Scaling Laws with Hidden Structure

Charles Arnal, Clement Berenfeld, Simon Rosenberg, Vivien Cabannes

Scaling Laws with Hidden Structure

Charles Arnal
Inria

Clement Berenfeld
Postdam University

Simon Rosenberg
C3AI

Vivien Cabannes
Meta AI, FAIR

Abstract

Statistical learning in high-dimensional spaces is challenging without a strong underlying data structure. Recent advances with foundational models suggest that text and image data contain such hidden structures, which help mitigate the curse of dimensionality. Inspired by results from nonparametric statistics, we hypothesize that this phenomenon can be partially explained in terms of decomposition of complex tasks into simpler subtasks. In this paper, we present a controlled experimental framework to test whether neural networks can indeed exploit such "hidden factorial structures." We find that they do leverage these latent patterns to learn discrete distributions more efficiently, and derive scaling laws linking model sizes, hidden factorizations, and accuracy. We also study the interplay between our structural assumptions and the models' capacity for generalization.

1 Introduction

Context and motivations The ability of artificial intelligence systems to solve highly complex tasks by extracting information from a vast amount of data remains poorly understood. On the one hand, the *curse of dimensionality* states that learning becomes virtually impossible as the data dimension grows large. On the other hand, it has been now long observed that these systems can develop a fine understanding of even high-dimensional data. This is for instance the case for Large Languege Models (LLMs), which process lists of tokens whose different combinations can easily exceed 10^{80} in number (Brown et al., 2020).

It is well-known that the curse of dimensionality can be overcome when the data exhibits a lower-dimensional underlying structure, and if the learning procedure can adapt to this intrinsic dimension. Many structural assumptions have been proposed by statisticians, such as the existence of a low-dimensional manifold on which the data lies (Fefferman et al., 2016), smoothness of the functional to learn (Caponnetto and De Vito, 2006) or sparsity of the later (Hristache et al., 2001). While these assumptions allow for the derivation of theorems, they are arguably too simplistic to explain the current successes of machine learning (Cabannes and Vigogna, 2023). A recent line of research has argued that the factorization of a complex task into simpler sub-tasks could explain the ability to learn such a complex task efficiently (Parascandolo et al., 2018; Arora and Goyal, 2023; Ahuja and Mansouri, 2024; Cagnetta et al., 2024). We approach this work from this perspective.

Contributions We adopt a discrete data framework, which is a reasonable choice given the two following considerations: first, the frequently discrete nature of data (in particular text data); and second, the operational basis of LLMs, which rely on tokenized inputs and are of particular interest in contemporary research, and one of the main motivations behind this work.

Within this framework, we propose a factorization-based model where both the input and output spaces are decomposed into a products of small unknown factors, and where the tasks happen factor-wise. This data structure is motivated by examples detailed further below. Our goal is to *test whether neural networks can leverage hidden factorial structure to learn more efficiently*, both in terms of computational and statistical complexity. This approach departs from continuous models and their usual structural assumptions and gives insights into the impressive performances of current state-of-the-art models on complex discrete data.

Due to the lack of theoretical tools at hand (see, e.g., Allen-Zhu and Li, 2024, for discussion on the matter), we focus on an empirical approach through a controlled exploratory study that connects the data structure and the performance a Multilayer Perceptron (MLP). The reason behind the choice of MLPs is that they serve as the essential building blocks of many modern machine learning techniques, and in particular of LLMs, where they represent the majority of the model's parameters. This way, we hope to shed new light on some of the core principles behind LLMs learning. We formulate our findings in term of scaling laws (Kaplan et al., 2020), by extensively studying how the performances of our model evolve with changes in train data size, number of parameters, compute resources, and complexity of the hidden data structure.

Related works In their seminal work on scaling laws, Kaplan et al. (2020) demonstrated that for large-scale neural networks, performance improves predictably as a power-law function of parameters, data, and compute. Henighan et al. (2020) extended these findings to more specific domains, revealing consistent patterns in both vision and text models. Our work relates to a recent stream of research that has emerged exploring the theoretical underpinnings of scaling laws, proposing that scaling behaviors may be linked to the structural assumption on the data (see, e.g., Hutter, 2021; Maloney et al., 2022; Cabannes et al., 2024; Bahri et al., 2024; Bordelon et al., 2024). Our assumptions are inspired both by classical structural approaches from nonparametric statistics (Hristache et al., 2001; Fefferman et al., 2016) and by recent works in machine learning suggesting the importance of compositionality and sub-task decomposition to understand and improve learning (e.g., Dziri et al., 2023; Alabdulkareem et al., 2024; Valvoda et al., 2023; Wang et al., 2024; Wies et al., 2022; Zhang et al., 2024; Guo et al., 2024), as well as hidden sparsity (e.g., Barak et al., 2022; Marion et al., 2024).

Summary of contributions

1. We propose a new type of structural assumptions on discrete data distributions (Section 2);
2. We provide theoretical insights into the impact of these assumptions on the difficulty of learning the distributions (Section 3);
3. We provide experimental proofs that Neural Networks can leverage our hypotheses to learn more efficiently and we infer scaling laws (Section 4).

2 Setting

In this paper, we make Neural Networks (NNs) learn discrete conditional distributions $p(y|x)$ and we analyze both theoretically and experimentally how specific structural assumptions about these distributions can make the task easier.

More precisely, we consider a set of inputs $\mathcal{X}$ of cardinality $N \in \mathbb{N}$ and a set of outputs $\mathcal{Y}$ of cardinality $M \in \mathbb{N}$. We assume that the input/output pairs are generated from a joint distribution $(X, Y) \sim p$ on $\mathcal{X} \times \mathcal{Y}$, and we task a NN with learning the conditional distribution $p(y|x)$ for each input $x \in \mathcal{X}$. The quality of a learned estimator $\widehat{p}$ is measured through the Kullback-Leibler divergence, defined as

$$\mathcal{L}(\widehat{p}) = \mathbb{E}_{(X,Y)\sim p}\left[-\log \frac{\widehat{p}(Y|X)}{p(Y|X)}\right]. \tag{1}$$

From an optimization point of view, this loss is equivalent to the cross-entropy loss used by practitioners. Indeed, the loss $\mathcal{L}$ is the excess risk of the cross-entropy loss, i.e. the cross-entropy loss minus its minimizer.

In order to learn from discrete data, we embed our problem in a continuous space $\mathbb{R}^d$ with two (learnable) embeddings:

$$e : \mathcal{X} \to \mathbb{R}^d, \qquad u : \mathcal{Y} \to \mathbb{R}^d. \tag{2}$$

In addition to the embeddings, the NN learns a transformation $F : \mathbb{R}^d \to \mathbb{R}^d$ of the embedding space. The final estimator is parameterized through the softmax function

$$\widehat{p}(y|x) = \frac{\exp(u_y^\top F(e_x))}{\sum_{y' \in \mathcal{Y}} \exp(u_{y'}^\top F(e_x))}, \tag{3}$$

where we use the abbreviations $e_x := e(x)$ and $u_y := u(y)$.

This setting, and in particular the fact that e_x and u_y belong to the same embedding space, was designed to closely match current state-of-the-art architectures, where residual connections are commonly used (He et al., 2015).

Among all possible conditional density functions $p(y|x)$, we are particularly interested in those that satisfy certain structural conditions, which we call and which we define and motivate thereafter. Our goal is to establish whether NNs can efficiently leverage those assumptions to better learn and represent $p(y|x)$ in two slightly different settings: one in which the embeddings e_x are learned, and one in which fixed, factorization-compatible embeddings are used.

2.1 The Factorization Hypothesis

Our factorization hypothesis assumes that both the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ can be decomposed into $k \in \mathbb{N}$ and $\ell \in \mathbb{N}$ *unknown factors* respectively. In other terms,

$$\mathcal{X} \cong \prod_{i\in[k]} \mathcal{X}_i \quad \text{and} \quad \mathcal{Y} \cong \prod_{j\in[\ell]} \mathcal{Y}_j,$$

where each $\mathcal{X}_i$ and $\mathcal{Y}_i$ is a discrete space containing $p_i = |\mathcal{X}_i|$, respectively $q_j = |\mathcal{Y}_j|$, elements. By "unknown", we mean that the mappings $\mathcal{X} \cong \prod \mathcal{X}_i$ and $\mathcal{Y} \cong \prod \mathcal{Y}_j$ are not given to the agent (the NN) trying to learn the conditional distribution $p(y|x)$. We further assume that for all index $j \in [\ell]$, there exists a subset $I_j \subseteq [k]$ such that the conditional probability distribution $p(y|x)$ factors into

$$p(y|x) = \prod_{j\in[\ell]} p(y_j|\mathrm{pa}_j), \tag{FH}$$

where

$$\mathrm{pa}_j = (x_i)_{i\in I_j} \in \prod_{i\in I_j} \mathcal{X}_i.$$

Here, $x_i \in \mathcal{Y}_j$ denotes the i-th coordinate of x in its factor decomposition (respectively $y_j \in \mathcal{Y}_j$ is the j-th coordinate of y), hence pa_j is the set of coordinates that influence the factor y_j. In other words, the Factorization Hypothesis (FH) states that the coordinates y_j of y in the decomposition are independent given the input x, and that each y_j only depends on the subset pa_j of the input coordinates. This model allows for structures where only a (potentially small)

subset of hidden characteristics of the input influences each feature of the output. As a consequence, learning the task $p(y|x)$ can be done by learning the subtasks $p(y_j|\mathrm{pa}_j)$.

The notation pa_j stands for "parents" and is borrowed from the field of graphical probabilistic model (Jordan, 2004), as our problem can naturally be represented by a bipartite directed acyclic graph, where a first set of k vertices represents the factors of x and a second set of l vertices represents those of y and an edge between x_i and y_j indicates a non-trivial causal relationship between the two random variables, see Figure 1.

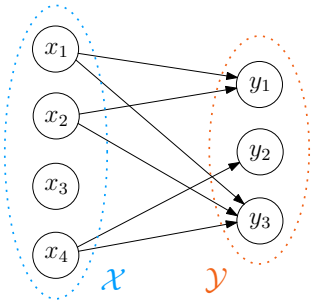

Figure 1: A graphical representation of a model for $k = 4$ and $\ell = 3$. In this example, $\mathrm{pa}_1 = (x_1, x_2)$, $\mathrm{pa}_2 = (x_4)$, $\mathrm{pa}_3 = (x_1, x_2, x_4)$, and the third hidden variable x_3 has no impact on the output y.

Remark 1 (Compression and representation)**.** *Each layer of an MLP tasked with approximating a map typically performs a mix of representation, i.e. applies some information-preserving transformation to its input to make it more suited to later computations, and compression, i.e. processes the information in a many-to-one fashion. In our setting, factorizing $\mathcal{X}$ corresponds to representation and mapping (probabilistically) the factors of $\mathcal{X}$ to those of $\mathcal{Y}$ to compression.*

The following examples provide motivations for the factorization hypothesis.

Example 1 (Text data)**.** *Text data implicitly factors into characteristics of small cardinality, such as e.g. language, grammatical function, length, sentiment, complexity, etc. With most tasks, each of the characteristics of the expected output only depends on a subset of those of the input: in a next-word prediction setting, for example, the language of the next word would be the same as that of the input, its grammatical function would depend on the grammatical functions and language of the previous words, etc.*

Example 2 (Recommender systems)**.** *Likewise, consider a recommender system tasked with matching users with targets (e.g. movies on a streaming platform, products on a marketplace, etc.). One expects that the relevant data for both users and targets factor in such a way that only a small number of (potentially unknown) characteristics of a given user (e.g. nationality, language, socioeconomic status, etc.) impacts each factor of the recommended target (e.g. language, movie genre, etc.). In this context, or setting is to be compared with low-rank matrix completion approaches (Candes and Plan, 2010) where users and target are implicitly assumed to be embedded in a low-dimensional space (each dimension corresponding to a factor) and that preferences are measured through inner products (i.e. each factor of the target is only affected by one of the user's factors).*

Remark 2 (Link to mixture estimation)**.** *The law of the variable y as defined above can be written as $p(y) = \sum \alpha_{\mathbf{z}} \prod p(y_j|\mathrm{pa}_j = \mathbf{z}_j)$ where $\alpha_{\mathbf{z}} = \mathbb{P}(\mathrm{pa}_1 = \mathbf{z}_1, \dots, \mathrm{pa}_\ell = \mathbf{z}_\ell)$. Thus learning $p(y|x)$ involves in particular learning the components of a mixture of products of distributions over a discrete space (Feldman et al., 2008), with the added difficulty that one does not know a priori the product decomposition on $\mathcal{Y}$.*

2.2 Input Embeddings

This subsection discusses two possible embeddings of our discrete problem in a continuous space, which are then used as inputs to the transform F: a learned encoding, and a fixed factorization-compatible embedding. Both are defined further below.

Learned embedding In this classic setting, used e.g. in most LLMs, (see Brown et al., 2020; Touvron et al., 2023), each discrete element $x \in \mathcal{X}$ is mapped to a learned vector $e_x \in \mathbb{R}^d$. The embeddings are typically initialized as random Gaussian vectors, and learned during training jointly with the embedding transform F and the output embedding u. Though very natural, this setting lacks a crucial property: as no structure is enforced on the embeddings, a trained NN has no hope of correctly generalizing to an input x that was not part of its training set, and whose embedding would simply be equal to its random initialization. This stands in contrast to the factorization-compatible embeddings described below.

Factorization-compatible embedding In this setting, we assume that an oracle or a previously trained NN provides us with an embedding $\tilde{e}_x$ that is adapted to the (unknown) factorization of $\mathcal{X}$ in the following sense:

$$\tilde{e}_x = [E_1 \dots E_k] \begin{bmatrix} \mathbf{1}_{x_1} \\ \vdots \\ \mathbf{1}_{x_k} \end{bmatrix} = \sum_{i \in [k]} e^i(x_i), \tag{4}$$

where $E_i \in \mathbb{R}^{d \times p_i}$ is some fixed matrix, $\mathbf{1}_{x_i} \in \mathbb{R}^{p_i}$ is the one-hot encoding of $x_i \in \mathcal{X}_i$ and $e^i(x_i) = E_i \mathbf{1}_{x_i} \in \mathbb{R}^d$. In other words, each discrete element $x_i \in \mathcal{X}_i$ is mapped to some fixed, arbitrary embedding $e^i(x_i)$. The total embedding $\tilde{e}_x$ of x is then the sum of the embeddings of its factors. See Figure 2 for an illustration.

Note that when $d \geq \sum_{i \in [k]} p_i$, any matrix $[E_1 \dots E_k]$ with coefficients sampled from continuous distributions will almost surely be of rank $\sum_{i \in [k]} p_i$. As a consequence, the

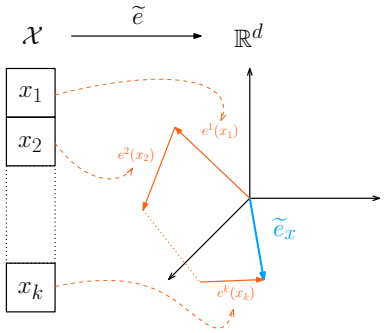

Figure 2: A diagram of a factorization-compatible embedding $x \mapsto \tilde{e}_x$.

embedding $\tilde{e}$ will be linearly invertible, in the sense that learning a simple linear transformation would enable the NN to recover the factors $(x_1, \dots, x_k)$ from its input e_x. This transformation could be learned from $\sum p_i$ examples, which can be much smaller than the cardinality of the full set $\mathcal{X}$ of $\prod p_i$ elements (as small as $\log N$). Note that, thanks to its nonlinearities, a NN may retrieve these factors even when $d < \sum p_i$.

In contrast to learned embeddings, factorization-compatible embeddings allow for generalization across inputs. Indeed, consider the simple case where $\mathcal{X} \cong \mathcal{X}_1 \times \mathcal{X}_2$, $\mathcal{Y} \cong \mathcal{Y}_1 \times \mathcal{Y}_2$ and $p(y|x) = p(y_1|x_1)p(y_2|x_2)$. Assume that the NN has learned to invert the embedding $\tilde{e}$, and can recover the factors (x_1, x_2) of x given an input $\tilde{e}_x$. Then the conditional distribution $p(y|x)$ for some yet unseen input $x = (x_1, x_2)$ can be estimated from past observations as long as points of the shape (x_1, x_2') and (x_1', x_2), for some $x_1' \in \mathcal{X}_1, x_2' \in \mathcal{X}_2$, were seen before –since $p(y_1|x_1)$ and $p(y_2|x_2)$ can be computed from (x_1, x_2') and (x_1', x_2) respectively. Likewise, given a more complicated factorization, the conditional distribution $p(y|x)$ can be estimated for a yet unseen x as long as each of the parent coordinates $\mathrm{pa}_j = (x_i)_{i \in I_j}$ have already appeared in previously observed data points.

The examples below provide motivations for this setting.

Remark 3 (Link to transformers)**.** *The functional form of the factorization-compatible embeddings closely matches that of the outputs of the attention layers in transformers, which are produced as a weighted sum of value vectors. Those outputs are then used as inputs to MLPs.*

Remark 4 (Link to other statistical problems)**.** *When the images of the maps $(e^1, \dots, e^k)$ are linearly free, there exists some (unknown) linear mapping $(T_1, \dots, T_k)$ such that $T_i(e_x) = e^i(x_i)$ for all $i \in [k]$. When furthermore e^i is injective over $\mathcal{X}_i$, the target distribution can be expressed as*

$$p(y|x) = \prod_{j \in [\ell]} p(y_j|\mathrm{pa}_j) = \prod_{j \in [\ell]} p(y_j|\mathrm{Pa}_j(e_x)),$$

where

$$\mathrm{Pa}_j : z \in \mathbb{R}^d \mapsto (T_i(z))_{i \in I_j}$$

is a linear map (the Parent map). If we now assume that $\ell = 1$ and that we are given an embedding $u : \mathcal{Y} \to \mathbb{R}^d$, the model takes the simpler form $p(u|e) = p(u|\mathrm{Pa}(e))$. When the distribution $p(y|x)$ is close to a Dirac distribution, i.e. when the assignment $x \mapsto y$ can be seen as a deterministic function with some added noise, the situation can be reframed as a regression problem $u = f(\mathrm{Pa}(e)) + \text{noise}$, which coincides with the well-studied multi-index problem (Hristache et al., 2001). In particular, it has been shown that even with Pa unknown, the rate of estimation of the regression function depends on the rank of Pa and not on the embedding dimension of e. Closely related settings involving hidden sparsity include (Marion et al., 2024) and (Barak et al., 2022).

3 Theoretical Analysis

We consider both the computational and statistical difficulty of learning a factorizable data distribution –that is to say both the minimal theoretical number of parameters needed in an MLP and the minimal theoretical number of training points needed for this task.

3.1 Approximation Complexity

We aim to assess the impact of our factorization hypothesis on the computational complexity involved in approximating the conditional distributions $p(y|x)$ for our models.

Consider first the learned embeddings defined in the previous subsection. Let $E \in \mathbb{R}^{d \times N}$ and $U \in \mathbb{R}^{d \times M}$ be the learned embedding matrices that map the one-hot encoding of x to e_x, respectively the one-hot encoding of y to u_y, and define the matrix $G := F(E) \in \mathbb{R}^{d \times N}$, where the transform F is applied column-wise (i.e. to each e_x). Note that as each token x can be mapped to any vector $e_x \in \mathbb{R}^d$ by the embedding process, the transform F adds nothing to the expressivity of the overall model (though it might impact the training dynamics). Constrained by the functional form of our estimator stated in (3), we want to represent $p(y|x)$ as proportional to $\exp(u_y^\top F(e_x))$. This is equivalent to representing the matrix $\Lambda := (\log p(y|x))_{y,x} \in \mathbb{R}^{M \times N}$ as a matrix product

$$\Lambda = U^\top G + \text{column-wise constant}, \tag{5}$$

where the column-wise constant accounts for the renormalizing factor from (3). Under (FH), an exact solution is given by

$$G = \begin{bmatrix} (\log p(y_1|x))_{y_1 \in \mathcal{Y}_1, x \in \mathcal{X}} \\ \vdots \\ (\log p(y_\ell|x))_{y_\ell \in \mathcal{Y}_\ell, x \in \mathcal{X}} \end{bmatrix} \in \mathbb{R}^{Q \times N}$$

and

$$U = \begin{bmatrix} (\mathbf{1}_{y_1})_{y \in \mathcal{Y}} \\ \vdots \\ (\mathbf{1}_{y_\ell})_{y \in \mathcal{Y}} \end{bmatrix} \in \mathbb{R}^{Q \times M},$$

where

$$Q = \sum_{j \in [\ell]} q_j,$$

and $(\mathbf{1}_{y_j})_{y \in \mathcal{Y}} \in \mathbb{R}^{q_j \times M}$ is the matrix such that its column indexed by $y \in \mathcal{Y}$ is the one-hot encoding of the factor y_j of y in $\mathcal{Y}_j$. This shows that in the learned embedding setting, the distribution can be perfectly approximated using as embedding dimension $d = Q$ and $(M + N) \cdot d$ parameters. In fact, another exact solution to (5), which mixes the first one with a slightly altered version of it (see the Supplementary Materials for details), shows that the same property holds for $d = \bar{\chi}$, where

$$\bar{\chi} = \sum_{j \in [\ell]} \min\{|\mathrm{pa}_j|, q_j\}, \tag{AC}$$

and

$$|\mathrm{pa}_j| = \prod_{i \in I_j} |\mathcal{X}_i|.$$

This showcases how the factorization hypothesis may enable an *exponential gain* regarding the computational difficulty of the task at hand, going from $d = \min(M, N)$ when no structure is assumed on the data, to d potentially as small as $\min\{\log_2 M, \log_2 N\}$.

Note that while the quantity $\sum_{j \in [\ell]} \min\{|\mathrm{pa}_j|, q_j\}$ bounds the rank of Λ, it is easy to show that Λ having a small rank does not imply the existence of a factorization. Simple counter-examples are provided in the Supplementary Materials.

Let us now consider the case of factorization-compatible embeddings. As before, Λ must be expressed as a product $U^\top F(E)$ (up to some column-wise additive constant), but now we have the constraint that E is of the form (4). To ease notations, let us set

$$P = \sum_{i \in [k]} p_i, \qquad \bar{P} = \sum_{j \in [\ell]} \left|\mathrm{pa}_j\right|.$$

Assume that the matrix $[E_1 \dots E_k]$ is invertible, which is generically the case as soon as $d \geq P$ (as noted in Subsection 2.2), and let T be its inverse, which maps e_x to the product $(\mathbf{1}_{x_1}, \dots, \mathbf{1}_{x_k}) \in \mathbb{R}^P$ of the one-hot encodings of the factors x_i of x. Let also

$$\Xi : \mathbb{R}^P \to \mathbb{R}^{\bar{P}}$$

maps $(\mathbf{1}_{x_1}, \dots, \mathbf{1}_{x_\ell})$ to the corresponding product $(\mathbf{1}_{\mathrm{pa}_1}, \dots, \mathbf{1}_{\mathrm{pa}_\ell})$ of one-hot encoding of the parents. The map Ξ is non-linear, but it can be represented by a single feedforward layer (whose number of parameters is roughly $P \times \bar{P}$). Then an exact solution can be expressed as

$$F : e \mapsto \Xi \cdot T \cdot e$$

(where some padding can be added to ensure that the ambient dimension remains constant at each step) and

$$U = \begin{bmatrix} (\log p(y_1|z_1))_{z_1 \in \mathrm{pa}_1, y \in \mathcal{Y}} \\ \vdots \\ (\log p(y_\ell|z_\ell))_{z_\ell \in \mathrm{pa}_\ell, y \in \mathcal{Y}} \end{bmatrix} \in \mathbb{R}^{\bar{P} \times M},$$

where the rows of the submatrix $(\log p(y_j|z_j))_{z_j \in \mathrm{pa}_j, y \in \mathcal{Y}} \in \mathbb{R}^{|\mathrm{pa}_j| \times M}$ are indexed by the elements z_j of pa_j. This shows that while the situation is slightly more complex than in the case of learned embeddings, we can still bound the number of parameters and the embedding dimension needed to approximate the conditional distribution in term of factorization characteristics.

3.2 Statistical Complexity

The sample complexity of learning a given distribution $p(y|x)$ (for the KL / cross entropy loss) without any structural assumption at accuracy $\varepsilon > 0$ is given by (Canonne, 2020):

$$\frac{1}{\varepsilon} |\mathcal{X}| \times |\mathcal{Y}|. \tag{6}$$

Because the (hidden) factorization assumption transforms the learning task into ℓ independent (but unknown) learning tasks, we conjecture the optimistic bound χ/ε on the sample complexity, where

$$\chi = \sum_{j=1}^{\ell} q_j \times |\mathrm{pa}_j|, \tag{SC}$$

which is always smaller than (6) (and often *exponentially smaller*). The bound is optimistic because it is computed as if the factorization was known in advance; this can be heuristically justified by the fact that for most classical tasks in nonparametric statistics, the rate of estimation depends directly on structural assumptions (i.e. multi-index models, manifold hypothesis, etc) even when one does not know the precise instance of these assumptions (i.e. indices in the multi-index models, location of the manifold, etc).

4 Experiments

We explore the practical implications of our discrete factorized structure (FH) on the performance of MLPs. We link these performances to the parameters of the factorization and to the various hyperparameters of our models, highlighting that MLPs perform better under (FH) while requiring fewer parameters and at a lower computational cost. Our code is available at `https://github.com/facebookresearch/pal/`.

4.1 Experimental Design

We describe in this section the data generation process, namely our procedure for generating distributions $p(y, x)$

satisfying (FH), and then describe our chosen MLP architecture. The precise values of the parameters of our data generation process, of the model's hyperparameters and of their associated default values can be found in Table 1 in the Supplementary.

Data specification Unless otherwise specified, we let the token spaces be $\mathcal{X} = \mathcal{Y} \cong [4096]$; as N and M are equal to 2^{12}, this choice allows for multiple possible factorizations. Our data model depends on four parameters.

(P1) Input factors: $(p_i)_{i\in[k]} \in \mathbb{N}^k$.

By default, we let $k = 12$ and $(p_i)_i = (2)_{i\in[12]}$.

(P2) Output factors: $(q_i)_{i\in[\ell]} \in \mathbb{N}^k$

By default, we let $\ell = 4$ and $(q_j)_j = (8)_{j\in[4]}$.

(P3) Number of parents: $|I_j| \in \mathbb{N}$.

Although this results in slightly less general factorizations (as all the y-factors have the same number of parents), fixing $|I_j|$ reduces randomness in the graph generation process. Notably, it turns χ (SC) and $\bar{\chi}$ (AC) into deterministic quantities.

Alternatively, we may want to consider:

(P3') Connectivity parameter: $\beta \in [0, 1]$.

For each tuple of integers $(i, j) \in [k] \times [\ell]$, we draw a random variable Z_{ij} uniformly on $[0, 1]$. We let i belong to I_j, which is equivalent to the coordinates x_i appearing in pa_j, when $Z_{ij} \leq \beta$. In other terms, β controls the probability of activation of each edge in the bipartite graph linking the x-factors to the y-factors.

Unless otherwise specified, we set ourselves in (P3) (rather than (P3')) with $|I_j| = 2$.

(P4) Concentration parameter: $\alpha \in \mathbb{R}_+$.

We let the marginal distribution $p(x)$ be the uniform law on $\mathcal{X}$ for all experiments. We generate conditional distributions as follows: for each $j \in [k]$, and each point $\mathrm{pa}_j \in \prod_{i\in I_j} \mathcal{X}_i$, we sample the vector $(p(y_j|\mathrm{pa}_j))_{y_j\in\mathcal{Y}_j}$ according to a Dirichlet distribution of parameter $(\alpha, \dots, \alpha) \in \mathbb{R}^{q_j}$. When $\alpha = 10^{-3}$, this leads to $(p(y_j|\mathrm{pa}_j))$ being very close in distribution to a Dirac, while $\alpha = 1$ leads to $(p(y_j|\mathrm{pa}_j))$ being more uniformly sampled across the simplex. We let $\alpha = 10^{-1}$ be our default value to achieve a balance between the deterministic and uniform cases, to reflect the kind of probability distributions that we expect to encounter in real-world use cases such as next-word prediction.

Model specification To keep our experimental design simple and in line with LLMs being one of our main sources of inspiration, we use the same MLP architecture as used for the feedforward layers of Mistral's open-source implementation of transformers at the time of writing.[1] It has three degrees of freedom:

(P5) The embedding dimension: $d \in \mathbb{N}$;
(P6) The hidden dimension: $h \in \mathbb{N}$;
(P7) The number of layers: $L \in \mathbb{N}$.

The full MLP corresponds to the composition of L functions F_i of the form:

$$F_i : z \in \mathbb{R}^d \mapsto z + W_{i,2}^\top \left(\sigma\left(W_{i,1}^{(i)} \frac{z}{\|z\|} \right) \odot W_{i,3} \frac{z}{\|z\|} \right)$$

where $W_{i,1}, W_{i,2}, W_{i,3} \in \mathbb{R}^{h\times d}$, σ is the logistic function, and $\odot$ is the element-wise product. This architecture was found to be more efficient in practice than purely vanilla MLP (He et al., 2015; Ba et al., 2016; Shazeer, 2020). Unless otherwise specified, we set $d = 32$, $h = 64$, and $L = 1$. These choices are motivated by compute-optimal design experiments, to be found in the Supplementary Materials (Figures 11 to 15).

Optimizer specification To keep the optimization simple and mitigate the number of hyperparameters, we use the Adam optimizer as implemented in PyTorch and with default values for β_1, β_2. Similarly, we initialize the NN's weights with PyTorch's default scheme. We distinguish between two settings regarding the number of epochs.

In the first setting, we focus on quantifying the speed of learning based on different underlying factorizations. For this setting, we consider small numbers of epochs and use a cosine annealing learning rate schedule, defined as

$$\eta_t = \lambda_t \eta, \quad \text{where} \quad \lambda_t = \left(\frac{\cos(\pi t/T) + 1}{2} \right) \in [0, 1],$$

which leads to two additional hyperparameters:

(P8) The initial learning rate: $\eta > 0$;
(P9) The number of epochs: $T \in \mathbb{N}$.

We set the initial learning rate to $\eta = 3{\cdot}10^{-2}$ and the number of epochs T to be 10^3. These choices are motivated by ablation studies to be found in the Supplementary Materials (Figure 16). In this setting, our figures represent averages over 100 independent runs.

In our second setting, we wait until the training procedure converges to study the optimal point reached by the networks. This requires a larger number of epochs, which we set to $T = 10^6$. For these experiments, we only average our figures over 10 runs. Ablation studies in the Supplementary Materials (Figure 17) show that the cosine learning rate schedule is not optimal in this setting. On the one hand, choosing a large initial learning rate, such as $\eta = 3 \cdot 10^{-2}$, allows for fast learning at the beginning but leads to instability towards the end of training. On the other hand, choosing

[1] See the `one_file_ref` of `https://github.com/mistralai/mistral-inference` (commit 26a52a1).

a small learning rate alleviates these instabilities but slows down the training at the beginning. An analysis of the loss instabilities leads to the following "custom" scheduler:

$$\log(\eta_t) = \lambda_t \log(\eta) + (1 - \lambda_t) \log(3 \cdot 10^{-4}),$$

This allows us to keep the same hyperparameters (P8) and (P9), which we set to $\eta = 3 \cdot 10^{-2}$ and $T = 10^6$.

4.2 Single Pass Study

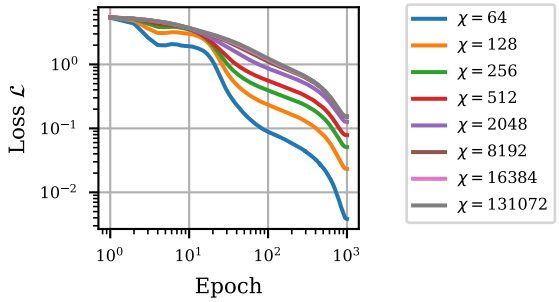

Figure 3: Value of the population loss $\mathcal{L}$ (1) in the single pass setting as a function of the number of epochs for various values of the statistical complexity parameter χ (SC). We obtain various values of χ by modifying the data model parameter (P1).

In our first experiment, we consider the typical setting of LLMs training: a single pass over the data. In this setting, each update is done by sampling a batch of i.i.d. samples $(x^{(t)}, y^{(t)}) \sim p$ drawn according to a data distribution generated as described in Subsection 4.1. We use batches of size $n = 8096$, we set all parameters and hyperparameters to default values except for the input factorization $(p_i)_{i\in[\ell]}$, and we study how the test loss evolves with respect to the number of epochs.[2] We do not vary output factorization (q_j) to avoid confounding factors caused by changes in the distribution of the entropy of $p(y|x)$.[3] In this setting, the embedding $e : \mathcal{X} \to \mathbb{R}^d$ is learned.

We find that the learning speed is correlated with the statistical complexity χ (SC), as highlighted in Figure 3. This suggests that the MLP is able to leverage the implicit product form of the target distribution to improve its learning.

4.3 Compression Study

In this setting, we assume that the batch size n approaches to infinity, so that we directly observe the conditional probabilities $p(y|x)$ for all values of x, and can optimize directly on the population loss $\mathcal{L}$. Once again, the embedding $e : \mathcal{X} \to \mathbb{R}^d$ is learned. We know from the theoretical analysis in Subsection 3.1 that choosing $d \geq \bar{\chi}$ enables $\widehat{p}$ to match the ground-truth distribution p for a well-chosen set of weights defining u, e and F. However, it is well-known that the class of functions that a neural network can represent is much bigger than the class of functions that are actually reachable after a reasonable number of gradient updates (Zhang et al., 2017), hence the need for experimental validation of the impact of both the complexity $\bar{\chi}$ associated with a probability distribution satisfying (FH) and the model's embedding dimension d.

[2] More specifically, we let (p_i) be $(2)_{i\in[12]}$, $(4)_{i\in[6]}$, $(8)_{i\in[4]}$, $(16)_{i\in[3]}$, $(64)_{i\in[2]}$, or $(4096)_{i\in[1]}$.

[3] Indeed, we observe that neural networks performance is impacted by the entropy of the target conditional distributions.

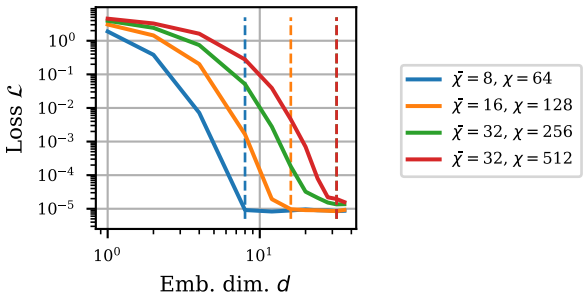

Figure 4: Value of the population loss $\mathcal{L}$ (1) in the compression setting after $T = 10^6$ epochs of training as a function of the embedding dimension d for $|I_j| \in \{1, 2, 3, 4\}$ (P3). The dashed lines indicate $d = \bar{\chi}$.

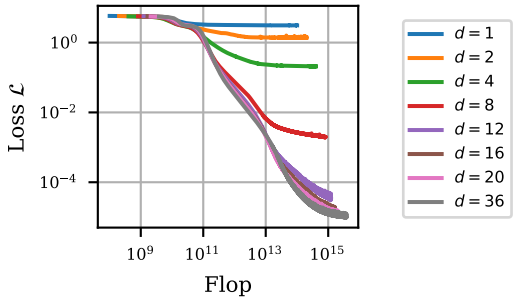

Figure 5: Value of the population loss $\mathcal{L}$ (1) in the compression setting as a function of the compute for various values of the embedding dimension d (P5). Note that $\bar{\chi} = 16$ (AC) with our default experimental parameters.

Our findings are presented in Figure 4 and 5. In Figure 4, we observe that letting $d \geq \bar{\chi}$ (AC) leads to the network learning the problem well, with the loss saturating close to machine precision, which supports our claims from Section 3.1. When $d \leq \bar{\chi}$, the loss also depends on the statistical complexity χ (SC). This is not surprising, as it is also true of the global minimum over all functions taking the form of (3). In Figure 5, we see that the loss initially follows a scaling law as a function of the compute, before eventually saturating due to limitations imposed by the embedding dimension d.

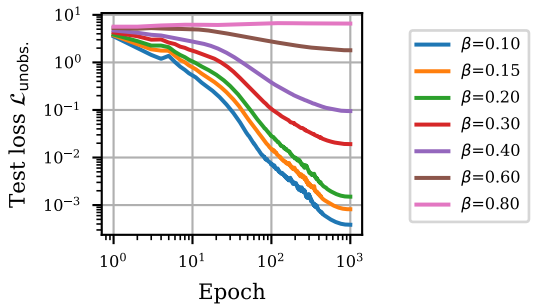

Figure 6: Value of the loss $\mathcal{L}_{\text{unobs.}}$ on unobserved data in the generalization setting as a function of the number of epochs for various values of the connectivity parameter β (P3').

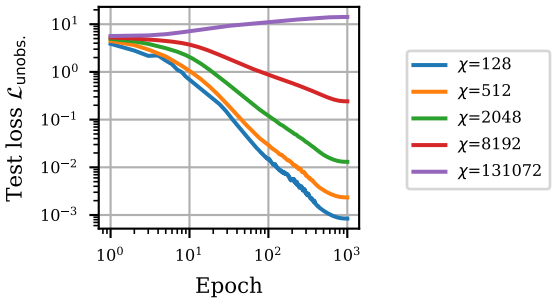

Figure 7: Value of the loss $\mathcal{L}_{\text{unobs.}}$ on unobserved data in the generalization setting as a function of the number of epochs for various values of the statistical parameter χ (SC). We obtain various values of χ by modifying (P1).

4.4 Generalization Study

In this setting, we study NNs' capacity for generalizing their learning of $p(y|x)$ to yet unseen inputs by leveraging the factorial nature of the task at hand. More concretely, we assume that we only have access to a subset $\mathcal{X}_{\text{obs}} \subset \mathcal{X}$ of the data at training time. This results in one additional experimental parameter:

(P10) The data split: $\gamma = |\mathcal{X}_{\text{obs}}| \,/\, |\mathcal{X}|$

By default we let $\gamma = 0.9 = 90\%$. As in the compression setting, we set ourselves in the ideal scenario where we observe $p(y|x)$ for $x \in \mathcal{X}_{\text{obs}}$, as if we were provided with very large batches. We are interested in understanding how the network can generalize to unseen values of $x \in \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$. This setting falls into the scope of covariate shift (Sugiyama et al., 2007) where the distribution of x changes but the conditional distribution of $y|x$ remains the same, with the additional challenge here that the support of the shifted covariate is disjoint from the support of the observed one. To allow for generalization, the embedding is set to a fixed factorization-compatible embedding $\tilde{e} : \mathcal{X} \to \mathbb{R}^d$ (as defined in Subsection 2.2), with the embedding e^i of each factor sampled as a centered isotropic Gaussian vector.

We observe in Figure 6 the effect of the connectivity of the factorization graph (see Figure 1), driven by the connectivity parameter β (P3'), on the MLP's ability to generalize. The y-axis represents the loss $\mathcal{L}_{\text{unobs.}}$ on *unobserved data*. As expected, when $\beta = 0$, there is no causal effect of x on y, and, as $p(y|x)$ is thus constant across $\mathcal{X}$, it is easy to generalize. On the other end of the spectrum, when $\beta = 1$, then $\text{pa}_j = \mathcal{X}$ for all j and one cannot hope to generalize on unseen data. We also observe that the generalization capability is linked to the statistical complexity (Figure 7). This figure is obtained by varying the input factors as in the compression setting, with all the other values set to default.

Finally, we study the effect of the percentage of training data $\gamma = |\mathcal{X}_{\text{obs}}|/|\mathcal{X}|$. We observe that when dealing with highly sparse graphs, one does not need a lot of data to learn the

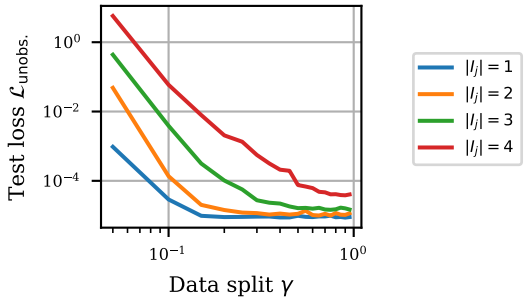

Figure 8: Value of the loss $\mathcal{L}_{\text{unobs.}}$ on unobserved data in the generalization setting after $T = 10^6$ epochs of training as a function of the data split parameter γ (P10) for various values of the number of parents $|I_j|$ (P3).

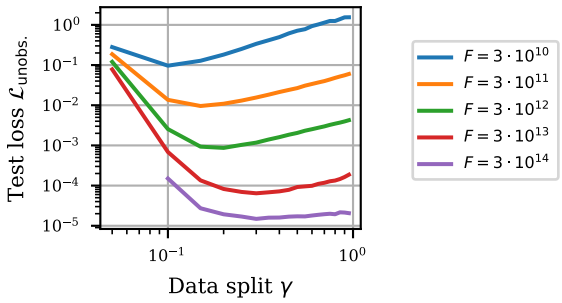

Figure 9: Value of the loss $\mathcal{L}_{\text{unobs.}}$ on unobserved data in the generalization setting as a function of the data split parameter γ (P10) for various values of the number F of flops.

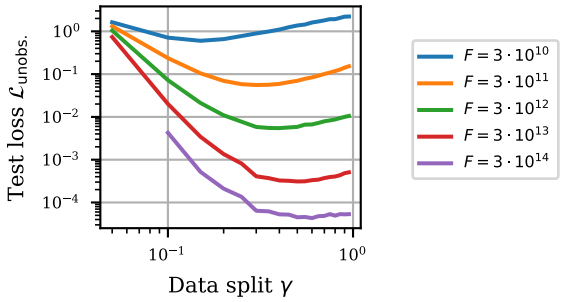

Figure 10: Same as Figure 9, but with $|I_j| = 3$ rather than $|I_j| = 2$ (P3).

underlying mechanisms that generated the data (Figure 8). In Figures 9 and 10, we also plot the "isoflop" graphs of the generalization loss $\mathcal{L}_{\text{unobs.}}$ as a function of the data split γ given different constraints on the number F of floating-point operations (FLOP), which in turn constrain the number of training epochs T since $F \propto \gamma T$ (as each epoch consists in a single pass over each training point). Interestingly, we observe that if one wants to optimize for computation, it is better given a certain compute constraint to train on the same data multiple times, rather than to use new data. This is particularly true when in the presence of highly structured data (compare Figure 9 with Figure 10). This resonates with the recent findings of Charton and Kempe (2024).

5 Discussion

In this work, we introduced a new set of structural assumptions on discrete data. Through our theoretical analysis, we provided new insights into how these assumptions can simplify the learning process, and we showed through extensive experiments that NNs can exploit these hidden structures to enhance their performances.

Our key takeaway is the presence of implicit scaling laws of the form

$$\mathcal{L} \propto \Lambda(\xi, G),$$

where $\mathcal{L}$ is the loss, ξ is either the compute (e.g., Figure 3), the model capacity (e.g., Figure 4), or the number of data (e.g., Figure 8); and G represents the hidden "graphical" factorization underlying the data. The function Λ is typically a decreasing function of ξ and an increasing function of complexity parameters of G, such as the number of factors (Figures 3, 4, and 7), the connectivity of the graph (Figures 6 and 8), or $\bar{\chi}$ and χ defined in (AC) and (SC).

While our findings align with the idea that neural network performance scales with structural complexity, they do not conform to the explicit power-law patterns that are usually reported in other scaling law studies Kaplan et al. (2020). We hypothesize that this is due to the scale of our experiments, leading to some alternative scaling regime. As a side note, a close look at some experiments (e.g., Figure 6 or 7) suggests the existence of a power-law regime, which posits a functional $\omega(G)$ such that $\mathcal{L} \propto \xi^{-\omega(G)}$, where $\omega(G)$ captures how *simple* the underlying factorization is, and is thus expected to be a decreasing function of the complexity parameters of G. We leave the precise identification of such a functional for future work.

On a final note, and as hinted at in Subsection 2.2, we previse a possible strong link between our data model and *transfer learning*. Transfer learning focuses on transferring knowledge learned from one task to improve learning in another, often related but distinct task or domain (Pan and Yang, 2009). Our framework could give a theoretical ground to explain the performance of transfer learning procedures on discrete data when compatible factorizations arise in different tasks, in the sense that a learned embedding adapted to a given task could become a factorization-compatible embedding for another subsequent task.

References

Ahuja, K. and Mansouri, A. (2024). On provable length and compositional generalization.

Alabdulkareem, A., Arnold, C. M., Lee, Y., Feenstra, P. M., Katz, B., and Barbu, A. (2024). Securellm: Using compositionality to build provably secure language models for private, sensitive, and secret data.

Allen-Zhu, Z. and Li, Y. (2024). Physics of language models: Part 3.3, knowledge capacity scaling laws.

Arora, S. and Goyal, A. (2023). A theory for emergence of complex skills in language models.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Bahri, Y., Dyer, E., Kaplan, J., Lee, J., and Sharma, U. (2024). Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27):e2311878121.

Barak, B., Edelman, B. L., Goel, S., Kakade, S. M., Malach, E., and Zhang, C. (2022). Hidden progress in deep learning: Sgd learns parities near the computational limit. *ArXiv*, abs/2207.08799.

Bordelon, B., Atanasov, A., and Pehlevan, C. (2024). A dynamical model of neural scaling laws.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

Cabannes, V., Dohmatob, E., and Bietti, A. (2024). Scaling laws for associative memories.

Cabannes, V. and Vigogna, S. (2023). How many samples are needed to leverage smoothness? In *NeurIPS*.

Cagnetta, F., Petrini, L., Tomasini, U. M., Favero, A., and Wyart, M. (2024). How deep neural networks learn compositional data: The random hierarchy model. *Physical Review X*.

Candes, E. J. and Plan, Y. (2010). Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936.

Canonne, C. L. (2020). A short note on learning discrete distributions. *arXiv preprint arXiv:2002.11457*.

Caponnetto, A. and De Vito, E. (2006). Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*.

Charton, F. and Kempe, J. (2024). Emergent properties with repeated examples.

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jiang, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., Sanyal, S., Welleck, S., Ren, X., Ettinger, A., Harchaoui, Z., and Choi, Y. (2023). Faith and fate: Limits of transformers on compositionality.

Fefferman, C., Mitter, S., and Narayanan, H. (2016). Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29:983–1029.

Feldman, J., O'Donnell, R., and Servedio, R. A. (2008). Learning mixtures of product distributions over discrete domains. *SIAM Journal on Computing*, 37(5):1536–1564.

Guo, S., Tóth, V., Schölkopf, B., and Huszár, F. (2024). Causal de finetti: On the identification of invariant causal structure in exchangeable data. *Advances in Neural Information Processing Systems*, 36.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Henighan, T., Kaplan, J., Katz, M., Chen, M., Hesse, C., Jackson, J., Jun, H., Brown, T. B., Dhariwal, P., Gray, S., et al. (2020). Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.

Hristache, M., Juditsky, A., Polzehl, J., and Spokoiny, V. (2001). Structure adaptive approach for dimension reduction. *Annals of Statistics*, pages 1537–1566.

Hutter, M. (2021). Learning curve theory.

Jordan, M. (2004). Graphical Models. *Statistical Science*, 19(1):140 – 155.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models.

Maloney, A., Roberts, D. A., and Sully, J. (2022). A solvable model of neural scaling laws.

Marion, P., Berthier, R., Biau, G., and Boyer, C. (2024). Attention layers provably solve single-location regression.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Parascandolo, G., Kilbertus, N., Rojas-Carulla, M., and Schölkopf, B. (2018). Learning independent causal mechanisms.

Shazeer, N. (2020). Glu variants improve transformer.

Sugiyama, M., Krauledat, M., and Müller, K.-R. (2007). Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(5).

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models.

Valvoda, J., Saphra, N., Rawski, J., Williams, A., and Cotterell, R. (2023). Benchmarking compositionality with formal languages.

Wang, B., Yue, X., Su, Y., and Sun, H. (2024). Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization.

Wies, N., Levine, Y., and Shashua, A. (2022). Sub-task decomposition enables learning in sequence to sequence tasks. *arXiv preprint arXiv:2204.02892*.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2017). Understanding deep learning requires rethinking generalization.

Zhang, J., Nolte, N., Sadhukhan, R., Chen, B., and Bottou, L. (2024). Memory mosaics. *arXiv preprint arXiv:2405.06394*.

Scaling Laws with Hidden Structure: Supplementary Materials

A Theory Supplement

In Section 3, we exhibited a solution to the matrix equation

$$(\log p(y|x))_{y,x} \in \mathbb{R}^{M\times N} = U^\top G + \text{column-wise constant} \tag{5}$$

with $U \in \mathbb{R}^{Q\times M}$ and $G \in \mathbb{R}^{Q\times N}$ with as before $Q = \sum_{j\in[\ell]} q_j$. Another exact solution, which works as a dual of sorts to the first one, is given by

$$G' = \begin{bmatrix} (\mathbf{1}_{\mathrm{pa}_1})_{x\in\mathcal{X}} \\ \vdots \\ (\mathbf{1}_{\mathrm{pa}_\ell})_{x\in\mathcal{X}} \end{bmatrix} \in \mathbb{R}^{\bar{P}\times N}$$

and

$$U' = \begin{bmatrix} (\log p(y_1|z_1))_{z_1\in\mathrm{pa}_1, y\in\mathcal{Y}} \\ \vdots \\ (\log p(y_\ell|z_\ell))_{z_\ell\in\mathrm{pa}_\ell, y\in\mathcal{Y}} \end{bmatrix} \in \mathbb{R}^{\bar{P}\times M},$$

where $\bar{P} = \sum_{j\in[\ell]} \left|\mathrm{pa}_j\right|$, $(\mathbf{1}_{\mathrm{pa}_j})_{x\in\mathcal{X}} \in \mathbb{R}^{|\mathrm{pa}_j|\times N}$ is the matrix whose column indexed by $x \in \mathcal{X}$ is the one-hot encoding of the combined factor pa_j of x, and the rows of the matrix $(\log p(y|z_j))_{z_j\in\mathrm{pa}_j, y\in\mathcal{Y}} \in \mathbb{R}^{|\mathrm{pa}_j|\times M}$ are indexed by the elements z_j of pa_j. Combining those two solutions yields a third solution: for each $j \in [\ell]$, let G_j be the matrix $(\mathbf{1}_{\mathrm{pa}_j})_{x\in\mathcal{X}} \in \mathbb{R}^{\left|\mathrm{pa}_j\right|\times N}$ if $\left|\mathrm{pa}_j\right| < q_j$, and let it be the matrix $(\log p(y_j|x))_{y_j\in\mathcal{Y}_j, x\in\mathcal{X}} \in \mathbb{R}^{q_j\times N}$ otherwise. Likewise, let U_j be the matrix $(\log p(y_j|z_j))_{z_j\in\mathrm{pa}_j, y\in\mathcal{Y}} \in \mathbb{R}^{\left|\mathrm{pa}_j\right|\times M}$ if $\left|\mathrm{pa}_j\right| < q_j$, and let it be the matrix $(\mathbf{1}_{y_j})_{y\in\mathcal{Y}} \in \mathbb{R}^{q_j\times M}$ otherwise. Then the matrices

$$G'' = \begin{bmatrix} G_1 \\ \vdots \\ G_\ell \end{bmatrix} \in \mathbb{R}^{\sum_{j\in[\ell]} \min\{|\mathrm{pa}_j|, q_j\}\times N}$$

and

$$U'' = \begin{bmatrix} U_1 \\ \vdots \\ U_\ell \end{bmatrix} \in \mathbb{R}^{\sum_{j\in[\ell]} \min\{|\mathrm{pa}_j|, q_j\}\times M}$$

are solutions to (5), which shows that the embedding dimension d can in fact be as small as $\sum_{j\in[\ell]} \min\{|\mathrm{pa}_j|, q_j\}$ and still allow for a perfect representation of the conditional distribution.

As alluded to in Section 3, the matrix $\Lambda = (\log p(y|x))_{y,x}$ having a small rank does not imply the existence of a non-trivial factorization. Consider the following example: let $N, M \in \mathbb{N}$ and pick any $t_1, \ldots, t_N \in (0, 1)$, as well as any $v \in (0, 1)^M$ such that $\sum_{i=1}^M v_i = 1$. The matrix

$$\Lambda := \begin{bmatrix} \log t_1 & \ldots & \log t_N \\ \log(1-t_1) + \log v_1 & \ldots & \log(1-t_N) + \log v_1 \\ \vdots & \ddots & \vdots \\ \log(1-t_1) + \log v_M & \ldots & \log(1-t_N) + \log v_M \end{bmatrix} \in \mathbb{R}^{(M+1)\times N}$$

defines a conditional (log-)distribution $\log p(y|x)$, and its rank is at most 3. However, the distribution admits no non-trivial factorization for generic values of $t_1, \ldots, t_N$ and v.

B Experimental Supplement

All our experiments were run on a single V100 GPU. We summarize in Table 1 our default hyperparameters.

Data calibration The data model, corresponding to (P1), (P2), (P3), (P3') and (P4) was chosen to allow for relatively large and sparse graphs (in the sense of Figure 1), with $p(y|x)$ charging a small number of elements. The other parameters were chosen based on compute optimal considerations, which are detailed below.

	Name	Notation	Default values
(P1)	Input factors	$(p_i)_{i\in[k]}$	$(2)_{i\in[12]}$
(P2)	Output factors	$(q_j)_{i\in[\ell]}$	$(8)_{j\in[4]}$
(P3)	Number of parents	$\lvert I_j\rvert$	2
(P3')	Connectivity parameter	β	–
(P4)	Concentration parameter	α	10^{-1}
(P5)	Embedding dimension	d	64
(P6)	Hidden dimension	h	128
(P7)	Number of layers	L	1
(P8)	Learning rate	η	$3\cdot 10^{-2}$
(P9)	Number of epochs	T	$\{10^3, 10^6\}$
(P10)	Pourcentage of seen data (*generalization setting*)	γ	0.9

Table 1: The different parameters and hyperparameters from in Section 4 and their default values used in the experiments.

FLOPs In order to define the compute cost of our training, we use an idealized model for floating-point operations (FLOPs), consistent with common practice in similar studies (Kaplan et al., 2020).

1. Forward pass
   (a) The embedding layer is a look-up dictionary, resulting in 0 FLOPs.
   (b) For each feedforward block, the total FLOPs C across all layers is given by:

$$C = 6 \times L \times d \times h.$$

   Indeed, for a single layer, we have three matrix multiplication of a vector of size a by a matrix of size $a \times b$, where $\{a, b\} = \{d, h\}$, resulting in $2 \times a \times b$ FLOPs for each multiplication.
   (c) The output layer, which is a linear layer, has a computational complexity of:

$$C = 2 \times d \times M$$

The backward pass is estimated to require approximately twice the compute of the forward pass (Kaplan et al., 2020).

Model calibration experiments Given the data model specified by (P1), (P2), (P3), (P4) from Table 1, as well as the optimization model specified by $T = 10^3$ and $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$, we optimize the model parameters, d, h and L, greedily one after the other. We start with $h = 4d$, and $L = 1$, leading to Figure 11. This figure suggests setting $d \in \{64, 128\}$. Between these two options, we choose $d = 64$ as it reduces the compute cost of our experiments (P5). From there, Figure 12 suggests choosing $h \in \{2d, 3d, \dots\}$. Between these options, we choose $h = 2d$ to save compute (P6). Finally, Figure 14 explains our choice of $L = 1$ (P7). Each figure was computed as an average over 100 independent runs.

It should be noted that in our calibration experiments, we choose $N = M = 4096$, which is much bigger than $d = 64$. As a consequence, L and h do have a significant impact on the total number of FLOPs. In order to show a bigger difference in the choice of these parameters, Figures 13 and 15 consider the case where $N = 360$ and $M = 36$.

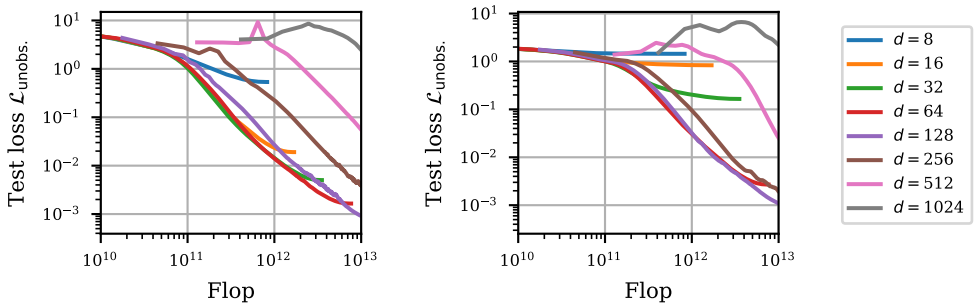

Figure 11: (Left) Effect of the embedding dimension d on the test loss with respect to the number of FLOPs in the generalization setting with (P1), (P2), (P3), (P4) from Table 1, $T = 10^3$, $\eta = 10^{-2}$, which is best among the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$, and $h = 4d$, $L = 1$. (Right) Same figure yet with $(p_i) = (8)_{i\in[4]}$ and $(q_j) = (4096)_{j\in[1]}$. We observe that choosing $d = 64$ yields good results in both experiments. This explains our choice of (P5).

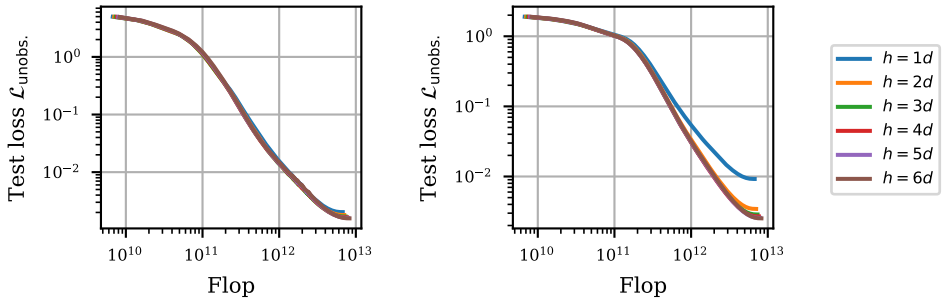

Figure 12: (Left) Effect of the hidden dimension h on the test loss with respect to the number of FLOPs in the generalization setting with (P1), (P2), (P3), (P4), (P5) from Table 1, $T = 10^3$, $\eta = 10^{-2}$, which is best among the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$, and $L = 1$. (Right) Same figure yet with $(p_i) = (8)_{i\in[4]}$ and $(q_j) = (4096)_{j\in[1]}$. The choice of $h = 2d$ (P6) yields good results.

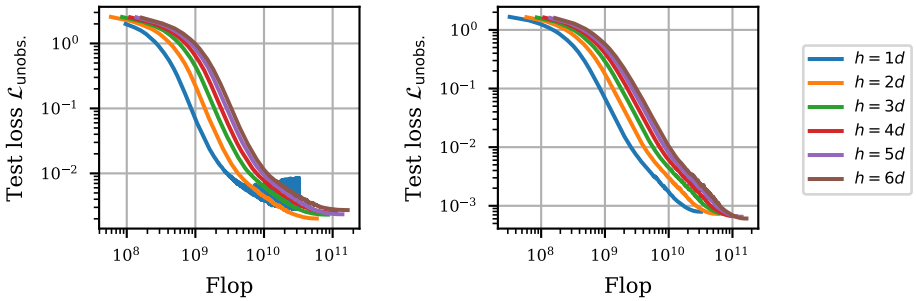

Figure 13: (Right) Same figure as Figure 12, yet with $(p_i) = (2, 2, 2, 3, 3, 5)$, and $(q_j) = (2, 2, 3, 3)$. (Left) Same figure yet with $(p_i) = (3, 4, 5, 6)$, and $(q_j) = (36)$ This shows that the choice of $h = 2d$ is a decent one in general. Although for simple problems where $\bar{\chi}$ (AC) is much smaller than $d = 64$, we observe that $h = d$ is more compute-efficient.

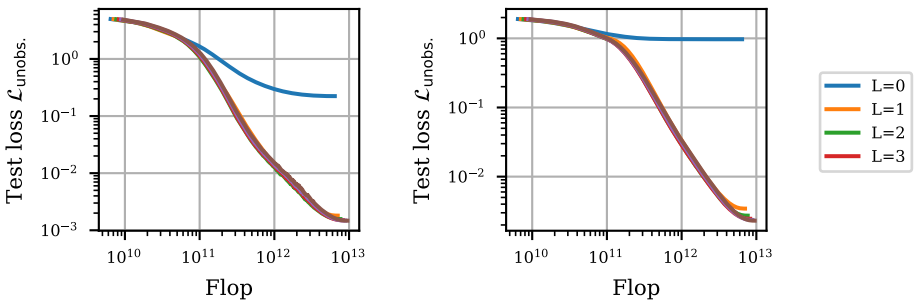

Figure 14: (Left) Effect of the number of layers L on the test loss with respect to the number of FLOPs in the generalization setting with (P1), (P2), (P3), (P4), (P5), (P6) from Table 1, $T = 10^3$ and $\eta = 10^{-2}$ which is best among the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$. (Right) Same figure yet with $(p_i) = (8)_{i\in[4]}$ and $(q_j) = (4096)_{j\in[1]}$. The choice of $L = 1$ (P7) yields good results.

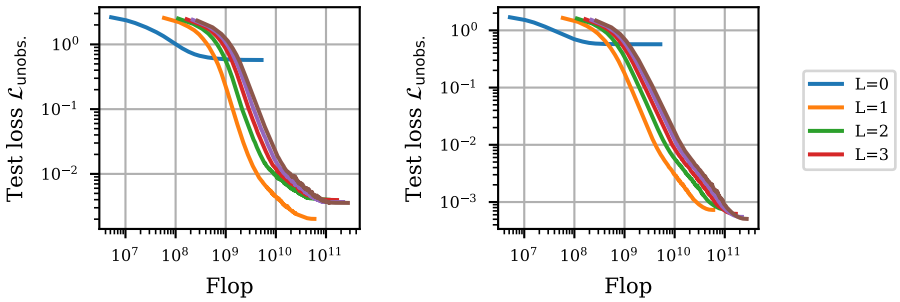

Figure 15: (Right) Same figure as Figure 14, yet with $(p_i) = (2, 2, 2, 3, 3, 5)$, and $(q_j) = (2, 2, 3, 3)$. (Left) Same figure yet with $(p_i) = (3, 4, 5, 6)$, and $(q_j) = (36)$ This supports the choice of $L = 1$.

Optimization calibration experiments In experiments to quantify the speed of learning, we set $T = 10^3$. Figure 16 shows that choosing $\eta = 3 \cdot 10^{-2}$ yields good results in this setting. In experiments to inspect NNs at convergence, we set $T = 10^6$. Figure 17 shows that the cosine learning-rate schedule is not optimal in our setting, as well as the performance of our custom scheduler:

$$\log(\eta_t) = \lambda_t \log(3 \cdot 10^{-2}) + (1 - \lambda_t) \log(3 \cdot 10^{-4}), \qquad \text{where} \qquad \lambda_t = \frac{\cos(\pi t / 10^6) + 1}{2}. \tag{7}$$

This is the learning rate scheduler we use for these long runs.

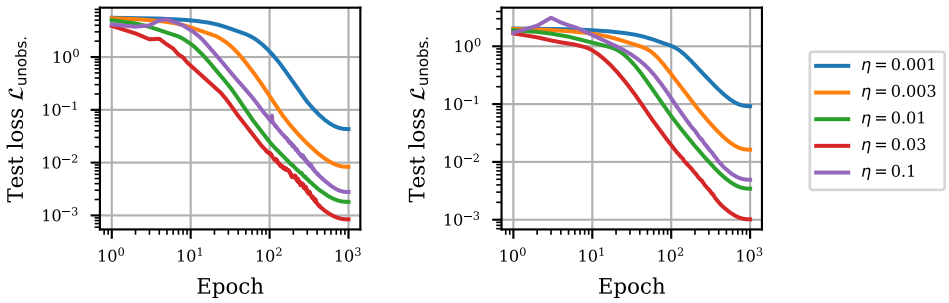

Figure 16: (Left) Effect of initial learning rate of the cosine schedule on the test loss with respect to the number of epochs in the generalization setting with (P1), (P2), (P3), (P4), (P5), (P6), (P7) from Table 1, $T = 10^3$. (Right) Same figure yet with $(p_i) = (8)_{i \in [4]}$ and $(q_j) = (4096)_{j \in [1]}$. The choice of $\eta = 3 \cdot 10^{-2}$ yields good results in both experiments explaining our choice of (P9) given (P8).

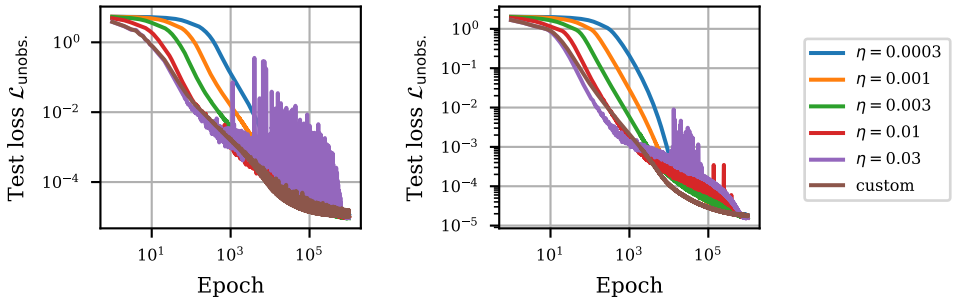

Figure 17: Same figure as Figure 16 yet with $T = 10^6$. Instabilities appear after around 10^3 epochs. This suggests that we should modify the scheduler to decrease the learning rate earlier, for example with the "custom" scheduling of Eq. (7) (brown plot).

Additional figures In addition to the calibration experiments in this appendix and the figures in the main text, we report some additional experiments here. The following figures are the results of averaging over 10 runs, rather than 100 runs. Figure 19 repeats the compression study for another choice of experimental parameters. Figure 18 investigates the interplay between the approximation complexity $\bar{\chi}$ and the embedding dimension d in the generalization setting, rather than in the compression setting. Figure 20 provides an alternative visualization of the results presented in Figure 9 and illustrates the effect of the data split γ and the number of flops on the generalization loss $\mathcal{L}_{\text{unobs.}}$ in the generalization setting for various factorizations. Note that in order to save compute, Figures 8, 9 and 20 were made with $d = 32$ (P5) instead of $d = 64$.

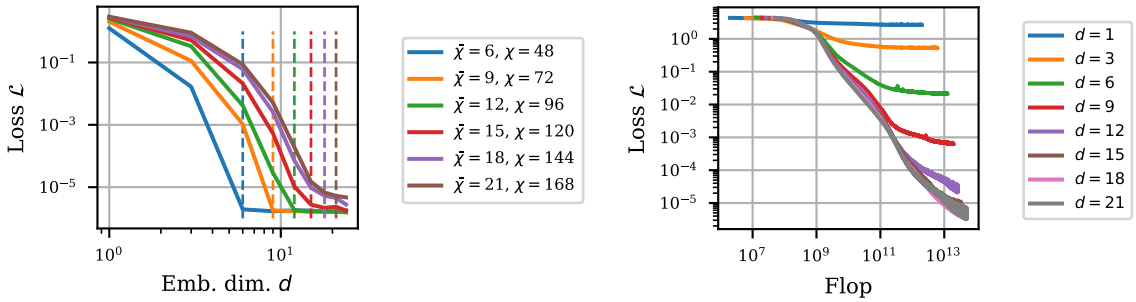

Figure 18: Same graphs as Figure 4 (left) and Figure 5 (right), except that $(q_j) = (8)_{j\in[3]}$, $(p_i) = (x)_{i\in[4]}$, and $|I_j| = 1$, with $x \in [2, 7]$ for the plot on the left and $x = 6$ for the plot on the right (which results in $\bar{\chi} = 18$).

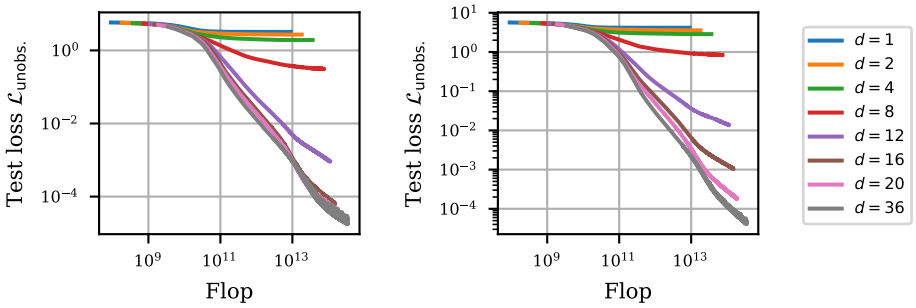

Figure 19: (Left) Same figure as Figure 5 yet in the generalization setting and with $T = 10^5$. (Right) Same figure yet with $|I_j| = 3$ (P3), which sets the compression complexity to $\bar{\chi} = 32$, instead of $\bar{\chi} = 16$ (AC)

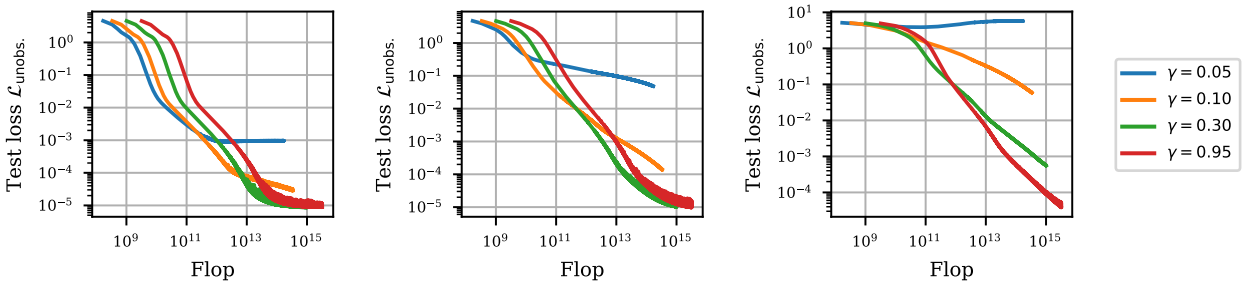

Figure 20: (Middle) Value of the loss $\mathcal{L}_{\text{unobs.}}$ on unobserved data in the generalization setting as a function of the number of FLOPs for various data split parameter γ (P10). Same graph with $|I_j| = 1$ (P3) on the left and $|I_j| = 4$ (P3) on the right.