Deep Generative Models: Imitation Learning, Image Synthesis, and Compression

Jonathan Ho

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-67
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-67.html

May 27, 2020

Deep Generative Models: Imitation Learning, Image Synthesis, and Compression

by

Jonathan Ho

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Dawn Song
Professor Michael DeWeese

Spring 2020

Deep Generative Models: Imitation Learning, Image Synthesis, and Compression

Abstract

Deep Generative Models: Imitation Learning, Image Synthesis, and Compression

by

Jonathan Ho

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

Machine learning has its roots in the design of algorithms that extract actionable structure from real world data. For high dimensional and high entropy data, machine learning techniques must cope with a fundamental tension arising from the curse of dimensionality: they must be computationally and statistically efficient despite the sheer size of the exponentially large spaces where meaningful signal is hidden. This thesis explores and attempts to resolve this tension in deep generative modeling.

The first part of this thesis addresses imitation learning, the problem of reproducing the behavior of experts acting in dynamic environments. Supervised learning, to predict expert actions from states, suffers in statistical efficiency because large amounts of expert data are required to prevent action prediction errors from compounding over long behaviors. We propose a resolution in the form of an algorithm that learns a policy by matching the expert's state distribution. During learning, our algorithm continually executes the policy in the task environment and compares its states to the expert's on a gradually learned reward function. Allowing our algorithm to interact with the environment during training in this manner allows it to learn policies that stay on expert states even when expert data is extremely scarce.

The second part of this thesis addresses modeling and compressing natural images using likelihood-based generative models, which are generative models trained with maximum likelihood to explicitly represent the probability distribution of data. When these models are scaled to high entropy datasets, they become computationally inefficient to employ for downstream tasks like image synthesis and compression. We present progress on these problems in the form of developments in flow models, a class of likelihood-based generative models that admit fast sampling and inference. We reduce flow model codelengths to be competitive with those of other types of likelihood-based generative models. Then, we develop the first computationally efficient compression algorithms for flow models, making our improved codelengths realizable in practice with fully parallelizable encoding and decoding.

To my parents.

Contents

Contents **ii**

List of Figures **iii**

List of Tables **iv**

1 Introduction **1**

2 Imitation Learning **3**
2.1 Introduction . . . 3
2.2 Background . . . 4
2.3 Characterizing the induced optimal policy . . . 5
2.4 Practical occupancy measure matching . . . 8
2.5 Generative adversarial imitation learning . . . 11
2.6 Experiments . . . 12
2.7 Discussion . . . 15
2.8 Recent related work . . . 15

3 Likelihood-Based Generative Modeling **17**
3.1 Introduction . . . 17
3.2 Improving flow models . . . 18
3.3 Flow model experiments . . . 23
3.4 Compression with flows . . . 28
3.5 Local bits-back coding . . . 29
3.6 Compression experiments . . . 39
3.7 Discussion . . . 40
3.8 Recent related work . . . 43

4 Conclusion **44**

Bibliography **45**

List of Figures

2.1 Performance of learned policies. The y-axis is negative cost, scaled so that the expert achieves 1 and a random policy achieves 0. 14
2.2 Causal entropy regularization λ on Reacher. 14

3.1 Ablation training (light) and validation (dark) curves on unconditional CIFAR10 density estimation. These runs are not fully converged, but the gap in performance is already visible. 25
3.2 CIFAR10 Samples. Flow++ captures local dependencies well and generates good samples at the quality level of PixelCNN, but with the advantage of efficient sampling. 26
3.3 32x32 ImageNet Samples. Flow++ samples match those of PixelCNN in diversity on this dataset, which is much larger than CIFAR10. 26
3.4 64x64 ImageNet Samples. The diversity of samples from Flow++ matches the diversity of samples from PixelRNN with multi-scale ordering. 27
3.5 5-bit 64x64 CelebA Flow++ samples, without low-temperature sampling. 27
3.6 Effects of precision and noise parameters δ and σ on coding random subsets of CIFAR10 (top), ImageNet 32x32 (middle), and ImageNet 64x64 (bottom) 41

List of Tables

3.1 Flow++ unconditional image modeling results (in bits per dimension) 24
3.2 CIFAR10 ablation results after 400 epochs of training. Models not converged for the purposes of ablation study. 24
3.3 Local bits-back codelengths (in bits per dimension) 39
3.4 Encoding time per datapoint (in seconds) 42
3.5 Decoding time per datapoint (in seconds) 42

Acknowledgments

The work here would not have been possible without the generous mentorship of Professor Pieter Abbeel, who guided me from the beginning in my undergraduate years, and Professor Stefano Ermon, who guided me early on in graduate school. I am indebted to the innumerable colleagues whom I have been fortunate to meet at Berkeley, Stanford, OpenAI, and Google; these exceptional researchers have repeatedly inspired me with beautiful ideas and helped me pursue them to ends that I could have never imagined before starting in graduate school. I am eternally grateful for friends who have stayed with me from all these institutions going back as early as high school; their companionship has made all these days worth experiencing. Most importantly, I thank my family for their unconditional love and support. This thesis is for you.

Chapter 1

Introduction

This thesis covers the interplay between ideas in deep generative models and other domains involving machine learning, in particular imitation learning and compression. A common theme is the curse of dimensionality, how it manifests in these domains, and how it can be resolved by careful algorithm design. These domains consist of problems of learning, reproducing, and exploiting learned properties of raw data distributions using deep learning techniques, which through their inductive biases are currently some of the most promising for coping with high dimensional and high entropy data in a wide variety of modalities.

Chapter 2 addresses imitation learning. The particular setting we consider is the following: given a dataset of expert trajectories of the form $((s_1, a_1), \ldots, (s_T, a_T))$, where s_t are states and a_t are actions in some environment, to produce a policy $\pi(a|s)$ that reproduces the same behavior upon execution in that environment; no interaction with the expert or access to a reinforcement signal is allowed in this setting. One approach is to recover the expert's cost function with inverse reinforcement learning, then extract a policy from that cost function with reinforcement learning. This approach is indirect and can be slow. Another is behavioral cloning, which is to learn $\pi(a|s)$ by supervised learning, but this suffers from compounding error when trajectories are long.

Our contribution is to propose a general framework for directly extracting a policy from data as if it were obtained by reinforcement learning following inverse reinforcement learning. We instantiate this with our algorithm, generative adversarial imitation learning, which is a model-free imitation learning algorithm that is able to imitate complex behaviors in high dimensional continuous control environments using very little expert data. Its design draws an analogy between imitation learning and generative adversarial networks; it was published in NeurIPS 2016 [36] based on earlier ideas from ICML 2016 [37].

Chapter 3 addresses likelihood-based generative modeling in the form of a new flow model architecture and a matching lossless compression algorithm. A flow model is a parameterized smooth transformation f_θ between data $\mathbf{x}$ and latents $\mathbf{z} = f_\theta(\mathbf{x})$. A standard normal prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ induces through f_θ a distribution on data given by $-\log p_\theta(\mathbf{x}) = -\log p(f_\theta(\mathbf{x})) - \log |\det \mathbf{J}_\theta(\mathbf{x})|$, where $\mathbf{J}_\theta$ is the Jacobian of f_θ. One criterion for a useful flow model architecture is that this expression is tractable to evaluate and differentiate with respect

to θ, enabling efficient training by maximum likelihood. Another criterion is that the inverse $\mathbf{x} = f_\theta^{-1}(\mathbf{z})$ is easy to evaluate, which enables samples to be generated efficiently. A notable group of architectures that satisfies both criteria is the RealNVP family of models, which have seen recent successes in image and audio synthesis [20, 21, 52, 80]. Inspired by these successes, we propose Flow++, an improved flow model architecture in this family. We also improve the training procedure of flow models in general by introducing variational dequantization, a mechanism to fit continuous density models to discrete data using variational inference.

We then present our computationally efficient compression algorithm, local bits-back coding, which attains codelengths that match Flow++'s negative log probabilities. This means that we produce a lossless code that describes data $\mathbf{x}$ using approximately $-\log p_\theta(\mathbf{x})\delta$ bits, where δ is a small discretization volume of the data space. Backed by Flow++ and other RealNVP-type flows, our encoders and decoders run in linear time and space in a parallelizable fashion over data dimensions. We also show how to increase δ to a large discretization volume for discrete data via a compression interpretation of variational dequantization, and by doing so, we achieve state of the art lossless codelengths on the CIFAR10 and downsampled ImageNet datasets.

Our compression work confronts the fact that naively applying existing codes like Huffman coding [45] requires computing the model likelihood for all possible values of the data, expending computational resources that scale exponentially with the data dimension. This inefficiency stems from the lack of assumptions about the generative model's structure, so coding algorithms must be tailored to specific types of generative models to achieve computational efficiency. While there is already a rich literature of tailored coding algorithms for autoregressive models and variational autoencoders built from conditional distributions already tractable for coding [88, 23, 35, 26, 102], prior to our work there were no such algorithms for general types of flow models [67]. Our contributions, published in ICML 2019 [38] and NeurIPS 2019 [40], are exactly these tailor-made efficient coding algorithms for flow models.

Chapter 2

Imitation Learning

2.1 Introduction

This chapter addresses a specific setting of imitation learning—the problem of learning to perform a task from expert demonstrations—in which the learner is given only samples of trajectories from the expert, is not allowed to query the expert for more data while training, and is not provided a reinforcement signal of any kind. Two standard approaches suitable for this imitation learning setting are behavioral cloning [79], which learns a policy as a supervised learning problem over state-action pairs from expert trajectories, and inverse reinforcement learning [91, 71], which finds a cost function under which the expert is uniquely optimal.

Behavioral cloning, while appealingly simple, only tends to succeed with large amounts of data, due to compounding error caused by covariate shift [89, 90]. Inverse reinforcement learning (IRL), on the other hand, learns a cost function that prioritizes entire trajectories over others, so compounding error, a problem for methods that fit single-timestep decisions, is not an issue. Accordingly, IRL has succeeded in a wide range of problems, from predicting behaviors of taxi drivers [112] to planning footsteps for quadruped robots [83].

Unfortunately, many IRL algorithms are extremely expensive to run, requiring reinforcement learning in an inner loop. Scaling IRL methods to large environments has thus been the focus of much recent work [25, 60]. Fundamentally, however, IRL learns a cost function, which explains expert behavior but does not directly tell the learner how to act. Given that the learner's true goal often is to take actions imitating the expert—indeed, many IRL algorithms are evaluated on the quality of the optimal actions of the costs they learn—why, then, must we learn a cost function, if doing so possibly incurs significant computational expense yet fails to directly yield actions?

We desire an algorithm that tells us explicitly how to act by directly learning a policy. To develop such an algorithm, we begin in Section 2.3, where we characterize the policy given by running reinforcement learning on a cost function learned by maximum causal entropy IRL [112, 113]. Our characterization introduces a framework for directly learning policies

from data, bypassing any intermediate IRL step.

Then, we instantiate our framework in Sections 2.4 and 2.5 with a new model-free imitation learning algorithm. We show that our resulting algorithm is intimately connected to generative adversarial networks [31], a technique from the deep learning community that has led to recent successes in modeling distributions of natural images: our algorithm harnesses generative adversarial training to fit distributions of states and actions defining expert behavior. We test our algorithm in Section 2.6, where we find that it outperforms competing methods by a wide margin in training policies for complex, high-dimensional physics-based control tasks over various amounts of expert data.

2.2 Background

2.2.1 Preliminaries

$\overline{\mathbb{R}}$ will denote the extended real numbers $\mathbb{R} \cup \{\infty\}$. Section 2.3 will work with finite state and action spaces $\mathcal{S}$ and $\mathcal{A}$, but our algorithms and experiments later in the paper will run in high-dimensional continuous environments. Π is the set of all stationary stochastic policies that take actions in $\mathcal{A}$ given states in $\mathcal{S}$; successor states are drawn from the dynamics model $P(s'|s,a)$. We work in the γ-discounted infinite horizon setting, and we will use an expectation with respect a policy $\pi \in \Pi$ to denote an expectation with respect to the trajectory it generates: $\mathbb{E}_\pi[c(s,a)] \coloneqq \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)\right]$, where $s_0 \sim p_0$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$ for $t \geq 0$. We will use $\hat{\mathbb{E}}_\tau$ to denote an empirical expectation with respect to trajectory samples τ, and we will always use π_E to refer to the expert policy.

2.2.2 Inverse reinforcement learning

Suppose we are given an expert policy π_E that we wish to rationalize with IRL. For the remainder of this paper, we will adopt and assume the existence of solutions of maximum causal entropy IRL [112, 113], which fits a cost function from a family of functions $\mathcal{C}$ with the optimization problem

$$\underset{c \in \mathcal{C}}{\text{maximize}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s,a)] \right) - \mathbb{E}_{\pi_E}[c(s,a)] \tag{2.1}$$

where $H(\pi) \coloneqq \mathbb{E}_\pi[-\log \pi(a|s)]$ is the γ-discounted causal entropy [8] of the policy π. In practice, π_E will only be provided as a set of trajectories sampled by executing π_E in the environment, so the expected cost of π_E in Eq. (2.1) is estimated using these samples. Maximum causal entropy IRL looks for a cost function $c \in \mathcal{C}$ that assigns low cost to the expert policy and high cost to other policies, thereby allowing the expert policy to be found via a certain reinforcement learning procedure:

$$\text{RL}(c) = \arg\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s,a)] \tag{2.2}$$

which maps a cost function to high-entropy policies that minimize the expected cumulative cost.

2.3 Characterizing the induced optimal policy

To begin our search for an imitation learning algorithm that both bypasses an intermediate IRL step and is suitable for large environments, we will study policies found by reinforcement learning on costs learned by IRL on the largest possible set of cost functions $\mathcal{C}$ in Eq. (2.1): *all* functions $\mathbb{R}^{\mathcal{S}\times\mathcal{A}} = \{c : \mathcal{S}\times\mathcal{A} \to \mathbb{R}\}$. Using expressive cost function classes, like Gaussian processes [61] and neural networks [25], is crucial to properly explain complex expert behavior without meticulously hand-crafted features. Here, we investigate the best IRL can do with respect to expressiveness by examining its capabilities with $\mathcal{C} = \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$.

Of course, with such a large $\mathcal{C}$, IRL can easily overfit when provided a finite dataset. Therefore, we will incorporate a (closed, proper) convex cost function regularizer $\psi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \overline{\mathbb{R}}$ into our study. Note that convexity is a not particularly restrictive requirement: ψ must be convex as a function defined on all of $\mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, not as a function defined on a small parameter space; indeed, the cost regularizers of Finn et al. [25], effective for a range of robotic manipulation tasks, satisfy this requirement. Interestingly, ψ will play a central role in our discussion and will not serve as a nuisance in our analysis.

Let us define an IRL primitive procedure, which finds a cost function such that the expert performs better than all other policies, with the cost regularized by ψ:

$$\mathrm{IRL}_\psi(\pi_E) = \arg\max_{c\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}} -\psi(c) + \left(\min_{\pi\in\Pi} -H(\pi) + \mathbb{E}_\pi[c(s,a)]\right) - \mathbb{E}_{\pi_E}[c(s,a)] \tag{2.3}$$

Let $\tilde{c} \in \mathrm{IRL}_\psi(\pi_E)$. We are interested in a policy given by $\mathrm{RL}(\tilde{c})$—this is the policy given by running reinforcement learning on the output of IRL. To characterize $\mathrm{RL}(\tilde{c})$, let us first define for a policy $\pi \in \Pi$ its occupancy measure $\rho_\pi : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$ as $\rho_\pi(s,a) = \pi(a|s)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$. The occupancy measure can be interpreted as the unnormalized distribution of state-action pairs that an agent encounters when navigating the environment with the policy π, and it allows us to write $\mathbb{E}_\pi[c(s,a)] = \sum_{s,a}\rho_\pi(s,a)c(s,a)$ for any cost function c. We will also need the concept of a convex conjugate: for a function $f : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \overline{\mathbb{R}}$, its convex conjugate $f^* : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \overline{\mathbb{R}}$ is given by $f^*(x) = \sup_{y\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}} x^T y - f(y)$.

Now, we are prepared to characterize $\mathrm{RL}(\tilde{c})$, the policy learned by RL on the cost recovered by IRL:

Proposition 2.3.1. $\mathrm{RL}\circ\mathrm{IRL}_\psi(\pi_E) = \arg\min_{\pi\in\Pi} -H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E})$ (2.4)

The proof of Proposition 2.3.1, deferred to the end of this section, relies on the observation that the optimal cost function and policy form a saddle point of a certain function. IRL finds one coordinate of this saddle point, and running RL on the output of IRL reveals the other coordinate.

Proposition 2.3.1 tells us that ψ-regularized inverse reinforcement learning, implicitly, seeks a policy whose occupancy measure is close to the expert's, as measured by ψ^*. Enticingly, this suggests that various settings of ψ lead to various imitation learning algorithms that directly solve the optimization problem given by Proposition 2.3.1. We explore such algorithms in Sections 2.4 and 2.5, where we show that certain settings of ψ lead to both existing algorithms and a novel one.

The special case when ψ is a constant function is particularly illuminating, so we state and show it first using concepts from convex optimization.

Proposition 2.3.2. *Suppose $\rho_{\pi_E} > 0$. If ψ is a constant function, $\tilde{c} \in \mathrm{IRL}_\psi(\pi_E)$, and $\tilde{\pi} \in \mathrm{RL}(\tilde{c})$, then $\rho_{\tilde{\pi}} = \rho_{\pi_E}$.*

In other words, if there were no cost regularization at all, the recovered policy will exactly match the expert's occupancy measure. (The condition $\rho_{\pi_E} > 0$, inherited from Ziebart et al. [113], simplifies our discussion and in fact guarantees the existence of $\tilde{c} \in \mathrm{IRL}_\psi(\pi_E)$. Elsewhere in the paper, as mentioned in Section 2.2, we assume the IRL problem has a solution.)

To show Proposition 2.3.2, we need the basic result that the set of valid occupancy measures $\mathcal{D} \coloneqq \{\rho_\pi : \pi \in \Pi\}$ can be written as a feasible set of affine constraints [81]: if $p_0(s)$ is the distribution of starting states and $P(s'|s,a)$ is the dynamics model, then

$$\mathcal{D} = \left\{ \rho : \rho \geq 0 \quad \text{and} \quad \sum_a \rho(s,a) = p_0(s) + \gamma \sum_{s',a} P(s|s',a)\rho(s',a) \;\; \forall\, s \in \mathcal{S} \right\}$$

Furthermore, there is a one-to-one correspondence between Π and $\mathcal{D}$:

Lemma 2.3.1 (Theorem 2 of Syed et al. [98])**.** *If $\rho \in \mathcal{D}$, then ρ is the occupancy measure for $\pi_\rho(a|s) \coloneqq \rho(s,a)/\sum_{a'} \rho(s,a')$, and π_ρ is the only policy whose occupancy measure is ρ.*

We are therefore justified in writing π_ρ to denote the unique policy for an occupancy measure ρ. We also need a lemma that lets us speak about causal entropies of occupancy measures:

Lemma 2.3.2. *Let $\bar{H}(\rho) = -\sum_{s,a} \rho(s,a) \log(\rho(s,a)/\sum_{a'} \rho(s,a'))$. Then, $\bar{H}$ is strictly concave, and for all $\pi \in \Pi$ and $\rho \in \mathcal{D}$, we have $H(\pi) = \bar{H}(\rho_\pi)$ and $\bar{H}(\rho) = H(\pi_\rho)$.*

The proof of this lemma is in [36]. Lemma 2.3.1 and Lemma 2.3.2 together allow us to freely switch between policies and occupancy measures when considering functions involving causal entropy and expected costs, as in the following lemma:

Lemma 2.3.3. *If $L(\pi, c) = -H(\pi) + \mathbb{E}_\pi[c(s,a)]$ and $\bar{L}(\rho, c) = -\bar{H}(\rho) + \sum_{s,a} \rho(s,a)c(s,a)$, then, for all cost functions c, $L(\pi, c) = \bar{L}(\rho_\pi, c)$ for all policies $\pi \in \Pi$, and $\bar{L}(\rho, c) = L(\pi_\rho, c)$ for all occupancy measures $\rho \in \mathcal{D}$.*

Now, we are ready to verify Proposition 2.3.2.

Proof of Proposition 2.3.2. Define $\bar{L}(\rho, c) = -\bar{H}(\rho) + \sum_{s,a} c(s,a)(\rho(s,a) - \rho_E(s,a))$. Given that ψ is a constant function, we have the following, due to Lemma 2.3.3:

$$\tilde{c} \in \mathrm{IRL}_\psi(\pi_E) = \arg\max_{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} \min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] + \text{const.} \tag{2.5}$$

$$= \arg\max_{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} \min_{\rho \in \mathcal{D}} -\bar{H}(\rho) + \sum_{s,a} \rho(s,a)c(s,a) - \sum_{s,a} \rho_E(s,a)c(s,a) \tag{2.6}$$

$$= \arg\max_{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}} \min_{\rho \in \mathcal{D}} \bar{L}(\rho, c). \tag{2.7}$$

This is the dual of the optimization problem

$$\underset{\rho \in \mathcal{D}}{\text{minimize}} -\bar{H}(\rho) \quad \text{subject to} \quad \rho(s,a) = \rho_E(s,a) \quad \forall\ s \in \mathcal{S}, a \in \mathcal{A} \tag{2.8}$$

with Lagrangian $\bar{L}$, for which the costs $c(s,a)$ serve as dual variables for equality constraints. Thus, $\tilde{c}$ is a dual optimum for (2.8). In addition, strong duality holds for (2.8): $\mathcal{D}$ is compact and convex, $-\bar{H}$ is convex, and, since $\rho_E > 0$, there exists a feasible point in the relative interior of the domain $\mathcal{D}$. Moreover, Lemma 2.3.2 guarantees that $-\bar{H}$ is in fact strictly convex, so the primal optimum can be uniquely recovered from the dual optimum [9, Section 5.5.5] via $\tilde{\rho} = \arg\min_{\rho \in \mathcal{D}} \bar{L}(\rho, \tilde{c}) = \arg\min_{\rho \in \mathcal{D}} -\bar{H}(\rho) + \sum_{s,a} \tilde{c}(s,a)\rho(s,a) = \rho_E$, where the first equality indicates that $\tilde{\rho}$ is the unique minimizer of $\bar{L}(\cdot, \tilde{c})$, and the third follows from the constraints in the primal problem (2.8). But if $\tilde{\pi} \in \mathrm{RL}(\tilde{c})$, then Lemma 2.3.3 implies $\rho_{\tilde{\pi}} = \tilde{\rho} = \rho_E$. □

Finally, we return to the more general Proposition 2.3.1.

Proof of Proposition 2.3.1. Let $\tilde{c} \in \mathrm{IRL}_\psi(\pi_E)$, $\tilde{\pi} \in \mathrm{RL}(\tilde{c}) = \mathrm{RL} \circ \mathrm{IRL}_\psi(\pi_E)$, and

$$\pi_A \in \arg\min_\pi -H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}) \tag{2.9}$$

$$= \arg\min_\pi \sup_{c \in \mathcal{C}} -H(\pi) - \psi(c) + \sum_{s,a} (\rho_\pi(s,a) - \rho_{\pi_E}(s,a))c(s,a) \tag{2.10}$$

We wish to show that $\pi_A = \tilde{\pi}$. To do this, let ρ_A be the occupancy measure of π_A, let $\tilde{\rho}$ be the occupancy measure of $\tilde{\pi}$, and define $\bar{L} : \mathcal{D} \times \mathcal{C} \to \mathbb{R}$ by

$$\bar{L}(\rho, c) = -\bar{H}(\rho) - \psi(c) + \sum_{s,a} \rho(s,a)c(s,a) - \sum_{s,a} \rho_{\pi_E}(s,a)c(s,a). \tag{2.11}$$

The following relationships then hold, due to Lemma 2.3.1:

$$\rho_A \in \arg\min_{\rho \in \mathcal{D}} \sup_{c \in \mathcal{C}} \bar{L}(\rho, c), \tag{2.12}$$

$$\tilde{c} \in \arg\max_{c \in \mathcal{C}} \min_{\rho \in \mathcal{D}} \bar{L}(\rho, c), \tag{2.13}$$

$$\tilde{\rho} \in \arg\min_{\rho \in \mathcal{D}} \bar{L}(\rho, \tilde{c}). \tag{2.14}$$

(Recall that we can write Eq. (2.13) because we assumed the existence of a solution to the IRL problem Eq. (2.1).) Now $\mathcal{D}$ is compact and convex and $\mathcal{C}$ is convex; furthermore, due to convexity of $-\bar{H}$ and ψ, we also have that $\bar{L}(\cdot, c)$ is convex for all c, and that $\bar{L}(\rho, \cdot)$ is concave for all ρ, and hence:

$$\min_{\rho \in \mathcal{D}} \sup_{c \in \mathcal{C}} \bar{L}(\rho, c) = \max_{c \in \mathcal{C}} \min_{\rho \in \mathcal{D}} \bar{L}(\rho, c) \tag{2.15}$$

Consequently, from Eqs. (2.12) and (2.13), $(\rho_A, \tilde{c})$ is a saddle point of $\bar{L}$. In particular,

$$\rho_A \in \arg\min_{\rho \in \mathcal{D}} \bar{L}(\rho, \tilde{c}). \tag{2.16}$$

Because $\bar{L}(\cdot, c)$ is strictly convex for all c (Lemma 2.3.2), Eqs. (2.14) and (2.16) imply $\rho_A = \tilde{\rho}$. Since policies corresponding to occupancy measures are unique (Lemma 2.3.1), $\pi_A = \tilde{\pi}$. □

Let us summarize our conclusions. First, *IRL is a dual of an occupancy measure matching problem*, and the recovered cost function is the dual optimum. Classic IRL algorithms that solve reinforcement learning repeatedly in an inner loop, such as the algorithm of Ziebart et al. [112] that runs a variant of value iteration in an inner loop, can be interpreted as a form of dual ascent, in which one repeatedly solves the primal problem (reinforcement learning) with fixed dual values (costs). Dual ascent is effective if solving the unconstrained primal is efficient, but in the case of IRL, it amounts to reinforcement learning! Second, *the induced optimal policy is the primal optimum.* The induced optimal policy is obtained by running RL after IRL, which is exactly the act of recovering the primal optimum from the dual optimum; that is, optimizing the Lagrangian with the dual variables fixed at the dual optimum values. Strong duality implies that this induced optimal policy is indeed the primal optimum, and therefore matches occupancy measures with the expert. IRL is traditionally defined as the act of finding a cost function such that the expert policy is uniquely optimal, but we can alternatively view IRL as a procedure that tries to *induce a policy that matches the expert's occupancy measure.*

2.4 Practical occupancy measure matching

We saw in Proposition 2.3.2 that if ψ is constant, the resulting primal problem (2.8) simply matches occupancy measures with expert at all states and actions. Such an algorithm is not practically useful. In reality, the expert trajectory distribution will be provided only as a finite set of samples, so in large environments, most of the expert's occupancy measure values will be small, and exact occupancy measure matching will force the learned policy to rarely visit these unseen state-action pairs simply due to lack of data. Furthermore, in the cases in which we would like to use function approximation to learn parameterized policies π_θ, the resulting optimization problem of finding an appropriate θ would have an intractably large number of constraints when the environment is large: as many constraints as points in $\mathcal{S} \times \mathcal{A}$.

Keeping in mind that we wish to eventually develop an imitation learning algorithm suitable for large environments, we would like to relax Eq. (2.8) into the following form, motivated by Proposition 2.3.1:

$$\underset{\pi}{\text{minimize}} \; d_\psi(\rho_\pi, \rho_E) - H(\pi) \tag{2.17}$$

by modifying the IRL regularizer ψ so that $d_\psi(\rho_\pi, \rho_E) \coloneqq \psi^*(\rho_\pi - \rho_E)$ smoothly penalizes violations in difference between the occupancy measures.

2.4.1 Entropy-regularized apprenticeship learning

It turns out that with certain settings of ψ, Eq. (2.17) takes on the form of regularized variants of existing *apprenticeship learning* algorithms, which indeed do scale to large environments with parameterized policies [37]. For a class of cost functions $\mathcal{C} \subset \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, an apprenticeship learning algorithm finds a policy that performs better than the expert across $\mathcal{C}$, by optimizing the objective

$$\underset{\pi}{\text{minimize}} \; \max_{c\in\mathcal{C}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] \tag{2.18}$$

Classic apprenticeship learning algorithms restrict $\mathcal{C}$ to convex sets given by linear combinations of basis functions $f_1, \ldots, f_d$, which give rise a feature vector

$$f(s,a) = [f_1(s,a), \ldots, f_d(s,a)]$$

for each state-action pair. Abbeel and Ng [1] and Syed et al. [98] use, respectively,

$$\mathcal{C}_{\text{linear}} = \{\textstyle\sum_i w_i f_i \;:\; \|w\|_2 \le 1\} \quad \text{and} \quad \mathcal{C}_{\text{convex}} = \{\textstyle\sum_i w_i f_i \;:\; \sum_i w_i = 1, w_i \ge 0 \;\forall i\}\,. \tag{2.19}$$

$\mathcal{C}_{\text{linear}}$ leads to feature expectation matching [1], which minimizes ℓ_2 distance between expected feature vectors: $\max_{c\in\mathcal{C}_{\text{linear}}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] = \|\mathbb{E}_\pi[f(s,a)] - \mathbb{E}_{\pi_E}[f(s,a)]\|_2$. Meanwhile, $\mathcal{C}_{\text{convex}}$ leads to MWAL [97] and LPAL [98], which minimize worst-case excess cost among the individual basis functions, as $\max_{c\in\mathcal{C}_{\text{convex}}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] = \max_{i\in\{1,\ldots,d\}} \mathbb{E}_\pi[f_i(s,a)] - \mathbb{E}_{\pi_E}[f_i(s,a)]$.

We now show how Eq. (2.18) is a special case of Eq. (2.17) with a certain setting of ψ. With the indicator function $\delta_\mathcal{C} : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \overline{\mathbb{R}}$, defined by $\delta_\mathcal{C}(c) = 0$ if $c \in \mathcal{C}$ and $+\infty$ otherwise, we can write the apprenticeship learning objective (2.18) as

$$\max_{c\in\mathcal{C}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] = \max_{c\in\mathbb{R}^{\mathcal{S}\times\mathcal{A}}} -\delta_\mathcal{C}(c) + \sum_{s,a}(\rho_\pi(s,a) - \rho_{\pi_E}(s,a))c(s,a) = \delta^*_\mathcal{C}(\rho_\pi - \rho_{\pi_E})$$

Therefore, we see that entropy-regularized apprenticeship learning

$$\underset{\pi}{\text{minimize}} \; -H(\pi) + \max_{c\in\mathcal{C}} \mathbb{E}_\pi[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] \tag{2.20}$$

is equivalent to performing RL following IRL with cost regularizer $\psi = \delta_\mathcal{C}$, which forces the implicit IRL procedure to recover a cost function lying in $\mathcal{C}$. Note that we can scale the policy's entropy regularization strength in Eq. (2.20) by scaling $\mathcal{C}$ by a constant α as $\{\alpha c : c \in \mathcal{C}\}$, recovering the original apprenticeship objective (2.18) by taking $\alpha \to \infty$.

2.4.2 Cons of apprenticeship learning

It is known that apprenticeship learning algorithms generally do not recover expert-like policies if $\mathcal{C}$ is too restrictive [98, Section 1]—which is often the case for the linear subspaces used by feature expectation matching, MWAL, and LPAL, unless the basis functions $f_1, \ldots, f_d$ are very carefully designed. Intuitively, unless the true expert cost function (assuming it exists) lies in $\mathcal{C}$, there is no guarantee that if π performs better than π_E on all of $\mathcal{C}$, then π equals π_E. With the aforementioned insight based on Proposition 2.3.1 that apprenticeship learning is equivalent to RL following IRL, we can understand exactly why apprenticeship learning may fail to imitate: it forces π_E to be encoded as an element of $\mathcal{C}$. If $\mathcal{C}$ does not include a cost function that explains expert behavior well, then attempting to recover a policy from such an encoding will not succeed.

2.4.3 Pros of apprenticeship learning

While restrictive cost classes $\mathcal{C}$ may not lead to exact imitation, apprenticeship learning with such $\mathcal{C}$ can scale to large state and action spaces with policy function approximation. Ho et al. [37] rely on the following policy gradient formula for the apprenticeship objective (2.18) for a parameterized policy π_θ:

$$\begin{aligned}&\nabla_\theta \max_{c\in\mathcal{C}} \mathbb{E}_{\pi_\theta}[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)] = \nabla_\theta \mathbb{E}_{\pi_\theta}[c^*(s,a)] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q_{c^*}(s,a)\right] \\ &\text{where } c^* = \arg\max_{c\in\mathcal{C}} \mathbb{E}_{\pi_\theta}[c(s,a)] - \mathbb{E}_{\pi_E}[c(s,a)],\ Q_{c^*}(\bar{s},\bar{a}) = \mathbb{E}_{\pi_\theta}[c^*(\bar{s},\bar{a}) \mid s_0 = \bar{s}, a_0 = \bar{a}]\end{aligned} \tag{2.21}$$

Observing that Eq. (2.21) is the policy gradient for a reinforcement learning objective with cost c^*, Ho et al. propose an algorithm that alternates between two steps:

1. Sample trajectories of the current policy π_{θ_i} by simulating in the environment, and fit a cost function c_i^*, as defined in Eq. (2.21). For the cost classes $\mathcal{C}_{\text{linear}}$ and $\mathcal{C}_{\text{convex}}$ (2.19), this cost fitting amounts to evaluating simple analytical expressions [37].

2. Form a gradient estimate with Eq. (2.21) with c_i^* and the sampled trajectories, and take a trust region policy optimization (TRPO) [94] step to produce $\pi_{\theta_{i+1}}$.

This algorithm relies crucially on the TRPO policy step, which is a natural gradient step constrained to ensure that $\pi_{\theta_{i+1}}$ does not stray too far π_{θ_i}, as measured by KL divergence between the two policies averaged over the states in the sampled trajectories. This carefully constructed step scheme ensures that the algorithm does not diverge due to high noise in estimating the gradient (2.21). We refer the reader to Schulman et al. [94] for more details on TRPO.

With the TRPO step scheme, Ho et al. were able train large neural network policies for apprenticeship learning with linear cost function classes (2.19) in environments with hundreds of observation dimensions. Their use of these linear cost function classes, however, limits

their approach to settings in which expert behavior is well-described by such classes. We will draw upon their algorithm to develop an imitation learning method that both scales to large environments and imitates arbitrarily complex expert behavior. To do so, we first turn to proposing a new regularizer ψ that wields more expressive power than the regularizers corresponding to $\mathcal{C}_{\text{linear}}$ and $\mathcal{C}_{\text{convex}}$ (2.19).

2.5 Generative adversarial imitation learning

As discussed in Section 2.4, the constant regularizer leads to an imitation learning algorithm that exactly matches occupancy measures, but is intractable in large environments. The indicator regularizers for the linear cost function classes (2.19), on the other hand, lead to algorithms incapable of exactly matching occupancy measures without careful tuning, but are tractable in large environments. We propose the following new cost regularizer that combines the best of both worlds, as we will show in the coming sections:

$$\psi_{\text{GA}}(c) \coloneqq \begin{cases} \mathbb{E}_{\pi_E}[g(c(s,a))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \quad \text{where } g(x) = \begin{cases} -x - \log(1 - e^x) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases} \tag{2.22}$$

This regularizer places low penalty on cost functions c that assign an amount of negative cost to expert state-action pairs; if c, however, assigns large costs (close to zero, which is the upper bound for costs feasible for ψ_{GA}) to the expert, then ψ_{GA} will heavily penalize c. An interesting property of ψ_{GA} is that it is an average over expert data, and therefore can adjust to arbitrary expert datasets. The indicator regularizers $\delta_{\mathcal{C}}$, used by the linear apprenticeship learning algorithms described in Section 2.4, are always fixed, and cannot adapt to data as ψ_{GA} can. Perhaps the most important difference between ψ_{GA} and $\delta_{\mathcal{C}}$, however, is that $\delta_{\mathcal{C}}$ forces costs to lie in a small subspace spanned by finitely many basis functions, whereas ψ_{GA} allows for any cost function, as long as it is negative everywhere.

Our choice of ψ_{GA} is motivated by the following fact, shown in the full paper [36]:

$$\psi^*_{\text{GA}}(\rho_\pi - \rho_{\pi_E}) = \sup_{D \in (0,1)^{\mathcal{S}\times\mathcal{A}}} \mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] \tag{2.23}$$

where the supremum ranges over discriminative classifiers $D : \mathcal{S} \times \mathcal{A} \to (0,1)$. Equation (2.23) is proportional to the optimal negative log loss of the binary classification problem of distinguishing between state-action pairs of π and π_E. It turns out that this optimal loss is, up to a constant shift and scaling, the Jensen-Shannon divergence $D_{\text{JS}}(\bar{\rho}_\pi, \bar{\rho}_{\pi_E}) \coloneqq D_{\text{KL}}\left(\bar{\rho}_\pi \| (\bar{\rho}_\pi + \bar{\rho}_E)/2\right) + D_{\text{KL}}\left(\bar{\rho}_E \| (\bar{\rho}_\pi + \bar{\rho}_E)/2\right)$, which is a squared metric between the normalized occupancy distributions $\bar{\rho}_\pi = (1-\gamma)\rho_\pi$ and $\bar{\rho}_{\pi_E} = (1-\gamma)\rho_{\pi_E}$ [31, 72]. Treating the causal entropy H as a policy regularizer controlled by $\lambda \geq 0$ and dropping the $1 - \gamma$ occupancy measure normalization for clarity, we obtain a new imitation learning algorithm:

$$\underset{\pi}{\text{minimize}} \ \psi^*_{\text{GA}}(\rho_\pi - \rho_{\pi_E}) - \lambda H(\pi) = D_{\text{JS}}(\rho_\pi, \rho_{\pi_E}) - \lambda H(\pi), \tag{2.24}$$

which finds a policy whose occupancy measure minimizes Jensen-Shannon divergence to the expert's. Equation (2.24) minimizes a true metric between occupancy measures, so, unlike linear apprenticeship learning algorithms, it can imitate expert policies exactly.

2.5.1 Algorithm

Equation (2.24) draws a connection between imitation learning and generative adversarial networks [31], which train a generative model G by having it confuse a discriminative classifier D. The job of D is to distinguish between the distribution of data generated by G and the true data distribution. When D cannot distinguish data generated by G from the true data, then G has successfully matched the true data. In our setting, the learner's occupancy measure ρ_π is analogous to the data distribution generated by G, and the expert's occupancy measure ρ_{π_E} is analogous to the true data distribution.

We now present a practical imitation learning algorithm, called *generative adversarial imitation learning* or GAIL (Algorithm 2.1), designed to work in large environments. GAIL solves Eq. (2.24) by finding a saddle point (π, D) of the expression

$$\mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi) \tag{2.25}$$

with both π and D represented using function approximators: GAIL fits a parameterized policy π_θ, with weights θ, and a discriminator network $D_w : \mathcal{S} \times \mathcal{A} \to (0, 1)$, with weights w. GAIL alternates between an Adam [51] gradient step on w to increase Eq. (2.25) with respect to D, and a TRPO step on θ to decrease Eq. (2.25) with respect to π (we derive an estimator for the causal entropy gradient $\nabla_\theta H(\pi_\theta)$ in [36]). The TRPO step serves the same purpose as it does with the apprenticeship learning algorithm of Ho et al. [37]: it prevents the policy from changing too much due to noise in the policy gradient. The discriminator network can be interpreted as a local cost function providing learning signal to the policy—specifically, taking a policy step that decreases expected cost with respect to the cost function $c(s, a) = \log D(s, a)$ will move toward expert-like regions of state-action space, as classified by the discriminator.

2.6 Experiments

We evaluated GAIL against baselines on 9 physics-based control tasks, ranging from low-dimensional control tasks from the classic RL literature—the cartpole [5], acrobot [28], and mountain car [69]—to difficult high-dimensional tasks such as a 3D humanoid locomotion, solved only recently by model-free reinforcement learning [95, 94]. All environments, other than the classic control tasks, were simulated with MuJoCo [101]. See the full paper [36] for a complete description of all the tasks.

Each task comes with a true cost function, defined in the OpenAI Gym [10]. We first generated expert behavior for these tasks by running TRPO [94] on these true cost functions to create expert policies. Then, to evaluate imitation performance with respect to sample

Algorithm 2.1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{2.26}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \,|\, s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \tag{2.27}$$

6: **end for**

complexity of expert data, we sampled datasets of varying trajectory counts from the expert policies. The trajectories constituting each dataset each consisted of about 50 state-action pairs. We tested GAIL against three baselines:

1. Behavioral cloning: a given dataset of state-action pairs is split into 70% training data and 30% validation data. The policy is trained with supervised learning, using Adam [51] with minibatches of 128 examples, until validation error stops decreasing.

2. Feature expectation matching (FEM): the algorithm of Ho et al. [37] using the cost function class $\mathcal{C}_{\text{linear}}$ (2.19) of Abbeel and Ng [1]

3. Game-theoretic apprenticeship learning (GTAL): the algorithm of Ho et al. [37] using the cost function class $\mathcal{C}_{\text{convex}}$ (2.19) of Syed and Schapire [97]

We used all algorithms to train policies of the same neural network architecture for all tasks: two hidden layers of 100 units each, with tanh nonlinearities in between. The discriminator networks for GAIL also used the same architecture. All networks were always initialized randomly at the start of each trial. For each task, we gave FEM, GTAL, and GAIL exactly the same amount of environment interaction for training. We ran all algorithms 5-7 times over different random seeds in all environments except Humanoid, due to time restrictions.

Figures 2.1 and 2.2 depict the results (except for Humanoid, shading in these figures indicate standard deviation over 5-7 reruns), and [36] provides exact performance numbers and details of our experiment pipeline, including expert data sampling and algorithm hyperparameters.

We found that on the classic control tasks (cartpole, acrobot, and mountain car), behavioral cloning generally suffered in expert data efficiency compared to FEM and GTAL, which for the most part were able produce policies with near-expert performance with a wide range of

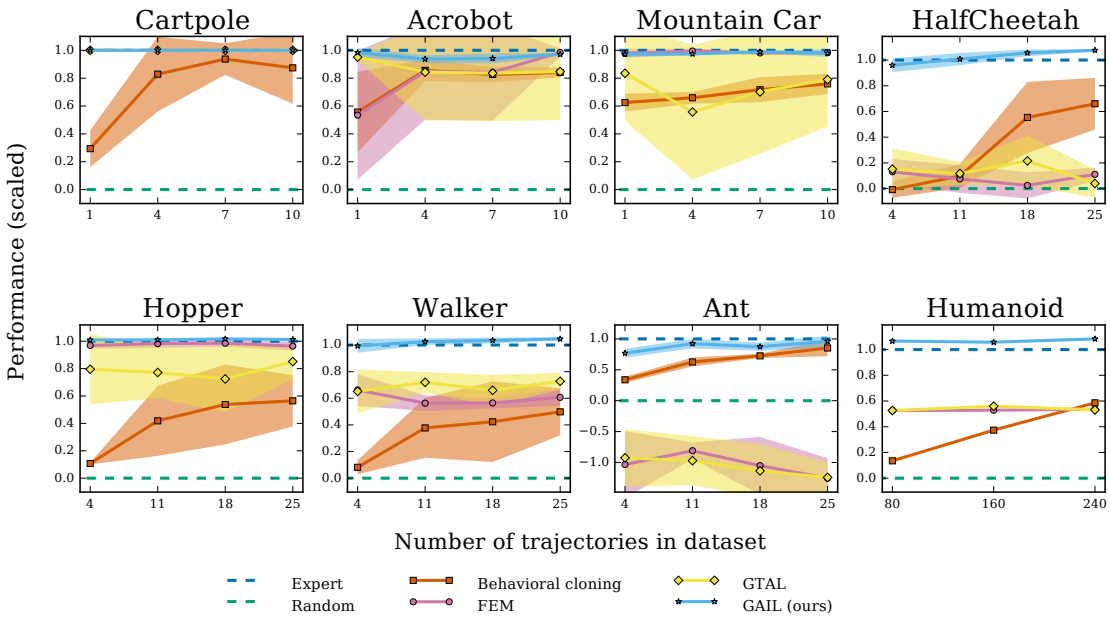


Figure 2.1: Performance of learned policies. The y-axis is negative cost, scaled so that the expert achieves 1 and a random policy achieves 0.

dataset sizes, albeit with large variance over different random initializations of the policy. On these tasks, GAIL consistently produced policies performing better than behavioral cloning, FEM, and GTAL.

However, behavioral cloning performed excellently on the Reacher task, on which it was more sample efficient than GAIL. We were able to slightly improve GAIL's performance on Reacher using causal entropy regularization—in the 4-trajectory setting, the improvement from $\lambda = 0$ to $\lambda = 10^{-3}$ was statistically significant over training reruns, according to a one-sided Wilcoxon rank-sum test with $p = .05$. We used no causal entropy regularization for all other tasks.

On the other MuJoCo environments, GAIL almost always achieved at least 70% of expert performance for all dataset sizes we tested and reached it exactly with the larger datasets, with very little variance among random seeds. The baseline algorithms generally could not reach expert performance even with the largest datasets. FEM and GTAL performed poorly for Ant, producing policies consistently worse than a policy

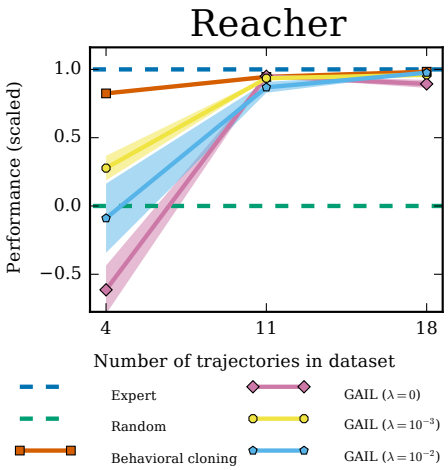


Figure 2.2: Causal entropy regularization λ on Reacher.

that chooses actions uniformly at random. Behavioral cloning was able to reach satisfactory performance with enough data on HalfCheetah, Hopper, Walker, and Ant, but was unable to achieve more than 60% for Humanoid, on which GAIL achieved exact expert performance for all tested dataset sizes.

2.7 Discussion

As we demonstrated, GAIL is generally quite sample efficient in terms of expert data. However, it is not particularly sample efficient in terms of environment interaction during training. The number of such samples required to estimate the imitation objective gradient (2.27) was comparable to the number needed for TRPO to train the expert policies from reinforcement signals. We believe that we could significantly improve learning speed for GAIL by initializing policy parameters with behavioral cloning, which requires no environment interaction at all.

Fundamentally, our method is model free, so it will generally need more environment interaction than model-based methods. Guided cost learning [25], for instance, builds upon guided policy search [59] and inherits its sample efficiency, but also inherits its requirement that the model is well-approximated by iteratively fitted time-varying linear dynamics. Interestingly, both GAIL and guided cost learning alternate between policy optimization steps and cost fitting (which we called discriminator fitting), even though the two algorithms are derived completely differently.

Our approach builds upon a vast line of work on IRL [112, 1, 98, 97], and hence, just like IRL, our approach does not interact with the expert during training. Our method explores randomly to determine which actions bring a policy's occupancy measure closer to the expert's, whereas methods that do interact with the expert, like DAgger [90], can simply ask the expert for such actions. Ultimately, we believe that a method that combines well-chosen environment models with expert interaction will win in terms of sample complexity of both expert data and environment interaction.

2.8 Recent related work

There have been numerous contributions to imitation learning, inverse reinforcement learning, and related areas since the publication the work presented in this chapter. For example, Finn et al. [24] show a connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. Duan et al. [22] propose one-shot imitation learning, which meta-learns an algorithm that imitates given one demonstration trajectory. Christiano et al. [15] show how to learn from trajectory ranking feedback collected from humans online. Peng et al. [77] learn a variety of locomotion and acrobatic skills in physically simulated characters using imitation learning and deep reinforcement learning. See Osa et al. [73] for a broad review of these areas.

Investigation of directions related to GAIL specifically include the work of Li et al. [62], who propose InfoGAIL to learn latent variables in trajectory data, and the work of Song et al. [96], who investigate GAIL in the multi-agent setting. Fu et al. [27] propose the AIRL algorithm, which learns reward functions from data using a learning procedure similar to GAIL, and Peng et al. [78] demonstrate that a discriminator information bottleneck improves the performance of these algorithms.

Chapter 3

Likelihood-Based Generative Modeling

3.1 Introduction

We now turn to generative modeling in one of its purest forms in an interaction-free setting. In this chapter, we devise a computationally efficient lossless compression algorithm for high dimensional, high entropy data.

To devise a lossless compression algorithm means to devise a uniquely decodable code whose expected length is as close as possible to the entropy of the data. A general recipe for this is to first train a likelihood-based generative model by minimizing cross entropy to the data distribution, and then construct a code that achieves lengths close to the negative log likelihood of the model. This recipe is justified by classic results in information theory that ensure that the second step is possible—in other words, optimizing cross entropy optimizes the performance of some hypothetical compressor compatible with the model. In the following sections, we address both parts of this recipe with computational efficiency of compression as the primary goal.

First, in Sections 3.2 to 3.3, we present our advances in flow models, a type of likelihood-based generative model that is fast for sampling and inference while attaining strong density estimation performance on high entropy datasets. These properties guarantee short codelengths for a compressor compatible with the model while ensuring that natural model operations are computationally efficient. Our new flow model architecture, Flow++, is powered by an improved training procedure for continuous likelihood models and a number of architectural extensions of RealNVP-type flow models [20, 21], and it achieves density estimation performance on images competitive with variational autoencoders and early generations of autoregressive models. This is our contribution to a recent trend of advances in likelihood-based generative models on a wide variety of real world datasets [107, 108, 93, 76, 13, 66, 14, 39, 49, 111, 20, 21, 52, 80, 38, 106, 48, 53, 54, 64].

Second, in Sections 3.4 to 3.7, we construct the first practical coding algorithms for flow models; prior to our work, only intractable algorithms existed for them [67]. We begin with local bits-back coding, our new coding algorithm for general types of flow models that runs

in polynomial time and space with respect to the data dimensionality. Then, we show how to specialize local bits-back coding to Flow++ and other RealNVP-type flows, and in doing so, we produce compression algorithms that run in linear time and space in a fully parallelizable fashion for both encoding and decoding, mirroring the fast sampling and inference paths that distinguish these models from other types of likelihood-based generative models. Finally, we test our compression algorithms on standard image modeling benchmarks, and we find that they indeed are computationally efficient on modern hardware and attain codelengths in close agreement with theoretical predictions.

3.2 Improving flow models

A flow model f is a smooth invertible transformation that maps observed data $\mathbf{x}$ to a standard Gaussian latent variable $\mathbf{z} = f(\mathbf{x})$, as in nonlinear independent component analysis [7, 47, 46]. The key idea in recent flow model designs is to form f by composing a number of individual simple invertible transformations [20, 21, 52, 85, 54, 63]. Explicitly, f is constructed by composing a series of invertible flows as $f(\mathbf{x}) = f_1 \circ \cdots \circ f_L(\mathbf{x})$, with each f_i having a tractable inverse and a tractable Jacobian determinant. This way, sampling is efficient, as it can be performed by computing $f^{-1}(\mathbf{z}) = f_L^{-1} \circ \cdots \circ f_1^{-1}(\mathbf{z})$ for $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and so is training by maximum likelihood, since the model density

$$\log p(\mathbf{x}) = \log \mathcal{N}(f(\mathbf{x}); \mathbf{0}, \mathbf{I}) + \sum_{i=1}^{L} \log \left| \det \frac{\partial f_i}{\partial f_{i-1}} \right| \tag{3.1}$$

is easy to compute and differentiate with respect to the parameters of the flows f_i.

We now describe three modeling inefficiencies in prior work on flow models: (1) uniform noise is a suboptimal dequantization choice that hurts both training loss and generalization; (2) commonly used affine coupling flows are not expressive enough; (3) convolutional layers in the conditioning networks of coupling layers are not powerful enough. Our proposed model, Flow++, consists of a set of improved design choices: (1) variational flow-based dequantization instead of uniform dequantization; (2) logistic mixture CDF coupling flows; (3) self-attention in the conditioning networks of coupling layers.

3.2.1 Dequantization via variational inference

Many real-world datasets, such as CIFAR10 and ImageNet, are recordings of continuous signals quantized into discrete representations. Fitting a continuous density model to discrete data, however, will produce a degenerate solution that places all probability mass on discrete datapoints [104]. A common solution to this problem is to first convert the discrete data distribution into a continuous distribution via a process called "dequantization," and then model the resulting continuous distribution using the continuous density model [104, 21, 93].

3.2.1.1 Uniform dequantization

Dequantization is usually performed in prior work by adding uniform noise to the discrete data over the width of each discrete bin: if each of the D components of the discrete data $\mathbf{x}$ takes on values in $\{0, 1, 2, \ldots, 255\}$, then the dequantized data is given by $\mathbf{y} = \mathbf{x} + \mathbf{u}$, where $\mathbf{u}$ is drawn uniformly from $[0, 1)^D$. Theis et al. [99] note that training a continuous density model p_{model} on uniformly dequantized data $\mathbf{y}$ can be interpreted as maximizing a lower bound on the log-likelihood for a certain discrete model P_{model} on the original discrete data $\mathbf{x}$:

$$P_{\text{model}}(\mathbf{x}) \coloneqq \int_{[0,1)^D} p_{\text{model}}(\mathbf{x} + \mathbf{u})\, d\mathbf{u} \tag{3.2}$$

The argument of Theis et al. [99] proceeds as follows. Letting P_{data} denote the original distribution of discrete data and p_{data} denote the distribution of uniformly dequantized data, Jensen's inequality implies that

$$\begin{aligned}
\mathbb{E}_{\mathbf{y} \sim p_{\text{data}}} \left[\log p_{\text{model}}(\mathbf{y})\right] &= \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \int_{[0,1)^D} \log p_{\text{model}}(\mathbf{x} + \mathbf{u})\, d\mathbf{u} && (3.3)\\
&\leq \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \int_{[0,1)^D} p_{\text{model}}(\mathbf{x} + \mathbf{u})\, d\mathbf{u} && (3.4)\\
&= \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log P_{\text{model}}(\mathbf{x})\right] && (3.5)
\end{aligned}$$

Consequently, maximizing the log-likelihood of the continuous model on uniformly dequantized data cannot lead to the continuous model degenerately collapsing onto the discrete data, because its objective is bounded above by the log-likelihood of a discrete model.

3.2.1.2 Variational dequantization

While uniform dequantization successfully prevents the continuous density model p_{model} from collapsing to a degenerate mixture of point masses on discrete data, it asks p_{model} to assign uniform density to unit hypercubes $\mathbf{x} + [0, 1)^D$ around the data $\mathbf{x}$. It is difficult and unnatural for smooth function approximators, such as neural network density models, to excel at such a task. To sidestep this issue, we now introduce a new dequantization technique based on variational inference.

Again, we are interested in modeling D-dimensional discrete data $\mathbf{x} \sim P_{\text{data}}$ using a continuous density model p_{model}, and we will do so by maximizing the log-likelihood of its associated discrete model $P_{\text{model}}(\mathbf{x}) \coloneqq \int_{[0,1)^D} p_{\text{model}}(\mathbf{x} + \mathbf{u})\, d\mathbf{u}$. Now, however, we introduce a dequantization noise distribution $q(\mathbf{u}|\mathbf{x})$, with support over $\mathbf{u} \in [0, 1)^D$. Treating q as an

approximate posterior, we have the following variational lower bound, which holds for all q:

$$\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_{\text{model}}(\mathbf{x})\right] = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log \int_{[0,1)^D} q(\mathbf{u}|\mathbf{x}) \frac{p_{\text{model}}(\mathbf{x}+\mathbf{u})}{q(\mathbf{u}|\mathbf{x})}\, d\mathbf{u}\right] \tag{3.6}$$

$$\geq \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\int_{[0,1)^D} q(\mathbf{u}|\mathbf{x}) \log \frac{p_{\text{model}}(\mathbf{x}+\mathbf{u})}{q(\mathbf{u}|\mathbf{x})}\, d\mathbf{u}\right] \tag{3.7}$$

$$= \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\mathbb{E}_{\mathbf{u}\sim q(\cdot|\mathbf{x})}\left[\log \frac{p_{\text{model}}(\mathbf{x}+\mathbf{u})}{q(\mathbf{u}|\mathbf{x})}\right] \tag{3.8}$$

We will choose q itself to be a conditional flow-based generative model of the form $\mathbf{u} = q_{\mathbf{x}}(\boldsymbol{\epsilon})$, where $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$ is Gaussian noise. In this case, $q(\mathbf{u}|\mathbf{x}) = p(q_{\mathbf{x}}^{-1}(\mathbf{u})) \cdot |\partial q_{\mathbf{x}}^{-1}/\partial \mathbf{u}|$, and thus we obtain the objective

$$\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_{\text{model}}(\mathbf{x})\right] \geq \mathbb{E}_{\mathbf{x}\sim P_{\text{data}},\boldsymbol{\epsilon}}\left[\log \frac{p_{\text{model}}(\mathbf{x} + q_{\mathbf{x}}(\boldsymbol{\epsilon}))}{p(\boldsymbol{\epsilon})\left|\partial q_{\mathbf{x}}/\partial \boldsymbol{\epsilon}\right|^{-1}}\right] \tag{3.9}$$

which we maximize jointly over p_{model} and q. When p_{model} is also a flow model $\mathbf{x} = f^{-1}(\mathbf{z})$ (as it is throughout this paper), it is straightforward to calculate a stochastic gradient of this objective using the pathwise derivative estimator, as $f(\mathbf{x} + q_{\mathbf{x}}(\boldsymbol{\epsilon}))$ is differentiable with respect to the parameters of f and q.

Notice that the lower bound for uniform dequantization – Eqs. (3.3) to (3.5) – is a special case of our variational lower bound – Eqs. (3.6) to (3.8), when the dequantization distribution q is a uniform distribution that ignores dependence on $\mathbf{x}$. Because the gap between our objective (3.8) and the true expected log-likelihood $\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_{\text{model}}(\mathbf{x})\right]$ is exactly $\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}\left[D_{\text{KL}}\left(q(\mathbf{u}|\mathbf{x}) \parallel p_{\text{model}}(\mathbf{u}|\mathbf{x})\right)\right]$, using a uniform q forces p_{model} to unnaturally place uniform density over each hypercube $\mathbf{x} + [0,1)^D$ to compensate for any potential looseness in the variational bound introduced by the inexpressive q. Using an expressive flow-based q, on the other hand, allows p_{model} to place density in each hypercube $\mathbf{x} + [0,1)^D$ according to a much more flexible distribution $q(\mathbf{u}|\mathbf{x})$. This is a more natural task for p_{model} to perform, improving both training and generalization loss.

3.2.2 Improved coupling layers

Recent progress in the design of flow models has involved carefully constructing flows to increase their expressiveness while preserving tractability of the inverse and Jacobian determinant computations. One example is the invertible 1×1 convolution flow, whose inverse and Jacobian determinant can be calculated and differentiated with standard automatic differentiation libraries [52]. Another example, which we build upon in our work here, is the affine coupling layer [21]. It is a parameterized flow $\mathbf{y} = f_\theta(\mathbf{x})$ that first splits the components of $\mathbf{x}$ into two parts $\mathbf{x}_1, \mathbf{x}_2$, and then computes $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$, given by

$$\mathbf{y}_1 = \mathbf{x}_1, \qquad \mathbf{y}_2 = \mathbf{x}_2 \cdot \exp(\mathbf{a}_\theta(\mathbf{x}_1)) + \mathbf{b}_\theta(\mathbf{x}_1) \tag{3.10}$$

Here, $\mathbf{a}_\theta$ and $\mathbf{b}_\theta$ are outputs of a neural network that acts on $\mathbf{x}_1$ in a complex, expressive manner, but the resulting behavior on $\mathbf{x}_2$ always remains an elementwise affine transformation – effectively, $\mathbf{a}_\theta$ and $\mathbf{b}_\theta$ together form a data-parameterized family of invertible affine transformations. This allows the affine coupling layer to express complex dependencies on the data while keeping inversion and log-likelihood computation tractable. Using $\cdot$ and exp to respectively denote elementwise multiplication and exponentiation, the affine coupling layer is defined by:

$$\mathbf{x}_1 = \mathbf{y}_1, \tag{3.11}$$

$$\mathbf{x}_2 = (\mathbf{y}_2 - \mathbf{b}_\theta(\mathbf{y}_1)) \cdot \exp(-\mathbf{a}_\theta(\mathbf{y}_1)), \tag{3.12}$$

$$\log \left| \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right| = \mathbf{1}^\top \mathbf{a}_\theta(\mathbf{x}_1) \tag{3.13}$$

The splitting operation $\mathbf{x} \mapsto (\mathbf{x}_1, \mathbf{x}_2)$ and merging operation $(\mathbf{y}_1, \mathbf{y}_2) \mapsto \mathbf{y}$ are usually performed over channels or over space in a checkerboard-like pattern [21].

3.2.2.1 Expressive coupling transformations with continuous mixture CDFs

We found in our experiments that density modeling performance of these coupling layers could be improved by augmenting the data-parameterized elementwise affine transformations by more general nonlinear elementwise transformations. For a given scalar component x of $\mathbf{x}_2$, we apply the cumulative distribution function (CDF) for a mixture of K logistics – parameterized by mixture probabilities, means, and log scales $\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}$ – followed by an inverse sigmoid and an affine transformation parameterized by a and b:

$$x \longmapsto \sigma^{-1}\left(\text{MixLogCDF}(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s})\right) \cdot \exp(a) + b \tag{3.14}$$

where

$$\text{MixLogCDF}(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}) \coloneqq \sum_{i=1}^{K} \pi_i \sigma\left((x - \mu_i) \cdot \exp(-s_i)\right) \tag{3.15}$$

The transformation parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}, a, b$ for each component of $\mathbf{x}_2$ are produced by a neural network acting on $\mathbf{x}_1$. This neural network must produce these transformation parameters for each component of $\mathbf{x}_2$, hence it produces vectors $\mathbf{a}_\theta(\mathbf{x}_1)$ and $\mathbf{b}_\theta(\mathbf{x}_1)$ and tensors $\boldsymbol{\pi}_\theta(\mathbf{x}_1), \boldsymbol{\mu}_\theta(\mathbf{x}_1), \mathbf{s}_\theta(\mathbf{x}_1)$ (with last axis dimension K). The coupling transformation is then given by:

$$\mathbf{y}_1 = \mathbf{x}_1, \tag{3.16}$$

$$\mathbf{y}_2 = \sigma^{-1}\left(\text{MixLogCDF}(\mathbf{x}_2; \boldsymbol{\pi}_\theta(\mathbf{x}_1), \boldsymbol{\mu}_\theta(\mathbf{x}_1), \mathbf{s}_\theta(\mathbf{x}_1))\right) \cdot \exp(\mathbf{a}_\theta(\mathbf{x}_1)) + \mathbf{b}_\theta(\mathbf{x}_1) \tag{3.17}$$

where the formula for computing $\mathbf{y}_2$ operates elementwise.

The inverse sigmoid ensures that the inverse of this coupling transformation always exists: the range of the logistic mixture CDF is $(0, 1)$, so the domain of its inverse must stay

within this interval. The CDF itself can be inverted efficiently with bisection, because it is a monotonically increasing function. Moreover, the Jacobian determinant of this transformation involves calculating the probability density function of the logistic mixtures, which poses no computational difficulty.

3.2.2.2 Expressive conditioning architectures with self-attention

In addition to improving the expressiveness of the elementwise transformations on $\mathbf{x}_2$, we found it crucial to improve the expressiveness of the conditioning on $\mathbf{x}_1$ – that is, the expressiveness of the neural network responsible for producing the elementwise transformation parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \mathbf{s}, \mathbf{a}, \mathbf{b}$. Our best results were obtained by stacking convolutions and multi-head self attention into a gated residual network [68, 13], in a manner resembling the Transformer [109] with pointwise feedforward layers replaced by 3×3 convolutional layers. Our architecture is defined as a stack of blocks. Each block consists of the following two layers connected in a residual fashion, with layer normalization [2] after each residual connection:

$$\begin{aligned} \text{Conv} &= \text{Input} \to \text{Nonlinearity} \to \text{Conv}_{3\times3} \to \text{Nonlinearity} \to \text{Gate} \\ \text{Attn} &= \text{Input} \to \text{Conv}_{1\times1} \to \text{MultiHeadSelfAttention} \to \text{Gate} \end{aligned}$$

where Gate refers to a 1×1 convolution that doubles the number of channels, followed by a gated linear unit [17]. The convolutional layer is identical to the one used by PixelCNN++ [93], and the multi-head self attention mechanism we use is identical to the one in the Transformer [109]. (We always use 4 heads in our experiments, since we found it to be effective early on in our experimentation process.)

With these blocks in hand, the network that outputs the elementwise transformation parameters is simply given by stacking blocks on top of each other, and finishing with a final convolution that increases the number of channels to the amount needed to specify the elementwise transformation parameters.

3.2.3 Related Work

Likelihood-based models constitute a large family of deep generative models. One subclass of such methods, based on variational inference, allows for efficient approximate inference and sampling, but does not admit exact log likelihood computation [53, 86, 54]. Another subclass, which we called exact likelihood models in this work, does admit exact log likelihood computation. These exact likelihood models are typically specified as invertible transformations that are parameterized by neural networks [19, 58, 104, 20, 29, 107, 93, 13].

There is prior work that aims to improve the sampling speed of deep autoregressive models. The Multiscale PixelCNN [84] modifies the PixelCNN to be non-fully-expressive by introducing conditional independence assumptions among pixels in a way that permits sampling in a logarithmic number of steps, rather than linear. Such a change in the autoregressive structure allows for faster sampling but also makes some statistical patterns

impossible to capture, and hence reduces the capacity of the model for density estimation. WaveRNN [50] improves sampling speed for autoregressive models for audio via sparsity and other engineering considerations, some of which may apply to flow models as well.

There is also recent work that aims to improve the expressiveness of coupling layers in flow models. Kingma and Dhariwal [52] demonstrate improved density estimation using an invertible 1×1 convolution flow, and demonstrate that very large flow models can be trained to produce photorealistic faces. Huang et al. [44] show how to design elementwise transformations which themselves are neural networks. Müller et al. [70] introduce piecewise polynomial couplings that are similar in spirit to our mixture of logistics couplings and found them to be more expressive than affine couplings, but reported little performance gains in density estimation. We leave a detailed comparison between our coupling layer and these other types of coupling layers for future work.

3.3 Flow model experiments

Here, we show that Flow++ achieves state-of-the-art density modeling performance among non-autoregressive models on CIFAR10 and 32x32 and 64x64 ImageNet. We also present ablation experiments that quantify the improvements proposed in Section 3.2, and we present example generative samples from Flow++ and compare them against samples from autoregressive models.

Our experiments employed weight normalization and data-dependent initialization [92]. We used the checkerboard-splitting, channel-splitting, and downsampling flows of Dinh et al. [21]; we also used before every coupling flow an invertible 1x1 convolution flows of Kingma and Dhariwal [52], as well as a variant of their "actnorm" flow that normalizes all activations independently (instead of normalizing per channel). Our CIFAR10 model used 4 coupling layers with checkerboard splits at 32x32 resolution, 2 coupling layers with channel splits at 16x16 resolution, and 3 coupling layers with checkerboard splits at 16x16 resolution; each coupling layer used 10 convolution-attention blocks, all with 96 filters. More details on architectures, as well as details for the other experiments, are in our source code release [38].

3.3.1 Density modeling results

Table 3.1 shows that Flow++ achieved state-of-the-art density modeling results out of all non-autoregressive models at the time of publication. Moreover, Flow++ is competitive with autoregressive models: its performance is on par with the first generation of PixelCNN models [107], and in fact it outperforms Multiscale PixelCNN [84]. Our results are reported using 16384 importance samples in our CIFAR experiment and 1 sample in our ImageNet experiments [11]. With 1 sample only, our CIFAR model attains 3.12 bits/dim. Our listed ImageNet 32x32 and 64x64 results are evaluated on a NVIDIA DGX-1; they are worse by 0.01 bits/dim when evaluated on a NVIDIA Titan X GPU.

Table 3.1: Flow++ unconditional image modeling results (in bits per dimension)

Model family	Model	CIFAR10	ImageNet 32x32	ImageNet 64x64
Non-autoregressive	RealNVP [21]	3.49	4.28	–
	Glow [52]	3.35	4.09	3.81
	IAF-VAE [54]	3.11	–	–
	Flow++ (ours)	**3.08**	**3.86**	**3.69**
Autoregressive	Multiscale PixelCNN [84]	–	3.95	3.70
	PixelCNN [107]	3.14	–	–
	PixelRNN [107]	3.00	3.86	3.63
	Gated PixelCNN [108]	3.03	3.83	3.57
	PixelCNN++ [93]	2.92	–	–
	Image Transformer [76]	2.90	3.77	–
	PixelSNAIL [13]	2.85	3.80	3.52

Table 3.2: CIFAR10 ablation results after 400 epochs of training. Models not converged for the purposes of ablation study.

Ablation	bits/dim	parameters
uniform dequantization	3.292	32.3M
affine coupling	3.200	32.0M
no self-attention	3.193	31.4M
Flow++ (not converged for ablation)	**3.165**	31.4M

3.3.2 Ablations

We ran the following ablations of our model on unconditional CIFAR10 density estimation: variational dequantization vs. uniform dequantization; logistic mixture coupling vs. affine coupling; and stacked self-attention vs. convolutions only. As each ablation involves removing some component of the network, we increased the number of filters in all convolutional layers (and attention layers, if present) in order to match the total number of parameters with the full Flow++ model.

In Fig. 3.1 and Table 3.2, we compare the performance of these ablations relative to Flow++ at 400 epochs of training, which was not enough for these models to converge, but far enough to see their relative performance differences. Switching from our variational dequantization to the more standard uniform dequantization costs the most: approximately 0.127 bits/dim. The remaining two ablations both cost approximately 0.03 bits/dim: switching from our logistic mixture coupling layers to affine coupling layers, and switching from our hybrid

convolution-and-self-attention architecture to a pure convolutional residual architecture. Note that these performance differences are present despite all networks having approximately the same number of parameters: the improved performance of Flow++ comes from improved inductive biases, not simply from increased parameter count.

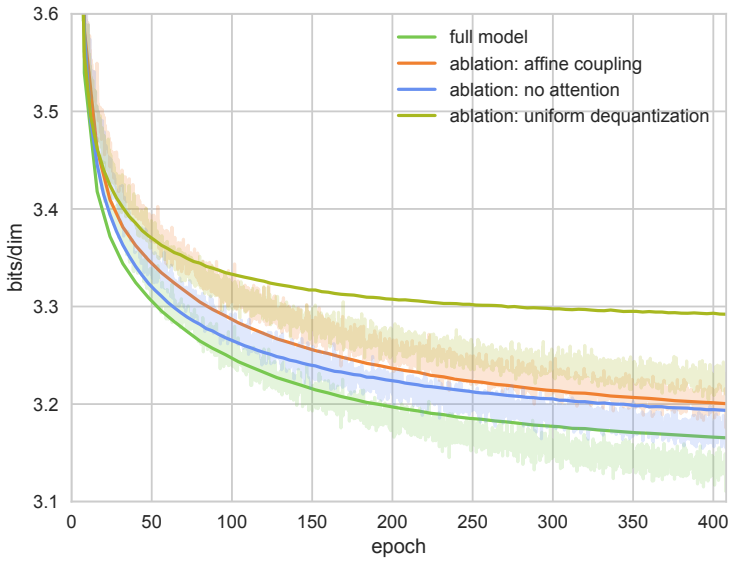
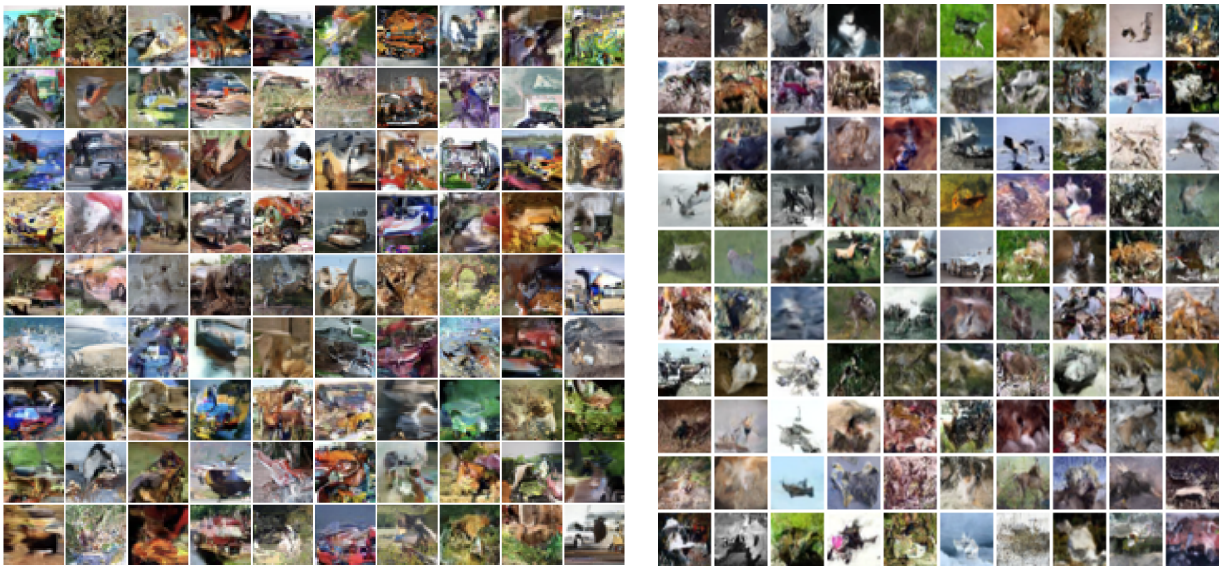


Figure 3.1: Ablation training (light) and validation (dark) curves on unconditional CIFAR10 density estimation. These runs are not fully converged, but the gap in performance is already visible.

The most interesting result is probably the effect of the dequantization scheme on training and generalization loss. At 400 epochs of training, the full Flow++ model with variational dequantization has a train-test gap of approximately 0.02 bits/dim, but with uniform dequantization, the train-test gap is approximately 0.06 bits/dim. This confirms our claim in Section 3.2.1.2 that training with variational dequantization is a more natural task for the model than training with uniform dequantization.

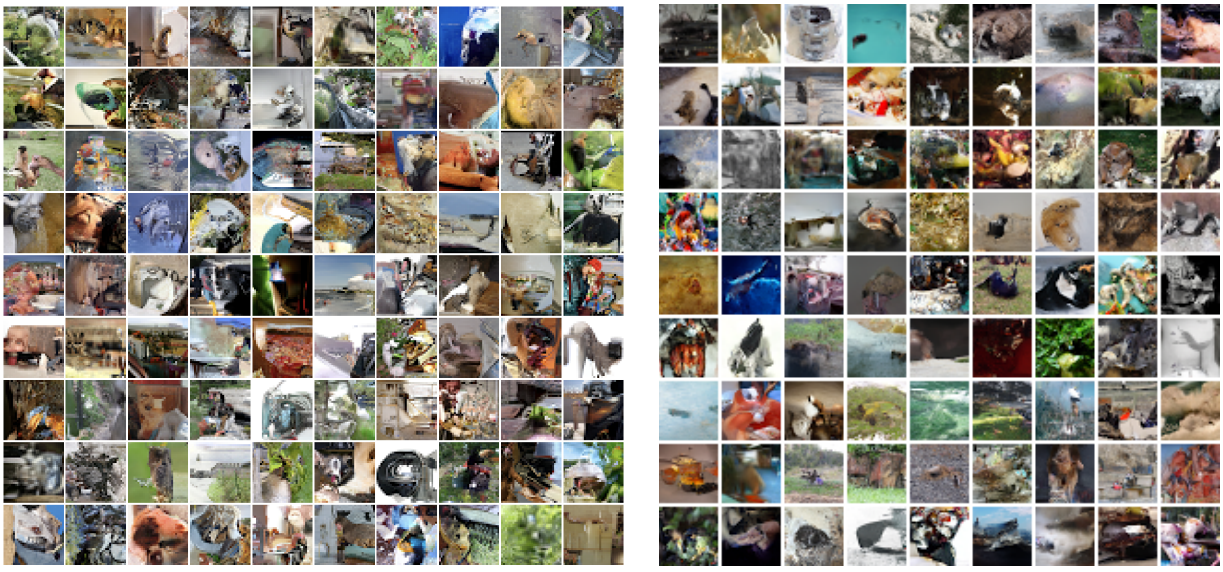
3.3.3 Samples

We present the samples from our trained density models of Flow++ on CIFAR10, 32x32 ImageNet, 64x64 ImageNet, and 5-bit CelebA in Figs. 3.2 to 3.5. The Flow++ samples match the perceptual quality of PixelCNN samples, showing that Flow++ captures both local and global dependencies as well as PixelCNN and is capable of generating diverse samples on large datasets. Moreover, sampling is fast: our CIFAR10 model takes approximately 0.32 seconds to generate a batch of 8 samples in parallel on one NVIDIA 1080 Ti GPU, making it more than an order of magnitude faster than PixelCNN++ with sampling speed optimizations [82]. More samples are available in the supplementary [38].



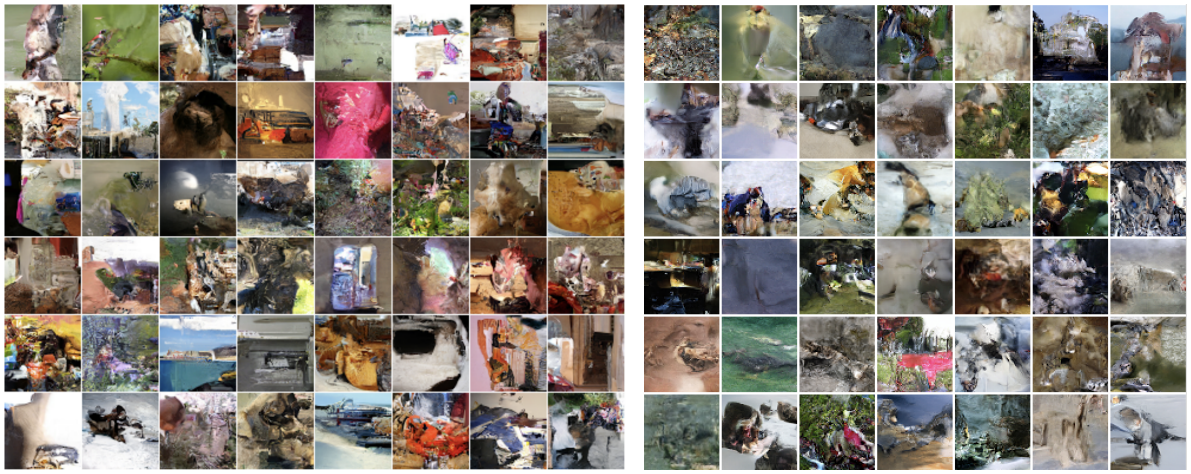
(a) PixelCNN [107] (b) Flow++

Figure 3.2: CIFAR10 Samples. Flow++ captures local dependencies well and generates good samples at the quality level of PixelCNN, but with the advantage of efficient sampling.



(a) PixelCNN [107] (b) Flow++

Figure 3.3: 32x32 ImageNet Samples. Flow++ samples match those of PixelCNN in diversity on this dataset, which is much larger than CIFAR10.



(a) Multi-Scale PixelRNN [107] (b) Flow++

Figure 3.4: 64x64 ImageNet Samples. The diversity of samples from Flow++ matches the diversity of samples from PixelRNN with multi-scale ordering.

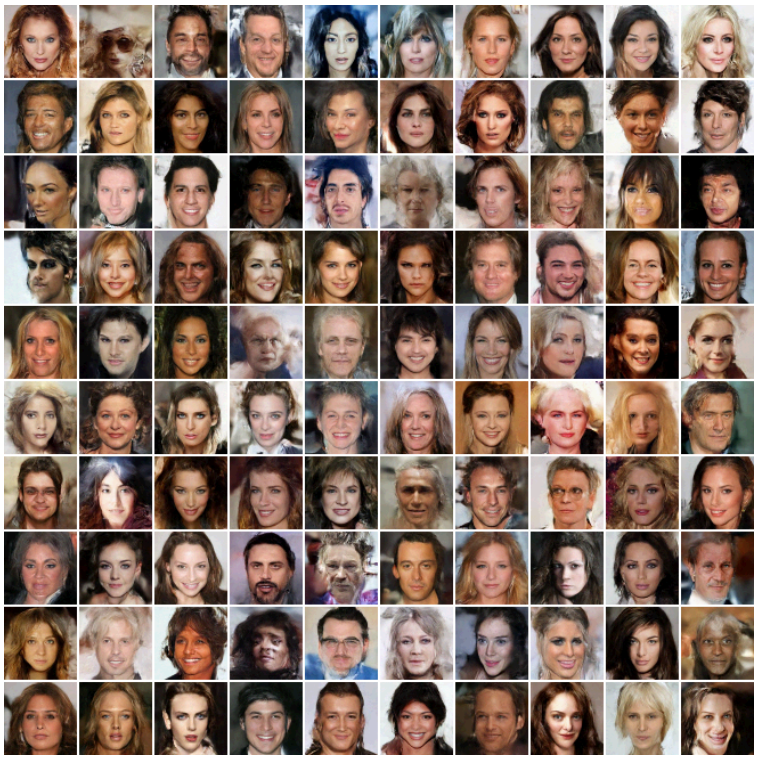


Figure 3.5: 5-bit 64x64 CelebA Flow++ samples, without low-temperature sampling.

3.4 Compression with flows

Having described Flow++, our improved RealNVP-type flow model, we now present our construction of its corresponding computationally efficient compression algorithm. In Section 3.5, we introduce *local bits-back coding*, our new technique for turning a general, pretrained, off-the-shelf flow model into an efficient coding algorithm suitable for continuous data discretized to high precision. Section 3.5.4.1 shows how to implement local bits-back coding without assumptions on the flow structure, leading to an algorithm that runs in polynomial time and space with respect to the data dimension.

Going further, Sections 3.5.4.2 to 3.5.4.3 show how to tailor local bits-back coding to various specific types of flows, culminating in a fully parallelizable algorithm for Flow++ that runs in linear time and space with respect to the data dimension and is fully parallelizable for both encoding and decoding. We then show in Section 3.5.5 how to adapt local bits-back coding to losslessly code data discretized to arbitrarily low precision, and in doing so, we obtain a compression interpretation of variational dequantization, the method we introduced in Section 3.2.1.2 to train flow models on discrete data. In Section 3.6, we conclude with experiments on CIFAR10 and downsampled ImageNet that verify that local bits-back coding efficiently attains codelengths close to the negative log likelihoods of flow models.

3.4.1 Preliminaries

Here, we remind the reader of basic notions on compression, and we re-introduce flow models and variational dequantization in notation suitable for our discussion on compression. All logarithms in the following sections are taken base 2.

3.4.1.1 Lossless compression

We begin by defining lossless compression of d-dimensional discrete data $\mathbf{x}^\circ$ using a probability mass function $p(\mathbf{x}^\circ)$ represented by a generative model. It means to construct a uniquely decodable code C, which is an injective map from data sequences to binary strings, whose lengths $|C(\mathbf{x}^\circ)|$ are close to $-\log p(\mathbf{x}^\circ)$ [16]. The rationale is that if the generative model is expressive and trained well, its cross entropy will be close to the entropy of the data distribution. So, if the lengths of C match the model's negative log probabilities, the expected length of C will be small, and hence C will be a good compression algorithm. Constructing such a code is always possible in theory because the Kraft-McMillan inequality [56, 65] ensures that there always exists some code with lengths $|C(\mathbf{x}^\circ)| = \lceil -\log p(\mathbf{x}^\circ) \rceil \approx -\log p(\mathbf{x}^\circ)$.

3.4.1.2 Flow models and variational dequantization

We wish to construct a computationally efficient code specialized to a flow model f, as in Section 3.2, which is a differentiable bijection between continuous data $\mathbf{x} \in \mathbb{R}^d$ and latents $\mathbf{z} = f(\mathbf{x}) \in \mathbb{R}^d$ [19, 20, 21]. A flow model comes with a density $p(\mathbf{z})$ on the latent space and

thus has an associated sampling process—$\mathbf{x} = f^{-1}(\mathbf{z})$ for $\mathbf{z} \sim p(\mathbf{z})$—under which it defines a probability density function via the change-of-variables formula for densities:

$$-\log p(\mathbf{x}) = -\log p(\mathbf{z}) - \log \left|\det \mathbf{J}(\mathbf{x})\right| \tag{3.18}$$

where $\mathbf{J}(\mathbf{x})$ denotes the Jacobian of f at $\mathbf{x}$. Flow models are straightforward to train with maximum likelihood, as Eq. (3.18) allows unbiased exact log likelihood gradients to be computed efficiently.

Standard datasets such as CIFAR10 and ImageNet consist of discrete data $\mathbf{x}^\circ \in \mathbb{Z}^d$. To make a flow model suitable for such discrete data, we follow Section 3.2.1.2 and define a derived discrete model $P(\mathbf{x}^\circ) \coloneqq \int_{[0,1)^d} p(\mathbf{x}^\circ + \mathbf{u})\, d\mathbf{u}$ to be trained by minimizing a variational dequantization objective, which is a variational bound on the codelength of $P(\mathbf{x}^\circ)$:

$$\mathbb{E}_{\mathbf{u}\sim q(\mathbf{u}|\mathbf{x}^\circ)}\left[-\log \frac{p(\mathbf{x}^\circ + \mathbf{u})}{q(\mathbf{u}|\mathbf{x}^\circ)}\right] \geq -\log \int_{[0,1)^d} p(\mathbf{x}^\circ + \mathbf{u})\, d\mathbf{u} = -\log P(\mathbf{x}^\circ) \tag{3.19}$$

Here, $q(\mathbf{u}|\mathbf{x}^\circ)$ proposes dequantization noise $\mathbf{u} \in [0,1)^d$ that transforms discrete data $\mathbf{x}^\circ$ into continuous data $\mathbf{x}^\circ + \mathbf{u}$; it can be fixed to either a uniform distribution [105, 99, 93] or to another parameterized flow to be trained jointly with f, as we did with Flow++ (Section 3.2.1.2). This variational dequantization objective serves as a theoretical codelength for flow models trained on discrete data, just like negative log probability mass serves as a theoretical codelength for discrete generative models [99].

3.5 Local bits-back coding

Our goal is to develop computationally efficient coding algorithms for flows trained on the variational dequantization objective Eq. (3.19). In Sections 3.5.1 to 3.5.4, we develop algorithms that use flows to code continuous data discretized to high precision. In Section 3.5.5, we adapt these algorithms to losslessly code data discretized to low precision, attaining our desired codelength (3.19) for discrete data.

3.5.1 Coding continuous data using discretization

We first address the problem of developing coding algorithms that attain codelengths given by negative log densities of flow models, such as Eq. (3.18). Probability density functions do not directly map to codelength, unlike probability mass functions which enjoy the result of the Kraft-McMillan inequality. So, following standard procedure [16, section 8.3], we discretize the data to a high precision k and code this discretized data with a certain probability mass function derived from the density model. Specifically, we tile $\mathbb{R}^d$ with hypercubes of volume $\delta_x \coloneqq 2^{-kd}$; we call each hypercube a bin. For $\mathbf{x} \in \mathbb{R}^d$, let $B(\mathbf{x})$ be the unique bin that contains $\mathbf{x}$, and let $\bar{\mathbf{x}}$ be the center of the bin $B(\mathbf{x})$. We call $\bar{\mathbf{x}}$ the discretized version of $\mathbf{x}$. For a sufficiently smooth probability density function $p(\mathbf{x})$, such as a density coming from

a neural network flow model, the probability mass function $P(\bar{\mathbf{x}}) \coloneqq \int_{B(\bar{\mathbf{x}})} p(\mathbf{x})\, d\mathbf{x}$ takes on the pleasingly simple form $P(\bar{\mathbf{x}}) \approx p(\bar{\mathbf{x}})\delta_x$ when the precision k is large. Now we invoke the Kraft-McMillan inequality, so the theoretical codelength for $\bar{\mathbf{x}}$ using P is

$$-\log P(\bar{\mathbf{x}}) \approx -\log p(\bar{\mathbf{x}})\delta_x \tag{3.20}$$

bits. This is the compression interpretation of the negative log density: it is a codelength for data discretized to high precision, when added to the total number of bits of discretization precision. It is this codelength, Eq. (3.20), that we will try to achieve with an efficient algorithm for flow models. We defer the problem of coding data discretized to low precision to Section 3.5.5.

3.5.2 Background on bits-back coding

The main tool we will employ is bits-back coding [110, 35, 26, 41], a coding technique originally designed for latent variable models (the connection to flow models is presented in Section 3.5.3 and is new to our work). Bits-back coding codes $\mathbf{x}$ using a distribution of the form $p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$, where $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$ includes a latent variable $\mathbf{z}$; it is relevant when $\mathbf{z}$ ranges over an exponentially large set, which makes it intractable to code with $p(\mathbf{x})$ even though coding with $p(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$ may be tractable individually. Bits-back coding introduces a new distribution $q(\mathbf{z}|\mathbf{x})$ with tractable coding, and the encoder jointly encodes $\mathbf{x}$ along with $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ via these steps:

1. Decode $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$ from an auxiliary source of random bits
2. Encode $\mathbf{x}$ using $p(\mathbf{x}|\mathbf{z})$
3. Encode $\mathbf{z}$ using $p(\mathbf{z})$

The first step, which decodes $\mathbf{z}$ from random bits, produces a sample $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$. The second and third steps transmit $\mathbf{z}$ along with $\mathbf{x}$. At decoding time, the decoder recovers $(\mathbf{x}, \mathbf{z})$, then recovers the bits the encoder used to sample $\mathbf{z}$ using q. So, the encoder will have transmitted extra information in addition to $\mathbf{x}$—precisely $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[-\log q(\mathbf{z}|\mathbf{x})\right]$ bits on average. Consequently, the net number of bits transmitted regarding $\mathbf{x}$ only will be $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}, \mathbf{z})\right]$, which is redundant compared to the desired length $-\log p(\mathbf{x})$ by an amount equal to the KL divergence $D_{\mathrm{KL}}\left(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{x})\right)$ from q to the true posterior.

Bits-back coding also works with continuous $\mathbf{z}$ discretized to high precision with negligible change in codelength [35, 102]. In this case, $q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ are probability density functions. Discretizing $\mathbf{z}$ to bins $\bar{\mathbf{z}}$ of small volume δ_z and defining the probability mass functions $Q(\bar{\mathbf{z}}|\mathbf{x})$ and $P(\bar{\mathbf{z}})$ by the method in Section 3.5.1, we see that the bits-back codelength remains

approximately unchanged:

$$\mathbb{E}_{\bar{\mathbf{z}}\sim Q(\bar{\mathbf{z}}|\mathbf{x})}\left[-\log\frac{p(\mathbf{x}|\bar{\mathbf{z}})P(\bar{\mathbf{z}})}{Q(\bar{\mathbf{z}}|\mathbf{x})}\right] \approx \mathbb{E}_{\bar{\mathbf{z}}\sim Q(\bar{\mathbf{z}}|\mathbf{x})}\left[-\log\frac{p(\mathbf{x}|\bar{\mathbf{z}})p(\bar{\mathbf{z}})\delta_z}{q(\bar{\mathbf{z}}|\mathbf{x})\delta_z}\right] \tag{3.21}$$

$$\approx \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}\left[-\log\frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\right] \tag{3.22}$$

When bits-back coding is applied to a particular latent variable model, such as a VAE, the distributions involved may take on a certain meaning: $p(\mathbf{z})$ would be the prior, $p(\mathbf{x}|\mathbf{z})$ would be the decoder network, and $q(\mathbf{z}|\mathbf{x})$ would be the encoder network [53, 86, 18, 12, 26, 102, 55]. However, it is important to note that these distributions do not need to correspond explicitly to parts of the model at hand. Any will do for coding data losslessly, though some choices result in better codelengths. We exploit this fact in Section 3.5.3, where we apply bits-back coding to flow models by constructing artificial distributions $p(\mathbf{x}|\mathbf{z})$ and $q(\mathbf{z}|\mathbf{x})$, which do not come with a flow model by default.

3.5.3 Local bits-back coding

We now present *local bits-back coding*, our new high-level principle for using a flow model f to code data discretized to high precision. Following Section 3.5.1, we discretize continuous data $\mathbf{x}$ into $\bar{\mathbf{x}}$, which is the center of a bin of volume δ_x. The codelength we desire for $\bar{\mathbf{x}}$ is the negative log density of f (3.18), plus a constant depending on the discretization precision:

$$-\log p(\bar{\mathbf{x}})\delta_x = -\log p(f(\bar{\mathbf{x}})) - \log\left|\det \mathbf{J}(\bar{\mathbf{x}})\right| - \log\delta_x \tag{3.23}$$

where $\mathbf{J}(\bar{\mathbf{x}})$ is the Jacobian of f at $\bar{\mathbf{x}}$. We will construct two densities $\tilde{p}(\mathbf{z}|\mathbf{x})$ and $\tilde{p}(\mathbf{x}|\mathbf{z})$ such that bits-back coding attains Eq. (3.23). We need a small scalar parameter $\sigma > 0$, with which we define

$$\tilde{p}(\mathbf{z}|\mathbf{x}) \coloneqq \mathcal{N}(\mathbf{z}; f(\mathbf{x}), \sigma^2\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^\top) \qquad \text{and} \qquad \tilde{p}(\mathbf{x}|\mathbf{z}) \coloneqq \mathcal{N}(\mathbf{x}; f^{-1}(\mathbf{z}), \sigma^2\mathbf{I}) \tag{3.24}$$

To encode $\bar{\mathbf{x}}$, local bits-back coding follows the bits-back coding method for continuous $\mathbf{z}$, as described in Section 3.5.2:

1. Decode $\bar{\mathbf{z}} \sim \tilde{P}(\bar{\mathbf{z}}|\mathbf{x}) = \int_{B(\bar{\mathbf{z}})} \tilde{p}(\mathbf{z}|\mathbf{x})\, d\mathbf{z} \approx \tilde{p}(\bar{\mathbf{z}}|\mathbf{x})\delta_z$ from an auxiliary source of random bits
2. Encode $\bar{\mathbf{x}}$ using $\tilde{P}(\bar{\mathbf{x}}|\bar{\mathbf{z}}) = \int_{B(\bar{\mathbf{x}})} \tilde{p}(\mathbf{x}|\bar{\mathbf{z}})\, d\mathbf{z} \approx \tilde{p}(\bar{\mathbf{x}}|\bar{\mathbf{z}})\delta_x$
3. Encode $\bar{\mathbf{z}}$ using $P(\bar{\mathbf{z}}) = \int_{B(\bar{\mathbf{z}})} p(\mathbf{z})\, d\mathbf{z} \approx p(\bar{\mathbf{z}})\delta_z$

The conditional density $\tilde{p}(\mathbf{z}|\mathbf{x})$ (3.24) is artificially injected noise, scaled by σ (the flow model f remains unmodified). It describes how a local linear approximation of f would behave if it were to act on a small Gaussian around $\bar{\mathbf{x}}$.

To justify local bits-back coding, we simply calculate its expected codelength. First, our choices of $\tilde{p}(\mathbf{z}|\mathbf{x})$ and $\tilde{p}(\mathbf{x}|\mathbf{z})$ (3.24) satisfy the following equation:

$$\mathbb{E}_{\mathbf{z}\sim\tilde{p}(\mathbf{z}|\mathbf{x})}\left[\log\tilde{p}(\mathbf{z}|\mathbf{x}) - \log\tilde{p}(\mathbf{x}|\mathbf{z})\right] = -\log|\det\mathbf{J}(\mathbf{x})| + O(\sigma^2) \tag{3.25}$$

Next, just like standard bits-back coding (3.22), local bits-back coding attains an expected codelength close to $\mathbb{E}_{\mathbf{z}\sim\tilde{p}(\mathbf{z}|\bar{\mathbf{x}})}L(\bar{\mathbf{x}},\mathbf{z})$, where

$$L(\mathbf{x},\mathbf{z}) \coloneqq \log\tilde{p}(\mathbf{z}|\mathbf{x})\delta_z - \log\tilde{p}(\mathbf{x}|\mathbf{z})\delta_x - \log p(\mathbf{z})\delta_z \tag{3.26}$$

Equations (3.24) to (3.26) imply that the expected codelength matches our desired codelength (3.23), up to first order in σ:

$$\mathbb{E}_{\mathbf{z}}L(\mathbf{x},\mathbf{z}) = -\log p(\mathbf{x})\delta_x + O(\sigma^2) \tag{3.27}$$

Note that local bits-back coding exactly achieves the desired codelength for flows (3.23), up to first order in σ. This is in stark contrast to bits-back coding with latent variable models like VAEs, for which the bits-back codelength is the negative evidence lower bound, which is redundant by an amount equal to the KL divergence from the approximate posterior to the true posterior [53].

Equation (3.27) may be derived in a more complete manner via the following argument (sufficient conditions are that the prior log density and the inverse of the flow have bounded derivatives of all orders). Let $\mathbf{y} = f(\mathbf{x})$ and let $\mathbf{J}$ be the Jacobian of f at $\mathbf{x}$. If we write $\mathbf{z} = \mathbf{y} + \sigma\mathbf{J}\boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})$, the local bits-back codelength satisfies:

$$\begin{aligned}\mathbb{E}_{\mathbf{z}}L(\mathbf{x},\mathbf{z}) + \log\delta_x &= \mathbb{E}_{\boldsymbol{\epsilon}}L(\mathbf{x},\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon}) + \log\delta_x \\ &= \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\log\mathcal{N}(\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon};\mathbf{y},\sigma^2\mathbf{J}\mathbf{J}^\top)}_{\text{(a)}} - \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\log\mathcal{N}(\mathbf{x};f^{-1}(\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon}),\sigma^2\mathbf{I})}_{\text{(b)}} - \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\log p(\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon})}_{\text{(c)}}\end{aligned} \tag{3.28}$$

We proceed by calculating each term. The first term (a) is the negative differential entropy of a Gaussian with covariance matrix $\sigma^2\mathbf{J}\mathbf{J}^\top$:

$$\mathbb{E}_{\boldsymbol{\epsilon}}\log\mathcal{N}(\sigma\mathbf{J}\boldsymbol{\epsilon};\mathbf{0},\sigma^2\mathbf{J}\mathbf{J}^\top) = -\frac{d}{2}\log(2\pi e\sigma^2) - \log|\det\mathbf{J}| \tag{3.29}$$

We calculate the second term (b) by taking a Taylor expansion of f^{-1} around $\mathbf{y}$. Let f_i^{-1} denote the i^{th} coordinate of f^{-1}. The inverse function theorem yields

$$f_i^{-1}(\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon}) = f_i^{-1}(\mathbf{y}) + \nabla f_i^{-1}(\mathbf{y})^\top(\sigma\mathbf{J}\boldsymbol{\epsilon}) + \frac{1}{2}(\sigma\mathbf{J}\boldsymbol{\epsilon})^\top\nabla^2 f_i^{-1}(\mathbf{y})(\sigma\mathbf{J}\boldsymbol{\epsilon}) + O(\sigma^3) \tag{3.30}$$

$$= x_i + \sigma\epsilon_i + \frac{\sigma^2}{2}\boldsymbol{\epsilon}^\top\mathbf{M}_i\boldsymbol{\epsilon} + O(\sigma^3) \tag{3.31}$$

where $\mathbf{M}_i \coloneqq \mathbf{J}^\top\nabla^2 f_i^{-1}(\mathbf{y})\mathbf{J}$. Write $\mathbf{v}_{\boldsymbol{\epsilon}} \coloneqq [\boldsymbol{\epsilon}^\top\mathbf{M}_1\boldsymbol{\epsilon} \quad \cdots \quad \boldsymbol{\epsilon}^\top\mathbf{M}_d\boldsymbol{\epsilon}]^\top$, so that the previous equation can be written in vector form as $f^{-1}(\mathbf{y}+\sigma\mathbf{J}\boldsymbol{\epsilon}) = \mathbf{x} + \sigma\boldsymbol{\epsilon} + \frac{\sigma^2}{2}\mathbf{v}_{\boldsymbol{\epsilon}} + O(\sigma^3)$. With this

in hand, term (b) reduces to:

$$-\mathbb{E}_{\boldsymbol{\epsilon}} \log \mathcal{N}(\mathbf{x}; f^{-1}(\mathbf{y} + \sigma \mathbf{J}\boldsymbol{\epsilon}), \sigma^2 \mathbf{I}) \tag{3.32}$$

$$= -\mathbb{E}_{\boldsymbol{\epsilon}} \log \mathcal{N}\left(\mathbf{x}; \mathbf{x} + \sigma\boldsymbol{\epsilon} + \frac{\sigma^2}{2}\mathbf{v}_{\boldsymbol{\epsilon}} + O(\sigma^3), \sigma^2 \mathbf{I}\right) \tag{3.33}$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\frac{d}{2}\log(2\pi\sigma^2) + \frac{\log e}{2\sigma^2}\left(\|\sigma\boldsymbol{\epsilon}\|^2 + \sigma^3 \boldsymbol{\epsilon}^\top \mathbf{v}_{\boldsymbol{\epsilon}} + O(\sigma^4)\right)\right] \tag{3.34}$$

$$= \frac{d}{2}\log(2\pi e\sigma^2) + \frac{\sigma \log e}{2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}^\top \mathbf{v}_{\boldsymbol{\epsilon}}\right] + O(\sigma^2) \tag{3.35}$$

Because the coordinates of $\boldsymbol{\epsilon}$ are independent and have zero third moment, we have

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}^\top \mathbf{v}_{\boldsymbol{\epsilon}}\right] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_i \epsilon_i \boldsymbol{\epsilon}^\top \mathbf{M}_i \boldsymbol{\epsilon}\right] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{i,j,k} (\mathbf{M}_i)_{jk} \epsilon_i \epsilon_j \epsilon_k\right] = \sum_{i,j,k} (\mathbf{M}_i)_{jk} \mathbb{E}_{\boldsymbol{\epsilon}}\left[\epsilon_i \epsilon_j \epsilon_k\right] = 0 \tag{3.36}$$

which implies that

$$-\mathbb{E}_{\boldsymbol{\epsilon}} \log \mathcal{N}(\mathbf{x}; f^{-1}(\mathbf{y} + \sigma \mathbf{J}\boldsymbol{\epsilon}), \sigma^2 \mathbf{I}) = \frac{d}{2}\log(2\pi e\sigma^2) + O(\sigma^2) \tag{3.37}$$

The final term (c) is given by

$$-\mathbb{E}_{\boldsymbol{\epsilon}} \log p(\mathbf{y} + \sigma \mathbf{J}\boldsymbol{\epsilon}) = -\mathbb{E}_{\boldsymbol{\epsilon}}\left[\log p(\mathbf{y}) + \nabla \log p(\mathbf{y})^\top (\sigma \mathbf{J}\boldsymbol{\epsilon}) + O(\sigma^2)\right] \tag{3.38}$$

$$= -\log p(\mathbf{y}) - (\nabla \log p(\mathbf{y})^\top \sigma \mathbf{J}) \mathbb{E}_{\boldsymbol{\epsilon}} \boldsymbol{\epsilon} + O(\sigma^2) \tag{3.39}$$

$$= -\log p(\mathbf{y}) + O(\sigma^2) \tag{3.40}$$

Altogether, summing Eqs. (3.29), (3.37) and (3.40) yields the total codelength

$$\mathbb{E}_{\mathbf{z}} L(\mathbf{x}, \mathbf{z}) = -\log p(\mathbf{y}) - \log |\det \mathbf{J}| - \log \delta_x + O(\sigma^2) \tag{3.41}$$

which, to first order, does not depend on σ, and matches Eq. (3.23).

Local bits-back coding always codes $\bar{\mathbf{x}}$ losslessly, no matter the setting of σ, δ_x, and δ_z. However, σ must be small for the $O(\sigma^2)$ inaccuracy in Eq. (3.27) to be negligible. But for σ to be small, the discretization volumes δ_z and δ_x must be small too, otherwise the discretized Gaussians $\tilde{p}(\bar{\mathbf{z}}|\mathbf{x})\delta_z$ and $\tilde{p}(\bar{\mathbf{x}}|\mathbf{z})\delta_x$ will be poor approximations of the original Gaussians $\tilde{p}(\mathbf{z}|\mathbf{x})$ and $\tilde{p}(\mathbf{x}|\mathbf{z})$. So, because δ_x must be small, the data $\mathbf{x}$ must be discretized to high precision. And, because δ_z must be small, a relatively large number of auxiliary bits must be available to decode $\bar{\mathbf{z}} \sim \tilde{p}(\bar{\mathbf{z}}|\mathbf{x})\delta_z$. We will resolve the high precision requirement for the data with another application of bits-back coding in Section 3.5.5, and we will explore the impact of varying σ, δ_x, and δ_z on real-world data in experiments in Section 3.6.

3.5.4 Concrete local bits-back coding algorithms

We have shown that local bits-back coding attains the desired codelength (3.23) for data discretized to high precision. Now, we instantiate local bits-back coding with concrete algorithms.

3.5.4.1 Black box flows

Algorithm 3.1 is the most straightforward implementation of local bits-back coding. It directly implements the steps in Section 3.5.3 by invoking an external procedure, such as automatic differentiation, to explicitly compute the Jacobian of the flow. It therefore makes no assumptions on the structure of the flow, and hence we call it the *black box* algorithm.

Algorithm 3.1 Local bits-back coding: for black box flows

Require: flow f, discretization volumes δ_x, δ_z, noise level σ

procedure ENCODE($\bar{\mathbf{x}}$)
  $\mathbf{J} \leftarrow \mathbf{J}_f(\bar{\mathbf{x}})$ ▷ Compute the Jacobian of f at $\bar{\mathbf{x}}$
  Decode $\bar{\mathbf{z}} \sim \mathcal{N}(f(\bar{\mathbf{x}}), \sigma^2 \mathbf{J}\mathbf{J}^\top)\,\delta_z$ ▷ By converting to an AR model (Section 3.5.4.1)
  Encode $\bar{\mathbf{x}}$ using $\mathcal{N}(f^{-1}(\bar{\mathbf{z}}), \sigma^2 \mathbf{I})\,\delta_x$
  Encode $\bar{\mathbf{z}}$ using $p(\bar{\mathbf{z}})\,\delta_z$
end procedure

procedure DECODE( )
  Decode $\bar{\mathbf{z}} \sim p(\bar{\mathbf{z}})\,\delta_z$
  Decode $\bar{\mathbf{x}} \sim \mathcal{N}(f^{-1}(\bar{\mathbf{z}}), \sigma^2 \mathbf{I})\,\delta_x$
  $\mathbf{J} \leftarrow \mathbf{J}_f(\bar{\mathbf{x}})$ ▷ Compute the Jacobian of f at $\bar{\mathbf{x}}$
  Encode $\bar{\mathbf{z}}$ using $\mathcal{N}(f(\bar{\mathbf{x}}), \sigma^2 \mathbf{J}\mathbf{J}^\top)\,\delta_z$ ▷ By converting to an AR model (Section 3.5.4.1)
  return $\bar{\mathbf{x}}$
end procedure

Coding with $\tilde{p}(\mathbf{x}|\mathbf{z})$ (3.24) is efficient because its coordinates are independent [102]. The same applies to the prior $p(\mathbf{z})$ if its coordinates are independent or if another efficient coding algorithm already exists for it (see Section 3.5.4.3). Coding efficiently with $\tilde{p}(\mathbf{z}|\mathbf{x})$ relies on the fact that any multivariate Gaussian can be converted into a linear autoregressive model, which can be coded efficiently, one coordinate at a time, using arithmetic coding or asymmetric numeral systems. To see how, suppose $\mathbf{y} = \mathbf{J}\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{J}$ is a full-rank matrix (such as a Jacobian of a flow model). Let $\mathbf{L}$ be the Cholesky decomposition of $\mathbf{J}\mathbf{J}^\top$. Since $\mathbf{L}\mathbf{L}^\top = \mathbf{J}\mathbf{J}^\top$, the distribution of $\mathbf{L}\boldsymbol{\epsilon}$ is equal to the distribution of $\mathbf{J}\boldsymbol{\epsilon} = \mathbf{y}$, and so solutions $\tilde{\mathbf{y}}$ to the linear system $\mathbf{L}^{-1}\tilde{\mathbf{y}} = \boldsymbol{\epsilon}$ have the same distribution as $\mathbf{y}$. Because $\mathbf{L}$ is triangular, $\mathbf{L}^{-1}$ is easily computable and also triangular, and thus $\tilde{\mathbf{y}}$ can be determined with back substitution: $\tilde{y}_i = (\epsilon_i - \sum_{j<i} (L^{-1})_{ij} \tilde{y}_j)/(L^{-1})_{ii}$, where i increases from 1 to d. In other words, $p(\tilde{y}_i|\tilde{\mathbf{y}}_{<i}) \coloneqq \mathcal{N}(\tilde{y}_i; -L_{ii} \sum_{j<i} (L^{-1})_{ij} \tilde{y}_j, L_{ii}^2)$ is a linear autoregressive model that represents the same distribution as $\mathbf{y} = \mathbf{J}\boldsymbol{\epsilon}$.

If nothing is known about the structure of the Jacobian of the flow, Algorithm 3.1 requires $O(d^2)$ space to store the Jacobian and $O(d^3)$ time to compute the Cholesky decomposition. This is certainly an improvement on the exponential space and time required by naive algorithms (Section 3.4), but it is still not efficient enough for high-dimensional data in practice. To make our coding algorithms more efficient, we need to make additional assumptions on the

flow. If the Jacobian is always block diagonal, say with fixed block size $c \times c$, then the steps in Algorithm 3.1 can be modified to process each block separately in parallel, thereby reducing the required space and time to $O(cd)$ and $O(c^2 d)$, respectively. This makes Algorithm 3.1 efficient for flows that operate as elementwise transformations or as convolutions, such as activation normalization flows and invertible 1×1 convolution flows [52].

3.5.4.2 Autoregressive flows

An autoregressive flow $\mathbf{z} = f(\mathbf{x})$ is a sequence of one-dimensional flows $z_i = f_i(x_i; \mathbf{x}_{<i})$ for each coordinate $i \in \{1, \ldots, d\}$ [74, 54]. Algorithm 3.2 shows how to code with an autoregressive flow in linear time and space. It never explicitly calculates and stores the Jacobian of the flow, unlike Algorithm 3.1. Rather, it invokes one-dimensional local bits-back coding on one coordinate of the data at a time, thus exploiting the structure of the autoregressive flow in an essential way.

Algorithm 3.2 Local bits-back coding: for autoregressive flows

Require: autoregressive flow f, discretization volumes δ_x, δ_z, noise level σ

procedure ENCODE($\bar{\mathbf{x}}$) ▷ Neural net operations parallelizable over i
  for $i = d, \ldots, 1$ **do** ▷ Iteration ordering not mandatory, but convenient for ANS
    Decode $\bar{z}_i \sim \mathcal{N}(f_i(\bar{x}_i; \bar{\mathbf{x}}_{<i}), (\sigma f_i'(\bar{x}_i; \bar{\mathbf{x}}_{<i}))^2)\, \delta_z^{1/d}$
    Encode $\bar{x}_i$ using $\mathcal{N}(f_i^{-1}(\bar{z}_i; \bar{\mathbf{x}}_{<i}), \sigma^2)\, \delta_x^{1/d}$
  end for
  Encode $\bar{\mathbf{z}}$ using $p(\bar{\mathbf{z}})\, \delta_z$
end procedure

procedure DECODE( )
  Decode $\bar{\mathbf{z}} \sim p(\bar{\mathbf{z}})\, \delta_z$
  for $i = 1, \ldots, d$ **do** ▷ Order should be the opposite of encoding when using ANS
    Decode $\bar{x}_i \sim \mathcal{N}(f_i^{-1}(\bar{z}_i; \bar{\mathbf{x}}_{<i}), \sigma^2)\, \delta_x^{1/d}$
    Encode $\bar{z}_i$ using $\mathcal{N}(f_i(\bar{x}_i; \bar{\mathbf{x}}_{<i}), (\sigma f_i'(\bar{x}_i; \bar{\mathbf{x}}_{<i}))^2)\, \delta_z^{1/d}$
  end for
  return $\bar{\mathbf{x}}$
end procedure

A key difference between Algorithm 3.1 and Algorithm 3.2 is that the former needs to run the forward and inverse directions of the entire flow and compute and factorize a Jacobian, whereas the latter only needs to do so for each one-dimensional flow on each coordinate of the data. Consequently, Algorithm 3.2 runs $O(d)$ time and space, excluding resource requirements of the flow itself. The encoding procedure of Algorithm 3.2 resembles log likelihood computation for autoregressive flows, so the model evaluations it requires are completely parallelizable over data dimensions. The decoding procedure resembles sampling,

so it requires d model evaluations in serial. These tradeoffs are entirely analogous to those of coding with discrete autoregressive models.

Autoregressive flows with further special structure lead to even more efficient implementations of Algorithm 3.2. As an example, let us focus on a NICE/RealNVP coupling layer [20, 21]. This type of flow computes $\mathbf{z}$ by splitting the coordinates of the input $\mathbf{x}$ into two halves, $\mathbf{x}_{\leq d/2}$, and $\mathbf{x}_{>d/2}$. The first half is passed through unchanged as $\mathbf{z}_{\leq d/2} = \mathbf{x}_{\leq d/2}$, and the second half is passed through an elementwise transformation $\mathbf{z}_{>d/2} = f(\mathbf{x}_{>d/2}; \mathbf{x}_{\leq d/2})$ which is conditioned on the first half. Specializing Algorithm 3.2 to this kind of flow produces Algorithm 3.3, which has both encoding and decoding parallelized over coordinates, resembling how the forward and inverse directions for inference and sampling can be parallelized for these flows [20, 21].

Algorithm 3.3 Local bits-back coding: for coupling layers

Require: coupling layer f, discretization volumes δ_x, δ_z, noise level σ

f given by $\mathbf{z}_{\leq d/2} = \mathbf{x}_{\leq d/2}, \mathbf{z}_{>d/2} = f(\mathbf{x}_{>d/2}; \mathbf{x}_{\leq d/2})$, where $f(\,\cdot\,; \mathbf{x}_{\leq d/2})$ operates elementwise
procedure ENCODE($\bar{\mathbf{x}}$)
    for $i = d, \ldots, d/2 + 1$ **do** ▷ Neural net operations parallelizable over i
        Decode $\bar{z}_i \sim \mathcal{N}(f_i(\bar{x}_i; \bar{\mathbf{x}}_{\leq d/2}), (\sigma f_i'(\bar{x}_i; \bar{\mathbf{x}}_{\leq d/2}))^2)\, \delta_z^{1/d}$
        Encode $\bar{x}_i$ using $\mathcal{N}(f_i^{-1}(\bar{z}_i; \bar{\mathbf{x}}_{\leq d/2}), \sigma^2)\, \delta_x^{1/d}$
    end for
    for $i = d/2, \ldots, 1$ **do**
        $\bar{z}_i \leftarrow \bar{x}_i$
    end for
    Encode $\bar{\mathbf{z}}$ using $p(\bar{\mathbf{z}})\, \delta_z$
end procedure

procedure DECODE( )
    Decode $\bar{\mathbf{z}} \sim p(\bar{\mathbf{z}})\, \delta_z$
    for $i = 1, \ldots, d/2$ **do**
        $\bar{x}_i \leftarrow \bar{z}_i$
    end for
    for $i = d/2 + 1, \ldots, d$ **do** ▷ Neural net operations parallelizable over i
        Decode $\bar{x}_i \sim \mathcal{N}(f_i^{-1}(\bar{z}_i; \bar{\mathbf{x}}_{\leq d/2}), \sigma^2)\, \delta_x^{1/d}$
        Encode $\bar{z}_i$ using $\mathcal{N}(f_i(\bar{x}_i; \bar{\mathbf{x}}_{\leq d/2}), (\sigma f_i'(\bar{x}_i; \bar{\mathbf{x}}_{\leq d/2}))^2)\, \delta_z^{1/d}$
    end for
    return $\bar{\mathbf{x}}$
end procedure

Algorithms 3.2 and 3.3 are not the only known efficient coding algorithms for autoregressive flows. For example, if f is an autoregressive flow whose prior $p(\mathbf{z}) = \prod_i p_i(z_i)$ is independent over coordinates, then f can be rewritten as a continuous autoregressive model $p(x_i|\mathbf{x}_{<i}) =$

$p_i(f(x_i; \mathbf{x}_{<i}))|f'(x_i; \mathbf{x}_{<i})|$, which can be discretized and coded one coordinate at a time using arithmetic coding or asymmetric numeral systems. The advantage of Algorithms 3.2 and 3.3, as we will see next, is that they apply to more complex priors that prevent the distribution over $\mathbf{x}$ from naturally factorizing as an autoregressive model.

3.5.4.3 Compositions of flows

Flows like NICE, RealNVP, Glow, and Flow++ [20, 21, 52, 38] are composed of many intermediate flows: they have the form $f(\mathbf{x}) = f_K \circ \cdots \circ f_1(\mathbf{x})$, where each of the K layers f_i is one of the types of flows discussed above. These models derive their expressiveness from applying simple flows many times, resulting in a expressive composite flow. The expressiveness of the composite flow suggests that coding will be difficult, but we can exploit the compositional structure to code efficiently. Since the composite flow $f = f_K \circ \cdots \circ f_1$ can be interpreted as a single flow $\mathbf{z}_1 = f_1(\mathbf{x})$ with a flow prior $f_K \circ \cdots \circ f_2(\mathbf{z}_1)$, all we have to do is code the first layer f_1 using the appropriate local bits-back coding algorithm, and when coding its output $\mathbf{z}_1$, we recursively invoke local bits-back coding for the prior $f_K \circ \cdots \circ f_2$ [55]. A straightforward inductive argument shows that this leads to the correct codelength. If coding any $\mathbf{z}_1$ with $f_K \circ \cdots \circ f_2$ achieves the expected codelength $-\log p_{f_K \circ \cdots \circ f_2}(\mathbf{z}_1)\delta_z + O(\sigma^2)$, then the expected codelength for f_1, using $f_K \circ \cdots \circ f_2$ as a prior, is $-\log p_{f_K \circ \cdots \circ f_2}(f_1(\mathbf{x})) - \log|\det \mathbf{J}_{f_1}(\mathbf{x})| - \log \delta_x + O(\sigma^2)$. Continuing the same into $f_K \circ \cdots \circ f_2$, we conclude that the resulting expected codelength

$$-\log p(\mathbf{z}_K) - \sum_{i=1}^{K} \log|\det \mathbf{J}_{f_i}(\mathbf{z}_{i-1})| - \log \delta_x + O(\sigma^2), \tag{3.42}$$

where $\mathbf{z}_0 \coloneqq \mathbf{x}$, is what we expect from coding with the whole composite flow f. This codelength is averaged over noise injected into each layer $\mathbf{z}_i$, but we find that this is not an issue in practice. Our experiments in Section 3.6 show that it is easy to make σ small enough to be negligible for neural network flow models, which are generally resistant to activation noise.

We call this the *compositional* algorithm. Its significance is that, provided that coding with each intermediate flow is efficient, coding with the composite flow is efficient too, despite the complexity of the composite flow as a function class. The composite flow's Jacobian never needs to be calculated or factorized, leading to dramatic speedups over using Algorithm 3.1 on the composite flow as a black box (Section 3.6). In particular, coding with RealNVP-type models needs just $O(d)$ time and space and is fully parallelizable over data dimensions.

3.5.5 Dequantization for coding unrestricted-precision data

We have shown how to code data discretized to high precision, achieving codelengths close to $-\log p(\bar{\mathbf{x}})\delta_x$. In practice, however, data is usually discretized to low precision; for example, images from CIFAR10 and ImageNet consist of integers in $\{0, 1, \ldots, 255\}$. Coding this kind

of data directly would force us to code at a precision $-\log \delta_x$ much higher than 1, which would be a waste of bits.

To resolve this issue, we propose to use this extra precision within another bits-back coding scheme to arrive at a good lossless codelength for data at its original precision. Let us focus on the setting of coding integer-valued data $\mathbf{x}^\circ \in \mathbb{Z}^d$ up to bins of volume 1. Recall from Section 3.4.1 that flow models are trained on such data by minimizing a dequantization objective (3.19), which we reproduce here:

$$\mathbb{E}_{\mathbf{u} \sim q(\mathbf{u}|\mathbf{x}^\circ)} \left[\log q(\mathbf{u}|\mathbf{x}^\circ) - \log p(\mathbf{x}^\circ + \mathbf{u}) \right] \tag{3.43}$$

Above, $q(\mathbf{u}|\mathbf{x}^\circ)$ is a dequantizer, which adds noise $\mathbf{u} \in [0,1)^d$ to turn $\mathbf{x}^\circ$ into continuous data $\mathbf{x}^\circ + \mathbf{u}$ [105, 99, 93, 38].

We assume that the dequantizer is itself provided as a flow model, specified by $\mathbf{u} = q_{\mathbf{x}^\circ}(\boldsymbol{\epsilon}) \in [0,1)^d$ for $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, as in [38]. In Algorithm 3.4, we propose a bits-back coding scheme in which $\bar{\mathbf{u}} \sim q(\bar{\mathbf{u}}|\mathbf{x}^\circ)\delta_x$ is decoded from auxiliary bits using local bits-back coding, and $\mathbf{x}^\circ + \mathbf{u}$ is encoded using the original flow $p(\mathbf{x}^\circ + \bar{\mathbf{u}})\delta_x$, also using local bits-back coding.

Algorithm 3.4 Local bits-back coding with variational dequantization

Require: flow density p, dequantization flow conditional density q, discretization volume δ_x

procedure ENCODE($\mathbf{x}^\circ$) ▷ $\mathbf{x}^\circ$ is discrete data
  Decode $\bar{\mathbf{u}} \sim q(\bar{\mathbf{u}}|\mathbf{x}^\circ)\,\delta_x$ via local bits-back coding
  $\bar{\mathbf{x}} \leftarrow \mathbf{x}^\circ + \bar{\mathbf{u}}$ ▷ Dequantize
  Encode $\bar{\mathbf{x}}$ using $p(\bar{\mathbf{x}})\,\delta_x$ via local bits-back coding
end procedure

procedure DECODE( )
  Decode $\bar{\mathbf{x}} \sim p(\bar{\mathbf{x}})\,\delta_x$ via local bits-back coding
  $\mathbf{x}^\circ \leftarrow \lfloor \bar{\mathbf{x}} \rfloor$ ▷ Quantize
  $\bar{\mathbf{u}} \leftarrow \bar{\mathbf{x}} - \mathbf{x}^\circ$
  Encode $\bar{\mathbf{u}}$ using $q(\bar{\mathbf{u}}|\mathbf{x}^\circ)\,\delta_x$ via local bits-back coding
  return $\mathbf{x}^\circ$
end procedure

The decoder, upon receiving $\mathbf{x}^\circ + \bar{\mathbf{u}}$, recovers the original $\mathbf{x}^\circ$ and $\bar{\mathbf{u}}$ by rounding. So, the net codelength for Algorithm 3.4 is given by subtracting the bits needed to decode $\bar{\mathbf{u}}$ from the bits needed to encode $\mathbf{x}^\circ + \bar{\mathbf{u}}$:

$$\log q(\bar{\mathbf{u}}|\mathbf{x}^\circ)\delta_x - \log p(\mathbf{x}^\circ + \bar{\mathbf{u}})\delta_x + O(\sigma^2) = \log q(\bar{\mathbf{u}}|\mathbf{x}^\circ) - \log p(\mathbf{x}^\circ + \bar{\mathbf{u}}) + O(\sigma^2) \tag{3.44}$$

This codelength closely matches the dequantization objective (3.43) on average, and it is reasonable for the low-precision discrete data $\mathbf{x}^\circ$ because, as we stated in Section 3.4.1, it is a variational bound on the codelength of a certain discrete generative model for $\mathbf{x}^\circ$, and

modern flow models are explicitly trained to minimize this bound [105, 99, 38]. Since the resulting code is lossless for $\mathbf{x}^\circ$, Algorithm 3.4 provides a new compression interpretation of dequantization: it converts a code suitable for high precision data into a code suitable for low precision data, just as the dequantization objective (3.43) converts a model suitable for continuous data into a model suitable for discrete data [99].

3.6 Compression experiments

We designed experiments to investigate the following: (1) how well local bits-back codelengths match the theoretical codelengths of modern flow models on high-dimensional data, (2) the effects of the precision and noise parameters δ and σ on codelengths (Section 3.5.3), and (3) the computational efficiency of local bits-back coding for use in practice. We focused on Flow++ [38], a recently proposed RealNVP-type flow with a flow-based dequantizer. We used all concepts presented in this chapter: Algorithm 3.1 for elementwise and convolution flows [52], Algorithm 3.2 for coupling layers, the compositional method of Section 3.5.4.3, and Algorithm 3.4 for dequantization. We used asymmetric numeral systems (ANS) [23], following the BB-ANS [102] and Bit-Swap [55] algorithms for VAEs (though the ideas behind our algorithms do not depend on ANS). We expect our implementation to easily extend to other models, like flows for video [57] and audio [80], though we leave that for future work.

3.6.1 Codelengths

Table 3.3 lists our codelengths on the test sets of CIFAR10, 32x32 ImageNet, and 64x64 ImageNet. The listed theoretical codelengths are the average negative log likelihoods of our model reimplementations, without importance sampling for the variational dequantization bound, and we find that our coding algorithm attains very similar lengths. To the best of our knowledge, these results are state-of-the-art for lossless compression with fully parallelizable compression and decompression when auxiliary bits are available for bits-back coding.

Table 3.3: Local bits-back codelengths (in bits per dimension)

Compression algorithm	CIFAR10	ImageNet 32x32	ImageNet 64x64
Theoretical	3.116	3.871	3.701
Local bits-back (ours)	3.118	3.875	3.703

3.6.2 Effects of precision and noise

Recall from Section 3.5.3 that the noise level σ should be small to attain accurate codelengths. This means that the discretization volumes δ_x and δ_z should be small as well to make

discretization effects negligible, at the expense of a larger requirement of auxiliary bits, which are not counted into bits-back codelengths [35]. Above, we fixed $\delta_x = \delta_z = 2^{-32}$ and $\sigma = 2^{-14}$, but here, we study the impact of varying $\delta = \delta_x = \delta_z$ and σ: on each dataset, we compressed 20 random datapoints in sequence, then calculated the local bits-back codelength and the auxiliary bits requirement; we did this for 5 random seeds and averaged the results. See Fig. 3.6 for the results, and see the published paper [40] for more detailed results with standard deviation bars.

We indeed find that as δ and σ decrease, the codelength becomes more accurate, and we find a sharp transition in performance when δ is too large relative to σ, indicating that coarse discretization destroys noise with small scale. Also, as expected, we find that the auxiliary bits requirement grows as δ shrinks. If auxiliary bits are not available, they must be counted into the codelength for the first datapoint, which can make our method impractical when coding few datapoints or when no pre-transmitted random bits are present [102, 55]. The cost can be made negligible by coding long sequences, such as entire test sets or audio or video with large numbers of frames [80, 57].

3.6.3 Computational efficiency

We used OpenMP-based CPU code for compression with parallel ANS streams [30], with neural net operations running on a GPU. See Table 3.4 for encoding timings and Table 3.5 for decoding timings, all averaged over 5 runs, on 16 CPU cores and 1 Titan X GPU. We computed total CPU and GPU time for the black box algorithm (Algorithm 3.1) and the compositional algorithm (Section 3.5.4.3) on single datapoints, and we also timed the latter with batches of datapoints, made possible by its low memory requirements (this was not possible with the black box algorithm, which already needs batching to compute the Jacobian for one datapoint).

We found that encoding and decoding times were nearly identical. Furthermore, we found that the total CPU and GPU time for the compositional algorithm was only slightly slower than running a pass of the flow model without coding, whereas the black box algorithm was significantly slower due to Jacobian computation. This confirms that our Jacobian-free coding techniques are crucial for practical use.

3.7 Discussion

We have built upon bits-back coding [110, 35, 88, 26, 23, 102, 55] to enable flow models to perform lossless compression, which is already possible with VAEs and autoregressive models with certain tradeoffs. VAEs and flow models (RealNVP-type models specifically) currently attain similar theoretical codelengths on image datasets [38, 64] and have similarly fast coding algorithms, but VAEs are more difficult to train due to posterior collapse [12], which implies worse net codelengths unless carefully tuned by the practitioner. Compared with the most recent instantiation of bits-back coding for hierarchical VAEs [55], our algorithm and models

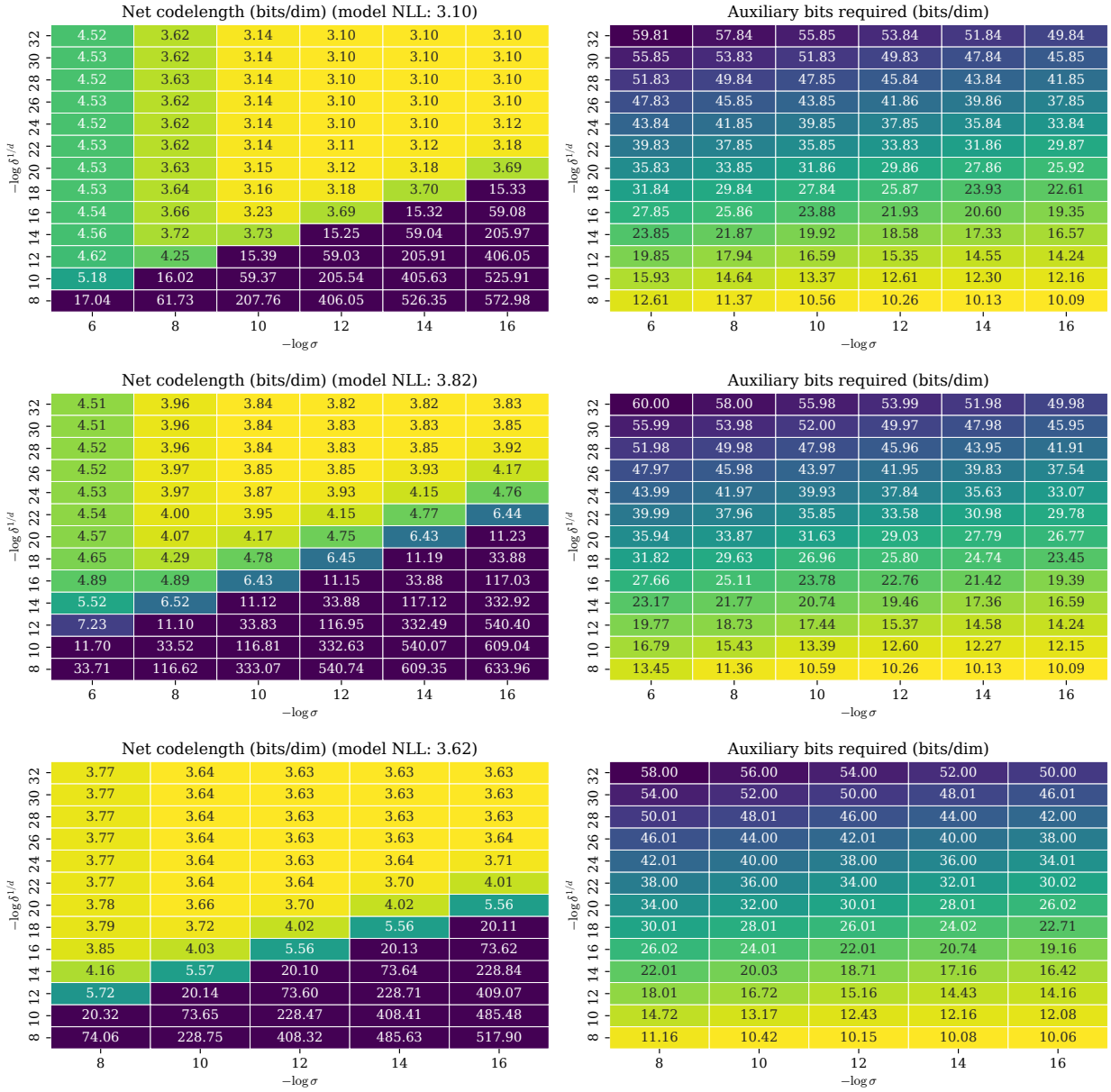


Figure 3.6: Effects of precision and noise parameters δ and σ on coding random subsets of CIFAR10 (top), ImageNet 32x32 (middle), and ImageNet 64x64 (bottom)

attain better net codelengths at the expense of a large number of auxiliary bits: on 32x32 ImageNet, we attain a net codelength of 3.88 bits/dim at the expense of approximately 40 bits/dim of auxiliary bits (depending on hyperparameter settings), but the VAEs can attain a net codelength of 4.48 bits/dim with only approximately 2.5 bits/dim of auxiliary bits [55]. As discussed in Section 3.6, auxiliary bits pose a problem when coding a few datapoints at a

Table 3.4: Encoding time per datapoint (in seconds)

Algorithm	Batch size	CIFAR10	ImageNet 32x32	ImageNet 64x64
Black box (Algorithm 3.1)	1	64.37 ± 1.05	534.74 ± 5.91	1349.65 ± 2.30
Compositional (Section 3.5.4.3)	1	0.77 ± 0.01	0.93 ± 0.02	0.69 ± 0.02
	64	0.09 ± 0.00	0.17 ± 0.00	0.18 ± 0.00
Neural net only, without coding	1	0.50 ± 0.03	0.76 ± 0.00	0.44 ± 0.00
	64	0.04 ± 0.00	0.13 ± 0.00	0.05 ± 0.00

Table 3.5: Decoding time per datapoint (in seconds)

Compression algorithm	Batch size	CIFAR10	ImageNet 32x32	ImageNet 64x64
Black box (Algorithm 3.1)	1	65.90 ± 0.10	564.42 ± 15.26	1351.04 ± 3.31
Compositional (Section 3.5.4.3)	1	0.78 ± 0.02	0.92 ± 0.00	0.71 ± 0.03
	64	0.09 ± 0.00	0.17 ± 0.00	0.18 ± 0.00
Neural net only, without coding	1	0.50 ± 0.03	0.76 ± 0.00	0.44 ± 0.00
	64	0.04 ± 0.00	0.13 ± 0.00	0.05 ± 0.00

time, but not when coding long sequences like entire test sets or long videos. It would be interesting to examine the compression performance of models which are VAE-flow hybrids, of which dequantized flows are a special case (Section 3.5.5).

Meanwhile, autoregressive models currently attain the best codelengths (2.80 bits/dim on CIFAR10 and 3.44 bits/dim on ImageNet 64x64 [14]), but decoding, just like sampling, is extremely slow due to serial model evaluations. Both our compositional algorithm for RealNVP-type flows and algorithms for VAEs built from independent distributions are parallelizable over data dimensions and use a single model pass for both encoding and decoding.

Concurrent work [34] proposes Eq. (3.24) and its analysis in Section 3.5.3 to connect flows with VAEs to design new types of generative models, while by contrast, we take a pretrained, off-the-shelf flow model and employ Eq. (3.24) as artificial noise for compression. While the local bits-back coding concept and the black-box Algorithm 3.1 work for any flow, our fast linear time coding algorithms are specialized to autoregressive flows and the RealNVP family; it would be interesting to find fast coding algorithms for other types of flows [32, 6], investigate non-image modalities [57, 80], and explore connections with other literature on compression with neural networks [3, 4, 33, 100, 87].

3.8 Recent related work

A survey by Papamakarios et al. [75] summarizes much of recent research in areas involving flow model architectures. Recent related work in data compression includes a paper of Townsend et al. [103], who show that fully convolutional VAEs trained on small images are able to compress large images without retraining. Hoogeboom et al. [43, 42] present related work in the intersection of data compression and flow models in the form of improved dequantizers, generalizations of variational dequantization, and integer discrete flows, which they show are amenable to computationally efficient compression without bits-back coding.

Chapter 4

Conclusion

The methods we have introduced are our attempts to resolve statistical and computational inefficiencies raised by the curse of dimensionality in machine learning. Our attempt to improve the statistical efficiency of imitation learning in the low expert data regime led to our algorithm, generative adversarial imitation learning, which synthesizes the training procedure of generative adversarial networks with the frameworks of apprenticeship learning, inverse reinforcement learning, and imitation learning. Our attempt to improve the computational efficiency of compression in the high entropy data regime led to our new flow model architecture and matching coding algorithm, local bits-back coding, and together they form a fast, parallelizable image compression system that may serve as a template for future practical compression systems based on deep generative models.

These are only beginnings of contributions to the longstanding endeavor of building intelligent, practically useful agents that maximally exploit past information to act for future benefit in a wide variety of problem domains. Resource limits, both statistical and computational, are fundamental to the existence of these agents because real agents are subsystems of the world—time passes and energy is consumed by each step of data acquisition and computation. We hope that our work may serve as inspiration for future research that confronts these limits in new settings.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, 2004.

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

[3] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[4] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

[5] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-13(5):834–846, 1983.

[6] Jens Behrmann, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *arXiv preprint arXiv:1811.00995*, 2018.

[7] Anthony J Bell and Terrence J Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural computation*, 7(6):1129–1159, 1995.

[8] Michael Bloem and Nicholas Bambos. Infinite time horizon maximum causal entropy inverse reinforcement learning. In *53rd IEEE Conference on Decision and Control*, pages 4911–4916. IEEE, 2014.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[12] Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. In *International Conference on Learning Representations*, 2017.

[13] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. PixelSNAIL: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 863–871, 2018.

[14] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[15] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[16] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[17] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International Conference on Machine Learning*, pages 933–941, 2017.

[18] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

[19] Gustavo Deco and Wilfried Brauer. Higher order statistical decorrelation without information loss. In *Advances in Neural Information Processing Systems*, pages 247–254, 1995.

[20] Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[21] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

[22] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, pages 1087–1098, 2017.

[23] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[24] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *arXiv preprint arXiv:1611.03852*, 2016.

[25] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[26] Brendan J. Frey and Geoffrey E. Hinton. Efficient stochastic source coding and an application to a Bayesian network source model. *The Computer Journal*, 40(2_and_3): 157–165, 1997.

[27] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018.

[28] Alborz Geramifard, Christoph Dann, Robert H Klein, William Dabney, and Jonathan P How. Rlpy: A value-function-based reinforcement learning framework for education and research. *JMLR*, 2015.

[29] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. MADE: Masked autoencoder for distribution estimation. In *International Conference on Machine*

Learning, pages 881–889, 2015.

[30] Fabian Giesen. Interleaved entropy coders. *arXiv preprint arXiv:1402.3392*, 2014.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[32] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[33] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In *Advances In Neural Information Processing Systems*, pages 3549–3557, 2016.

[34] Alexey A Gritsenko, Jasper Snoek, and Tim Salimans. On the relationship between normalising flows and variational-and denoising autoencoders. *ICLR Deep Generative Models for Highly Structured Data Workshop*, 2019.

[35] Geoffrey E Hinton and Drew Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13, 1993.

[36] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[37] Jonathan Ho, Jayesh Gupta, and Stefano Ermon. Model-free imitation learning with policy optimization. In *International Conference on Machine Learning*, pages 2760–2769, 2016.

[38] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, 2019.

[39] Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019.

[40] Jonathan Ho, Evan Lohn, and Pieter Abbeel. Compression with flows via local bits-back coding. In *Advances in Neural Information Processing Systems*, pages 3874–3883, 2019.

[41] Antti Honkela and Harri Valpola. Variational learning and bits-back coding: an information-theoretic view to Bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810, 2004.

[42] Emiel Hoogeboom, Jorn Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems*, pages 12134–12144, 2019.

[43] Emiel Hoogeboom, Taco S Cohen, and Jakub M Tomczak. Learning discrete distributions by dequantization. *arXiv preprint arXiv:2001.11235*, 2020.

[44] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2083–2092, 2018.

[45] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[46] Aapo Hyvärinen and Petteri Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439, 1999.

[47] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.

[48] Nal Kalchbrenner, Lasse Espheholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.00527*, 2016.

[49] Nal Kalchbrenner, Aäron Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning*, pages 1771–1779, 2017.

[50] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. *arXiv preprint arXiv:1802.08435*, 2018.

[51] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[52] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

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

[54] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[55] Friso H Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *International Conference on Machine Learning*, 2019.

[56] Leon Gordon Kraft. *A device for quantizing, grouping, and coding amplitude-modulated pulses*. PhD thesis, Massachusetts Institute of Technology, 1949.

[57] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. VideoFlow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2019.

[58] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37, 2011.

[59] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

[60] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *International Conference on Machine Learning*, pages 41–48, 2012.

[61] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*,

pages 19–27, 2011.

[62] Yunzhu Li, Jiaming Song, and Stefano Ermon. InfoGAIL: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pages 3812–3822, 2017.

[63] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational Bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227, 2017.

[64] Lars Maaløe, Marco Fraccaro, Valentin Liévin, and Ole Winther. BIVA: A very deep hierarchy of latent variables for generative modeling. In *Advances in Neural Information Processing Systems*, pages 6548–6558, 2019.

[65] Brockway McMillan. Two inequalities implied by unique decipherability. *IRE Transactions on Information Theory*, 2(4):115–116, 1956.

[66] Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. In *International Conference on Learning Representations*, 2019.

[67] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. *arXiv preprint arXiv:1811.12817*, 2018.

[68] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[69] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

[70] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *arXiv preprint arXiv:1808.03856*, 2018.

[71] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[72] XuanLong Nguyen, Martin J Wainwright, and Michael I Jordan. On surrogate loss functions and f-divergences. *The Annals of Statistics*, pages 876–904, 2009.

[73] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018.

[74] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017.

[75] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[76] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064, 2018.

[77] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM*

Transactions on Graphics (TOG), 37(4):1–14, 2018.

[78] Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Variational discriminator bottleneck: Improving imitation learning, inverse RL, and GANs by constraining information flow. In *International Conference on Learning Representations*, 2019.

[79] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[80] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. WaveGlow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

[81] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[82] Prajit Ramachandran, Tom Le Paine, Pooya Khorrami, Mohammad Babaeizadeh, Shiyu Chang, Yang Zhang, Mark A Hasegawa-Johnson, Roy H Campbell, and Thomas S Huang. Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*, 2017.

[83] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

[84] Scott Reed, Aäron van den Oord, Nal Kalchbrenner, Sergio Gómez Colmenarejo, Ziyu Wang, Yutian Chen, Dan Belov, and Nando Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning*, pages 2912–2921, 2017.

[85] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015.

[86] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[87] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In *International Conference on Machine Learning*, pages 2922–2930. JMLR. org, 2017.

[88] Jorma J Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of research and development*, 20(3):198–203, 1976.

[89] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *AISTATS*, pages 661–668, 2010.

[90] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, pages 627–635, 2011.

[91] Stuart Russell. Learning agents for uncertain environments. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 101–103. ACM, 1998.

[92] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[93] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017.

[94] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

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

[96] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 7461–7472, 2018.

[97] Umar Syed and Robert E. Schapire. A game-theoretic approach to apprenticeship learning. In *Advances in Neural Information Processing Systems*, pages 1449–1456, 2007.

[98] Umar Syed, Michael Bowling, and Robert E. Schapire. Apprenticeship learning using linear programming. In *International Conference on Machine Learning*, pages 1032–1039, 2008.

[99] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, pages 1–10, 2016.

[100] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *International Conference on Learning Representations*, 2017.

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

[102] James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *International Conference on Learning Representations*, 2019.

[103] James Townsend, Thomas Bird, Julius Kunze, and David Barber. HiLLoC: lossless image compression with hierarchical latent variable models. In *International Conference on Learning Representations*, 2020.

[104] Benigno Uria, Iain Murray, and Hugo Larochelle. RNADE: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013.

[105] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016.

[106] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[107] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *International Conference on Machine Learning*, 2016.

[108] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with PixelCNN decoders. *arXiv preprint arXiv:1606.05328*, 2016.

[109] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[110] Christopher S Wallace and David M Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, 1968.

[111] Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. *arXiv preprint arXiv:1906.02634*, 2019.

[112] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, AAAI'08, 2008.

[113] Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey. Modeling interaction via the principle of maximum causal entropy. In *International Conference on Machine Learning*, pages 1255–1262, 2010.