit36, Koo36, Dar35] states that the exponential family are the only distributions that summarize an arbitrary amount of **independent and identically distributed (iid)** datapoints using a finite number of parameters, and that members of the exponential family are derived from the maximum-entropy principle. We thus investigate the scaling properties of an RL algorithm derived from the principle of maximum entropy.

First, as before in Chapter 1, we set up a maximum-entropy Lagrangian:

$$\begin{aligned} \max_{p} \quad & \mathrm{H}[\pi(a \mid s)] \quad \text{(over all } s\text{)} \\ \text{subject to} \quad & \int \pi(a \mid s)\,\mathrm{d}a = 1 \quad \text{(over all } s\text{)} \\ \text{and} \quad & \int Q_\theta(s,a)\,\pi(a \mid s)\,\mathrm{d}a = \bar{Q} \quad \text{(over all } s\text{)} \end{aligned}$$

$$\text{given} \quad \theta = \arg\min_\theta \left[Q_\theta(s,a) - \bar{Q}\right] \quad (\text{over all } s)$$

The objective is the conditional entropy for the policy and not the entropy over the entire trajectory because the task is Markovian. Similarly, the energy function considered should only take a state-action tuple, resulting in the Q-function. Here, $\bar{Q}$ is the numerical value obtained from the Bellman (optimality) equation:

$$\bar{Q} = \iint p(s' \mid s,a)\, \pi(a \mid s) \left(r(s,a,s') + \gamma \max_{a'} Q_\theta(s',a')\right) \mathrm{d}a \, \mathrm{d}s'$$

Unlike supervised or unsupervised learning respectively introduced in Subsections 1.1.4 and 1.1.1, $\bar{Q}$ is not an empirical value calculated with respect to a reference distribution, because the reference distribution would have to be an optimal policy or the policy we would like to learn, which is unknown. As before, solving the Lagrangian of

$$\mathrm{H}[\pi(a \mid s)] - \alpha \left(\int \pi(a \mid s)\, \mathrm{d}a - 1\right) + \tau \left(\int Q_\theta(s,a)\, \pi(a \mid s)\, \mathrm{d}a - \bar{Q}\right)$$

yields a Boltzmann distribution (Equation 1.1.1) over Q-values:

$$\pi(a \mid s) = \frac{e^{\tau Q_\theta(s,a)}}{\int e^{\tau Q_\theta(s,a')}\, \mathrm{d}a'}$$

Here, τ is positive because actions should be taken that *maximize* the expected discounted return approximated by the Q-function. However, τ may not be subsumed and optimized into θ, because unlike previous supervised or unsupervised learning, $\bar{Q}$ is not an empirical value that can be calculated or estimated over a fixed dataset.

A practical RL algorithm would update a Q-function by one gradient step following:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(Q_\theta(s,a) - r - \gamma \max_{a'} Q_{\bar{\theta}}(s',a')\right)^2\right]$$

before acting according to the maximum-entropy policy with τ as a hyperparameter.

$$\pi(a \mid s) = \frac{e^{\tau Q_\theta(s,a)}}{\int e^{-\tau Q_\theta(s,a')}\, \mathrm{d}a'}$$

Acting according to the Boltzmann distribution over the Q-function is **Boltzmann Exploration**, a classic exploration method which is well-discussed in [CBGLN17], where a lower τ increases randomness in decision-making, which might lead to more exploration through more coverage of the environment.

Boltzmann Exploration is mainly used in discrete control where the integral in the **partition function** (the denominator $\int e^{\tau Q_\theta(s,a')}\, \mathrm{d}a'$) simplifies into a sum that is computable. In continuous control, the integral is not easy to compute and π is instead approximated by another neural network π_ϕ. The objective for training π_ϕ is chosen to be the reverse-KL divergence to remove the partition function from the optimization objective:

$$\begin{aligned}\min_{\phi} \mathrm{D}_{\mathrm{KL}}[\pi_\phi \mid\mid \pi] &= \min_{\phi} \int_a \pi_\phi(a \mid s) \log \frac{\pi_\phi(a \mid s)}{\pi(a \mid s)} \,\mathrm{d}a \\ &= \max_{\phi} \left[\mathrm{H}[\pi_\phi(a \mid s)] + \int_a \pi_\phi(a \mid s) \log \frac{e^{\tau Q_\theta(s,a)}}{\int e^{\tau Q_\theta(s,a')} \,\mathrm{d}a'} \,\mathrm{d}a \right] \\ &= \max_{\phi} \left[\mathrm{H}[\pi_\phi(a \mid s)] + \int_a \pi_\phi(a \mid s) \, \tau Q_\theta(s,a) \,\mathrm{d}a \right]\end{aligned}$$

which may be rewritten as:

$$\max_{\phi} \mathop{\mathbb{E}}_{a \sim \pi_\phi(a|s)} [\tau Q_\theta(s,a) - \log \pi_\phi(a \mid s)] \tag{2.1.1}$$

and trained by gradient ascent:

$$\phi \leftarrow \phi + \eta \mathop{\mathbb{E}}_{a \sim \pi_\phi(a|s)} [\tau Q_\theta(s,a) - \log \pi_\phi(a \mid s)]$$

Typically, the objective is divided by τ and the learning rate η is adjusted accordingly. Letting $\beta = \frac{1}{\tau}$ derives the objective for training the actor:

$$\phi \leftarrow \phi + \eta \mathop{\mathbb{E}}_{a \sim \pi_\phi(a|s)} [Q_\theta(s,a) - \beta \log \pi_\phi(a \mid s)]$$

The resultant actor-critic algorithm that was derived from the maximum-entropy principle has previously been introduced to the community as **SACLite** (Algorithm 2) [YZX22], which was presented as an empirically-found successor to the SAC [HZAL18, HZH+18] algorithm.

Algorithm 2 SACLite

```
while training do
    Obtain action $a = \pi_\phi(s)$ from current state $s$
    Get reward $r$ and next state $s'$ from environment
    Store $(s, a, r, s')$ in replay buffer
    $s \leftarrow s'$
    Sample a batch $(s_i, a_i, r_i, s'_i)_{1:n}$ from replay buffer
    Update critic parameters $\theta \leftarrow \theta - \eta \frac{\partial}{\partial \theta} \frac{1}{n} \sum_{i=1}^{n} (Q_\theta(s_i, a_i) - (r_i + \gamma Q_{\bar{\theta}}(s'_i, \pi_\phi(s'_i))))^2$
    Update actor parameters $\phi \leftarrow \phi + \eta \frac{\partial}{\partial \phi} \frac{1}{n} \sum_{i=1}^{n} Q_\theta(s_i, \pi_\phi(s_i)) - \beta \log \pi_\phi(a \mid s)$
while testing do
    Obtain action $a = \pi_\phi(s)$ from current state $s$
    Get next state $s'$ from environment
    $s \leftarrow s'$
```

2.1.1. Maximum-Entropy Reinforcement Learning

Maximum-Entropy Reinforcement Learning (MaxEnt RL) is not the same as an RL algorithm derived from the variational principle of maximum entropy. MaxEnt RL regularizes the discounted return objective of RL with entropy [HTAL17, Lev18]:

$$\mathop{\mathbb{E}}_{\tau_{0:n} \sim p_\pi(\tau_{0:n})} \left[\sum_{i=0}^{n} \gamma^i (r(\tau_i) - \beta \log \pi(a_i \mid s_i)) \right]$$

A policy that solves MaxEnt RL must maximize entropy alongside the expected discounted return. The goal is to produce a policy that is as uniform as possible such that all actions are equally likely to be selected and the environment will be explored and covered better. **Soft Actor Critic (SAC)** [HZH+18] is a MaxEnt RL algorithm and has a critic loss of:

$$\min_\theta \mathop{\mathbb{E}}_{s,a,r,s'} \left[\left(Q_\theta(s,a) - r - \gamma \mathop{\mathbb{E}}_{a \sim \pi_\phi(a|s)} [Q_{\bar{\theta}}(s',a') - \beta \log \pi_\phi(a' \mid s')] \right)^2 \right]$$

and the same policy loss as SACLite that was derived in the previous section (Equation 2.1.1). The authors of SACLite empirically found that removing the entropy regularizer in the SAC critic resulted in more stable training. Removing the entropy regularizer has also been shown to be more stable for offline RL [KFTN21, KZTL20, KFTN21].

SAC (and SACLite) utilize automatically tune the value of β by gradient descent on β to optimize:

$$\min_\beta \mathop{\mathbb{E}}_{s} \left[\mathop{\mathbb{E}}_{a \sim \pi_\phi(a|s)} \left[\bar{H} - \beta \log \pi_\phi(a \mid s) \right] \right] \tag{2.1.2}$$

with gradient descent in a process called **automatic entropy tuning**. Here, **target entropy** $\bar{H}$ is a fixed hyperparameter, typically chosen to be the logarithm of a uniform distribution over the action dimension that *analytically maximizes* the entropy regularizer and would be the smallest numerical value for $\log \pi_\phi$ [HZH+18].

2.1.2. Types of Probabilistic Actors

In continuous control, the action space is n-dimensional, with each dimension ranging between -1 and 1. Policies or actors in the actor-critic algorithm take a state input and output an action vector where each element is between -1 and 1 as well as the corresponding probability for taking that action. In addition to scaling up actors by increasing their parameters, actors may be scaled up by increasing the complexity of their probability distribution.

The simplest policy of all has a **fixed variance** and was introduced in **Deep Deterministic Policy Gradients (DDPG)**. The **deterministic policy** first samples a latent variable $z = \tanh(x)$. Noise sampled from a Gaussian distribution with fixed variance is added to these outputs, which are then clipped to -1 and 1, yielding the action. The distribution of the DDPG distribution is a conditional truncated Gaussian distribution with a state-input-dependent mean. The fixed variance of this distribution has a fixed entropy, which simplifies the actor loss of DDPG to:

$$\min_{\phi} \mathbb{E}_{s}\left[\mathbb{E}_{a\sim\pi_\phi(a|s)}\left[Q_\theta(s,a)\right]\right]$$

The policy in SACLite (and hence SAC) is a conditional **squashed Gaussian** distribution with state-dependent mean and variance. First, a latent variable z is sampled from a conditional Gaussian with learned mean $\mu_\phi(s)$ and variance $\sigma_\phi(s)$ produced by a neural network with parameters ϕ with input state s. z has the same dimensionality as the output action a and a hyperbolic tangent function applied to z squashes the outputs to -1 and 1. Like a normalizing flow (Subsection 1.1.2), this changes the volume of the support from the space of real numbers of the Gaussian distribution to -1 and 1, thus requiring a change in probability density. The Squashed Gaussian policy and its probability density is given by:

$$\log \pi_\phi(a \mid s) = \log N_{\mu_\phi(s),\sigma_\phi(s)}(z) + \log(1-a)^2,$$
$$\text{where } a = \tanh(z) \text{ and } z \sim N_{\mu_\phi(s),\sigma_\phi(s)}$$

A **Conditional Normalizing Flow (CNF)** policy introduces a more complex reparameterization of the Gaussian latent variable z with learned parameters. The normalizing flow layers condition the transformation of z to a and vice versa conditional on state-input s [WWHW19]. This is denoted by $z = g_\phi^{-1}(a \mid s)$ and $a = g_\phi(z \mid s)$. We choose to use RealNVP layers because of their simplicity to implement and effectiveness [DSDB16], and also because it has previously been applied to reinforcement learning in [WSB19, SLZ$^+$20, HHAL18]. Scaling the number of RealNVP layers increases the depth and complexity of reparameterization.

$$\log p_\phi(a \mid s) = \log p(g_\phi^{-1}(a,z)) + \log\left|\frac{\mathrm{d}}{\mathrm{d}z} g_\phi^{-1}(a \mid s)\right|, \quad \text{where } a = g_\phi(z \mid s) \text{ and } z \sim N_{\mu_\phi(s),\sigma_\phi(s)}$$

Both the Squashed Gaussian and CNF policy may be trained with Equation 2.1.1.

(Conditional) normalizing flows require sampling from a latent variable with the same dimensionality as the target space (Section 1.1.2) [WWHW19, KPB20]. More complex normalizing flows may be attained from deeper transformations, but the amount of randomness remains the same. Increased randomness and increased exploration may result from a latent variable sampled from higher-dimensional distribution than the resultant action. Exact inference of this method cannot be performed and this has to be done instead by variational inference

(Section 1.1.3). The **Variational Q-Learning (VQL)** policy uses a latent variable:

$$\begin{aligned}\pi_\phi(a \mid s) &= \int \pi_\phi(a, z \mid s)\,\mathrm{d}z \\ &= \int \pi_{\phi_1}(a \mid z, s)\,\pi_{\phi_2}(z \mid s)\,\mathrm{d}z\end{aligned}$$

resulting in a two-stage policy that samples an action by first sampling a latent action before then using it as a context variable to sample the observed action. Typically, in variational inference, a lower bound for the log-probaility would be found and maximized. In RL, $\log \pi$ is maximized in Equation 2.1.1 and it does not make sense to then minimize a the lower bound. A lower bound to $-\log \pi$ must instead be constructed. This may be achieved by subtracting a non-negative KL-divergence constraining $\pi(z \mid a, s)$ to a prior distribution $q(z \mid a, s)$:

$$\begin{aligned}-\mathop{\mathbb{E}}_{a\sim\pi(a|s)}[\log \pi(a \mid s)] &\geqslant -\mathop{\mathbb{E}}_{a\sim\pi(a|s)}[\log \pi(a \mid s) + \mathrm{D}_{\mathrm{KL}}[\pi(z \mid a, s) \mid\mid q(z \mid a, s)]] \\ &= -\mathop{\mathbb{E}}_{a\sim\pi(a|s)}\left[\log \pi(a \mid s) + \mathop{\mathbb{E}}_{z\sim\pi(z|a,s)}[\log \pi(z \mid a, s) - \log q(z \mid a, s)]\right] \\ &= -\mathop{\mathbb{E}}_{\substack{z\sim\pi(z|a,s)\\ a\sim\pi(a|s)}}[\log \pi(a \mid s) + \log \pi(z \mid a, s) - \log q(z \mid a, s)] \\ &= -\mathop{\mathbb{E}}_{\substack{z\sim\pi(z|s)\\ a\sim\pi(a|z,s)}}[\log \pi(a \mid z, s) + \log \pi(z \mid s) - \log q(z \mid a, s)]\end{aligned}$$

pi_{ϕ_2} and q_{ϕ_3} would be conditional Gaussian distributions of the same dimensionality, while pi_{ϕ_1} is a squashed Gaussian distribution. The actor objective for **Variational Q-Learning (VQL)** is:

$$\max_\phi \mathop{\mathbb{E}}_{\substack{z\sim\pi_{\phi_2}(z|s)\\ a\sim\pi_{\phi_1}(a|z,s)}}[Q_\theta(s, a) - \log \pi_{\phi_1}(a \mid z, s) - \log \pi_{\phi_2}(z \mid s) + \log q_{\phi_3}(z \mid a, s)]$$

and trained by sampling z from π_{ϕ_2}, a from π_{ϕ_1} before the above objective is calculated. This lower bound to $-\log \pi$ has previously been used to train hierarchical variational models [RTB16].

2.2. Scaling Experimental Setup

We investigate performance gains for the maximum-entropy actor-critic family of algorithms when both the actor and the critic are scaled up.

Scaling up the critic may be done by increasing the number of parameters in its neural network. In reinforcement learning from pixel observations within DMC, [BGW21b] found that scaling up the DrQ [KYF20] and DrQv2 [YFLP21] algorithms yielded sample-efficiency improvements. Algorithms were scaled up with a residual neural network architecture, while Spectral Norm [MKKY18] was applied to all linear layers apart from the first and last to stabilize the gradients.

Our experiments adapt the architecture of [BGW21b] for state observations following Figure 2.1. The critic is scaled by increasing the number of blocks in Figure 2.1. We found that applying spectral norm to the critic resulted in no learning at all, so removed it from our experiments. The addition of the input of the block to the output is reminiscent of the skip-connection technique in [HZRS16], which has been shown to improve network performance by increasing the number of ways information may propagate from input to output [VWB16, AN16].

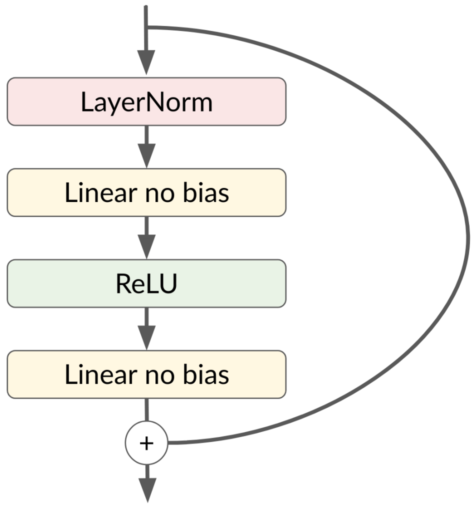

Fig. 2.1. Architecture of a residual block adapted from continuous control from pixel observations from [BGW21b] to continuous control from states

Scaling the actor is investigated by comparing the performance improvements between a deterministic policy, squashed Gaussian policy, conditional normalizing flow (CNF) policy and variational policy, which we name Variational Q-Learning (VQL). The deterministic and squashed Gaussian policies are scaled up with the residual blocks just like the critic, while the CNF policy is scaled by increasing the number of residual layers and VQL is scaled by increasing the dimensionality of its latent variable.

Experiments were carried out in the MuJoCo environment (with state observations) because this was the environment SAC was developed in. Good and reliable performance within that environment is therefore expected and scaling up the algorithm might lead to better performance. Experiments are performed in the HalfCheetah, Ant, and Humanoid domains for a mixture of problems of easy, medium, and hard difficulties. DMC with state observations is used as a second environment to validate whether scaling claims found in MuJoCo hold.

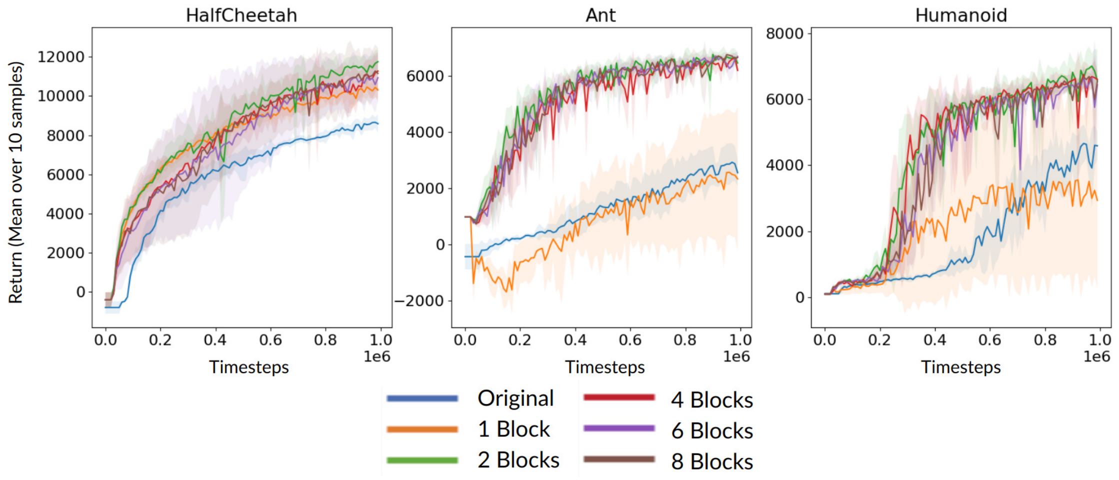

Fig. 2.2. Scaling the critic with varying numbers of residual blocks described in Figure 2.1. The original critic is a neural network with two hidden layers of 256. A deterministic policy is used.

As suggested by [HIB$^+$18], ten seeds per environment were used to calculate aggregate statistics. Experiments were run for a million timesteps. The policy was tested every 1000 timesteps and the average return from ten rollouts was recorded. The small neural networks used in the following algorithms had two hidden layers of size 256 and ReLU activations [GBB11] as standard. The blocks of [BGW21b] have two linear layers: the first has width 2048 and the second 1024. Automatic entropy tuning is used throughout unless stated otherwise, and the fixed standard deviation of the deterministic policy was set to 0.2.

2.3. MuJoCo Results

Figures 2.2 indicate that scaling the critic works up to two blocks before performance saturates. Table 2.1 show a significant increase between one and two blocks before further increases do not result in many changes. In experiments with scaling the actor, a large critic with four blocks was used as this is comfortably larger than two blocks so any performance increases with the large critic should be down to the actor.

Scaling the actor was ineffective. Figures 2.3, 2.4, 2.5, and 2.6 did not show any improvement when each type of actor was scaled. The only improvements shown in Figures 2.3 and 2.4 were down to scaling the critic. Moreover, Table 2.2 was constructed by taking the best result from all scaling configurations of each policy class and showing that no class of policy was significantly better than the others.

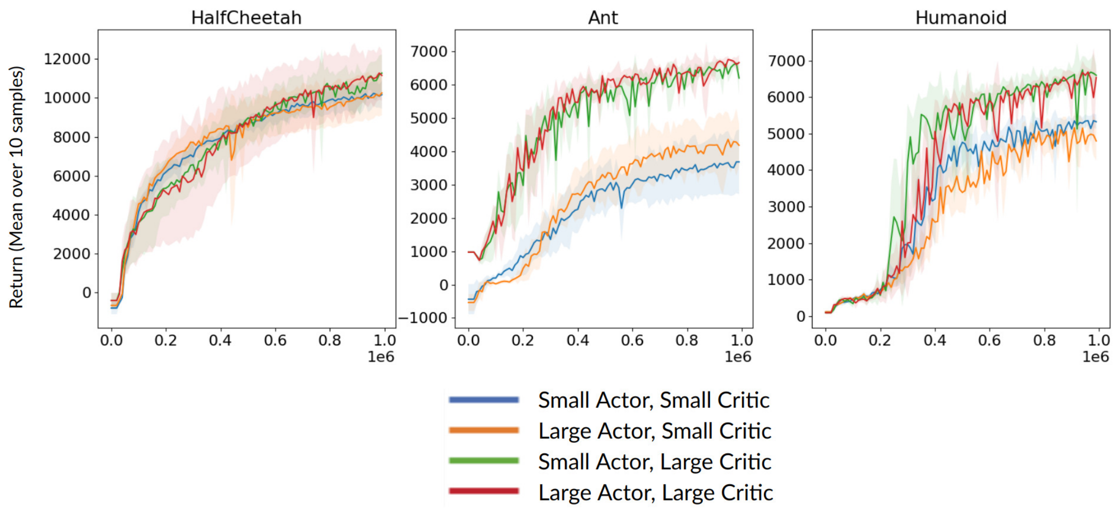

Fig. 2.3. Scaling the deterministic policy. Large refers to a network with four blocks described in Figure 2.1 and small refers to a two-hidden-layer neural network of width 256.

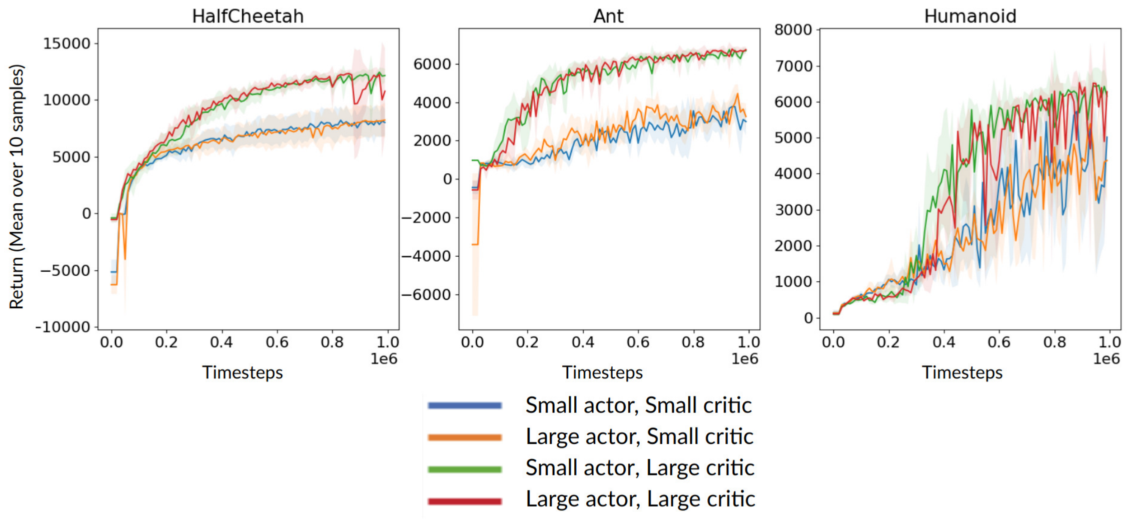

Fig. 2.4. Scaling the Squashed Gaussian policy. Large refers to a network with four blocks described in Figure 2.1 and small refers to a two-hidden-layer neural network of width 256.

Scale	**HalfCheetah-v3**	**Ant-v3**	**Humanoid-v3**
Original	8590 ± 454	2550 ± 459	459 ± 425
1 Block	10300 ± 595	2330 ± 2550	2930 ± 2680
2 Blocks	11800 ± 621	6460 ± 351	6570 ± 949
3 Blocks	11200 ± 966	6200 ± 490	6600 ± 276
4 Blocks	10900 ± 1190	6700 ± 50.6	6430 ± 481
5 Blocks	11200 ± 1290	6670 ± 224	6540 ± 196

Table 2.1. Return averaged over 10 seeds of 10 rollouts after training for a million timesteps for with different scales with different-sized critics. Corresponds to a table of Figure 2.2. Mean ± standard error is given to three significant figures.

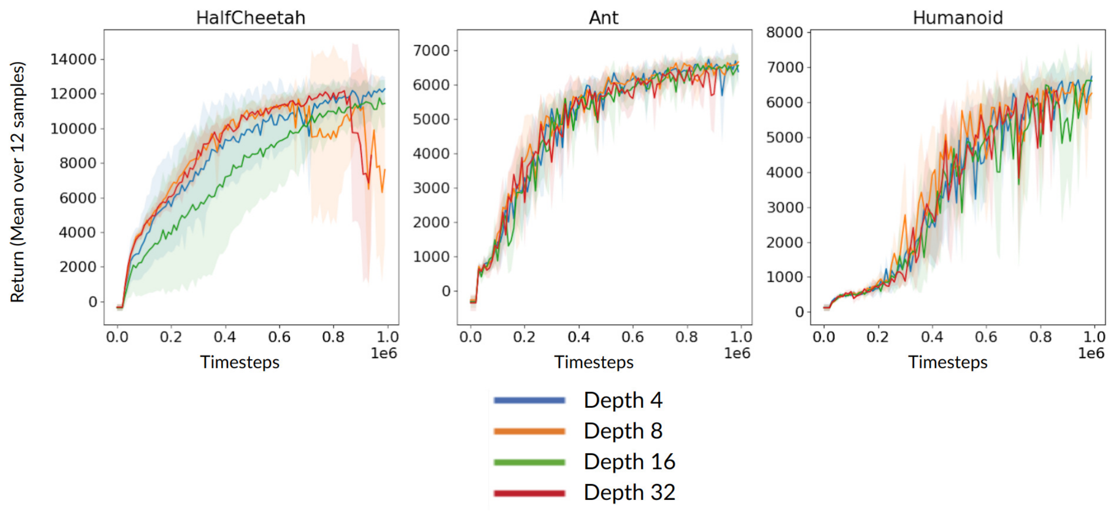

Fig. 2.5. Scaling the Conditional Normalizing Flow policy with increasing depth of Real-NVP layers. A large critic with four blocks described in Figure 2.1 was used.

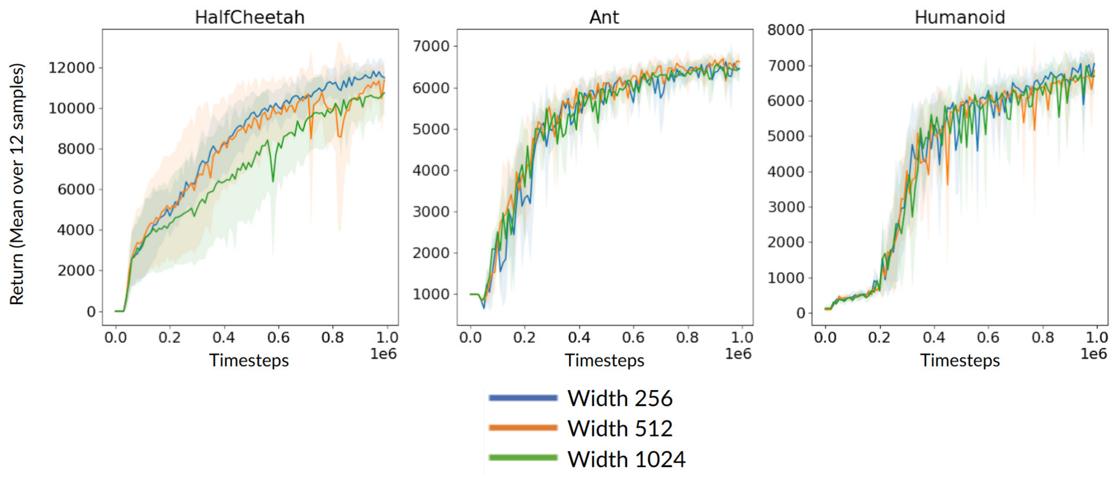

Fig. 2.6. Scaling the size of the latent variable used in Variational Q-learning. A large critic with four blocks described in Figure 2.1 was used.

Environment	Det. Policy	Squashed	CNF	VQL
HalfCheetah-v3	11300 $\pm$ 966	12200 $\pm$ 609	12300 $\pm$ 675	115 $\pm$ 452
Ant-v3	6670 $\pm$ 224	6720 $\pm$ 154	6660 $\pm$ 217	6630 $\pm$ 228
Humanoid-v3	6600 $\pm$ 276	6150 $\pm$ 429	6740 $\pm$ 536	7040 $\pm$ 322

Table 2.2. Comparing the return averaged over 10 seeds of 10 rollouts after training for a million timesteps. Numbers in each entry were selected from the best-performing configuration for each type of policy. Mean $\pm$ standard error numbers are given to three significant figures.

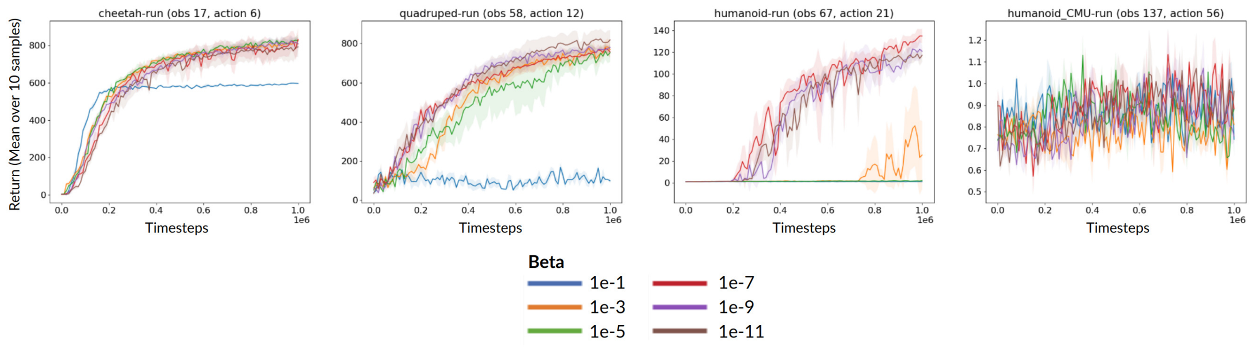

Fig. 2.7. The effect of decreasing β on learning in four DMC environments.

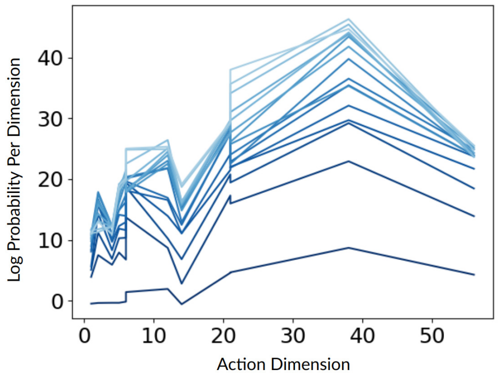

Fig. 2.8. Per-dimension change in $\log \pi_\phi(a \mid s)$ from 1 to 20 gradient steps on ϕ increase. Lighter lines indicate more gradient steps have been taken

2.4. DeepMind Control Results

This section investigates whether a larger critic produces better results in DMC. There is also no reason not to use a more complicated policy than the squashed Gaussian or deterministic policies commonly used.

Automatic entropy tuning of the squashed Gaussian policy (Equation 2.1.2) is ineffective in DMC. Figure 2.7 shows that higher dimensional DMC environments train better when β is set to increasingly small values. Figure 2.8 shows that the average $\log \pi_\phi(a \mid s)$ per element of a increases with the number of gradient steps and the action dimension.

As the best value of β might vary between environments, we instead use a deterministic policy. Figure 2.9 and Table 2.3 show that a deterministic policy is competitive with a squashed Gaussian policy, regardless of autotuning.

Finally, Figure 2.10 showed that scaling up the critic with a deterministic policy is shown to be ineffective. Five seeds were used in this experiment due to the computational cost of running the large critic over twelve DMC tasks.

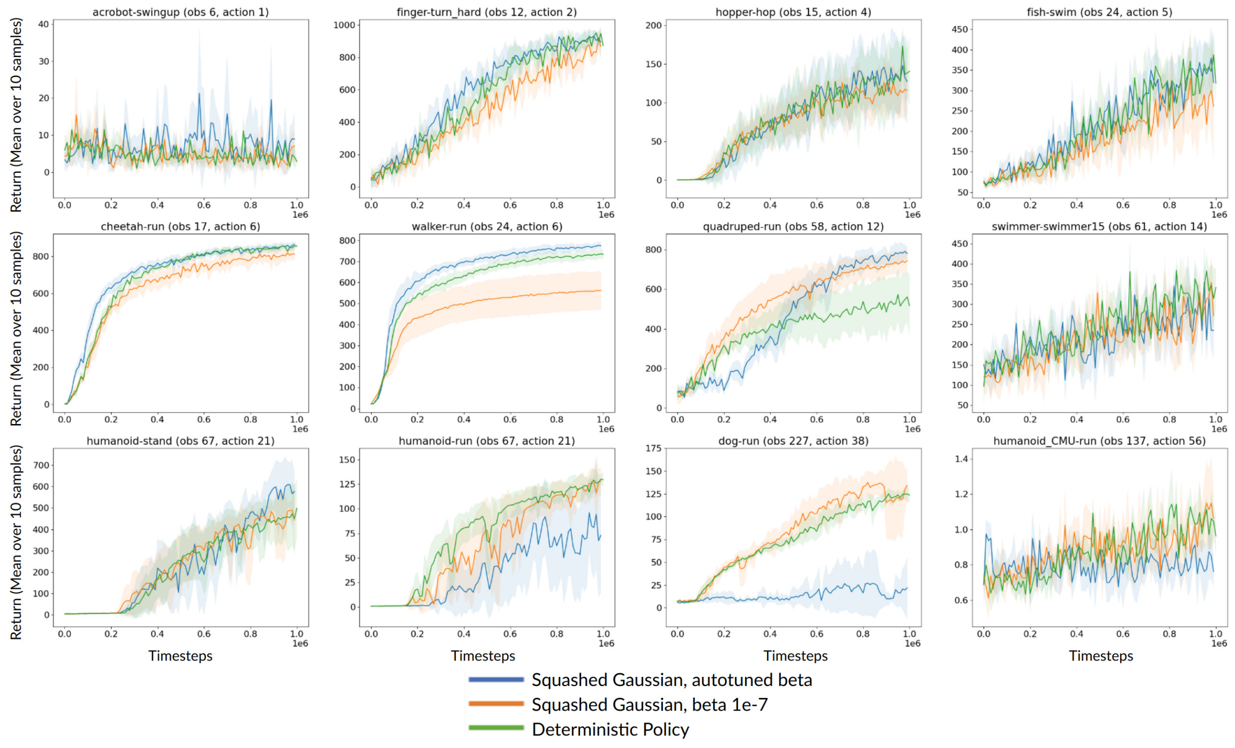

Fig. 2.9. Comparison of autotuned and fixed $\beta = 1e^{-7}$ squashed Gaussian policies and deterministic policies used with a small critic of two hidden layers of size 256.

Environment	Autotuned	Fixed	Deterministic
acrobot-swingup	8.91 ± 7.24	7.01 ± 3.96	2.89 ± 0.673
finger-turn_hard	894 ± 91.2	868 ± 79.3	872 ± 4.36
hopper-hop	127 ± 52.6	116 ± 27.5	141 ± 44.5
fish-swim	320 ± 88.4	262 ± 129	318 ± 48.7
cheetah-run	864 ± 14.4	812 ± 35	856 ± 9.39
walker-run	775 ± 17.8	562 ± 90.7	734 ± 21.9
quadruped-run	780 ± 33.6	745 ± 52.3	516 ± 153
swimmer-swimmer15	235 ± 68.1	272 ± 51.5	341 ± 46.4
humanoid-stand	577 ± 84.4	444 ± 173	499 ± 135
humanoid-run	73 ± 58.7	128 ± 13.6	130 ± 8.22
dog-run	21.8 ± 33.7	134 ± 17.1	124 ± 7.6
humanoid_CMU-run	0.762 ± 0.0857	1.06 ± 0.141	0.963 ± 0.111

Table 2.3. Return averaged over 10 seeds of 10 rollouts after training for a million timesteps with autotuned and fixed $\beta = 1e^{-7}$ squashed Gaussian policies and deterministic policies used with a small critic of two hidden layers of size 256. Mean ± standard error numbers are given to three significant figures.

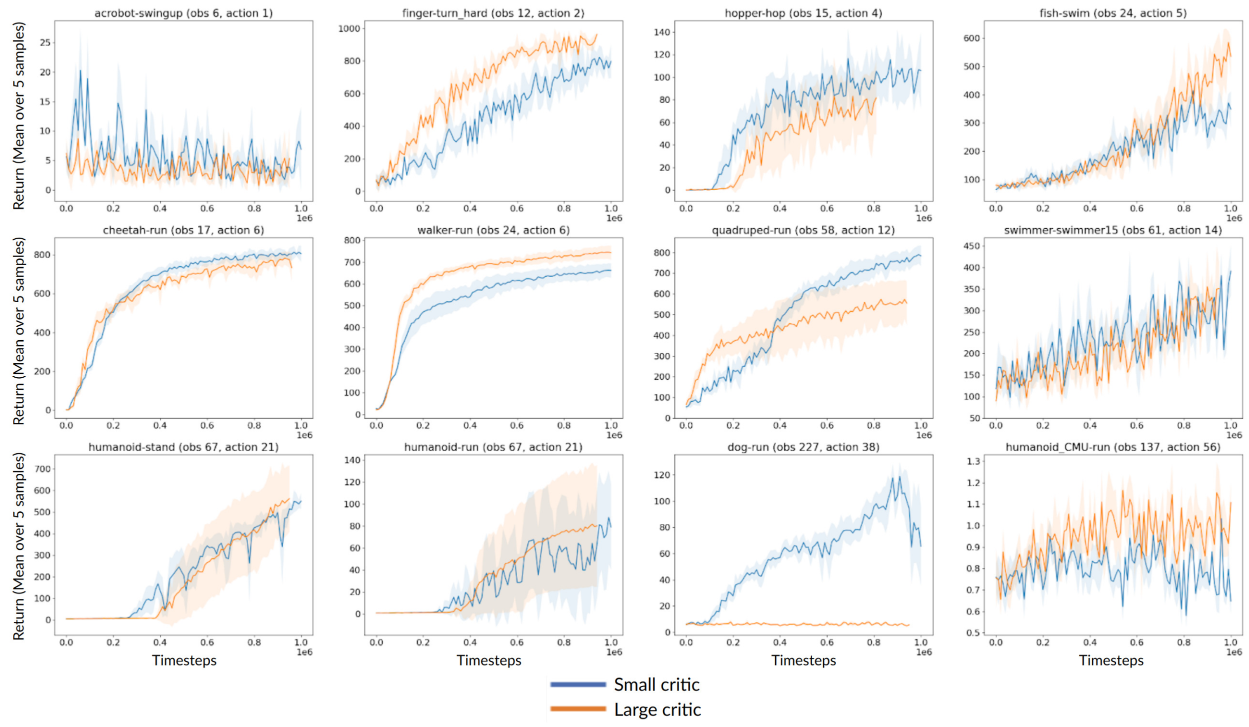

Fig. 2.10. Scaling the critic with varying numbers of residual blocks described in Figure 2.1. A large critic consisted of four blocks described in Figure 2.1, while a small critic had a neural network with two hidden layers of size 256. A deterministic policy was used.

2.5. Related Work

[YFLP21] also found that a deterministic policy was more stable than a squashed Gaussian but in the visual domain of DMC from pixels. The squashed Gaussian experienced the same entropy collapse as in Figure 2.8, but it was not specified whether the collapse was more likely with increased action dimensions.

[SBSG20] found benefits of using identity skip-connections similar to [BGW21b], except that the identity was concatenated similar to [HLVDMW17] rather than added to the output. Scaling benefits were shown in MuJoCo from states and DMC from pixels, but results in DMC from states were not presented.

Finally, previous works have presented some evidence that the SAC critic loss does not scale whereas the DDPG critic does: [KYF20] presents SAC trained on DMC from pixels, showing that good performance is only obtained with regularization while the appendix of [HIB$^+$18], page 22 presents scaling improvements of larger DDPG networks on MuJoCo ran on rllab [DCH$^+$16].

Normalizing Flows have been applied to actor-critic continuous control but with not much success. [HHAL18] trains by incrementally adding RealNVP normalizing flow layers to learn more complex policies, showing some success after training on MuJoCo tasks for more than a million timesteps. [MDD$^+$20] show mixed results for a MuJoCo agent trained with planar normalizing flows while [WSB19] used a RealNVP policy for continuous gridworld tasks. However, both these methods have a normalizing flow transformation that is not conditioned on the state input, performing $z = g_\phi^{-1}(a)$ and $a = g_\phi(z)$ instead of $z = g_\phi^{-1}(a \mid s)$ and $a = g_\phi(z \mid s)$. In offline RL, [SLZ$^+$20] used a pretrained normalizing flow to transform a Gaussian to learned density regions of the action space, whose parameters were frozen and transferred over to a new agent to facilitate data-efficient transfer learning for the same agent on a different task.

Variational Q-Learning trains a hierarchical policy, an approach that bears some similarities with the options framework [SPS99, Pre00], but without the persistence of an option for multiple timesteps. The latent variable may be thought of as a distributed representation of a context variable for the state. Options are a special case of the latent variable framework when the latent variable collapses to a set of discrete values that might be the same for several timesteps. [ZCB$^+$22] train a latent-variable policy for reinforcement learning in a similar way to our work, but the latent variable is formed from the hidden state of a recurrent neural network applied to all previous observations in the problem of DMC from states.

2.6. Conclusion

This chapter was motivated by deriving an RL algorithm from the principle of maximum entropy and testing whether it when scaled. SACLite and DDPG were shown to have maximum-entropy justifications, while SAC did not and was instead derived from a different viewpoint of Maximum-Entropy RL.

Scaled-up critics were found to yield improvements in MuJoCo but not DMC, indicating that improvements in MuJoCo might not hold in DMC or vice versa. Future work and experiments should look into understanding the difference and benchmarking in both.

Scaling up the actor both in terms of increasing its parameters and increasing its complexity were found to be ineffective in MuJoCo. We attribute this to the fact that the optimization problem in actor-critic systems learns a critic first then an actor from the critic. We hypothesize that the actor's quality is bounded by the quality of the critic and that efforts should focus on improving the critic and understanding its training dynamics.

In DMC, the log-probability of a non-deterministic policy collapses to a high value, indicating that it has become deterministic as most actions are low-probability. This motivates the use of a “deterministic” policy with fixed variance, resulting in the DDPG algorithm with Twin Critics.

Chapter 3

Convergent Continuous Control

An exciting application of continuous control is robotics. Robotic systems that control the physical world require controlling and understanding continuous quantities such as positions, velocities, and accelerations [TDM$^+$18]. However, continuous control trains unstably [Irp18, HIB$^+$18], with minor variations in hyperparameters or even random seed yield drastically different training behaviors. Although deep reinforcement learning (DRL) has produced superhuman agents in Chess, Shogi, Go and Atari [SHM$^+$16, SSS$^+$17, SAH$^+$20, KCJ$^+$22] with discrete control, DRL struggles to solve continuous control in simulated robotics environments such as DeepMind Control (DMC) or MuJoCo with ideal physics have been fully solved.

Within these environments, DDPG with Twin Networks (which we name TD2 for convenience) has been shown to be the most stable and sample-efficient continuous control algorithm (Chapter 2), [YFLP21]. TD2 is an elegant continuous control algorithm because it makes minimal modifications to the mathematics of actor-critic continuous control with only *two* regularization techniques.

Deep Deterministic Policy Gradients (DDPG) was the first actor-critic algorithm that successfully showed learning in continuous-control tasks with deep learning networks. The crucial component that stabilizes DDPG was an *exponential moving average* in the parameters of target networks (herein abbreviated to **target networks**) [LHP$^+$15]. **Twin Networks**, introduced in TD3, take the minimum of two Q-functions to compute the bootstrap target in the Bellman backup for training the critic. Twin Networks were shown to greatly improve sample-efficiency within MuJoCo [FHM18]. Both Twin Networks and Target Networks are *ubiquitous* components of nearly all following actor-critic continuous control algorithms. However, TD2 only solves environments where the action dimensionality is not low or high (Figure 2.9, Chapter 2). Its regularizers might be hindering performance on these environments. These regularizers are empirically discovered but may interfere with the mathematics of an actor-critic algorithm. Understanding the impact of deep learning regularizers on training requires *investigating the training dynamics* of Deep Q-Learning.

3.1. Training Dynamics of Q-Learning

Using standard RL notation, the Q-Learning loss is given by:

$$L_\theta = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\epsilon_{\theta,i}^2 = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{2}\Big(Q_\theta(s_i,a_i) - r_i - \gamma \max_a Q_{\bar{\theta}}(s_i',a)\Big)^2\right]$$

where $\bar{\theta}$ indicates a stop-gradient operation and (s,a,r,s') tuples are sampled from a replay buffer. The Q-learning gradient update is given by the semi-gradient update rule of:

$$\theta \leftarrow \theta - \eta\frac{1}{n}\sum_{i=1}^{n}\epsilon_{\theta,i}\,\nabla_\theta Q_\theta(s_i,a_i) \tag{3.1.1}$$

As Chapter 2 showed that modifications to the critic had a greater effect on learning, while changes to the actor barely mattered, we thus turn our attention to analyzing the training dynamics of the critic. We analyze the training dynamics of Q-learning following the Neural Tangent Kernel approach introduced in Subsection 1.2.1. We first focus on the Q-learning loss *without* target networks for ease of analysis.

A first-order Taylor expansion of $L_{\theta+\delta\theta}$ with perturbation $\delta\theta$ is used to find a first-order approximation of the change in loss δL. To compute this value, we first take a first-order Taylor expansion of the online Q:

$$Q_{\theta+\delta\theta,i} = Q_{\theta,i} + \delta\theta^\top\nabla_\theta Q_{\theta,i} + \cdots$$

and the target Q:

$$Q'_{\overline{\theta,i+\delta\theta}} = Q'_{\theta,i+\delta\theta} = Q'_{\theta,i} + \delta\theta^\top\nabla_\theta Q'_{\theta,i} + \cdots$$

where $Q_\theta(s_i,a_i)$ and $\max_a Q_\theta(s_i',a)$ are respectively abbreviated as $Q_{\theta,i}$ and $Q'_{\theta,i}$. These yield a Taylor expansion of $L_{\theta+\delta\theta,i}$, the loss for sample i as:

$$\begin{aligned}
L_{\theta+\delta\theta,i} &= \frac{1}{2}(Q_{\theta,i} + \delta\theta^\top\nabla_\theta Q_{\theta,i} \;-\; r \;-\gamma(Q'_{\theta,i} + \delta\theta^\top\nabla_\theta Q'_{\theta,i}))^2 \\
&= \frac{1}{2}(\epsilon_i + \delta\theta^\top(\nabla_\theta Q_{\theta,i} - \gamma\,\nabla_\theta Q'_{\theta,i}))^2 \\
&= L_\theta + \epsilon_i\,\delta\theta^\top(\nabla_\theta Q_{\theta,i} - \gamma\,\nabla_\theta Q'_{\theta,i}) + \cdots
\end{aligned}$$

and

$$\delta L_i = \epsilon_i\,\delta\theta^\top(\nabla_\theta Q_{\theta,i} - \gamma\,\nabla_\theta Q'_{\theta,i})$$

When the semi-gradient update rule of $\delta\theta$ (Equation 3.1.1) is substituted, the change in the loss for a sample i becomes:

$$\delta L_i = -\epsilon_i\, \eta\, \frac{1}{n} \sum_{j=1}^{n} \epsilon_j\, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \gamma\, \nabla_\theta Q'_{\theta,i})$$

With expectations over multiple samples in the loss function, the change in loss is:

$$\delta L = -\eta\, \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\, \epsilon_j\, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \gamma\, \nabla_\theta Q'_{\theta,i})$$

The Q-Learning loss is a squared loss and bounded below at 0. As the loss is minimized iteratively by gradient descent, the convergence of the loss may be shown by a negative change in the loss at every iteration. This convergence is only guaranteed if

$$\delta L < 0$$

which implies:

$$\begin{aligned} -\eta\, \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\, \epsilon_j\, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \gamma\, \nabla_\theta Q'_{\theta,i}) < 0 \\ - \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\, \epsilon_j\, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \gamma\, \nabla_\theta Q'_{\theta,i}) < 0 \end{aligned}$$

Note that there is no guarantee that this inequality is positive because the signs of ϵs are undetermined.

3.1.1. Target Networks Do Not Guarantee Convergence

Target networks have been empirically shown to prevent divergence. As mentioned previously, target networks take an exponential moving average of the target network parameters, which will be denoted as

$$\hat\theta \leftarrow (1-\alpha)\hat\theta_{\text{old}} + \alpha\theta$$

where $0 < \alpha < 1$. The smaller α is, the slower $\hat\theta$ changes between updates. The intuition behind target networks is to provide slow-moving (and therefore more consistent and stable) estimates for the online networks to learn from. Target Networks also modify the convergence condition as follows. The $\delta\theta$ updates of $Q'_{\hat\theta}$ becomes

$$\begin{aligned} Q'_{\overline{(1-\alpha)\hat\theta_{\text{old}} + \alpha(\theta+\delta\theta)}} &= Q'_{(1-\alpha)\hat\theta_{\text{old}} + \alpha(\theta+\delta\theta)} \\ &= Q'_{\hat\theta + \alpha\delta\theta} \\ &= Q'_{\hat\theta} + \alpha\, \delta\theta^\top \nabla_{\hat\theta} Q'_{\hat\theta} \end{aligned}$$

which after substitution into $L_{\theta-\delta\theta}$ yields

$$\delta L = -\eta \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \alpha \, \gamma \, \nabla_\theta Q'_{\theta,i})$$

which modifies the inequality for convergence into:

$$-\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \, \nabla_\theta Q_{\theta,j}^\top (\nabla_\theta Q_{\theta,i} - \alpha \, \gamma \, \nabla_\theta Q'_{\theta,i}) < 0$$

Decreasing α slows down the exponential moving average and the change in $\hat{\theta}$, decreasing the magnitude of the bracketed term, which may speed up convergence. Note that target networks do not guarantee convergence of Q-learning because, as above, there is no guarantee that this inequality is positive because the signs of the ϵs are undetermined.

3.1.2. Convergence with LayerNorm

Placing LayerNorm [BKH16] on ρ satisfies the condition for convergence and prevents divergence. LayerNorm standardizes a vector by subtracting its mean and dividing by its standard deviation. As shown in Subsection 1.2.2, the L2 norm of the output of LayerNorm is proportional to the dimensionality of its input and not the input's magnitude. Consider linear $Q_\theta(s,a) = \theta^\top \rho(s,a)$, with

$$\nabla_\theta \, \theta^\top \rho(s,a) = \rho(s,a)$$

This simplifies the inequality to

$$-\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \, \rho(s_j,a_j)^\top (\rho(s_i,a_i) - \alpha \, \gamma \, \rho(s_i,a_i)) < 0$$

Let the magnitude of the feature vector with LayerNorm applied be $\|\rho(s,a)\|_2 = c$

$$\begin{aligned}
& -\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \left(\rho(s_j,a_j)^\top \rho(s_i,a_i) - \alpha \, \gamma \, \rho(s_j,a_j)^\top \rho(s_i,a_i) \right) \\
\leqslant & -\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \left(|\rho(s_j,a_j)^\top \rho(s_i,a_i)| - \alpha \, \gamma \, |\rho(s_j,a_j)^\top \rho(s_i,a_i)| \right) \quad \text{(magnitude)} \\
\leqslant & -\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \left(\|\rho(s_j,a_j)^\top \rho(s_i,a_i)\|_2 - \alpha \, \gamma \, \|\rho(s_j,a_j)^\top \rho(s_i,a_i)\|_2 \right) \quad \text{(Cauchy-Schwartz)} \\
= & -\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i \, \epsilon_j \left(c^2 - \alpha \, \gamma \, c^2 \right) \quad \text{(define } \|\rho(s,a)\|_2 = c\text{)}
\end{aligned}$$

$$= -c^2\,(1 - \alpha\,\gamma) \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\,\epsilon_j$$

Now, to show that the following inequality holds,

$$-c^2\,(1 - \alpha\,\gamma) \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\,\epsilon_j < 0$$

The inequality only depends on the product of ϵs, which may be rewritten as a square term and a correlation term

$$\sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_i\,\epsilon_j$$
$$= \sum_{i=1}^{n} \epsilon_i^2 + \sum_{\substack{j=1 \\ j \neq i}}^{n} \epsilon_i \epsilon_j$$
$$= \lim_{n \to \infty} \sum_{i=1}^{n} \epsilon_i^2$$

As the number of samples approaches infinity, the correlation term has an expectation of 0 because in the limit of infinite samples ϵ_i and ϵ_j would be considered to be independent. Thus only the first squared term needs to be considered. Now

$$-c^2\,(1 - \alpha\,\gamma) \sum_{i=1}^{n} \epsilon_i^2 < 0$$
$$\alpha\gamma - 1 < 0$$

By definition, $\gamma < 1$, $\alpha < 1$, and thus LayerNorm facilitates convergence. Note that in theory, LayerNorm enables convergence of Q-learning even without target networks, in other words, if $\alpha = 1$.

3.2. Empirical Convergence Across Action Dimensions

The impact of target networks and LayerNorm are investigated in TD2. LayerNorm is only applied to the critic and placed on every layer after the preactivations and before the activation function of ReLU. This is because the theory only suggests that LayerNorm and target networks are necessary on the critic. Although the theory was developed for LayerNorm only on the penultimate features of a neural network, LayerNorm is placed at *every layer*, just as in supervised learning ReLUs and LayerNorm are placed throughout the network while only having theory (Subsection 1.2.2) that places them on the penultimate layer.

In addition to DMC and MuJoCo, the resultant algorithm is tested on an artificial n-dimensional navigation task that should be fairly easy to solve. DMC and MuJoCo have environments of varying action dimensions but the problem semantics (most notably the dynamics functions) differ across environments. In contrast, the n-dimension navigational space is designed to yield environments that are as similar as possible across varying dimensionalities.

In n-dimensional navigation, an agent finds itself at an n-dimensional (Euclidean) position for every timestep. The agent changes its position between -1 and 1 for each dimension, receiving a reward that is the difference between the $L2$ of the origin and its position for two consecutive timesteps, divided by the square root of n. The agent starts in a position that is sampled from a unit Gaussian and acts in this environment for 1000 timesteps. The dimensionality n may be varied to verify whether Q-learning only works for medium-range action dimensions.

Like the previous section, experiments were run for a million timesteps. The policy was tested every 1000 timesteps and the average return from ten rollouts was recorded. However, aggregate statistics were instead computed by the **InterQuartile Mean (IQM)**, which had recently been shown to be a more sensitive aggregate metric [ASC+21]. Twelve seeds were used because it was the smallest as a multiple of four greater than 10 so the windsorizing in the IQM would be over whole numbers. The small neural networks used in the following algorithms had two hidden layers of size 256 and ReLU activations [GBB11] as standard. A major difference from the DDPG [LHP+15] and [FHM18] algorithms was that weights were initialized with **orthogonal intialization** [SMG13] rather than the historically used **Xavier initialization** [GB10]. A gain of $\sqrt{2}$ was used for all layers apart from the last layer of the actor, which had a gain of 1. The fixed standard deviation of the deterministic policy was set to 0.2.

3.3. LayerNorm Results and Discussion

Figure 3.1 shows that increasing the dimensionality of the navigation environment makes it harder for TD2 to solve. TD2 without LayerNorm fails at dimension 10. The ablation of DDPG (removing target networks from LayerNorm) indicates that this failure is due to the lack of LayerNorm. Analysis of the overestimation in the Q-function shows that the Q-values diverge for high action-dimension environments in Figure 3.2. Although TD2 and LayerNorm fail to achieve comparable performance for navigation tasks below and above a dimensionality of 50, we hypothesize that they might be solvable given more interactions than the $1e^5$ used in these experiments given that the critic values in the higher action

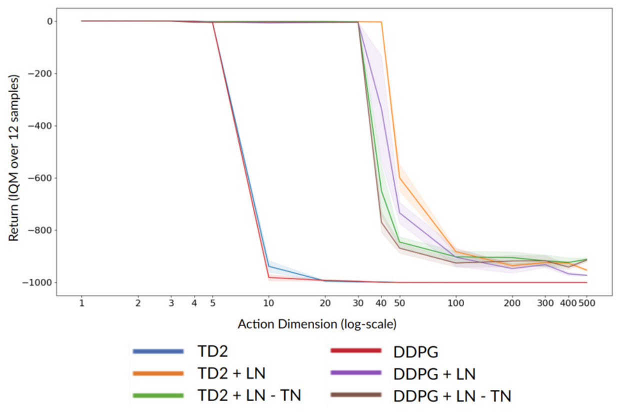

Fig. 3.1. Return after $1e^5$ timesteps calculated by the interquartile mean (IQM) over 12 seeds, each evaluated 100 times for navigation tasks across varying dimensions. Shaded regions indicate the standard error computed over 12 seeds. LN stands for LayerNorm while TN stands for Target Networks.

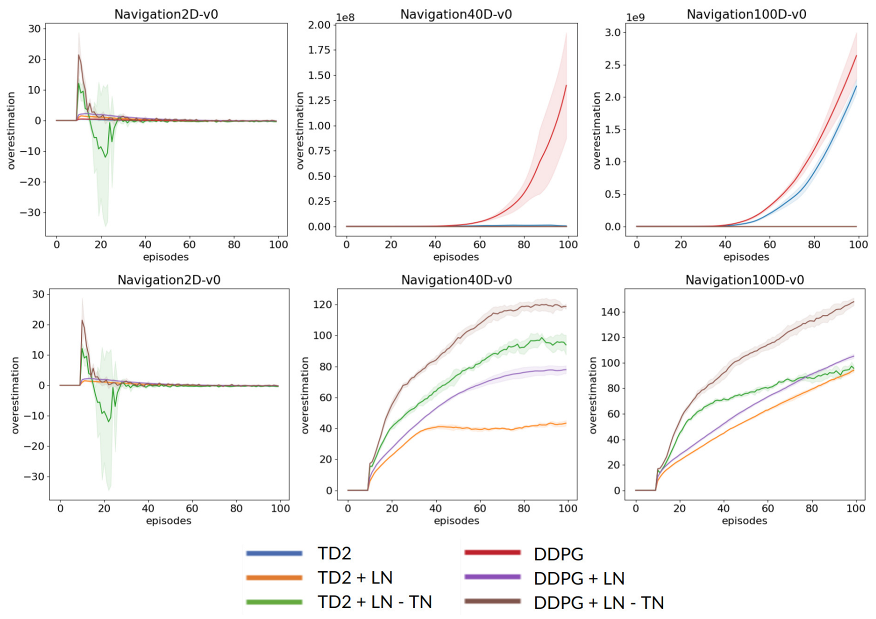

Fig. 3.2. Approximate overestimation of Q-function calculated by subtracting the Monte-Carlo return from $Q(s_0, a_0)$ calculated from one state. Bottom plot has blue lines removed for clarity of inspecting the other two. Values are means over 12 seends and shaded regions indicate standard error computed over 12 seeds. LN stands for LayerNorm while TN stands for Target Networks.

dimensions of Figure 3.2 have not diverged and require more samples to become inaccurate, just as the critic values for Navigation in a 2D environment eventually recovered from an initial inaccuracy. We further note that consistent with our theory, LayerNorm enables good Q-learning to be performed without the need for target networks.

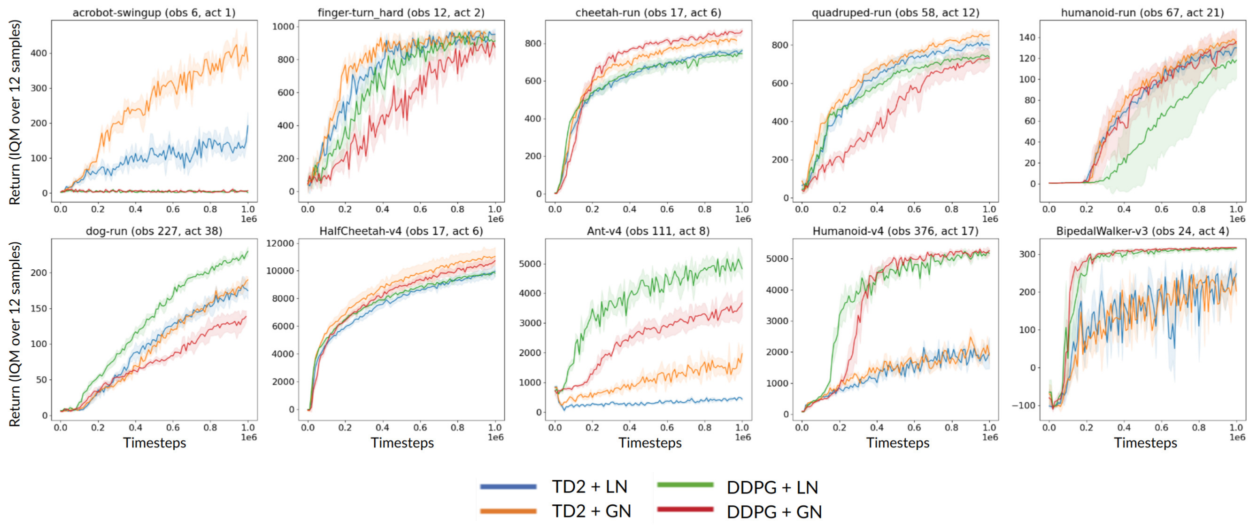

Fig. 3.3. Ablation of using GroupNorm (GN) with 16 groups over LayerNorm (LN) in DMC, MuJoCo, and Box2D environments. Return after $1e^5$ timesteps calculated by the interquartile mean (IQM) over 12 seeds, each evaluated 100 times. Shaded regions indicate standard error computed over 12 seeds.

Figure 3.3 shows that a variant of LayerNorm, GroupNorm [WH18], is more effective over DMC, MuJoCo and Box2D. **GroupNorm** splits a vector into a fixed number of groups before standardizing each, producing a smaller variance than LayerNorm. We choose to use 16 groups because the size of the layer is 256, whose square root is 16. The number of groups should be large enough such that its behavior is distinct from LayerNorm but small enough such that there is still enough variation between groups.

Figure 3.4 shows that GroupNorm enables good Q-learning to be performed without the need of target networks, but the best results are obtained with both LayerNorm and target networks, which is consistent with our theory. Close inspection of individual training runs in Figure 3.5 show that over twelve seeds, all training runs with GroupNorm retain behaviors that obtain positive returns, unlike the ablations which are also higher variance.

Figure 3.4 shows that DMC with one critic (DDPG) instead of two (Twin Critics) may sometimes be more effective, especially for acrobot, where TD2 does not learn good behaviors, even with GroupNorm. However, DDPG is not superior to TD2 over all environments and is inferior in MuJoCo and Box2D. This indicates that there is more than just a convergence issue at fault with actor-critic algorithms.

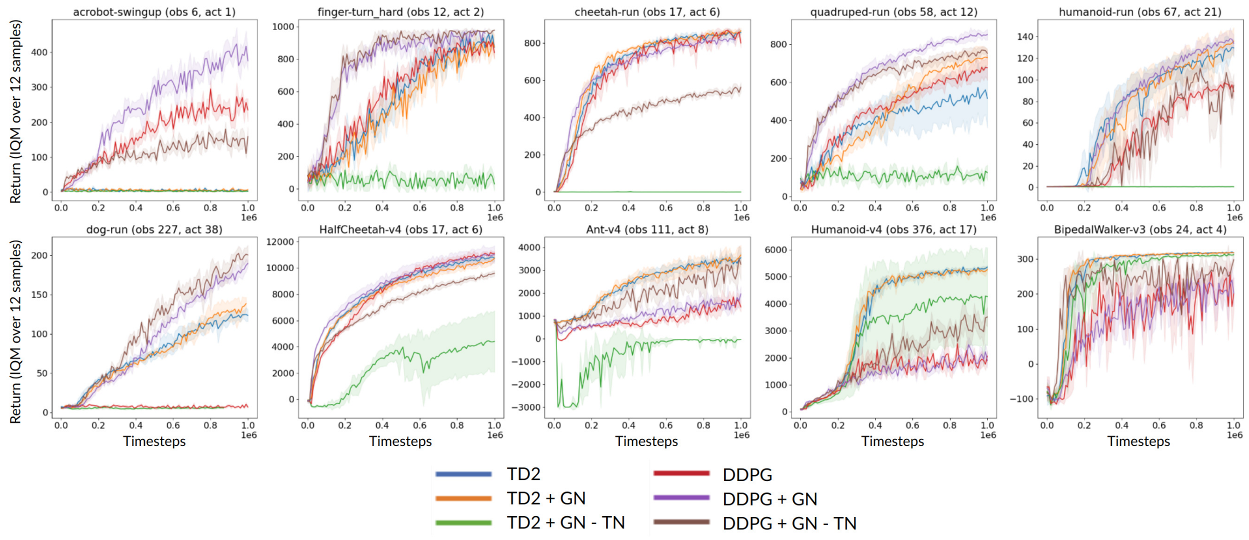

Fig. 3.4. Benchmarks of TD2 with GroupNorm (GN) and Target Network (TN) ablations in DMC, MuJoCo, and Box2D environments. Return after $1e^5$ timesteps calculated by the interquartile mean (IQM) over 12 seeds, each evaluated 100 times. Shaded regions indicate standard error computed over 12 seeds.

3.4. Fixed Asymmetry With Twin Networks

The interplay between Twin Critics and actor-critic continuous control is investigated. Consider the Q-Learning loss with standard RL notation:

$$L_\theta(s,a,r,s') = \left(Q_\theta(s,a) - r - \gamma Q_{\bar{\theta}}(s',a')\right)^2$$

where $a' = \pi_\phi(s')$. This loss is minimized by gradient descent wrt θ.

Twin Networks modify the loss by introducing an ensemble of two Q-network and selecting the minimum of the two for a target value:

$$L^{\text{Twin}}_{\theta_j}(s,a,r,s') = \sum_{j\in\{1,2\}} \left(Q_{\theta_j}(s,a) - r - \gamma \min_{i\in\{1,2\}} Q_{\overline{\theta_i}}(s',a')\right)^2$$

Twin Networks was motivated to reduce the overestimation of the target Q-values. We hypothesize that Twin Networks might promote underestimation which may sometimes cripple policy learning. The source of underestimation might be traced back to a rearrangement of the Twin-Q-Network loss:

$$\begin{aligned}L^{\text{Twin}}_{\theta}(s,a,r,s') &= \sum_{j\in\{1,2\}} \left(Q_{\theta_j}(s,a) - r - \gamma \min_{i\in\{1,2\}} Q_{\overline{\theta_i}}(s',a')\right)^2 \\ &= \sum_{j\in\{1,2\}} \left(\max_{i\in\{1,2\}} \left[Q_{\theta_j}(s,a) - r - \gamma Q_{\overline{\theta_i}}(s',a')\right]\right)^2\end{aligned}$$

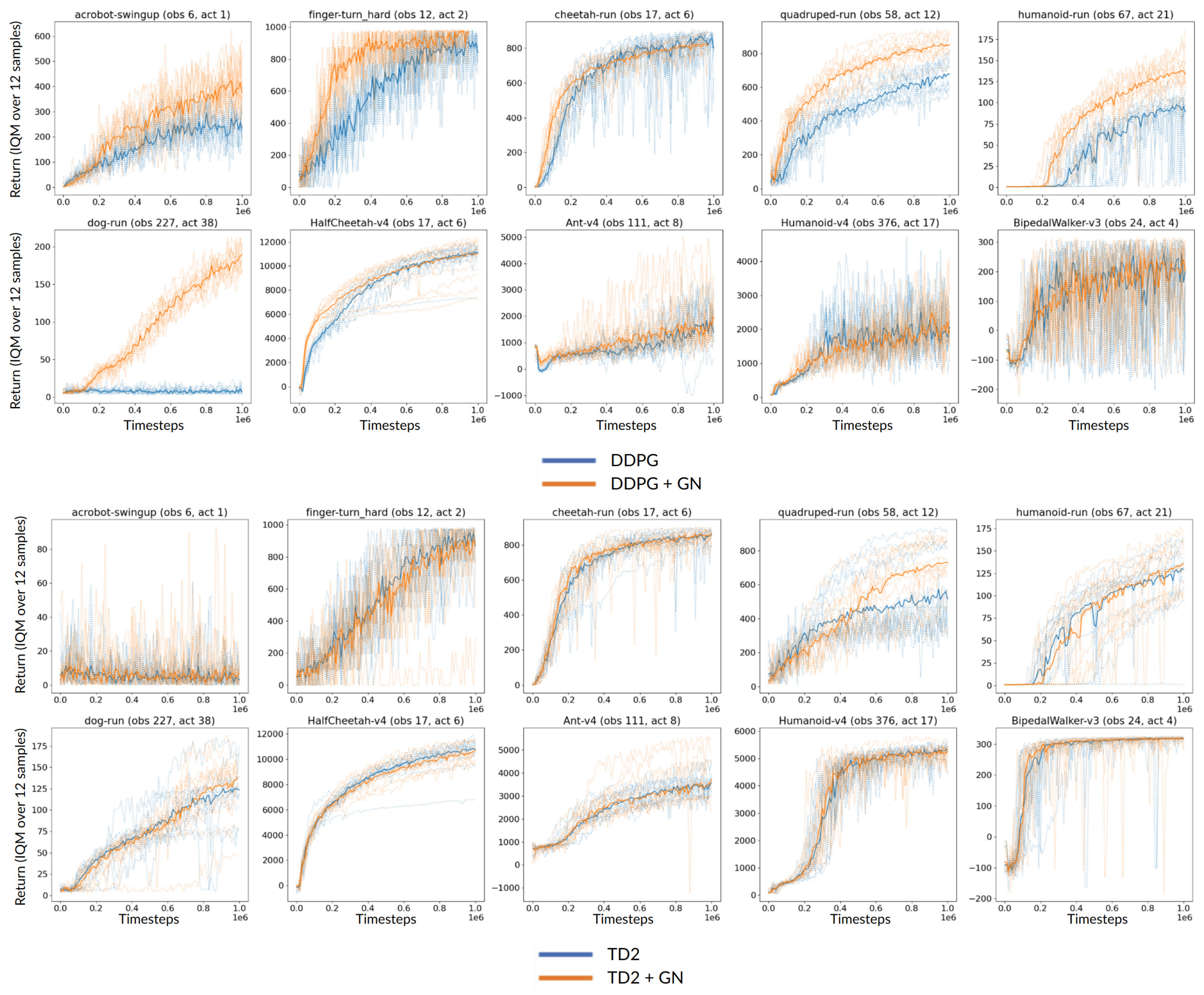

Fig. 3.5. Individual training runs of all twelve seeds in DMC, MuJoCo, and Box2D environments. The solid line is the return after $1e^5$ timesteps calculated by the interquartile mean (IQM) over 12 seeds. A dashed line indicates the return evaluated over 100 times for one particular run.

This loss is asymmetric for positive and negative quantities. Positive quantities are more heavily penalized than negative quantities. Minimizing this loss (wrt θ) might be interpreted as trying to minimize the more positive of a pair of (non-squared) TD losses – the loss furthest from zero is minimized. However, this only holds if both TD losses are positive. If both losses are negative, the less negative loss with be selected to be minimized – the loss closer to zero minimized. This may mean that in some environments (such as acrobot-swingup in DMC), Twin Networks are less likely to recover from underestimation.

The degree of asymmetry required varies between environments. Less asymmetry is needed in acrobot-swingup than in the MuJoCo Humanoid environment. A simple way to vary the degree of asymmetry is to introduce a hyperparameter α that varies the steepness of the loss

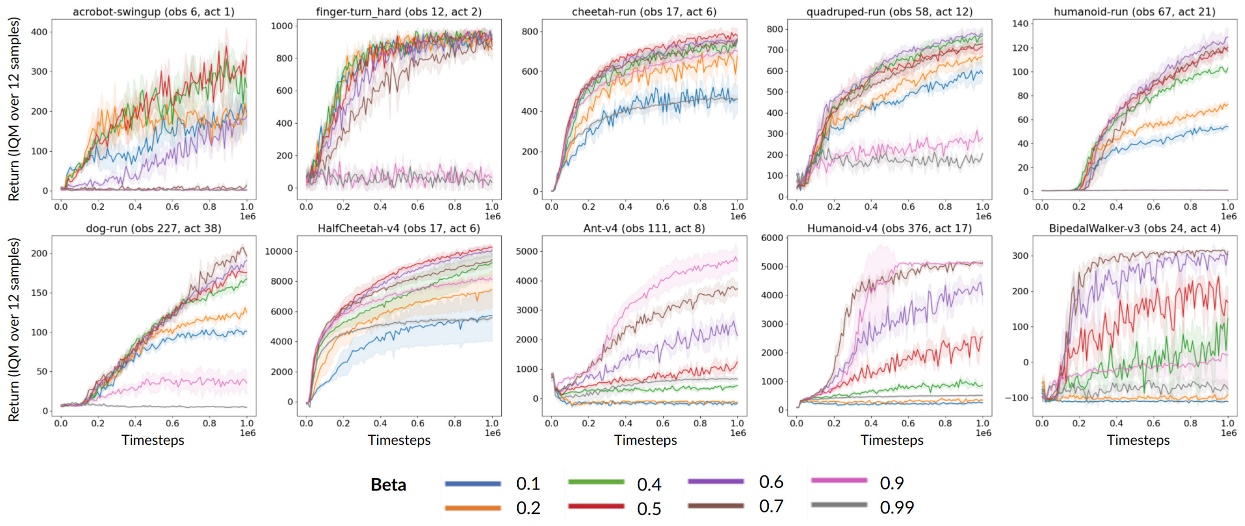

Fig. 3.6. Asymmetric Q-Learning results with varying degrees of asymmetry determined by β, with more positive βs denoting a steeper positive half. Returns were calculated by the interquartile mean (IQM) over 12 seeds, each evaluated 100 times. Shaded regions indicate standard error computed over 12 seeds.

function of DDPG for positive and negative residuals:

$$\alpha \left(\max\left[0, Q_\theta(s,a) - r - \gamma Q_{\bar{\theta}}(s',a')\right]\right)^2 + (1-\alpha)\left(\min\left[0, Q_\theta(s,a) - r - \gamma Q_{\bar{\theta}}(s',a')\right]\right)^2$$

Experiments were run on DMC, MuJoCo, and Box2D. Figure 3.6 showed that an asymmetric loss could match and sometimes surpass the TD2 and DDPG algorithms. Note that DDPG corresponds to a symmetric loss when $\alpha = 0.5$. Table 3.1 found that higher degrees of asymmetry were needed to solve DMC environments of higher action dimension, while MuJoCo and Box2D environments needed the highest amount of asymmetry at all. There is a ‘Goldilocks Zone’ of asymmetry that is neither too low nor too high that produces the best learning.

3.5. Related Work

Normalization has previously been used in Q-learning but has not been shown to cause convergence of online RL. DDPG [LHP$^+$15] used BatchNorm in the critic but was removed in TD3 [FHM18] as it was shown to hurt performance. [BAAB19] showed that LayerNorm could replace target networks in DDPG but with marginally worse performance in MuJoCo and did not have a mathematical justification for using LayerNorm. [BGW21a] used LayerNorm on the penultimate layer of the critic and showed a reduced variance for DMC from pixels. An alternate method of normalization – dividing the penultimate layer by its L2 norm – also reduced variance to a marginally better effect. Finally, [KAM$^+$21] analyzed training

Environment	0.1	0.2	0.4	0.5
acrobot-swingup	199 ± 54.6	188 ± 29.8	260 ± 65.9	**340 ± 44**
finger-turn_hard	915 ± 38.1	904 ± 39.6	**941 ± 44.6**	876 ± 8.5
cheetah-run	463 ± 68.9	676 ± 33.5	757 ± 10	**778 ± 35.7**
quadruped-run	588 ± 25.2	672 ± 30.1	767 ± 31.2	715 ± 26
humanoid-run	54.2 ± 2.74	72.8 ± 3.26	103 ± 3.72	120 ± 3.05
dog-run	101 ± 2.71	125 ± 4.57	168 ± 8.57	176 ± 9.24
HalfCheetah-v4	5710 ± 1700	7470 ± 1020	9230 ± 950	**10300 ± 145**
Ant-v4	-161 ± 14.6	-117 ± 35.2	437 ± 54	1240 ± 251
Humanoid-v4	247 ± 41.6	340 ± 33.9	859 ± 147	2550 ± 235
BipedalWalker-v3	-111 ± 1.15	-91.4 ± 17.1	124 ± 60	168 ± 43.5

Environment	0.6	0.7	0.9	0.99
acrobot-swingup	187 ± 29.8	14.3 ± 14.6	1.76 ± 1.54	2.27 ± 1.39
finger-turn_hard	921 ± 43.7	854 ± 65	66.7 ± 47.1	33.4 ± 47.1
cheetah-run	763 ± 18.7	761 ± 16.9	701 ± 23.8	463 ± 17.6
quadruped-run	**776 ± 32.4**	729 ± 29.8	280 ± 24.7	205 ± 8.87
humanoid-run	**129 ± 6.72**	117 ± 5.29	0.969 ± 0.0945	1.21 ± 0.0786
dog-run	191 ± 8.03	**197 ± 3.67**	35 ± 13.7	4.3 ± 0.349
HalfCheetah-v4	10000 ± 310	9440 ± 464	8100 ± 174	5580 ± 83.1
Ant-v4	2140 ± 167	3730 ± 238	**4670 ± 367**	655 ± 74.4
Humanoid-v4	4010 ± 375	**5130 ± 82.1**	5090 ± 78.4	510 ± 38.2
BipedalWalker-v3	**312 ± 2.99**	302 ± 12.8	20.4 ± 107	-74.5 ± 19.5

Table 3.1. Interquartile Mean (IQM) averaged over 12 seeds of 10 rollouts after training for a million timesteps varying asymmetry. Mean ± standard error numbers are given to three significant figures. Methods that give the highest returns per environment are bolded.

dynamics for offline RL using a different method that yielded the same convergence condition, and [KAG$^+$22] showed that the condition could also be satisfied by dividing the penultimate layer by its L2 norm. [WU21] also presents the convergence condition but using one sample rather than an expectation.

It is well-known that different environments require different amounts of pessimism. [MPHP$^+$21] notes that neither DDPG nor TD3 consistently get better results than the other, and learns to select a target estimate from either the mean or minimum of two critics. [KSGV20] train an ensemble of distributional critics where n is a fixed hyperparameter determined at the start of training. [KGT$^+$22] design a mechanism to adjust n during training by adjusting it such that the aggregated values of truncated critic values are close to Monte-Carlo returns. [KLA$^+$21] find a convex combination of an optimistic and pessimisic Q-value that is learned during training.

Asymmetric losses in reinforcement learning are less well-studied. [KFTN21] use a similar weighted quadratic loss as our work but in the offline RL setting. Two anonymous submissions [Ano22b] and [Ano22a], concurrent work to ours, use asymmetric loss functions in actor-critic continuous control. [Ano22b] uses the asymmetric **LINEX (Linear-Exponential)** [Var75] loss function and [Ano22a] compute a softmax over the TD-error to weigh the importance of each squared TD-loss. The temperature of the softmax is adjusted during training by a trust-region method.

3.6. Conclusion

The training dynamics of deep Q-learning were investigated via a similar method to the Neural Tangent Kernel. The lack of a theoretical convergence guarantee for Target Networks and a convergence guarantee for LayerNorm was empirically verified, as was the theoretical claim that using LayerNorm with Target Networks would converge faster and be more sample efficient. A variant of LayerNorm, GroupNorm, was found to be empirically better. Moreover, Twin Networks was found to harm performance in DMC but removing Twin Networks hurt performance in MuJoCo. Further analysis revealed that Twin Networks were a form of asymmetric loss and that adjusting the degree of asymmetry led to improvements across all domains, but had to be tuned per environment. We propose that future work automatically adjusts the amount of asymmetry.

Chapter 4

Conclusion

This dissertation improved the use of deep networks within a particular set of environments in deep reinforcement learning called continuous control.

New mathematical justifications for supervised and unsupervised learning were found, motivated by whether the empirical success of deep networks in those learning paradigms could be harnessed and transferred to reinforcement learning. The maximum-entropy principle was introduced as a first principle behind deriving the loss functions used in supervised and unsupervised learning, and the effectiveness of layer normalization in the optimization process of deep networks were theoretically shown. When applied to reinforcement learning, the maximum-entropy principle provided a mathematical justification for actor-critic algorithms. Applying a Neural Tangent Kernel-inspired framework to supervised learning showed that layer normalization and the ReLU activation function helped training converge, while that same framework applied to temporal difference learning with deep Q-functions showed that layer normalization was a sufficient condition for convergence in that setting. The NTK-inspired analysis showed that Target Networks were insufficient for convergence but could speed up the rate of convergence when combined with layer normalization. Applying layer normalization to actor-critic algorithms trained exceptionally stably, with all seeds learning and retaining behaviors that obtained positive reward and low-variance between seeds. Future work should empirically investigate whether layer normalization and asymmetric losses may be applied to even more challenging reinforcement learning tasks such as continuous control from pixels, discrete control and other reinforcement learning domains.

Empirically, this dissertation found that changes to the critic had a greater impact on training than changes to the actor. Many types of probabilistic policies were proposed but changing their type made little difference to training overall, giving no reason not to use the simplest policy type: deterministic. Increasing the number of parameters of the critic networks changed performance but did not lead to monotonic improvements with scale. The most sensitive component of an actor-critic agent was the choice of using Twin Critics or not to compute bootstrapped targets. Generalizing the binary choice of using Twin Critics or not into the continuous relaxation of using asymmetric loss functions was empirically shown to be advantageous, especially when the degree of asymmetry was tuned per environment. This leads to an obvious avenue for future work as automatically finding the optimal degree of asymmetry, and theoretical justifications for why this would be needed.

In general, this dissertation highlights the need for further mathematics explaining the interaction between deep networks and reinforcement learning. The maximum-entropy principle was found to derive the actor loss, but there is no justification for what critic loss to use. A potential avenue to determine that loss function would be to understand what noise is introduced into Q-learning when function approximation is used, which could then be incorporated into a maximum-likelihood framework. If the correct critic noise is taken into account, it would hold for arbitrarily powerful function approximators, enabling the most powerful tool in deep learning – scaling – to be used in deep Q-learning.

Bibliography

[AB14] Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014.

[Aba20] Romina Abachi. *Policy-aware model learning for policy gradient methods*. PhD thesis, University of Toronto (Canada), 2020.

[AJKS19] Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, pages 10–4, 2019.

[AN16] Masoud Abdi and Saeid Nahavandi. Multi-residual networks: Improving the speed and accuracy of residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

[Ano22a] Anonymous. AsymQ: Asymmetric Q-loss to mitigate overestimation bias in off-policy reinforcement learning. November 2022.

[Ano22b] Anonymous. Extreme Q-Learning: MaxEnt RL without Entropy. November 2022.

[ASC$^+$21] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

[BAAB19] Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. Crossnorm: Normalization for off-policy td reinforcement learning. *arXiv preprint arXiv:1902.05605*, 2019.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BCP$^+$16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[BCV13] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis* *and machine intelligence*, 35(8):1798–1828, 2013.

[BDK$^+$21] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining scaling laws of neural network generalization. 2021.

[BDPDP96] Adam Berger, Stephen A Della Pietra, and Vincent J Della Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

[Bel57] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[BGW21a] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Is high variance unavoidable in rl? a case study in continuous control. *arXiv preprint arXiv:2110.11222*, 2021.

[BGW21b] Nils Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning with spectral normalization. *Advances in Neural Information Processing Systems*, 34:8242–8255, 2021.

[Bis94] Christopher M Bishop. Novelty detection and neural network validation. *IEE Proceedings-Vision, Image and Signal processing*, 141(4):217–222, 1994.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[BMR$^+$20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. Pattern recognition and machine learning, 2006.

[Bri89] John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in neural information processing systems*, 2, 1989.

[Bri90] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[BTLLW21] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *arXiv preprint arXiv:2103.04922*, 2021.

[BYAV13] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. *Advances in neural information processing systems*, 26, 2013.

[C$^+$01] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[Car90] Sadi Carnot. *Reflections on the motive power of heat and on machines fitted to develop that power.* J. Wiley, 1890.

[CBGLN17] Nicolò Cesa-Bianchi, Claudio Gentile, Gábor Lugosi, and Gergely Neu. Boltzmann exploration done right. *Advances in neural information processing systems*, 30, 2017.

[CVMG$^+$14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[Dar35] Georges Darmois. Sur les lois de probabilitéa estimation exhaustive. *CR Acad. Sci. Paris*, 260(1265):85, 1935.

[DCH$^+$16] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DSDB16] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[FHM18] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[Fis34] Ronald Aylmer Fisher. Two new properties of mathematical likelihood. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 144(852):285–307, 1934.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[GKB+19] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179. PMLR, 2019.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[Has10] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[HBM+22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[HCS+22] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[HHAL18] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1851–1860. PMLR, 2018.

[HIB+18] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[HLBN19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[HLF+19] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[HMVH+18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David

Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[How60] Ronald A Howard. Dynamic programming and markov processes. 1960.

[HS18] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[HSG$^+$22] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[HTAL17] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[HZH$^+$18] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Irp18] Alex Irpan. Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`, 2018.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[Jay57a] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[Jay57b] Edwin T Jaynes. Information theory and statistical mechanics. ii. *Physical review*, 108(2):171, 1957.

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[KAG$^+$22] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

[KAM$^+$21] Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. Dr3: Value-based deep reinforcement learning requires explicit regularization. *arXiv preprint arXiv:2112.04716*, 2021.

[KCJ$^+$22] Steven Kapturowski, Víctor Campos, Ray Jiang, Nemanja Rakićević, Hado van Hasselt, Charles Blundell, and Adrià Puigdomènech Badia. Human-level atari 200x faster. *arXiv preprint arXiv:2209.07550*, 2022.

[KFTN21] Ilya Kostrikov, Rob Fergus, Jonathan Tompson, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[KGT$^+$22] Arsenii Kuznetsov, Alexander Grishin, Artem Tsypin, Arsenii Ashukha, Artur Kadurin, and Dmitry Vetrov. Automating control of overestimation bias for reinforcement learning. 2022.

[KLA$^+$21] Thommen George Karimpanal, Hung Le, Majid Abdolshah, Santu Rana, Sunil Gupta, Truyen Tran, and Svetha Venkatesh. Balanced q-learning: Combining the influence of optimistic and pessimistic targets. *arXiv preprint arXiv:2111.02787*, 2021.

[KMH$^+$20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[Koo36] Bernard Osgood Koopman. On distributions admitting a sufficient statistic. *Transactions of the American Mathematical society*, 39(3):399–409, 1936.

[KPB20] Ivan Kobyzev, Simon JD Prince, and Marcus A Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):3964–3979, 2020.

[Kra91] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[KSGV20] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics. In *International Conference on Machine Learning*, pages 5556–5566. PMLR, 2020.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[KYF20] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[KZTL20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[LB+95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LCH+06] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[Lev18] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[LGR12] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[LHP+15] 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.

[Lin92] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3):293–321, 1992.

[LKTF20] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[Max60a] James Clerk Maxwell. Ii. illustrations of the dynamical theory of gases. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 20(130):21–37, 1860.

[Max60b] James Clerk Maxwell. V. illustrations of the dynamical theory of gases.—part i. on the motions and collisions of perfectly elastic spheres. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 19(124):19–32, 1860.

[MBM+16] Volodymyr Mnih, Adria Puigdomenech 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. PMLR, 2016.

[MDD$^+$20] Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pages 430–444. PMLR, 2020.

[MKKY18] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[MKS$^+$15] 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.

[MPHP$^+$21] Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, and Michael Jordan. Tactical optimism and pessimism for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12849–12863, 2021.

[NAAB22] Evgenii Nikishin, Romina Abachi, Rishabh Agarwal, and Pierre-Luc Bacon. Control-oriented model-based reinforcement learning with implicit differentiation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7886–7894, 2022.

[Pit36] Edwin James George Pitman. Sufficient statistics and intrinsic accuracy. In *Mathematical Proceedings of the cambridge Philosophical society*, volume 32, pages 567–579. Cambridge University Press, 1936.

[Pre00] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[RB18] Mirco Ravanelli and Yoshua Bengio. Speaker recognition from raw waveform with sincnet. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 1021–1028. IEEE, 2018.

[RTB16] Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International conference on machine learning*, pages 324–333. PMLR, 2016.

[RWC$^+$19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[SAH$^+$20] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[Sam37] Paul A Samuelson. A note on measurement of utility. *The review of economic studies*, 4(2):155–161, 1937.

[SB18a] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SB18b] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[SBSG20] Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning. *arXiv preprint arXiv:2010.09163*, 2020.

[SE19] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

[SHM$^+$16] 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.

[SK22] Utkarsh Sharma and Jared Kaplan. Scaling laws from the data manifold dimension. *J. Mach. Learn. Res.*, 23:9–1, 2022.

[SLA$^+$15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[SLZ$^+$20] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[SMG13] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[SMSM99] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[SPS99] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[SSS$^+$17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[SWD$^+$17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[TDM$^+$18] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[TMD$^+$20] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[TS93] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, volume 6, pages 1–9, 1993.

[Var75] Hal R Varian. A bayesian approach to real estate assessment. *Studies in Bayesian econometric and statistics in Honor of Leonard J. Savage*, pages 195–208, 1975.

[VHGS16] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[VI19] Vladimir Vapnik and Rauf Izmailov. Rethinking statistical learning theory: learning using statistical invariants. *Machine Learning*, 108(3):381–423, 2019.

[VWB16] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[WH18] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[WSB19] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019.

[WU21] Zhikang T Wang and Masahito Ueda. Convergent and efficient deep q learning algorithm. In *International Conference on Learning Representations*, 2021.

[WWHW19] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv preprint arXiv:1912.00042*, 2019.

[XSZ$^+$19] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[YFLP21] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[YZX22] Haonan Yu, Haichao Zhang, and Wei Xu. Do you need the entropy reward (in practice)? *arXiv preprint arXiv:2201.12434*, 2022.

[ZCB$^+$22] Dinghuai Zhang, Aaron Courville, Yoshua Bengio, Qinqing Zheng, Amy Zhang, and Ricky TQ Chen. Latent state marginalization as a low-cost approach for improving exploration. *arXiv preprint arXiv:2210.00999*, 2022.