Resource Constraints on Practical Minimum Description Lengths of Neural Networks

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The relationship between compression and learning has long been a rich area of research, inviting analysis from a variety of methodologies including the minimum description length (MDL) principle. Recent work has shown that the MDL technique known as online coding produces state of the art description lengths when applied to deep neural networks. MDL results have important implications for the performance and generalization properties of neural networks. However, in this paper we demonstrate that computing realizable online codes is, in practice, highly dependent on outside factors that are independent of the choice of neural network architecture, training data, and training procedure. In particular, we show that such codes are extremely sensitive to resource constraints on their computation. This suggests that real-world computations of neural network online codes are strongly influenced by the parameters of the coding process itself - an undesirable property for the MDL approach.

1 Introduction

For decades, computer scientists have been interested in the following question: "What is the shortest, lossless description length of an object?" This question is at the heart of the field of algorithmic information theory [1–5] where it is captured by the concept of Kolmogorov complexity. The Kolmogorov complexity of a binary string is simply the length of the shortest computer program which outputs that string. This is an attractive concept, not only because it is, in a sense, the ultimate form of lossless compression, but also due to its utility when developing rigorous foundations for inductive inference [6, 7].

The concepts of compression and Kolmogorov complexity are deeply intertwined with the study of machine learning and artificial intelligence [8–10]. Unfortunately, the Kolmogorov complexity of an object is fundamentally uncomputable [7]. Nevertheless, compression remains a popular topic within the recent machine learning literature, particularly in regard to deep neural networks. Within this line of research one can find a variety of motivations from storage and energy consumption reduction [11, 12] to implications for generalization [13].

A popular setting for the compression analysis of learning algorithms is the minimum description length (MDL) principle [14]. Informally, the MDL principle advocates that in accordance with Occam's razor one should select a model that minimizes the description length of both itself and the data (see Section 2.3). The importance of the MDL viewpoint is underscored by its relevance to generalization bounds [15] as well as its central, motivating role in Hinton and Van Camp's seminal work on variational methods [16].

Recent work has shown that online coding (also called prequential coding) obtains the smallest compression sizes out of the major approaches to MDL for deep neural networks performing supervised learning [17]. Online coding is an incremental approach to the MDL principle that uses progressively better trained models to compress label data (for a full exposition see Section 2.3.3). Blier and Ollivier found that online codes were significantly shorter than codes based on variational methods and advanced network compression schemes [17].

Considering the compactness of online codes for deep neural networks, it is easy to see the potential impact of these results on the aforementioned MDL applications. Consequently, a natural question to ask is: Given a network architecture, training scheme, and dataset, how does one compute the optimal realizable online code length? In particular, how might one compute online code lengths under practical considerations such as time constraints? In their paper, Blier and Ollivier leave these questions unanswered, instead relying on a heuristic approach [17]; however, as others have noted, computing optimal online codes is expensive [18].

In this work we make the following contributions: First, we introduce and develop the realizable online code length minimization problem for neural networks under a general, constrained setting. Second, we show that the non-convex continuous version of the core optimization problem is, in a sense, equivalent to a problem from a much more manageable optimization class known as 'difference of convex' programming. Lastly, through this approach we are able to experimentally demonstrate that the real-world computed online code length for a given architecture, dataset and training regimen is highly sensitive to computing time constraints.

Determining the minimum description length of a neural network is, in principle, an attempt to capture an intrinsic feature of the architecture and training regimen for a given dataset. However, we show that realizable computations of some of the smallest MDL codes (i.e. online codes) are entirely at the whim of external factors (i.e. computing resources). This raises questions about the value of such MDL codes in practice.

The remainder of this paper is organized as follows: Section 2 introduces necessary background information to understand online coding and frame its efficiency. Section 3 builds the problem structure for computing optimized online codes and provides analysis about the idealized case. Section 4 develops and motivates the constrained problem, shows how to develop a workable reformulation of the non-convex continuous case, and provides experimental results and discussion on the sensitivity to resource constraints. Section 5 concludes the paper.

2 Background

Let us begin with an informal description of our motivating problem within the context of supervised learning. Consider the following two entities: a sender, $\mathfrak{S}$, and a receiver, $\mathfrak{R}$. Both entities have access to computing resources as well as a finite set of input objects (for example, a collection of images). Each input object has exactly one label; however, only $\mathfrak{S}$ knows these labels initially. Our fundamental question is, how can $\mathfrak{S}$ communicate the correct labels for all of the input objects to $\mathfrak{R}$ using as little information as possible?

Initially, the solution to this question might appear to be nothing more than a trivial application of information theory. For example, $\mathfrak{S}$ can simply employ arithmetic coding based on the frequency distribution of the labels in the data and then communicate the stream code to $\mathfrak{R}$ (assuming the, generally much smaller, information about label frequencies has been sent beforehand). However, recall that both entities have access to computing resources. $\mathfrak{S}$ can use these resources to create an advanced prediction model trained using supervised learning on the input-label pairs. The information necessary to recreate the trained model can then be transmitted to $\mathfrak{R}$ along with codes to translate the model outputs to the correct labels. As we shall see, under certain circumstances the total size of this information is significantly less than codes based solely on naive label frequencies.

2.1 Notation

Let $\mathcal{D} := \{(x_1, y_1), \ldots, (x_N, y_N)\}$ denote a classification dataset where $x \in \mathcal{X}$ are input objects and $y \in \mathcal{Y}$ are labels. The set of labels $\mathcal{Y}$ is finite with size $|\mathcal{Y}| = K$ classes. We define $x_{j:l} := (x_j, x_{j+1}, \ldots, x_{l-1}, x_l)$ and extend the corresponding notation to the labels.

A model p_θ is an instance of a model class $\mathcal{M} := \{p_\theta : \theta \in \Theta\}$ corresponding to a particular parameter vector $\theta \in \Theta$. Each model output is defined as a conditional probability distribution

$p_\theta(y|x)$ over the K label classes given x as the input to model p_θ. Each model is trained according to a training regimen r_ω, which is an instance of a training regimen class $\mathcal{R} := \{r_\omega : \omega \in \Omega\}$ corresponding to a particular hyperparameter vector $\omega \in \Omega$. The training regimen class encompasses concepts that, if included, are decided before training (e.g. stopping criterion, dropout rate).

2.2 Coding Rudiments

Consider a dataset $\mathcal{D}$ consisting of N independent data samples from a classification task, generated by a model p. There exists a lossless code for the labels with a length no more than two bits greater than the idealized length $L_p(y_{1:N}|x_{1:N}) := -\sum_{i=1}^{N} \log_2 p(y_i|x_i)$.

This result is due to the well-known concept of arithmetic coding [6, 19]. Throughout the remainder of this paper we generally refer to $L_p(y_{1:N}|x_{1:N})$ as the label code length for a model p and dataset $\mathcal{D}$. However, in later sections we shall revisit the difference between realizable and idealized codes.

Note that the length of the label code is $L_p(y_{1:N}|x_{1:N}) = N \cdot \log_2(K)$ when the model p is a uniform probability distribution over the K classes. As our trained model p_θ increases in accuracy, the length of the label code becomes significantly less than this value. However, the presumably sophisticated trained model must now be communicated to $\mathfrak{R}$ as well.

2.3 Minimum Description Length Principle

The trade-off between the size of the trained model's description and the size of the data when encoded by the trained model is at the heart of the hypothesis selection criterion known as the minimum description length principle. Informally, crude MDL advocates that one should select the hypothesis (model) H which minimizes the sum $L(H) + L(\mathcal{D}|H)$, where $L(H)$ is the length of the description of the hypothesis and $L(\mathcal{D}|H)$ is the length of the description of the data when encoded with the hypothesis [20]. In our probabilistic supervised learning setting, the description length of the encoded data given a model p_θ is $L(\mathcal{D}|p_\theta) = -\sum_{i=1}^{N} \log_2 p_\theta(y_i|x_i)$.

The topic of model length description will be discussed shortly. For a complete treatment of MDL, including concepts such as refined MDL and ideal MDL, we refer the reader to the following references [20, 7]. We will now review several key coding schemes from the MDL literature that are relevant for neural network based approaches to the problem outlined in the beginning of this section.

2.3.1 Two-Part Codes

Assume that both the sender, $\mathfrak{S}$, and the receiver, $\mathfrak{R}$, have agreed before transmission to use the same model class $\mathcal{M}$ (e.g. a specific neural network architecture) for label prediction. Because $\mathfrak{S}$ has access to the data labels, it can learn a trained model p_θ that allows for a compressed encoding of the data. In a *two-part code* the trained model parameters θ and the label encoding are communicated to $\mathfrak{R}$ separately. Formally, let Γ represent a parameter encoding scheme whose domain is the model class $\mathcal{M}$, and denote $L_\Gamma(\theta)$ as the parameter code length of model $p_\theta \in \mathcal{M}$ under Γ. Then the corresponding two-part code length is defined as $L_\Gamma(\theta) - \sum_{i=1}^{N} \log_2 p_\theta(y_i|x_i)$.

A particularly naive neural network parameter code Γ_{naive} is to transmit each weight and bias as a 32-bit floating point number. The resulting parameter code length $L_{\Gamma_{\text{naive}}}$ is huge, even for small neural networks. For example, consider a fully-connected multi-layer perceptron with a single hidden layer of size 64. When applied to the MNIST dataset [21] this network has a naive parameter code length of approximately 1.628 Mbits. In contrast, the worst-case uniform coding of the 60,000 MNIST training labels is only about 199 kbits. There are many examples of more advanced network compression strategies such as pipelines based on pruning, quantization, and Huffman coding [22], or random parameter subspace training [23]. However, online coding can outperform even advanced network compression strategies [17] (also see Section 2.3.4).

2.3.2 Variational Codes

In addition to the choice of model class $\mathcal{M}$, say that $\mathfrak{S}$ and $\mathfrak{R}$ have also previously agreed to a choice of prior α over the parameter space Θ. When using Bayesian inference for neural networks we are often interested in calculating the posterior distribution of the parameters given the data. Because this calculation is generally intractable, a common strategy is to use a variational approximation to the posterior distribution [16, 24]. This approximate distribution β_ϕ over the model parameter space Θ is in turn paramterized by ϕ.

Variational learning minimizes the Kullback-Leibler (KL) divergence between β_ϕ and the Bayesian posterior $P(\theta|\mathcal{D})$ over the parameters ϕ. The variational cost function $\text{KL}[\beta_\phi \parallel P(\theta|\mathcal{D})]$ can be reframed as $\text{KL}[\beta_\phi \parallel \alpha] - \mathbb{E}_{\theta\sim\beta_\phi}\left[\sum_{i=1}^{N} \log_2 p_\theta(y_i|x_i)\right]$.

Using *bits-back* coding, $\text{KL}[\beta_\phi \parallel \alpha]$ can be interpreted as the cost of describing β_ϕ to $\mathfrak{R}$, given that $\mathfrak{R}$ already knows the prior distribution α [16, 24]. The second term in the cost function is simply the expected cost of the data. Therefore, the entire cost function is actually the length of an MDL coding scheme. Surprisingly, Blier and Ollivier also discovered that such *variational codes* are inefficient for deep neural networks in comparison to the online approach [17].

2.3.3 Online (Prequential) Codes

So far we have discussed coding schemes that transmit label data and model parameters after training has finished. However, we will now discuss a highly effective approach based on prequential statistics [25] that transmits while the training process is ongoing (in the sense that not all of the available data has been used). As before, assume that $\mathfrak{S}$ and $\mathfrak{R}$ have previously settled on a particular model class $\mathcal{M}$. Additionally, assume that they have also agreed to specific values r_ω of a training regimen class $\mathcal{R}$ (e.g. mini-batch size, learning rate, etc.).

In an *online (prequential) coding* scheme, $\mathfrak{S}$ first transmits a small batch of data labels to $\mathfrak{R}$; using a simple uniform encoding this transmission costs $t_1 \log_2 K$ bits, where t_1 is the size of the batch. The transmitted labels are then used by $\mathfrak{R}$ to train a model from the agreed-upon model class. $\mathfrak{S}$ trains an identical model by using the same training regimen and restricting the training data to the batch sent to $\mathfrak{R}$.

$\mathfrak{S}$ uses the freshly trained model to encode and transmit a new, unseen batch of data labels which are then decoded by $\mathfrak{R}$ using its identical model. Now that $\mathfrak{R}$ has access to even more labels, it can train a better-informed model (with $\mathfrak{S}$ again training an identical model using exactly the same training data and regimen as $\mathfrak{R}$). This process of improving the identical models and sending over new data using those updated models is repeated until all of the labels have been transmitted. Note that the length of the transmitted code drops as the trained models improve their accuracy on unseen data. Thus, online coding is an effective choice when using models that generalize well.

More formally, consider the following set of M transmission indices $1 \leq t_1 < t_2 < \cdots < t_M = N$. Let $p_{\theta_{t_i}}$ represent a model trained to completion, according to a given regimen, on the first t_i data pairs $(x, y)_{1:t_i}$ in $\mathcal{D}$. Given a set of transmission indices, the online code length $L^{\text{preq}}(\mathcal{D})$ is defined as

$$t_1 \log_2 K - \sum_{i=1}^{M-1} \log_2 p_{\theta_{t_i}}\left(y_{t_i+1:t_{i+1}} | x_{t_i+1:t_{i+1}}\right)$$

where a uniform encoding cost is used for the first transmission batch.

2.3.4 Online Efficiency in Neural Networks

As mentioned, it has been shown for deep neural networks that online codes are shorter than both variational codes and a variety of approaches based on two-part codes and network compression [17]. Additionally, these experiments showed that the difference in compression was quite significant. Why is it possible that online coding, a rather simple scheme, can beat such purpose built methods? We offer a brief explanation: it is largely due to the generalization capabilities of neural networks. By merging the training and coding processes, the online approach allows compression performance to be dictated by how well a model can infer unseen data - something neural networks are highly adept at. However, we note that due to this merger, the total training time for $\mathfrak{R}$ is essentially the decoding time and should be taken into careful consideration.

3 Online Code Length as the Objective

In light of the compelling results about online efficiency, a few instinctive questions arise: How might one compute and optimize the online code length? What would be the structure and variables of such a problem? How can we build practical considerations such as resource limitations into this analysis? In this section we begin to answer these questions by presenting the framework for the underlying optimization problem and demonstrating that the unconstrained, idealized case has a simple solution. Then in Section 4 we will motivate, discuss, and present solution strategies for the constrained case.

3.1 Problem Structure and Scope

Before we discuss the computation of minimal online code lengths, we must first clarify the variables and assumptions of the problem. In order for our analysis to extend to as general a setting as possible, we will assume that the choice of model class and training regimen design and parameters are fixed and therefore cannot be changed in order to improve online code lengths. This includes decisions such as neural network architecture, optimization method, learning rate, loss function, etc. Such choices clearly have an effect on online code lengths as they have a direct impact on the accuracy of unseen label predictions. However, the question of how to choose and train a model that generalizes well is quite possibly *the* fundamental problem of machine learning and beyond the scope of this paper. Fortunately, by assuming that these choices have been made prior to the decision to optimize for online code length, we can focus on solution methods that work regardless of model and training regimen.

Due to data access limitations for the receiver, special consideration must be given to the stopping criteria component of the training regimen. We will operate under the principle that model parameters are determined either at the end of a maximal number of epochs or after an early stopping condition has been triggered. If an early stopping condition is based on validation criteria from unseen data not available to $\mathfrak{R}$ then there are repercussions when optimizing for online code length (see Section 3.1.2).

3.1.1 Dataset Considerations

The selection of label transmission batches is the core decision sequence in the online code length minimization problem. However, before we discuss the elements of this selection process, we first note that we assume our data is randomly ordered. This assumption, which is generally implicit, again allows us to target as general a setting as possible as non-randomized datasets can easily be randomized.

A corollary to this assumption is that transmission batches must be filled sequentially. We do not allow $\mathfrak{S}$ to pick and choose labels from across the dataset to send in batches to $\mathfrak{R}$. If $\mathfrak{S}$ were allowed to fill transmission batches non-sequentially then, in general, the indices of the labels would also have to be sent. As the data is randomly ordered, the cost of coding these indices would be prohibitively expensive.

Under these assumptions, the label transmission component of the online code length problem is equivalent to finding the sequential indices $1 \leq t_1^* < t_2^* < \cdots < t_M^* = N$ that minimize $L^{\text{preq}}(\mathcal{D})$. We will discuss how to optimally select these indices in Sections 3.2 and 4. Note the number of batches M is not fixed and is an implicit variable.

3.1.2 Resource and Batch-Count Constraints

The problem scenario detailed in Section 2 engenders a number of optimization considerations that are not explicitly described in the definition of online code length. Chief among these is the fact that the computing resources available to $\mathfrak{S}$ and $\mathfrak{R}$ are in practice limited by both memory size and processing time. Memory size constrains the choice of model class and, often to a lesser extent, training regimen. However, because we have fixed the choice of model class and training regimen, we will also assume that memory limits are not violated. On the other hand, time constraints are highly relevant to practical online code minimization. While large transmission batch counts can lead to a short theoretical online code length, they can potentially require very long training times (see Section 3.2). In Sections 3.2 and 4 we will consider and expand on the minimization problem for the unconstrained and time constrained settings, respectively.

Although not the focus of this paper, it is important to note that each batch transmitted can come with a overhead penalty cost depending on a variety of meta considerations. For example, consider a training regimen that uses an early-stopping condition based on the performance of the trained model on a validation set after each epoch. If the receiver $\mathfrak{R}$ does not have access to this validation set the sender $\mathfrak{S}$ will have to transmit the stopping epoch for each batch, thus creating a batch count penalty.

3.2 Unconstrained Ideal Online Coding

We begin our analysis by discussing the minimization of the online code length $L^{\text{preq}}(\mathcal{D})$ without constraints. Disregarding time constraints does not mean that we allow infinite time training schemes. Rather, given a finite length training regimen for each transmission batch, the sender $\mathfrak{S}$ can sequentially partition the training set into transmission batches in any way and amount they want without regard for the resulting training time. Additionally, we are excluding all batch count related penalties from the total code length.

Note that a greater number of partitions (equivalent to a smaller average transmission batch size) can lead to longer online code development times. For example, consider two extreme partition schemes: T_{small} where $t_i = i$ for all $i \in \mathbb{N}_{\leq N} = \{1, 2, \dots M = N\}$, and T_{large} where $M = 2$ and $t_1 = \frac{N}{2}$. In a training regimen where the network is trained on the entire dataset received by $\mathfrak{R}$ for a fixed amount of epochs, T_{small} will have to process $\mathcal{O}(N)$ times more data than T_{large} - a significant increase given how large N can be for real world datasets. As previously stated, for now we disregard any processing time variation between partitions of all sizes (as well as batch count penalties) and instead focus on the optimal partition to minimize $L^{\text{preq}}(\mathcal{D})$.

Consider a fixed model class $\mathcal{M}$, training regimen instance r_ω, and training dataset $\mathcal{D}$. We define the generalization function $\phi_{\mathcal{M},r_\omega,\mathcal{D}}(d) := \mathbb{E}[p_{\theta_d}(\tilde{y}|\tilde{x})]$ as the expected accuracy of any model $p_{\theta_d} \in \mathcal{M}$ on an unseen data-label pair $(\tilde{x}, \tilde{y})$ from the same generating source as $\mathcal{D}$, given that the model was trained according to r_ω on d pieces of data. Note this expectation is over both random selections of $(\tilde{x}, \tilde{y})$ as well as random orderings of $\mathcal{D}$. Fix $\phi_{\mathcal{M},r_\omega,\mathcal{D}}(0) := \frac{1}{K}$ so that untrained models are no better than uniform random guessing. For the remainder of the paper we will drop the subscript and instead write $\phi(d)$. We proceed under the mild assumption that the generalization function strictly increases over its domain as more training data will likely only help the expected model accuracy (see Figure 1). This assumption allows us to see that the partition which minimizes the expected online code length is simply T_{small}.

Theorem 1. *If $\phi(d)$ is a strictly increasing function, the optimal partition to minimize the expected online code length $-\sum_{i=0}^{M-1}(t_{i+1} - t_i)\log_2 \phi(t_i)$, where $t_0 := 0$, is $t_i = i$ for all $i \in \mathbb{N}_{\leq N}$.*

The proof of this theorem can be found in the supplementary material.

4 Constrained Online Coding

We now turn our attention to the far more interesting problem of computing the optimal online code length given practical constraints. As motivation, consider the disadvantages of employing the 'optimal' partition T_{small} in a real-world online code. Not only would training time be greatly increased under certain training regimens, but transmitting each batch individually could incur an overhead coding cost per transmission that would lead to large code lengths. In this section, we will assume a fixed batch count and consider the online code length minimization problem solely under time constraints. We develop this topic in detail in Section 4.2; however, we must first discuss optimizing the choice of partition for a given fixed batch count M without time constraints.

4.1 Fixed Batch Count Optimization

Let us begin by relaxing the domain of the generalization function $\phi(d)$ from $\{0, 1, 2, \dots, N\}$ to the closed interval $[0, N]$. We will also assume that $\phi(d)$, in addition to being strictly increasing, is also concave on this interval. We can justify this assumption by noting that it is reasonable to expect, on average, roughly diminishing returns in accuracy from more and more training data (see Figure 1).

Let $G_N^{M-1} := \{\mathbf{z} \in \mathbb{R}_{(0,N)}^{M-1} \mid (\forall i \in \mathbb{N}_{\leq M})\ z_i > z_{i-1}\}$ where $\mathbb{R}_{(0,N)}^{M}$ is the M-1 dimensional space over the real open interval $(0, N)$. Note that neither z_0 nor z_M are components of $\mathbf{z}$; rather, they are constants where $z_0 := 0$ and $z_M := N$. Additionally, we require that the constant M is at least two and no more than N. Then the continuous version of the expected online code length optimization

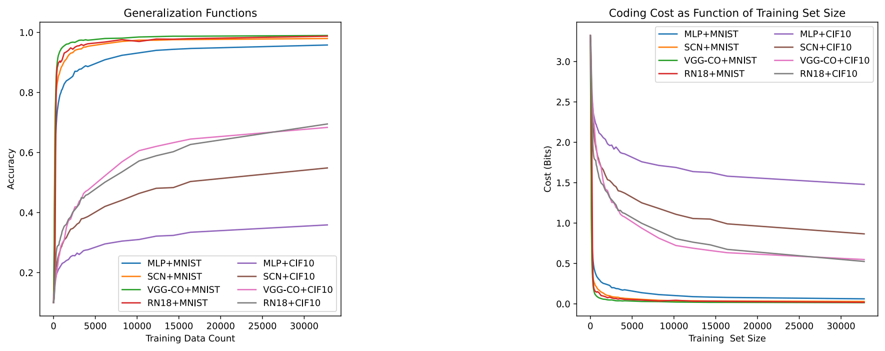

Figure 1: Monte Carlo estimates of coding-cost functions $-\log_2 \phi(d)$ and corresponding generalization functions $\phi(d)$ for various networks on both MNIST and CIFAR-10 [21, 26]. Networks used are a multilayer perceptron (MLP), a small convolutional/feed-forward network (SCN), a compact version of VGG (VGG-CO) [27], and ResNet18 (RN18) [28]. For further details please refer to the supplementary material.

problem can be stated as

$$\min_{\mathbf{z}\in G_N^{M-1}} \quad -\sum_{i=0}^{M-1} (z_{i+1} - z_i) \log_2 \phi(z_i) \tag{1}$$

Clearly, despite the convexity of $-\log_2 \phi(d)$, this objective function is not necessarily convex. Due to this difficulty, we shall instead consider an alternative formulation of the continuous problem. Let $b_i \in [0, N]$ represent the size of the i^{th} transmission batch. Note that minimizing $-\sum_{i=0}^{M-1} b_{i+1} \log_2 \phi\left(\sum_{j=0}^{i} b_j\right)$ such that $\sum_{i=1}^{M} b_i = N$ and $b_0 := 0$ is an equivalent optimization problem to the original given in Equation 1.

For each batch b_i we will now denote the cost rate per data point of that batch as c_i. The continuous objective function can now be written as $\sum_{i=1}^{M} b_i c_i$. Now consider the M ordered pairs $\left(\sum_{j=0}^{i} b_i,\ c_{i+1}\right)$ where $i = 0, 1, \ldots M-1$. Set $c_1 := \log_2 K$ and note that the first ordered pair is fixed at $(0,\ \log_2 K)$. We will regard these ordered pairs as points on the two-dimensional Cartesian plane that, with the exception of the first point, are free to move around in a partially bounded region above the function $-\log_2 \phi(d)$ (see Figure 2).

In practice we demarcate this area using the supporting hyper-planes of the decreasing convex hull of a Monte Carlo estimate of $-\log_2 \phi(d)$. We also further bound all c_i to the interval defined by the minimum and maximum values of the range of this convex hull (denoted by h_{min} and h_{max}). We now employ a well-known trick from geometric programming by substituting $b_i := e^{u_i}$ and $c_i := e^{v_i}$ for all $i = 1, 2, \ldots M$. Our new optimization problem is:

$$\begin{aligned} \min_{\mathbf{u},\mathbf{v}} \quad & \sum_{i=1}^{M} e^{u_i + v_i} \\ \text{s.t.} \quad & A \begin{bmatrix} \sum_{j=1}^{i} e^{u_j} \\ e^{v_{i+1}} \end{bmatrix} \geq \mathbf{w} \quad \forall i \in \mathbb{N}_{\leq M-1} \\ & \sum_{i=1}^{M} e^{u_i} \geq N, \quad \ln h_{\text{min}} \leq v_i \leq \ln h_{\text{max}} \end{aligned} \tag{2}$$

where $A \in \mathbb{R}_{\geq 0}^{s\times 2}$ and $\mathbf{w} \in \mathbb{R}_{\geq 0}^{s}$ given that there are s supporting hyper-planes. Note that A and $\mathbf{w}$ are non-negative by convention (the elements in each row must have the same sign as they parameterize non-increasing lines in the non-negative quadrant).

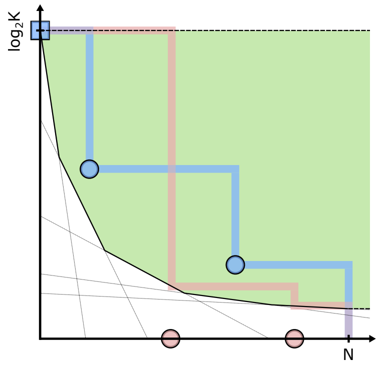

Figure 2: The initial continuous problem searched for the optimal partition sequence (represented by the red circles fixed to the bottom axis). The updated formulation allows points (the blue circles) to move around within established bounds (the green area) formed by the coding cost $-\log_2 \phi(d)$ and the acceptable range of cost values. In practice Monte Carlo estimates of $-\log_2 \phi(d)$ might not be perfectly convex, therefore we use the decreasing convex hull and represent the constraints using the supporting hyper-planes (dotted lines). Note the online code lengths of the blue and red schemes are given by the area under their lines.

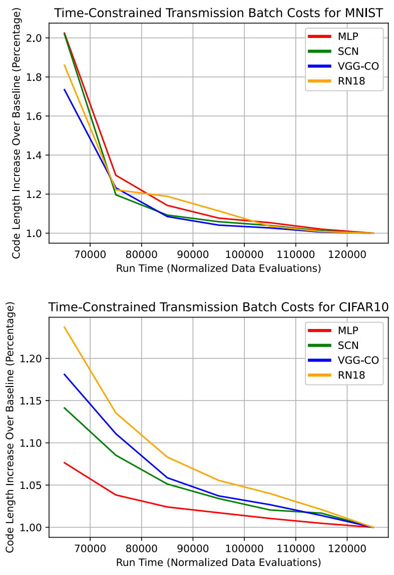

Figure 3: Relative optimized online code lengths for a variety of datasets and run times compared to a baseline of 125000. There is a clear variation in code length depending on how much time is given to the decoding process. For full experiment details please refer to the appendix.

Theorem 2. *Equations 1 and 2 have the same optimal objective function value in the case where $-\log_2 \phi(d)$ is equal to the decreasing convex hull of the Monte Carlo estimate demarcated by the supporting hyperplanes represented by the linear equations formed by A and $\mathbf{w}$.*

The proof of this theorem can be found in the supplementary material.

Note that although this is still not a convex optimization problem, it now belongs to a class of optimization problems known as 'difference of convex' (DC) programming problems [29]. DC programming problems have objective functions of the form $f_0(\mathbf{x}) - g_0(\mathbf{x})$ and constraints of the form $f_i(\mathbf{x}) - g_i(\mathbf{x}) \leq 0$ where all f and g are scalar-valued, convex functions. There is a well-established literature on DC programming [30, 31] which can be solved globally by branch and bound methods [32].

These methods can be slow in practice, however there are highly effective, non-global algorithms such as the convex-concave procedure (CCP) [33, 34] which is a type of majorization-minimization algorithm that solves simpler convex sub-problems. In this paper we use a CCP variant that was introduced by Shen et al. [32] as an extension to the penalty CCP method of Lipp and Boyd [29]. We also employ an implementation Shen et al. have made available as a plug-in to the popular CVXPY optimization software [35, 36]. In our experiments we also added the common sense constraint that u_i be positive, thus preventing batch sizes from being inconsequently small. Although Theorem 2 (and its corresponding proof) refer to a formulation without this constraint, we found it to improve optimizer results in practice.

4.2 Time-Constrained Optimization

It is relatively straightforward to add time constraints on the online code computation using the DC problem formed in Equation 2. Let us first consider a training scheme where the model is trained from scratch on all of the available training data for a fixed number of epochs τ. If time is measured by the total amount of data evaluations throughout the online coding process, then the time cost is simply $\sum_{i=1}^{M}\sum_{j=1}^{i}\tau \cdot b_j$ and we can add the convex constraint $\sum_{i=1}^{M}\sum_{j=1}^{i}\tau \cdot e^{u_j} \leq \lambda$ for some time budget λ.

Other forms of time constraints can be added with similar ease. For example, consider a training regime where early stopping is employed. Although τ is no longer fixed, one might instead consider the time cost as a weighted sum of sums $\sum_{i=1}^{M}\tau_i \sum_{j=1}^{i} b_j$ where each τ_i is a differently paramterized random variable based on some prediction scheme for early stopping. The time constraints could then be formulated in regards to the expectations of this cost.

4.3 Experimental Effect of Time Constraints on Code Lengths

It is obvious from Equation 2 that the introduction of time constraints will limit the computable minimal online code length. However, we cannot determine, in general, the non-trivial extents to which these constraints will matter. In order to investigate this impact, we solve Equation 2 (with the added constraint that u_i be positive) using the Shen CCP variant for a variety of datasets, networks, and time constraints.

In particular we use the decreasing convex hull of the coding-cost functions $-\log_2 \phi(d)$ seen in Figure 1. Networks and datasets were largely chosen for their familiarity with a general audience along with a demonstration of a variety of generalization function response rates – rapid (e.g. VGG-CO + MNIST), middling (e.g. VGG-CO + CIF10), and slow (e.g. MLP + CIF10). Because response rates are determined by the combination of model power and dataset simplicity/difficulty, these choices of networks and models cover a reasonable range of response rates that a practitioner might expect to see on their own model and dataset.

The results, summarized in Figure 3, clearly show that under time constraints (measured in total data evaluations if training was run on one epoch per transmission on the entire received dataset) there is significant variance. The range of time was chosen to range from just above the size of the actual datasets to about twice this size. The experiments support the thesis that realizable online codes are greatly limited by a factor, resource constraints, completely independent to the objects described. Note that our analysis has presumed access to $\phi(d)$ or a corresponding Monte Carlo estimate. However, we do not include any related computation cost as a time constraint, but rather categorize it as preprocessing step just as a hyperparameter search would be. When discussing time constraints we are purely interested in the burden on the receiver $\mathfrak{R}$ as this is tantamount to a decoding cost. This is also why we are interested in the practical, realizable online MDL code lengths. If given the generalization function it is, of course, easy to determine the ideal online code length (roughly equal to the area under the generalization curve).

Although as Blier and Ollivier demonstrated, online code lengths are extremely efficient for deep neural networks, there clearly exists a trade-off between code length and practical decoding time. This fact becomes all the more striking when one considers decoding times for coding schemes that do not involve receiver training. Such a trade-off suggests that perhaps, in practical terms, it might be worthwhile to introduce time penalties (somewhat akin to the extension to Kolmogorov complexity known as Levin complexity [7]).

5 Conclusion

Online coding is an essential MDL technique that has proven to give some of the most compact codes for neural networks - an important topic with many practical and theoretical implications. In this paper we have shown that this powerful MDL coding scheme is highly sensitive to the amount of computing time available. Specifically, we have introduced and developed the realizable online coding minimization problem, shown how to overcome the inherent non-convexity, and then used this approach to show experimental evidence of this sensitivity on a selection of commonly used datasets and architectures.

Time dependence is an external factor to the actual choice of neural network architecture, training regimen and dataset - the core features that are meant to determine description lengths. However, the fact that realizable online codes are highly subject to resource constraints highlight a serious drawback to the usage of this approach. These results suggest that perhaps an approach that properly weighs computation times might be more appropriate.

References

[1] Andrei Nikolaevich Kolmogorov. Three Approaches to the Quantitative Definition of Information. *International journal of computer mathematics*, 2(1-4):157–168, 1968.

[2] Ray J Solomonoff. A Formal Theory of Inductive Inference. Part I. *Information and control*, 7 (1):1–22, 1964.

[3] Ray J Solomonoff. A Formal Theory of Inductive Inference. Part II. *Information and control*, 7 (2):224–254, 1964.

[4] Gregory J Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

[5] 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.

[6] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006. ISBN 0471241954.

[7] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 4th edition, 2019.

[8] Jürgen Schmidhuber. Discovering Neural Nets with Low Kolmogorov Complexity and High Generalization Capability. *Neural Networks*, 10(5):857 – 873, 1997.

[9] Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer Science & Business Media, 2004.

[10] Gregory J Chaitin. *Thinking About Godel And Turing: Essays on Complexity, 1970–2007*, chapter On the Intelligibility of the Universe and the Notions of Simplicity, Complexity and Irreducibility. World scientific, 2007.

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model Compression and Acceleration for Deep Neural Networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

[12] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[13] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger Generalization Bounds for Deep Nets via a Compression Approach. In *International Conference on Machine Learning*, pages 254–263. PMLR, 2018.

[14] Jorma Rissanen. Modeling By Shortest Data Description. *Automatica*, 14:465–471, 1978.

[15] Avrim Blum and John Langford. PAC-MDL Bounds. In *Learning Theory and Kernel Machines*, pages 344–357. Springer, 2003.

[16] Geoffrey E Hinton and Drew Van Camp. Keeping Neural Networks Simple by Minimizing the Description Length of the Weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 5–13, 1993.

[17] Léonard Blier and Yann Ollivier. The Description Length of Deep Learning Models. In *Advances in Neural Information Processing Systems*, pages 2216–2226, 2018.

[18] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. Rissanen Data Analysis: Examining Dataset Characteristics via Description Length. *Neural Compression Workshop at ICLR 2021*, 2021.

[19] David JC MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge university press, 2003.

[20] Peter D Grünwald. *The Minimum Description Length Principle*. MIT press, 2007.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural networks with Pruning, Trained Quantization and Huffman Coding. In *International Conference on Learning Representations*, 2016.

[23] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the Intrinsic Dimension of Objective Landscapes. In *International Conference on Learning Representations*, 2018.

[24] Alex Graves. Practical Variational Inference for Neural Networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[25] A Philip 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.

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009.

[27] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

[29] Thomas Lipp and Stephen Boyd. Variations and Extension of the Convex–Concave Procedure. *Optimization and Engineering*, 17(2):263–287, 2016.

[30] Reiner Horst and Nguyen V Thoai. Dc Programming: Overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, 1999.

[31] Reiner Horst, Panos M Pardalos, and Nguyen Van Thoai. *Introduction to Global Optimization*. Springer Science & Business Media, 2000.

[32] Xinyue Shen, Steven Diamond, Yuantao Gu, and Stephen Boyd. Disciplined Convex-Concave Programming. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 1009–1014. IEEE, 2016.

[33] Alan L Yuille and Anand Rangarajan. The Concave-Convex Procedure. *Neural computation*, 15(4):915–936, 2003.

[34] Gert Lanckriet and Bharath K Sriperumbudur. On the Convergence of the Concave-Convex Procedure. In *Advances in Neural Information Processing Systems*, volume 22, pages 1759–1767, 2009.

[35] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[36] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Yes in the sense that we must use an experimental approach instead of a theoretical.

(c) Did you discuss any potential negative societal impacts of your work? [No] The paper is too far removed from this form of impact.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [Yes]

   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix items.

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplementary material.

   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In supplementary material.

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In supplementary material.

   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In supplementary material.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes] In supplementary material.

   (b) Did you mention the license of the assets? [Yes] In supplementary material.

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] New models are in supplementary code.

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We used well-known, publicly available datasets.

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]