UNIVERSITY OF
CAMBRIDGE

Methods for Constructing and Exploiting Information Measures for Neural Networks

Paris Dominic Louis Flood

Darwin College

June, 2024

This dissertation is submitted for the degree of Doctor of Philosophy

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Paris Dominic Louis Flood
June, 2024

Abstract

Methods for Constructing and Exploiting Information Measures for Neural Networks

Paris Dominic Louis Flood

Gregory Chaitin, the noted mathematician and co-founder of algorithmic information theory, once put forth the claim that "compression is comprehension" [28]. This hypothesis, which echoes similar sentiments found in principles and theories such as Occam's Razor and Solomonoff induction, underscores the duality of information and learning. In this thesis we put forth a selection of methods studying and leveraging this duality under the context of neural networks.

The first subject of discussion is the description length of deep neural networks (DNNs) and the various methodologies for estimating minimum description length (MDL) codelengths. The symbiotic relationship between neural networks and compression research has produced profound work in both fields. For example, many state-of-the-art compression algorithms are driven by neural networks; in the other direction, variational inference for neural networks has important roots in the MDL principle and its compression perspective. However, DNNs appear to violate MDL tenants and the compression-comprehension hypothesis as they achieve excellent generalisation on a variety of complex tasks, yet have exceedingly large parameter counts. In an important paper which studied this apparent contradiction, Blier and Ollivier [16] found that the description lengths of DNNs could be significantly reduced using a coding technique derived from the prequential approach to statistics. Prequential coding outperformed all other algorithms, including two-part and variational methods, and has endured as the standard, even powering the benchmark-topping `NNCP` compression algorithm [12, 13]. Our research asks the following question: are there better alternatives to prequential coding for obtaining description lengths of DNNs? We start our investigation by generalising the prequential method into a broad framework we call instructional coding and find that other compression algorithms also fall under this paradigm. We proceed to describe a particular instance of instructional coding based on intermediate datasets which allow false labelling. This approach, which we term prechastic coding, achieves promising results and is competitive with prequential

coding under select scenarios.

In the subsequent chapter we progress to the topic of diffusion and conditional generation and present our paper on importance-guided diffusion. Diffusion models are a class of generative models inspired by non-equilibrium thermodynamics which have become extremely popular in recent years for image synthesis and other related tasks. The generative process involves an extensive sequence of latent variables which iteratively refine Gaussian noise into an image that resembles the images in the training set. This process has a clear information-theoretic interpretation and in our research we harness this perspective to design a principled conditional generative algorithm which does not require any retraining of the original neural network. This is accomplished using techniques from relative entropy coding, a subject which was also employed in our research on prequential coding, and achieves visually compelling results.

Our final setting is the algorithmic notion of information (Kolmogorov complexity) along with related topics such as algorithmic probability which, together, have laid important theoretical foundations for machine learning. While practical results are elusive due to the uncomputability of Kolmogorov complexity, recent work on subjects such as simplicity bias and generalization in deep neural networks have brought algorithmic information theory back into the applied spotlight. We present our paper on regularising link prediction for graphs with potentially simple generating mechanisms. Specifically, we detail a regularised loss function for link prediction with graph neural networks utilising machinery which claims to estimate the Kolmogorov complexity of graphs. Although the loss function demonstrates improved link prediction on a selection of real-world networks, this performance is roughly equivalent to that of a control version which does not directly estimate Kolmogorov complexity. Consequently, we conclude that the successful regularisation results were not due to any actual approximation of the Kolmogorov complexity and frame our findings within the context of an ongoing debate about such estimation methods.

The algorithms presented in this thesis offer a glance into the rich intersection of information theory, description lengths and neural networks. Furthermore, they advance the current understanding of methods for gauging and manipulating notions of information pertaining to neural networks for both compressive and generative purposes. The resulting insights represent stepping stones towards future research into neural network performance from an information-theoretic and MDL perspective.

Acknowledgements

This thesis would not have been possible without the support of a large cast of splendid people:

My **parents** for everything,

My **grandparents** for their eternal presence,

My **siblings and cousins** for the lifelong companionship,

My **aunts and uncles** for their extended support,

My **advisor** for his kindness and his brilliance,

My **friends** for the comfort and the laughter,

My **mentors** for the confidence,

My **collaborators** for their insights,

My **teammates** for the camaraderie,

My **family-in-law** for their care,

and **Laura** for her enduring love.

Contents

Abbreviations

ANS Asymmetric Numeral Systems.

AP Average Precision.

AUC Area Under the Receiver Operating Characteristic Curve.

DNN Deep Neural Network.

ELBO Evidence Lower Bound.

FIFO First In, First Out.

GAE Graph Autoencoder.

GCN Graph Convolutional Network.

GNN Graph Neural Network.

IGD Importance-Guided Diffusion.

LIFO Last In, First Out.

LTCB Large Text Compression Benchmark.

MDL Minimum Description Length.

MLP Multilayer Perceptron.

NELBO Negative Evidence Lower Bound.

ORC Ordered Random Coding.

REC Relative Entropy Coding.

VAE Variational Autoencoder.

VGAE Variational Graph Autoencoder.

Chapter 1

Introduction

> Learning is most intense and effective when it has an emotional, not just an intellectual, component, when there are no explicit rules and the organism is thrown upon its basic resources for survival.
>
> *Heinz Pagels* [124]

The story of life is a story of information - information generation, information storage, information destruction and, of course, information transfer. This relationship is well conveyed by the major evolutionary transitions as theorised by Maynard Smith and Szathmáry [118]. The earliest objects to display the essential characteristics of life were replicating molecules which progressed to populations of molecules cooperating in a protocell. From there, independent molecular replicators, *i.e.* genes, formed chains known as chromosomes which replicated together and consequently fostered further cooperation between the molecules. Eventually the RNA world gave rise to DNA and protein enzymes which marked a separation between information storage and the catalysis of chemical reactions. The next four transitions nearly bridged the gap to the current state of affairs: prokaryote to eukaryote, asexual to sexual, protists to multicellular organisms, and individuals to colonies. Yet it was the final transition, the development of language, which marked the shift from primate societies to human societies and enabled the creation of science and advanced technologies.

These eight major transitions are not necessarily authoritative; for example, Maynard Smith and Szathmáry themselves suggested that the exclusion of the development of the nervous system was a mistake [118]. However, they paint a milestone-based picture of evolution where information is acquired, processed and transmitted in increasingly sophisticated and cooperative ways. Many of the resulting improvements in efficiency

are both patently obvious and, in some cases, mathematically demonstrable. For example, one can easily show that on a toy model of evolution, sexual reproduction obtains information/increases fitness significantly faster than asexual reproduction [112].

The story of life is also a story of learning, especially if one adopts an expansive definition of learning based on adaptation and information retention. In this broad sense, the very nature of evolution could be construed as form of generational learning. Even under a stronger, non-inherited definition restricted to an organism's lifespan, learning can speed up the evolutionary process, a phenomenon which has been demonstrated in rudimentary computer simulations [73].

While the connections between information and learning are readily manifested in the biological world, these connections are even more apparent when both concepts are imbued with formalism. Rigorous definitions of information have a long history with origins in classical philosophy and Plato's theory of forms [1]. In the mid-20^{th} century, Shannon [142] introduced a theory of communication which advanced a notion of information based on probabilistic uncertainty. In his landmark paper, Shannon defined information entropy as a property of a random variable which measures that variable's inherent uncertainty and parallels the concept of entropy in statistical mechanics.

Shannon's work served as the foundation of the field of information theory and endures as one of the most widely used approaches to quantitatively addressing the concept of information. Unsurprisingly, information theory has had a widespread impact on the fields of artificial intelligence and machine learning, the magnitude of which is difficult to overstate. Standard information theory concepts such as entropy or Kullback-Leibler divergence are ubiquitous tools for developing and analysing learning algorithms. This utility extends to the most important research questions in machine learning today; for example, the information bottleneck method [158] has been shown to be an important tool for studying generalisation in deep learning [157].

In contrast to Shannon's approach, in the 1960s, Solomonoff, Kolmogorov, and Chaitin independently discovered an alternative definition of information rooted in computation [25, 96, 148–150]. Informally, the algorithmic or Kolmogorov complexity of an object is the length of the shortest computer program which produces that object. Intuitively, as a gauge of information content, Kolmogorov complexity assigns higher values to objects with more 'randomness' (in fact, randomness can be formally defined using Kolmogorov complexity [108]). This stands as a philosophical divergence to classic information theory which views information content as a measure of surprise, however there are deep equivalences between Kolmogorov complexity and Shannon entropy (see Section 2.3.2.2).

Kolmogorov complexity is at the heart of many of the attempts to answer the deepest questions in learning theory. For example, Solomonoff's theory of inductive inference [149, 150] is a formal theory of induction based on Occam's razor and Epicurus' principle

of multiple explanations. Marcus Hutter's AIXI formalism expands many of these ideas to a reinforcement learning setting and is a universal theory of sequential decision making [82].

1.1 Themes, Publications and Outline

This thesis addresses a set of questions and ideas which are situated at the fruitful intersection of information and learning. Broadly these topics can be organized along the three themes: information measures for the creation, existence, and usage of learning models. To further clarify, we investigate the utility of information measures during the creation of learning models, the intrinsic information contained within an existing learning model, and the extrinsic information generated by a trained learning model. In keeping with the current machine learning zeitgeist, our primary choice of learning modality is the neural network and, in particular, the deep neural network. DNNs have been pivotal to recent successes in several branches of machine learning [101, 139], however there is still no comprehensive theoretical consensus as to why deep learning works as well as it does [187]. In fact, at a high level such networks appear to contradict Occam's razor as they have massive amount of tunable parameters yet generalise extremely effectively.

Motivated by this discrepancy we first begin with the theme of existence and study the description length of deep neural networks. There is a large body of existing literature exploring the relationship between the amount of information contained by a network and its performance and, furthermore, contributing algorithms for measuring this information content [16, 74]. If one ignored the data, one might consider the true lower bound to be the Kolmogorov complexity of the trained network; however, the Kolmogorov complexity is uncomputable. In this thesis we will adopt an MDL approach (which, informally, is concerned with the cost of transmitting a model and the data encoded with the model, sometimes in a somewhat simultaneous manner) and examine these procedures in detail. Specifically, we study the question "Can shorter description lengths than the prevailing prequential codes be generated for neural networks?" To this end we design a novel deep neural network description length methodology which achieves levels of performance close to the state-of-the-art in specific contexts and has a large capacity for improvement.

Subsequently, we progress our discourse from the construction of intrinsic measures of neural network information to the exploitation of information measures for generative and predictive purposes, *i.e.* the theme of usage. More precisely, we study how some of the underlying machinery in our previous discussion of description lengths can be repurposed to alter the generative process of an important class of models. We explicitly answer the question "Can we alter the flow of information in diffusion models to generate conditional images using principals from relative entropy coding?"

Finally, we end with the theme of measuring information during the creation of learning models and switch our working definition of information from the Shannon approach to algorithmic complexity and ask if Kolmogorov approximation methods can improve link prediction in GNNs. Precisely, we ask the question "Can a measure of algorithmic complexity be used to regularize link prediction in graph neural networks?" We will expand upon these research directions in the following paragraphs, however a high-level visual summary can be found in Figure 1.1.

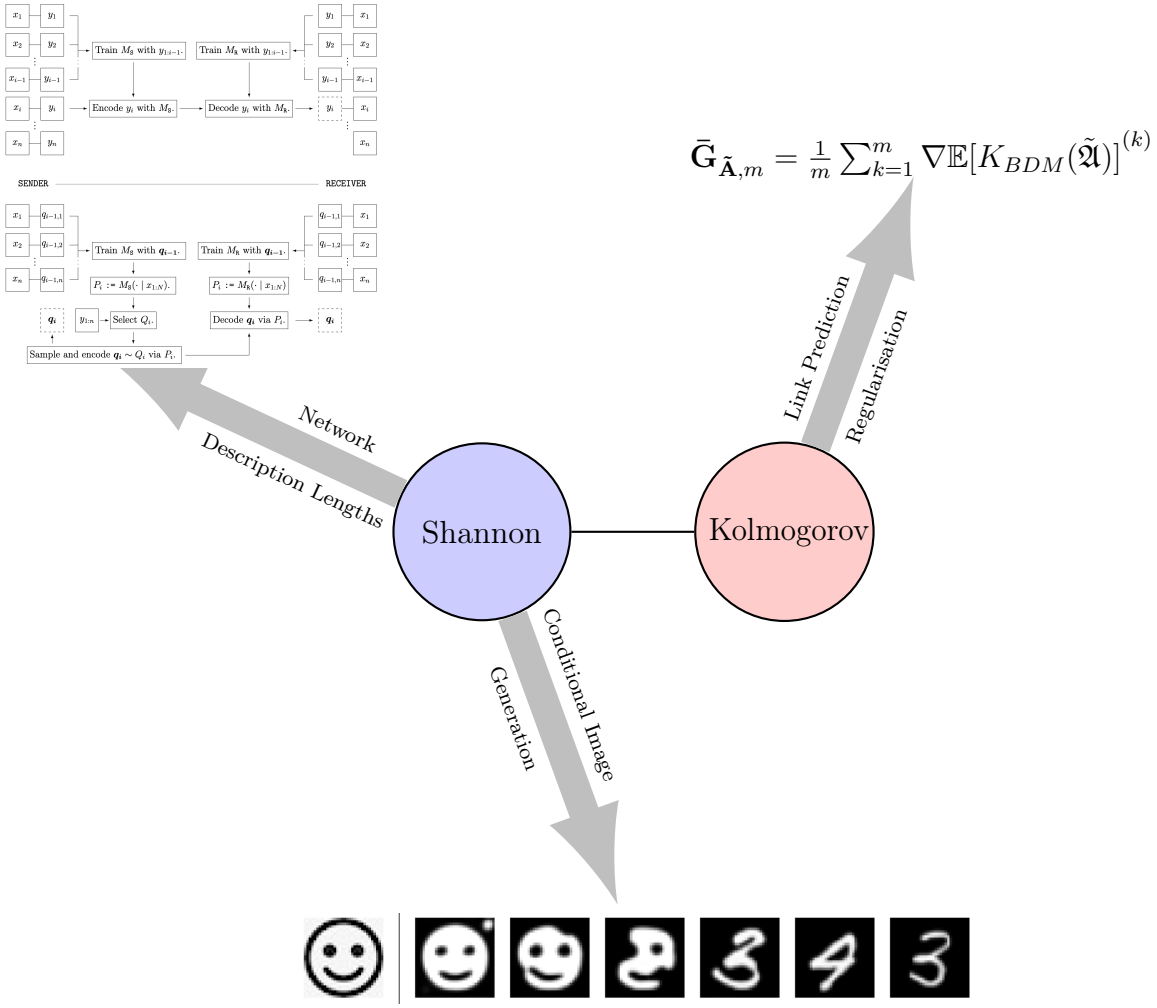


Figure 1.1: Visualisation of the research undertaken in this thesis. We consider definitions of information rooted in both classic information theory (Shannon) and algorithmic (Kolmogorov) complexity. Chapter 3 studies the description lengths of neural networks and introduces a new estimation paradigm called prechastic coding which engenders a greedy algorithm that is competitive with the state-of-the-art. Chapter 4 repurposes tools underlying prechastic coding for principled conditional image generation using diffusion models. Chapter 5 investigates whether an algorithm which claims to be a useful approximater of Kolmogorov complexity in graphs can be used to regularise link prediction in graph neural networks.

Throughout the course of my doctoral studies, I authored three papers whose contents

are presented within this thesis:

1. Flood, Paris DL, Viñas, Ramon, and Liò, Pietro (2020). "Investigating Estimated Kolmogorov Complexity as a Means of Regularization for Link Prediction." *2020 NeurIPS Causal Discovery & Causality-Inspired Machine Learning Workshop*. [52]

2. Flood, Paris and Liò, Pietro (2023). "Importance-Guided Diffusion." *NeurIPS 2023 Workshop on Diffusion Models*. [51]

3. Flood, Paris Dominic Louis and Liò, Pietro (2024). "Prechastic Coding: An Alternative Approach to Neural Network Description Lengths." *Workshop on Machine Learning and Compression, NeurIPS 2024*. [53]

Additionally, I co-authored three papers on various topics in machine learning which, although not directly related to the contents in this thesis, were engaging research endeavours:

1. Flood, Ian and Flood, Paris DL (2022). "Intelligent Control of Construction Manufacturing Processes using Deep Reinforcement Learning." *SIMULTECH*, pp. 112–122. [50]

2. Jensen, Andrew J, Flood, Paris DL, Palm-Vlasak, Lindsey S, Burton, William S, Chevalier, Amélie, Rullkoetter, Paul J, and Banks, Scott A (2023). "Joint track machine learning: an autonomous method of measuring total knee arthroplasty kinematics from single-plane X-ray images." *The Journal of Arthroplasty*, 38(10), pp. 2068–2074. Elsevier. [84]

3. Yeh, King Fai, Flood, Paris, Redman, William, and Liò, Pietro (2023). "Learning Linear Embeddings for Non-Linear Network Dynamics with Koopman Message Passing." *arXiv:2305.09060*. [179]

We will now outline the structure of thesis and introduce the three research directions which we present in Chapters 3-5. Each of these chapters investigates a question linked with the previously discussed themes and culminates with novel findings. The thesis content is arranged in the reverse chronological order of when the research was performed, as this was pedagogically superior. Chapter 2 contains both notation and background information; to the qualified reader, this chapter can be safely skipped and used simply as a reference. Because these works were undertaken in collaboration with my advisor Pietro Liò and, in one instance, Ramon Viñas (another doctoral student at the department), each of the following synopses will also include a detailed explanation of my contribution to the project. To briefly summarise my contributions, in all three cases I was the lead researcher and I was always listed as the first author on the corresponding papers.

Chapter 3 The first content-based chapter focuses on the description lengths of deep neural networks and related topics in lossless compression. Deep neural networks are renowned for both their exceptional performance on a wide range of tasks as well as their extensive parameter counts. Under a naive coding scheme where each parameter is stored as a floating point number, the resulting size of a trained DNN is of the same order as the parameter count. This large size seemingly contrasts with the ability of DNNs to generalise as one might expect such models to over-fit.

The minimum description length principle is a model selection principle which has had a significant impact on the field of machine learning and neural networks in particular. Early MDL work on neural networks by Hinton and van Camp [74] introduced well-known topics such as the bits-back argument and paved the way for future variational methods [63]. In 2018, Blier and Ollivier [16] discovered that previous MDL/network compression techniques for estimating the description lengths of deep neural networks were subpar. Instead, they found that a simple scheme known as prequential or online coding produced state-of-the-art results. These results were the best evidence that DNNs can be represented along with their datasets using far less information than would be suggested by their large parameter counts, thereby resolving the earlier contradiction.

Prequential coding stems from the prequential approach to statistics pioneered by Dawid [35]. The prequential approach essentially operates by coding a data unit or batch, then improving the model using the coded data in order to reduce the cost of sending future data. This is repeated iteratively and in sequential order until the entire dataset has been transmitted, thus translating the ability of the model to generalise into coding efficiency. Although prequential coding has achieved state-of-the-art results, in this chapter we ask, "is it possible to create a better method of determining description lengths of deep neural networks?" We start our investigation by generalising prequential coding into a paradigm we call instructional coding. We demonstrate that other existing compression techniques also fall under this framework, but note the difficulties associated with designing instructional coding schemes.

In order to challenge prequential coding, we develop an alternative, general-purpose instructional coding algorithm called prechastic coding. This approach views intermediate data as inherently stochastic and progressively lowers the levels of noise using the given learning model and some external information. Eventually all noise is completely eliminated and the original dataset is recovered. Our experiments demonstrate that under certain circumstances, prequential coding can achieve compression levels close to that of prequential coding; we also note that prechastic coding has a large margin for improvement and is a promising topic for future research. The corresponding paper that I authored with my advisor Pietro Liò, "Prechastic Coding: An Alternative Approach to Neural Network Description Lengths" was accepted at the *Workshop on Machine Learning and Compression*

at the 2024 edition of *Neural Information Processing Systems*. We were honoured to have the paper selected for an oral presentation at the workshop (which I gave virtually) - one of only a handful of papers to achieve this recognition out of a highly competitive pool of submissions.

Chapter 4 After our detailed investigation into the description lengths of deep neural networks we shift our focus to generative networks and the flow of information throughout the generative process. More specifically, we study deep diffusion models - a relatively new class of generative modelling strongly influenced by non-equilibrium thermodynamics. Introduced by Sohl-Dickstein et al. [145], diffusion essentially works by adding varying levels of Gaussian noise to objects of interest, *e.g.* images, from a training set. A model, typically a deep neural network, then learns to denoise each object using the noisy object and an indication of the noise level as inputs and the original object as the ground truth. Then, during the generative process, the model is supplied with pure Gaussian noise which it iteratively denoises until some final object is produced, *e.g.* an image similar to the images from the training set.

The diffusion methodology has a direct information-theoretic interpretation and, furthermore, can be recast as a codec-style coding procedure which broadly fits into the instructional coding paradigm. We draw from this insight and leverage the earlier relative entropy coding machinery in order to design a conditional generative algorithm which hijacks the original trained network and steers it towards objects of interest which can be completely different from the training set. This is all accomplished without retraining the original network. We call our algorithm `Importance-Guided Diffusion` and demonstrate its ability to achieve blended image styles based on a hyper-parameter which controls the degree of conditioning.

This project stemmed from discussions with Pietro Liò, where he suggested I channel various research questions I was pursuing (pertaining to deep neural networks and information theory) towards the burgeoning class of generative models based on the diffusion process. After a few months I developed the aforementioned algorithm based on relative entropy coding techniques and established the proof of concept by designing and conducting the experiments. As the first author, I wrote the method up as the paper "Importance-Guided Diffusion" with Pietro Liò as the second author. This paper was accepted as a poster at the peer-reviewed *Workshop on Diffusion Models* at the 2023 edition of *Neural Information Processing Systems*. In December of 2023 I presented the corresponding poster in-person at the workshop in New Orleans.

Chapter 5 In general, machine learning techniques that explicitly appeal to the concept of information do so under the Shannon-style variant based on probability theory and statistics. As previously mentioned, algorithmic or Kolmogorov complexity is rooted in

computing and is in some sense a more fundamental notion of information. Unfortunately, it is uncomputable and, as a result, in machine learning research it is typically relegated to theoretical work. However, attempts to integrate Kolmogorov complexity into applied machine learning research have been made through the use of approximation methods. While these methods often rely on crude techniques such as adapting general purpose compressors, there is evidence that some of these approaches can achieve meaningful results.

The research presented in this chapter originated from questions concerning whether or not such approximation methods were powerful enough to train neural networks using a controllable level of inductive bias towards algorithmically simple objects. Further discussions with Pietro Liò refined our focus to graph neural networks (GNNs) and, consequently, I spent a large portion of my first year working on simplicity regularisation techniques for link prediction problems using GNNs. I had recently come across research that promised meaningful approximations of Kolmogorov complexity for graphs, and was eager to see if these could be baked into GNN training. My colleague Ramon Viñas, was interested in causality and thought that were important connections to be made between causality and biasing the link prediction problem towards algorithmically simple graphs. Spurred on, we wrote the paper "Investigating Estimated Kolmogorov Complexity as a Means of Regularization for Link Prediction."

To clarify our contributions, I developed the regularisation algorithm and was first author, Ramon and I both wrote the paper, and Pietro provided guidance and supervision. During this process I realised that, although the technique presented in the paper appeared to work well, if we controlled for the Kolmogorov complexity estimation procedure we used, the results were the same. The results of this experiment led to the conclusion that the regularisation technique worked for reasons other than any actual direct estimation of the Kolmogorov complexity. The final paper was accepted at the peer-reviewed *Causal Discovery & Causality-Inspired Machine Learning Workshop* at the 2020 edition of *Neural Information Processing Systems*. Due to the Covid-19 pandemic, in December of 2020 I presented the corresponding poster online at the virtual-only workshop.

Chapter 2

Background

> The limits of my language mean the limits of my world.
>
> *Ludwig Wittgenstein* [171]

In this chapter we will establish common terminology, outline a mathematical notation system, and concisely present a selection of prerequisite topics. Knowledge of these topics will generally be assumed in later expositions, therefore it is important to establish some level of familiarity before proceeding. Specifically, we will review some core tenants of information theory, learning models, and algorithmic information theory.

2.1 Terminology and Notation

This section introduces much of the general terminology and notation used throughout the thesis and at points we will use alternative notations local to the relevant topics. Naturally, as we progress through the chapters, new definitions will arise. However, the following paragraphs implement a comprehensive baseline from which we later build upon.

Linear Algebraic Objects Scalars are represented with italic, lowercase Latin or Hellenic letters: x, y, z, μ, γ, etc. Vectors use the same set of symbols but have an additional boldface formatting: $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\gamma}$, etc. Matrices retain the use of boldface but change the case from lower to upper and do not use Hellenic letters: $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$, etc. Hellenic letters are not permitted for matrices in order to avoid potential confusion with various identical uppercase letters in the Latin alphabet as well as to avoid conflicts with common operators such as Σ which, while not boldfaced, are still uppercase.

Probability and Random Variables Random variables are denoted by uppercase italic Latin letters regardless of the dimension of the variable. Outcomes of random

variables use the corresponding lowercase letters and are boldfaced if the variable is multidimensional. The probability that a random variable Y takes on a particular value x is expressed as $P(y = x)$ or equivalently $P_Y(x)$. In scenarios where it is obvious which random variable is under consideration, brevity is preferred and $P(x)$ is used. We will often refer to probability distributions as P or Q; in these cases the probability the distribution assigns to a particular value x is given by $P(x)$ or $Q(x)$, respectively.

Binary Strings and Computation When referring to binary strings, algorithms, Turing machines, and general entities which perform computations we make use of the `typewriter font`. We let $\mathcal{B}^* = \{\mathtt{\Lambda}, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}, \mathtt{000}, \dots\}$ be the set of finite binary strings, where Λ is the empty string. The infinite sequence $\mathtt{\Lambda}, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \dots$ is known as the length-increasing lexicographic order for finite binary strings and, in a slight abuse of notation, we define $\mathcal{B}^*_i$ as the zero-indexed i^{th} value in the sequence (*e.g.* $\mathcal{B}^*_4 \coloneqq \mathtt{01}$).

Datasets When a dataset $\mathcal{D}$ is finite, we represent the cardinality as $|\mathcal{D}| = N$. If the dataset is ordered, each unit of data identifies its place in the ordering via subscript (*e.g.* $x_1, x_2, \dots x_N$). In accordance with our vector notation, if each datum is multidimensional we use the boldfaced font $\boldsymbol{x}_i$. If it is unclear or purposely ambiguous whether each datum is multidimensional or not, the non-boldfaced font is used. Following common machine learning conventions, when referring to objects which are interpreted as units of data we retain the lowercase even if the underlying object is a matrix. The notation $x_{i:k}$ denotes the contiguous subsequence of data $x_i, x_{i+1}, \dots, x_k$. If the data is discrete we use K to denote the number of classes; the set of all data classes, $\mathcal{A}$, is referred to as the alphabet and the data classes $a_1, a_2, \dots a_K$ are known as the symbols.

Note that insufficient descriptions can lead to ambiguity over what is considered data and what is not considered data. For example, in an ordered collection of text, the data type has many common specifications (*e.g.* word-level, character-level, etc.). In such cases, great care will be given to further clarification; however, the guiding principle will be to follow the resolution of the model predictions.

Modelling A model, $\mathcal{M}$, is a set of functions which share the same functional form and differ only in the values of a parameter set $\Theta_{\mathcal{M}}$. Although we will leave the functional forms unrestricted for now, models that are compatible with a given dataset will typically assign a probability distribution, either over each symbol in the alphabet or over all possible strings of symbols of a pre-determined size. The notation $\mathcal{M}_i$ denotes the i^{th} model in a pool of candidate models.

The cardinality of the parameter set is known as the dimension of the model. Parameters are typically real-valued, but can take other forms as well such as binary or complex numbers. When referring to an instance of a model $\mathcal{M}$ with specific parameter values $\boldsymbol{\theta} \in \Theta_{\mathcal{M}}$, we use the notation $M_{\boldsymbol{\theta}}$. The notation M is not associated with a specific set of parameters, but instead serves as a variable which represents a model instance. We will sometimes use the phrase "model" to refer to a "model instance", however it should be clear from the context what the precise meaning is.

Learning A learning regimen, R, is an algorithm that selects a particular model instance M_{θ^*} when given a model $\mathcal{M}$ and a dataset $\mathcal{D}$. The dataset must be compatible with the model and, in turn, the the model must be compatible with the learning regimen. Additionally, the algorithm is associated with a potentially empty set of hyper-parameters, $\Phi_{\mathcal{R}}$. We denote a set of learning regimens as $\mathcal{R}$ and when referring to a regimen with specific hyper-parameter values $\boldsymbol{\phi}$, we use the notation $R_{\boldsymbol{\phi}}$. The goal of the learning process is to step through the parameter space of the model in search of an instance which performs well on some criteria for the given dataset.

Coding L is used as a base notation to represent functions which measure the size of the aforementioned digital objects (e.g. data, models, parameter values, etc.) in bits under a given coding scheme. Although there will be a variety of different coding schemes presented in this chapter, the notation $L(x_i \mid M_{\boldsymbol{\theta}})$ is used to represent $-\log_2(p_{M_{\boldsymbol{\theta}}}(x_i))$, the Shannon information content of x_i given the model instance. When discussing coding schemes, we will often frame the central problem as a transmission between two parties: a sender, `S`, and a receiver, `R`.

2.2 Information Theory and Learning

Consider the following two finite strings:

$$\mathbf{s_A} = \texttt{0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1}\ldots\texttt{0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1}$$
$$\mathbf{s_B} = \texttt{1 0 1 0 0 0 1 1 1 0 0 1 0 1 0 1}\ldots\texttt{0 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1}$$

The above representations are truncated in the interest of saving space, but if we were to examine the full strings we could observe the following details:

- $\mathbf{s_A}$ and $\mathbf{s_B}$ are both binary with identical lengths of one million bits.
- $\mathbf{s_A}$ is composed entirely of a repeated pattern of four `0`s followed by four `1`s.
- $\mathbf{s_B}$ has no readily discernible pattern, but it does have a roughly even split of `0`s and `1`s.

We will not imbue these strings with any particular meaning, nor will we discuss their intended usage. Despite this absence, both strings are clear examples of an intuitive notion of *information*. However, one would naturally prefer a precise and useful definition of information that allows for rigorous quantification of the level of information present. In this chapter we shall discuss two important means of accomplishing this, beginning with the widespread definition advanced by Claude Shannon in his classic 1948 paper "A Mathematical Theory of Communication" [142].

The essence of Shannon's approach is to interpret a given discrete object x as an outcome of a random variable X and define the *information content* of the object as $-\log_2(P(x))$. In this manner, unlikely outcomes convey high amounts of information while likely ones convey low amounts. Additionally, the *entropy* or average information content of a random variable is set as $H(X) = -\sum_x P(x) \cdot \log_2(P(x))$. This definition fulfils an axiomatic rationale based on desirable characteristics of an information measure.

Paramount to the usefulness of Shannon information are valuable properties such as the convexity of the negative logarithm as well as the natural integration with statistical modelling techniques. Consider a simple probabilistic model of the earlier strings $\mathbf{s_A}$ and $\mathbf{s_B}$ which sets the probability of each bit at 0.5. In this scenario both strings have a Shannon information content of one megabit. However, if we allow our model to learn on the job we quickly find that we can do much better on the first string $\mathbf{s_A}$. For example, say that our model processes each string one bit at a time and predicts the next bit by assigning the probability 0.75 to the value of the previous bit and 0.25 to the alternate outcome. Under this scheme the information content of $\mathbf{s_A}$ is approximately .811 megabits, nearly 200 kilobits less than before, while $\mathbf{s_B}$ balloons to approximately 1.208 megabits.

This simple example illustrates the importance of the prediction model when measuring the Shannon information. Furthermore, Shannon's source coding theorem, which essentially states that entropy is the fundamental limit to the lossless data compression rate in the limit, underscores the close relationship between compression and learning. The field of data compression is often concerned with finding improved data models in order to reduce the compression size. As a result, advanced learning algorithms such as deep neural networks hold an important role in many highly efficient compression schemes [177]. In Chapter 3 we explore a slight twist on this topic where, instead of trying to improve the quality of the learning algorithm, we study the coding schemes oriented towards a specific class of learning algorithms (neural networks).

2.3 Algorithmic Information Theory

In the field of algorithmic information theory, the concept of information is made precise through the use of computer programs. This stands in contrast to the approach of Shannon

where information is defined within the frameworks of probability and stochastic variables. Recall the two strings $\mathbf{s_A}$ and $\mathbf{s_B}$ from Section 2.2 and note that $\mathbf{s_A}$ can be generated by a rather short algorithm, *i.e.* `print '00001111' x 125000`. On the other hand, the simplest program that prints $\mathbf{s_B}$ is likely `print` followed by the full, million bit transcription of $\mathbf{s_B}$.

In this section we shall discuss fundamental definitions and results from algorithmic information theory. As this subject tends to be less familiar than the field of information theory, we will include substantially more detail than in the previous section. The primary aim of this review is to inform later discussions about applied algorithmic complexity and to cover the requisite material needed to follow the paper presented in Chapter 5. However, we will briefly delve into a few advanced topics of interest (specifically the subsections on randomness and Chaitin's Ω) purely for the sake of completeness as they underscore the rich variety of research in the field and add context to the motivations underlying Chapter 3. Most of the content in this section is derived from either Chaitin [27], Li et al. [108] or Cover and Thomas [34], and we would highly encourage the interested reader to consult these texts for a more comprehensive introduction to algorithmic information theory and Kolmogorov complexity.

2.3.1 Computation

In 1936, Alan Turing created a model of computation which was simple to understand, yet rich enough to perform any calculation that could be completed by a mechanical method [161]. Although Turing machines can be defined in many ways, we will use the following construction: consider a two-way infinite tape comprised of squares which are filled with exactly one symbol from the set $\Gamma = \{\mathtt{0}, \mathtt{1}, \sqcup\}$, where $\sqcup$ represents a blank. The input to the Turing machine is some finite binary string $x \in \mathcal{B}$ which is written on $|x|$ contiguous squares in the tape. All other tape squares start off as blanks. The machine has a head which points to exactly one square at a time and is initially located at the first square in the input sequence (this is arbitrary if the initial input is the empty string).

At each time step the machine can either move the head to the left, move the head to the right, or write a symbol from Γ at the current location of the head. These five choices constitute the set of actions $A = \{L, R, W_0, W_1, W_\sqcup\}$. Additionally, the machine is always in exactly one state from a finite set of possible states Q. One of these states is designated as the initial state q_{start}.

Computation is carried out according to a fixed set of rules which can be represented as a partial function $\delta : Q \times \Gamma \to Q \times A$. The machine deterministically chooses the next state and action as the value of δ for the current state and the symbol pointed to by the head. Note that because δ is a partial function, there may not be a corresponding procedure for a given state and symbol. In this case the machine is said to halt. A Turing

machine is deemed *universal* if it can simulate any other Turing machine given the correct input program.

In algorithmic information theory we make considerable use of a special type of Turing machine known as a prefix Turing machine. As with Turing machines, prefix Turing machines can be defined in many ways. We will mostly follow the construction used in [27]: consider a two-way infinite *work* tape and a finite *program* tape. Both tapes are comprised of squares filled with exactly one symbol from Γ. The work tape allows both read and write operations, while the program tape is read-only.

The program tape always begins with an initial blank square known as the dummy square. Subsequent squares constitute the program and must be either `0` or `1`. The program-head is initially placed at the dummy square and can only move to the right. The work tape is initially filled with blanks except for a finite binary string $x \in \mathcal{B}$ which is written on $|x|$ contiguous squares in the tape. The work-head is initially placed at the beginning of this string and can move in either direction as well as write any symbol in Γ.

We will use the shortened notation $\mathtt{T}(p)$ when referring to computations which start with a blank work tape (*i.e.* $q = \Lambda$). We are specifically interested in optimal universal prefix-free computers, which are defined in the following existential lemma.

Lemma 2.3.1. *There exists a universal, prefix-free computer* $\mathtt{U}$ *such that for every prefix-free computer* $\mathtt{T}$ *and* $p, q \in \mathcal{B}^*$*, if* $\mathtt{T}(p \mid q)$ *is defined, then there exists a* $p' \in \mathcal{B}^*$ *where*

$$\begin{aligned} \mathtt{U}(p' \mid q) &= \mathtt{T}(p \mid q), \\ |p'| &\leq |p| + c_{\mathtt{T}} \end{aligned}$$

for come constant $c_{\mathtt{T}}$ *which does not depend on Any such computer* $\mathtt{U}$ *that satisfies this property is deemed an optimal universal prefix-free computer.*

2.3.2 Kolmogorov Complexity

Discovered independently by Solomonoff [148, 149] Kolmogorov [96] and Chaitin [25, 26], Kolmogorov or algorithmic complexity is the cornerstone of algorithmic information theory. At a high level, the Kolmogorov complexity of a binary string is the length of the shortest computer program that outputs the string. As we shall see, this seemingly simple characterisation of complexity provides a fundamental basis for definitions of information and randomness. However, first we must provide a technically precise definition; although multiple variants exist, we will present the prefix definition introduced by Levin [105], Gács [56], and Chaitin [27].

The *Kolmogorov complexity* of a finite binary string $x \in \mathcal{B}^*$ with respect to an optimal,

prefix-free universal computer $\mathtt{U}$ is

$$K_{\mathtt{U}}(x) \coloneqq \min_{p\,:\,\mathtt{U}(p)=x} |p|$$

where $p \in \mathcal{B}^*$ is a program and the notation $\mathtt{U}(p) = x$ implies that p outputs x and halts. At first this might seem to be an arbitrary definition based on the choice of machine. One might correctly note that identical algorithms can have vastly differing implementation lengths based on the choice of programming language. However, this issue is resolved by the following invariance/universality result.

Theorem 2.3.2 (Invariance/Universality of Kolmogorov Complexity)**.** *Given two optimal universal prefix-free computers* $\mathtt{U}$ *and* $\mathtt{U'}$*, there exists a constant* c *such that for all* $x \in \mathcal{B}^*$

$$|K_{\mathtt{U}}(x) - K_{\mathtt{U'}}(x)| < c$$

The constant c is essentially equivalent to the length of an interpreter and, crucially, does not depend on the choice of x. This is an incredibly important result as it establishes Kolmogorov complexity as a language independent measure, up to an additive constant. As such, we shall fix an arbitrary optimal prefix-free universal computer $\mathtt{U}$ throughout the remaining text and simplify the notation $K_{\mathtt{U}}$ to K. Unfortunately, although Kolmogorov complexity is, in the sense of Theorem 2.3.2, a universal measure, it is also an incomputable one:

Theorem 2.3.3 (Incomputability of Kolmogorov Complexity)**.** *There is no program which takes any string* $x \in \mathcal{B}^*$ *and returns the Kolmogorov complexity* $K(x)$*.*

The incomputability result is directly related to the undecidability of the halting problem [161]. A simple informal proof that Kolmogorov complexity is incomputable can be developed by first assuming by contradiction that there exists a program $\mathtt{getKC}$ such that $\mathtt{getKC}(x)$ returns the Kolmogorov complexity of x. Note that $\mathtt{getKC}$ will have a finite program length $|\mathtt{getKC}|$. Now consider the $\mathtt{getComplex}$ program (see Algorithm 1) which iterates through each $\mathcal{B}_i^*$ until it finds a string that has larger Kolmogorov complexity than the sum of $|\mathtt{getKC}|$ and c, the number of extra bits to wrap $\mathtt{getKC}$ and obtain $\mathtt{getComplex}$. It follows from a simple counting argument that the number of iterations before such a string appears is $\mathcal{O}(2^N)$ where N is the size of the sum. However, there is a contradiction - the returned string has Kolmogorov complexity greater than N but has been computed by a program of size N; therefore, $\mathtt{getKC}$ cannot exist.

Now consider the following theorem about the frequency of low Kolmogorov complexity strings, already alluded to in the informal proof of incomputability.

Theorem 2.3.4 (Rarity of Low Kolmogorov Complexity Strings)**.** *For a given constant* c *there are less than* 2^c *finite binary strings with Kolmogorov complexity less than* c*.*

```
for i := 0 to ∞ {
        x ≔ B_i^*
        if getKC(x) > |getKC| + c {
                return x
        }
}
```

Algorithm 1: The `getComplex` program.

This result stems from a simple counting argument about the number of short programs and emphasises how rare low Kolmogorov complexity strings are. Amongst all strings of length $n > c$, the proportion of these strings which will have a Kolmogorov complexity of less than c is less than 2^{c-n}. In addition to these theorems, there are a handful of useful definitions directly related to the Kolmogorov complexity which we will briefly review:

Canonical Program Intuitively, the *canonical program* for x is the program which realises the Kolmogorov complexity of x. However, this formulation could lead to ambiguity if there are multiple programs of length $K(x)$ which output x. Therefore, the canonical program is defined as the first program x^* in the length-increasing lexicographic ordering of the finite binary strings such that $\mathtt{U}(x^*) = x$. Note that, trivially, the length of x^* is still $K(x)$.

Conditional Kolmogorov Complexity In our model of a computer, there is a clear distinction between the program tape and the work tape. The most straightforward definition of the conditional Kolmogorov complexity of $x \in \mathcal{B}^*$ given $y \in \mathcal{B}^*$ is $K(x \mid y)$ is the length of the smallest program to output x given y as input (*i.e.* the work tape is initialised to y). This definition is satisfactory and for the remainder of this thesis we shall employ it. However, it is important to note that there is a version which defines conditioning on y as providing the canonical program y^*, a change which improves information symmetry properties. In order to maintain consistency with our earlier policy, when denoting the Kolmogorov complexity of x conditioned on the canonical program of y we shall explicitly write $K(x \mid y^*)$ rather than $K(x \mid y)$.

2.3.2.1 Important Inequalities and Relations

In this section we shall establish essential properties of Kolmogorov complexity beyond the aforementioned theorems. Before beginning our enumeration of important inequalities and relations, we will establish notation for the concepts of additive relativity and equality based on the notation in [67]. Consider two functions f, g which both map $\mathcal{B}^*$ to $\mathbb{N}$. If there exists a constant c such that for all $x \in \mathcal{B}^*$ we have $f(x) \leq g(x) + c$, then we can write $f(x) \overset{+}{\leq} g(x)$.

Similarly, if the opposite direction is true we can use the expression $f(x) \overset{+}{\geq} g(x)$. If both directions hold, *i.e.* $f(x) \overset{+}{\leq} g(x)$ and $f(x) \overset{+}{\geq} g(x)$, then we denote the relationship between the two functions as $f(x) \overset{\pm}{=} g(x)$. We will now proceed with basic inequalities and relations, some of which have clear analogues with similar results in classical information theory.

Theorem 2.3.5 (Kolmogorov Complexity Bounds Conditioned on Length)**.** *For any* $x \in \mathcal{B}^*$,

$$K(x \mid |x|) \overset{+}{\leq} |x|$$

Theorem 2.3.6 (Kolmogorov Complexity Bounds)**.** *Define* $\log^* r$ *as the sum of iterated logarithms* $\log r + \log\log r + \log\log\log r \ldots$ *continued until all positive terms have been exhausted. For any* $x \in \mathcal{B}^*$,

$$\begin{aligned} K(x) &\overset{+}{\leq} K(x \mid |x|) + \log^* |x| \\ &\overset{+}{\leq} |x| + \log^* |x| \end{aligned}$$

Theorem 2.3.7 (Kolmogorov Complexity of Integers)**.** *For any integer* n*,*

$$K(n) \overset{+}{\leq} \log^* n$$

Theorem 2.3.8 (Relationship of Conditional, Unconditional, and Joint Kolmogorov Complexity)**.** *For any* $x, y \in \mathcal{B}^*$,

$$K(x \mid y) \overset{+}{\leq} K(x) \overset{+}{\leq} K(x, y)$$

Theorem 2.3.9 (Subadditivity of Kolmogorov Complexity)**.** *For any* $x, y \in \mathcal{B}^*$,

$$K(x, y) \overset{+}{\leq} K(x) + K(y \mid x) \overset{+}{\leq} K(x) + K(y)$$

Theorem 2.3.10 (Canonical Equivalence)**.** *For any* $x, y \in \mathcal{B}^*$,

$$K(y \mid x^*) \overset{\pm}{=} K(y \mid x, K(x))$$

Theorem 2.3.11 (Chain Rule for Kolmogorov Complexity)**.** *For any* $x, y \in \mathcal{B}^*$,

$$K(y) + K(x \mid y^*) \overset{\pm}{=} K(x, y) \overset{\pm}{=} K(x) + K(y \mid x^*)$$

Theorem 2.3.12 (Symmetry of Algorithmic Mutual Information)**.** *If we define the algorithmic mutual information in* $x \in \mathcal{B}^*$ *about* $y \in \mathcal{B}^*$ *as* $I(x : y) \coloneqq K(y) - K(y \mid x^*)$

then

$$I(x : y) \stackrel{+}{=} I(y : x)$$

Note that the algorithmic entropy of y conditioned on x is additively greater than or equal to the difference between the joint algorithmic entropy $K(x, y)$ and $K(x)$, while the algorithmic entropy of y conditioned on the canonical program x^* is additively equal. Similarly, if we were to define algorithmic mutual information without the canonical program, we would find that the additive equality would no longer hold. Additive equality is desirable as these relations hold with perfect equality in classical information theory. As a point of interest, it is possible to derive equality statements that hold to a logarithmic accuracy when not conditioning on canonical programs.

2.3.2.2 Relationship to Entropy and Randomness

Although the inequalities and relations presented in Section 2.3.2.1 mimic many of the results found in classical information theory, the extent of the relationship goes much further. In fact, the expected Kolmogorov complexity of a sequence of i.i.d. samples, given the length of the sequence, is close in value per sample to the entropy of the random variable for longer sequence samples. More formally, we have the following theorem:

Theorem 2.3.13 (Kolmogorov Complexity and Shannon Entropy)**.** *Consider a stochastic process $\{X_t\}_{t\in\mathbb{N}}$ where each X_t is drawn independently from some discrete probability distribution. If $X_{1:n}$ denotes the random variable corresponding to a sequential sample of length n, then there exist constants c_1, c_2 such that for all n*

$$H(X) \leq \frac{1}{n}\mathbb{E}[K(X_{1:n} \mid n)] \leq H(X) + c_1\frac{\log n}{n} + \frac{c_2}{n}.$$

This result is perhaps rather unsurprising, considering both concepts stem from notions of information, uncertainty, and compression. However, the depth of the relationship between Kolmogorov complexity and randomness goes even further as it turns out the former can be leveraged to define a powerful notion of randomness. In fact a bit sequence $x_{1:n}$ is *algorithmically random* if it satisfies the following inequality:

$$K(x_{1:n} \mid n) \geq n$$

This intuitive definition satisfies many philosophical considerations about what it means to be random and is an important concept in the larger field of algorithmic randomness initiated by Martin-Löf [116].

2.3.3 Algorithmic Probability

In the mid 1960s Ray Solomonoff, who had long been interested in induction [151], published his landmark theory of formal induction [149, 150] based on four philosophical pillars of modern science:

1. **Epicurus' Principle of Multiple Explanations** Keep all hypotheses which are consistent with the observed data.

2. **Occam's Razor** Entities should not be multiplied beyond necessity. In a certain sense this advocates for the simplest hypothesis from the set of all hypotheses which are consistent with the observed data. Note though that the concept of simplicity, while intuitive, can be hard to define.

3. **Bayes' Rule** The probability of a hypothesis given the observed data is proportional to the probability of the observed data given the hypothesis multiplied by the prior probability of the hypothesis.

4. **Church-Turing Thesis and Universality** Any real-world computation can be effectively carried out by a Turing machine; furthermore, there exist universal Turing machines which are capable of simulating any Turing machine.

At the core of Solomonoff's theory lies algorithmic probability, which assigns a prior to each finite binary string. This prior is known as the *universal probability* and is defined as

$$P_{\mathtt{U}}(x) \coloneqq \sum_{p\,:\,\mathtt{U}(p)=x} 2^{-|p|}$$

Because the collection of all valid programs for $\mathtt{U}$ form a prefix-free set, Kraft's inequality implies that $\sum_x P_{\mathtt{U}}(x) \leq 1$. As we shall in Section 2.3.3.1, $\sum_x P_{\mathtt{U}}(x)$ is a special value known as the *halting probability* or Ω and is, in fact, strictly less than one. The universal probability has a few special properties which make it an appropriate prior for a formal theory of induction.

First, note that hypotheses, *i.e.* programs that halt, formalise Occam's Razor by contributing more weight if they are shorter and less weight if they are longer. The explicit probabilities assigned can be derived from the following scenario: say that $\mathtt{U}$ is fed completely random bits. For a given output x, we would expect that a shorter program which outputs x is more likely to have generated x than a longer program which also outputs x. This is because the shorter sequence is more likely to appear as a prefix than the longer program. For example half of the random sequences will start with `0` but only one eighth will start with `111`.

Second, up to a multiplicative constant, the universal probability dominates any other probability distributions induced by a prefix-free Turing machine `T`. More formally, it

follows from the definition of an optimal universal prefix-free Turing machine that there exists some constant c such that $P_{\mathtt{U}}(x) \geq cP_{\mathtt{T}}(x)$, where c does not depend on x. An important takeaway from this bound is that if both $\mathtt{U}$ and $\mathtt{T}$ are universal, then there exists c_1, c_2 such that for all x, $c_1 \leq \frac{P_{\mathtt{U}}(x)}{P_{\mathtt{T}}(x)} \leq c_2$. In other words, the likelihood ratio between competing hypotheses that a string was drawn according to $P_{\mathtt{U}}(x)$ or $P_{\mathtt{T}}(x)$ is bounded.

The final property that we will discuss is the connection between universal probability and Kolmogorov complexity. From inspection we can see that $2^{-K(x)} \leq P_{\mathtt{U}}(x)$ since $\mathtt{U}(x^*) = x$. However, rather surprisingly, it turns out that there exists a constant c such that for all values of x, $P_{\mathtt{U}}(x) \leq c2^{-K(x)}$. This theorem, restated below in terms of additive equality, demonstrates the equivalence between universal probability and Kolmogorov complexity. The contribution of the smallest program dominates and essentially characterises the universal probability, despite the fact that there are an infinite number of programs which generate a given string.

Theorem 2.3.14 (Coding Theorem)**.** *For any* $x \in \mathcal{B}^*$,

$$-\log P_{\mathtt{U}}(x) \stackrel{+}{=} K(x)$$

2.3.3.1 Chaitin's Ω

Informally, Chaitin's Ω represents an extremely condensed form of knowledge encapsulated within a single real number. As mentioned in the previous section, Ω is simply the halting probability or, more precisely:

$$\Omega := \sum_{p\,:\,\mathtt{U}(p)\text{ halts}} 2^{-|p|}$$

Although the exact value of Ω is dependent on the choice of optimal prefix-free universal Turing machine, we shall refer to Ω assuming a fixed $\mathtt{U}$. Naturally, for reasons evident from the halting problem, Ω is non-computable and strictly less than one.

Ω has a rather powerful property which makes it of interest: if given $\Omega_{1:n}$, the first n bits of Ω, one can decide the truth of any provable theorem which can be stated in no more than n bits. The proof of this is rather straightforward. First, assuming that all terminating binary rational numbers are represented with an infinite number of trailing zeros instead of an infinite number of trailing ones, we have $\Omega < \Omega_{1:n} + 2^{-n}$. Now, using the length increasing lexicographic ordering, we can run all programs in parallel by dovetailing the computation in the following manner: at time i run the first i programs for one cycle (assuming they haven't halted). Eventually, the universal probability of all the programs that have halted will be greater than or equal to $\Omega_{1:n}$. At this point, no program with length less than or equal to n bits can halt; if it did, then the sum of the halting probabilities would exceed Ω.

Even knowledge of $\Omega_{1:n}$ up to a small precision of a few thousand bits would likely

be sufficient to express programs evaluating many statements of interest; for example, consider some concise program which attempts to generate a counter example to a classic problem from number theory. Truncated values for particular halting probabilities have been calculated by Calude and Dinneen [22], albeit to highly limited precisions. However, even if one were to calculate $\Omega_{1:n}$ to a large enough value of n so that interesting programs could be expressed, the amount of time to run the dovetailed halting check would be unfeasibly large.

Chapter 3

Prechastic Coding and Deep Neural Network Description Lengths

> Why unify information theory and machine learning? Because they are two sides of the same coin.
>
> *Sir David MacKay* [112]

At a high level, in this chapter we will study and present techniques for gauging the amount of information present in deep neural networks (DNN), mainly from an minimum description length principle (MDL) perspective. DNNs are an important class of learning algorithms which, in recent years, have shot to the forefront of machine learning research due to their dominant performance on a large variety of learning tasks [101]. For the most part we will gloss over the specific details of how these networks are constructed as they are largely irrelevant to the research presented in this chapter. Nevertheless, it is important to note that DNNs are characterised by extremely large parameter counts.

The large size of DNNs initially appears to create a contradiction as their excellent performance seemingly violates the spirit of Solomonoff's research and formalised notions of Occam's Razor. However, this contradiction relies on a naive coding scheme where parameters are encoded in full as floating points. The more sophisticated coding schemes which we shall discuss throughout this chapter are able to greatly reduce the estimated MDL codelengths for DNNs. In fact, there is a large body of research studying the realisable information of DNNs from an MDL perspective and in the coming sections we will perform critical analyses of this research covering a variety of perspectives.

The chapter is structured as follows: in Section 3.1 we discuss various direct and indirect motivations surrounding topics in the related fields of compression and the minimum description length principle, all under the context of neural networks. Section 3.2 presents the variational approach to description lengths which is of historical importance

and also ties in to later discussions about relative entropy coding. In Section 3.3 we review prequential coding which is a simple, state-of-the-art technique for obtaining description lengths of deep learning models. We then proceed in Section 3.4 to describe a generalisation of prequential coding and detail other existing algorithms which fall under this paradigm. Section 3.4, and the sections that follow, present our novel contributions whereas previous sections largely contain background information. In Section 3.5 we present a promising novel method called *prechastic coding* which is derived from this generalisation, is competitive with the state-of-the-art in select instances, and is amenable to further improvement. Finally, we discuss these results and conclude the chapter in Section 3.6. Portions of this chapter, and in particular Sections 3.5 and 3.6, were presented in the paper "Prechastic Coding: An Alternative Approach to Neural Network Description Lengths" with myself as the first author and Pietro Liò as the second author. This paper was accepted at the *Workshop on Machine Learning and Compression* at the 2024 edition of *Neural Information Processing Systems* where it was awarded with a workshop oral presentation. Much of the text, figures, tables, etc. in the paper appear largely as they do in this chapter with minor edits and modifications.

3.1 Compression and MDL Motivations

Without additional details, notions of compression and 'information measurement' in the context of neural networks/DNNs can have different meanings depending on the motivation. In many situations we are interested in bounds on the compressed size of a trained DNN indirectly through the efficient communication of some dataset. Before training, a DNN can be succinctly expressed using a description of its architecture and initialisation procedure, the latter of which might be considered part of the training regimen. In the case of a stochastic initialisation, identical initial parameters can be transmitted through the use of pseudorandom seeds (also considered part of the regimen). After/during training on a dataset of interest, the neural network parameters have meaning endowed by the training data and learning regimen.

This chapter exclusively uses the practical Shannon-style of information quantification; however, the ultimate description of information quantity in a trained model is still the Kolmogorov complexity. Note that, using notation introduced in Chapter 2, $K(M_{\theta^*}) \overset{+}{\leq} K(\mathcal{M}) + K(R) + K(\mathcal{D})$. In this chapter the methods we are interested in estimate MDL codelengths, however in many other contexts, often concerning bits-back coding and occasionally relative entropy coding, information content is defined relative to future data and the neural network is assumed to be trained and given. This is a markedly different task to estimating an MDL codelength, however it is of great theoretical and practical importance and we shall discuss it first.

3.1.1 Data Compression

One of the most straightforward and self-evident reasons for studying neural networks through an information-theoretic perspective is data compression. Numerous state-of-the-art coding methods leverage neural networks to compress commonly used data forms such as text and images far below their original sizes [13, 177].

3.1.1.1 Codecs

A fairly common paradigm in the neural compression literature is to pair a trained neural network with an entropy coder and treat the two as a codec for some data stream of interest. For example, consider a variational autoencoder trained to minimise the Negative Evidence Lower Bound (NELBO) for a dataset of images [91]. Together with an asymmetric numeral systems coder [44], the trained neural network can be used as a codec for future images which are similar to the training set [160]. This codec operates at the NELBO under the vanilla bits-back scheme (see Section 3.2.1), assuming there are enough images to amortise an initial cost.

The codec scenario often disregards the cost of transmitting the trained model when evaluating compression levels. There are several reasons why this could be an acceptable practice. It may be the case that the amount of data one would like to compress is so large that the savings from using a trained neural network render the contribution of the model transmission negligible. There may also be a timing factor where the real-time cost of communicating data is not always constant. For example, the neural codec might be installed before a period of streaming usage where bandwidth is at a premium. For some further discussion, please see Townsend et al. [160].

3.1.1.2 Fixed Datasets

Under the data compression umbrella, but distinct from the codec motivation is the task of compressing a fixed dataset. This is tantamount to developing upper bounds on the Kolmogorov complexity of a given dataset $\mathcal{D}$. Neural networks are a powerful tool for stationary compression and currently form the basis for state-of-the art methods for compressing fixed text datasets [13, 177]. Unlike codecs, in this setting one does not assume that the model has already been transmitted; the cost of compression includes, in some form, a communication of the model.

We will discuss this topic in further detail in Section 3.3, however model communication, if done efficiently, can often come with considerable time costs. For example, the prequential coding strategy requires multiple network training sessions throughout the coding process. As a consequence of the potentially high time costs, interest in the fixed dataset motivation is more academic than practical. In fact, two of the best known

competitions for compressing a fixed dataset, the Large Text Compression Benchmark (LTCB) [115] and its offshoot the Hutter prize [83], either do not restrict time (LTCB) or give a fairly generous amount of time (Hutter)[†]. Accordingly, these competitions use compression sizes to gauge and motivate advances in artificial intelligence, not to encourage the development of better algorithms for the sake of solely compressing fixed datasets.

3.1.2 Model Selection and Overfitting

A good deal of interest around measuring the information content of neural networks stems from the relationship between simplicity and generalisation [16, 74, 140]. Underpinning much of this research is the minimum description length principle, which advocates an inductive philosophy rooted in notions of compression [64]. At a high level, MDL requires model selection to be based on both model performance and complexity. For example, if two models M_a and M_b perform equally well on the data, MDL prefers the model which is simpler - a term which we shall soon make precise. The inclusion of information-based model complexity as either an implicit or explicit selection criterion can also be cast as a compromise with goodness-of-fit. As made clear by both classical and algorithmic information theory, both complexity and description length can be equated in a fundamental manner. Thus, model selection is clearly a motivating factor for the analysis of the information content of neural networks.

Highly complex models can easily memorize their training sets and become poor predictors of future data. In a similar spirit to model selection, complexity avoidance can also be deployed as a safeguard against overfitting [16]. Under this objective, the MDL philosophy extends into the learning process via the minimization of MDL loss functions [63]. The result is a learning procedure which, by incorporating MDL tenets, avoids overly complex model instances during the search for an optimal parameter set.

3.1.2.1 The MDL Principle

Because of the profound relevance of the MDL principle, we shall give an overview of the principle and its applications to the study of neural networks. For a thorough introduction to MDL, please consult Grünwald [66]. Pioneered by Jorma Rissanen [132–135], the MDL principle, much like Solomonoff's approach, draws upon a form of Occam's razor to perform inductive reasoning. The general concept of practical MDL is to descend from the abstract realm of the universal Turing machine and instead turn to useful but less powerful description methods. For historical reasons there are many versions of the MDL [66]; we will begin with arguably the simplest of these versions, the crude two-part MDL.

[†]Both competitions use one gigabyte of data for the compression task. The Hutter prize allows $^{70000}/_{T}$ hours of compute, where T is the computer's Geekbench score. On a 2.4 GHz i9 MacBook Pro this translates to about 52 hours of single-core processing.

Let $\mathcal{M} = \bigcup_i \mathcal{M}_i$ represent a pool of candidate models $\mathcal{M}_1, \mathcal{M}_2, \ldots$ from which we must select some model instance $M_\theta \in \mathcal{M}$ to explain the dataset $\mathcal{D}$. The crude two-part version of the MDL principle advocates that the best model instance for the data minimises $L(M_\theta) + L(\mathcal{D} \mid M_\theta)$. If there are multiple model instances which achieve this minimal value, the model instance with the smallest $L(M_\theta)$ is chosen; there is no preference beyond this criterion.

Information theory suggests that the natural choice for the code $L(\mathcal{D} \mid M_\theta)$ is one where the size of the data given the model instance is $-\log_2(p_{M_\theta}(\mathcal{D}))$. A code which achieves this length is optimal for the given model instance and can be implemented with minimal overhead for multiple predictions using arithmetic codes or asymmetric numeral systems (see Section 3.3.2 or Section 3.2.1). Unfortunately, while the choice of code for $L(\mathcal{D} \mid M_\theta)$ is relatively straightforward, the situation is quite different for $L(M_\theta)$.

Under the crude MDL framework, the only restriction on the code design for $L(M_\theta)$ is that it cannot depend on N, the size of the dataset [65]. Although this grants great flexibility, it also creates an arbitrariness as the code length for a given model instance can be large under one coding scheme and small under another. This issue is a blight on the crude variant and we would prefer a more thoughtful methodology for model coding. Early work by Barron and Cover [11] proposed principles for model coding that would anticipate the modern version of the MDL known as refined MDL.

While crude MDL is based on coding data using model instances, refined MDL uses a different approach built around the model and a one-part code with length $L'(\mathcal{D} \mid \mathcal{M})$. Central to the refined MDL is the concept of universal codes which satisfy the property that if there exists a $M_\theta \in \mathcal{M}$ such that $L(\mathcal{D} \mid M_\theta)$ is small, then $L'(\mathcal{D} \mid \mathcal{M})$ is small [10, 66]. Examples of universal codes include the normalised maximum likelihood code and the prequential code, the latter of which is based on the prequential approach to statistics [35] and turns out to highly useful for gauging the information content of deep neural networks (see Section 3.3).

In fact, the minimum description length principle has had a profound impact on the development of neural networks. In a seminal paper, Hinton and van Camp [74] framed the MDL approach as a form of network regularisation and introduced the bits-back argument (see Section 3.2.1). This work was the genesis of variational inference for neural networks which has, in turn, led to an influential practical variational implementation by Graves [63]. Furthermore, the bits-back method has been repurposed into a chain-like compression scheme [54] which, due to recent advances by Townsend et al. [160], can realise the negative evidence lower bound for variational autoencoders. Unsurprisingly, autoencoder training has long been associated with the MDL framework [75].

3.2 Variational Coding

Before we discuss the state-of-the-art, we will review the variational approach to description lengths as some of the underlying problem structure will be of importance to our later work on prechastic coding in Section 3.5. Although variational coding has been found to be inferior to the prequential approach when it comes to the description lengths of deep neural networks (see Section 3.3.5), variational methods in general are profoundly useful for codec-style compression [92, 160] and variational approaches to training have demonstrated truly revolutionary generative capabilities [91, 145, 152]. In this section we shall present variational coding through a description of the bits-back method, a thought experiment turned important practical class of algorithms for lossless compression. Later in Section 3.5.2 we will discuss how an alternate sample communication tool, relative entropy coding (REC), avoids an important drawback to bits-back methods. This motivates the usage of relative entropy coding in the prechastic coding process to great practical effect. We again revisit relative entropy coding in Chapter 4 where it guides us in the creation of a principled conditional generative algorithm.

3.2.1 The Bits-Back Method

Originally introduced by Wallace [165] under the context of the Minimum Message Length method, the bits-back method was later rediscovered by Hinton and van Camp [74], who were interested in improving neural network generalisation through an MDL approach. The latter has endured as a seminal work on variational methods for neural networks [63], and for this reason we shall first review the bits-back method under the variational inference setting presented in the paper.

Consider a neural network, parameterised by $\boldsymbol{\theta} \in \mathbb{R}^M$, an input vector $\boldsymbol{x} \in \mathbb{R}^N$, and an output vector $\boldsymbol{y} \in \mathbb{R}^N$. We are interested in sending the output vector to a receiver `R` who has access to the input vector and a description of the neural network, but not the specific weights. If `R` did have access to $\boldsymbol{\theta}$, the cost of sending the correct values for a uniformly quantised $\boldsymbol{y}$ is $-\log_2\left(p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})\right)$ where $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})$ is the product of the probabilities that the neural network assigns to the intervals containing each element of $\boldsymbol{y}$. However, this raises the question of how one should interpret the network's predictions as interval assignments? A naive approach is to bound each element y_i above and below (*i.e.* $\boldsymbol{y} \in [l, h]^N$ for some $l < h$), and, given an interval width of t, output probabilities for each of the $\lceil \frac{h-l}{t} \rceil$ intervals; however, increasing the precision increases the number of intervals and, consequently, the size of the network.

Instead we set the network to output $\tilde{\boldsymbol{y}} \in \mathbb{R}^N$ and define $\Delta y_i \coloneqq y_i - \tilde{y}_i$ as the difference between the true value and the predicted value at index i. Say that `R` assigns the prior $\mathcal{N}(0, \sigma_i)$ for each Δy_i and quantises each distribution into intervals of width t. If $t \ll \sigma_i$, the

portion of probability assigned by the prior to the interval containing Δy_i is approximately

$$p(\Delta y_i) \approx \frac{t}{\sigma_i \sqrt{2\pi}} e^{\frac{-(y_i - \tilde{y}_i)^2}{2\sigma_i^2}}.$$

In order to minimise the expected code length for the output, the optimal value of σ_i is set to $\sigma^* = \sqrt{\frac{1}{N}\sum_i (y_i - \tilde{y}_i)^2}$. The resulting expected code length can be concisely expressed as $\frac{N}{2} \log\left(\frac{1}{N}\sum_i (y_i - \tilde{y}_i)^2\right) + cN$ where c is a constant that depends solely on t.

This is the cost in nats to send the correct output vector if the receiver already had access to the trained parameters $\boldsymbol{\theta}$. However, in reality R's neural network is untrained and the sender S will have to transmit $\boldsymbol{\theta}$. One might decide to encode these weights in a similar manner to the output vector, by assuming a Gaussian prior and then encoding the true value through quantisation. More specifically, assign the prior $\mathcal{N}(0, \sigma_W^2)$ to each weight, and note that the lowering the parameter description length is equivalent to minimising the sum of the squares of the weights.

Ideally we would like to get rid of the inefficient quantisation term in the MDL code length. To do this, we will change the concept of a weight from a learnable scalar to a Gaussian distribution with learnable parameters. Now assume that S and R have agreed to a Gaussian prior p over the weights. Once the neural network has been trained, the resulting distribution will be termed q. We will now describe the essence of the bits-back argument which, in this context, is a thought experiment about the allocated cost of communicating the posterior distribution and the true output vector.

First the sender samples q and obtains a precise value $\boldsymbol{\theta}$ for the weights of the network. Using a sufficiently fine quantisation t, these weights are transmitted at a cost of $C(\boldsymbol{\theta}) = -M \log t - \log p(\boldsymbol{\theta})$ nats. The sender S can now send over the aforementioned output vector code, as both S and R have the same set of weights. The receiver R can now perfectly recover the true output vector and, after training its network using this data, R can also learn the trained distribution q. With this knowledge of the posterior, R can determine the random bits used to collapse q to the weights $\boldsymbol{\theta}$ that were transmitted.

In some sense, the transmission of the specific weights $\boldsymbol{\theta}$ and corresponding output vector code was actually a communication of the true model posterior q, the true output vector $\boldsymbol{y}$, and the random bits to collapse the posterior. Since we are only interested in the cost of transmitting the model (*i.e.* the posterior) and the data, we need to subtract the cost of the random bits to obtain the MDL code length. The cost of collapsing the posterior q to a particular sample $\boldsymbol{\theta}$ for a sufficiently fine quantisation t is $R(\boldsymbol{\theta}) = -M \log t - \log q(\boldsymbol{\theta})$

nats. Therefore, the expected cost of transmitting the model is

$$\begin{aligned}\mathbb{E}_{\boldsymbol{\theta}\sim q}[\mathcal{L}_{q,p}] &= \mathbb{E}_{\boldsymbol{\theta}\sim q}[C(\boldsymbol{\theta}) - R(\boldsymbol{\theta})] \\ &= \mathbb{E}_{\boldsymbol{\theta}\sim q}[\log q(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})] \\ &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta})} \,\mathrm{d}\boldsymbol{\theta} \\ &= \mathrm{KL}[q \parallel p].\end{aligned}$$

Although this formulation of the bits-back argument is of relevance to the study of the MDL of neural networks [16], subsequent works have often taken an interest in bits-back from a latent variable perspective [55, 75, 160]. In this version of the method, the sender `S` is only attempting to communicate a realisation $\boldsymbol{y}_0$ of the output random variable $\boldsymbol{y}$ to the receiver `R`. Both `S` and `R` have access to a generative model, which takes the latent random variable $\boldsymbol{z}$ and outputs $p(\boldsymbol{y} \mid \boldsymbol{z})$, as well as a prior over the latents $p(\boldsymbol{z})$ and an approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{y})$. To simplify our exposition, we assume that all random variables are discrete; however, this is not strictly necessary.

Using entropy coding, `S` can leverage the prior and generative model to transmit some $\boldsymbol{z}_0 \sim q(\boldsymbol{z} \mid \boldsymbol{y}_0)$ at a cost of $-\log p(\boldsymbol{z}_0)$ and $\boldsymbol{y}_0$ at a cost of $-\log p(\boldsymbol{y}_0 \mid \boldsymbol{z}_0)$. However, just as before, the sender has actually transmitted more information than just the bits needed to obtain $\boldsymbol{z}_0$ from $p(\boldsymbol{z})$ and $\boldsymbol{y}_0$ from $p(\boldsymbol{y}_0 \mid \boldsymbol{z}_0)$. The receiver `R` can now recover the bits required to encode $\boldsymbol{z}_0$ from $q(\boldsymbol{z} \mid \boldsymbol{y}_0)$ as `R` has access to $\boldsymbol{z}_0, \boldsymbol{y}_0$, and $q(\boldsymbol{z} \mid \boldsymbol{y})$. After subtracting the recovered bits, the expected total transmission cost is

$$\begin{aligned}\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{y}_0)}[\mathcal{L}(\boldsymbol{y}_0)] &= \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{y}_0)}[-\log p(\boldsymbol{z}) - \log p(\boldsymbol{y}_0 \mid \boldsymbol{z}) + \log q(\boldsymbol{z} \mid \boldsymbol{y}_0)] \\ &= \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{y}_0)}\left[\log \frac{q(\boldsymbol{z} \mid \boldsymbol{y}_0)}{p(\boldsymbol{z})p(\boldsymbol{y}_0 \mid \boldsymbol{z})}\right] \\ &= \mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{y}_0)}\left[\log \frac{q(\boldsymbol{z} \mid \boldsymbol{y}_0)}{p(\boldsymbol{y}_0, \boldsymbol{z})}\right] \\ &= \mathrm{KL}[q(\boldsymbol{z} \mid \boldsymbol{y}_0) \parallel p(\boldsymbol{y}_0, \boldsymbol{z})].\end{aligned}$$

Now say that we are interested in transmitting a sequence of outputs $\boldsymbol{y}_{0:n}$. The bits-back with feedback trick [54] cleverly notes that after encoding the first value in the sequence, we can effectively use the auxiliary bits in the sequence to sample the posterior conditioned on the next value in the sequence. This process can be repeated for each sample, effectively reducing the average cost per sample to approximately $\mathbb{E}_{\boldsymbol{y}\sim p(\boldsymbol{y})}[\mathrm{KL}[q(\boldsymbol{z} \mid \boldsymbol{y}) \parallel p(\boldsymbol{y}, \boldsymbol{z})]]$ for large values of n.

To give a better understanding of how the process works at a high-level, let us step through the i^{th} encoding iteration of the bit-back with feedback method based on BB-ANS [160]. At this point in the process, the sender `S` has created a stack of bits $\mathbb{B}^{(i-1)}$

which contains the information the receiver will need to decode $\boldsymbol{y}_{0:i-1}$. S draws a sample $\boldsymbol{z}_i \sim q(\boldsymbol{z} \mid \boldsymbol{y}_i)$ by popping the top $-\log q(\boldsymbol{z}_i \mid \boldsymbol{y}_i)$ bits from $\mathbb{B}^{(i-1)}$ and using these bits to "decode" $\boldsymbol{z}_i$ from $q(\boldsymbol{z} \mid \boldsymbol{y}_i)$. As the auxiliary bits have essentially been discarded, S is now left with a new, shorter bit stack $\tilde{\mathbb{B}}^{(i-1)}$. S now encodes $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ at costs of $-\log p(\boldsymbol{y}_i \mid \boldsymbol{z}_i)$ and $-\log p(\boldsymbol{z}_i)$, respectively. These two codes are pushed onto the bit stack $\tilde{\mathbb{B}}^{(i-1)}$ yielding a new bit stack $\mathbb{B}^{(i)}$ which is $-\log p(\boldsymbol{z}_i) - \log p(\boldsymbol{y}_i \mid \boldsymbol{z}_i) + \log q(\boldsymbol{z}_i \mid \boldsymbol{y}_i)$ bits longer than $\mathbb{B}^{(i-1)}$.

If, at this point, S transmits $\mathbb{B}^{(i)}$, R can recover $\boldsymbol{y}_i$ and $\mathbb{B}^{(i-1)}$ in the following manner: first, R pops $-\log p(\boldsymbol{z}_i)$ bits to decode $\boldsymbol{z}_i$, then pops $-\log p(\boldsymbol{y}_i \mid \boldsymbol{z}_i)$ bits to decode $\boldsymbol{y}_i$. In order to transform the resulting bit stack $\tilde{\mathbb{B}}^{(i-1)}$ back to $\mathbb{B}^{(i-1)}$, R encodes $\boldsymbol{z}_i$ using $q(\boldsymbol{z} \mid \boldsymbol{y}_i)$ then pushes the resulting $-\log q(\boldsymbol{z}_i \mid \boldsymbol{y}_i)$ bits onto $\tilde{\mathbb{B}}^{(i-1)}$.

The BB-ANS approach developed by Townsend et al. [160] uses a combination of bits-back and Asymmetric Numeral Systems (ANS) [44] to create a lossless compression pipeline and performs well with Variational Autoencoder (VAE) [91] latent models. ANS is a form of entropy coding which operates as a stack on a LIFO basis as opposed to arithmetic coding (see Section 3.3.2) which behaves like a queue and is FIFO. This solved an existing difficulty in the implementation of bits-back with feedback which had relied on an inefficient hack to arithmetic coding [54]. Townsend et al. [160] also describe how their method can be adapted for continuous variables, using arguments that are similar to those found in [74].

The bits-back method is of historical importance to variational learning [63, 81] and, more recently, has found a variety of use cases including compression with flow models [77, 188] which have achieved impressive results on image datasets. Follow-ups to BB-ANS have extended the original method to produce results better than the NELBO [137] and take advantage of multiset structure in datasets [141]. The clean bits problem (*i.e.* sampling from the bit stack is not truly random) that existed in the original formulation has been solved using a clever encryption scheme [88] (see Appendix N in the paper). However, bits-back methods do suffer from a fundamental flaw - they only achieve optimal compression rates asymptotically. This drawback can be important in one-shot or few-shot tasks; in such scenarios one might prefer to use REC methodology (see Section 3.5.2).

3.3 The Prequential Approach

The prequential approach to statistics is a framework for analysing forecasting systems based on the principle that any assessment of such systems should be derived from the validity of their sequential predictions [35, 36]. This motivation is reflected in the etymology of "prequential" as a portmanteau of "predictive" and "sequential" [37] [†]. In prequential

[†]Dawid's original paper implies that the word "prequential" is also derived from "probability" [35].

statistics, data is assumed to be ordered and is often, although not necessarily, part of a time series. Data is processed in sequence and can subsequently be used to update the forecast for the next, unseen item of data. This unseen data is not revealed to the forecaster until after it has formed its prediction. In the following section we will present an overview of a coding process based on prequential statistics known as *prequential* or *online coding*. After building up a prequential toolkit, we will delve into the relationship between prequential coding and the information content of deep neural networks.

3.3.1 Overview

Imagine a communication scenario where we have a sender `S` and a receiver `R`, both of which have access to their own set of computing resources. `S` would like to send `R` a trained model which is a member of the model class $\mathcal{M}$ and has been trained on the dataset $\mathcal{D}$. We might alternatively say that `S` would like to send the dataset $\mathcal{D}$ to `R` and is constrained to the model class $\mathcal{M}$. For simplicity, we will assume that each data point belongs to one of K classes and that the model, based on some corresponding input, forms a probability prediction over the classes.

To initiate the prequential scheme, `S` first sends `R` a description of the model class $\mathcal{M}$. The size of the description will typically be inconsequentially small as it can be specified in a high-level programming language. `S` and `R` then initialise identical models according to the agreed-upon training protocol. If any portion of the initialisation or training is stochastic, `S` must also send a random seed so that `S` and `R` use identical pseudorandom numbers.

Next, `S` sends over the first piece of data, x_1, using equal probabilities for each of the K possible data classes. The cost of this transmission (i.e. the length of the code) is $\log_2(K)$ bits which is simply the Shannon entropy of a uniform distribution over K classes. We will discuss the specifics of how this is encoded and why it is perfectly acceptable to specify transmission costs in ideal terms in Section 3.3.2. The uniform distribution is used for the first transmission as neither model has been trained due to the receiver's lack of data. It is generally assumed that as this stage neither model is better at predicting than a random guess.

Once `R` has received the data, it trains its model on x_1 according to the agreed-upon training regimen. `S` trains an identical model by using the same training regimen and restricting its training data to x_1. `S` uses this new model to encode and send the next piece of data, x_2, to `R`. As before, this prediction process may not use any data other than x_1. The cost of this transmission is $-\log_2(p_M(x_2 \mid x_1))$ where $p_M(x_2 \mid x_1)$ is the probability that the model assigns to x_2 when provided with x_1 for training and prediction.

`R` decodes the transmission with its model and then uses the expanded, ordered dataset (x_1, x_2) to train an improved model. The entire process repeats itself in this manner until

every piece of data has been transmitted (see Algorithm 2 for a high-level overview). The overall prequential code length is simply the sum of all the transmission costs:

$$L_{\text{preq}}(p_M, \mathcal{D}) = -\sum_{i=0}^{N-1} \log_2(p_M(x_{i+1} \mid \mathcal{D}_i))$$

where $p_M(x_{i+1} \mid \mathcal{D}_i)$ is the probability that the model assigns to x_{i+1} when provided with the ordered dataset $\mathcal{D}_i = (x_1, x_2, \ldots x_i)$. In this notation $\mathcal{D}_0$ represents an empty dataset and, as the initial transmission uses the uniform distribution, $p_M(x_1 \mid \mathcal{D}_0)$ is defined as $\frac{1}{K}$. The specific details of how the model uses a given dataset are provided by its high-level description; critically, this is the only dataset available for both training and prediction.

When using prequential coding, there is an expectation that the model will improve with the expanding dataset, allowing `S` to send progressively shorter codes and consequently reducing the overall code length. Note that the size of the model description is not included in the calculation of $L_{\text{preq}}(\mathcal{M}, \mathcal{D})$. As mentioned, such high-level information is typically considerably smaller than the cost of transmitting the data and is therefore omitted.

The prequential template presented readily admits modifications to suit a variety of settings. For example, consider a model which has some *a priori* bias towards the data. In this scenario it makes sense to replace the initial uniform prior with the actual model prediction. In fact, in general it could be argued that this change creates a more natural definition of the prequential code length. Beyond model-based priors, any prior can be used as long as it has a low communication cost - a fact that is implicit in the uniform choice.

In Section 3.3.4 we will present a common modification known as block transmission. The only difference between the original algorithm and the block transmission variant is that data is sent in sequential bundles rather than individually. As we shall see, this change is vital for the practical usage of prequential coding.

We can also utilise prequential coding in a variety of learning settings. For example, it might seem best suited for autoregressive sequential prediction (*e.g.* a transformer performing text prediction), but it can easily be programmed for tasks such as supervised learning where each item of data consists of both a set of features and a label. In this latter case we might assume, as in [16], that the receiver already has access to a dataset of features in an identical order to the sender's dataset. Under this assumption the sender transmits label data. Therefore, a logical alteration to the prediction process is to condition on the corresponding feature data in addition to the available label data.

```
S initialises a model M_S.
R initialises an identical model M_R.
for i := 0 to N − 1 {
        M_S is trained on D_i.
        M_R is trained on D_i in an identical manner.
        M_S generates a probability for x_{i+1}.
        S encodes x_{i+1} according to this prediction.
        S transmits the code for x_{i+1} to R.
        M_R generates an identical probability for x_{i+1}.
        R uses this prediction to decode the transmission for x_{i+1}.
}
```

Algorithm 2: High-level overview of the non-batched prequential coding process.

3.3.2 Implementation

The overview given in Section 3.3.1 provides a pedagogically focused account of the prequential approach. However, the coding process differs subtly from this description when it is implemented in practice. To understand why, first note that if S actually transmitted each item of data incrementally using Huffman codes, there would be up to a one bit overhead per transmission. This extra cost is incurred because the Huffman code must have a positive integer length. The immediate communication of data between S and R is also antithetical to the reasonable desire to send the entire compressed dataset in one go.

To solve theses issues, practical implementations of prequential coding do not send data to a receiver as soon as the data has been assigned a probability by a model. Instead, each time a prediction is made by the sender's model, the output $p_M(x_{i+1} \mid \mathcal{D}_i)$ is stored locally in a special format known as an arithmetic code. In this encoding stage, there are no transmissions to the receiver and all computation is performed by the sender. Once the final piece of data has been encoded, the arithmetic code and the high-level information (model description, minor schematic details, etc.) are combined to form a self-extracting archive. This archive is then passed to the receiver which recovers the original data in a manner similar to Section 3.3.1. The essential difference between this decoding stage and the previous exposition is that the receiver queries the arithmetic code for information as the sender has finished all computation.

In order to further explain why this approach is used in practice, we will briefly delve into arithmetic codes and their inherent suitability for the prequential scheme. Arithmetic coding arose from the cumulative effort of multiple researchers broadly studying the compression of message streams [99, 125, 130, 131]. Unlike Huffman coding, arithmetic coding does not require at least one bit of information for each item of data. Conceptually, this is accomplished by first assigning a subinterval $[l, h) \subseteq [0, 1)$ to the dataset as specified in Algorithm 3. Note that a finite binary string $b_1 b_2 \ldots b_k$ can be interpreted as a subinterval

of $[0, 1)$ in the following manner:

- The lower bound is the string prefaced by a decimal (e.g. $.b_1 b_2 \dots b_k$).
- If at least one bit in the string is a zero, the upper bound is the subsequent string in the length-increasing lexicographic ordering, also prefaced by a decimal.
- If every bit is a one, the upper bound is one.

The dataset is encoded with a binary string which, using the above interpretation, lies within the interval determined by Algorithm 3. There is always a binary string that meets this criterion and whose length is within two bits of $-\log_2(h - l)$ [112].

```
l := 0.0
h := 1.0
for i := 1 to N {
        u := 0.0
        for j := 1 to x_i - 1 {
                u := u + p_M(j | D_{i-1})
        }
        r := h - l
        l := l + r · u
        h := l + r · p_M(x_i | D_{i-1})
}
return [l, h)
```

Algorithm 3: Method to assign an interval $[l, h)$ to the ordered dataset $\mathcal{D}$ during the arithmetic coding process (adapted from [112]).

The encoding process is simple: the sender processes the data one unit at a time, only adding a bit to the code when the corresponding interval still contains the entire data stream interval. On the final piece of data, the minimum number of bits are appended to ensure that the data stream interval contains the code interval. The decoding process is also relatively straightforward: the receiver extracts the data sequentially by comparing the current model's predictions with the code. When enough bits have been read from the start of the code to unambiguously decipher the subsequent class (i.e. when the corresponding interval is contained by the data stream interval), this item of data is added to the data stream and the prediction model is updated. Note that this process behaves in a First In, First Out (FIFO) queue-like manner. This is desirable for prequential coding, however this stands in contrast to the bits-back method which needs a Last In, First Out (LIFO) stack-like behaviour for which arithmetic coding is not suitable (see Section 3.2.1).

Both the ability to process data incrementally and the roughly ideal code length make arithmetic coding a natural fit for the prequential approach. However, the conceptual outline of arithmetic codes that we have presented contains a fatal flaw - if implemented

as described, it will rely on floating-point arithmetic. Common floating-point standards are only able to encode small messages as they quickly run out of the precision required to specify narrow intervals. Fortunately, this issue can be resolved through the clever use of finite-precision integer arithmetic which, in practice, gives very good performance [170].

3.3.3 Information Theoretic Perspective

In order to study prequential coding from an information theoretic perspective, we will view a given dataset $\mathcal{D} = (x_1, x_2, \dots x_N)$ as a stochastic sample. Each x_i is itself an outcome of a discrete random variable X_i which represents the i^{th} data member and assigns probabilities over the alphabet $\mathcal{A} = \{a_1, a_2, \dots a_K\}$. No assumptions are made about the dependencies between $X_1, X_2, \dots X_N$, and their joint distribution defines the dataset random variable D over $\mathcal{A}^N$. Note that every actual dataset $\mathcal{D}$ is an outcome of D.

In classic information theory the expected information content of such a stochastic variable is the Shannon entropy $H(D)$ or, equivalently, the joint Shannon entropy $H(X_1, X_2, \dots X_N)$. We can easily see from the proof of Theorem 3.3.1 that the expected information content of a dataset variable is the same value as the expected prequential code length of that dataset if the predictions use the true distribution. This proof purposely uses the chain rule for entropy, as it demonstrates that the expected size of each transmission in the prequential coding scheme for the true model can be interpreted as the relative entropy of the data to be transmitted given all of the data that has already been sent.

Theorem 3.3.1. *The entropy of a stochastic dataset is equal to the expected prequential code length when encoded with the true model.*

Proof.

$$
\begin{aligned}
H(D) &= H(X_1, X_2, \dots X_N) \\
&= \sum_{i=0}^{N-1} H(X_{i+1} \mid X_1, X_2, \dots X_i) \\
&= -\sum_{i=0}^{N-1} \sum_{\mathcal{D}_{i+1} \in \mathcal{K}^{i+1}} p(\mathcal{D}_{i+1}) \cdot \log_2(p(x_{i+1} \mid \mathcal{D}_i)) \\
&= -\sum_{i=0}^{N-1} \mathbb{E}[\log_2(p(x_{i+1} \mid \mathcal{D}_i))] \\
&= \mathbb{E}\left[-\sum_{i=0}^{N-1} \log_2(p(x_{i+1} \mid \mathcal{D}_i))\right] \\
&= \mathbb{E}[L_{\text{preq}}(D)]
\end{aligned}
$$

□

Of course, when computing prequential codes in practice, we rarely have access to the true model for the necessary marginal or conditional probabilities. Instead we rely on an alternative model that supplies theorised values for the requisite prequential probabilities. These predictions define a joint probability distribution over $\mathcal{K}^N$ which we will refer to as q; the true probability distribution will be denoted as p. As demonstrated in the proof of Theorem 3.3.2, the penalty added to the expected prequential code length for using an alternative model is $\mathrm{KL}[p \parallel q]$.

Theorem 3.3.2. *The expected prequential code length of a stochastic dataset encoded with a hypothesised model is equal to sum of the entropy of that dataset and the Kullback–Leibler divergence between the true distribution and the hypothesised distribution.*

Proof.

$$\begin{aligned}
\mathbb{E}[L_{\mathrm{preq}}(q, D)] &= -\sum_{\mathcal{D}\in\mathcal{K}^N} p(\mathcal{D}) \cdot \sum_{i=0}^{N-1} \log_2(q(x_{i+1} \mid \mathcal{D}_i)) \\
&= -\sum_{\mathcal{D}\in\mathcal{K}^N} p(\mathcal{D}) \cdot \log_2(q(\mathcal{D})) \\
&= -\sum_{\mathcal{D}\in\mathcal{K}^N} p(\mathcal{D}) \cdot \left(\log_2(p(\mathcal{D})) + \log_2\left(\frac{q(\mathcal{D})}{p(\mathcal{D})}\right)\right) \\
&= H(p) + \mathrm{KL}[p \parallel q]
\end{aligned}$$

□

Theorem 3.3.2 should have a familiar feel to those with a background in machine learning. The sum of the entropy of the true distribution and KL divergence between the true distribution and the theorised distribution is commonly known as the cross entropy. The cross entropy of q relative to p represents the expected code length of an outcome when encoded with q, but drawn according to p. In machine learning, cross entropy is often used as a loss function in classification problems where p is fixed, as the minimum value can only be reached when q is equal to p. In this scenario, the cross entropy minimisation problem is equivalent to using the KL divergence as a loss function.

3.3.4 Block Transmissions

At each iteration of the prequential coding process the model trains on the available data and then predicts the subsequent data. This computation incurs a time cost that can become quite significant, especially when using high-parameter learning models such as deep neural networks. Additionally, as the coding procedure progresses the time burden per iteration can increase; for example, consider a training regimen that specifies its termination criterion in epochs.

In order to reduce lengthy coding times, data can be sent in blocks rather than as a single unit. These blocks maintain the order of the data internally and are transmitted sequentially. More formally, consider a sequence of $W+1$ integers $0 = t_0 < t_1 < t_2 < \cdots < t_W = N$ which we will refer to as the transmission indices. The transmission blocks are simply the contiguous data sequences demarcated by the transmission indices (i.e. $x_{t_0+1:t_1}, x_{t_1+1:t_2}, \ldots x_{t_{W-1}+1:t_W}$). Under the block model, the prequential code length is defined as:

$$L_{\text{preq}}(\mathcal{M}, \mathcal{D}) = -\sum_{i=0}^{W-1} \log_2\big(p_M\big(x_{t_i+1:t_{i+1}} \mid \mathcal{D}_{t_i}\big)\big).$$

To understand why block transmission can markedly reduce coding times for certain learning schemes, consider the following scenario: say that at each iteration in the coding process the model is trained exactly once on each item of data in the available dataset. In order to avoid a bias towards the early data, the model is reset to its initial state after each transmission. There are, of course, many ways to avoid such a bias - this is simply one of the more straightforward techniques. Such a reset also ensures that we are evaluating the original training regimen and not a prequential, transfer learning mashup. By the end of the standard prequential process the model will have processed $\mathcal{O}(N^2)$ units of data.

Now say that the sender chooses to use block transmissions with exponentially increasing block sizes. More precisely, let $W = \lceil \log_2(N+1) \rceil$ and for $j = 1, 2, \ldots W-1$ set $t_j = \sum_{i=0}^{j-1} 2^i$. Under this setting, the total amount of data processed by the prequential coding scheme is $\mathcal{O}(N)$. Considering the enormous size of many datasets, without the drop from quadratic to linear time, computing a prequential code for sophisticated models would often be practically infeasible. Many recent papers on prequential coding and neural networks have used block sizes that follow some sort of exponential increase [16, 18, 86].

Sending data in blocks instead of one at a time will generally lead to an increase in prequential code length. Transmitting incrementally gives the model the opportunity to extract information from as much data as possible in the prequential sense. Under the block model, earlier data within a block does not have the chance to inform the model before later data within that same block is encoded. This drop in performance is often mitigated by the diminishing returns that increased dataset sizes provide to a model. Still, the trade-off between computation time and prequential code length is an important area of research for practical concerns. Recently, in a paper that studied the prequential approach to the minimum description length principle, Bornschein et al. [19] found that time efficient methods based on continuous learning produced better code lengths in many scenarios.

3.3.5 Prequential Success

Why the fuss over prequential coding, especially in comparison to other methods of model and data transmission? As we shall see, the same feature of deep learning that has led to its widespread success across a variety of domains also begets incredible prequential efficiency - the ability of deep neural networks to generalise well to unseen data. This connection between deep learning and prequential coding has a largely recent history.

In a landmark 2018 paper, Blier and Ollivier determined that prequential coding was an extremely effective method for generating short description lengths of deep neural networks [16]. More specifically, they studied the description lengths of deep neural networks performing image classification on the well-known MNIST [100] and CIFAR-10 [98] datasets. Prequential coding outperformed every alternative technique tested, including network compression strategies [68, 175] and variational coding [63, 74]. Network compression strategies are essentially two-part codes where the network parameters are transmitted separately from the encoding of the data according to the model. In the naive case the parameters are simply passed on as floating points typically at a cost of four bytes per parameter. The network compression strategies tested by Blier and Ollivier were significantly more complex and efficient than the naive strategy, yet still produced code lengths that were up to several orders of magnitude more than the prequential length. Variational inference fared better, but was still inferior by approximate factors of two to five. This result was especially surprising as the variational inference approach was purpose-built to minimise description lengths.

In the years since Blier and Ollivier's work, many researchers have used prequential coding as their preferred tool for generating minimal description lengths when employing complex models and, in particular, deep neural networks. Yogatama et al. leveraged prequential code length as an evaluation metric to gauge the ability of a model to generalise rapidly [180]. As part of an investigation into the inductive biases of pre-trained neural networks, Lovering et al. studied the extractibility of linguistic features through experiments that measured prequential code lengths [111]. Perez et al. used prequential coding in the MDL estimation component of their novel dataset probing method, "Rissanen Data Analysis" [127]. Because Perez et al. were primarily interested in the data itself, they made use of an ensemble of models in order to generate shorter codes. Work by Bornschein et al. [19], which we previously mentioned in Section 3.3.4, found that the block transmission approach taken by [16] in order to mitigate computational costs, could be improved upon using inspiration from continuous learning.

While the popularity of the prequential coding paradigm is certainly a testament to its usefulness, its achievements in premiere data compression competitions arguably provide even more compelling evidence of its ability. In an earlier section (see Section 3.1.1.2) we introduced the Large Text Compression Benchmark run by Matt Mahoney [115]. In

more detail, the LTCB is a competition to compress one gigabyte of text from the English language version of Wikipedia as it appeared on March 3, 2006. The dataset, which is known as `enwik9`, is formatted as UTF-8 encoded XML and contains nearly 250,000 articles, of which approximately one-third are merely redirects to fix broken links. Entries to the competition are ranked by the combined size of the compressed `enwik9` file and a zipped folder containing the decompressor and any accessories.

As of early 2024, the best total size submitted to the competition is 106,632,363 bytes, achieved by version 3.2 of Fabrice Bellard's `nncp` algorithm [12, 13]. `nncp` is essentially a prequential approach with extra trimmings: given a neural network, the receiver learns on some already transmitted text, the sender transmits the true text at the canonical cost, then the process repeats itself using the updated dataset. Early iterations of `nncp` use a deep neural network (either an LSTM [79] or a transformer [163]) and an arithmetic coder to form a prequential code over a preprocessed version of the dataset [12]. Documented later versions exclusively use transformers and employ minor implementation tweaks to improve performance[†] [13]. The LTCB is the gold-standard for text compression benchmarks with no restrictions on compute; the fact that `nncp` outperforms the competition, including many highly engineered task-specific algorithms, is an important vindication of the prequential approach.

3.4 Generalising Prequential Coding

Despite the success of prequential coding, it is not the be-all and end-all approach to compression with neural networks. Many common network architectures do not have the requisite structure to form prequential predictions. Additionally, the datasets themselves might not be ordered in a manner particularly conducive to learning. Intuitively, the learning process is progressive, not prequential. If a piano student were to learn to play a concerto from zero-knowledge, it would be highly ineffective to teach them from the concerto itself, starting with the early notes and moving on sequentially. Rather, they would be better served if they were first taught basic music theory, then a few easier pieces, before eventually progressing to the concerto.

3.4.1 An Instructional Approach

We will capture this intuition by creating a generalisation of the block-transmission approach to prequential coding, using the same setting that was described in Section 3.3.1. A sender, `S`, would like to send a model $M \in \mathcal{M}$ trained on a dataset $\mathcal{D}$ to a receiver, `R`, or alternatively `S` would like to send $\mathcal{D}$ to `R` and is constrained to the model class $\mathcal{M}$. As

[†]Up-to-date information about `nncp` is available at `https://bellard.org/nncp/` along with source code.

before, the dataset is finite with size N and each unit of data belongs to a finite set $\mathcal{K}$ compromised of K unique classes.

S first transmits a description of the model and a batch of data of length t_1 to R. This data is the output of some function, $f_1 : \mathcal{K}^N \to \mathcal{K}^{t_1}$ whose input is the original dataset. If we assume this initial prediction is equivalent to a standard uniform encoding the transmission costs $t_1 \log_2 K$ bits along with the size of the model description.

Subsequently, the transmitted data, $f_1(\mathcal{D})$, is used by R to train the model. S trains an identical model by using the same training regimen and restricting the training data to the batch sent to R. Note that $f_1(\mathcal{D})$ is not necessarily a subset of the original dataset; there are no restrictions placed on f_1 other than the domain and range.

S uses the freshly trained model to encode and transmit a new batch of data (potentially via an arithmetic code) which is then decoded by R using its identical model. This new batch of data consists of t_2 individual units and is the output of some function, $f_2 : \mathcal{K}^N \to \mathcal{K}^{t_2}$, whose input is also the original dataset. The trained model has developed its predictions for the new batch of data, $f_2(\mathcal{D})$, using only the original batch of data given by $f_1(\mathcal{D})$ (via training or possibly even directly).

Now both R and S can continue to train identical models, but with access to an expanded dataset that includes the outputs of both f_1 and f_2 over $\mathcal{D}$. This process of training identical models to predict subsequent outputs of functions whose inputs are the original dataset carries on for W iterations, until the following dataset has been transmitted:

$$\mathcal{F}_\mathcal{D} = \{f_1(\mathcal{D}), f_2(\mathcal{D}), \dots f_W(\mathcal{D})\}.$$

We will refer to $f_1, f_2, \dots f_W$ as the instruction functions, $f_1(\mathcal{D}), f_2(\mathcal{D}), \dots f_W(\mathcal{D})$ as the instruction data, and $\mathcal{F}_\mathcal{D}$ as the instruction dataset. The original dataset $\mathcal{D}$ must be recoverable from the instruction dataset $\mathcal{F}_\mathcal{D}$ (i.e. there exists some recovery function η such that $\eta(\mathcal{F}_\mathcal{D}) = \mathcal{D}$). Furthermore, η must have a known, short description length.

In general, this function should be fairly trivial; for instance, one might require that $f_W(\mathcal{D}) = \mathcal{D}$, and therefore the recovery function simply observes the final transmission. As with the description length of the model, the description length of the recovery function is considered negligible and the instructional code length for the given instruction dataset is defined as:

$$L_{\text{inst}}(\mathcal{M}, \mathcal{D}) = -\sum_{i=0}^{W-1} \log_2 p_{M_{f_i(\mathcal{D})}}(f_{i+1}(\mathcal{D}))$$

where $p_{M_{f_i(\mathcal{D})}}$ represents a model trained according to a given regimen on some subset of the first i transmissions, $f_1(\mathcal{D})$ through $f_i(\mathcal{D})$, and $p_{M_{f_0(\mathcal{D})}}$ denotes the untrained model. A full high-level overview of the instructional process is given in Algorithm 4.

At first it may seem counter-intuitive to transmit data that might not belong to the original dataset, $\mathcal{D}$. In the prequential scheme, all of the data sent is unaltered and the

S initialises a model $M_{\mathtt{S}}$.
R initialises an identical model $M_{\mathtt{R}}$.
for i := 0 to $W-1$ {
    $M_{\mathtt{S}}$ is trained on some subset of $\{f_1(\mathcal{D}), \dots f_i(\mathcal{D})\}$.
    $M_{\mathtt{R}}$ is identically trained on the same subset of $\{f_1(\mathcal{D}), \dots f_i(\mathcal{D})\}$.
    $M_{\mathtt{S}}$ generates a prediction for $f_{i+1}(\mathcal{D})$.
    S encodes $f_{i+1}(\mathcal{D})$ according to this prediction.
    S transmits the code for $f_{i+1}(\mathcal{D})$ to R.
    $M_{\mathtt{R}}$ generates an identical prediction for $f_{i+1}(\mathcal{D})$.
    R uses this prediction to decode the transmission for $f_{i+1}(\mathcal{D})$.
}
R recovers $\mathcal{D}$ using η and $\mathcal{F}_{\mathcal{D}}$.

Algorithm 4: High-level overview of the generic instructional coding process.

model is trained solely on subsets of $\mathcal{D}$. However, the cost of the seemingly extraneous instructional data can be outweighed by the improvement in performance granted by controlling the data which the model sees and predicts.

In some sense, the instructional coding process is analogous to a teacher leading a student to understand a complex topic or concept. At first, the teacher might oversimplify the original topic, then ask the student to make the next conceptual leap that is within their grasp. If the lessons or instructions provided by the teacher are suitable, the student should be able to eventually arrive at the full complex form with relative ease.

Instructional coding can be viewed as a generalisation of prequential coding by letting the instructional functions represent progressive views on $\mathcal{D}$. More specifically, if we define $f_{i+1}(\mathcal{D}) = x_{t_i+1:t_{i+1}}$ and set the recovery function to concatenate the transmitted data, we have perfectly described the batched prequential scheme (batch sizes can be set to one in order to recover the non-batched version). Therefore, all prequential coding schemes are instructional coding schemes, and the prequential code length is never better than the minimal instructional code length.

Image Super-Resolution Example The methodology of instructional coding and its benefits over prequential coding are perhaps best conveyed by a simple real-world example. Consider a dataset, $\mathcal{D}$, which consists of a single image, X, with a total size of N pixels. Each pixel belongs to one of K classes, and we denote the i^{th} pixel in raster order as x_i. In our demonstration, we will use this form of dataset in conjunction with deep neural networks in order to increase the resolution of the given image - a challenging problem known as super resolution [3]. In order to avoid any immaterial complexities, we will select an image that is 8-bit grayscale ($K = 256$) with a dimension of 512×512 pixels ($N = 512^2$) (see Figure 3.1). Additionally, we will assume that our super resolution neural network doubles both the width and the height of any image it processes. For the runs

discussed below we use a simple neural network comprised of four convolutional layers with a middle upscaling interpolation layer.

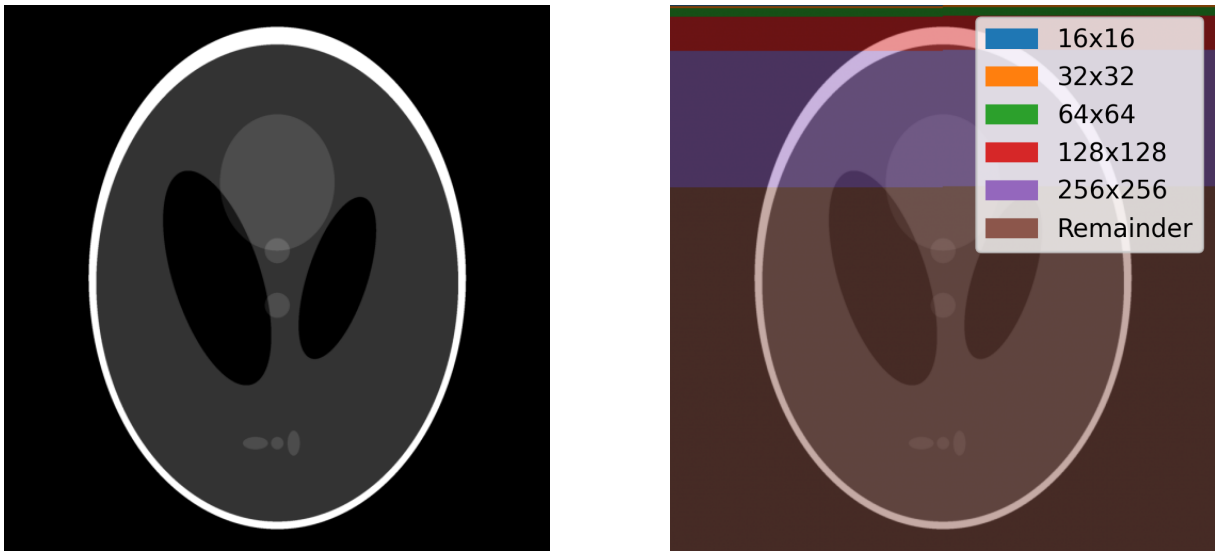


Figure 3.1: (Left) The baseline grayscale image of the Shepp-Logan phantom rendered with a high contrast [143, 159]. (Right) An exponentially increasing batched prequential division of the original image in raster order.

Initially, S sends R two down-scaled versions of the original image: one at 16×16 pixels and the other at 32×32 pixels. In other words, our initial instruction function is defined as $f_1(\mathcal{D}) = (X_{16}, X_{32})$ where X_k is a $k \times k$ version of the original image, resized according to a standard interpolation method. The cost of this initial transmission is $\left(16^2 + 32^2\right) \cdot 8 = 10,240$ bits, as we use a basic uniform encoding over the $K = 256$ different possibilities for each pixel.

S now trains its copy of the neural network using X_{16} as the input and X_{32} as the output. The model is validated after each epoch by feeding X_{32} as the input and calculating the code length to encode X_{64} using the model's output. After training for 50 epochs using the standard Adam optimiser [89], the epoch index with the lowest code length is sent to R at a small cost of a byte along with the corresponding code for X_{32}.

This is conceptually equivalent to defining our second instruction function as $f_2(\mathcal{D}) = X_{32}$. The cost of this transmission is $-\log_2 p_{\theta_{f_1(\mathcal{D})}}(f_2(\mathcal{D}))$ which, assuming our neural network is learning, should come at a cheaper per pixel cost than the first transmission. We have not included the cost of encoding the best epoch index as this is negligible.

R can now train its copy of the neural network in an identical manner using X_8, X_{16} and the stopping index. Once training is complete, R can decode the transmission from S and recover X_{32}. Note that at this point in the process, R has access to X_8, X_{16}, and X_{32}.

We now define three more instruction functions for the instructional coding scheme: $f_3(\mathcal{D}) = X_{64}$, $f_4(\mathcal{D}) = X_{128}$, and $f_5(\mathcal{D}) = X$. As before, S encodes each image by training its neural network on the two largest images that R currently possesses, then validates on

the next image that will be sent, saving the best epoch index. This information is sent over, `R` trains an identical network, and then decodes the new, larger image. The final image sent over is the original image, thus completing the process.

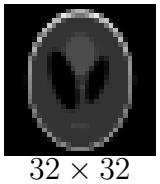
Training	**Pred. Input**	**Pred. Output**	**Truth**	**Cost**
Input: 16×16 Label: 32×32	 32×32	 64×64	 64×64	≈ 10.7 kilobits
Input: 32×32 Label: 64×64	 64×64	 128×128	 128×128	≈ 18.3 kilobits
Input: 64×64 Label: 128×128	 128×128	 256×256	 256×256	≈ 34 kilobits
Input: 128×128 Label: 256×256	 256×256	 512×512	 512×512	≈ 69.3 kilobits

Table 3.1: Instructional coding example for an image compression task where the intermediate tasks are to sequentially upscale the image.

For pedagogical purposes, an overview of the process for a specific run can be found in Table 3.1. The total cost of transmitting the 512×512 Shepp-Logan image in this run, without including small overheads, is $142,479$ bits - over one order of magnitude smaller than the original image and comparable to the size of the PNG version, despite using an extremely simplistic neural network. Now let us compare this result to batch prequential coding using the raster order to arrange the pixels in a sequence (see Fig. 3.1 for an image of the prequential division). We will use identical batch sizes to the instructional transmissions, although the final transmission will have to be padded with zeros in order to reach a dimension of 512×512 pixels and remain compatible with a 256×256 input image. We will not include the transmission cost of the padding in the prequential code length as it has a small description complexity.

Note that even though we are now assuming the data is sequentially ordered, each transmission will have to be rearranged back into a square in order to maintain compatibility with our neural network - these 'images' will be referred to as $Y_{16}, Y_{32}, Y_{64}, Y_{128}, Y_{256}$, and Y_{512}. The transmission cost of Y_{16} and Y_{32} remains the same at $10,240$ bits as we continue

to use a uniform encoding for the initial transmission. However, in a single demonstration run, we observed the following costs: $Y_{64} \approx 0$ kilobits (the images were blank at this stage), $Y_{128} \approx 111.2$ kilobits, $Y_{256} \approx 341.2$ kilobits, and $Y_{512} \approx 303$ kilobits (this final cost is for the non-padded pixels in Y_{512}). The total cost of the prequential-style transmission is $765,648$ bits, substantially larger than the instructional code length. This huge discrepancy is because the neural network cannot infer much about subsequent batches of information (visually they appear very different). Note of course that as these were single runs these results do not constitute a proper experiment but rather serve to hone intuition about scenarios where basic prequential coding is clearly a flawed methodology.

Although we have described instructional coding and shown it to be a generalisation of prequential coding, the literature is filled with compression algorithms which one might consider instances of instructional coding. The instructional approach outlined in the image super resolution example is similar in spirit to a simplified version of the super resolution compression algorithm `SReC` [23]. Beyond super resolution, algorithms such as generative diffusion [145], though not directly instructional coding schemes, also work on the concept of incremental clarity and recent work by Theis et al. [156] has investigated diffusion as a viable compression model. We will explore diffusion further in Chapter 4 as the setting for our paper 'Importance-Guided Diffusion'.

3.5 Prechastic Coding

In Section 3.4 we introduced instructional coding as a generalisation of batched prequential coding and noted that instructional coding can outperform prequential coding in select scenarios. However, in general the problem of determining instruction functions is highly non-trivial and, at first glance, seems to require bespoke solution methods based on the learning model and the structure of the data.

In this section we ask the following question: is it possible to create a generic instructional coding scheme that works well on a variety of architectures and can compete with prequential coding on rudimentary supervised learning tasks? As it turns out this question can be answered in the affirmative by adjusting what conceptually constitutes training data. Rather than defining intermediate datasets as cumulative, sequential partitions (as in the prequential approach) or even as general subsets of the original dataset, highly efficient codes can be generated by *allowing fake labels* at progressively diminishing rates. In this setting the given learning model is used to iteratively denoise the stochastic intermediate data under guidance until the original dataset is recovered. We shall refer to this approach as *prechastic coding*, named, in the spirit of prequential, as a portmanteau of predictive and stochastic. The contents of this section present the paper "Prechastic Coding: An Alternative Approach to Neural Network Description Lengths" by myself and Pietro Liò

which appeared at the *Workshop on Machine Learning and Compression* at the 2024 edition of *Neural Information Processing Systems*. Much of the content remains either similar to or as it appears in the paper, with modifications and edits largely made to conform to the thesis framework.

3.5.1 Overview

Although the preceding description should give an intuitive understanding of the prechastic approach, we shall now provide a more formal definition and tie up some loose ambiguities. Although by no means a requirement, for pedagogical purposes (and in line with Blier and Ollivier [16]) we shall use the following standard supervised learning scenario: a sender and a receiver both have access to a sequence of N inputs $x_{1:N}$ and their own identical copies of some agreed upon learning model. The ordering for the input data is random, but fixed and identical for both entities; for example, $x_{1:N}$ could be a shuffled set of images. Each x_i has a corresponding label $y_i \in \mathcal{K} \coloneqq \{1, 2, \dots K\}$ which, initially, only the sender has access to. The sender would like to transmit $y_{1:N}$ using as little information as possible.

With the supervised learning scenario established, now consider a sequence of distributions $Q_1, Q_2, \dots Q_T$ over the product space $\mathcal{K}^N$. Each Q_i assigns probabilities to all K^N possible permutations of labels, true or false, for the dataset $x_{1:N}$. Note of course that $y_{1:N}$ is the only completely true permutation. Samples from these distributions constitute intermediate datasets for the selected model to learn from and, consequently, we shall refer to $Q_1, Q_2, \dots Q_T$ as the *guiding distributions*. As the sender also knows the true labels $y_{1:N}$ it uses this information to compute efficient guiding distributions. We shall discuss the specifics of selecting $Q_1, Q_2, \dots Q_T$ in Section 3.5.3; for now, simply note that the receiver does not have access to the guiding distributions throughout the coding process.

To initialize the prechastic approach, the receiver uses its untrained model to create predictions for each input from $x_{1:N}$. The receiver then compiles the resulting outputs into a probability distribution P_1 over the product space $\mathcal{K}^N$. Due to the lack of training, P_1 is likely to be roughly equivalent to a uniform distribution over all K^N possible permutations. The sender, who has also computed an identical copy of P_1, subsequently transmits a sample $\boldsymbol{q_1} \sim Q_1$ using $\mathcal{O}(\mathrm{KL}[Q_1 \parallel P_1])$ bits (the machinery of this is discussed in Section 3.5.2). The sample $\boldsymbol{q_1}$ is an n-length vector of noisy labels which correspond to the input data $\boldsymbol{x_{1:N}}$. The receiver uses these noisy labels to train the model and compute an updated set of predictions P_2 for the true labels.

This entire process is repeated for a further $T - 1$ iterations. At each step the sender transmits some sample $\boldsymbol{q_i} \sim Q_i$ at a cost of $\mathcal{O}(\mathrm{KL}[Q_i \parallel P_i])$ bits which the receiver uses to train their model and form an updated set of predictions P_{i+1} for the original data labels. Figure 3.2 illustrates the prechastic process for a given iteration and contrasts it with a prequential iteration. After the final set of predictions P_{T+1} is computed, the

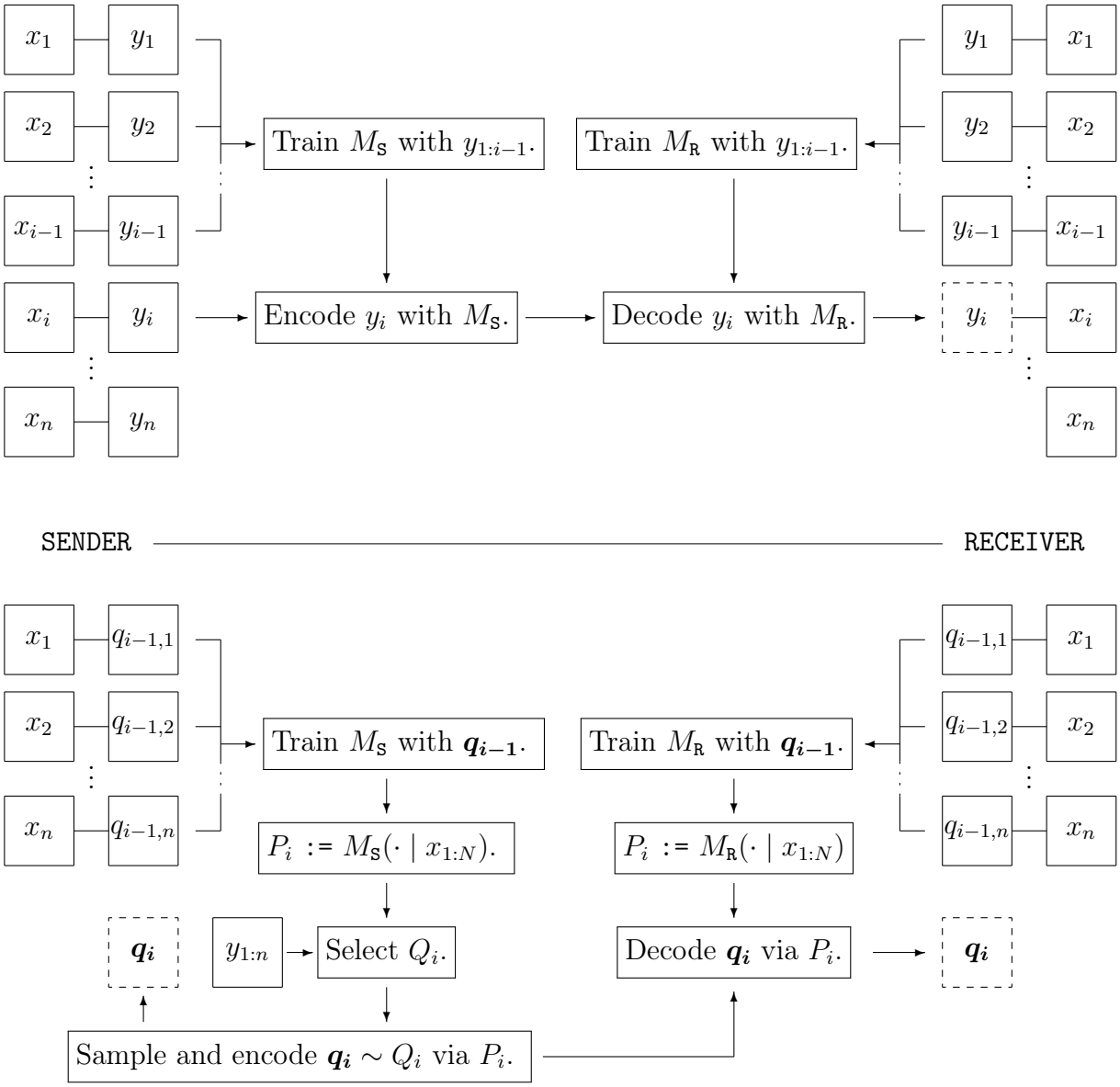


Figure 3.2: High level illustrations of an iteration of prequential coding process (top) and prechastic coding (bottom). In later descriptions of the general prechastic algorithm, (such as Algorithm 5) training is considered to occur at the end of the previous iteration. Note that the models are trained identically, *i.e.* $M_{\mathtt{S}} \equiv M_{\mathtt{R}}$.

true labels $\boldsymbol{y_{1:N}}$ are losslessly communicated using an entropy coding technique such as arithmetic coding [170] or asymmetric numeral systems [44]. Thus, the total amount of external data delivered by the sender, *i.e.* length of the prechastic code, is approximately $-\log_2(P_{T+1}(\boldsymbol{y_{1:N}})) + \sum_{i=1}^{T} \mathrm{KL}[Q_i \parallel P_i]$ bits. For a high-level overview of the prechastic coding routine, please see Algorithm 5; experimental results on real datasets for various prechastic flavours will be given in Section 3.5.4

S initialises a model $M_{\mathtt{S}}$.
R initialises an identical model $M_{\mathtt{R}}$.
for i := 1 to T {
    P_i := $M(\cdot \mid x_{1:N})$ where $M \equiv M_{\mathtt{R}} \equiv M_{\mathtt{S}}$.
    S generates a sample $\boldsymbol{q_i} \sim Q_i$.
    S transmits $\mathcal{O}(\mathrm{KL}[Q_i \parallel P_i])$ bits which enable R to recreate $\boldsymbol{q_i}$.
    $M_{\mathtt{R}}$ is trained on $\boldsymbol{q_i}$.
    $M_{\mathtt{S}}$ is trained on $\boldsymbol{q_i}$ in an identical manner.
}
S encodes $y_{1:N}$ using $M_{\mathtt{S}}(y_{1:N} \mid x_{1:N})$.
S transmits the code for $y_{1:N}$ to R.
R decodes $y_{1:N}$ using $M_{\mathtt{R}}(y_{1:N} \mid x_{1:N})$.

Algorithm 5: High-level overview of the basic prechastic coding process. In practice, the sender simulates the entire process in order to send one single transmission.

Note that while most deep learning models have the capacity to train to very low levels of error on the noisy label set, training regimens are typically designed for performance on unseen data. Assuming the goal of training is generalisation (*e.g.* through regularisation techniques such as early stopping) then, in general, as long as the guiding distributions are selected appropriately, each P_{i+1} should improve on P_i in expectation over Q_i.

Relation to Instructional Paradigm It is relatively straightforward to show that the general prechastic coding scheme as described in Algorithm 5 is a form of instructional coding. Each intermediate datasets is given by the original inputs $x_{1:n}$ and the corresponding labels in the sample $\boldsymbol{q_i}$. Furthermore, the prediction for the subsequent dataset is simply P_i.

3.5.2 Relative Entropy Coding

In order to create a practical prechastic coding algorithm, we must take a detour to discuss the necessary machinery for the following scenario: say that a sender S would like to communicate a sample from the distribution q to a receiver R; importantly, S does not care which particular sample is communicated. Additionally, both S and R have agreed to a prior distribution p, which R can use to decode the information transmitted by S into a sample from q.

Recall that the relative entropy or Kullback-Leibler (KL) divergence between two distributions q and p is

$$\mathrm{KL}[q \parallel p] \coloneqq \mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p(x)}\right].$$

In this subsection, the relative entropy is given in nats; however, in later sections we shall return to bits. The *relative entropy coding problem* [48, 49] asks, "How can one communicate a sample from q through a uniquely decodable code with an expected code length $\mathcal{O}(\mathrm{KL}[q \parallel p])$?" There are a few assumptions concerning the setting of this question, namely that $\mathrm{KL}[q \parallel p]$ is finite and that both the sender and receiver have access to a public source of randomness (typically implemented through a mutually agreed seed for a pseudo-random number generator). Early work in this area was done by Harsha et al. [69] who proposed a computationally intractable rejection sampling algorithm.

Later on, Havasi et al. [70] developed an improved approach based on importance sampling which does not require managing acceptance probabilities across the entire domain. The procedure works by first generating $M = \lceil \exp(\mathrm{KL}[q \parallel p]) \rceil$ samples $x_1, x_2, \ldots x_M$ from the prior distribution p. Then each x_i is assigned a weight $w_i \coloneqq \frac{q(x_i)}{p(x_i)}$ and an index j is randomly selected based on a categorical distribution of size M, where the probability $p(j)$ is given by $\frac{w_j}{\sum_i w_i}$. This sample index is encoded at a cost of $\log M \approx \mathrm{KL}[q \parallel p]$ nats and is subsequently transmitted to the receiver.

The receiver then uses an identical source of randomness to perfectly recreate the samples $x_1, x_2, \ldots x_M$. In the final step, the index j is decoded and the corresponding $\boldsymbol{x}_j$ is selected. Although this approach is promising, it comes with two caveats: M grows exponentially with the relative entropy, and $\boldsymbol{x}_j$ is actually a sample from an approximation of q. Havasi et al. [70] addressed the latter concern by leveraging a result from Chatterjee and Diaconis [29] to show that $M = \lceil \exp(\mathrm{KL}[q \parallel p]) \rceil$ is a large enough sample size to keep the bias low.

The exponential growth issue was tackled in a paper by Flamich et al. [48] who split the process of transmitting over a sequence of intermediate auxiliary steps. Their technique, which they called index coding, essentially takes advantage of the fact that for some fixed integer $K > 1$, Ke^x grows exponentially slower than e^{Kx}. More precisely, index coding splits the random variable of interest x into a sequence of K auxiliary random variables $a_1, a_2, \ldots a_K$ with corresponding prior distributions $p(a_1), p(a_2), \ldots p(a_k)$. There must exist some function f such that x is recoverable from the auxiliary variables, *i.e.* $f(a_{1:K}) = x$. For a chosen set of prior distributions, one must determine target distributions $q(a_k \mid a_{1:k-1})$ for each auxiliary variable a_k which will subsequently be used to obtain auxiliary samples and recover a sample from the original q.

When choosing a reconstruction function, set of priors, and set of targets, one must

satisfy the marginalisation properties

$$p(x) = \int \delta(f(a_{1:K}) - x) \prod_{k=1}^{K} p(a_k) \, \mathrm{d}a_{1:K}$$
$$q(x) = \int \delta(f(a_{1:K}) - x) \prod_{k=1}^{K} q(a_k \mid a_{1:k-1}) \, \mathrm{d}a_{1:K}$$

where δ is the Dirac delta function. Additionally, in order to avoid adding an overhead cost, these choices should be made so that the relative entropy of the total auxiliary process matches the relative entropy of the original transmission.

Flamich et al. [48] go on to develop auxiliary forms for the multivariate Gaussian prior $\mathcal{N}(\mathbf{0}, \mathbb{I})$, specifically choosing auxiliary priors $p(a_k) = \mathcal{N}(\mathbf{0}, \sigma_k^2 \mathbb{I})$ such that $\sum_{k=1}^{K} \sigma_k^2 = \sigma^2$ and the ratio of each σ_k^2 to $\sigma^2 - \sum_{i=1}^{k-1} \sigma_i^2$ roughly follows the power law $(K + 1 - k)^{-0.79}$. The reconstruction function is simply $f(a_{1:K}) = \sum_{k=1}^{K} a_k$ and the corresponding auxiliary target distributions are determined to also be Gaussians. Several additional techniques are incorporated into the algorithm to enhance its practical performance, including the specification of a tolerance parameter for intermediate relative entropy values and the reduction of bias through a type of beam search.

Our in-depth review of importance sample and index coding was included as they will be of relevance to prechastic coding and our later work in diffusion methods in Chapter 4 (although without much of the trimmings presented after the initial description of the split into auxiliary variables). It is also worth briefly mentioning some more of the expansive literature on topics concerning and related to the relative entropy coding problem. Recent work by Flamich et al. [49] introduced algorithms based on A* sampling [113] that provably achieved an expected code length of $\mathcal{O}(\mathrm{KL}[q \parallel p])$. Critically, they proved that one of their algorithms reaches this code length in $\mathcal{O}(\mathrm{D}_\infty[q \parallel p])$ runtime, where $\mathrm{D}_\infty[q \parallel p]$ is the limit of the Rényi divergence as $\alpha \to \infty$ (they show experimentally that their other algorithms have a similar runtime). This is an important result as the only additional assumption placed on q and p is that they are continuous with unimodal density ratio. In the general case without this assumption, Agustsson and Theis [2] proved that given $RP \neq NP$[†], REC algorithms have an expected runtime that is non-polynomial in $\mathrm{KL}[q \parallel p]$.

Work by Li and Gamal [106] introduced a type of REC algorithm known as Poisson functional representation (PFR) and bounded the expected code length of PFR between $\mathrm{KL}[q \parallel p]_2$[‡] and $\mathrm{KL}[q \parallel p]_2 + \log_2(\mathrm{KL}[q \parallel p]_2 + 1) + \mathcal{O}(1)$; unfortunately, the expected runtime is bounded below by $\mathrm{D}_\infty[q \parallel p]$. Theis and Ahmed [154] proposed Ordered Random Coding (ORC) in the context of the related *reverse channel coding problem* [14], however

[†]RP is the randomised polynomial time complexity class and NP is the nondeterministic polynomial time complexity class. The question of their equivalence is an open problem.

[‡]This notation signifies that the KL divergence is given in bits.

similar to [70], ORC only provides an approximate sample and also has a runtime that is an exponential function of the relative entropy.

Many approaches to compression with neural networks follow the transform coding paradigm [59], mapping the data to a latent space, quantising these latents, and then performing entropy coding [4, 85, 155]. However, these approaches are inherently difficult to train due to the non-differentiable quantisation which can be awkward to overcome [48]. Augmented varieties using techniques such as dithering [2, 5], but as a result introduce new constraints to the process [49]. For further detailed discussion of the full set of options available for the relative entropy coding/reverse channel coding problems, please see [49, 154].

3.5.3 Implementation

With our detour into relative entropy coding complete, we can now discuss the implementation of a complete prechastic coding scheme. Assuming we have a method for selecting the guiding distribution, fitting the importance sampling REC approach of [70] into Algorithm 5 is straightforward. Both the receiver and sender generate the requisite number of samples from P_i identical to each other using a pre-agreed random seed and protocol. Then the sender communicates the index of one of these samples which has been chosen proportionally to the importance ratio ${}^{Q_i(\boldsymbol{p}_i^{(j)})}/_{P_i(\boldsymbol{p}_i^{(j)})}$ for each sample $\boldsymbol{p}_i^{(j)}$ (see Algorithm 6). However, we must still consider the principle drawback of the importance sampling method which is the exponential runtime. As we shall see in the following discussion about guiding distributions, it is reasonable to set the information gap between Q_i and P_i to small increments, diminishing the computational impact from sampling.

3.5.3.1 Convex First Approach

It should be clear from the earlier description of the prechastic approach that the selection of the guiding distributions is a critical determinant of the resulting code length. However, choosing efficient $Q_1, Q_2, \dots Q_T$ is a complex problem that, as in the general instructional case, essentially requires one to design an appropriately challenging curriculum for highly non-trivial learning models. Nevertheless, as a first-pass consider the following convex

S initialises a model $M_{\mathtt{S}}$.
R initialises an identical model $M_{\mathtt{R}}$.
for i := 1 to T {
    P_i := $M(\cdot \mid x_{1:N})$ where $M \equiv M_{\mathtt{R}} \equiv M_{\mathtt{S}}$.
    J := $\lceil 2^{\mathrm{KL}[Q_i \parallel P_i]} \rceil$
    R and S generate identical sets of J samples $\boldsymbol{p_i^{(1)}}, \boldsymbol{p_i^{(2)}}, \ldots, \boldsymbol{p_i^{(J)}} \sim P_i$.
    S computes importance weights $\frac{Q_i(\boldsymbol{p_i^{(j)}})}{P_i(\boldsymbol{p_i^{(j)}})}$ for each $\boldsymbol{p_i^{(j)}}$.
    S samples an index $k \in \{1, 2, \ldots J\}$ according to the importance weights.
    S transmits k to R at a cost of $\mathcal{O}(\mathrm{KL}[Q_i \parallel P_i])$ bits.
    $M_{\mathtt{R}}$ is trained on $\boldsymbol{p_i^{(k)}}$.
    $M_{\mathtt{S}}$ is trained on $\boldsymbol{p_i^{(k)}}$ in an identical manner.
}
S encodes $y_{1:N}$ using $M_{\mathtt{S}}(y_{1:N} \mid x_{1:N})$.
S transmits the code for $y_{1:N}$ to R.
R decodes $y_{1:N}$ using $M_{\mathtt{R}}(y_{1:N} \mid x_{1:N})$.

Algorithm 6: High-level overview of the prechastic coding process using the method of Havasi et al. [70] to pass samples from sender to receiver. Note the relative entropy is given in bits.

optimisation problem at iteration i of the prechastic algorithm:

$$\begin{aligned}
\min_{\beta_i, [\boldsymbol{Q_i}]} \quad & \beta_i - \sum_{j=1}^{N} \log_2\Big([Q_i]_{j,y_j}\Big) \\
\text{s.t.} \quad & \sum_{j=1}^{N} \mathrm{KL}[[\boldsymbol{Q_i}]_{\boldsymbol{j}} \parallel [\boldsymbol{P_i}]_{\boldsymbol{j}}] \leq \beta_i \\
& \sum_{k=1}^{K} [Q_i]_{j,k} = 1, \quad \forall j \\
& 0 \leq [Q_i]_{j,k} \leq 1, \quad \forall j, k
\end{aligned}$$

In this notation $[\boldsymbol{Q_i}]$ is an $N \times K$ matrix which represents N independent categorical distributions across the K label classes for each of the inputs $x_{1:N}$. For each input x_j, the model prediction is also a distribution $[\boldsymbol{P_i}]_{\boldsymbol{j}}$ across the K label classes independently from other inputs. Naturally, Q_i and P_i are simply the joint distributions of their respective sets of probabilities. The implicitly non-negative budget variable β_i constrains $\mathrm{KL}[Q_i \parallel P_i]$ via the additive property of relative entropy for independent distributions.

The objective function measures the approximate amount of information β_i that the sender would have to transmit in order for the receiver to draw a sample from Q_i summed with the cost of encoding the true labels if the receiver were to form predictions equivalent to Q_i after training on the sample. Of course, the true cost of encoding the labels is

determined by P_{i+1}; however, using P_{i+1} would inject training cycles into the optimisation problem. Instead, by minimising the objective function over β_i and $[\boldsymbol{Q_i}]$, a direct trade-off is struck at iteration i between the quality of the guiding distribution and the overhead to communicate a sample from it.

Although iteratively using this convex optimisation problem to select each Q_i seems like a promising approach, it also entails a potentially large source of inefficiency. Because the optimisation is only occurring within a given iteration, the optimal budget β_i^*, can often be reapportioned to achieve better results by splitting the iteration into multiple subproblems. Fortunately, we can use insights from this issue with the first-pass attempt as well as our earlier discussions about the computational efficiency of relative entropy coding to design a method of selecting guiding distributions which works very well in practice. Instead of allowing β_i to be an optimisation parameter, we can instead fix it to some pre-determined value β. For the remainder of this thesis we will refer to this approach as *beta* prechastic coding. If β is set sufficiently low, we can avoid intractably high computation costs from the importance sampling REC procedure. Additionally, lower values hedge against the first-pass concerns about overshooting the size of the relative entropy gap between Q_i and P_i. Note of course that lower β values can increase the overall amount of iterations needed in the prechastic scheme; balancing the trade-off in overall computation cost is explored further in Section 3.5.4.

3.5.3.2 Greedy Approach

Previously we avoided including potential P_{i+1} values in our optimisation scheme, however we shall now describe a greedy approach which makes use of them, naturally incurring a cost of increased network calls. Instead of framing the process of choosing Q_i as a convex optimisation problem with a specific relative entropy budget β, we instead generate a fixed number of samples G from P_i then individually train the learning model on each of these samples. We select the sample which produces the best results, *i.e.* minimizes the cost of encoding the true values after training. In this manner, the greedy approach is a slight deviation from the general prechastic scheme as we are not directly creating a guiding distribution Q_i. Algorithm 7 presents a full high-level overview of the greedy prechastic scheme.

3.5.4 Experiments

We will now experimentally evaluate the various prechastic schemes, compare them to the prequential approach, and test the effects of the relevant hyper-parameters. Broadly we undertake supervised learning classification tasks on the MNIST [100] and Fashion-MNIST [173] datasets. Learning is performed on a multilayer perceptron (MLP) with two hidden

S initialises a model $M_{\mathtt{S}}$.
R initialises an identical model $M_{\mathtt{R}}$.
for i := 1 to T {
    P_i := $M(\cdot \mid x_{1:N})$ where $M \equiv M_{\mathtt{R}} \equiv M_{\mathtt{S}}$.
    R and S generate identical sets of G samples $\boldsymbol{p_i^{(1)}}, \boldsymbol{p_i^{(2)}}, \ldots, \boldsymbol{p_i^{(G)}} \sim P_i$.
    S initializes a model M'.
    V := ∞, g^* := 1
    for j := 1 to G {
        S creates a clone of M' and trains it on $\boldsymbol{p_i^{(j)}}$, obtaining M'_j.
        if $-\log_2(M'_j(y_{1:N} \mid x_{1:N})) < V$ {
            V := $-\log_2(M'_j(y_{1:N} \mid x_{1:N}))$
            g^* := j
        }
    }
    S transmits g^* to R at a cost of $\mathcal{O}(\log(G))$ bits.
    $M_{\mathtt{R}}$ is trained on $\boldsymbol{p_i^{(g^*)}}$.
    $M_{\mathtt{S}}$ is trained on $\boldsymbol{p_i^{(g^*)}}$ in an identical manner.
}
S encodes $y_{1:N}$ using $M_{\mathtt{S}}(y_{1:N} \mid x_{1:N})$.
S transmits the code for $y_{1:N}$ to R.
R decodes $y_{1:N}$ using $M_{\mathtt{R}}(y_{1:N} \mid x_{1:N})$.

Algorithm 7: High-level overview of the greedy prechastic coding algorithm.

layers (128 neurons each) and a more powerful LeNet-style network [100]. Note that the unbatched prequential coding scheme requires $\mathcal{O}(N^2)$ items of data to be processed per epoch of training. The greedy prechastic coding scheme processes $\mathcal{O}(TNG)$ whilst, ignoring any potential contributions from guiding distribution computation, the general prechastic coding scheme processes $\mathcal{O}(TN)$. Based on the values of T and G used to achieve efficient results, in addition to the computational cost of computing a full prequential code, our experiments restrict both datasets to smaller sizes of $N = 128, 256$, and 512. Although the prequential computational burden could be offset through the use of exponentially increasing batch sizes (as in Blier and Ollivier [16]), we leave the reduction of prechastic computation as a topic for future work. The implications of the dataset restrictions are discussed further in Section 3.6.

Prequential Baseline Prequential results were obtained for each model and dataset combination over a total of ten trials each, the results of which can be found in Table 3.2. Training was performed using batch sizes of 32 over five epochs. Predictions formed at the end of the last epoch were used to compute test code lengths as transmitting the best value across all epochs for each step of the prequential process would require the transmission of the epoch index and thus incur a large penalty to the total cost.

Prechastic Experiments Prechastic experiments were conducted for the greedy approach from Section 3.5.3.2 as well as the first-pass convex approach from Section 3.5.3.1. A total of 25 trials were conducted for each experiment setting with random training sets of the specified size chosen from the training sets of the corresponding datasets (MNIST or FashionMNIST). The convex approach was conducted under two settings: in one version Q_i was directly sampled (`FPC-Q`) whilst the other version (`FPC-R`) used the REC importance sampling procedure to indirectly sample Q_i through P_i. Clearly, the former is not a valid compression scheme as the receiver cannot know Q_i, however we include it in our results in order to show the impact of the bias in the importance sampling relative entropy coding procedure on the compressed sizes.

Q_i was determined by solving the aforementioned convex optimisation problem via the CVX software framework [61, 62]. In order to improve stability we maintain a running average of the (up to) five previously transmitted samples as a training signal. Furthermore we used $\beta = 7$ for all experiments; we computed the cost of each iteration as $\log_2(\lceil 2^{\beta_i^*} \rceil)$, where β_i^* is the optimized value of β_i. Across all models and both MNIST and FashionMNIST, we used maximum iteration counts of $25, 50$, and 100 for the dataset sizes of $128, 256$, and 512, respectively. In our experiments we also included the approximate cost of transmitting the best index (*i.e.* $\log_2$ of the best index).

All greedy experiments were performed with a G value of 128, thus transmitting 7 bits of information per iteration. In order to mitigate the instability of small dataset training, instead of generating one sample from P_i for each of the G trials, multiple samples were generated and then the average was used as a training signal. Note that no additional information needed to be transmitted per iteration i as we still chose from exactly G options. Larger numbers of multiple samples drive the training signal towards P_i therefore great care was chosen to find multiple samples values which offset the small dataset effects, but were not too large. For the dataset sizes of $128, 256$, and 512 we used multiple sample values of $25, 5$, and 2, respectively (across all models and both MNIST and FashionMNIST).

Naturally, larger datasets required more training iterations to compress to their minimal prechastic sizes. For the dataset sizes of $128, 256$, and 512 we used maximum iteration counts of $20, 40$, and 60, respectively (again, across all models and both MNIST and FashionMNIST). As with the convex experiments, we included the approximate cost of transmitting the best index (*i.e.* $\log_2$ of the best index). All results from the prequential and prechastic experiments are presented in Table 3.2 and discussed in detail in Section 3.6.

Size	Coding	Mnist	
		Mlp	Lenet
128	Preq.	0.896 ± 0.008 (380.8 ± 3.2)	0.895 ± 0.005 (380.6 ± 2.1)
	FPC-Q	0.981 ± 0.006 (417.0 ± 2.5)	0.997 ± 0.008 (424.1 ± 3.3)
	FPC-R	1.002 ± 0.005 (425.9 ± 2.1)	1.014 ± 0.004 (431.3 ± 2.1)
	Greedy	0.948 ± 0.006 (402.9 ± 2.4)	0.933 ± 0.006 (396.5 ± 2.4)
256	Preq.	0.726 ± 0.006 (617.7 ± 5.0)	0.710 ± 0.004 (603.6 ± 3.8)
	FPC-Q	0.879 ± 0.012 (747.3 ± 10.3)	0.805 ± 0.012 (684.6 ± 10.1)
	FPC-R	0.971 ± 0.006 (826.2 ± 4.9)	0.935 ± 0.010 (795.2 ± 8.6)
	Greedy	0.744 ± 0.010 (632.6 ± 7.7)	0.696 ± 0.006 (592.0 ± 5.3)
512	Preq.	0.540 ± 0.006 (917.9 ± 9.4)	0.512 ± 0.005 (870.0 ± 7.8)
	FPC-Q	0.762 ± 0.013 (1296.9 ± 22.1)	0.622 ± 0.007 (1057.8 ± 11.1)
	FPC-R	0.919 ± 0.006 (1563.2 ± 9.3)	0.770 ± 0.008 (1309.0 ± 13.9)
	Greedy	0.620 ± 0.006 (1055.3 ± 9.8)	0.517 ± 0.004 (878.7 ± 6.0)

Size	Coding	Fashion-Mnist	
		Mlp	Lenet
128	Preq.	0.718 ± 0.011 (305.2 ± 4.7)	0.836 ± 0.008 (355.4 ± 3.2)
	FPC-Q	0.909 ± 0.013 (386.5 ± 5.4)	0.980 ± 0.009 (416.5 ± 3.9)
	FPC-R	0.994 ± 0.013 (422.6 ± 5.3)	1.018 ± 0.007 (433.1 ± 3.1)
	Greedy	0.851 ± 0.007 (361.7 ± 3.0)	0.926 ± 0.007 (393.7 ± 2.9)
256	Preq.	0.555 ± 0.010 (472.4 ± 8.3)	0.715 ± 0.004 (608.3 ± 3.3)
	FPC-Q	0.746 ± 0.010 (634.3 ± 8.9)	0.869 ± 0.008 (739.3 ± 7.0)
	FPC-R	0.897 ± 0.011 (763.1 ± 9.5)	0.952 ± 0.009 (809.5 ± 7.4)
	Greedy	0.619 ± 0.007 (526.7 ± 6.0)	0.743 ± 0.007 (631.6 ± 5.6)
512	Preq.	0.447 ± 0.005 (761.0 ± 8.4)	0.577 ± 0.004 (980.8 ± 7.2)
	FPC-Q	0.688 ± 0.008 (1170.5 ± 13.2)	0.756 ± 0.007 (1285.5 ± 11.2)
	FPC-R	0.815 ± 0.008 (1386.1 ± 12.9)	0.868 ± 0.007 (1476.6 ± 12.1)
	Greedy	0.494 ± 0.004 (840.8 ± 7.6)	0.619 ± 0.004 (1053.5 ± 6.4)

Table 3.2: Experimental results from prequential and prechastic tests; three prechastic approaches were tested, the greedy algorithm from Algorithm 7 as well as two first pass convex approaches FPC-Q and FPC-R. Both the average compression ratio and the corresponding average size in bits are reported along with standard error values. The greedy algorithm demonstrates competitive compression values with the prequential approach, especially on the LeNet/MNIST tests. As mentioned in Section 3.5.4 FPC-Q directly samples Q_i and is not a feasible compression scheme, but does illustrate the degradation in performance from the REC importance sampling procedure of [70]. Relative performance differences between the prequential and the greedy algorithms vary with dataset size although no definitive trend appears.

3.6 Conclusion

In our experiments, summarised in Table 3.2, the greedy prechastic approach performs well and approaches the prequential performance in several settings. For the LeNet/MNIST experiments, the greedy results are roughly equivalent to the prequential results; furthermore, across the size 512 datasets the absolute compression rate of the greedy approach never exceeds that of the prequential approach by more than eight percent (approximately fifteen percent in terms of relative performance).

Naturally, as a first-pass attempt the convex results were not as impressive as the greedy results; nevertheless, there were still several interesting findings from `FPC-Q` and `FPC-R` experiments. First, `FPC-R` required larger datasets (*i.e.* the 256 and 512 sizes) in order to demonstrate any significant compression. Second, although overall performance was decreased, relative performance roughly mirrored that of the prequential and greedy with respect to the choice of dataset (MNIST or FashionMNIST) and model. Finally, there was a substantial difference between `FPC-Q` and `FPC-R` which indicated that the REC importance sampling method of Havasi et al. [70] can introduce noticeable bias in this setting.

These experiments, although promising, contained limitations which, to varying degrees, hampered their ability to fully evaluate the prechastic method. Notably, current experiments were limited to small restrictions of the full datasets for computational reasons (see Section 3.5.4). Fortunately this limitation is likely a minor concern as recent research has found that model selection based on small dataset restrictions may give similar results to model selection based on the entire dataset [17]. Additionally, the prechastic method was only evaluated on two relatively simple architectures. This decision was also made with respect to computational considerations, but in order to further establish the relevance of prechastic coding to cutting edge compression schemes more complicated architectures should be taken into account.

Looking beyond these experiments, prechastic coding is a highly flexible coding approach with significant room for further development and improvement. Since the basic prechastic coding does not specify a procedure for choosing the guiding distributions, future work might investigate more advanced selection techniques. The greedy approach could itself also be expanded upon by potentially considering higher order decisions at each iteration. Another topic for future research is the reduction of the computational cost of the prechastic scheme. One possible avenue to reduce the computational cost could be to bootstrap from smaller datasets up to larger datasets, sharing the total iterations across the spectrum of sizes. Finally, we note that future work in the more general instructional coding paradigm is, given its extensive scope, also highly promising.

Chapter 4

Guided Diffusion from Relative Entropy Coding Principles

Diffusion probabilistic models are a class of generative algorithms which, inspired by non-equilibrium statistical physics, iteratively restore structure in data from a known distribution such as an isotropic unit Gaussian. Introduced by Sohl-Dickstein et al. [145], diffusion is flexible enough to model arbitrary data distributions yet tractable enough to allow for straightforward loss minimisation with closed form expressions. We include diffusion models in our discussion of compression methods as they are the cutting edge representative of variational learning. Not only are they strongly related to deep variational auto-encoders with important connections to the ELBO [90], but they have also achieved state-of-the-art likelihood scores on density estimation tasks, beating autoregressive models [88].

Diffusion models also have an interpretation as an instructional coding method, where the increasing levels of complexity are introduced by using decreasing levels of Gaussian noise. At first this might appear to be a contradiction as less noise seemingly corresponds to less complexity, however when transmitting information we are not concerned with specific noisy samples but rather *any* noisy sample from a given Gaussian. For example, it is costly to describe an exact $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ for some high dimension, however a message of the form "sample from $\mathcal{N}(\mathbf{0}, \mathbb{I})$" is essentially sufficient to communicate some $\boldsymbol{x} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$.

Later in this section we will discuss the suitability of diffusion models as the basis for a lossless coding scheme. We will also investigate whether some of the insights and machinery from the compression interpretation can be used to improve diffusion's generative capabilities. This line of inquiry culminates in our paper on importance sampling and guided diffusion.

However, we must first discuss the essentials of diffusion probabilistic models along with recent improvements which have helped transform diffusion into one of the most popular and effective generative methods of the current day. This discussion proceeds until

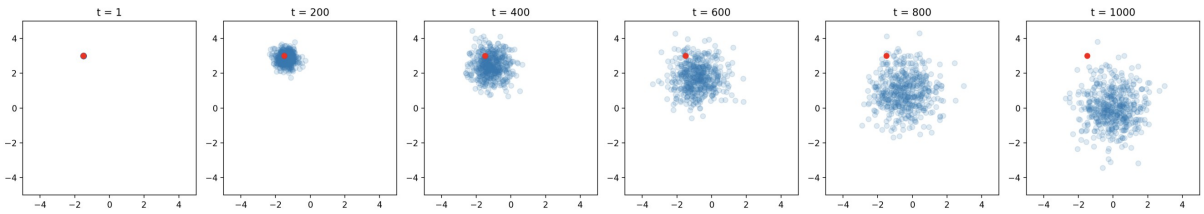


Figure 4.1: The forward diffusion process for a single 2D point (red). Blue points represent samples from the marginal distribution at a given time point.

Section 4.1.1 where we present the relationship between diffusion and the instructional coding process and then delve into the contents of the paper 'Importance-Guided Diffusion' by myself and Pietro Liò. First consider a piece of data $\boldsymbol{x}_0 \in \mathbb{R}^N$ which is sampled from some distribution $q(\boldsymbol{x}_0)$. The forward diffusion process progressively degrades the data by moving it through a Markov chain defined by the transition

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}) \coloneqq \mathcal{N}\Big(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \mathbb{I}\Big) \tag{4.1}$$

where $t = 1, 2, \ldots, T$ is the time step and $\beta_t \in (0, 1)$ is the variance schedule (see Figure 4.1). Naturally, the full joint distribution is defined as

$$q(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T \mid \boldsymbol{x}_0) \coloneqq \prod_{i=1}^{T} q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})$$

and, as denoted in Ho et al. [78], if we set $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$, we can obtain the following useful marginal:

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) \coloneqq \mathcal{N}\big(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\mathbb{I}\big). \tag{4.2}$$

We would like to perform ancestral sampling with $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$, *i.e.* reverse the diffusion process for some $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$ and obtain $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$. For small enough steps, the functional form of $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ is Gaussian [47], however it is also intractable as it depends on the entire data distribution. Instead we must approximate $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ using

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) \coloneqq \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t))$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ are learning algorithms designed to predict the mean and variance, respectively. In order to train the parametrised reverse diffusion process, we will derive a loss function, closely following the presentations in [78, 145]. The loss function is directly related to the variational lower bound (*i.e.* the evidence lower bound or ELBO); optimising the variational lower bound is a training strategy famously employed by VAEs [91]. A thorough analysis of the relationship between the ELBO and diffusion model

objectives can be found in Kingma and Gao [90].

To begin with, first consider the negative log-likelihood $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$ which represents the code length to describe $\boldsymbol{x}_0$ using the model $p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$ and define

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) \coloneqq p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t).$$

The negative log-likelihood can be expanded as

$$\begin{aligned}
-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) &= -\log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})\, d\boldsymbol{x}_{1:T} \\
&= -\log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) \frac{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)}{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)}\, d\boldsymbol{x}_{1:T} \\
&= -\log \int q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0) \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)}\, d\boldsymbol{x}_{1:T} \\
&= -\log \int q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0) p(\boldsymbol{x}_T) \prod_{t=1}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})}\, d\boldsymbol{x}_{1:T} \\
&= -\log \mathbb{E}_{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)} \left[p(\boldsymbol{x}_T) \prod_{t=1}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})} \right].
\end{aligned}$$

It follows from an application of Jensen's inequality [112], that the negative log-likelihood can be bounded by

$$-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0) \leq -\mathbb{E}_{q(\boldsymbol{x}_{1:T} \mid \boldsymbol{x}_0)} \left[\log \left(p(\boldsymbol{x}_T) \prod_{t=1}^{T} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)}{q(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})} \right) \right].$$

We would like to train the diffusion model to minimise the cross entropy $-\mathbb{E}_{q(\boldsymbol{x}_0)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)]$

which is correspondingly bounded by

$$
\begin{aligned}
&-\mathbb{E}_{q(\boldsymbol{x}_0)}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0)] \\
&\quad \leq -\mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[\log\left(p(\boldsymbol{x}_T)\prod_{t=1}^{T}\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)}{q(\boldsymbol{x}_t\mid\boldsymbol{x}_{t-1})}\right)\right] \\
&\quad = \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[-\log p(\boldsymbol{x}_T)+\sum_{t=1}^{T}\log\frac{q(\boldsymbol{x}_t\mid\boldsymbol{x}_{t-1})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)}\right] \\
&\quad = \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[-\log p(\boldsymbol{x}_T)+\sum_{t=2}^{T}\log\frac{q(\boldsymbol{x}_t\mid\boldsymbol{x}_{t-1})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)}+\log\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_0\mid\boldsymbol{x}_1)}\right] \\
&\quad = \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[-\log p(\boldsymbol{x}_T)+\sum_{t=2}^{T}\log\left(\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_0)}\right)+\log\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_0\mid\boldsymbol{x}_1)}\right] \\
&\quad = \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[-\log p(\boldsymbol{x}_T)+\sum_{t=2}^{T}\log\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)}+\log\frac{q(\boldsymbol{x}_T\mid\boldsymbol{x}_0)}{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}+\log\frac{q(\boldsymbol{x}_1\mid\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_0\mid\boldsymbol{x}_1)}\right] \\
&\quad = \mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[\log\frac{q(\boldsymbol{x}_T\mid\boldsymbol{x}_0)}{p(\boldsymbol{x}_T)}+\sum_{t=2}^{T}\log\frac{q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)}{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)}-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0\mid\boldsymbol{x}_1)\right].
\end{aligned}
$$

The full bound can be expressed using KL divergence to obtain the standard form for the loss function:

$$
\begin{aligned}
&\mathbb{E}_{q(\boldsymbol{x}_{0:T})}\left[\ell_T+\sum_{t-2}^{T}\ell_{t-1}+\ell_0\right] \\
\text{where } \ell_T &= \mathrm{KL}[q(\boldsymbol{x}_T\mid\boldsymbol{x}_0)\parallel p(\boldsymbol{x}_T)] \\
\ell_{t-1} &= \mathrm{KL}[q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0)\parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)] \text{ for } t=2,3,\ldots T \\
\ell_0 &= -\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0\mid\boldsymbol{x}_1)
\end{aligned}
\tag{4.3}
$$

Since we can directly sample $q(\boldsymbol{x}_t\mid\boldsymbol{x}_0)$ using Equation (4.2), we can efficiently train a model using this loss function by randomly selecting a time step t, then generating a sample at this step and optimising the relevant loss term in Equation (4.3). Furthermore, if we condition the reverse process $q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t)$ on the original data $\boldsymbol{x}_0$ we can derive the following expression using Bayes' theorem:

$$
q(\boldsymbol{x}_{t-1}\mid\boldsymbol{x}_t,\boldsymbol{x}_0) \coloneqq \mathcal{N}\Big(\boldsymbol{x}_{t-1};\tilde{\boldsymbol{\mu}}_t,\tilde{\beta}_t\mathbb{I}\Big) \tag{4.4}
$$

$$
\text{where } \tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0+\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t \tag{4.5}
$$

$$
\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{4.6}
$$

We can now train a neural network that takes $\boldsymbol{x}_t$ and t as inputs and forms a prediction

$\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ for $\boldsymbol{x}_0$. As discussed in Ho et al. [78], a reasonable choice for $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ is to deterministically set its value to either $\beta_t \mathbb{I}$ or $\tilde{\beta}_t \mathbb{I}$. Although both choices produce similar experimental results [78], for notational consistency we will proceed with $\tilde{\beta}_t \mathbb{I}$ for the remainder of this thesis. It is worth mentioning that $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ can also be learnt instead of set to a fixed value; for example, Nichol and Dhariwal [122] found parametrising $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ to predict an interpolation of $\beta_t \mathbb{I}$ and $\tilde{\beta}_t \mathbb{I}$ in the log domain improved results.

Using $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) = \tilde{\beta}_t \mathbb{I}$, the corresponding $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ for the neural network that forms the prediction $\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ is

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = \mathcal{N}\bigg(\boldsymbol{x}_{t-1}; \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t, \tilde{\beta}_t \mathbb{I}\bigg). \tag{4.7}$$

Recall that the KL divergence between two single variable Gaussian distributions $q \sim \mathcal{N}\big(\mu_q, \sigma_q^2\big)$ and $p \sim \mathcal{N}\big(\mu_p, \sigma_p^2\big)$ is

$$\mathrm{KL}[q \parallel p] = \frac{(\mu_q - \mu_p)^2}{2\sigma_p^2} + \log\frac{\sigma_p}{\sigma_q} + \frac{1}{2}\bigg(\frac{\sigma_q^2}{\sigma_p^2} - 1\bigg).$$

Because $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ are isotropic with identical variance, we can easily derive that

$$\mathrm{KL}[q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)] \propto \mathrm{MSE}(\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \boldsymbol{x}_0) \tag{4.8}$$

where MSE is the Mean Squared Error. The basic procedure for each network training iteration is as follows:

1. Sample $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$ and $t \sim \mathrm{Uniform}(\{1, 2, \dots T\})$.
2. Sample $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t \mid \boldsymbol{x}_0)$.
3. Compute the loss $w_t \cdot \|\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \boldsymbol{x}_0\|_2^2$ where w_t is the weighting constant omitted from Equation (4.8), and perform gradient descent.

Once the network is trained, sampling from $q(\boldsymbol{x}_0)$ is performed by generating a random sample $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$, then iteratively computing samples $\boldsymbol{x}_t \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ from $t = T$ down to $t = 1$.

Although this is a perfectly acceptable network parametrisation (and we make significant use of it in our paper on importance-guided diffusion), Ho et al. [78] found that predicting the noise at each step, rather than the original image, lead to better sample quality. To re-parametrise the network for noise prediction, we can expand Equation (4.2) into

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$, and note that for each particular $\boldsymbol{x}_t$ there is an associated $\boldsymbol{\epsilon}_t$ such that

$$\boldsymbol{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\big(\boldsymbol{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t\big)$$

Therefore, Equation (4.5) can be rewritten as

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right). \tag{4.9}$$

and consequently the approximation $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ for the noise prediction network $\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ is

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t) = \mathcal{N}\left(\boldsymbol{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)\right), \tilde{\beta}_t\mathbb{I}\right). \tag{4.10}$$

As we will see in Section 4.1, this alternative formulation has ramifications for the flow of information throughout the reconstruction process. However it is important to note that Ho et al. [78] also modified the loss function to drop the weighting terms and simply compute $\|\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2$. This has the effect of down weighting smaller t (see Figure 4.2) which allows the model to concentrate its efforts on noisier data. Ho et al. [78] found that because of this shifted emphasis, the simplified loss function produced better quality images; further motivation was established through connections with generative score matching algorithms [153].

4.1 Variance Schedule and Code Length Implications

The choice of variance schedule is an important hyper-parameter decision that can significantly impact the quality of generated images as well as the lossless code length. The standard functional form for $\beta_t \in (0, 1)$ is a linear interpolation between specified values for β_1 and β_T. Ho et al. [78] found that $\beta_1 = 1 \times 10^{-4}$ and $\beta_T = 2 \times 10^{-2}$ worked well for $T = 1000$ as those values ensured the incremental diffusion process was sufficiently small for data that had been scaled to $[-1, 1]$. Later, Nichol and Dhariwal [122] proposed a smoother cosine-based variance schedule where

$$\alpha_t = \frac{f(t)}{f(0)},\ f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2,\ \beta_t = \min\left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999\right).$$

This formulation was designed to address quality issues for smaller images stemming from too much noise at the end of the forward diffusion process.

Naturally, the variance schedule significantly influences the overall code length at each stage of the diffusion process. First, consider the initial transmission cost ℓ_T; as most variance schedules will taper $q(\boldsymbol{x}_T \mid \boldsymbol{x}_0)$ down to an approximately unit isotropic Gaussian distribution, ℓ_T should be relatively negligible. However, as ℓ_T is relevant to our subsequent

discussions about diffusion-based compression schemes, we present the following detailed description:

$$\begin{aligned}\ell_T &= \mathrm{KL}[q(\boldsymbol{x}_T \mid \boldsymbol{x}_0) \parallel p(\boldsymbol{x}_T)] \\ &= \mathrm{KL}\big[\mathcal{N}\big(\boldsymbol{x}_T; \sqrt{\bar{\alpha}_T}\boldsymbol{x}_0, (1-\bar{\alpha}_T)\mathbb{I}\big) \parallel \mathcal{N}(\mathbf{0}, \mathbb{I})\big] \\ &= \frac{\bar{\alpha}_T}{2}\|\boldsymbol{x}_0\|_2^2 - \frac{N}{2}(\bar{\alpha}_T + \log(1-\bar{\alpha}_T)).\end{aligned}$$

Note that we selected $\mathcal{N}(\mathbf{0}, \mathbb{I})$ as our prior $p(\boldsymbol{x}_T)$. Additionally, if we assume that the distribution of data units in $\boldsymbol{x}_0$ roughly forms a unit Gaussian, then the expected per datum cost $\mathbb{E}\Big[\dot{\ell}_T\Big]$ is $-\frac{1}{2}\log(1-\bar{\alpha}_T)^{\dagger}$.

In order to characterise the impact of the variance schedule on the reconstruction loss term ℓ_0, we must first specify how the final distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)$ is constructed and how the subsequent predictions are assigned to discrete outcomes (*e.g.* pixel intensities). A straightforward choice is to set

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_0; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_1, 1), \beta_1\mathbb{I})$$

and assign uniform bins for each discrete possibility based on the total number of possible outcomes. For example, say that we need exactly ω bits to enumerate every datum value (*i.e.* there are 2^ω possible discrete options for each element in $\boldsymbol{x}_0$). If these values are normalised to $[-1, 1]$, then for each $\dot{x} \in [-1, 1]$, the probability that the model predicts $\dot{x}$ at the i^{th} index of $\boldsymbol{x}_0$ is

$$\int_{\dot{x}-\frac{1}{2^\omega-1}}^{\dot{x}+\frac{1}{2^\omega-1}} \mathcal{N}\Big(\mu_{\boldsymbol{\theta}}^{(i)}(\boldsymbol{x}_1, 1), \beta_1\Big)\, dx$$

Assuming that the model $\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_1, 1)$ forms reasonably close predictions for $\boldsymbol{x}_0$, we obtain the following approximation for the per datum reconstruction cost:

$$\dot{\ell}_0 \approx -\log\left(\Phi\left(\frac{1}{(2^\omega-1)\sqrt{\beta_1}}\right) - \Phi\left(\frac{-1}{(2^\omega-1)\sqrt{\beta_1}}\right)\right)$$

where Φ is the standard normal CDF. It follows from this derivation as well as the derivation for $\mathbb{E}\Big[\dot{\ell}_T\Big]$, that for a given dataset the reconstruction and initial transmission costs are essentially determined by $1-\bar{\alpha}_1 \equiv \beta_1$ and $\bar{\alpha}_T$, respectively. The further the values of β_1 and $\bar{\alpha}_T$ are from zero, the larger the cost for reconstruction and initial transmission, respectively.

[†]Alternatively, if we assume that the data falls in the range $[-1, 1]$ this value is an upper bound on $\dot{\ell}_T$.

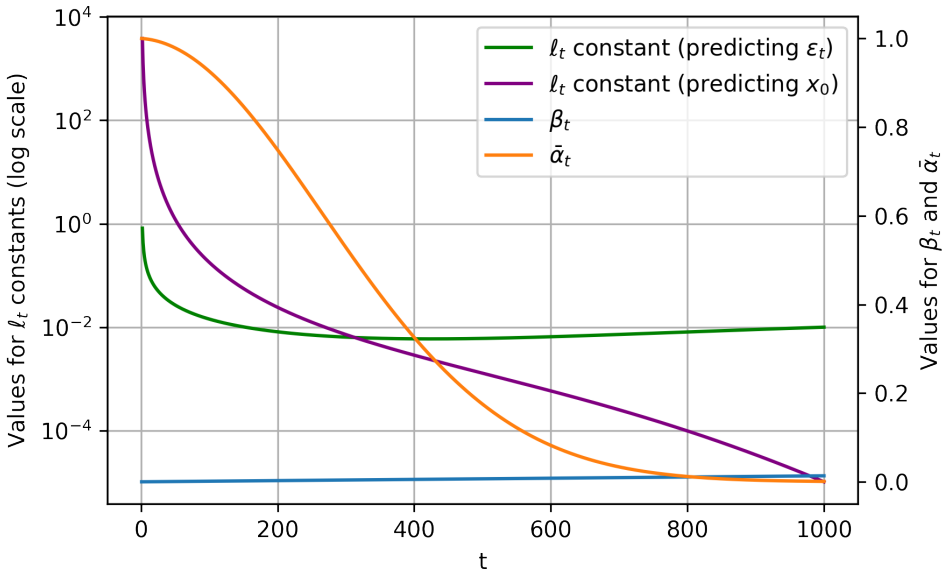


Figure 4.2: A plot of the weighting constants for intermediate ℓ_t (measured in bits) for both $\boldsymbol{\epsilon}_t$ and $\boldsymbol{x}_0$ prediction schemes, overlaid with β_t and $\bar{\alpha}_t$ values. The variance schedule is linear with $T = 1000$, $\beta_1 = 1 \times 10^{-4}$ and $\beta_T \approx 1.36 \times 10^{-2}$ (which gives $\bar{\alpha}_T \approx 1 \times 10^{-3}$).

For $t = 2, 3, \ldots T$ the loss term ℓ_{t-1} can be written as

$$\ell_{t-1} = \mathrm{KL}[q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)] \tag{4.11}$$

$$= \begin{cases} \frac{\beta_t}{2(\alpha_t - \bar{\alpha}_t)} \|\tilde{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2, & \text{if predicting } \boldsymbol{\epsilon}_t \\ \frac{\bar{\alpha}_{t-1}\beta_t}{2(1-\bar{\alpha}_{t-1})(1-\bar{\alpha}_t)} \|\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \boldsymbol{x}_t\|_2^2, & \text{if predicting } \boldsymbol{x}_0. \end{cases} \tag{4.12}$$

Unlike ℓ_0 and ℓ_T, the intermediate loss terms $\ell_{1:T-1}$ can hypothetically be completely mitigated by a trained model for a fixed variance schedule.

4.1.1 Diffusion and Instructional Coding

The vanilla instructional diffusion coding scheme is initiated with the transmission of two highly noisy pieces of data, $\boldsymbol{x}_{T+1}$ and $\boldsymbol{x}_T$. Two items of noisy data in successive time steps are necessary for the chosen neural network to learn sequential denoising no matter the choice of network parametrisation. Because the variance schedule will typically be chosen in a manner that brings $q(\boldsymbol{x}_T \mid \boldsymbol{x}_0)$ close to $\mathcal{N}(\mathbf{0}, \mathbb{I})$, the sender simply samples $\boldsymbol{x}_{T+1}$ from $\mathcal{N}(\mathbf{0}, \mathbb{I})$. The transmission cost of $\boldsymbol{x}_{T+1}$ is negligible as we can ensure that both entities use a shared source of randomness such as an identical random seed. Consequently, the receiver generates an identical $\boldsymbol{x}_{T+1}$ to the receiver by also sampling from $\mathcal{N}(\mathbf{0}, \mathbb{I})$.

As for $\boldsymbol{x}_T$, recall from Section 4.1 that the per datum transmission cost of $\boldsymbol{x}_T$ is approximately $-\frac{1}{2}\log(1 - \bar{\alpha}_T)$. We remark that the scheme could easily be initialised with the standard constructions for $\boldsymbol{x}_T$ and $\boldsymbol{x}_{T-1}$. However, by using a dummy $\boldsymbol{x}_{T+1}$ we find

the presentation to be a bit more refined.

Once both $\boldsymbol{x}_{T+1}$ and $\boldsymbol{x}_T$ have been transmitted, the sender and the receiver use the data to train their own copies of a neural network according to a pre-agreed regimen. Because every facet of this process (architecture, hyper-parameters, training data, etc.) is identical on both ends, the sender and receiver will create the same predicted distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T)$ for $q(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T, \boldsymbol{x}_0)$. As a result, the sender can simulate a sample $\boldsymbol{x}_{T-1}$ from $q(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T, \boldsymbol{x}_0)$ then given the prior $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T)$, communicate the information to generate such a sample to the receiver, allowing it to also sample from $q(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T, \boldsymbol{x}_0)$ without having access to $\boldsymbol{x}_0$. Under the condition that both entities use a shared source of randomness, this sample is identical and thus the ideal transmission cost of $\boldsymbol{x}_{T-1}$ is $\mathrm{KL}[q(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{T-1} \mid \boldsymbol{x}_T)]$.

This training-coding cycle is repeated another $T-2$ times until $\boldsymbol{x}_1$ has been transmitted. Note that as the procedure progresses, the receiver will have access to increasing amounts of training data, all of which it can be used to improve predictions. Once the final prediction $p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)$ is made, the sender can communicate the original data at a cost of $-\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0 \mid \boldsymbol{x}_1)$.

```
S and R sample identical x_{T+1} ~ N(0, I)
S samples x_T ~ q(x_T | x_0).
S transmits x_T to R at a cost of KL[q(x_T | x_0) || N(0, I)].
for t := T to 2 {
        S and R train identical p_θ using x_{t:T+1}.
        S samples x_{t-1} ~ q(x_{t-1} | x_t, x_0).
        S transmits x_{t-1} to R at a cost of KL[q(x_{t-1} | x_t, x_0) || p_θ(x_{t-1} | x_t)].
}
S and R train identical p_θ using x_{1:T+1}.
S transmits x_0 to R at a cost of −log p_θ(x_0 | x_1).
```

Algorithm 8: High-level overview of the instructional diffusion coding process.

The instruction diffusion coding process is summarised in Algorithm 8. In order to compress $\boldsymbol{x}_0$, at each iteration t the model $p_{\boldsymbol{\theta}}$ will need to learn how to approximate the distribution $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$ solely from $\boldsymbol{x}_{t:T+1}$. However, there is a subtle yet important difference between the instructional process and the standard training procedure for a given time step t. In the standard process, the network predicts $\boldsymbol{x}_0$ or $\boldsymbol{\epsilon}_t$ and then directly uses the actual values to compute the MSE. In the instructional process, the receiver will never have access to the true $\boldsymbol{x}_0$ or $\boldsymbol{\epsilon}_t$.

The most straightforward choice to train $p_{\boldsymbol{\theta}}$ is to have the network learn from estimates of $\boldsymbol{x}_0$ or $\boldsymbol{\epsilon}_t$ derived from the available data $\boldsymbol{x}_{t:T+1}$. For example, consider the network parametrisation $\tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ and let $\hat{\boldsymbol{x}}_0^{(k-1)}$ represent the estimate of $\boldsymbol{x}_0$ derived from

Equation (4.5) where

$$\hat{\boldsymbol{x}}_0^{(k-1)} \coloneqq \frac{1-\bar{\alpha}_k}{\sqrt{\bar{\alpha}_{k-1}}\beta_k}\left(\boldsymbol{x}_{k-1} - \frac{\sqrt{\alpha_k}(1-\bar{\alpha}_{k-1})}{1-\bar{\alpha}_k}\boldsymbol{x}_k\right)$$

and $k \in \{t+1, t+2, \dots T+1\}$. The estimator is unbiased but suffers from high variance for large k. As with the super-resolution instructional coding example, the instructional diffusion scheme presented above is an alternative interpretation/formulation of pre-existing compression algorithms based on diffusion [156].

4.2 Importance-Guided Diffusion

This section presents the paper 'Importance-Guided Diffusion' by myself and Pietro Liò which appeared at the peer-reviewed 2023 *NeurIPS Workshop on Diffusion Models* as a poster. As the first author, I conceived of the idea of using relative entropy coding methods for conditional diffusion and also wrote the paper; my co-author Pietro Liò provided helpful supervision and guidance. Accordingly, all of the figures and much of the text in this section and the following section (Section 4.3) remains largely unchanged apart from modifications and edits made to align with the structure of the thesis.

In the past few years, conditional diffusion-based generative models have emerged as an indispensable tool for advancing the fidelity and diversity of sample generation, pushing the boundaries of applications such as image synthesis, style transfer, and adaptive content generation (see Section 4.2.1). In this paper, we introduce a novel approach to guide the diffusion process towards a given condition, such as a reference image, by using importance sampling procedures for efficient sample communication. Our method is rooted in information theory and can condition without modifying the original diffusion network or adding additional passes through the network. As a result, this plug-and-play approach to conditional diffusion integrates seamlessly with existing unconditioned diffusion architectures.

More specifically, our method is derived from relative entropy coding and the index coding procedure [48], recasting diffusion as an auxiliary variable importance sampling procedure. Guided generation is performed by a selection process on random samples from the Gaussian distribution parametrised to approximate the reverse diffusion process. Consequently, it is able to influence the generative process without the need for gradient information or tampering with the network in any manner. Using an approach with a clear information-theoretic framework offers a principled mechanism to both quantify and adjust the degree of conditioning, enabling precise navigation across a large spectrum of generative outputs. Finally, we present experimental results which indicate the technique produces intuitively meaningful conditional outputs while maintaining a relatively minimal

increase in computational burden.

4.2.1 Previous Work

Conditional approaches to diffusion often use a static network and alter the sampling process to achieve the desired guidance [7, 31, 32, 60, 167] One particularly notable example of this approach was given by Dhariwal and Nichol [39] who proposed a classifier-guided approach which trained an external classifier on noisy images and used the gradients to alter the predictions of the diffusion model at each iteration. Alternatively, a classifier-free approach was adopted by Ho and Salimans [76]; their network was trained on both conditioned and unconditioned pairs, with the condition dropped out for a specified percentage of instances, and achieved a good balance between the Fréchet inception distance [72] and the inception score [138]. The classifier-free diffusion approach was later shown to excel in image synthesis [123], markedly surpassing CLIP guidance [128]. The strategy falls within the larger framework of conditioned diffusion which operates directly on the diffusion network [6, 166]. Although this framework produces excellent results, such methods are computationally expensive as the conditioned diffusion models must be trained.

Although our paper is concerned with conditional/guided diffusion, we establish a theoretical basis for our approach through compression methods for variational learning from the relative entropy coding literature. As such, we would be remiss not to mention previous work on diffusion and compression. In their foundational work "Denoising Diffusion Probabilistic Models" Ho et al. [78] include a study of the rate-distortion behaviour of the model which assumes access to Havasi et al. [70]'s procedure. In Kingma et al. [88], diffusion was incorporated into a lossless compression paradigm via bits-back coding. As discussed in Section 4.1.1, Theis et al. [156] developed a lossy approach and corresponding theoretical analysis derived from a rate-distortion perspective. Further work on lossy compression and conditional diffusion can be found in Yang and Mandt [176].

4.2.2 The Importance-Guided Diffusion Algorithm

In order to leverage REC methods for conditional diffusion, we first present a guided generation algorithm inspired by the importance sampling/index coding approaches discussed in Section 3.5.2 called Importance-Guided Diffusion (IGD) (see Algorithm 9 for an overview). Fundamentally, IGD recasts the reverse diffusion process as an auxiliary variable sequence in the index coding paradigm from a standard multivariate Gaussian to a distribution of interest and uses this interpretation to push the generative process in a desired direction. We do not need to derive the auxiliary forms as is required in standard index coding as we will work directly with the diffusion process corresponding to ℓ_1 through ℓ_{T-1}.

More specifically, we define each auxiliary prior conditioned on the previous auxiliary

$\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbb{I})$
`for` $t := T-1$ `to` 1 `{`
    $k \leftarrow T - t$
    $\boldsymbol{x}_{t_1}, \boldsymbol{x}_{t_2}, \ldots \boldsymbol{x}_{t_{M_k}} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t+1})$
`}`
`for` $i := 1$ `to` M_k `{`
    $\boldsymbol{w}_i \leftarrow \frac{q\left(\boldsymbol{x}_{t_i} | \boldsymbol{x}_t, \tilde{\boldsymbol{x}}_0\right)}{p_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{t_i} | \boldsymbol{x}_t\right)}$
`}`
`for` $j := 1$ `to` N `{`
    $p_j(i) \coloneqq \frac{1-\lambda}{M_k} + \lambda \cdot \frac{\boldsymbol{w}_i[j]}{\sum_i \boldsymbol{w}_i[j]}$ where $i \in \{1, 2, \ldots, M_k\}$
    $i_j \sim p_j(i)$
    $\boldsymbol{x}_t[j] \leftarrow \boldsymbol{x}_{t_{i_j}}[j]$
`}`

Algorithm 9: Pseudocode for the importance-guided diffusion algorithm.

variable, $p(\boldsymbol{a}_k \mid \boldsymbol{a}_{k-1})$, as the approximate reverse diffusion $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ where $k = T-t+1$. Note that using an auxiliary prior conditioned on the previous auxiliary variable is a departure from the conventional index coding approach. However, even within the context of a dynamic coding scheme this alteration is perfectly acceptable as the receiver will have knowledge of the conditional values at the time when it needs to use the prior. The auxiliary target distributions are based on $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$ from the VLB; however, we are interested in conditional generation, therefore we set $q(\boldsymbol{a}_k \mid \boldsymbol{a}_{k-1}) = q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \tilde{\boldsymbol{x}}_0)$ where $\tilde{\boldsymbol{x}}_0$ is the conditioning term.

Note that our auxiliary prior and target distributions, which were chosen to match the iterative diffusion process, do not necessarily lead to even jumps in relative entropy between stages (see Figure 4.2 for graphical intuition as to why this is the case). This is a change from the standard index coding which sets these jumps approximately equal to the constant hyperparameter Ω and in this manner our method is more akin to iterative importance sampling. To adjust our algorithm to adapt to this scenario, we simply make the number of samples drawn at each stage k a dynamic variable M_k which we can determine in advance.

The selection of the dynamic M_k can be approached from an overly cautious perspective by examining the design of $\mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t))$ and Equations (4.4) to (4.6) or their reformulations, and subsequently estimating the relative entropies from trivial model predictions. However, in our experiments we find that such estimates are unnecessary for good image production and that M_k can be set rather simply (see Section 4.3). We also included a parameter λ in order to interpolate between the extremes of sampling from the importance-derived categorical distribution and sampling without respect to importance. Experimentally this hyperparameter proved to be an intuitive way to control

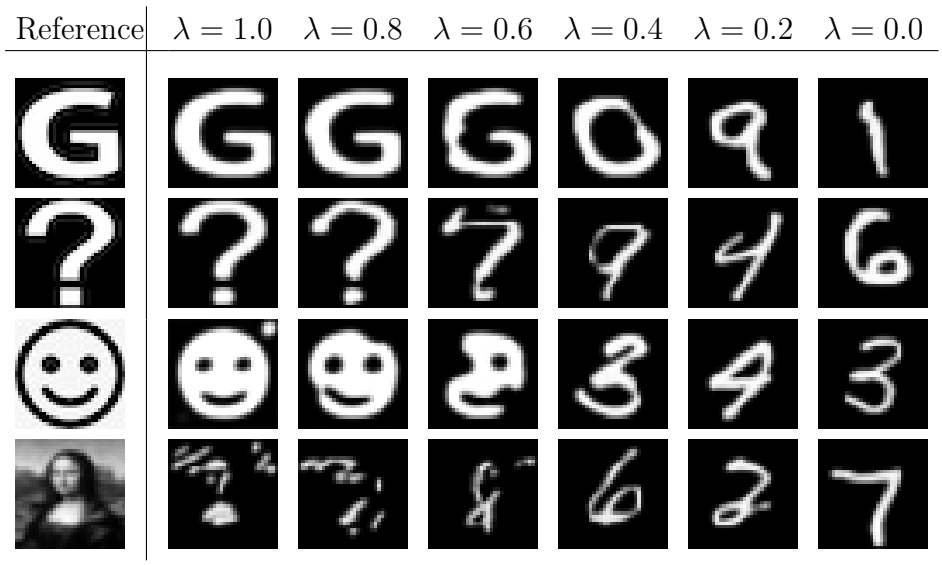


Figure 4.3: Samples from different values of λ given the reference images on the left.

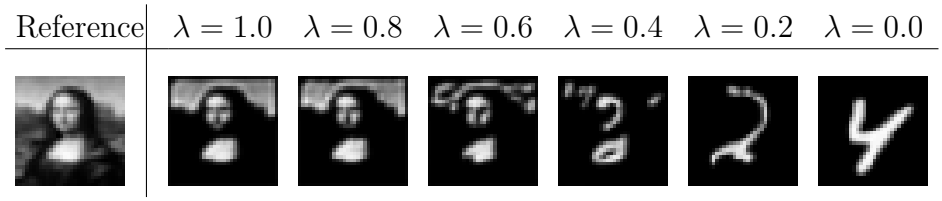


Figure 4.4: Samples for the Mona Lisa using an increased M_k towards the latter iterations.

the level of conditioning. Finally, in a departure from our coding foundations we generated the categorical distribution at the pixel level rather than at the image level. Although this change would lead to longer realised codes due to the corresponding overhead if a compression procedure were to be implemented, it significantly speeds up the conditional generation process in IGD because the relative entropies are smaller and no longer dependent on the image size.

4.3 Examples and Conclusion

In Figure 4.3 we present images generated using importance-guided diffusion across the range of λ values. A U-Net style network with ~ 110 million parameters was trained up to $30,000$ iterations on batch sizes of 32 using the AdamW optimiser[110] ($\text{LR} = 1 \times 10^{-4}$) on MNIST [100] to predict $\boldsymbol{x}_0$. We found predicting $\boldsymbol{x}_0$ rather than $\boldsymbol{\epsilon}_t$ produced superior results. With regards to the diffusion hyperparameters and setting, we employed a linear variance schedule with $\beta_1 = 1 \times 10^{-4}$, $\beta_T \approx 1.4 \times 10^{-2}$, and $T = 1,000$.

Because the target and prior distributions are presumably different, the minimum value of M_k must be 2. In fact, we fixed $M_k = 2$ at each sampling iteration for the images in Figure 4.3 as we found that this imparted a pleasing amount of the original MNIST

style. It is important to keep in mind that this value of M_k could be overkill for some iterations (*i.e.* when the relative entropy is close to zero) and underkill for others. In Figure 4.4 we used a more generous, dynamic $M_{T-t+1} = \min\{\lceil \exp\left(\frac{\bar{\alpha}_{t-1}\beta_t}{(1-\bar{\alpha}_t)(1-\bar{\alpha}_{t-1})}\right) \rceil, 7500\}$ which resulted in a heaver emphasis towards the reference image and a re-interpretation of the λ parameter.

Our experiments demonstrated a promising proof of concept that IGD can effectively span a large range of conditioning bias, generating new images from the reference image with both small and large amounts of input from the training dataset. This ability was showcased using conditional reference images markedly different than the MNIST images used to train the diffusion model. Intermediate images on the interpolation spectrum effectively imparted high-level training features such as the 'hand-written' MNIST characteristics.

In subsequent work we would like to investigate the successes and limitations of IGD in a larger variety of settings and experiments. Such research should assess the robustness of the conditional interpolation across these settings. Future applications might also explore alternative modalities such as audio and video generation. Much of the promise of IGD is in its flexibility, and expansion beyond image generation would present opportunities for challenging issues such as fine-grained conditioning on sequential video frames. Due in part to the principled nature by which IGD was derived, we might speculate that further work in this area could lead to alternative diffusion training methods based on resistance to the trained network's ability to generate conditional images from unseen datasets. To wit, stopping procedures could be designed based on the quality of images generated by IGD on non-training set data for pre-determined λ and M_k settings.

Chapter 5

Algorithmic Complexity and Regularisation in Graph Neural Networks

For there you see, friend Sancho Panza, thirty or more enormous giants with whom I intend to do battle and whose lives I intend to take, and with the spoils we shall begin to grow rich.

Don Quixote

What giants?

Sancho Panza [24]

5.1 Approximation Methods

In the summer of 2010, Gregory Chaitin and the renowned computer scientist Marvin Minsky participated in a panel at the World Science Festival. At the end of the discussion, titled 'The Limits of Understanding', Minsky made the following remark:

> It seems to me that the most important discovery since Gödel was the discovery by Chaitin, Solomonoff, and Kolmogorov of a concept called algorithmic probability...this is a beautiful theory, everybody should learn it, but its got one problem which is that you can't actually calculate what this theory predicts....however, it should be possible to make practical approximations to [this] theory that will make better predictions than anything we have today and everybody should learn all about that and spend the rest of their lives

working on it [121].

Minsky's status as one of the founding fathers of artificial intelligence has likely contributed to the popularity of this quote within the algorithmic information theory community. However, the statement extends beyond a simple endorsement into an expression of optimism about practical research potential. Despite Minsky's reputation, one might ask if such optimism is warranted and whether it is possible to make meaningful progress on approximation methods? Algorithmic probability and Kolmogorov complexity are incomputable, yet the extent of their utility in a theoretical capacity is considerable. This potency clearly allows the allure of their computation to endure in the form of approximation, but are such pursuits merely wild goose chases?

Naturally, any lossless compression algorithm can serve as the basis for an approximation of the Kolmogorov complexity. Many of the best lossless compression methods work by pairing an advanced learning algorithm, such as a neural network, with a form of entropy coding. We discussed this topic in greater detail in Chapter 3; however, as we shall now see, there are plenty of approximation algorithms that do not use complex learning procedures.

5.1.1 Canonical Compression Methods

Many popular lossless compression approaches are characterised by a lack of advanced modeling components, relying instead on simple constructions such as the Burrows-Wheeler transform [21] or Lempel-Ziv dictionaries [189, 190]. However, such broad approaches are, at their core, simplistic approximations. Re-purposing a quip from John von Neumann, anyone who believes that general-purpose compression methods, a category of algorithms which includes programs such as `gzip`, can form a drop-in replacement for the Kolmogorov complexity is, of course, living in a state of sin.

Despite the rudimentary nature of these algorithms in comparison to more sophisticated approaches such as deep learning, serious results in algorithmic information theory have found that basic compression methods can serve as meaningful complexity estimates. For example, the normalised information distance between $x, y \in \mathcal{B}^*$,

$$\frac{\max\{K(x \mid y), K(y \mid x)\}}{\max\{K(x), K(y)\}}$$

can be approximated by the normalised compression distance

$$\frac{\tilde{K}(xy) - \min\{\tilde{K}(x), \tilde{K}(y)\}}{\max\{\tilde{K}(x), \tilde{K}(y)\}}$$

which, in turn, has useful applications even when $\tilde{K}$ is supplied by crude estimators such as `bzip2` [33]. Simplicity bias, a result in algorithmic information theory somewhat akin

to a one-sided coding theorem for computable functions, has also been demonstrated in natural systems by using a complexity measure related to the general purpose Lempel-Ziv compression algorithm [41]. The simplicity bias phenomenon turns out to have interesting implications for deep neural networks, and is explored in further detail in Section 5.2.1. Lempel-Ziv complexity warrants a more detailed discussion as it has a provably deeper connection to Kolmogorov complexity and is one of the more popular methods for crude approximation.

Lempel-Ziv Complexity In 1976, Abraham Lempel and Jacob Ziv proposed a technique to evaluate the randomness of finite sequences [102]. This complexity measure would heavily inform the later LZ77 and LZ78 lossless compression algorithms [189, 190] which, in turn, became the foundation for other lossless compression methods such as the ubiquitous LZW algorithm [169].

The Lempel-Ziv complexity of a finite string $\boldsymbol{s}_{1:n}$ where $n \geq 2$ is calculated in the following manner: first, initialise $\boldsymbol{q} = s_1$ and $\boldsymbol{r} = s_2$, then set the complexity counter $c = 1$. Next, determine if $\boldsymbol{r}$ is a contiguous substring of $\boldsymbol{qr}[:-1]$, the concatenation of $\boldsymbol{q}$ and $\boldsymbol{r}$ followed by the deletion of the last digit. In the current initial case this is equivalent to checking if $s_1 == s_2$. If $\boldsymbol{r}$ is indeed a contiguous substring of $\boldsymbol{qr}[:-1]$ then $\boldsymbol{r}$ is updated via concatenation with the next symbol, *i.e.* $\boldsymbol{r} = \boldsymbol{r}s_3$. This process continues until $\boldsymbol{r}$ is no longer a contiguous substring of $\boldsymbol{qr}[:-1]$. If this occurs, c is incremented, $\boldsymbol{q}$ is set to $\boldsymbol{r}$, and $\boldsymbol{r}$ is set to the next symbol in the sequence. This entire process is repeated until immediately before the final symbol s_n is under consideration, at which point $c + 1$ is returned.

At its core, the Lempel-Ziv approach progressively collects special types of subsequences in a given string. For example, consider the following binary string:

$$\mathbf{s} = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1$$

If we record the initial $\boldsymbol{q}$ and each $\boldsymbol{r}$ before it is overwritten or the string is finished, we get the dictionary

$$\{1, 10, 101011, 100, 0100, 010100, 0111\},$$

thus, $c_{LZ}(\boldsymbol{s})$, the Lempel-Ziv complexity of $\boldsymbol{s}$ is seven. A similar Lempel-Ziv style approach to dictionary construction is to parse the string from left to right and collect substrings which have not yet appeared. If applied to the previous example, the algorithm would yield the dictionary

$$\{1, 10, 101, 0, 11, 100, 01, 00, 010, 1000, 111\}.$$

This method forms the basis for a universal compression algorithm [112], but can give

strikingly different dictionaries. For example, consider the string $\mathbf{0}^n$; the Lempel-Ziv complexity $c_{LZ}(\mathbf{0}^n)$ is two as the corresponding dictionary (constructed in the manner described above) is $\{\mathtt{0}, \mathtt{0}\}$. The second approach would lead to a dictionary of incrementally longer strings of zeros $\{\mathtt{0}, \mathtt{00}, \dots\}$ which has size $\mathcal{O}(\sqrt{n})$.

As the Kolmogorov complexity of $\mathbf{0}^n$ is $\mathcal{O}(\log n)$, one might surmise that $c_{LZ}(\boldsymbol{x}) \cdot \log(n)$ could form an approximation to the Kolmogorov complexity. In fact, the relationship between Kolmogorov complexity and Lempel-Ziv complexity bears resemblance to that of Kolmogorov complexity and the Shannon rate as, for a stationary and ergodic source, the Kolmogorov and Shannon rates are limit equivalent to the Lempel-Ziv complexity multiplied by the base-2 logarithm of the sequence size [104, 190]. Despite this equivalence, one of the more popular versions of Lempel-Ziv complexity makes several alterations to further improve approximation quality [40, 41] This complexity measure is defined as

$$C_{LZ}(\boldsymbol{x}) = \begin{cases} \log(n), & \text{if } \boldsymbol{x} = \mathbf{0}^n \text{ or } \mathbf{1}^n \\ \log(n)[c_{LZ}(x_1 x_2 ... x_n) + c_{LZ}(x_n ... x_2 x_1)]/2, & \text{otherwise} \end{cases}$$

and brings the trivial strings $\mathbf{0}^n$ and $\mathbf{1}^n$ in line with their Kolmogorov complexities, whilst splitting the difference between forward and reverse processing.

5.2 Implications for Neural Networks

Kolmogorov complexity has a long history of relevancy to machine learning far beyond the bonds of induction and information. Topics from PAC-learning [107] to transfer learning [114] have all benefited from the Kolmogorov perspective. With regards to neural networks, pioneering work was carried out by Schmidhuber [140] in the form of a method for finding algorithmically simple networks in toy settings. Recently Ard Louis' group at the University of Oxford have presented a series of papers connecting deep neural networks with a concept from applied algorithmic information theory called simplicity bias [119, 120, 162]. Before reviewing their work, it is essential to first discuss the simplicity bias phenomenon.

5.2.1 Simplicity Bias

Introduced in [41], simplicity bias is, at a high level, a tendency displayed by many input-output maps for the distribution of outputs formed by providing the map with random inputs to supply individual object probabilities that are bounded by the complexity of the output. To reach a more formal definition, first note that for any computable map $f : I \to O$ where I is a set of input sequences of size n and O is a set of discrete outputs, we have the following theorem:

Theorem 5.2.1 (Computable Function Coding Upper Bound)**.** *For a given $x \in O$, the probability $p(x)$ that a random input $y \in I$ satisfies $f(y) = x$ is bounded by*

$$p(x) \leq 2^{-K(x|f,n)+\mathcal{O}(1)}.$$

where $K(x \mid f, n)$ is the Kolmogorov complexity of x given programs to calculate f and n.

This result can be established via a standard enumeration approach to obtain the probability, followed by a prefix-free coding at a cost of $-\log(p(x)) + \mathcal{O}(1)$. Dingle et al. [41] justify and impose an additional four restrictions:

1. **Limited Complexity**: Maps must asymptotically satisfy $K(f) + K(n) \ll K(x) + \mathcal{O}(1)$ for large x. Under this constraint $K(x) \approx K(x \mid f, n) + \mathcal{O}(1)$ and the previous bound on $p(x)$ becomes $p(x) \lesssim 2^{-K(x)+\mathcal{O}(1)}$.

2. **Redundancy**: There are significantly more inputs than outputs, *i.e.* $|I| \gg |O|$.

3. **Finite Size**: In order to mitigate the effects of finite size, $|O| \gg 1$.

4. **Nonlinearity**: Linear functions cannot demonstrate output bias, therefore f must be nonlinear.

Under these restrictions one can find many real-world systems, given an approximation $\tilde{K}(x)$, which obey $p(x) \lesssim 2^{-a\tilde{K}(x)+b}$, where the constants $a > 0$ and b depend solely on f. In experiments conducted using the C_{LZ} complexity measure defined in Section 5.1.1, simplicity bias was demonstrated to occur in, amongst others, RNA sequence to structure maps and financial trading models. Later work has found evidence of simplicity bias in a variety of domains including biological settings [87], and economic and development time series [42]. Investigations into simplicity bias and neural networks have led to interesting conjectures about generalisation and are reviewed in further detail in the following section.

5.2.1.1 Simplicity and Inductive Bias in Neural Networks and Deep Learning

This section briefly details a suite of relatively recent papers from Ard Louis' group at the University of Oxford. The first of these papers by Valle-Pérez et al. [162] links simplicity bias with generalisation properties of Deep Neural Networks (DNNs). Valle-Pérez et al. accomplish this by demonstrating evidence of simplicity bias in the parameter-function map of a selection of DNNs and then building upon this viewpoint to derive tight PAC-Bayes generalisation bounds.

Later work by Mingard et al. [119] explores bias towards Boolean functions of low entropy present in randomly initialised neural networks. Finally in [120], Mingard et al. take a broader perspective and find that structured data in conjunction with a preference for

simplicity is important to help explain the generalisation capabilities of DNNs. Collectively, this body of work makes a compelling case for the relevancy and potency of algorithmic information theory as a tool to understand neural networks and deep learning.

5.3 Investigating Kolmogorov Complexity Regularisation in Graph Neural Networks

In this section we shall explore Kolmogorov complexity estimation in the context of network theory. Specifically, this section is largely comprised of the paper "Investigating Estimated Kolmogorov Complexity as a Means of Regularization for Link Prediction" written by myself, Ramon Viñas, and Pietro Liò [52]. All of the tables and much of the text in this section and the following section (Section 5.4) remains largely unchanged apart from edits as well as additional discourse in order to fit the context of this thesis. This paper was accepted and presented at the peer-reviewed *Causal Discovery & Causality-Inspired Machine Learning Workshop* at *Neural Information Processing Systems 2020*. I conceived of the central idea and methodology and wrote the majority of the paper; Ramon Viñas formed connections with causality and helped write the paper; Pietro Liò provided feedback and guidance.

The research project was initially inspired by the desire to improve methods for training graph neural networks for the task of link prediction. The basic premise was this: real-world networks often have relatively simple generating mechanisms - for instance, citation networks and social networks are typically scale-free, *i.e.* the degree distribution is well-approximated by a power law. Such networks are often well-modelled by concise equations (see later discussions about preferential attachment). I wanted to inject a preference for predicting links such that the resulting graphs could also be explained by simple generating mechanisms.

At the time, creating a GNN regulariser based on estimated Kolmogorov complexity seemed to be a promising route as I had recently read Hector Zenil's applied work on graphs and Kolmogorov complexity [183, 184]. Zenil and his collaborators had created a Kolmogorov complexity estimator based on a rather creative approach involving small Turing machines that purportedly achieved a level of algorithmic nuance [38, 147, 183]. This method and its extensions [183, 184] appeared to be ideal estimators for our purposes and our implementations of these concepts as regularisers achieved excellent results on a suite of GNN network prediction problems.

However, we proceeded to design a control for our experiments that essentially replaced the 'algorithmic complexity influence' with constant weights. This control performed just as well and led us to conclude that none of the regularisation improvements could be attributed to a direct estimation of Kolmogorov complexity. In and of itself our paper

does not directly refute the validity of these estimation methods. However, our results, in conjunction with a highly critical theoretical review of the aforementioned estimation methods by noted expert Paul Vitányi [164], have led us to a sceptical position on the usefulness of the approximation approaches examined.

5.3.1 Real-World Networks and GNNs

Network models have become an indispensable tool to study complex systems of discrete objects and their interactions and network science has been applied with great success to a variety of scientific disciplines, often resulting in rich data sets that can be studied using machine learning [8, 9, 103]. An important task in the study of networks is link prediction between graph nodes based on incomplete instances of data [117]. The importance of link prediction techniques is underscored by its application to a variety of topics such as protein function anticipation [80], friendship identification for social network users [43] and scientific collaboration inference [109].

As previously mentioned, many complex networks have simple causal mechanisms that underlie their generation. For example, it has been theorised that scale-free networks such as the World Wide Web are often generated by a system of preferential attachment whereby new nodes are more likely to attach to nodes that are already well connected [9]. Informally, the small-world property states that any two nodes in a graph can likely reach each other through a short path. This property is present in many real-world phenomenon such as social networks, and graphs with this property can be artificially generated by the simple Watts-Strogatz model [168]. The observation that complex networks can be characterised by such simple mechanisms raises the possibility of incorporating complexity biases for network modelling.

In recent years, GNNs have proven to be highly proficient at solving link prediction problems [94, 172, 174]. By taking advantage of relational inductive biases to obtain high-level node representations, architectures such as graph auto-encoders can decode meaningful, unseen links from their latent representations [94, 172]. GNNs can also be trained in the canonical way by defining a loss function and employing gradient-based learning methods. This flexible training framework presents an opportunity to incorporate information about the causal generating mechanisms of networks through a regularisation term.

5.3.2 Related Work

In this work, we investigate penalising graphs with large estimated Kolmogorov complexity to encourage the creation of graphs which have been generated through descriptionally simple mechanisms. We accomplish this by proposing a differentiable regularisation term

for link prediction based on a popular method to estimate the Kolmogorov complexity of graphs [183, 184]. By proposing Kolmogorov complexity as a regulariser for graph outputs, we aim to augment the ability of models such as GNNs to generalise when predicting links. Through a controlled experiment we learn that although our proposed regularisation term shows good results, it very likely works for reasons related to an aggregation routine rather than any direct estimation of the Kolmogorov complexity.

In [71], Kolmogorov complexity was proposed by Hernández-Orozco et al. as a regularisation term with a weighting parameter in conjunction with a general loss function. Similar to Schmidhuber's approach in [140], Hernández-Orozco et al.'s form of regularisation is designed to push a model towards a lower algorithmic complexity, thus increasing the algorithmic probability of the model. In our methodology, we attempt to use algorithmic complexity to regulate the output of the model, rather than the model. We took this approach because we wanted to reward the model for learning to generate objects from simpler rule sets.

In order to approximate the gradient of our estimated Kolmogorov complexity based regularisation term, we use a method based on perturbing the predicted adjacency matrix. The study of the algorithmic causality of an object by means of a perturbation calculus on the object's estimated Kolmogorov complexity was pioneered by Zenil et al. in [185]. Their approach was recently used to define an unsupervised learning algorithm for identifying generating mechanisms in graphs [186].

5.3.3 Selecting an Approximation Method

As discussed in Chapter 2 both Kolmogorov complexity and algorithmic probability are uncomputable for reasons related to the halting problem, therefore approximations are required. Established techniques such as Lempel-Ziv have strong theoretical connections, but might suffer from an inability to capture algorithmic nuances [181, 184]. In light of this, we turned to the Coding Theorem Method (CTM) which provides a straightforward approximation to the Kolmogorov complexity of an object that purports to capture algorithmic features [147]. The CTM directly approximates the algorithmic probability of small strings by exploring the large space of Turing machines with a fixed number of symbols and states. In more detail, let $(n, 2)$ be the class of all n-state 2-symbol Turing machines T using the Turing machine formalism outlined in the busy beaver game [129]. The CTM defines the following complexity measure for a binary string s:

$$D_{(n,2)}(s) = \frac{|\{T \in (n,2) : T \text{ produces } s\}|}{|\{T \in (n,2) : T \text{ halts}\}|}$$

Of course, as dictated by the halting problem it is impossible in general to know if a machine will halt; however, for the 2 symbol case the largest number of steps taken before

halting are known up to $n = 4$ (and theorised for $n = 5$) [147]. Therefore, $D_{(n,2)}(s)$ can be computed for small n through brute-force. Through an approximate form of Levin's Coding Theorem, the CTM estimate of Kolmogorov complexity, denoted by $CTM_{(n,2)}(s)$, is given as:

$$CTM_{(n,2)}(s) = -\log_2 D_{(n,2)}(s)$$

Unfortunately, it is ultimately not possible to use the CTM approximation on all but a relatively small finite set of objects due to the rapid growth of the busy beaver function. To address this limitation, the Block Decomposition Method (BDM) [184] was developed to extend the CTM via an aggregation rule designed to approximate the Kolmogorov complexity of a large object from its smaller components. The BDM has been applied to problems in machine learning and causality [71, 186] and is described as follows: For a given binary string s, decompose the string into the multiset $\mathcal{S}_s = \{s_1, s_2, \ldots, s_{\frac{|s|}{r}}\}$ where s_i are consecutive slices from s of size r. The value of r is chosen to be small enough so that the CTM can be used to compute an complexity approximation of the slice (we assume the length of s is divisible by r). Let $\mathcal{U}_s$ be the set of unique values in $\mathcal{S}_s$ and let c_u be the number of times slice $u \in \mathcal{U}_s$ appears in $\mathcal{S}_s$. The BDM is built on the following premise: if the CTM approximates the Kolmogorov complexity for each slice u, then a program with an estimated complexity of $\sum_{u \in \mathcal{U}_s} CTM_{(n,2)}(u)$ can be used to generate all of the unique building blocks of s. The number of times each slice u appears in s can be specified in $\log_2(c_u)$ bits, thus the BDM complexity approximation for s is:

$$BDM_{(n,2)}(s) = \sum_{u \in \mathcal{U}_s} CTM_{(n,2)}(u) + \log_2(c_u)$$

The BDM has an extension intended to approximate the Kolmogorov complexity of a graph by applying a two-dimensional variant of the BDM to the graph's binary adjacency matrix $\mathbf{A}$ [183, 184]. In this version the CTM component of the BDM approximation is computed using Turing machines that run on a 2-dimensional tape and produce arrays rather than strings. The BDM approximation of the graph's Kolmogorov complexity $BDM_{(n,2)}(\mathbf{A})$ is computed over a partition of $\mathbf{A}$ into block matrices small enough to have a CTM value. Larger values of n allow for CTM estimates of larger arrays, and, if the CTM is to be trusted, should lead to better BDM approximations [184]. Therefore, for the remainder of this chapter we will replace the notation $BDM_{(n,2)}$ with K_{BDM} where n is assumed to be the largest number of states for which there are CTM values available. Additionally, we will only use the function K_{BDM} in reference to the two-dimensional variant of the BDM.

5.3.4 Kolmogorov Regularisation

As a matter of setting we will consider an unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes. The $N \times N$ adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ has elements $a_{ij} \in \{0, 1\}$. Given a learning algorithm `M` that predicts links in the form of an adjacency matrix, we denote the output of `M` as an $N \times N$ matrix $\tilde{\mathbf{A}}$. In this work `M` will be driven by different types of GNNs. The elements of $\tilde{\mathbf{A}}$ have been mapped to the open interval $(0, 1)$ by the output activation function of `M`. $\tilde{\mathbf{A}}$ will be interpreted as a matrix of Bernoulli parameters where $\tilde{a}_{ij}$ represents the independent probability that there is an edge from node i to node j. The reasoning for treating $\tilde{\mathbf{A}}$ as a matrix of probabilities will be discussed in Section 5.3.4.2. When referring to the Bernoulli random variable parameterized by $\tilde{a}_{ij}$ we write $\tilde{\mathfrak{a}}_{ij}$. Finally, when referring to the matrix of independent Bernoulli random variables parameterized by the values in $\tilde{\mathbf{A}}$, we write $\tilde{\mathfrak{A}}$ - this is an important yet subtle distinction.

5.3.4.1 Regularisation Loss Function

The learning algorithm `M` is trained using $\mathbf{A}_{Train}$, a noisy or restricted view of $\mathbf{A}$, in conjunction with some loss function $\mathcal{L}$. For example, a reasonable choice of $\mathcal{L}$ is the binary cross entropy loss function with weighting to account for a sparsity of edges. Given a learning algorithm that predicts links in a graph, we will define Kolmogorov-regularised functions as the class of loss functions with the form:

$$\hat{\mathcal{L}} = \mathcal{L} + \lambda \cdot \mathbb{E}[K(\tilde{\mathfrak{A}})] \tag{5.1}$$

where $\lambda \in \mathbb{R}^+$ is a weighting hyperparameter and $\mathbb{E}[K(\tilde{\mathfrak{A}})]$ is the expected Kolmogorov complexity of $\tilde{\mathfrak{A}}$. As the Kolmogorov complexity of an object is incomputable, we will use the BDM to produce an approximation $\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ of the expected Kolmogorov complexity of $\tilde{\mathfrak{A}}$. Loss functions that use the BDM approximation will be denoted as:

$$\hat{\mathcal{L}}_{BDM} = \mathcal{L} + \lambda \cdot \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})] \tag{5.2}$$

5.3.4.2 Differentiable Adjustment

Partition the binary adjacency matrix $\mathbf{A}$ into blocks of size $R \times R$ as follows:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1N'} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2N'} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}_{N'1} & \mathbf{A}_{N'2} & \cdots & \mathbf{A}_{N'N'} \end{bmatrix}$$

where $N' = {}^{N}/_{R}$. We have made the mild assumption that N is divisible by R; if this is not the case we can simply pad $\mathbf{A}$ with zeros. We refer to the multiset of all blocks in $\mathbf{A}$ as $\mathcal{A}_{\mathbf{A}} = \{\mathbf{A}_{11}, \mathbf{A}_{21}, \mathbf{A}_{12}, \ldots, \mathbf{A}_{N'N'}\}$ and the set of unique elements in $\mathcal{A}_{\mathbf{A}}$ as $\mathcal{U}_{\mathbf{A}}$. Recall that the BDM approximation of the Kolmogorov complexity of a binary adjacency matrix $\mathbf{A}$ is:

$$K_{BDM}(\mathbf{A}) = \sum_{\mathbf{U} \in \mathcal{U}_{\mathbf{A}}} CTM(\mathbf{U}) + \log_2(c_{\mathbf{U}})$$

where $c_{\mathbf{U}}$ is the number of times a block $\mathbf{U} \in \mathcal{U}_{\mathbf{A}}$ appears in $\mathcal{A}_{\mathbf{A}}$. We must alter the BDM approximation to be differentiable as we would like our regularisation term to be used with gradient-based training algorithms such as backpropagation. This requirement motivates our designation of the model output $\tilde{\mathbf{A}}$ as an adjacency matrix of edge probabilities. By treating $\tilde{\mathfrak{A}}$ as a collection of N^2 independent Bernoulli random variables paramterized by $\tilde{\mathbf{A}}$, we have a regularisation term $\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ that is differentiable with respect to the elements of $\tilde{\mathbf{A}}$. Unfortunately, this decision also introduces an element of computational intractability. Note that each of the N'^2 blocks in $\tilde{\mathfrak{A}}$ has a unique probability mass function over the 2^{R^2} possible binary matrices of size $R \times R$. Therefore, there are $2^{R^2 N'^2} = 2^{N^2}$ unique probabilities to be computed in order to directly determine $\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ (this is also apparent from the fact that there are 2^{N^2} possible realisations of $\tilde{\mathfrak{A}}$).

To sidestep this issue, we adopt a Monte Carlo approach where we sample m times from $\tilde{\mathfrak{A}}$. However, instead of approximating $\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ directly we use the samples to approximate the gradient $\nabla\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$. We can formulate the partial derivative of $\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ with respect to $\tilde{a}_{ij}$ as follows:

$$\begin{aligned}
\frac{\partial \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]}{\partial \tilde{a}_{ij}} &= \frac{\partial}{\partial \tilde{a}_{ij}} \cdot \mathbb{P}(\tilde{\mathfrak{a}}_{ij} = 1) \cdot \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 1] \\
&+ \frac{\partial}{\partial \tilde{a}_{ij}} \cdot \mathbb{P}(\tilde{\mathfrak{a}}_{ij} = 0) \cdot \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 0] \\
&= \frac{\partial}{\partial \tilde{a}_{ij}} \cdot \tilde{a}_{ij} \cdot \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 1] \\
&+ \frac{\partial}{\partial \tilde{a}_{ij}} \cdot (1 - \tilde{a}_{ij}) \cdot \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 0] \\
&= \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 1] - \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}}) | \tilde{\mathfrak{a}}_{ij} = 0]
\end{aligned}$$

Let $\tilde{\mathbf{A}}^{(1)}, \tilde{\mathbf{A}}^{(2)}, \ldots, \tilde{\mathbf{A}}^{(m)}$ denote m binary matrices sampled from $\tilde{\mathfrak{A}}$. For each sample $\tilde{\mathbf{A}}^{(k)}$ we can partition the matrix into blocks of size $R \times R$ and compute a frequency table of all the different blocks in $\mathcal{O}(N^2)$ time. We will use $\tilde{\mathbf{A}}^{(k)}_{ij=1}$ and $\tilde{\mathbf{A}}^{(k)}_{ij=0}$ to denote a sample $\tilde{\mathbf{A}}^{(k)}$ that, regardless of the original value of $\tilde{a}^{(k)}_{ij}$, has element $\tilde{a}^{(k)}_{ij}$ set to 1 or 0, respectively. We also have access to a pre-computed lookup table of CTM values for every possible $R \times R$ binary matrix. Note that $R \ll N$ and is generally set at a fixed value of 4; therefore, in our analysis it will be treated as a constant. Using both the lookup table

and the frequency table, the difference $K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=1}) - K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=0})$ can be computed in constant time. This is accomplished by using the binary string of the values in the $R \times R$ block matrix containing $\tilde{a}^{(k)}_{ij}$ as the index key for both tables, then incrementing and decrementing BDM values based on the existing frequencies. The average value of $K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=1}) - K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=0})$ approaches $\frac{\partial \mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]}{\partial \tilde{a}_{ij}}$ as the sample count increases. Therefore, by computing the value of $K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=1}) - K_{BDM}(\tilde{\mathbf{A}}^{(k)}_{ij=0})$ for each element $\tilde{a}^{(k)}_{ij}$ we effectively sample from the gradient $\nabla\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ in $\mathcal{O}(N^2)$ time.

Let $\nabla\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]^{(k)}$ denote a sample from $\nabla\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]$ and, to simplify notation, denote the sample mean of the gradient as:

$$\bar{\mathbf{G}}_{\tilde{\mathbf{A}},m} = \frac{1}{m}\sum_{k=1}^{m} \nabla\mathbb{E}[K_{BDM}(\tilde{\mathfrak{A}})]^{(k)}$$

In order to incorporate the sample mean of the gradient into the gradient of the loss function we multiply each element in $\tilde{\mathbf{A}}$ with its corresponding element in $\bar{\mathbf{G}}_{\tilde{\mathbf{A}},m}$ and the weighting hyperparameter λ, then add the total sum of these products to the loss function $\mathcal{L}$. Note that despite the notation, the elements of $\bar{\mathbf{G}}_{\tilde{\mathbf{A}},m}$ are treated as constants. Our regularised loss function is summarised below as:

$$\hat{\mathcal{L}}^{*}_{BDM} = \mathcal{L} + \lambda \cdot \mathbf{1}^{T}\Big(\tilde{\mathbf{A}} \odot \bar{\mathbf{G}}_{\tilde{\mathbf{A}},m}\Big)\mathbf{1} \tag{5.3}$$

where $\odot$ denotes element-wise multiplication and $\mathbf{1}$ is the column vector of N ones.

5.3.5 Experiments

In order to assess the performance of our method, we measured the impact of the regularisation term $\lambda \cdot \mathbf{1}^{T}\Big(\tilde{\mathbf{A}} \odot \bar{\mathbf{G}}_{\tilde{\mathbf{A}},m}\Big)\mathbf{1}$ on the ability of standard GNNs to predict links. More specifically, we tested the regularisation term on a graph auto-encoder (GAE) and a variational graph auto-encoder (VGAE) [94]. For both models we followed the architecture described in [94] where the encoders are two-layer graph convolutional networks (GCNs) [95] with 32 and 16 hidden units, respectively. The decoders produce the edge probabilities by computing the sigmoid of the inner product of the latent node embeddings. The base loss functions used with the GAE network was the binary cross-entropy loss with weighting proportional to the ratio of negative to positive labels. The VGAE network used the same loss as a reconstruction term, but included an additional Kullback-Leibler divergence term to measure the discrepancy between the approximation of the posterior and the latent prior, which we defined as an isotropic unit Gaussian. We also tested a different version of the regularisation term where the weights from the pre-computed CTM lookup table were replaced by a single constant value (we used the average value of the entire CTM table for reasons discussed Section 5.3.5.1). This alternate regularisation term, which we

Table 5.1: Overview of the networks used in our experiments.

Network	Node Count	Edge Count	Category
Chameleon	2277	31421	Internet
Chicago	1467	1298	Transportation
Cora	2708	5429	Citation
PDZBase	212	244	Protein Interaction
Political Books	105	441	Purchasing

refer to as constant weight (CW) regularisation, served as a control to elucidate whether improvements in performance stemmed from the CTM (and its purported estimation of Kolmogorov complexity) or the aggregation rule in the BDM.

5.3.5.1 Link Prediction on Real-World Networks

We performed our experimentation on five diverse real-world networks: a network of links between Wikipedia pages on chameleons [136], a road transportation network from Chicago [20, 30, 46], the Cora citation network of scientific publications [178], a protein-protein interaction network from PDZBase [15, 126], and a network of co-purchases of US political books [97]. These networks were chosen to represent a broad range of applications with highly different generating mechanisms. As both our GAE and VGAE models needed a node feature matrix, we used an appropriately sized identity matrix as a dummy input to satisfy the requirement. Additionally, each network was cleaned to be undirected and contain no self-loops.

Experiment Design Our experiment design largely followed that of [93, 94]. We began by separating each of the five networks into training, validation, and testing data sets. The training input was the original adjacency matrix with 80% of the edges retained at random. Because the graph convolutional operator required the adjacency matrix to be updated with self-loops along the diagonal of the matrix, our training label was simply the sum of the training input and the identity matrix. The validation set consisted of half of the 20% of true original edges that were not selected for training along with an equal number of false edges that did not exist in the original graph. All of the true edges and false edges were randomly selected. The test set consisted of the remaining true original edges along with an equal number of random false edges that did not exist in either the original graph or the set of false validation edges. Note that different random splits would naturally have slightly different results.

We randomly initialised both the GAE and the VGAE models using the standard Glorot initialisation procedure [57] and performed multiple trials to account for different initialisations. We employed two well-known metrics for binary classification: area under

Table 5.2: Summary of λ values used for each data set.

Network	Chameleon	Chicago	Cora	PDZBase	Political Books
λ **Value**	5×10^{-7}	1×10^{-5}	4×10^{-7}	1×10^{-4}	3×10^{-5}

Table 5.3: Link prediction results for AUC metric on the GAE architecture.

Network	**GAE**		
	No Reg.	Kol. Reg.	CW Reg.
Chameleon	98.22 ± 0.01	$\mathbf{98.90 \pm 0.02}$	$\mathbf{98.91 \pm 0.02}$
Chicago	76.38 ± 1.21	$\mathbf{88.00 \pm 0.13}$	$\mathbf{88.23 \pm 0.13}$
Cora	81.85 ± 0.48	$\mathbf{82.60 \pm 0.24}$	$\mathbf{83.06 \pm 0.30}$
PDZBase	71.51 ± 2.48	$\mathbf{83.14 \pm 0.44}$	$\mathbf{83.77 \pm 0.56}$
Political Books	83.70 ± 0.41	$\mathbf{88.94 \pm 0.43}$	87.95 ± 0.49

the ROC curve (AUC) and average precision (AP). After splitting the data sets, we established preliminary results for both models without any regularisation over 10 trials on each of the five validation sets. All trials described in this paper were run for $1,000$ epochs. During each trial, we saved the model weights for both the maximum validation AUC and AP scores.

After the λ values were determined, we trained on all five data sets using Kolmogorov regularisation. For both models, we repeated this process for a total of 10 trials per data set, saving the model weights that gave the maximum validation AUC and AP scores for each trial and data set. This entire process was repeated for the constant weight version; however, identical λ values were used to the Kolmogorov regularisation as there was no validation set search. This meant that the constant weight regularisation values could have been even better if they had their own tailored λ values.

Nevertheless, because we chose the constant weight to be equal to the average value of the CTM table, we obtained good enough results on the constant weight regularisation term to fulfil its role, *i.e.* the results were roughly the same as the performance of Kolmogorov regularisation. Finally, we tested all the saved validation models (without regularisation, with Kolmogorov regularisation, and with constant weight regularisation) on the corresponding network test sets. We report the means and standard errors on the test sets for both AUC and AP scores in Tables 5.3, 5.4, 5.5, and 5.6. Values in bold indicate that the highest range of that value according to the given precision of standard error is at least as good as the lowest range of the other two values in the row.

Model training was performed on an Intel i7-4790k CPU with 8GB of RAM and an Nvidia GTX 970 GPU. We note that although the computation of the gradient sample for the regularisation term was done entirely on the CPU, this process should scale well on a GPU. On this machine, each trial of $1,000$ epochs took from approximately 30 seconds

Table 5.4: Link prediction results for AUC metric on the VGAE architecture.

Network	**VGAE**		
	No Reg.	Kol. Reg.	CW Reg.
Chameleon	98.17 ± 0.02	$\mathbf{98.82 \pm 0.02}$	$\mathbf{98.85 \pm 0.02}$
Chicago	82.40 ± 0.79	$\mathbf{88.11 \pm 0.08}$	87.82 ± 0.19
Cora	82.56 ± 0.28	$\mathbf{82.94 \pm 0.30}$	$\mathbf{83.54 \pm 0.34}$
PDZBase	75.28 ± 1.64	$\mathbf{83.89 \pm 0.63}$	$\mathbf{84.79 \pm 0.43}$
Political Books	86.61 ± 0.43	$\mathbf{88.96 \pm 0.33}$	$\mathbf{88.50 \pm 0.39}$

Table 5.5: Link prediction results for AP metric on the GAE architecture.

Network	**GAE**		
	No Reg.	Kol. Reg.	CW Reg.
Chameleon	98.48 ± 0.02	$\mathbf{98.96 \pm 0.02}$	$\mathbf{98.97 \pm 0.01}$
Chicago	78.35 ± 1.03	$\mathbf{86.17 \pm 0.31}$	$\mathbf{86.18 \pm 0.21}$
Cora	86.09 ± 0.23	86.21 ± 0.19	$\mathbf{86.65 \pm 0.14}$
PDZBase	$\mathbf{75.04 \pm 1.42}$	$\mathbf{73.80 \pm 2.25}$	$\mathbf{77.92 \pm 2.28}$
Political Books	80.75 ± 0.45	89.28 ± 0.40	$\mathbf{90.67 \pm 0.32}$

Table 5.6: Link prediction results for AP metric on the VGAE architecture.

Network	**VGAE**		
	No Reg.	Kol. Reg.	CW Reg.
Chameleon	98.52 ± 0.02	$\mathbf{98.88 \pm 0.02}$	$\mathbf{98.91 \pm 0.01}$
Chicago	82.59 ± 1.10	$\mathbf{86.22 \pm 0.15}$	85.94 ± 0.10
Cora	86.26 ± 0.25	$\mathbf{86.76 \pm 0.21}$	$\mathbf{86.79 \pm 0.25}$
PDZBase	$\mathbf{76.02 \pm 1.20}$	$\mathbf{77.16 \pm 2.52}$	$\mathbf{73.33 \pm 2.13}$
Political Books	85.04 ± 0.33	$\mathbf{91.08 \pm 0.25}$	90.54 ± 0.21

(Political Books) to approximately 35 minutes (Cora) depending on the size of the network.

5.4 Discussion and Conclusion

In Tables 5.3, 5.4, 5.5, and 5.6, both the Kolmogorov and constant weight regularisation terms appear to be effective for link prediction tasks on a broad variety of data sets. The performance gains were particularly impressive on the Chameleon, Chicago and Political Books networks (relative to the standard errors) when the graph neural networks were trained with regularisation. Despite already being very high, the results on the Chameleon data set were significantly superior with regularisation as the standard error ranges for this network were very small. The Cora data set displayed some marginal increases, but in comparison to other networks the improvements were not as drastic. Regularisation did have a notable positive effect on the AUC score of the PDZBase network, but the AP score did not improve significantly. The volatility shown on the PDZBase network was likely due to its small edge count.

However, it seems unlikely that any of the gains from the Kolmogorov regularisation were due to the CTM as the constant weight control regularisation performed essentially the same, even without tailored λ values which would have likely further improved its performance. The results of the experiments clearly show some form of regularisation (often both the constant weight regularisation and Kolmogorov regularisation) almost always improved performance, often significantly. This effect was demonstrated across all architectures, metrics, and datasets with the sole exception of the PDZBase dataset measured under the AP metric. This suggests that the chosen data and experimental setup were highly amenable to demonstrating improved link prediction through regularisation.

Additionally, the constant weight approach directly removed the contribution of the CTM to the BDM. Because the CTM directly attempts to estimate the Kolmogorov complexity, it seems safe to conclude that the Kolmogorov regularisation term does not improve results because of an appeal to Kolmogorov complexity. We can offer a few possible explanations for the lack of contribution from the CTM, and thus estimation of Kolmogorov complexity, to the improvements from regularisation. It is possible that the CTM engages in problematic usage of Levin's Coding Theorem - a concern highlighted in an article by Vitányi [164]. Specifically, Vitányi noted that the CTM is concerned with small Kolmogorov complexity values and thus cannot overcome the constant term in the coding theorem which likely is much higher. Zenil [181] issued a rebuttal to this criticism which highlighted the extensive testing of the CTM in [146, 182–184]. Independent to this dialogue, we note that the CTM has been used to demonstrate the simplicity bias phenomenon (supplementary note 7 in [41]); however, Vitányi's criticism is valid and, furthermore, we were not able to obtain notable increases in performance using the CTM

weights. The BDM also has limitations that are independent of the CTM and instead related to its status as an aggregate method. The method of combination dilutes its inputs as it is expanded; the primary original author, in a follow up collaborative paper investigating the properties of the BDM, noted that "in the 'worst case' [the BDM] behaves like Shannon entropy" [184]. It seems reasonable to speculate that such 'limit' behaviour readily manifests in real-world datasets of interest and might be responsible for the comparable performance between regularisation methods.

Another potential explanation is a lack of sensitivity to the differences in estimated Kolmogorov complexity for the CTM. A linear increase in bit size leads to an exponential increase in possible programs, however the regularisation term still punishes this increase in a linear sense. If this latter problem is the true issue, it could be addressed in future work by redesigning the regularisation term to be more sensitive to different CTM weights (which range from about 22 to 36 for $R = 4$). However, as has been suggested, if it is the case that the CTM cannot be reliably used to estimate Kolmogorov complexity due to difficulties with the size of the constant term in Levin's Coding Theorem, then future research should concentrate on alternative engines for preferring structures with simple causal mechanisms. More specifically, we would direct future work and testing to retain the methodological structure but estimate the algorithmic complexity using a different established form of approximation.

Chapter 6

Conclusion

> Science is a mosaic of partial and conflicting visions.
>
> *Freeman Dyson* [45]

As the thesis approaches its finish, one dutifully feels the need to package its contents into a neat, digestible form - a parting gift of the ideas within. In addition to this compendium, the impending sense of closure to several years of work demands a reflection on the value and future of the doctoral research as a whole. The following section presents content and assessments of both sorts on a chapter by chapter basis. We will aim to directly address the research questions that were first posed in the introduction and subsequently answered throughout the body of the thesis.

Chapter 3 The minimum description length principle is a framework for selecting learning models that advocates for the model which minimises the cost of describing the model summed with the cost of encoding the data using the model. The MDL viewpoint has a long history of application to the study of neural networks and was notably used by Hinton and van Camp [74] in a paper which introduced both variational inference for neural networks and the bits back method. Hinton and van Camp's Bayesian approach was motivated by a desire to improve the ability of the network to generalise.

Neural networks and, in particular deep neural networks, are an incredibly effective class of learning algorithms which have achieved unprecedented success on a variety of machine learning tasks [58, 101, 144]. This success seemingly stands in contrast to the large number of parameters typically found in such models. Intuition built from concepts such as the minimum description length principle would suggest that deep neural networks would perform poorly due to over-parametrisation. Blier and Ollivier [16] demonstrated that the description lengths provided by existing methods such as the variational approach were inefficient for explaining this discrepancy in a supervised learning setting and were

outperformed by prequential coding.

Prequential coding, derived from Dawid's prequential approach to statistics [35], is a straightforward MDL method which directly improves based on the given model's ability to generalise from restrictions of the original dataset. More specifically, a sender and a receiver agree to an untrained learning model, the sender codes and transmits a piece of data, then both sender and receiver train their models using this data. This process is repeated until all the data is sent. Capable models, such as neural networks, will typically improve with access to more data and equivalently decrease the coding costs for future data. Prequential coding is particularly well suited to capturing description lengths of deep neural networks as there is no direct encoding of the parameters and, furthermore, such models often show rapid improvement as the training dataset increases in size.

The research I conducted in this chapter broadly investigated whether description lengths of neural networks computed via prequential coding can be improved upon. The primary research question that I sought to answer in this chapter was, "Is it possible to create a better method of determining description lengths of deep neural networks than prequential coding?" Prequential coding is the dominant method in this domain and serves as the basis for the LTCB topping compression algorithm `nncp` [12, 13]. To begin this process we first generalised prequential coding into an approach we called instructional coding. Instructional coding retains the concept of training on intermediate datasets; however, unlike prequential coding which, in its unbatched form, only allows for progressively larger views of the original dataset $\mathcal{D}$, instructional coding allows intermediate datasets to be outputs of loosely restricted functions of $\mathcal{D}$. We noted that other existing powerful compression schemes such as `SReC` [23] can be viewed as instances of instructional coding.

We then asked the question, "can one develop an instructional coding algorithm which approaches the efficiency of prequential coding for supervised learning tasks using deep learning models?" We proceeded to describe an instructional coding algorithm we called prechastic coding which allows intermediate datasets to have false labels in diminishing amounts. This work was the subject of the paper "Prechastic Coding: An Alternative Approach to Neural Network Description Lengths" by myself and Pietro Liò. We traversed these datasets using a given learning model and showed experimentally that if the false labels are chosen in a principled manner (*i.e.* in our experiments a greedy approach), the resulting code lengths can be competitive with prequential coding in a variety of scenarios. Underlying the alternative prechastic coding algorithms we discuss are tools from the compression literature such as relative entropy coding, a topic we revisit in the following chapter. Therefore, the answer to the primary research question for this chapter is that it might be possible for prechastic coding to create shorter neural network description lengths, however further improvements will have to be made if it is to reliably

beat prequential coding. Having established prechastic coding as a promising method for gauging neural network description lengths, we believe there are multiple avenues for future work that focus on increasing efficiency through advanced techniques for the selection of fake and true labels. For example, one particularly interesting direction would be to expand investigations beyond the world of vanilla supervised learning. For instance, contrastive self-supervised learning could serve as an alternative setting for modified versions of coding strategies discussed in this thesis.

Chapter 4 In the previous chapter we examined methods for capturing the intrinsic information content of trained neural networks via an MDL perspective where the data is losslessly communicated. In this chapter we shifted focus and used some of the machinery of the previous chapter to alter the extrinsic flow of information through a trained deep neural network. As mentioned in the introduction we asked the research question, "Can we alter the flow of information in diffusion models to generate conditional images using principals from relative entropy coding?" The work presented in this chapter answers that question in the affirmative as we developed a conditional generative algorithm called `Importance-Guided Diffusion` which creates images conditioned on a reference through pre-trained diffusion networks.

Diffusion probabilistic models were introduced by Sohl-Dickstein et al. [145] and have rapidly become one of the most successful and popular generative methods of the last few years [39, 78, 152]. Diffusion essentially works by training a network to denoise an object from a training set injected with some level of Gaussian noise. Once trained, the network can incrementally and iteratively denoise a random Gaussian sample until the resulting object resembles a sample from the original training set. Training is performed using a variant of the variational lower bound, therefore the loss associated with the non-terminal steps in the reverse diffusion process admit an interpretation as a sample communication cost.

In the paper 'Importance-Guided Diffusion' by myself and Pietro Liò, we leveraged relative entropy coding methods to steer the reverse diffusion process towards an image that need not be from the original training set. Specifically we drew from work by Havasi et al. [70] which described an importance sampling method for sample communication and Flamich et al. [48] which ameliorated the computational burden of the importance sampling method through auxiliary variables. At its core, our algorithm worked by slowly transmitting information from the reference image to the network via a carefully designed importance sampling method. We adjusted the impact of our guidance using both an interpolation hyper-parameter and by adjusting the amount of samples taken; we note, that our algorithm worked at the pixel level for computational purposes and thus is not itself an efficient coding method. In our experiments the results were visually

compelling with guidance at varying levels of controlled degrees clearly evident. Because our algorithm allows for principled conditional generation without retraining, we believe it holds great promise in the current generative ML landscape. We expect future work will expand to different modalities/domains and also investigate further fine tuning of the guidance process. Specifically, we would recommend a more thorough investigation into the qualitative impacts of the hyper-parameter selection.

Chapter 5 Algorithmic complexity and the related fields of algorithmic information theory and algorithmic probability provide a fundamental rigour to topics such as information, randomness, and induction. Not only do they appeal to deeply held intuitions and philosophies, *e.g.* Occam's razor, but they provide a theoretical ultimacy to the field of machine learning, somewhat akin to the relationship between computability and algorithms. Despite the uncomputability of Kolmogorov complexity, there exists a large body of research investigating practical applications in artificial intelligence and machine learning.

In the 2020 paper 'Investigating Estimated Kolmogorov Complexity as a Means of Regularization for Link Prediction' by myself, Ramon Viñas, and Pietro Liò, we studied whether a popular approximation algorithm for Kolmogorov complexity in graphs could be transformed into a regularisation method for link prediction using graph neural networks. We were investigating the research question "Can a measure of Kolmogorov/algorithmic complexity (specifically the BDM) be used to regularize link prediction in graph neural networks? The Block Decomposition Method (BDM), the Kolmogorov approximation method we used, was driven by the Coding Theorem Method (CTM) [147] which estimates the algorithmic probability of short binary strings by examining the frequency of outputs of small state Turing machines. The BDM [184] has successfully demonstrated the ability capture the simplicity bias phenomenon [41]. We explicitly used a version designed for graphs which had been adapted to work on binary adjacency matrices [183, 184].

Link prediction in real-world networks is an active research topic in the fields of network theory and machine learning and has found a broad array of applications [43, 80, 109, 117]. Many of these networks have simple primary generating mechanisms and can be explained by succinct concepts such as preferential attachment or the Watts-Strogatz model [9, 168]. We hypothesised that link prediction algorithms for such real-world networks could benefit from regularisation towards graphs which are algorithmically simple. Specifically we focused on graph neural networks which are an important class of models that have demonstrated great ability for link prediction [94, 172, 174]. The graph-based version of the BDM was integrated into the GNN loss functions using a weighting parameter on the expected Kolmogorov approximation for the prediction. Furthermore, we adopted a standard Monte Carlo approach to estimate the gradient, thus avoiding potential tractability issues.

The resulting Kolmogorov loss function was tested on a suite of real world networks [15, 20, 30, 46, 97, 126, 136, 178] and two GNN models - a GAE and a VGAE [94]. We also tested a control regularisation function where the CTM values were replaced by a single constant value, thus removing any actual estimate of Kolmogorov complexity derived from the CTM. In our experiments the Kolmogorov regularisation performed better than no regularisation, but just as well as the control regularisation. This result was initially surprising as the control did not directly estimate Kolmogorov complexity using a purpose-built method as the CTM version did. This led to the conclusion that any actual influence from the CTM was negligible, and that the primary driver of regularisation was the reward for repeated small patterns (which one might expect in a network with simple generating mechanisms). Consequently, we do not believe the BDM can serve as a meaningful source of Kolmogorov based regularisation for GNN link prediction on the type of networks we evaluated.

Furthermore, very close to the time we conducted our research, a paper on the incomputability of Kolmogorov complexity was published which pointed out methodological flaws in the CTM based on problematic usage of Levin's Coding Theorem [164]. This, in conjunction with our experimental results, leads us to recommend that future work on simplicity based regularisation in link prediction consider alternative approximation methods. In other words, we answer the original research question in the negative for the BDM, although we remain optimistic about future work which attempts to introduce a regulatory bias through means which are more specific to the proposed generating mechanisms.

We would also like to offer a reflection on the field in light of the aforementioned research results. The appeal of algorithmic complexity to the machine learning scientist largely lies in its totality - it is a field which offers comprehensive theories of induction, randomness, and information. Naturally, the promise of what one could accomplish, if they could only meaningfully estimate the Kolmogorov complexity, is too great a temptation for some researchers to resist, despite the limits imposed by incomputability. Principled approaches, results, and frameworks such as simplicity bias [41] or AIXI [82] all offer insights into their respective domains; however, direct attempts to estimate the Kolmogorov complexity for downhill purposes are misguided for reasons other than the methodological flaws mentioned. As pointed out in [164], if we assume data from the natural world primarily contains "effective regularities" (as opposed to mathematical constants, etc.) the results from a good compressor are adequate. Perhaps this is the sanity check needed when embarking on such quests - we are almost always dealing with real-world data generated by real-world computations.

Coda The research conducted during my doctoral studies contributed several novel algorithms for constructing and exploiting various intrinsic and extrinsic measures of information for neural networks. This work encompassed both the classic Shannon-style conceptualisation of information as well as the algorithmic notion of Solomonoff, Kolmogorov, and Chaitin. These contributions have advanced knowledge in fundamental topics concerning compression and description lengths as well as practical topics in generative algorithms and link prediction. As discussed in this conclusion, it is my belief that the frameworks and methodologies presented in this thesis can serve as a fertile ground for future research for improving and understanding relevant topics in neural networks and deep learning.

References

[1] Pieter Adriaans. Information. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2024 edition, 2024.

[2] Eirikur Agustsson and Lucas Theis. Universally quantized neural compression. In *NeurIPS*, 2020.

[3] Saeed Anwar, Salman Khan, and Nick Barnes. A Deep Journey into Super-resolution: A Survey. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.

[4] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *ICLR*. OpenReview.net, 2017.

[5] Johannes Ballé, Philip A. Chou, David Minnen, Saurabh Singh, Nick Johnston, Eirikur Agustsson, Sung Jin Hwang, and George Toderici. Nonlinear transform coding. *IEEE J. Sel. Top. Signal Process.*, 15(2):339–353, 2021.

[6] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S. Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *CoRR*, abs/2208.09392, 2022.

[7] Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *CVPR Workshops*, pages 843–852. IEEE, 2023.

[8] Albert-László Barabási. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.

[9] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. ISSN 0036-8075. doi: 10.1126/science.286.5439.509. URL `https://science.sciencemag.org/content/286/5439/509`.

[10] Andrew Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE transactions on information theory*, 44(6): 2743–2760, 1998.

[11] Andrew R Barron and Thomas M Cover. Minimum complexity density estimation. *IEEE transactions on information theory*, 37(4):1034–1054, 1991.

[12] Fabrice Bellard. Lossless Data Compression with Neural Networks. Technical report, Self-published, May 2019. URL `https://bellard.org/nncp/nncp.pdf`.

[13] Fabrice Bellard. NNCP v2: Lossless Data Compression with Transformer. Technical report, Self-published, Feb 2021. URL `https://bellard.org/nncp/nncp_v2.1.pdf`.

[14] Charles H Bennett, Peter W Shor, John A Smolin, and Ashish V Thapliyal. Entanglement-assisted capacity of a quantum channel and the reverse shannon theorem. *IEEE transactions on Information Theory*, 48(10):2637–2655, 2002.

[15] Thijs Beuming, Lucy Skrabanek, Masha Y Niv, Piali Mukherjee, and Harel Weinstein. Pdzbase: a protein–protein interaction database for pdz-domains. *Bioinformatics*, 21(6):827–828, 2005.

[16] Léonard Blier and Yann Ollivier. The description length of deep learning models. In *NeurIPS*, pages 2220–2230, 2018.

[17] Jorg Bornschein, Francesco Visin, and Simon Osindero. Small data, big decisions: Model selection in the small-data regime. In *International conference on machine learning*, pages 1035–1044. PMLR, 2020.

[18] Jörg Bornschein, Silvia Chiappa, Alan Malek, and Nan Rosemary Ke. Prequential MDL for causal structure learning with neural networks. *CoRR*, abs/2107.05481, 2021. URL `https://arxiv.org/abs/2107.05481`.

[19] Jörg Bornschein, Yazhe Li, and Marcus Hutter. Sequential learning of neural networks for prequential MDL. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/pdf?id=dMMPUvNSYJr`.

[20] D. E. Boyce, K. S. Chon, M. E. Ferris, Y. J. Lee, K-T. Lin, and R. W. Eash. Implementation and evaluation of combined models of urban travel and location on a sketch planning network. *Chicago Area Transportation Study*, pages xii + 169, 1985.

[21] Michael Burrows. A block-sorting lossless data compression algorithm. *SRS Research Report*, 124, 1994.

[22] Cristian S. Calude and Michael J. Dinneen. Exact approximations of omega numbers. *Int. J. Bifurc. Chaos*, 17(6):1937–1954, 2007.

[23] Sheng Cao, Chao-Yuan Wu, and Philipp Krähenbühl. Lossless image compression through super-resolution. *CoRR*, abs/2004.02872, 2020. URL `https://arxiv.org/abs/2004.02872`.

[24] Miguel de Cervantes. *Don Quixote*. Harper Collins, 2009. Translated by Edith Grossman.

[25] Gregory J Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

[26] Gregory J Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM (JACM)*, 16(1):145–159, 1969.

[27] Gregory J. Chaitin. A Theory of Program Size Formally Identical to Information Theory. *J. ACM*, 22(3):329–340, July 1975.

[28] Gregory J Chaitin. On the intelligibility of the universe and the notions of simplicity, complexity and irreducibility. *arXiv preprint math/0210035*, 2002.

[29] Sourav Chatterjee and Persi Diaconis. The sample size required in importance sampling. *The Annals of Applied Probability*, 28(2):1099–1135, 2018.

[30] Chicago Network Dataset. Chicago network dataset – KONECT, April 2017. URL `http://konect.uni-koblenz.de/networks/tntp-ChicagoRegional`.

[31] Hyungjin Chung, Byeongsu Sim, Dohoon Ryu, and Jong Chul Ye. Improving diffusion models for inverse problems using manifold constraints. In *NeurIPS*, 2022.

[32] Hyungjin Chung, Jeongsol Kim, Michael Thompson McCann, Marc Louis Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. In *ICLR*. OpenReview.net, 2023.

[33] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[34] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.

[35] A. P. Dawid. Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach. *Journal of the Royal Statistical Society. Series A (General)*, 147(2):278–292, 1984.

[36] A. P. Dawid. Prequential Data Analysis. *Lecture Notes-Monograph Series*, 17: 113–126, 1992.

[37] A. P. Dawid. Prequential Statistics, Nov 2007. URL `http://131.111.150.181/talk/index/8866`.

[38] Jean-Paul Delahaye and Hector Zenil. Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Applied Mathematics and Computation*, 219(1):63–77, 2012.

[39] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, pages 8780–8794, 2021.

[40] Kamaludin Dingle. *Probabilistic Bias in Genotype-Phenotype Maps*. Ph.d. dissertation, University of Oxford, 2014.

[41] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 9(1):761, 2018.

[42] Kamaludin Dingle, Rafiq Kamal, and Boumediene Hamzi. A note on a priori forecasting and simplicity bias in time series. *Physica A: Statistical Mechanics and Its Applications*, 609:128339, 2023.

[43] Yuxiao Dong, Jie Tang, Sen Wu, Jilei Tian, Nitesh V. Chawla, Jinghai Rao, and Huanhuan Cao. Link prediction and recommendation across heterogeneous social networks. In Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu, editors, *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 181–190. IEEE Computer Society, 2012. doi: 10.1109/ICDM.2012.140. URL `https://doi.org/10.1109/ICDM.2012.140`.

[44] Jarek Duda. Asymmetric numeral systems. *arXiv preprint arXiv:0902.0271*, 2009.

[45] Freeman J Dyson. *The scientist as rebel*. NYRB Classics, New York, NY, December 2006.

[46] R. W. Eash, K. S. Chon, Y. J. Lee, and D. E. Boyce. Equilibrium traffic assignment on an aggregated highway network for sketch planning. *Transportation Research Record*, 994:30–37, 1983.

[47] W. Feller. On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the [First] Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, CA, USA, 1949. The Regents of the University of California.

[48] Gergely Flamich, Marton Havasi, and José Miguel Hernández-Lobato. Compressing images by encoding their latent representations with relative entropy coding. In *NeurIPS*, 2020.

[49] Gergely Flamich, Stratis Markou, and José Miguel Hernández-Lobato. Fast relative entropy coding with a* coding. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 6548–6577. PMLR, 2022.

[50] Ian Flood and Paris DL Flood. Intelligent control of construction manufacturing processes using deep reinforcement learning. In *SIMULTECH*, pages 112–122, 2022.

[51] Paris Flood and Pietro Lio. Importance-guided diffusion. In *NeurIPS 2023 Workshop on Diffusion Models*, 2023.

[52] Paris DL Flood, Ramon Viñas, and Pietro Liò. Investigating estimated kolmogorov complexity as a means of regularization for link prediction. In *2020 NeurIPS Causal Discovery & Causality-Inspired Machine Learning Workshop*, 2020.

[53] Paris Dominic Louis Flood and Pietro Lio. Prechastic coding: An alternative approach to neural network description lengths. In *Workshop on Machine Learning and Compression, NeurIPS 2024*, 2024.

[54] Brendan J Frey. *Bayesian networks for pattern classification, data compression, and channel coding*. PhD thesis, University of Toronto, 1997.

[55] Brendan J. Frey and Geoffrey E. Hinton. Free energy coding. In *Data Compression Conference*, pages 73–81. IEEE Computer Society, 1996.

[56] Péter Gács. On the symmetry of algorithmic information. In *Soviet Math. Dokl.*, volume 15, pages 1477–1480, 1974.

[57] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

[58] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[59] Vivek K Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5):9–21, 2001.

[60] Alexandros Graikos, Nikolay Malkin, Nebojsa Jojic, and Dimitris Samaras. Diffusion models as plug-and-play priors. In *NeurIPS*, 2022.

[61] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[62] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `https://cvxr.com/cvx`, March 2014.

[63] Alex Graves. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems*, volume 24, 2011.

[64] Peter Grünwald. Model selection based on minimum description length. *Journal of mathematical psychology*, 44(1):133–152, 2000.

[65] Peter Grünwald. Minimum description length tutorial. *Advances in minimum description length: Theory and applications*, 5:1–80, 2005.

[66] Peter D Grünwald. *The minimum description length principle*. MIT press, 2007.

[67] Peter D. Grünwald and Paul M. B. Vitányi. Algorithmic information theory. *CoRR*, abs/0809.2754, 2008.

[68] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[69] Prahladh Harsha, Rahul Jain, David McAllester, and Jaikumar Radhakrishnan. The communication complexity of correlation. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 10–23. IEEE, 2007.

[70] Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *ICLR (Poster)*. OpenReview.net, 2019.

[71] Santiago Hernández-Orozco, Hector Zenil, Jürgen Riedel, Adam Uccello, Narsis A Kiani, and Jesper Tegnér. Algorithmic probability-guided supervised machine learning on non-differentiable spaces. *arXiv preprint arXiv:1910.02758*, 2019.

[72] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, pages 6626–6637, 2017.

[73] Geoffrey E Hinton and Steven J Nowlan. How learning can guide evolution. *Complex systems*, 1(3):495–502, 1987.

[74] Geoffrey E. Hinton and Drew van Camp. Keeping the Neural Networks Simple By Minimizing the Description Length of the Weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, page 5–13. Association for Computing Machinery, Aug 1993.

[75] Geoffrey E Hinton and Richard Zemel. Autoencoders, Minimum Description Length and Helmholtz Free Energy. In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993.

[76] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *CoRR*, abs/2207.12598, 2022.

[77] Jonathan Ho, Evan Lohn, and Pieter Abbeel. Compression with flows via local bits-back coding. In *NeurIPS*, pages 3874–3883, 2019.

[78] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[79] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, Nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

[80] Petter Holme and Mikael Huss. Role-similarity based functional prediction in networked systems: application to the yeast proteome. *Journal of the Royal Society Interface*, 2(4):327–333, 2005.

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

[82] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.

[83] Marcus Hutter. The Hutter Prize, n.d. URL `http://prize.hutter1.net/`.

[84] Andrew J Jensen, Paris DL Flood, Lindsey S Palm-Vlasak, William S Burton, Amélie Chevalier, Paul J Rullkoetter, and Scott A Banks. Joint track machine learning: an autonomous method of measuring total knee arthroplasty kinematics from single-plane x-ray images. *The Journal of Arthroplasty*, 38(10):2068–2074, 2023.

[85] J Jiang. Image compression with neural networks–a survey. *Signal processing: image Communication*, 14(9):737–760, 1999.

[86] Zhijing Jin, Julius von Kügelgen, Jingwei Ni, Tejas Vaidhya, Ayush Kaushal, Mrinmaya Sachan, and Bernhard Schölkopf. Causal Direction of Data Collection Matters: Implications of Causal and Anticausal Learning for NLP. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, page 9499–9513. Association for Computational Linguistics, Nov 2021.

[87] Iain G Johnston, Kamaludin Dingle, Sam F Greenbury, Chico Q Camargo, Jonathan PK Doye, Sebastian E Ahnert, and Ard A Louis. Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proceedings of the National Academy of Sciences*, 119(11):e2113883119, 2022.

[88] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *Advances in Neural Information Processing Systems*, volume 34, pages 21696–21707, 2021.

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

[90] Diederik P. Kingma and Ruiqi Gao. Understanding the diffusion objective as a weighted integral of elbos. *CoRR*, abs/2303.00848, 2023.

[91] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[92] Friso H. Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3408–3417. PMLR, 2019.

[93] T Kipf. *Deep learning with graph-structured representations*. PhD thesis, University of Amsterdam, 2020.

[94] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[95] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

[96] Andrei N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):3–11, 1965.

[97] V. Krebs. Political books network data. `http://www-personal.umich.edu/~mejn/netdata/`, 2004. Accessed: 2020-06-01.

[98] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, Apr 2009.

[99] G. G. Langdon. An Introduction to Arithmetic Coding. *IBM Journal of Research and Development*, 28(2):135–149, Mar 1984.

[100] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

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

[102] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

[103] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. ISBN 978-1107077232. URL `http://www.mmds.org/`.

[104] Annick Lesne. Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics. *Mathematical Structures in Computer Science*, 24(3):e240311, 2014.

[105] L. A. Levin. Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems of Information Transmission*, 10(3): 30–35, 1974.

[106] Cheuk Ting Li and Abbas El Gamal. Strong functional representation lemma and applications to coding theorems. *IEEE Trans. Inf. Theory*, 64(11):6967–6978, 2018.

[107] Ming Li and Paul MB Vitányi. Learning simple concepts under simple distributions. *SIAM Journal on Computing*, 20(5):911–935, 1991.

[108] Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

[109] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[110] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR (Poster)*. OpenReview.net, 2019.

[111] Charles Lovering, Rohan Jha, Tal Linzen, and Ellie Pavlick. Predicting Inductive Biases of Pre-Trained Models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[112] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. Version 7.2 (fourth printing).

[113] Chris J. Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *NIPS*, pages 3086–3094, 2014.

[114] M. M. Hassan Mahmud and Sylvian R. Ray. Transfer learning using kolmogorov complexity: Basic theory and empirical evaluations. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 985–992. Curran Associates, Inc., 2007. URL `https://proceedings.neurips.cc/paper/2007/hash/b83aac23b9528732c23cc7352950e880-Abstract.html`.

[115] Matt Mahoney. Large Text Compression Benchmark, n.d. URL `http://mattmahoney.net/dc/text.html`.

[116] Per Martin-Löf. The definition of random sequences. *Information and control*, 9(6): 602–619, 1966.

[117] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4), December 2016. ISSN 0360-0300. doi: 10.1145/3012704. URL `https://doi.org/10.1145/3012704`.

[118] John Maynard Smith and Eörs Szathmáry. *The origins of life: From the birth of life to the origin of language*. Oxford University Press, 1999.

[119] Chris Mingard, Joar Skalse, Guillermo Valle Pérez, David Martínez-Rubio, Vladimir Mikulik, and Ard A. Louis. Neural networks are a priori biased towards boolean functions with low entropy. *CoRR*, abs/1909.11522, 2019. URL `http://arxiv.org/abs/1909.11522`.

[120] Chris Mingard, Henry Rees, Guillermo Valle Pérez, and Ard A. Louis. Do deep neural networks have an inbuilt occam's razor? *CoRR*, abs/2304.06670, 2023. doi: 10.48550/ARXIV.2304.06670. URL `https://doi.org/10.48550/arXiv.2304.06670`.

[121] Marvin Minsky. The Limits of Understanding. Panel discussion at The World Science Festival, 2010. Available at: `https://www.youtube.com/watch?v=DfY-DRsE86s`.

[122] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, 2021.

[123] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. GLIDE: towards photorealistic image generation and editing with text-guided diffusion models. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 16784–16804. PMLR, 2022.

[124] Heinz Pagels. *The dreams of reason*. Simon & Schuster, Inc., 1989.

[125] Richard Clark Pasco. *Source Coding Algorithms for Fast Data Compression*. PhD thesis, Stanford University, 1976.

[126] PDZBase Network Dataset. Pdzbase network dataset – KONECT, April 2017. URL `http://konect.uni-koblenz.de/networks/maayan-pdzbase`.

[127] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. Rissanen Data Analysis: Examining Dataset Characteristics via Description Length. In *Proceedings of the 38th International Conference on Machine Learning*, page 8500–8513. PMLR, Jul 2021.

[128] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 2021.

[129] Tibor Rado. On non-computable functions. *Bell System Technical Journal*, 41(3): 877–884, 1962.

[130] J. J. Rissanen. Generalized Kraft Inequality and Arithmetic Coding. *IBM Journal of Research and Development*, 20(3):198–203, May 1976.

[131] J. J. Rissanen and G. G. Langdon. Arithmetic Coding. *IBM Journal of Research and Development*, 23(2):149–162, Mar 1979.

[132] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[133] Jorma Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information theory*, 30(4):629–636, 1984.

[134] Jorma Rissanen. Stochastic complexity and modeling. *The annals of statistics*, pages 1080–1100, 1986.

[135] Jorma Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society: Series B (Methodological)*, 49(3):223–239, 1987.

[136] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2019. URL `https://github.com/benedekrozemberczki/datasets#wikipedia-article-networks`.

[137] Yangjun Ruan, Karen Ullrich, Daniel Severo, James Townsend, Ashish Khisti, Arnaud Doucet, Alireza Makhzani, and Chris J. Maddison. Improving lossless compression rates via monte carlo bits-back coding. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9136–9147. PMLR, 2021. URL `http://proceedings.mlr.press/v139/ruan21a.html`.

[138] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NIPS*, pages 2226–2234, 2016.

[139] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/J.NEUNET.2014.09.003. URL `https://doi.org/10.1016/j.neunet.2014.09.003`.

[140] Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857 – 873, 1997. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(96)00127-X. URL `http://www.sciencedirect.com/science/article/pii/S089360809600127X`.

[141] Daniel Severo, James Townsend, Ashish J Khisti, Alireza Makhzani, and Karen Ullrich. Your dataset is a multiset and you should compress it like one. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

[142] C. E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, Jul 1948. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[143] Larry A. Shepp and Benjamin F. Logan. The fourier reconstruction of a head section. *IEEE Transactions on Nuclear Science*, NS-21(3):21–43, Jun 1974.

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

[145] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2256–2265. JMLR.org, 2015.

[146] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. Correspondence and independence of numerical evaluations of algorithmic information measures. *Computability*, 2:125–140, 2013.

[147] Fernando Soler-Toscano, Hector Zenil, Jean-Paul Delahaye, and Nicolas Gauvrit. Calculating kolmogorov complexity from the output frequency distributions of small turing machines. *PLOS ONE*, 9(5):1–18, 05 2014. doi: 10.1371/journal.pone.0096223. URL `https://doi.org/10.1371/journal.pone.0096223`.

[148] R. J. Solomonoff. A preliminary report on a general theory of inductive inference. Technical Report ZTB-138, Zator Company, Cambridge, Mass., November 1960.

[149] Ray J Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[150] Ray J Solomonoff. A formal theory of inductive inference. part ii. *Information and control*, 7(2):224–254, 1964.

[151] Ray J. Solomonoff. The discovery of algorithmic probability: A guide for the programming of true creativity. In *EuroCOLT*, volume 904 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 1995.

[152] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*. OpenReview.net, 2021.

[153] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, pages 11895–11907, 2019.

[154] Lucas Theis and Noureldin Y. Ahmed. Algorithms for the communication of samples. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 21308–21328. PMLR, 2022.

[155] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *ICLR (Poster)*. OpenReview.net, 2017.

[156] Lucas Theis, Tim Salimans, Matthew D. Hoffman, and Fabian Mentzer. Lossy compression with gaussian diffusion. *CoRR*, abs/2206.08889, 2022. doi: 10.48550/ARXIV.2206.08889. URL `https://doi.org/10.48550/arXiv.2206.08889`.

[157] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015. URL `http://arxiv.org/abs/1503.02406`.

[158] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[159] Peter Toft. *The Radon Transform - Theory and Implementation*. PhD thesis, Department of Mathematical Modelling, Technical University of Denmark, Jun 1996.

[160] James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *ICLR (Poster)*. OpenReview.net, 2019.

[161] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[162] Guillermo Valle-Pérez, Chico Q. Camargo, and Ard A. Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=rye4g3AqFm`.

[163] 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*, volume 30. Curran Associates, Inc., 2017. URL `https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

[164] Paul Vitányi. How incomputable is kolmogorov complexity? *Entropy*, 22(4):408, 2020.

[165] Chris S. Wallace. Classification by minimum-message-length inference. In *ICCI*, volume 468 of *Lecture Notes in Computer Science*, pages 72–81. Springer, 1990.

[166] Weilun Wang, Jianmin Bao, Wengang Zhou, Dongdong Chen, Dong Chen, Lu Yuan, and Houqiang Li. Semantic image synthesis via diffusion models. *CoRR*, abs/2207.00050, 2022.

[167] Yinhuai Wang, Jiwen Yu, and Jian Zhang. Zero-shot image restoration using denoising diffusion null-space model. In *ICLR*. OpenReview.net, 2023.

[168] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

[169] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17 (06):8–19, 1984.

[170] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6):520–540, Jun 1987.

[171] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Kegan Paul Trench Trubner, London, 1922.

[172] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[173] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[174] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[175] Ting-Bing Xu, Peipei Yang, Xu-Yao Zhang, and Cheng-Lin Liu. Margin-Aware Binarized Weight Networks for Image Classification. In *Image and Graphics: 9th International Conference, ICIG 2017*, pages 590–601. Springer, 2017.

[176] Ruihan Yang and Stephan Mandt. Lossy image compression with conditional diffusion models. *CoRR*, abs/2209.06950, 2022.

[177] Yibo Yang, Stephan Mandt, Lucas Theis, et al. An introduction to neural data compression. *Foundations and Trends® in Computer Graphics and Vision*, 15: 113–200, 2023.

[178] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016. URL https://github.com/kimiyoung/planetoid/raw/master/data.

[179] King Fai Yeh, Paris Flood, William Redman, and Pietro Liò. Learning linear embeddings for non-linear network dynamics with koopman message passing. *arXiv preprint arXiv:2305.09060*, 2023.

[180] Dani Yogatama, Cyprien de Masson d'Autume, Jerome T. Connor, Tomás Kociský, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, and Phil Blunsom. Learning and evaluating general linguistic intelligence. *CoRR*, abs/1901.11373, 2019. URL http://arxiv.org/abs/1901.11373.

[181] Hector Zenil. A review of methods for estimating algorithmic complexity: Options, challenges, and new directions. *Entropy*, 22(6):612, 2020. doi: 10.3390/E22060612. URL https://doi.org/10.3390/e22060612.

[182] Hector Zenil, Fernando Soler-Toscano, Kamaludin Dingle, and Ard A Louis. Correlation of automorphism group size and topological properties with program-size complexity evaluations of graphs and complex networks. *Physica A: Statistical Mechanics and its Applications*, 404:341–358, 2014.

[183] Hector Zenil, Fernando Soler-Toscano, Jean-Paul Delahaye, and Nicolas Gauvrit. Two-dimensional kolmogorov complexity and an empirical validation of the coding theorem method by compressibility. *PeerJ Computer Science*, 1:e23, 2015.

[184] Hector Zenil, Santiago Hernández-Orozco, Narsis A. Kiani, Fernando Soler-Toscano, Antonio Rueda-Toicen, and Jesper Tegnér. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *Entropy*, 20(8), 2018. ISSN 1099-4300. doi: 10.3390/e20080605. URL https://www.mdpi.com/1099-4300/20/8/605.

[185] Hector Zenil, Narsis A. Kiani, Francesco Marabita, Yue Deng, Szabolcs Elias, Angelika Schmidt, Gordon Ball, and Jesper Tegnér. An algorithmic information calculus for causal discovery and reprogramming systems. *iScience*, 19:1160 – 1172, 2019. ISSN 2589-0042. doi: https://doi.org/10.1016/j.isci.2019.07.043. URL http://www.sciencedirect.com/science/article/pii/S2589004219302706.

[186] Hector Zenil, Narsis A Kiani, Allan A Zea, and Jesper Tegnér. Causal deconvolution by algorithmic generative models. *Nature Machine Intelligence*, 1(1):58–66, 2019.

[187] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun.*

ACM, 64(3):107–115, 2021. doi: 10.1145/3446776. URL https://doi.org/10.1145/3446776.

[188] Shifeng Zhang, Ning Kang, Tom Ryder, and Zhenguo Li. iflow: Numerically invertible flows for efficient lossless compression via a uniform coder. In *NeurIPS*, pages 5822–5833, 2021.

[189] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.

[190] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.